CENG 492
REVISED DESIGN REPORT
SECURE VIDEO STREAMING PROXY SERVER

Sponsored by

Portakal Technologies

TEAM PRIME'

Ant Ongun Kefeli

Doğan Poyraz

Yurdakul Göksu Orhun

Onur Cem Sırlı

February 26, 2011

Table of Contents

1. Introduction

1.1. Problem Definition

Video streaming and processing is issued in many areas. For example, nowadays there are car surveillance systems in Ankara, which aims to determine the speed of cars and fine them if they exceed the allowed limit. However, the system is not able to do all the work. Now, it just can determine the car speed and detect speed limit violations and record the relevant frames. Then someone needs to check all these frames and fine them one by one by hand.

Software is being developed to automate this procedure by removing the human work completely out of the system. Instead of the police officers manually writing tickets, software will automatically detect and fine the violators. Note that, only in Ankara, cameras pinpoint around 18.000 traffic violations per day, however; only 2000 of them can be fined with manual labor. Therefore, automatizing the fining procedure will raise an important problem. Currently only 2.000 traffic violation footages are being stream from the central media server, however, when the update is complete the number of requested streams will raise to 18.000. The problem is that current infrastructure will not support the increased load and there does not exist a system which can distribute this load automatically between number of reverse proxy servers to maintain the functionality of the fining process.

Another example is face recognition software that the governments are using to track and detect the criminals. These systems use the footage that the distributed camera systems provide, like MOBESE or security camera networks in some malls. The software processes the video footage for faces of wanted criminals, when one is detected the software tries to find out more about the activities of the person, like who he/she met, where he/she is living etc. To dig out this information the software needs to process more footage to find the travel route and recent activities of the mentioned criminal. Clearly the whole process needs hours and hours of video footage which is divided into few seconded segments. As central media server cannot respond to this volume of video requests face recognition software output is vastly outdated due to delay of the required footage.

In both of these systems, there is an overwhelming work load on the server since these operations require dramatic amount of video streams per unit of time. And if multiple systems that accesses heavily to the media server run concurrently, load is multiplied and after a moderate amount of connections to the server, system cannot scale the and it may start to collapse causing serious issues on the other system that depends on the media server. Our system will mainly focus on the scalability issue by introducing reverse proxy servers that will share the load.

1.2. Purpose

This document describes the initial design plan for our project and how the requirements stated in the Software Requirement Specifications will be fulfilled. Requirements in the SRS document will be translated into structural components. Design concepts and general design architecture of the project will be explained in a detailed way. Design issues discussed here are the guidelines we intend to follow; however, they are always subject to change. The intended audience is prospective software developers and all users.

1.3. Scope

This initial design document contains design information which demonstrates architecture, modules, functions, features, graphical user interface and tools. It describes how the system works properly in a detail. The architecture is intended as an infrastructure for future versions.

1.4. Overview

Rest of the document contains eight more chapters. Second chapter is System Overview in which we will give a detailed design of the system. Then the Design Considerations follows. In this part, design issues related to assumptions, dependencies and constraints will be discussed. The next chapter is the Data Design. This chapter will include information and data domain of the system and how it is transformed into structures. In the fifth chapter, we will describe architecture of the system by examining the components. Next, User Interface Designs are explained with screenshots from both similar projects and our templates we have been planning to use. Then, libraries and tools we are currently considering to choose among them will be listed. Finally, we will present our planning for the project and finish with conclusion.

1.5. Definitions, Acronyms and Abbreviations

CARP: Common Address Redundancy Protocol

CLI: Command Line Interface

Codec: Coding Decoding

COTS: Commercial, off-the-shelf

DHCP: Dynamic Host Configuration Protocol

DNS: Domain Name System

GUI: Graphical User Interface

HA: High Availability

HTTP: Hypertext Transfer Protocol

MPEG: Moving Pictures Experts Group

NAT: Network Address Translation

PPPoE: Point-to-Point Protocol over Ethernet

PUI: Prime User Interface

RF: Radio Frequency

RRD: RF Receiving Device

VLC: VideoLAN Client

VPN: Virtual Private Network

1.6. References

- VideoLAN - The cross-platform streaming solution, http://www.videolan.org/vlc/streaming.html

- Closeness of Actions and Objects in GUI Design (Jakob Nielsen's Alertbox), http://www.useit.com/alertbox/action-object-closeness.html
- VideoLAN Wiki, http://wiki.videolan.org/
- Software Requirements Specification Secure Video Streaming Proxy Server, Team Prime', Dec 17, 2010
- MediaTomb - Free UPnP MediaServer, http://mediatomb.cc/
- Projects @ Apache Documentation, http://projects.apache.org/docs/index.html
- nginx, http://nginx.org/en/
- PFSenseDocs, http://doc.pfsense.org/index.php/Main_Page
- Apsis Gmbh, http://www.apsis.ch/pound/

2. System Overview

For Media Server, Web Server and Load Balancer we will be using COTS software. Our goal is to make this system work as a secure and scalable one. So coding the sub-components is not in our scope at the moment.

We want to explain COTS with a few words. COTS stand for 'Commercial, off-the-shelf'. It represents technologies which are ready and available for sale, lease or licensing. It often refers to computer software or hardware and it may include free software with commercial support.

There is already open source COTS software serving our purpose, so we will be using them. Our system consists of 3 main parts. First one is media server. It is the component that interacts with cameras, records data from them and serves them to the web servers as requested.

Second one is web servers. It gets data from the media server and sends them to clients according to the load balancer's distribution.

The last one is the load balancer. It is the first component to handle incoming connections. According to the load of web servers, it distributes incoming connections to the web servers so that none of the servers are overloaded.

3. Design Considerations

3.1. Design Assumptions, Dependencies and Constraints

Our assumptions about the system:

- Media streams are working fine.
- Streams and server have enough bandwidth.
- System has low latency.
- Video processing and networking libraries are available on the system.
- One load balancer will be sufficient for our simple design and it will be stable.

3.2. Design Goals and Guidelines

In our system, we aim to create a highly scalable system rather than an efficient one. Since we will be using COTS tools mostly, our system's efficiency will be determined by our tool choices mostly. Of course we will try to be as efficient as possible it is not our main concern. Our main issue is to serve as many clients as possible without having any problems.

Our system's security issues will be mostly handled by load balancer since it handles incoming connections. It will not allow unauthorized accesses to the system. HTTP authentication protocol will be implemented in order to authenticate connections to the system. Users will authenticate themselves by supplying a username and password. The password will be verified by using HTTP digest access authentication in order to prevent undesired interception of the access credentials. All HTTP traffic will be encrypted using SSL encryption which will prevent theft, undesired access and/or corruption of data. We will utilize a firewall to secure the system from malicious attacks.

Also there might be a VPN installed between the cameras and media server. It is good for security of overall system but that issue is not our score at the moment.

Our system will be very simple to use. It has a few functions for people's use. Most of its job is internal and not available to users. There will be a web interface for clients to use and an admin interface for basic operations for system management.

4. Data Design

4.1. Data Description

Video File: Media Server records the video streams while serving them to users. As video files are relatively large, recording should occur after encoding the systems with appropriate codex. Furthermore, array of storage discs should be managed to ensure concurrent recording of multiple video streams. RAID 0 architecture can be employed on the discs for fast retrieval of video files.

Configuration File: In each of the components, there are configuration files that are to be used by sub-components. For example, admin will pass commands to the media server, web server and load balancer via their specific configuration files. They will read the settings from the file and operate accordingly.

Both of these files will be stored on the hard disk directly. As mentioned above, for efficiency in big video files, RAID 0 configuration may be used on array of disks.

User Information: Reaching the video streams are restricted to authorized users. Therefore, it is obligatory to store user information and authorization. Storage could be implemented in a file or relational database system, database system is chosen for performance concerns, as simple file system does not perform well in cases where user pool is relatively large.

Stream: Each video stream has its own unique ID in the SVSPS. This ID is used as an identifier at both stream processing and stream retrieval processes. Each stream source has several attributes related to it such as IP etc.

This information is not specific files, so they will be stored in a database. We are planning to use MySQL as our database.

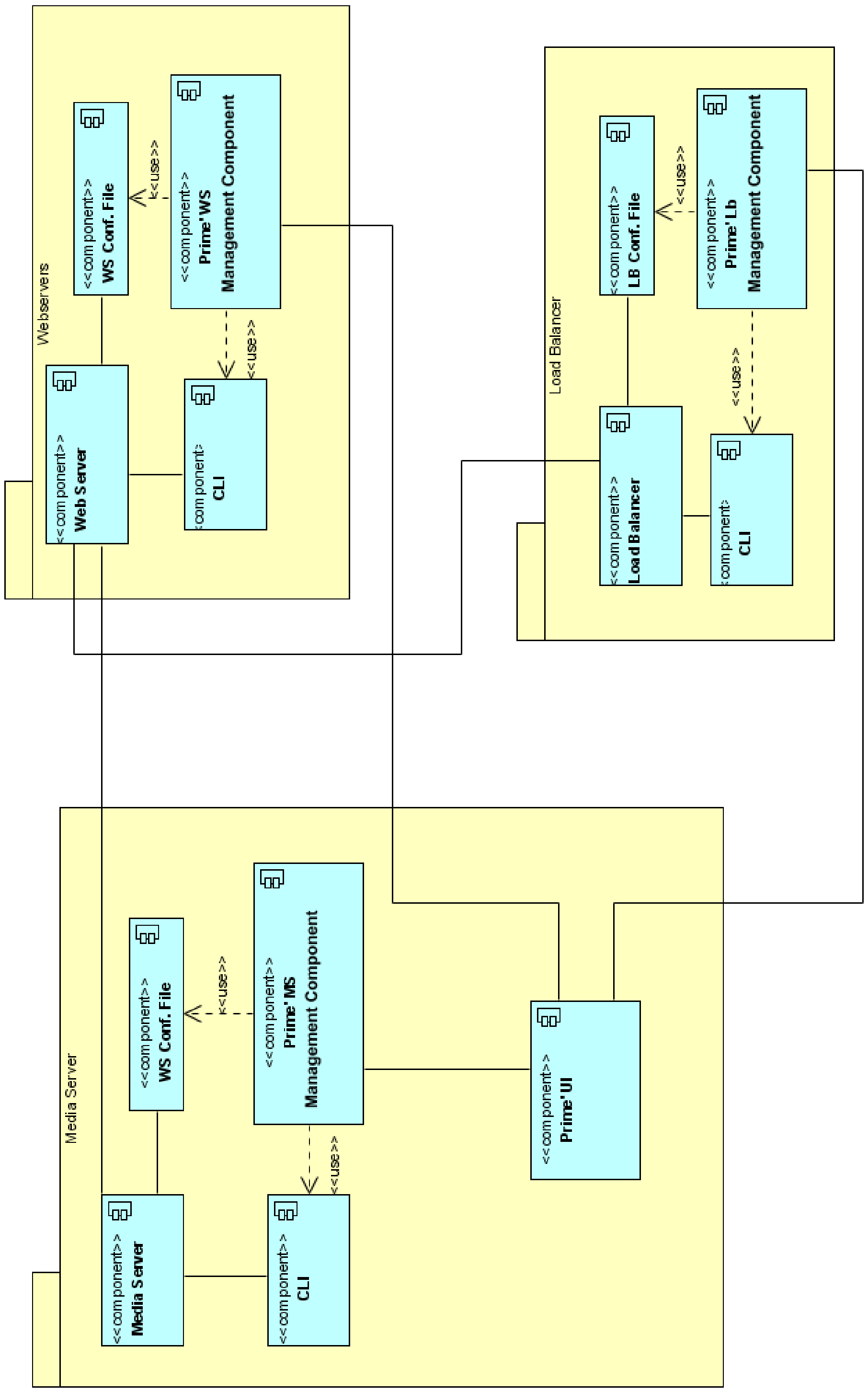
4.2. Data Dictionary

- Data structures:
  - Configuration file: .ini file that components read their settings from.
  - Stream info: Database entry that keeps info about each stream source.
  - User info: Database entry that keeps info about every user.
  - Video file: Media file that keeps the received video data separately from stream sources.

- Function prototypes:
  - intAddStream(Stream): takes a stream parameter that holds info about a stream and adds it to the media server. Returns a value depending on the result of the operation(succeeds or fails for some reason).
  - intAddUser(User): takes a user parameter that holds info about user to be added. Returns a value depending on the result of the operation (succeeds or fails for some reason).
  - intInjectConfigurationFile(String): takes a path parameter that points to a configuration file and sets it as active settings file. Returns a value depending on the result of the operation (succeeds or fails for some reason).
  - String LoadCheck(void): checks all web servers for their work load and decides to which server new clients will be directed. Returns identifier of the decided web server.
  - intRemoveStream(String): takes a string parameter that holds the unique ID of a stream and removes that stream from the media server. Returns a value depending on the result of the operation (succeeds or fails for some reason).
  - intRemoveUser(String): takes a string parameter that holds the unique ID of a user and removes that user from the user database. Returns a value depending on the result of the operation (succeeds or fails for some reason).

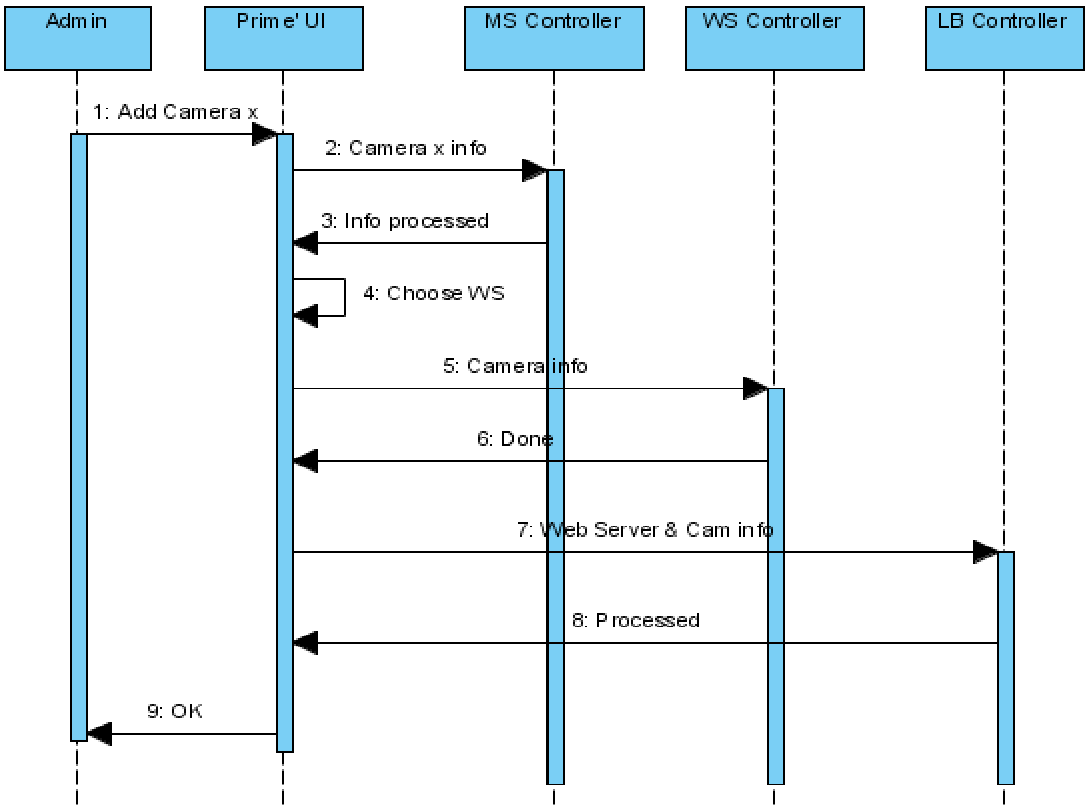
5. System Architecture

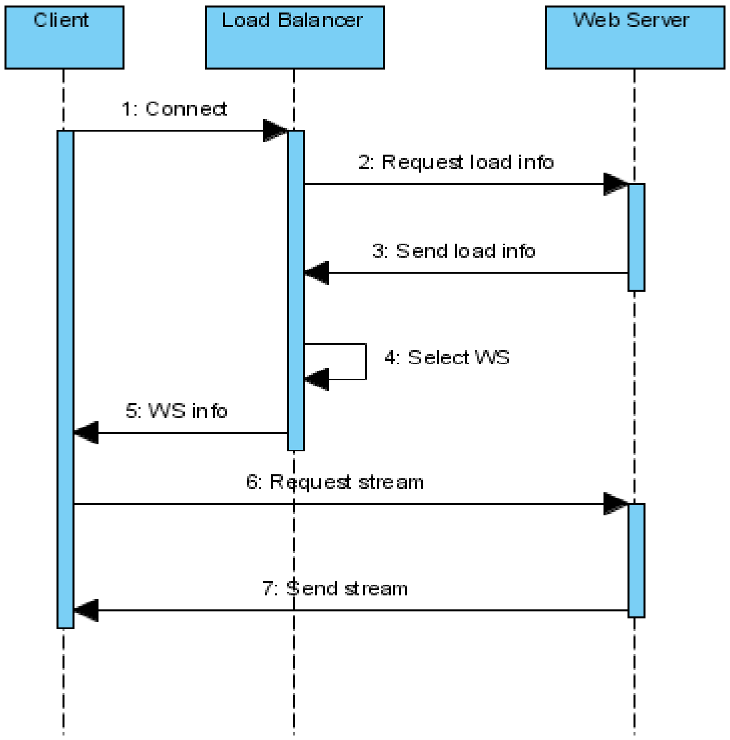
5.1. Architectural Design


Webservers
<<component>> WS Conf. File
<<use>>
<<component>> Prime'WS Management Component
<<use>>
<<component>> Web Server
<<component>> CLI
Load Balancer
<<component>> LB Conf. File
<<use>>
<<component>> Prime'Lb Management Component
<<use>>
<<component>> Load Balancer
<<component>> CLI
Media Server
<<component>> WS Conf. File
<<use>>
<<component>> Prime'MS Management Component
<<use>>
<<component>> Media Server
<<component>> CLI
<<component>> Prime'UI

Overall architecture of the system

5.2. Description of Components





Above there are sequence diagrams of adding camera to the system and sending data to a client. Operation of removing camera is almost same as adding camera. We will explain operations below

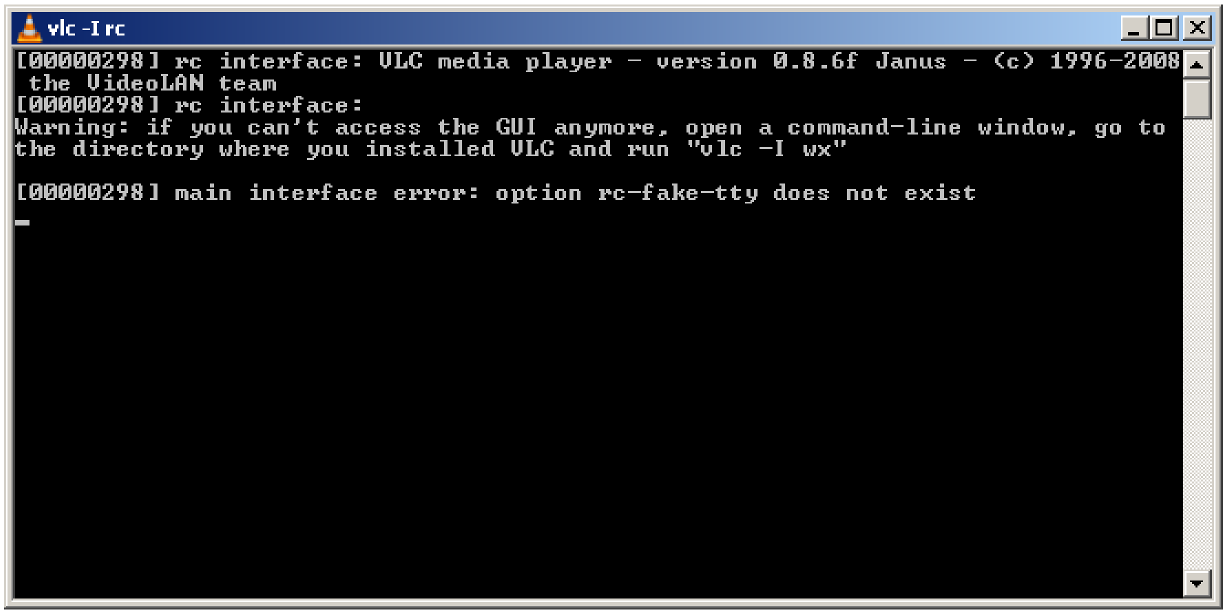
5.2.1. Media Server

5.2.1.1. Processing narrative for Media Server

Media server is a machine that collects data from streaming sources. While receiving and sending the data, media server may do some codec processing as well. In addition, resolution of the video streams will be most probably subject to change as it is preferable to standardize the resolution of the feeds. Media server receives data from cameras (or any other streaming sources) and delivers the processed streams to the media servers.

5.2.1.2. Media Server interface description

The entire media server tools that we have been examining use Command Line Interface for communication with user and other software. They usually do not have Graphical User Interfaces. For example here is a CLI screenshot from VideoLan Client, which is a tool that we most probably will be using.



```
[00000298] rc interface: VLC media player - version 0.8.6f Janus - (c) 1996-2008
 the VideoLAN team
[00000298] rc interface:
Warning: if you can't access the GUI anymore, open a command-line window, go to
the directory where you installed VLC and run "vlc -I wx"

[00000298] main interface error: option rc-fake-tty does not exist
_
```

Here are the CLI commands of VideoLAN, one of the media server alternatives we are considering.

To open a file with a codec, start VLC with:

% **vlc -vvv --codec ffmpeg my_file.mpg**

To receive a unicast UDP stream (sent by VLS or VLC's stream output), start VLC with:

% **vlc -vvvudp:[@:server_port]**

To receive a multicast UDP stream (sent by VLS or VLC's stream output), start VLC with:

% **vlc -vvvudp:@multicast_address[:server_port]**

To receive a HTTP stream, start VLC with :

% **vlc -vvv http://www.example.org/your_file.mpg**

To receive a RTSP stream, start VLC with:

% **vlc -vvv rtsp://www.example.org/your_stream**

5.2.1.3. Media Server processing detail

Media server is main database and data processing unit of our system. It has four main sub-components;

First main component is media server manager tool. This component will be responsible from; gathering the data from connected cameras, process the data in means of codec and resolution, store the data as video files in the storage devices, deliver the processed streams to the web servers according to the settings in the configuration file. Note that this process will display pipeline properties as for instance, a part of the stream could be at the processing stage while preceding seconds of the same stream is being delivered to the web servers. In addition, it is highly possible that this pipeline will be working for many streams in parallel. Therefore, this component will be drastically CPU-heavy, so the infrastructure hosting this component should be able to satisfy this need.

Second sub-component is the configuration file. It keeps the settings that will be used by the media server manager. Configuration file contains many parameters that is used to regulate processes of the media server. Some of these parameters are as follows; codec types, resolution of the output streams, TCP ports to be used, database information, log file path etc.

Another sub-component is Command Line Interface. It is used to command and modify behavior of the media server manager using Terminal interface of UNIX based systems. In the resulting system, this interface will not be used directly by the standard user or the admin, Prime' User Interface will map the functionalities of the CLI to a graphical interface.

Last sub-component of MS is the media server controller. This component will be designed and written by Prime' as a bridge for the communication between the general user interface and media server's CLI and configuration file.. It will deliver the commands from the user to the server by giving commands through CLI or modifying the configuration file and do some periodic checks and regulations on the system if needed.

5.2.1.4. Dynamic behavior of Media Server

Media server has two main dynamic events, namely, add and remove camera. This events can occur by manual stimulation by the admin or through automatic self-registry.
To add a camera (or any other streaming video source) to the list of cameras streamed to the media server admin needs to have distinguishing information (IP address of the source, format of the stream etc.) about the source and this information needs to be provided to the media server. To remove a video source, admins just needs to perform a graphical user interface operation which does not require any specific pieces of information about the source to be removed as information about the existing sources resides within the storage of the media server.
Second way of adding a streaming source is automatic self-registry. This method is the desired one on adding sources, as it is more dynamical and large quantities of cameras can be added much more rapidly than manual source addition. In this method, first of all cameras sends its information to the media server, then, if media server allows the connection, add operation is triggered, finally the source is added to the list and the streaming begins.

5.2.2. Web Servers

5.2.2.1. Processing narrative for Web Servers

Web Servers are responsible for distribution of data to users which is obtained from Media Server. Media server streams the videos using HTTP protocol in MPEG4 format. Web Servers have a vital role in scalability. For instance, for some random stream source, say X, Web Server receives one stream of X from media server but can feed up to 250 users with this stream. Therefore, a Web Server acts as a reverse proxy to the media server. User requests the stream from media server, but receives it from the Web Server, by this way work load on the media server gets drastically reduced.

5.2.2.2. Web Servers interface description

Web Servers will be controlled using with CLI as our current options Apache and nginx are both controlled with CLIs. Multiple Web Servers will exist on final implementation of the system, and these Web Servers will be running on UNIX based operation systems (most probably Linux). To control all Web Servers concurrently we will employ an open source tool called Puppet.

5.2.2.3. Web Servers processing detail

Web Servers can be virtual machines or real machines; however they consist of three components in both cases.

First one, Web Server Manager, is responsible for connection management. It establishes connection with media server and hosts feeds for users.

Second one is Configuration File. It holds necessary settings and options about streams.
Web Server Controller, the last one, is responsible for actualizing admins' or systems autonomous commands.

Web Server Manager and Configuration File will be chosen from open source alternatives. Web Server Controller will be designed and implemented by our team.

5.2.2.4. Dynamic behavior of Web Servers

Web Servers has four main dynamic events. These are adding and removing feed; activating and de-activating Web Server. All of these operations are initiated by Load Balancer; however, later two are consequences of first two operations. Adding feed operation results in adding another thread to user side and may cause a Web Server to be loaded to its limit. When a Web Server is loaded to its limit, a new Web Server will be activated. On the other hand, removing feed operation results in removing a thread from user side and may cause a Web Server to be completely empty, thus, it will be deactivated.

5.2.3. Load Balancer

5.2.3.1. Processing narrative for Load Balancer

Load Balancer stands between the client and the web servers. Its role is to balance the work load and command the web servers so that as the work load increases on the system, it is distributed among the servers so system can handle much more client connections.

Load Balancer's behavior is controlled by the configuration file supplied and modified by Prime' Load Balancer Management component. This configuration file contains parameters such as upper and lower limits of connections per web server. Note that Load Balancer component is a network firewall with load balancing capabilities, so it also acts as a security measure.

LB receives stream requests from the users. It checks the number of the users fed by each web server, if there is an appropriate web server that can handle a new feed; LB redirects the user using HTTP to the mentioned web server. If the active web servers are at their predetermined maximum capacity, LB injects a configuration to one of the idle web servers to activate it, and directs the incoming load to the activated web server.

5.2.3.2. Load Balancer interface description

Most of the load balancers have Command Line Interfaces. For the time being, we are considering pfsense, Pound and nginx as load balancer choices. We will examine them in more detail before making a final choice before our detailed design report.

Prime' Load Balancer Management Component will manage the load balancer using CLI and configuration file. However, note that LB can be classified as a daemon. It can make choices like deactivating a web server by observing the load on the web serves, without a command from a human or a system actor.

5.2.3.3. Load Balancer processing detail

The job of the Load Balancer is to distribute the connection load to the web servers so that no web server is over or under loaded. It has four sub-components, which are similar to the other components' sub-components. First one is load balancer core which does the main job, which is to be explained in 5.2.3.4. Second one is CLI to control the load balancer core from the command line. Third one is the configuration file that holds settings for balancer core to use and last one is load balancer controller which is a bridge between the load balancer and user interface.

Load Balancer is a very dynamic component. Almost the entire job it does is dynamic so its job will be explained in more detail in 5.2.3.4.

5.2.3.4. Dynamic behavior of Load Balancer

Load balancer is the most dynamic component in the system. It is the component that realizes our main goal, which is to scale the system so that it can handle large number of client connections. When a client tries to connect, connection is first handled by the load balancer. It first checks the work load on the web servers. According to the workloads it decides one of the two actions. First, if there is a server that has load below the maximum threshold, it sends the new connection to the server. Otherwise, it sends command to an idle web server, and then web server activates and starts to serve to the clients.

Other than coordinating the incoming connections, it also checks the workloads on the servers periodically and if there is a low loaded server, load balancer tries to distribute its load on other servers to reduce number of active web servers.

6. User Interface Design

6.1. Overview of User Interface

6.1.1. Admin Interface

The first thing an admin sees when he tries to accesses the system is the login screen. This is an essential part of user interface as it should be able to prevent theft, undesired access and/or corruption of data.

After logging in with a username and password, management selection screen appears. Management options of an admin are to manage streams or to manage users.

Stream management screen contains the list of registered streams to the system. There will be buttons to add or remove streams. Add stream button will initiate a window which requires admin to enter camera settings. Remove stream button will initiate another window for confirmation. Stream management screen will also have link to each stream and will be available to be seen in a media player window. Media player will have play/pause buttons and there will be sound control button as well.

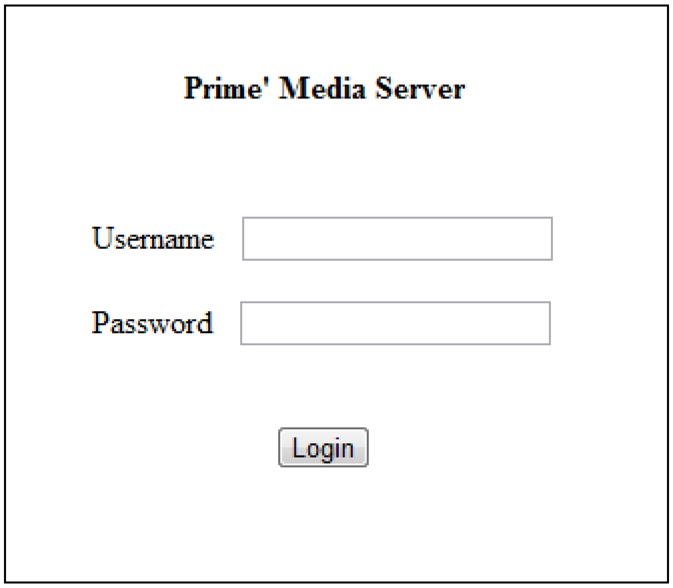
User management screen contains the list of user and their privileges. There will be add user and remove user buttons on this page. Add user button will require admin to enter user details and remove user button will open a confirmation window similar to the stream management removal window.

6.1.2. Standard User Interface

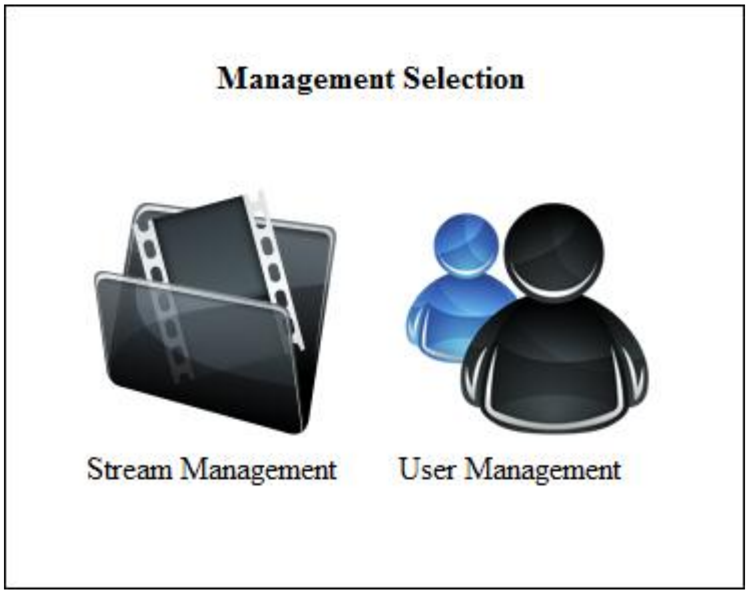
A standard user who does not have admin rights will face a login screen which is supposed to prevent access of unauthorized people.

After logging in the user will see a list of stream which he is authorized to access. These streams will have links for starting media player with their feed. Media player will be the same as the one admins use.

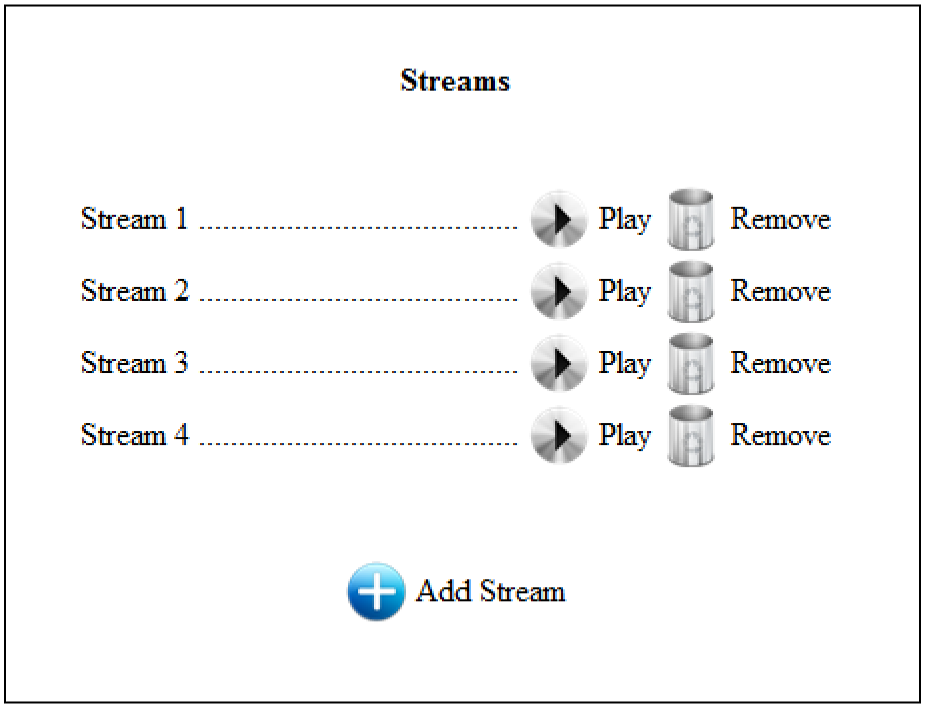
6.2. Screen Images



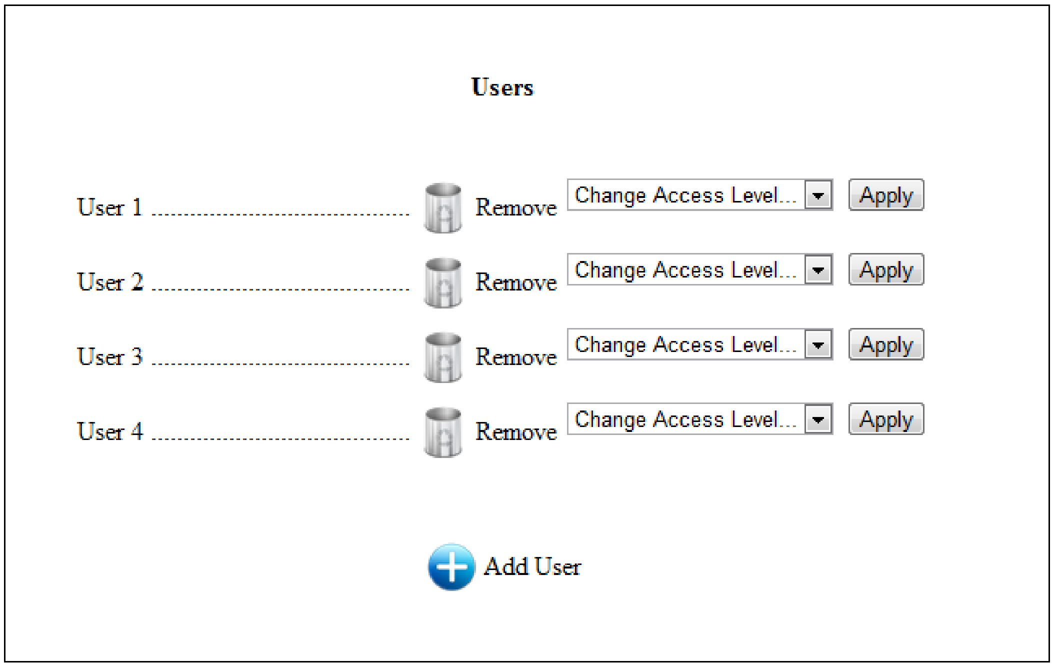
***Login Screen* -** Admins and users see this screen before they can use the system.



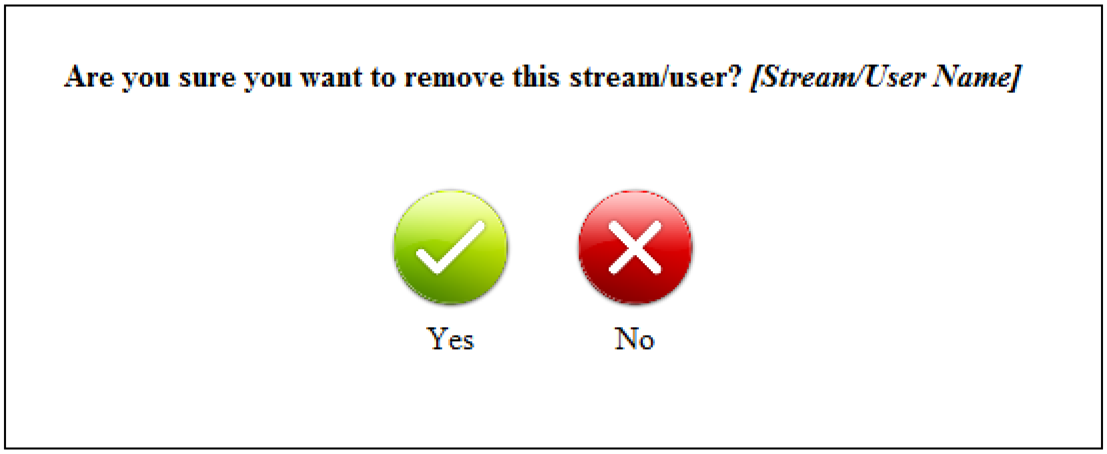
***Management Selection Screen* -** Admins can choose to manage streams or users on this screen which is available to admins after login procedure.



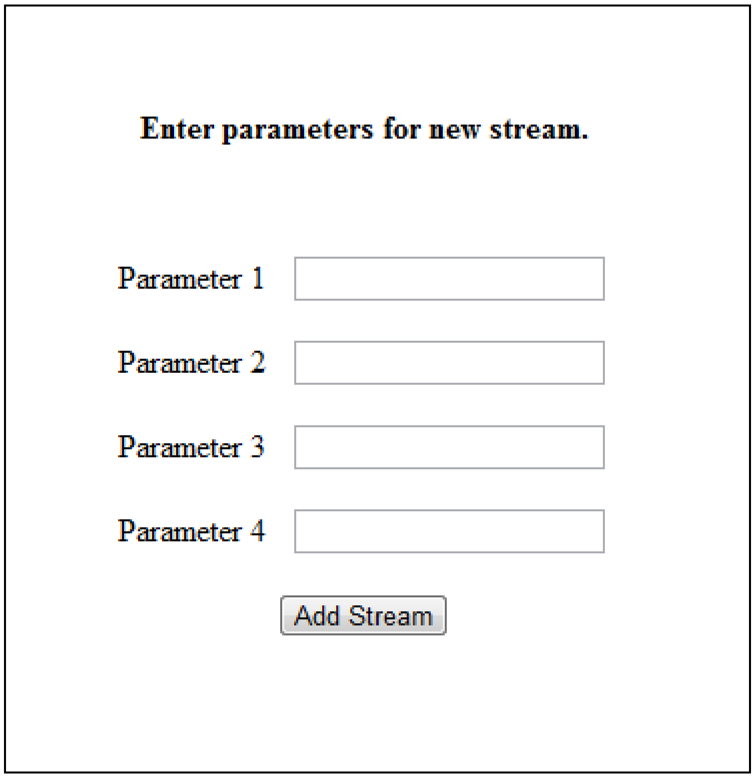
Stream Management Screen **-** This screen contains full list of streams and buttons to add/remove/play streams for admins. Users see the streams that they are authorized to see and "Remove" button is not visible to them.



User Management Screen **-** This screen contains full list of users and buttons to add/remove/change access level of users. Only admins can see this screen.

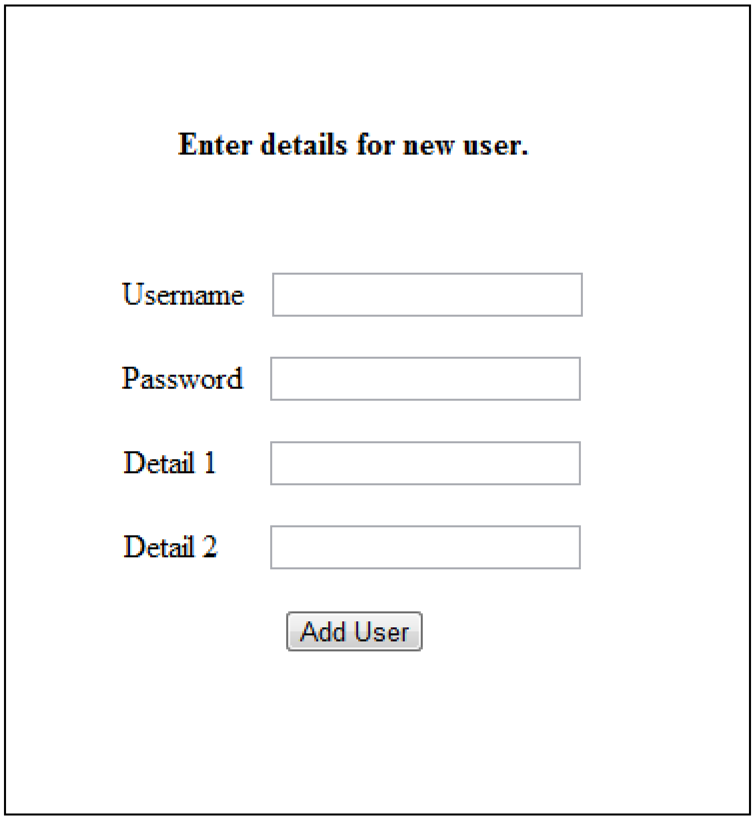

Are you sure you want to remove this stream/user? *[Stream/User Name]*

Removal Confirmation Screen - Admins see this screen whenever they want to remove a stream or user.


Enter parameters for new stream.

Parameter 1	
Parameter 2	
Parameter 3	
Parameter 4	

Add Stream Screen - Admins see this screen and prompted to enter parameters for stream they want to add a new one.



Enter details for new user.

Username

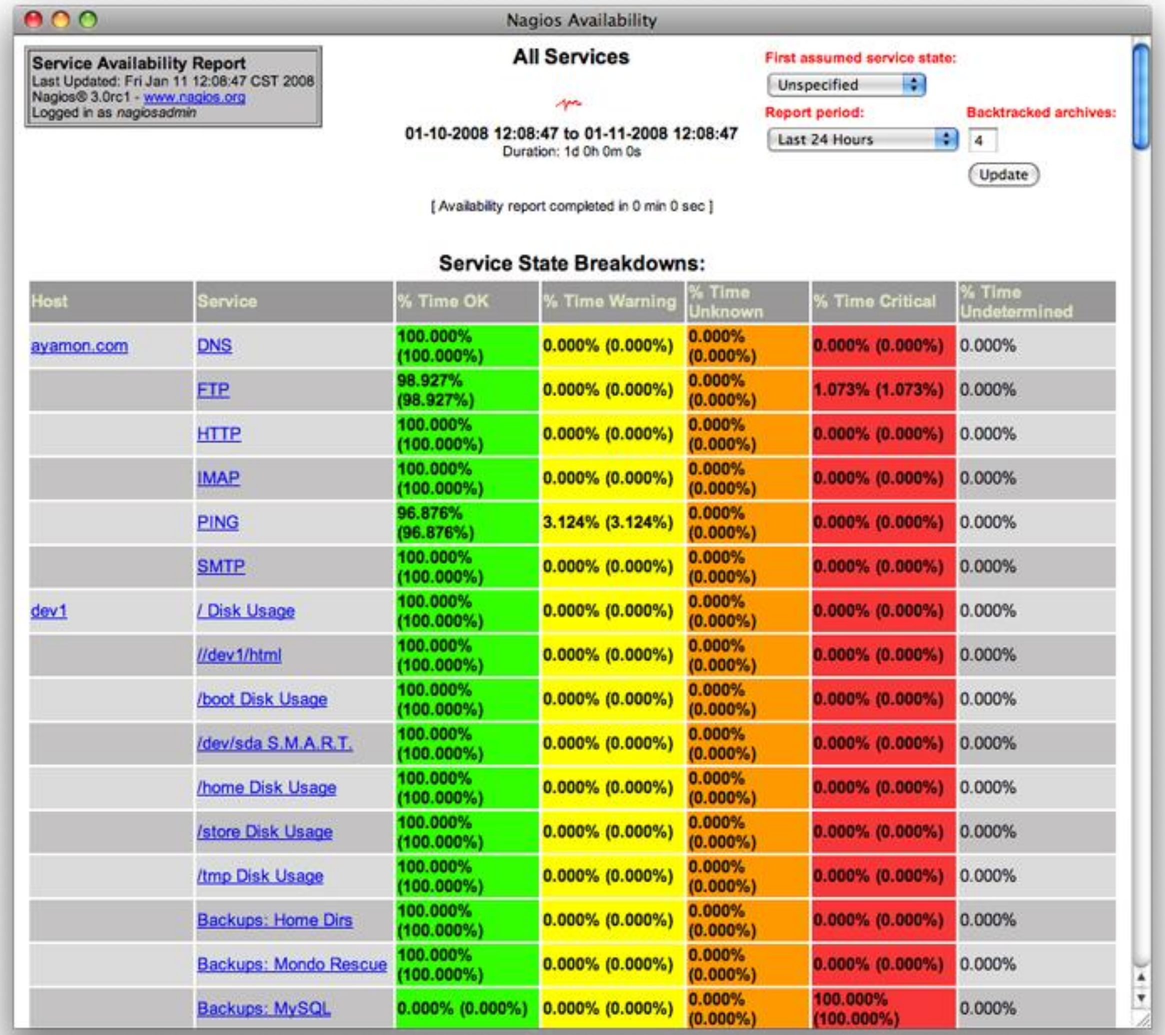
Password

Detail 1

Detail 2

Add User Screen - Admins see this screen when they want to add a new user. Details may be compulsory or optional.

Media Player Screen - Admins and users see this screen when they want to play a stream.



Nagios Availability

Service Availability Report
Last Updated: Fri Jan 11 12:08:47 CST 2008
Nagios® 3.0rc1 - www.nagios.org
Logged in as nagiosadmin

All Services

01-10-2008 12:08:47 to 01-11-2008 12:08:47
Duration: 1d 0h 0m 0s

First assumed service state: Unspecified
Report period: Last 24 Hours
Backtracked archives: 4

[Availability report completed in 0 min 0 sec]

Service State Breakdowns:

Host	Service	% Time OK	% Time Warning	% Time Unknown	% Time Critical	% Time Undetermined
ayamon.com	DNS	100.000% (100.000%)	0.000% (0.000%)	0.000% (0.000%)	0.000% (0.000%)	0.000%
	FTP	98.927% (98.927%)	0.000% (0.000%)	0.000% (0.000%)	1.073% (1.073%)	0.000%
	HTTP	100.000% (100.000%)	0.000% (0.000%)	0.000% (0.000%)	0.000% (0.000%)	0.000%
	IMAP	100.000% (100.000%)	0.000% (0.000%)	0.000% (0.000%)	0.000% (0.000%)	0.000%
	PING	96.876% (96.876%)	3.124% (3.124%)	0.000% (0.000%)	0.000% (0.000%)	0.000%
	SMTP	100.000% (100.000%)	0.000% (0.000%)	0.000% (0.000%)	0.000% (0.000%)	0.000%
dev1	/ Disk Usage	100.000% (100.000%)	0.000% (0.000%)	0.000% (0.000%)	0.000% (0.000%)	0.000%
	//dev1/html	100.000% (100.000%)	0.000% (0.000%)	0.000% (0.000%)	0.000% (0.000%)	0.000%
	/boot Disk Usage	100.000% (100.000%)	0.000% (0.000%)	0.000% (0.000%)	0.000% (0.000%)	0.000%
	/dev/sda S.M.A.R.T.	100.000% (100.000%)	0.000% (0.000%)	0.000% (0.000%)	0.000% (0.000%)	0.000%
	/home Disk Usage	100.000% (100.000%)	0.000% (0.000%)	0.000% (0.000%)	0.000% (0.000%)	0.000%
	/store Disk Usage	100.000% (100.000%)	0.000% (0.000%)	0.000% (0.000%)	0.000% (0.000%)	0.000%
	/tmp Disk Usage	100.000% (100.000%)	0.000% (0.000%)	0.000% (0.000%)	0.000% (0.000%)	0.000%
	Backups: Home Dirs	100.000% (100.000%)	0.000% (0.000%)	0.000% (0.000%)	0.000% (0.000%)	0.000%
	Backups: Mondo Rescue	100.000% (100.000%)	0.000% (0.000%)	0.000% (0.000%)	0.000% (0.000%)	0.000%
	Backups: MySQL	0.000% (0.000%)	0.000% (0.000%)	0.000% (0.000%)	100.000% (100.000%)	0.000%

***Screenshot of Nagios Open Source Monitoring* -** Nagios is one of the most popular monitoring software in the industry, thus, we may use its user interface as a reference point as we develop our project.

6.3. Screen Objects and Actions

Closeness of actions and objects is the primary element in our GUI design. We designed our interface such that all screen objects are close to each other so that users will be able to view and use them easily, in addition, this will provide a basis for future implementations on mobile devices which have relatively smaller displays. Furthermore, this way admins and users can understand what they can and cannot do in the system just by a quick look on the interface.

We have some pop-up windows (confirmation queries etc.) and embedded controls (such as drop down lists or radio buttons) for our system actions. This property also provides more usability and enables us to adapt our current interfaces to newly risen needs, modifications or improvement on the user interfaces or other actions more easily.

7. Detailed Design

7.1. System Administration Tool

This component is a module used to manage all other components of the system. It reaches the management components and controls them. There are management components in other components as we explained before. These components do not work on themselves. They work according to this administration tool's commands. Namely, it is a bridge between other components of the system. This component assumes that management components works fine and obey commands. Error recovery is not in our score at the moment. This module is used by the system administrator directly via the interface that we have described. It simply adds/removes cameras and users to the system and manages their settings. Moreover it specifies how the management components will work. To command an operation to the management components, admin simply enters information and parameters to the relevant interface and they are passed to the relevant component with operation info.

While managing other components, this module keeps info about the cameras and users. Periodically, cameras and users database is checked by this module and they are stored in separate lists. In further operations, data stored in these lists is used for faster access and unique ID of cameras and users are passed to the management components to make operations on a specific item. These data are kept in lists of Server and User classes. Information about these classes is explained below.

This module is in fact simply an interface for the Puppet that is installed on the system. Puppet is a configuration management tool for Unix-like systems as explained in Chapter-8. The interface will control the Puppet via its command line or editing its configuration. More detailed info is available in “Tools and Libraries” chapter.

Since this module is an interface, more design issues will come up as we implement. We will add those specifications to the following documents.

7.1.1. User Class

This component is a class which is present in our interface. This class is to keep information of users that has been added to the media server or edited via the main interface. This class is created because of performance issues. Without this class, when the administrator wants to do some operation on some users, system needs to get the user info from the media server and display it. In addition retrieved info includes lots of technical terms. In a list of User class, we keep information in the interface itself and displayed immediately when requested. Moreover, the technical terms are simplified and presented to the user in terms of more understandable words.

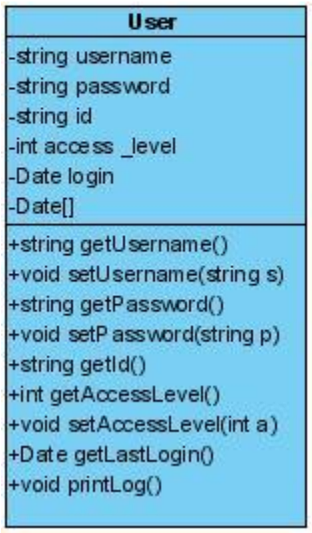
A user or administrator cannot reach this class directly. As admin does some operation via the interface, relevant item in the list is updated, created or deleted internally by the UI. This class will keep simple types and simple information about the users so it will not require much memory or processor so it will not be a burden to the overall system.
Data kept in the User class:

- string username : A user-chosen string that represents name for a user.
- string password : Initially system-assigned string that is required to login to the system. It can be changed later by the user if desired.
- string id : A unique ID for each user assigned by the system. It is not visible to outside it is used for internal operations.
- int access_level : An integer that describes which streams a user can access. It can be edited by the admin with a request from the user.

- Date last_login : A date object that keeps info about what is the most recent time that the user logged in.
- Date[] log : A date array that keeps track of user's activity and their times in the system.

Methods of the User class:

- string getUsername() : Returns the username of the user item in the list.
- void setUsername(string s) : Sets username of a specific item to the string argument s.
- string getPassword() : Returns the password of the user.
- void setPassword(string pw) : Sets password of the user to the string argument pw.
- string getId() : Returns unique id of the user. There is no set method for this item since it is assigned once by the system and its unique.
- int getAccessLevel() : Returns an integer that represents access level of the user.
- void setAccessLevel(int i) : Sets access level of the user to the argument i.
- Date getLastLogin() : Returns the date that used logged in last. last_login field can not be set since it is modified automatically by the system.
- void printLog() : Prints the user's activity log.



7.1.2. Server Class

This component is a class which is present in our interface. This class keeps information about the web servers that are connected to the media server. This class is created for performance issues like User class. Similarly, without this class, whenever the admin wants to do some operation with the web servers, their information needs to be retrieved from the network. In the Server class, we keep server information in the interface and display it immediately when requested. Moreover, the server information is simplified in the class and displayed so.

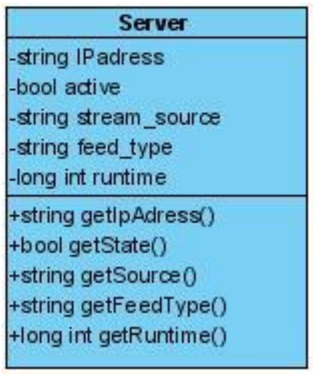
This class is not available to the admin or users. It is used by the system internally when some operation done with the web servers. In this class, we will keep simple data in primitive data types so it will not require much memory or processor. Data kept in the Server class:

- string IPadress : Keeps unique ip address of the server.
- bool active : Keeps info about state of the serve. Set true if the server is active or false if server is waiting idle.
- string stream_source : Keeps information about which stream does the server sends to the clients.
- string feed_type : Keeps information about the format of the video that is being sent by this server.
- long int runtime : Keep time since the activation of the server in terms of seconds.

Methods in the Server class:

- string getIpAddress() : Returns the unique IP address of the server. Address can not be set since it is static across the network.
- bool getState() : Returns state of the server(true for active false for idle). State can not be set since it is changed by the load balancer and this field will be updated then.
- string getSource() : Returns source stream of the server. This field can not be set since source will be changed by the load balancer according to the work load of other servers.

- string getFeedType() : Returns format of the video that is being sent by the server. This field cannot be set since requested format depends on the client.
- long int getRuntime() : Returns the time that server has been active in terms of seconds. This field cannot be set since it is update by the system automatically.



7.1.3. Stream Class

This component is a class in our interface. It keeps information about the video sources connected to the media server. This class is created for same reason with other two classes. When a new source connects to the system, its info is stored in the class in our interface. So, when the admin wants to do some operation its information does not need to be retrieved from the media server every time.

Again this class is not available for users and admin. It is used by the system internally when some operation is done with the sources. Data of this class is similar to the other ones so it will not require much memory or processor.

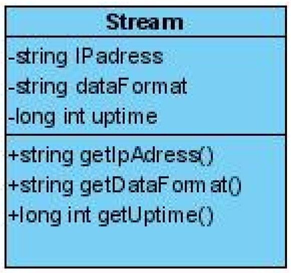
Data kept in the Stream class:

- string IPaddress : Keeps unique ip address of the server.
- string dataFormat : Keeps data format of the video data that is supplied by the source.
- long int uptime : Keep time since the activation of the source in terms of seconds.

Methods in the Stream class:

- string getIpAddress() : Returns the unique IP address of the source on the network. This field cannot be set since IP address is static across the network.

- string getDataFormat() : Returns the format of the video data that the source supplies. This field cannot be set since data format depends on the device only.
- long in getUptime() : Returns the time that source has been active in terms of seconds. This field cannot be set since it is update by the system automatically.



7.2. Management Tools

7.2.1. Media Server Management Tool

This component is a module that works as a bridge between sub-components of the media server and main interface. It controls the CLI and configuration file of the media server. When the admin does some operations from the main interface about the media server, the commands are sent to the management tool. And it sends it to the either CLI or configuration file. This tool assumes that main interface sends valid commands with valid parameters.

This component is used only by main interface; it serves to the main interface only and not available to any other components. It does some simple file operations on the configuration file and passes commands and parameters to the command line interface of the server.

7.2.2. Web Server Management Tool

This component is a module that works as a bridge between sub-components of the web servers and main interface. It is present in every web server. It controls the CLI and configuration file of web servers. When the admin does some operations from the main interface about a web server, the commands are sent to the management tool of that web

server. And it sends it to the either CLI or configuration file. This tool assumes that main interface sends valid commands with valid parameters.
This component is used only by main interface; it serves to the main interface only and not available to any other components. It does some simple file operations on the configuration file and passes commands and parameters to the command line interface of the server.

7.2.3. Load Balancer Management Tool

This component is a module that works as a bridge between sub-components of the load balancer and main interface. It controls the CLI and configuration file of the load balancer. When the admin does some operations from the main interface about the load balancer, the commands are sent to the management tool of the load balancer. And it sends it to the either CLI or configuration file. This tool assumes that main interface sends valid commands with valid parameters.

This component is used only by main interface; it serves to the main interface only and not available to any other components. It does some simple file operations on the configuration file and passes commands and parameters to the command line interface of the server.

7.3. Methods in the Management Tools

There are 2 main common methods in all three of the management tools. Contents of their arguments may change according to the management tool but their roles are almost the same.

Here are the operations:

- void executeCommand(string s) : This method takes a single string argument. The string is modified by the management according to the command from the main interface. Then it is sent to the CLI and executed there. An example command could be "*% vlc -vvv --codec ffmpeg my_file.mpg*" which is used to open a file with a codec or "*% vlc -vvv http://www.example.org/your_file.mpg*" which is used to receive a http stream. These example commands are for VLC media player/server.

- void editConfigurationFile(string op,string data) : This method takes two string arguments. First one defines operation that will be done, either delete data from or write data to the file. Second argument is data to be edited. If operation delete, second argument will be searched in the file and deleted. If the operation is add, second argument is added to the appropriate place in the file.

8. Libraries and Tools

8.1. Media Server Tools

As we researched for media servers, we have encountered many of them that can serve to our purposes. Capabilities of most of them are similar but we are examining three of the popular ones.

8.1.1. VideoLAN Server

The VideoLAN Server (VLS) is a dedicated streaming server from the VideoLAN project. The VideoLAN Server can stream each of the following data types in uni-cast or multi-cast mode:

- MPEG-1
- MPEG-2
- MPEG-4
- DVDs
- digital satellite channels
- digital DVB-TV
- Live-Videos

Much of the functionality is now integrated into the VLC media player (formerly VideoLAN Client). As a result, the development of the VideoLAN Server since 2003 is almost at a standstill. However, its capabilities are now embedded in VLC media player. Therefore VLC media player has all of the capabilities mentioned above and a few more that is explained below.

About the VLC media player, it is a open source media player and multimedia framework. It can also serve as encoder and streamer for many audio and video codecs. It is capable of streaming over networks, transcode multimedia files and save them into various formats.

VLC is a cross-platform media player. It has versions for Microsoft Windows, Mac OS X, GNU, Linux, BeOS, BSD, eComStation and iOS.

Default distribution of VLC comes with many free encoding and decoding libraries. VLC also has a remote control interface for console users. Since it can be used as a streaming server, it can be useful to control it remotely. The remote control interface is a text-based one. There are also interfaces using telnet and HTTP (ajax).

8.2. Web Server Tools

We have some popular choices for web servers but for the time being, we are focusing on two of them.

8.2.1. Apache

Apache web server that has a very important role in the growth of World Wide Web. It was first serious alternative to the Netscape Communications Corporation web server. As it continued to evolve, it became a rival to the other UNIX-based web servers in terms of functionality and performance.

Apache is developed and maintained by developers who are members of Apache Software Foundation. Apache web server is available for various operating systems such as Unix, GNU, FreeBSD, Linux, Solaris, Novell NetWare, Mac OS X, Microsoft Windows, OS/2, TPF, and eComStation.

Apache is distributed under Apache License and it is characterized as a open source software.

Here are some of Apache web server's features:

- Supported by many common languages
- Popular authentication modules, such as mod_access, mod_auth, mod_digest, and mod_auth_digest
- SSL and TLS support
- Proxy module
- URL rewriter
- Custom log files
- Filtering support
- Popular compression methods, including mod_gzip
- ModSecurity: an open source intrusion detection and prevention engine
- Virtual Hosting
- Configurable error messages
- DBMS-based authentication databases
- Content negotiation
- Supported by many GUIs

8.3. Load Balancer Tools

As load balancer component of the system, we are planning to employ an open source firewall distribution that has necessary load balancer feature. Choice for this tool has not been yet made, however we are focusing on few alternatives.

8.3.1.pfSense

pfSense project started in 2004 as a fork of the m0n0wall project, but focused towards full PC installations rather than the embedded hardware focus of m0n0wall. pfSense also offers an embedded image for Compact Flash based installations.

pfSense is a free, open source customized distribution of FreeBSD tailored for use as a firewall and router. In addition to being a powerful, flexible firewalling and routing platform, it includes a long list of related features and a package system allowing further expandability without adding bloat and potential security vulnerabilities to the base distribution.

Some features of pfSense:

- Firewall
- State Table
- NAT
- Redundancy
    - CARP - CARP from OpenBSD allows for hardware failover. Two or more firewalls can be configured as a failover group. If one interface fails on the primary or the primary goes offline entirely, the secondary becomes active. pfSense also includes configuration synchronization capabilities where changes made to the primary firewall will automatically synchronize to the secondary firewall.
    - pfsync - pfsync ensures the firewall's state table is replicated to all failover configured firewalls. This means your existing connections will be maintained in the case of failure, which is important to prevent network disruptions.
- Outbound and Inbound Load Balancing
- VPN - IPsec, OpenVPN, PPTP
- PPPoE Server
- RRD Graphs Reporting
- Real Time Information - Using AJAX
- Dynamic DNS
- Captive portal
- DHCP Server and Relay
- Live CD Version Available
- Proxy server
- Support for software extensions.
    - Notable expansions are : Squid proxy server and Snort intrusion prevention/detection system.

8.4. Puppet

Puppet is a configuration management tool for Linux. It manages the configuration of Linux-like systems as declared by the user. The developer provides Puppet templates for the states of the system and in the runtime, system is put into the declared state.

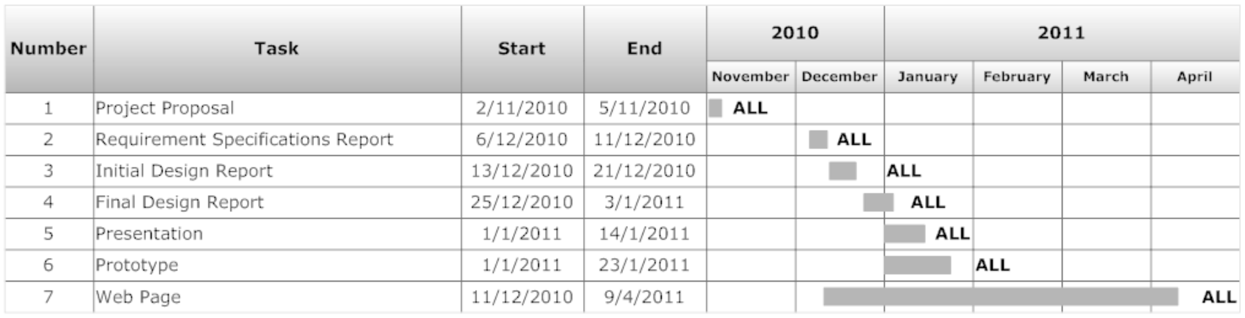
Puppet consists of a custom language to describe system configuration and a library to apply this configuration. It enables the administrators to describe configuration in very high-level terms.

Puppet is built to be cross-platform. It works mostly on Linux and Unix base systems, but it also has basic support for Microsoft Windows. It is an open source tool released under GPL and written in ruby.

We will use puppet in almost all of the components. It will enable us to control configuration files (explained in section 4) easier and more practical instead of editing them by hand.

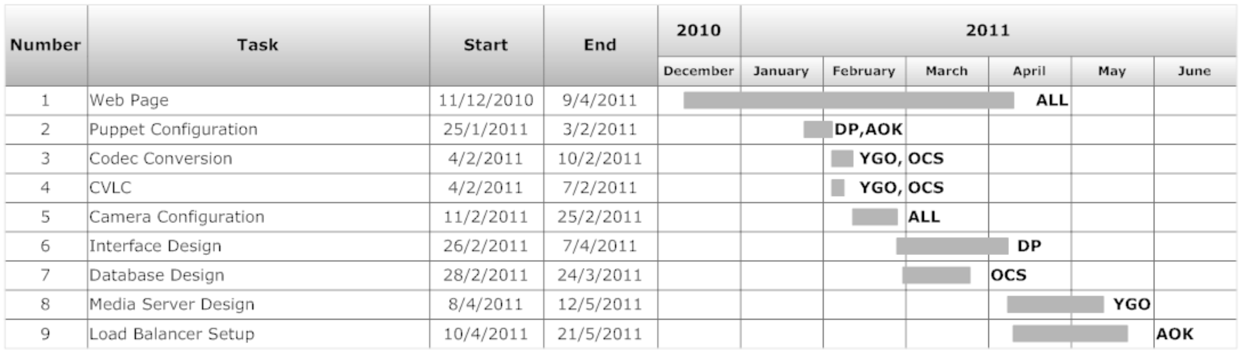
9. Time Planning (Gannt Chart)

9.1. Term 1 Gannt Chart



Number	Task	Start	End	2010		2011			
				November	December	January	February	March	April
1	Project Proposal	2/11/2010	5/11/2010	▨ ALL					
2	Requirement Specifications Report	6/12/2010	11/12/2010		▨ ALL				
3	Initial Design Report	13/12/2010	21/12/2010		▨	ALL			
4	Final Design Report	25/12/2010	3/1/2011		▨	ALL			
5	Presentation	1/1/2011	14/1/2011			▨ ALL			
6	Prototype	1/1/2011	23/1/2011			▨	ALL		
7	Web Page	11/12/2010	9/4/2011		▨	▨	▨	▨	▨ ALL

9.2. Term 2 Gannt Chart



Number	Task	Start	End	
1	Web Page	11/12/2010	9/4/2011	ALL
2	Puppet Configuration	25/1/2011	3/2/2011	DP,AOK
3	Codec Conversion	4/2/2011	10/2/2011	YGO, OCS
4	CVLC	4/2/2011	7/2/2011	YGO, OCS
5	Camera Configuration	11/2/2011	25/2/2011	ALL
6	Interface Design	26/2/2011	7/4/2011	DP
7	Database Design	28/2/2011	24/3/2011	OCS
8	Media Server Design	8/4/2011	12/5/2011	YGO
9	Load Balancer Setup	10/4/2011	21/5/2011	AOK

10. Conclusion

This document is a detailed initial guide for our process in the future. Structure and architecture of the system is explained. Detailed explanations of the modules are given with component and sequence diagrams.

We have mentioned about tools that we are considering to use, yet our research continues. So before we start our implementation, we will make our final decisions about which tool suits best for our purposes and explain these with our reasons.

Next step for us will be writing the Final Design Report which will cover design of the structure explained in this document in even more detail. We will state our reasons for decisions that we have not made yet.