CENTRALIZED PRINT SERVER USING CUPS

CS 16L2 Mini Project

CSU 142 46	**12150858**	**Sajith A Rahim**
CSU 142 45	**12150857**	**Rustum Thomas**
CSU 142 19	**12150826**	**Fahad Raja**
CSU 142 56	**12150865**	**Tarun I Kurian**

B. Tech. Computer Science & Engineering

Department of Computer Engineering
Model Engineering College Thrikkakara
Kochi 682021
Phone: +91.484.2575370
http://www.mec.ac.in hodcs@mec.ac.in

2017

Model Engineering College Thrikkakara
Dept. of Computer Engineering

C E R T I F I C A T E

This is to certify that, this report titled ***CENTRALIZED PRINT SERVER USING CUPS*** is a bonafide record of the **CS 16L2 Mini Project** work done by

CSU14246	**12150858**	**Sajith A Rahim**
CSU14245	**12150857**	**Rustum Thomas**
CSU14219	**12150826**	**Fahad Raja**
CSU14256	**12150865**	**Tarun I Kurian**

Sixth Semester B. Tech. Computer Science & Engineering students, under our guidance and supervision, in partial fulfillment of the requirements for the award of the degree, B. Tech. Computer Science and Engineering of **Cochin University of Science & Technology**.

Guide

Bijumon T
Asst. Professor
Computer Engineering

Coordinator

Jaimon Jacob
Associate Professor
Computer Engineering

Head of the Department

September 15, 2021

Manilal D L
Associate Professor
Computer Engineering

Acknowledgments

At this moment of accomplishment as we are presenting our work with great pride and pleasure we would like to express our sincere gratitude to all those who helped us in the successful completion of this venture.
First of all we would like to thank our Principal Professor. Dr. V.P Devasia for providing us with a conductive environment and requisite lab facilities. We wish to express our sincere thanks to Mr.Manilal D.L , Head of the Department, Computer Science and Engineering , for his inspiration and constant encouragement which made us take up project and bring it to the completion. We are indebted to our project coordinator Mr. Jaimon Jacob , Associate Professor, Dept. of Computer Science and Engineering for his constant help and support.We also make this opportunity to specially thank our parents and friends for their motivation encouragement and for being constant support throughout.
Above all we thank God Almighty for giving us the courage to move forward with confidence and for all his blessings throughout.

Sajith A Rahim
Rustum Thomas
Fahad Raja
Tarun I Kurian

Abstract

This report describes the analysis, design and implementation of a Centralized Print Server Using Common Unix Printing System named as CPS. The project aims to develop and implement a centralized print server for a Local Area Network (here College LAN). The host server will be setup at the office and will be running CUPS. Host accepts print jobs from client computers, process them, and send them to the printer. The print service is monetized using print vouchers which can be availed from the office. This restricts the possibility of abuse of resources as well us enables to assign quotas and budgets and furthermore to keep track of the printing activities.

Contents

Chapter 1

Introduction

1.1 Project Domain

Network Based Resource Sharing.

1.2 Project Outline

The project aims to develop and implement a centralized print server for a Local Area Network (here College LAN). The host server will be setup at the office and will be running CUPS. Host accepts print jobs from client computers, process them, and send them to the printer.
The print service is monetized using print vouchers which can be availed from the office. This restricts the possibility of abuse of resources as well us enables to assign quotas and budgets and furthermore to keep track of the printing activities.

1.3 Problem Identification

Printing is something every user needs to do. Even in this digital era hardcopies are still required for various purposes like documentation, safekeeping etc. In present day heterogeneous computing environment, providing a centralized printing solution can be hard. Those days of having a large, dedicated printer for only UNIX systems are gone. Today we have to make our systems print to printers from desks that are scattered throughout the college.

Such a situation exists where in our college. A number of printers are scattered throughout the college, with UNIX systems sitting on different LANs and basic firewalls in between. Most of the systems can talk directly to the LAN to which the printers are connected. So it is important to ensure the reliable and flexible sharing of the print h/w with all the nodes. Another important problem that needs to be resolved is the waste and abuse of resources. Today we have 100s of pages being printed out daily without anyone to control, track or limit it. No logging is done and its nearly impossible to determine who is printing what?

1.4 Project Goals

- Assign Quotas and Budget.
- Track who is printing what?
- Monetize Printing Facility.
- Achieve Cost Efficiency.
- Implement Print Access Control.
- Centralization of Printing Activities.
- Enable Cross Platform Printing (Samba*).
- Document Security.

Chapter 2

Literature Survey

2.1 Existing Systems

Presently there is no centralized control nor sharing of print resources implemented in our college. Each Department or section has a local area network bridged to other similar networks. Resources are shared within each such local networks and there is no centralized or global sharing.
More importantly there is no control over these resources. A printer is shared as a network discoverable printer within the network and each node connected to the network can use the resource as he/she pleases. There is no control-track mechanism implemented on these resources. Suppose a situation arises where a shared printer within a network is out of service and some important documents have to be printed but the printer which is in service in another local network, close in proximity cannot be used as it is not discoverable to the node. This all points to the need for a centralized printer shared among all networks which can be tracked, controlled and protected from unauthorized access.

2.1.1 Limitations Of Existing System

- Lack of Access Control Over Resources.
- No method to Track whos printing what?
- No way to implement Monetization.
- Less efficient.
- No Centralization.
- Lack of Cross Platform Flexibility.
- Limited Resource Sharing Capabilities.

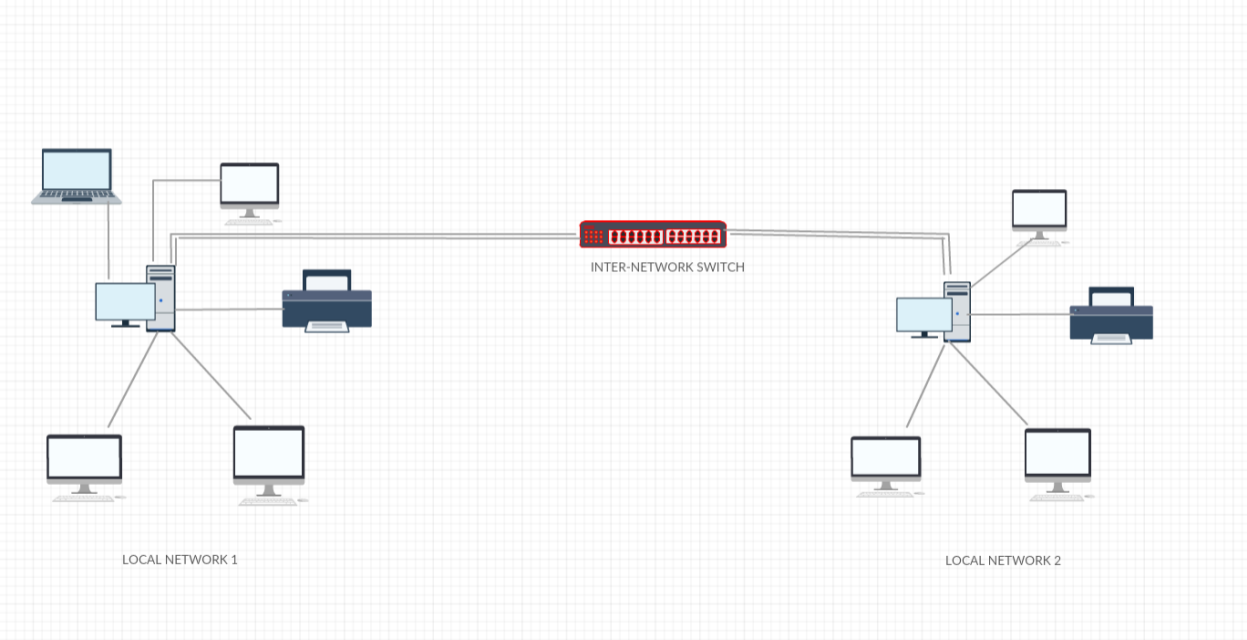


Figure 2.1: Existing Topology

2.2 Proposed System

The project aims to develop and implement a centralized print server for a Local Area Network (here College LAN). The host server will be setup at the office and will be running CUPS. Host accepts print jobs from client computers, process them, and send them to the printer. The print service is monetized using print vouchers which can be availed from the office. These print vouchers consist of a unique code and permits a specified number of prints. Host maintains databases of these voucher codes which is at first used to validate the code and then to keep the remaining print count. The system consists of 3 components;

2.2.1 Server-side Software

Server side software performs the key operational function of intercepting and validating the print requested sent to the configured printer. It uses Tea4CUPS and Pykota as backend wrappers to intercept the print request directed for CUPS. The software is programmed in Python and uses SQL as for implementing Database storage and retrieval.
Three Databases used within the server side; One for maintaining user information, one for maintaining printer information and a third to keep track of print credits. Database for print credits can be implemented separately and connected to user database using a foreign key or can be integrated within the user database.

2.2.2 Client Side Software

Client-side software provides the client or end user with a pop-up window for entering credentials for authentication upon sending a print request to CPS configured printer. Package consists a

Python file and a Visual Basic script and is to be installed on each participating client.

2.2.3 Web Interface

A web interface using CGI -Common Gateway Interface is used to provide an interface to the client for database retrieval. It allows users to check the amount of credits remaining , number of pages print and furthermore a cost estimator to estimate cost of printing in units of number of pages. Python is used for CGI Scripting but PERL or any other viable language can be used for the same. Easier implementations can be done using PHP.

The client side package will provide an interface to enter the voucher code once the print request is sent by the client.

Chapter 3

Requirement Analysis

3.1 Introduction

3.1.1 Purpose

The purpose of this project is to develop and implement a centralized print server for a Local Area Network (here College LAN). The host server will be setup at the office and will be running CUPS. Host accepts print jobs from client computers, process them, and send them to the printer.

3.1.2 Intended Audience

The SRS document is intended to inform the reader of the intended approach in creating the website.The users for which the document is intended for are developers, authorized users, testers and documentation writers. The reader will get a general understanding of the product including its functional and non-functional requirements and its features. This document will begin with an introduction to the product that is being developed will discuss any assumptions that have been made and the currently existing constraints. All the features included as part of the product are outlined in great detail so the reader gets a good understanding of how it will work. The SRS document has been developed for many readers of difficult background.

3.1.3 Project Scope

Centralization of print resources and Efficient Monetization,Tracking and Control.

3.1.4 Design and Implementation constraints

1.Networking is a troublesome process.
2.Authorization and Database require added amount of Security.
3.Grouping and Prioritization of User Groups can be complex

3.2 Functional and Nonfunctional requirements

3.2.1 Functional Requirements

- User Login
- Registration
- Print Accounting
- Recharge

3.2.2 Nonfunctional Requirements

- Maintainability
- Reliability
- Recoverability

3.2.3 Hardware and Software Requirements

Software Requirements

- Linux
- Python
- MySQLi
- MySQL
- Tea4CUPS

Hardware Requirements

- Apache Configured Server
- Network Printer

Chapter 4

Project Design

4.1 Operational Models

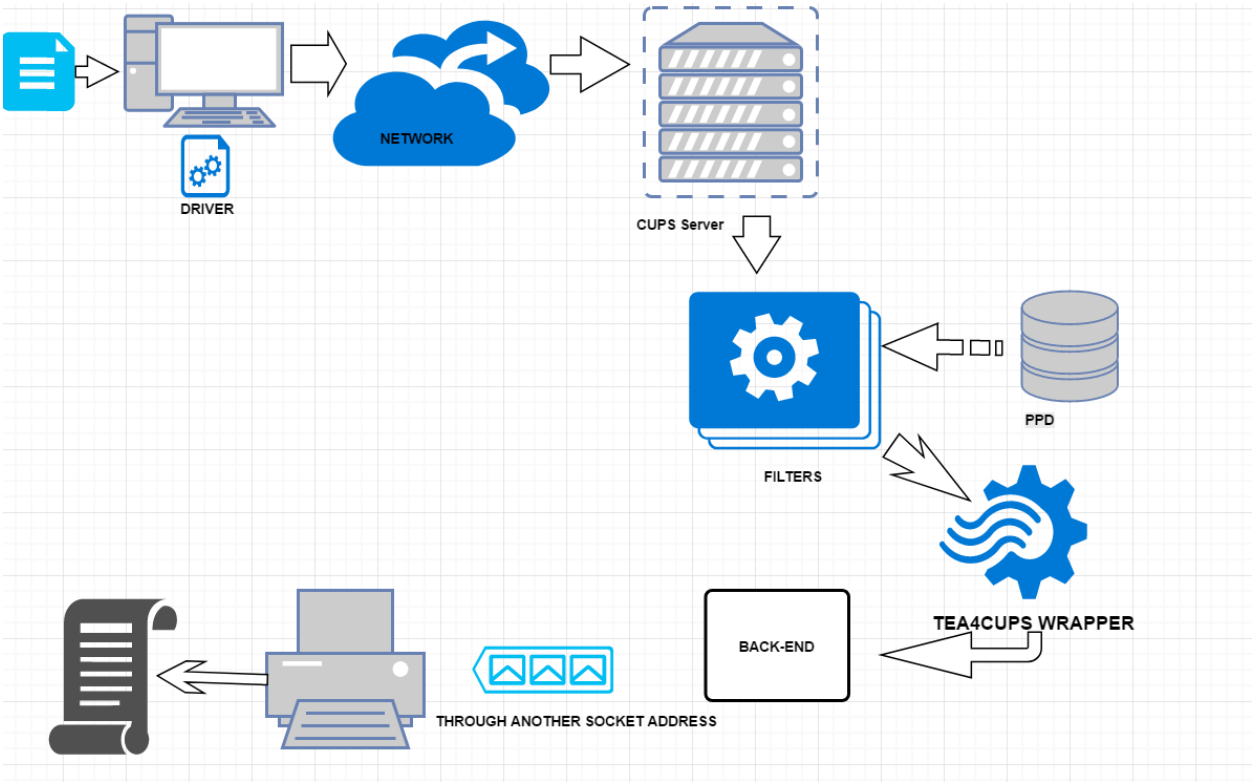


Figure 4.1: Tea4CUPS Operation Model

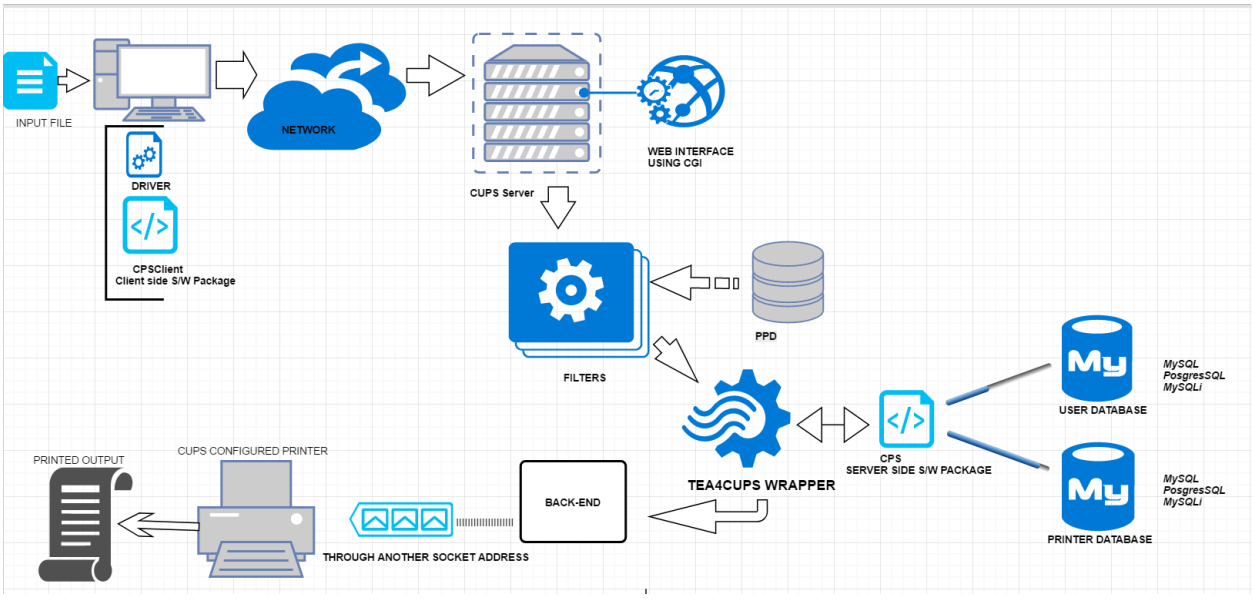


Figure 4.2: CPS Upon Tea4CUPS Operation Model

4.1.1 Input

- Print Request
- Username
- Password
- Confirmation

4.1.2 Output

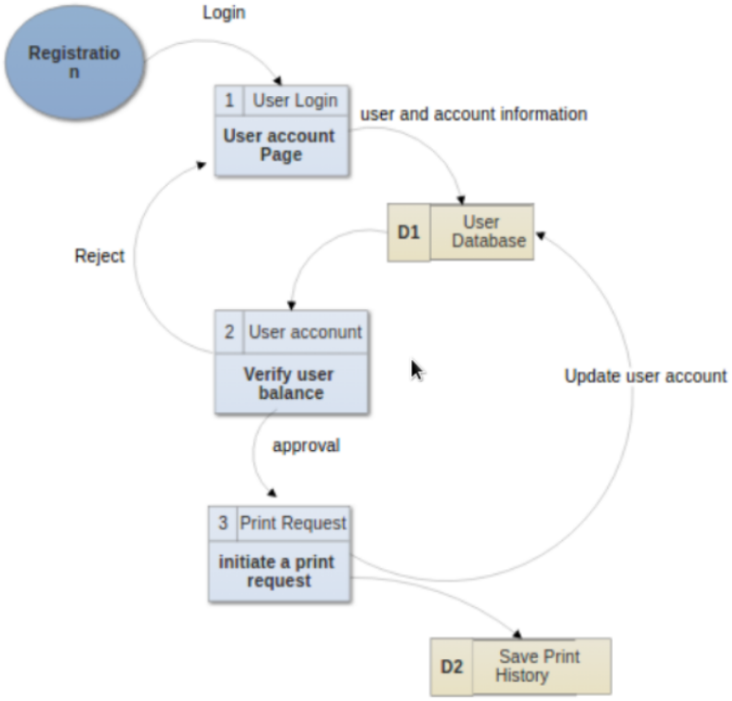
- Hardcopy

4.2 Design Documentation

4.2.1 Design Description

The print request is sent to the Server by CPS-Client, the Client-side software after authentication, where the CPS Server-side software intercepts the request using Tea4CUPS back-end wrapper. CPS-Server authenticates the credentials sent and asks for Confirmation from client. On positive response the and validation success the request is forwarded to CUPS for printing.

4.2.2 Data Flow Diagram



(a) Data Flow Diagram

4.2.3 Database Design

Database Tables

User table: Stores the username, password,credits and usage statistics of user.

Printer table: Stores Printer name,description ,cost of printing and specifications .

ER diagram

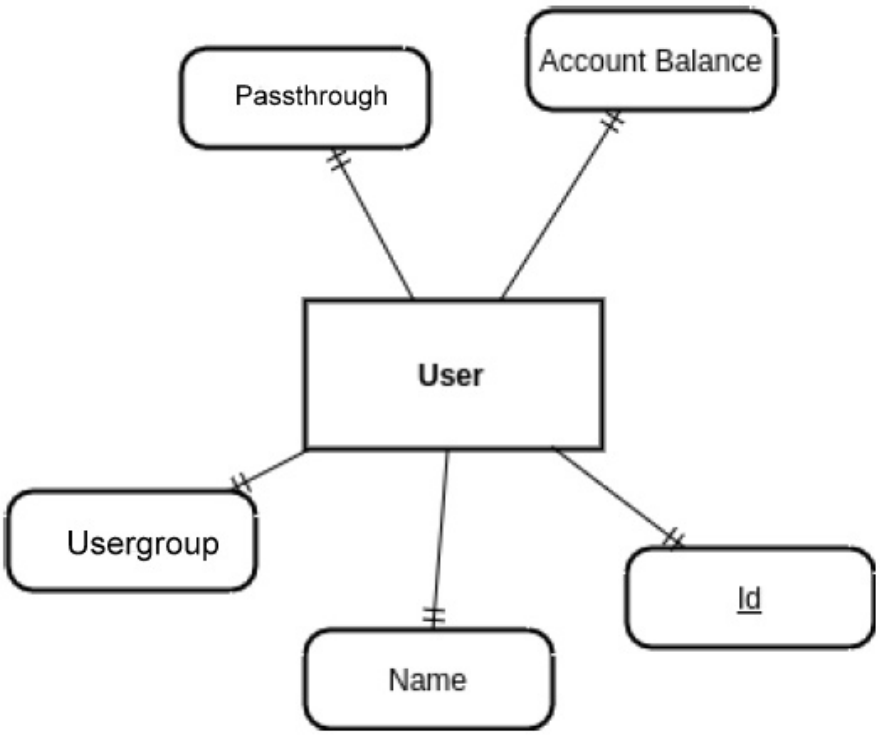


Figure 4.4: ER Diagram

Chapter 5

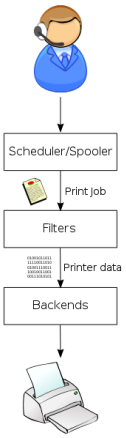
Background

In this chapter we will describe the existing softwares and tools used for the implementation of the project

5.1 CUPS - Common Unix Printing System

5.1.1 CUPS Server

CUPS (Common Unix Printing System) is a modular printing system for Unix-based computer Operating Systems which allows a computer to act as a print server. A computer running CUPS is a host that can accept print jobs from client computers, process them, and send them to the appropriate printer.



[h]

CUPS consists of a print spooler and scheduler, a filter system that converts the print data to a format that the printer will understand, and a backend system that sends this data to the print device. CUPS uses the Internet Printing Protocol (IPP) as the basis for managing print jobs and queues. It also provides the traditional command line interfaces.
System administrators can configure the device drivers which CUPS supplies by editing text files in Adobe's PostScript Printer Description (PPD) format. There are a number of user interfaces for different platforms that can configure CUPS, and it has a built-in web-based interface. CUPS

is free software, provided under the GNU General Public License and GNU Lesser General Public License, Version 2.

5.1.2 History

Michael Sweet, who owned Easy Software Products, started developing CUPS in 1997 and the first public betas appeared in 1999.The original design of CUPS used the LPD protocol, but due to limitations in LPD and vendor incompatibilities, the Internet Printing Protocol (IPP) was chosen instead. CUPS was quickly adopted as the default printing system for most Linux distributions. In March 2002, Apple Inc. adopted CUPS as the printing system for Mac OS X 10.2. In February 2007, Apple Inc. hired chief developer Michael Sweet and purchased the CUPS source code.

5.1.3 Overview

CUPS provides a mechanism that allows print jobs to be sent to printers in a standard fashion. The print-data goes to a scheduler which sends jobs to a filter system that converts the print job into a format the printer will understand.
The filter system then passes the data on to a backenda special filter that sends print data to a device or network connection. The system makes extensive use of PostScript and rasterization of data to convert the data into a format suitable for the destination printer.
CUPS offers a standard and modularised printing system that can process numerous data formats on the print server. Before CUPS, it was difficult to find a standard printer management system that would accommodate the very wide variety of printers on the market using their own printer languages and formats. For instance, the System V and Berkeley printing systems were largely incompatible with each other, and they required complicated scripts and workarounds to convert the program's data format to a printable format. They often could not detect the file format that was being sent to the printer and thus could not automatically and correctly convert the data stream. Additionally, data conversion was performed on individual workstations rather than a central server.
CUPS allows printer manufacturers and printer-driver developers to more easily create drivers that work natively on the print server. Processing occurs on the server, allowing for easier network-based printing than with other Unix printing systems. With Samba installed, users can address printers on remote Windows computers, and generic PostScript drivers can be used for printing across the network.

5.1.4 Scheduler

The CUPS scheduler implements Internet Printing Protocol (IPP) over HTTP/1.1. A helper application (cups-lpd) converts Line Printer Daemon protocol (LPD) requests to IPP. The scheduler also provides a web-based interface for managing print jobs, the configuration of the server, and for documentation about CUPS itself.
An authorization module controls which IPP and HTTP messages can pass through the system. Once the IPP/HTTP packets are authorized they are sent to the client module, which listens for and processes incoming connections. The client module is also responsible for executing external CGI programs as needed to support web-based printers, classes, and job status monitoring and

administration.
Once this module has processed its requests, it sends them to the IPP module which performs Uniform Resource Identifier (URI) validation to prevent a client from sidestepping any access controls or authentication on the HTTP server.
The URI is a text string that indicates a name or address that can be used to refer to an abstract or physical resource on a network. The scheduler allows for classes of printers. Applications can send requests to groups of printers in a class, allowing the scheduler to direct the job to the first available printer in that class. A jobs module manages print jobs, sending them to the filter and backend processes for final conversion and printing, and monitoring the status messages from those processes. The CUPS scheduler utilizes a configuration module, which parses configuration files, initializes CUPS data structures, and starts and stops the CUPS program. The configuration module will stop CUPS services during configuration file processing and then restart the service when processing is complete.
A logging module handles the logging of scheduler events for access, error, and page log files. The main module handles timeouts and dispatch of I/O requests for client connections, watching for signals, handling child process errors and exits, and reloading the server configuration files as needed.
Other modules used by the scheduler include:

- the MIME module, which handles a Multipurpose Internet Mail Extensions (MIME) type and conversion database used in the filtering process that converts print data to a format suitable for a print device;
- a PPD module that handles a list of Postscript Printer Description (PPD) files;
- a devices module that manages a list of devices that are available in the system;
- a printers module that handles printers and PPDs within CUPS.

5.1.5 Filter System

CUPS can process a variety of data formats on the print server. It converts the print-job data into the final language/format of the printer via a series of filters. It uses MIME types for identifying file formats.
The filtering process works by taking input data pre-formatted with six arguments:

- the job ID of the print job
- the user-name
- the job-name
- the number of copies to print
- any print options
- the filename

The default filters included with CUPS include:

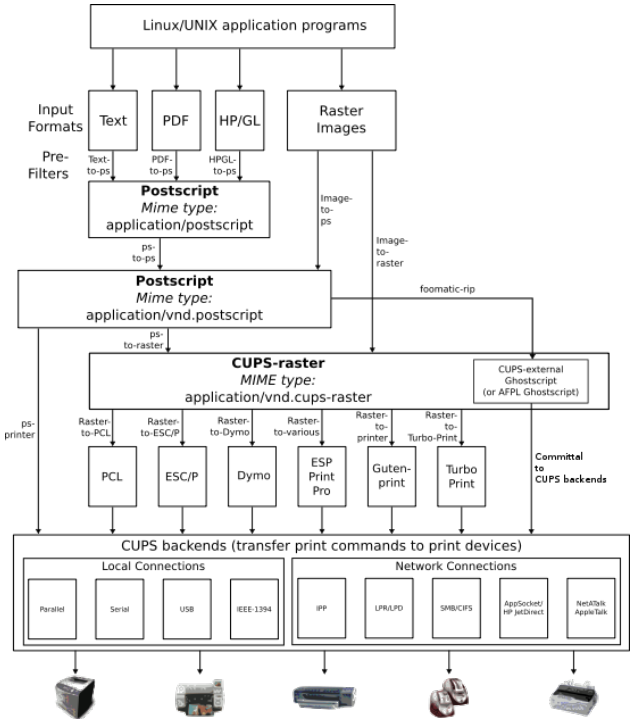
- raster to PCL

- raster to ESC/P or ESC/P2 (an Epson printer language, now largely superseded by their new ESC/P-Raster format)
- raster to Dymo (another printer company).
- raster to Zebra Programming Language or ZPL (a Zebra Technologies printer language)

As of 2009 other proprietary languages like GDI or SPL (Samsung Printer Language) are supported by Splix, a raster to SPL translator.
However, several other alternatives can integrate with CUPS. HPLIP (previously known as HP-IJS) provides Linux+CUPS drivers for HP printers, Gutenprint (previously known as Gimp-Print) is a range of high-quality printer drivers for (mostly) inkjet printers, and TurboPrint for Linux has another range of quality printer drivers for a wide range of printers.

5.1.6 Backends

The backends are the ways in which CUPS sends data to printers. There are several backends available for CUPS: parallel, serial, and USB ports, cups-pdf PDF Virtual Printing, as well as network backends that operate via the IPP, JetDirect (AppSocket), Line Printer Daemon ("LPD"), and SMB protocols.

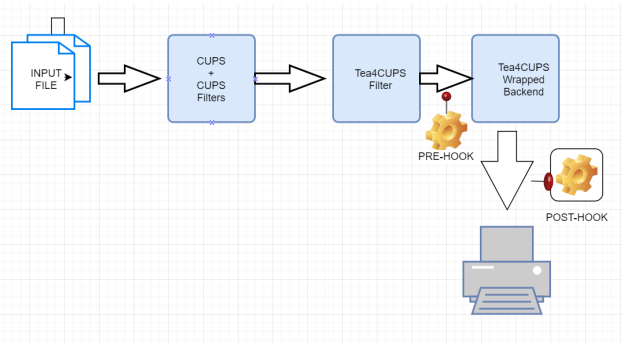


5.2 Tea4CUPS

5.2.1 Indroduction

Tea4CUPS

Tea4CUPS is a CUPS backend wrapper which can capture print data before they are sent to a printer and process, duplicate or dispatch them in a number of ways. Tea4CUPS is the Swiss army's knife of the CUPS' Administrator. Tea4CUPS is similar in functionality to the GNU/Linux tee command, but available as a 100 percent plug and play generic CUPS backend wrapper.
The final filter a print job passes through before being sent to a printer is the backend filter (usb, parallel, socket etc), which transports the printer-ready job to the printer.
Tea4CUPS is not intended to replace the function of any existing backend being used with the printer. It is a script (written in Python) which wraps round the chosen backend to manipulate print data before or after they are dispatched by the backend to the printer.



The manipulation may be done via a combination of prehooks, post-hooks and a filter. There can be only one filter per print queue but as many hooks as wanted can be specified for the same print queue. Hooks may be scripts and do not alter the data which are sent to the printer. If the data should be modified before being sent to a hook or the printer a filter should be defined. Hooks manipulate a data stream without modifying the original stream.
The data stream going to the printer (DATA3) comes from CUPS+cups-filters (DATA1) or the Tea4CUPS filter (DATA2), if there is one.

- There is very adequate documentation (including examples) for Tea4CUPS in /etc/cups/tea4cups.conf and the README in /usr/share/doc/cups-tea4cups/.

The contents of a hook directive and a filter are run as the root user.
The documentation has a list of useful environment variables which can be made available to the commands executed by a hook. The defined hook does all the required work to reformat the job and send it (and any options) to a second queue and/or another destination.
There can be multiple hooks, so the same job can be processed in different ways and sent to different

print queues and destinations.
Scripts run by hooks need not connect with another print queue.

5.2.2 Wrapping Back-end with Tea4CUPS:

A print queue managed by Tea4CUPS can be created with lpadmin, the web interface or system-config-printer.If the unwrapped DEVICE-URI is socket://192.168.7.200:9100 the wrapped one (the -v option to lpadmin) would be

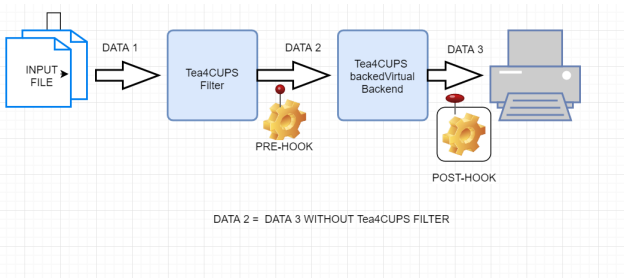
```
print('tea4cups:/socket://192.168.7.200:9100')
```

or

```
print('tea4cups:socket://192.168.7.200:9100')
```

5.2.3 Capturing an Input File

tea4cups:// is a special device-uri in that it is virtual. It is useful when sending data to a real printer with a wrapped backend is not wanted. The input is still available to be manipulated by hooks and/or a filter.
It is tempting to think tea4cups:/file:/tmp/out would print to /tmp/out. However, file:/ is built into CUPS and is not a backend device-uri. But, with a virtual device-uri, a hook could be used for capturing the input file. The following technique is used in some of the scripts on this page.
The input file avoids any filtering which would be done by cups-filters and instead is sent directly to the tea4cups wrapped backend:



A hook or filter can now work directly with the input file to modify it and send the result to a print queue which has been set up with:

```
print('lpadmin -p realq -v <DEVICE_URI> -E -m <PPD>')
```

Depending on the use case the queue virtq could be the one advertised to users. The queue realq would not be published and would only be visible on the server. This way any processing of the input file would be completely invisible to a user.

5.2.4 Virtual PDF Printer

Create a raw printer queue named TeaPDF:

```
print('lpadmin -p TeaPDF -v tea4cups:// -E -m raw')
```

Create pre and post hooks in /etc/cups/tea4cups.conf by adding the following lines at the end of the file:

```
print('prehook_rawpdf  : /bin/cat $TEADATAFILE |  su $TEAUSERNAME -l -c "cat - > '/
posthook_rawpdf : /bin/cat >/tmp/log_of_pdf_creation_for_job_$TEAJOBID')
```

In the user's home directory create a new directory named PDF otherwise print jobs to TeaPDF will be aborted. Any print jobs sent to TeaPDF will be saved to the new PDF directory with the file name format, job-¡JOB-ID¿-¡TITLE¿.pdf.

5.2.5 Print Accounting.

The accounting schemes which CUPS supports are outlined at the CUPS website and on on a machine with cups installed. Some of the problems associated with obtaining accurate information with software accounting are discussed on these openSUSE and KDE pages.

Whatever the pitfalls of software approaches to counting printed pages it is suggested you take a look at pkpgcounter. It is easy enough to use as a prehook and can provide information for a raw queue.

```
print('TEATITLE may contain spaces; they will need removing.
  prehook _accounting : echo TEAUSERNAME TEATITLE pkpgcounter ')
```

pkpgcounter calculates the number of pages to print a given document of a recognised file type. Multipling by the number of copies gives the total number of pages used. TEACOPIES is the number of copies recorded by CUPS in argv[4] of the error-log; that is, the number of copies asked for when the job is submitted to cups. How this is managed depends on CUPS and the printer driver. The CUPS filters could decide to produce a single copy but instruct the CUPS back-end to send that copy several times in a row.

5.3 SQLite3

SQLite s a relational database management system contained in a C programming library. In contrast to many other database management systems, SQLite is not a clientserver database engine.SQLite is a popular choice as embedded database software for local/client storage in application software such as web browsers. It is arguably the most widely deployed database engine, as it is used today by several widespread browsers, operating systems, and embedded systems (such as mobile phones), among others. SQLite has bindings to many programming languages

5.4 Python

5.4.1 History

Python is a widely used high-level programming language for general-purpose programming, created by Guido van Rossum and first released in 1991. An interpreted language, Python has a design philosophy which emphasizes code readability (notably using whitespace indentation to delimit code blocks rather than curly braces or keywords), and a syntax which allows programmers to express concepts in fewer lines of code than possible in languages such as C++ or Java.Python 2.0 was released on 16 October 2000 and had many major new features, including a cycle-detecting garbage collector and support for Unicode. With this release the development process was changed and became more transparent and community-backed. Python 3.0 (which early in its development was commonly referred to as Python 3000 or py3k), a major, backwards-incompatible release, was released on 3 December 2008[32] after a long period of testing. Many of its major features have been backported to the backwards-compatible Python 2.6.x and 2.7.x version series.

5.4.2 What is Python

Python features a dynamic type system and automatic memory management and supports multiple programming paradigms, including object-oriented, imperative, functional programming, and procedural styles. It has a large and comprehensive standard library.Python interpreters are available for many operating systems, allowing Python code to run on a wide variety of systems. CPython, the reference implementation of Python, is open source software and has a community-based development model, as do nearly all of its variant implementations. CPython is managed by the non-profit Python Software Foundation.

Chapter 6

Implementation

In this chapter we will describe components and implementation of our software project.

CPS Package consists of 3 elements;

- CPSServer : Server-side Software Package
- CPSClient : Client-side Software Package
- Web Interface: CGI Scripted Interface

6.1 CPSSERVER - Server-side S/W Package

CPS Server Software package implements the back-end processing of print request sent to CUPS configured Printer. It is a full featured, centralized, and extensible print quota system for CUPS. Server side software performs the key operational function of intercepting and validating the print requested sent to the configured printer. It uses Tea4CUPS and Pykota as back-end wrappers to intercept the print request directed for CUPS. The software is programmed in Python and uses SQL as for implementing Database storage and retrieval.
Three Databases used within the server side; One for maintaining user information, one for maintaining printer information and a third to keep track of print credits.
Database for print credits can be implemented separately and connected to user database using a foreign key or can be integrated within the user database. At present it is implemented integrated within user database.

6.1.1 Description

It consists of 6 python files integrated to the Tea4CUPS base package ,each implementing a specific functionality;

- pkprinters
- pkusers

- edpykota
- dumpykota
- pykotme
- pknotify
- repykota

PKPRINTERS Manage Printer Database.

pkprinters is the preferred tool to manage printers in CPS. With it you can add or delete printers or printer groups, or modify existing printers or printers groups. This is also the tool to use to put printers into one or more printers groups. . Although the pkturnkey command can be used to initialize your database and import printers into it, the pkprinters command offers additional functionalities like full management of printers.
Command line usage:
pkprinters [options] printer1 printer2 printer3 ...

OPTIONS:

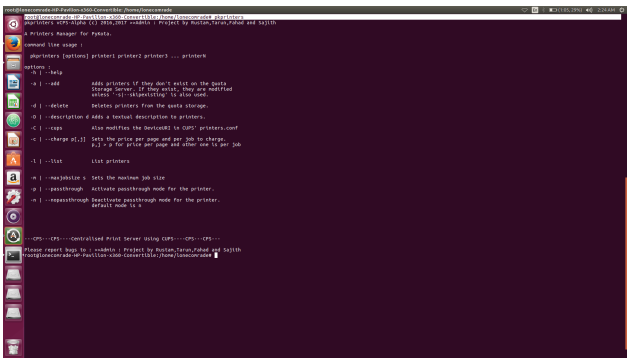
- -a — –add
  -Adds printers if they don't exist on the Quota Storage Server. If they exist, they are modified Unless -s——skipexisting is also used.
- -d — –delete
  -Deletes printers from the quota storage.
- -D — –description
  -Adds a textual description to printers.
- -C — –cups
  -Also modifies the DeviceURI in CUPS' printers.conf
- -l — –list
  -List information about the printer(s) and the printers groups it is a member of.
- -r — –remove
  -In combination with the –groups option above, remove printers from the specified printers groups.
- -s — –skipexisting
  -In combination with the –add option above, tells pkprinters to not modify existing printers.
- -m — –maxjobsize s
  -Sets the maximum job size allowed on the printer to s pages.
- -p — –passthrough
  -Activate passthrough mode for the printer. In this mode, users are allowed to print without any impact on their quota or account balance.

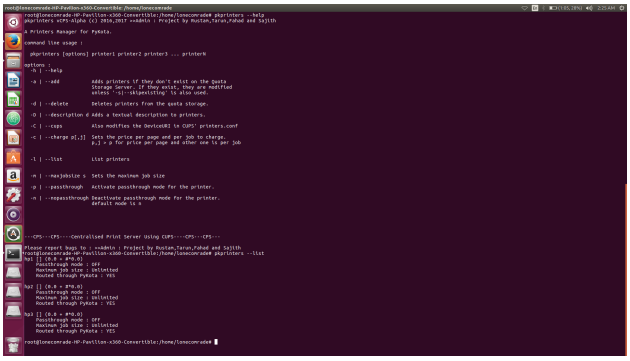
- -n — –nopassthrough
  -Deactivate passthrough mode for the printer. Without -p or -n, printers are created in normal mode, i.e. no passthrough. examples :

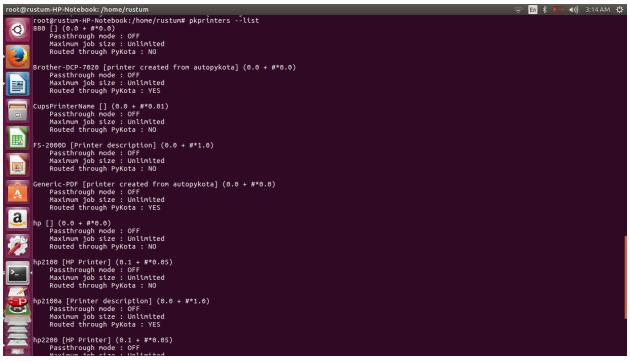
  pkprinters –add -D ”HP Printer” –charge 0.05,0.1 hp2100 hp2200 hp8000

  pkprinters –delete ”*”

SCREENSHOTS:







PKUSERS Manage User Database.

This command enables you to create, manage or delete users or users groups from the database. Before you can assign print quotas to a user with the edpykota. you must add this user or group to the database using pkusers

Command line usage :

pkusers [options] user1 user2 user3 ... userN
options :

- -a — –add
  -Adds users if they don't exist on the database.
  -If they exist, they are modified unless
  –s——skipexisting is also used.
- -d — –delete
  -Deletes users from the quota storage.
- -D — –description d
  -Adds a textual description to users or groups.
- -g — –groups
  -Edit users groups instead of users.
- -L — –list
  -Lists users or groups.
- -l — –limitby l
  -Choose if the user is limited in printing by its account balance or by its page quota. The default value is 'quota'. Allowed values are 'quota' 'balance' 'noquota' 'noprint' and 'nochange' :

  - quota : limit by number of pages per printer.
  - balance : limit by number of credits in account.
  - noquota : no limit, accounting still done.
  - nochange : no limit, accounting not done.
  - noprint : printing is denied.

- -b — –balance b
  -Sets the user's account balance to b.
  -Account balance may be increase or decreased if b is prefixed with + or -. WARNING : when decreasing account balance, the total paid so far by the user is decreased too.
  -Groups don't have a real balance, but the sum of their users' account balance.
- -r — –remove
  -In combination with the –ingroups option above, remove users from the specified users groups.

.

examples :

pkusers –add rustam tom john

This will add users rustam, tom and john, to the quota database.

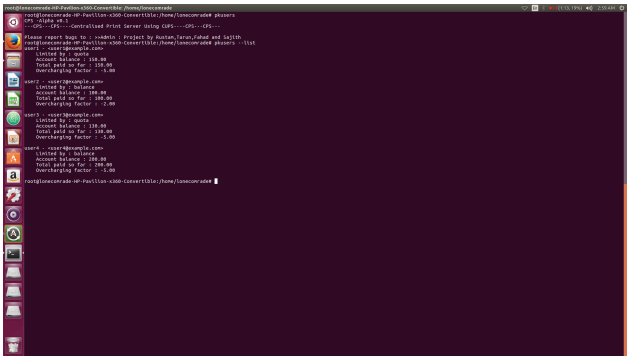
pkusers –limitby balance tom

This will tell CPS to limit tom by his account's balance when printing.

pkusers –balance +10.0 john

This will increase john's account balance by Rs 10,

SCREENSHOTS:



EDPYKOTA Manage User Quota.

This command enables you to create, manage or delete print quota entries for users on printers or printers groups. By default, before being allowed to print through CPS, a user must exist in the database and have a print quota entry on every printer he should be allowed to use.
But this is not sufficient to allow user Rahul to print. You have to create a print quota entry for Rahul on all printers he is allowed to print to. The easiest way to do so is to type:

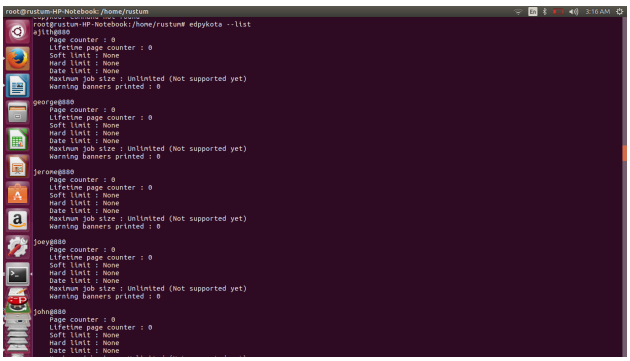
edpykota –add Rahul

Command line usage:

edpykota [opons] user1 user2 ... userN

- -a — –add // -Adds users or groups print quota entries if they don't exist in database.
- -d — –delete // -Deletes users or groups print quota entries. Users or groups are never deleted, you have to use the pkusers command to delete them.
- -P — –printer p // -Edit quotas on printer p only. The default value is *, meaning all printers. You can specify several names by separating them with commas
- -L — –list // -Lists users or groups print quota entries.

examples :
edpykota –add john paul george
This will create print quota entries for users john, paul, george and on all printers. It will have no limit set. SCREENSHOTS:



PKNOTIFY Connection / Session Manager.

Notifies or ask questions to end users who launched the CPS-client application. Client-side and authentication takes place through this command.

Other functions include:

Display an informational message to the end user

Display a message to the end user and asks for confirmation or cancellation. The result is sent back to the calling program, usually on the print server
Asks the end user to fill a simple form containing a numb er of labelled fields. The values the user typed in are sent back to the calling program, usually on the print server

Automatically shutdown the PyKotIcon application when asked to do so by an authorized host, usually the print server.

Command line usage :
pknotify [options] [arguments]

OPTIONS:

- -d — –destination h[:p] //-Sets the destination hostname and optional port onto which contact the remote CPS client application. When not specified, the port defaults to 7564
- -a — –ask //- Tells pknotify to ask something to the end user. Then pknotify will output the result.
- -C — –checkauth //-When –ask is used and both an 'username' and a 'password' are asked to the end user, then pknotify will try to authenticate the user through PAM.
- -c — –confirm //-Tells pknotify to ask for either a confirmation or abortion
- -n — –notify //-Tells pknotify to send an informational message to the end user.

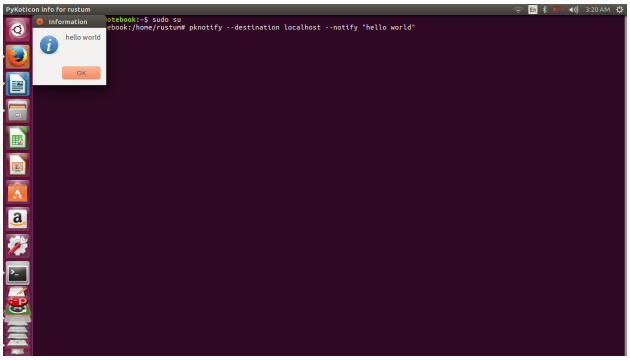
ARGUMENTS:

- -a — –ask : //-Several arguments are accepted, of the form "label:varname:defaultvalue". The result will be printed to stdout in the following format : VAR1NAME=VAR1VALUE VAR2NAME=VAR2VALUE ...
- -c — –confirm : //-A single argument is expected, representing the message to display. If the dialog is confirmed then pknotify will print OK, else CANCEL.
- -n — –notify : //- A single argument is expected, representing the message to display. In this case pknotify will always print OK.

Examples: pknotify -d client:7654 –confirm "This job costs 10 credits"

Would display the cost of the print job and asks for confirmation.

SCREENSHOTS:



DUMPYKOTA Database Dumper.

dumpykota can export CPS's data s in a number of formats. The supported output formats are comma separated values, semicolon separated values, tab separated values, and XML.
For the print job history's datas, a special format identical to CUPS' page*log format is also supported. This allows you Print Analyzer which is a log file analyzer for CUPS. More formats may be added in the future. In particular, SQLan The possible dump data types are* :

- Users
- Users groups
- Printers
- Printers groups membership
- Users groups membership
- Users print quota entries
- Users groups print quota entries
- History of payments
- History of print jobs
- Billing codes

You can then import the dumped datas into a spreadsheet for example, if you want to create complex reports with nice looking graphs and the like.
An important feature of this command is the possibility to use a simple but powerful filtering mechanism to only export the datas you want. You can pass any number of filter expressions which will be ANDed together to select only certain records in the database. For example the filter expression username=fahad would only dump datas pertaining to user fahad while the filter expression start=2005 used when dumping the history would only dump jobs printed during the year 2005. However when dumping the complete database, filters can't be used.

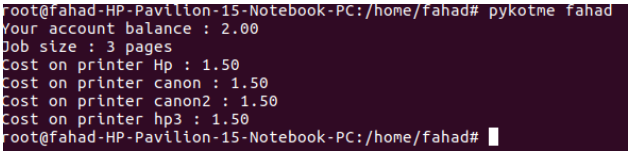
PYKOTME Quote Generator

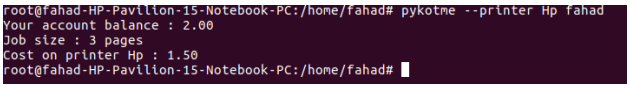
CPS features a print quote generator, named pykotme. This command line tool can be used to learn in advance how much a print job will cost to you if you really send it to a printer.
You can then decide if printing your document is a good idea or not, and do it knowingly. To get a print quote, you have to launch pykotme from the command line, passing your print jobs content in any format recognized by CPS 1 , either in the form of one or more file names on the command line, or on its standard input (i.e. in a shell pipe or redirection).
Without any command line argument, pykotme automatically reads your jobs data from its standard input.

command line usage :
pykotme [options] [files]

options :

- -v — –version //-Prints pykotme's version number then exits.
- -h — –help //-Prints this message then exits.
- -P — –printer p //-Gives a quote for this printer only. Actually p can use wildcards characters to select only some printers. The default value is *, meaning all printers.





REPYKOTA -Report Generator

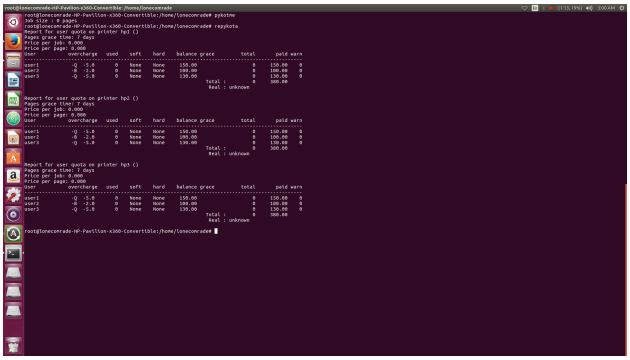
CPS features a quota report generator, named repykota , with which you can print the current state of the repykota database. repykota behaves differently when it is launched by a CPS administrator, compared to when it is launched by a normal user. In the first case, the print repykota report will contain current account balance, soft and hard limits, number of pages printed since last reset, total number of pages printed, total paid, for possibly all users or all groups, depending on command line options.
In the second case, i.e. when repykota is launched by a normal user, the user will only be allowed to see informations about himself or the groups he his a member of.
Any user can limit the report to only one or more printers, by specifying the -P or –printer command line option, followed by one or more printer name or wildcard. If more than one printer name or wildcard is used, they must be separated by commas. Launching repykota with no arguments will generate a complete print quota report, depending on what you are allowed to see.

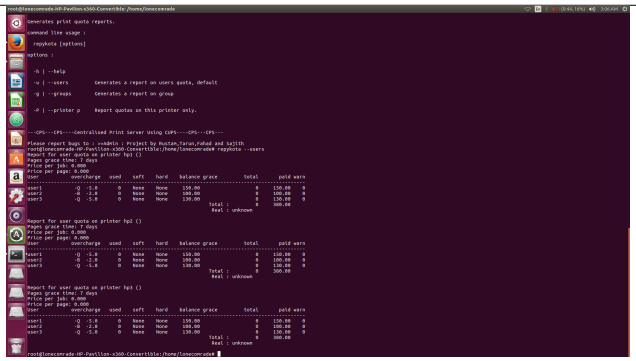
Command line usage :
repykota [options]
options :

- -v — –version // -Prints repykota's version number then exits.
- -h — –help // -Prints this message then exits.
- -u — –users // -Generates a report on users repykota , this is the default.
- -P — –printer p // -Report repykota s on this printer only. Actually p can use wildcards characters to select only some printers. The default value is *, meaning all printers. You can specify several names or wildcards,

EXAMPLES:

repykota users
repykota –version




Generates print quota reports.
command line usage :
repyquota [options]
options :
-h | --help
-u | --users Generates a report on users quota, default
-g | --groups Generates a report on group
-P | --printer p Report quotas on this printer only.
---CPS---CPS----Centralised Print Server Using CUPS----CPS---CPS---
Please report bugs to : sv4dmin : Project by Rustam,Taran,Rahad and Sajith
root@lonecoverade-HP-Pavilion-x360-Convertible:/home/lonecoverade# repyquota --users
Report for user quota on printer hp1 ()
Pages grace time: 7 days
Price per job: 0.000
Price per page: 0.000
user overcharge used soft hard balance grace total paid warn
user1 -Q- -1.0 0 None None 150.00 0 150.00 0
user2 -A- -1.0 0 None None 100.00 0 100.00 0
user3 -Q- -1.0 0 None None 130.00 0 130.00 0
Total : 0 380.00
Real : unknown
Report for user quota on printer hp2 ()
Pages grace time: 7 days
Price per job: 0.000
Price per page: 0.000
user overcharge used soft hard balance grace total paid warn
user1 -Q- -1.0 0 None None 150.00 0 150.00 0
user2 -A- -1.0 0 None None 100.00 0 100.00 0
user3 -Q- -1.0 0 None None 130.00 0 130.00 0
Total : 0 380.00
Real : unknown
Report for user quota on printer hp3 ()
Pages grace time: 7 days
Price per job: 0.000
Price per page: 0.000
user overcharge used soft hard balance grace total paid warn
user1 -Q- -1.0 0 None None 150.00 0 150.00 0
user2 -A- -1.0 0 None None 100.00 0 100.00 0
user3 -Q- -1.0 0 None None 130.00 0 130.00 0
Total : 0 380.00
Real : unknown
root@lonecoverade-HP-Pavilion-x360-Convertible:/home/lonecoverade#

6.2 CPSCLIENT - Client-side S/W Package

6.2.1 Description

CPSClient is a generic authentication dialogue manager. It is used as a client side companion for Centralized Print Server CPSvAlpha software.
But it can be modified to act as an authentication or credentials dialogue manager for any other applications as well by changing port address to target and necessary intelligible changes.

6.2.2 Working

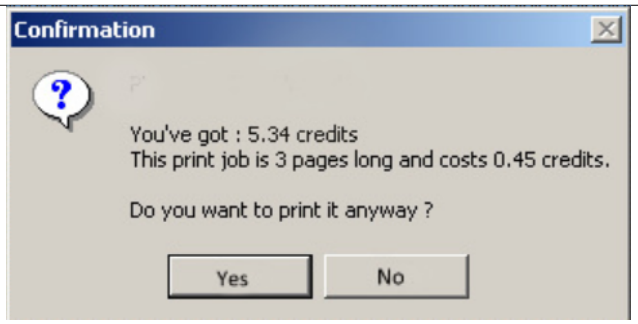
CPSClient is meant to be installed on each client system and run in background . So when a user sends a print request to one of the configured printers under CUPS it triggers the CPSClient listener and responds with a pop up dialogue asking for credentials.

Upon entering valid credentials CPS Client sends it to the CPS Server and validates the information and returns a confirmation dialogue box containing
The credit balance for the respective client
The print job size
Cost of current job

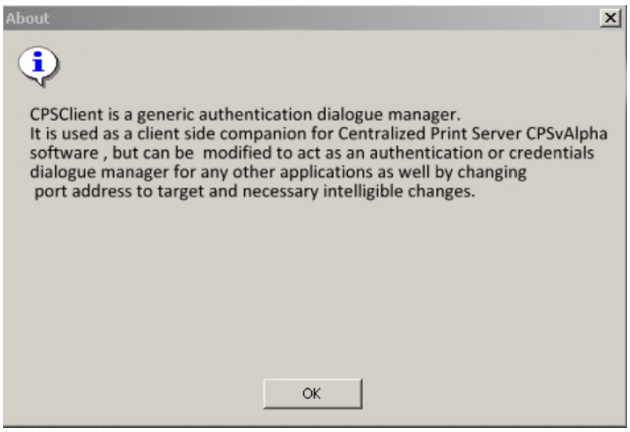
The port address is set as 7654 by default but can be assigned any dynamic port address.
So the socket address is

```
print('localhost 7654')
```

On confirmation the job is validated and response is sent to the Server which then performs the database operations and forwards the request to CUPS Scheduler for printing.



Additionally there exists a About buton which gives the information about the s/w package



6.2.3 Implementation

The package consists of 2 files ,

A python file which implements the communication with the server.

A Visual Basic script file which renders the pop up dialogue box

PYTHON PROGRAM

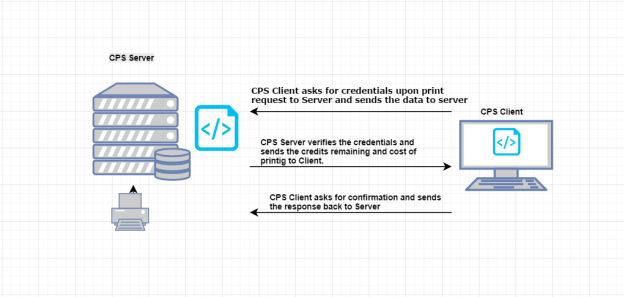
It
Initiates connection with the server
Checks dependency tree
Handles errors due to dependencies
Maps content into vbs frame , asks for credentials sent it to server authenticates asks for confirmation and sends the response to server.

VISUAL BASIC SCRIPT

VBS Script is used to render a pop up dialogue box for rendering credentials and confirmation.

```
wx.Dialog.__init__(self, parent, id,\
        _("CPS data input"), \
        style = wx.CAPTION \
            | wx.THICK_FRAME \
            | wx.STAY_ON_TOP\
            | wx.DIALOG_MODAL)
```

6.2.4 Process Diagram



6.2.5 CONNECTION TROUBLESHOOTER SCRIPT

Run CPSClient with no arguments on the same host, then
$ python ./test.py localhost 7654

```
import sys
import socket
import xmlrpclib

def main(arguments) :
    server = xmlrpclib.ServerProxy("http://%s:%s" % (arguments[0], arguments[1]))

    message1 = "Testing CPS Client\n init.... !"
    server.showDialog(xmlrpclib.Binary(message1.encode("UTF-8")), False)

    message2 = "Confirm ?"
    result = server.showDialog(xmlrpclib.Binary(message2.encode("UTF8")), True)
    print "Response  : %s" % result

      result = server.askDatas([xmlrpclib.Binary(v) for v in ["Username", "Password", "Country"]],\
                    ["username", "password", "country"], \
                    {"username": xmlrpclib.Binary(""), \
                     "password": xmlrpclib.Binary(""), \
                     "country" : xmlrpclib.Binary("")})
    if result["isValid"] :
        print "Answers :"
        print "\n".join(["\t%s => '%s'" % (k, v.data) for (k, v) in result.items() if k != "isValid"])
        answer = "You answered \n%s" % "\n".join(["%s => '%s'" % (k, v.data) for (k, v) in result.items()
if k != "isValid"])
        server.showDialog(xmlrpclib.Binary(answer.encode("UTF8")), False)
    else :
        print "The answers are not valid."

    message3 = "As soon as you'll click on the button below,to exit."
    server.showDialog(xmlrpclib.Binary(message3.encode('UTF-8')), False)
    server.quitApplication()

if __name__ == "__main__" :
    if len(sys.argv) < 3 :
        sys.stderr.write("usage : %s pykoticon.hostname_or_ip_address _TCPPort\n" % sys.argv[0])
    else :
```

6.3 WEB INTERFACE

6.3.1 COMMON GATEWAY INTERFACE - Python

CGI (Common Gateway Interface) is a standard way of running programs from a Web server. Often, CGI programs are used to generate pages dynamically or to perform some other action when someone fills out an HTML form and clicks the submit button. AOL server provides full support for CGI v1.1.

Basically, CGI works like this:
A reader sends a URL that causes the AOLserver to use CGI to run a program. The AOLserver passes input from the reader to the program and output from the program back to the reader. CGI acts as a "gateway" between the AOLserver and the program you write. The program run by CGI can be any type of executable file on the server platform. For example, you can use C, C++, Perl, Unix shell scripts, Fortran, or any other compiled or interpreted language. You can also use Tcl scripts with CGI, though the extensions to Tcl described in Chapter 5 of the AOLserver Tcl Developer's Guide are not available through CGI. With AOLserver, you have the option of using the embedded Tcl and C interfaces instead of CGI. Typically, the Tcl and C interfaces provide better performance than CGI. (See the AOLserver Tcl Developer's Guide for information on the Tcl interface and the AOLserver C Developer's Guide for information on the C interface.)

You may want to use CGI for existing, shareware, or freeware programs that use the standard CGI input, output, and environment variables. Since CGI is a standard interface used by many Web servers, there are lots of example programs and function libraries available on the World Wide Web and by FTP

6.3.2 Purpose.

Each web server runs HTTP server software, which responds to requests from web browsers. Generally, the HTTP server has a directory (folder), which is designated as a document collection files that can be sent to Web browsers connected to this server.
For example, if the Web server has the domain name example.com, and its document collection is stored at

```
print('/usr/local/apache/htdocs ')
```

.
the local file system, then the Web server will respond to a request for http://example.com/index.html by sending to the browser the (pre-written) file

```
print('     /usr/local/apache/htdocs/index.html.')
```

HTTP provides ways for browsers to pass such information to the web server, e.g. as part of the URL. The server software must then pass this information through to the script somehow. Conversely, upon returning, the script must provide all the information required by HTTP for a response to the request: the HTTP status of the request, the document content (if available), the document type (e.g. HTML, PDF, or plain text), etcetera. Initially, different server software would

use different ways to exchange this information with scripts. As a result, it wasn't possible to write scripts that would work unmodified for different server software, even though the information being exchanged was the same. Therefore, it was decided to establish a standard way for exchanging this information: CGI (the Common Gateway Interface, as it defines a common way for server software to interface with scripts). Webpage generating programs invoked by server software that operate according to the CGI standard are known as CGI scripts.
This standard was quickly adopted and is still supported by all well-known server software, such as Apache, IIS, Nginx, and (with an extension) node.js-based servers.
An early use of CGI scripts was to process forms. In the beginning of HTML, HTML forms typically had an "action" attribute and a button designated as the "submit" button. When the submit button is pushed the URI specified in the "action" attribute would be sent to the server with the data from the form sent as a query string. If the "action" specifies a CGI script then the CGI script would be executed and it then produces a HTML page.

6.3.3 Using CGI Scripts.

A web server allows its owner to configure which URLs shall be handled by which CGI scripts. This is usually done by marking a directory within the document collection as containing CGI scripts - its name is often cgi-bin.

```
print('/usr/local/apache/htdocs/cgi-bin ')
```

could be designated as a CGI directory on the web server. When a Web browser requests a URL that points to a file within the CGI directory (e.g., http://example.com/cgi-bin/printenv.pl/with/additional/path?a then, instead of simply sending that file (/usr/local/apache/htdocs/cgi-bin/printenv.pl) to the Web browser, the HTTP server runs the specified script and passes the output of the script to the Web browser.

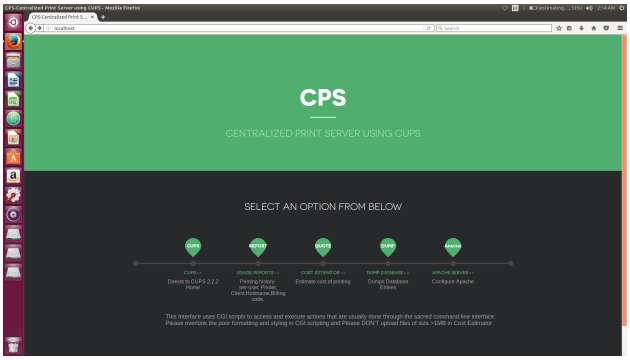
That is, anything that the script sends to standard output is passed to the Web client instead of being shown on-screen in a terminal window.

6.3.4 Description

The Web Interface for CPS consists of 3 CGI Scripts wjich are run on Apache Server.

- Quote Generator
- Report Generator
- Database Dumper

LANDING PAGE / HOME PAGE

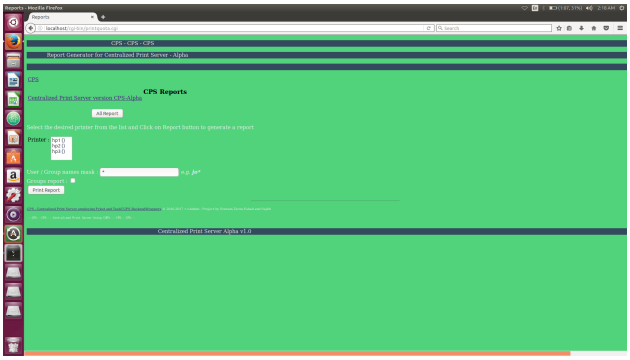


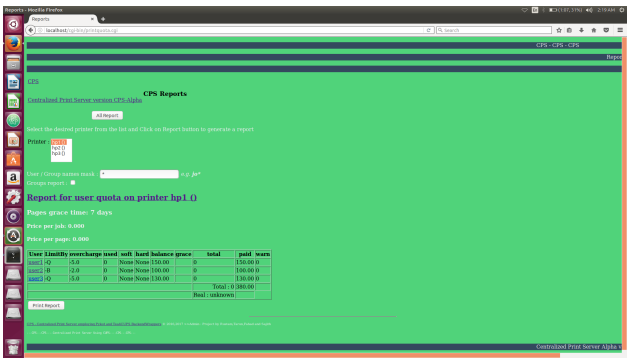
REPORT GENERATOR

printquota.cgi offers the same functionalities as the repykota command line action, but makes them available through web browser instead of terminal window. It enables us to generate reports based on printer used.

Can be accessed at

```
print('localhost/cgi-bin/printquota.cgi')
```



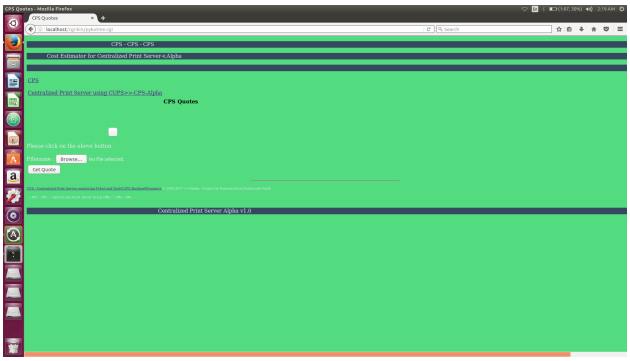


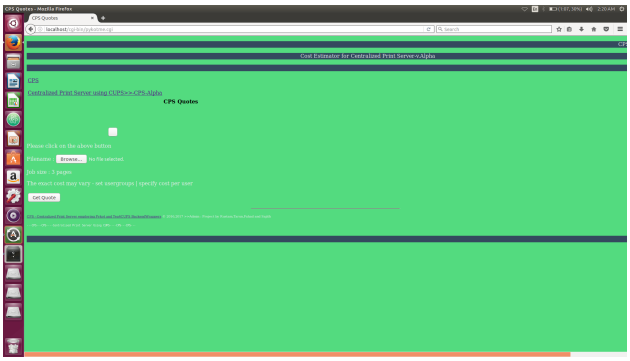
QUOTE GENERATOR

pykotme.cgi offers the very same functionalities as the pykotme command line tool, but makes them available to you in your web browser.
Quote generator enables us to check the cost of printing based on number of pages to be print.

Can be accessed at

```
print('localhost/cgi-bin/pykotme.cgi')
```

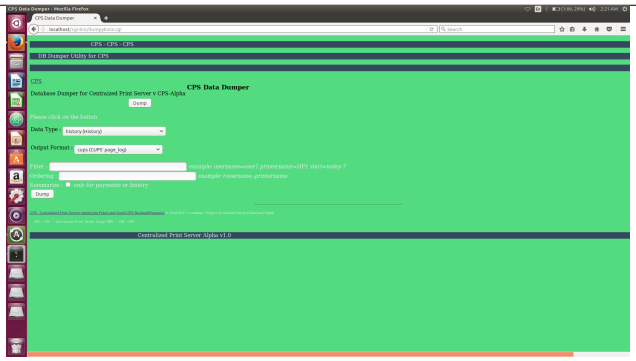




DATABASE DUMPER

dumpykota.cgi offers the same functionalities as the dumpykota command line tool, but makes them available through your web browser.It enables us to dump the database values.
Can be accessed at

```
print('localhost/cgi-bin/pykotme.cgi')
```



6.3.5 Prerequisites

- CUPS
- Python language interpreter
- mxDateTime
- Python-PygreSQL
- Python-PAM
- Python-OSD

Chapter 7

Conclusion and Future Enhancement

This chapter is based on the conclusions of what we have done so far and how the system can be further enhanced with an increase in requirements.

7.1 Conclusions

The project was aimed at the development and implementation a centralized print server for a Local Area Network (here College LAN).
The host server was setup and is running CUPS. Host accepts print jobs from client computers, process them, and send them to the printer. Host also performs Database retrieval and modifications based on print request.
Client-side software was implemented using python and Visual Basic which provides a pop up dialogue for credentials but was unable to make it run in background so that it would be activated on trigger.
A Web Interface using CGI-Python was setup and locally hosted in the Server using Linux Apache Server and is accessible as http://localhost

7.2 Future Scope

- Strict Printing Policy Enforcement based on User Groups
- Implementation of a Print Cache aimed to retrieve documents being printed
- IPP Printing from Remote Host over Internet

References

[1] Python Tutorials [Online] The New Boston Youtube Channel

[2] Apache 2 CGI Documentation [Online] https://httpd.apache.org/docs/2.4/howto/cgi.html

[3] Thomas C Bartee: Introduction to Computer Science: Third Edition, McGraw-Hill, New York, 1975

[4] Setup CGI [Online] https://code-maven.com/set-up-cgi-with-apache

[5] StackOverflow [Online] - For Anything and Everything

[6] James Marshall CGI Scripting - [Online] https://www.jmarshall.com/easy/cgi/

[7] Tutorials Point [Online] - https://www.tutorialspoint.com/perl/perlcgi.htm

[8] Tea4CUPS [Online] - https://directory.fsf.org/wiki/Tea4CUPS