Administering an Open-Source Wireless Network

This tutorial presents enhancements to an open-source wireless network discussed in the June 2007 issue of ITAL *that should reduce its administrative burden. In addition, it will demonstrate an open-source monitoring script written for the wireless network.*

As it has become increasingly important to provide wireless Internet access for their patrons, libraries and colleges are almost expected to offer this service. Inexpensive methods of providing wireless access—such as adding a commodity wireless access point to an existing network—can suffer from security issues, access by external entities, and bandwidth abuses. Designs that address these issues often involve more costly proprietary hardware as well as expertise and effort that are often not readily available. A wireless network built with open-source software and commodity hardware that addressed the cost, security, and equal access issues mentioned above was presented in the June 2007 issue of *ITAL*.[1] This tutorial highlights enhancements to the previous design that help to explain the technical hurdles in implementation, and includes a program that monitors the status of the various software and hardware components, helping to reduce the time required to administer the network.

The wireless network presented requires several different pieces of software that must work together. Because each of the required software programs are frequently updated, slight changes to the implementation may also be needed. A few issues that have arisen since the previous paper was written are addressed. A note is provided explaining the significance of setting the correct Media Access Control (MAC) address for the radius server and for Wireless Distribution System (WDS) when configuring the system. In addition, in order to provide secure exchange of authentication credentials (username and password), the Secure Socket Layer was used. A brief explanation of how to install a registered certificate on the gateway server is provided. Lastly, a program that monitors the status of the network, provides a Web page displaying the status of the various hardware and software components, and e-mails administrators with any changes to the network status—along with information on how this program is to be deployed within the network—is presented.

Configuration changes for previous design

As new exploits are discovered and patched on a continual basis, any system should be regularly updated to insure that the most recent software is being used. The network design provided in the previous article used many different software components including, but not limited to:

Access Point Software
    OpenWRT—Whiterussian rc3

DNS Cache
    Dnsmasq v2.32

Gateway
    Chillispot v1.0

Operating System
    Fedora Core 4

RADIUS Server
    Free Radius v1.0.4

Web Caching Server
    Squid v2.5

Web Server
    Apache 2.2.3

Many of these components can be kept up-to-date by using the Yellow dog Updater, Modified (yum). [2] For example, to update a given package, with root access, at the command line enter:

```
yum update packageName
```

The yum command may also be used to update each package that has an available update by simply removing the package name from the yum update command and entering the following:

```
yum update
```

Yum may also be used to upgrade the entire operating system.[3]

Keep in mind that with any change in software, the configuration of any particular package may change as well. For example, the newest version of Squid is currently 2.6. Appendix D in the previous paper explained how to allow transparent relay of Web requests so that client browsers did not have to be reconfigured. So, while version 2.5 required four changes to allow the transparent relay, the current version—found in appendix A—requires only one. In addition to changes in software, occasionally even entire websites move, as happened with Chillispot.[4]

Another change involved the configuration of the Linksys WRT54GS access points. The newer versions of this access point/router sold by Linksys have half the flash memory and half of the RAM of the older versions.[5] While the newer versions of the Linksys WRT54GS can be flashed with custom firmware,[6] the firmware that will fit on the newer unit lacks all the capability of the standard firmware. Given this, those wishing to implement such a wireless network should investigate the capability of models to be deployed, as well as the version numbers for the access points chosen. The current version of the Linksys WRT54GL and WRTSL54GS units retain enough flash memory and RAM to be updated with the standard firmware mentioned in the previous article.[7]

James Feher (jdfeher@mckendree.edu) is Associate Professor of Computer Science and Computer Information Systems at McKendree University, Lebanon, Illinois. **Tyler Sondag** (sondag@cs.iastate.edu) is a PhD candidate in Computer Science at Iowa State University, Ames.

In addition, the procedure for upgrading the firmware for the WRTSL54GS is simpler than the procedure outlined in appendix I of the previous paper. The factory-installed firmware on version 1.1 can be flashed directly using the Web interface provided by Linksys.

So, while this tutorial and the previous paper outline the design of a network, the administrator will need to be vigilant in updating the packages used and keep in mind that the configuration specifications may also change with those updates. The administrator for the network must also investigate the capability of the standard hardware used to insure that it retains the functionality required for the system.

Choosing the correct MAC address for the access point

The access points used will have more than one interface and as such more than one MAC address. When entering the MAC address of a given access point into either the users file for the Radius server or the access points that use the WDS, use the MAC address associated with the wireless interface.[8] Using the incorrect MAC address will result in problems when communicating with the various access points. For the Radius server, the access point will not get the correct IP address, which will prohibit the possibility of remotely administering the unit. Incorrect MAC addresses that are used for the WDS settings will cause even worse problems, as the unit will not be able to relay data from users who connect to this access point.

Installation of a registered SSL certificate

As users are required to enter their authentication credentials to gain access to the Internet, the exchange of this data is encrypted using the Secure Socket Layer.[9] While administrators can self-sign the certificates used for their Web servers, it is recommended that a registered certificate be obtained and installed for the system. This can help prevent common attacks and has the added benefit of eliminating warnings for the client browsers when they detect unregistered certificates being used by the SSL. A search of "*SSL certificate*" will yield any number of commercial vendors from which a certificate can be obtained. Generally the installation of a certificate is fairly straightforward. The openssl command line utility can be used to generate a SSL key and Certificate Signing Request (CSR).[10] Once the CSR is generated, pick a vendor/Certificate Authority who can sign your key.

It should be noted that the design presented required the authentication gateway to be behind the main router. This required a certificate to be signed for a server within an intranet that does not have a fully qualified domain name. So, when generating the SSL key and CSR, make sure to use GATEWAYHOSTNAME.localnet as the common name of your server. Of course, GATEWAYHOSTNAME is whatever you choose as the name of your gateway host. The term *localnet* is used to refer to the server existing within an intranet. Then make sure to place an entry for GATEWAYHOSTNAME.localnet into the hosts file of the server that is providing Domain Name Service for your network. An example entry for the hosts file which is in the /etc directory of a standard Fedora Core installation is found in appendix B.

Monitoring script for wireless network

As the wireless network has many separate hardware and software components, many possible points of failure exist for the system. The script from appendix C, which was written in Perl,[11] uses Ping to test if each access point is still connected to the network and nmap to test whether the port associated with a given network service is still available.[12] This program can be run manually or, even better, run automatically through the Unix cron utility to update a webpage that displays the current state of all the network components. The webpage generated by this script for the McKendree College wireless network may be found at http://lance.mckendree.edu/cgi-bin/wireless/status.cgi. (Additionally, a sample of this page is available as a figure in appendix D.)

This script actually contains a script within a script. The main script must be run on the gateway machine, Chilli on the diagram in appendix E, as only this machine has access to ping the access points. When the script determines that an access point or daemon is down, it will e-mail the system administrator. When an access point is down, in addition to sending the system administrator an e-mail, it can also send notification to an e-mail address associated with that device. This allows for someone other than the system administrator—who may have closer physical access to the unit—to check the access point on behalf of the administrators for simple issues, such as an access point losing power.

This script then generates another CGI script that can be transmitted to an external server that can be reached from anywhere on the Internet. In this case, this generated script can be run as a Web-based application or by the system itself using the cron utility. If run as by the cron daemon, it will also e-mail the administrators if the script has not been updated recently. The script requires the use of several Perl modules that will need to be installed.

- Expect
- Mail::Mailer
- Net::Ping

The script has been released using the GNU General Public License, Version 2 (GPL).[13] The first portion of the script contains a reference to the GPL, followed by a brief explanation of the script as well as a set of parameters that should be changed to fit the specifications of the network designed.

Conclusion

Administrators should be vigilant in updating the entire system to assure security, keeping in mind that new versions of software or hardware may necessitate changes in the overall configuration of the system. In addition, while the monitoring script provides a useful aid in monitoring the network, it could be further expanded to include a more comprehensive review of level of use for various access points by the different users. It is felt that this would be best done through a database, which would require a higher level of administrative effort. A brief frequently asked questions list along with the script and link to the code for the script can be found at http://lance.mckendree.edu/csi/wirelessfaq.html.

References

1. Sondag, Tyler and James Feher, "Open Source Wifi Hotspot Implementation," *Information Technology and Libraries* 26, no. 2: 35–43, http://ala.org/ala/lita/litapublications/ital/262007/2602jun/toc.cfm (accessed July 24, 2008).

2. Linux@DUKE, "Yum: Yellow dog Updater, Modified," http://linux.duke.edu/projects/yum (accessed July 24, 2008)

3. Upgrading Fedora Using Yum Frequently Asked Questions, http://fedoraproject.org/wiki/YumUpgradeFaq (accessed Mar. 16, 2007).

4. ChilliSpot—Open Source Wireless LAN Access Point Controller, "Spice up your HotSpot with Chilli," www.chillispot.info/ (accessed May 22, 2008).

5. OpenWrtDocs/Hardware/Linksys/WRT54GS—OpenWrt, http://wiki.openwrt.org/OpenWrtDocs/Hardware/Linksys/WRT54GS (accessed July 24, 2008).

6. Bitsum Technologies Wiki—WRT54G5 CFE, http://bitsum.com/openwiking/owbase/ow.asp?WRT54G5_CFE (accessed July 24, 2008).

7. OpenWrtDocs/Hardware/Linksys/WRTSL54GS—OpenWrt, http://wiki.openwrt.org/OpenWrtDocs/Hardware/Linksys/WRTSL54GS (accessed July 24, 2008).

8. OpenWrtDocs/WhiteRussian/Configuration, Wireless Distribution System (WDS)/Repeater/Bridge. http://wiki.openwrt.org/OpenWrtDocs/WhiteRussian/Configuration (accessed July 24, 2008).

9. Viega, John, Matt Messier, and Pravir Chandra, *Network Security with OpenSSL Cryptography for Secure Communications.* (Sebastopol, Calif.: O'Reilly and Associates, 2002).

10. Generating a Key Pair and CSR for an Apache Server with modssl. www.verisign.com/support/tlc/csr/modssl/v00.html (accessed Feb. 20, 2007).

11. Wall, Larry, Tom Christiansen, and Randal Schwartz, *Programming Perl, Third Edition* (Sebastopol, Calif.: O'Reilly and Associates).

12. Nmap—Free Security Scanner For Network Exploration and Security Audits. http://insecure.org/nmap/ (accessed Feb. 20, 2007).

13. GNU General Public License Version 2, June 2007. www.gnu.org/licenses/gpl.txt.

APPENDIX A. Squid configuration changes

```
# changes made to squid.conf
# Lines needed for Squid 2.5
#httpd_accel_port 80
#httpd_accel_host virtual
#httpd_accel_with_proxy on
#httpd_accel_uses_host_header on
#
# One line needed in version 2.6
http_port 3128 transparent
```

APPENDIX B. /etc/hosts entry on marla for localnet entry

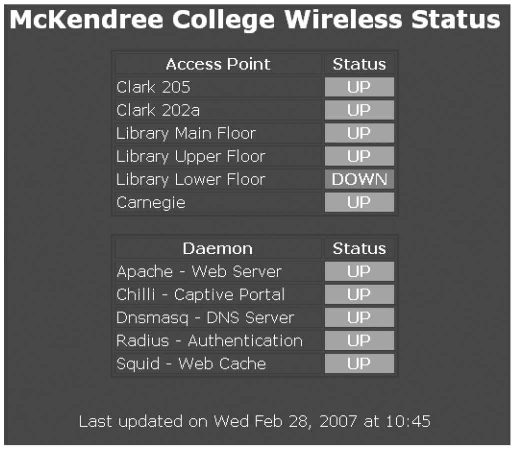
```
127.0.0.1          marla  localhost.localdomain      localhost
66.128.109.60                                        bob
66.99.172.252      lance.mckendree.edu               lance
# next line is for the ssl certificate to work properly
192.168.176.1      chilli.localnet                   chilli
```

APPENDIX C. Monitoring script

```
#!/usr/bin/perl
########################################################
# Code released 03/22/07 under:                        #
# the GNU GENERAL PUBLIC LICENSE, Version 2            #
# http://www.gnu.org/licenses/gpl.txt                  #
#                                                      #
# It is recommended that this script is run as a cron  #
# job frequently to find changes in the network.  This #
# script will check the status of the wireless access  #
# points/routers as well as the daemons necessary to   #
# run the network.  It will then output the results to #
# another perl file that is copied to a remote         #
# webserver.  When the script observes a change in the #
# availability of any access point or daemon, email    #
# will be sent to the specified administrator          #
# address(es).  The option exists to send an email to  #
# to an additional person for each access point.       #
#                                                      #
# Additionally, the output file on the remote webserver#
# will check when it was last updated, if that script  #
# is run from the command line or via cron.  If it has #
# not been updated for a specified number of minutes,  #
# it will send an email to the administrator.  It is   #
# also recommended that this output script be run as a #
# cron jobr.  This output script can also be executed  #
# as a cgi program to generate a display of network    #
# status.                                              #
########################################################

use strict;
use Expect();       # needed to scp to webserver
use Mail::Mailer;   # needed to send emails if outages
use Net::Ping;      # needed to check the status of aps

#variables for webserver to host status page's
my $webServUname    =      "username";
my $webServPass     =      "password";
my $webServUrl      =      "lance.mckendree.edu";
my $webServTarg     =      "/var/www/cgi-bin/wireless/";
my $webOutputUrl    =
 "http://lance.mckendree.edu/cgi-bin/wireless/status.cgi";

my $instName        =      "McKendree College";
#default background color of the status page
my $defBGColor      =      "#660066";

# If the page on the webserver has not been updated
# in $updateMin minutes send an email that the service
# is down (set to =~ 3*crontime)
my $updateMin       = 10;

#email address errors will be sent to
my $fromEmail       =      'admin1@email.com';
my $toEmail         =
         'admin1@email.com, admin2@email.com';

#file where errors will be stored on remote host
my $logFileName     =      "/tmp/wirelesLog.txt";

#hash for routers/ap's
#location is displayed on the webpage and in status emails
#owner - changes in status regarding this AP are sent to
#         this address as well (optional)
my %ipToLoc = (
  "192.168.182.10" => {
            "LOCATION" => "Clark 205",
            "OWNER"    => ''},
  "192.168.182.11" => {
            "LOCATION" => "Clark 202a",
            "OWNER"    => 'apUser1@email.com'},
  "192.168.182.12" => {
            "LOCATION" => "PAC Lounge",
            "OWNER"    => 'apUser2@email.com'},
  "192.168.182.20" => {
            "LOCATION" => "Library Main",
            "OWNER"    => 'apUser3@email.com'},
  "192.168.182.21" => {
            "LOCATION" => "Library Upper",
            "OWNER"    => ''},
  "192.168.182.22" => {
            "LOCATION" => "Library Lower",
            "OWNER"    => ''},
  "192.168.182.30" => {
            "LOCATION" => "Carnegie",
            "OWNER"    => 'apUser4@email.com'});

#hash for daemons
my %daemons = (
  "Dnsmasq - DNS Server"   => {
                      "IP_ADDR" =>"10.4.1.90",
                      "PORT"    =>"53",
                      "PROTO"   =>"TCP"},
  "Radius  - Authenticate" => {
                      "IP_ADDR" =>"10.4.1.90",
                      "PORT"    =>"1812",
                      "PROTO"   =>"UDP"},
  "Chilli  - Capt. Portal" => {
                      "IP_ADDR" =>"10.5.3.30",
                      "PORT"    =>"0",
                      "PROTO"   =>"LOCAL"},
  "Squid   - Web Cache"    => {
                      "IP_ADDR" =>"10.4.1.90",
                      "PORT"    =>"3128",
                      "PROTO"   =>"TCP"},
  "Apache  - Web Server"   => {
                      "IP_ADDR" =>"10.5.3.30",
                      "PORT"    =>"80",
                      "PROTO"   =>"TCP"});
```

```
#######################################################
#                                                     #
#   NO CHANGES NEED TO BE MADE TO THE FOLLOWING CODE  #
#                                                     #
#######################################################

# get the current time
my $currentTime = scalar localtime();
my $startTime = time();

# open old output status script to get previous status'
open(OLD, "status.cgi");
my @tmpOldStatFile = <OLD>;
my $oldStatFile = join("", @tmpOldStatFile);

# check routers/ap's using ping
my $diff = '';
my $allRouterStat;
foreach my $host (sort keys %ipToLoc){
  my $p = Net::Ping->new();
  my $pingResult = $p->ping($host);
  if(!$pingResult){
    sleep 10;
    $pingResult = $p->ping($host);
  }
  my $thisLastStat = ( $oldStatFile =~
    m/$ipToLoc{$host}{LOCATION}<\/TD><TD\ class="up/
  );
  my $location = $ipToLoc{$host}{LOCATION};
  my $owner    = $ipToLoc{$host}{OWNER};
  my ($thisStatLine, $toEmail, $tmpDiff) =
    &printStatus(  $location,      $pingResult,
                   $thisLastStat, $owner,
                   $toEmail,       $oldStatFile,
                   $currentTime
  );
  $diff .= $tmpDiff if ($tmpDiff);
  $allRouterStat .= $thisStatLine;
  $p->close();
}

#check the status of each daemon
my $allDaemonStat ='';
foreach my $i (sort keys %daemons){
  my $thisLastStat = ( $oldStatFile =~
    m/$i<\/TD><TD\ class="up/
  );
  my $currStat =
    &checkDaemon ($daemons{$i}{IP_ADDR},
                  $daemons{$i}{PORT},     $i,
                  $daemons{$i}{PROTO}
  );
  my ($thisStatLine, $toEmail, $tmpDiff) =
      &printStatus($i,             $currStat,
                    $thisLastStat, “”,
                    $toEmail,      $oldStatFile,
                    $currentTime
  );
  $diff .= $tmpDiff if ($tmpDiff);
  $allDaemonStat .= $thisStatLine;
}

#########################################################
#  the following block is the perl code that will       #
#  generate the status page on the external webserver   #
#########################################################
my $perlOutput = <<OUTPUT_FILE_FOR_REMOTE_HOST;
#!/usr/bin/perl

use strict;
use Mail::Mailer;
use CGI qw(:standard);

my \$currentUser    =      \$ENV{‘USER’};
my \$lastTime       =      $startTime;
my \$currentTime    =      time();
my \$message        =      “”;
my \$systemStatus   =      “$defBGColor”;
my \$toEmail        =      ‘$toEmail’;

# check system status (FF0000 = down)
if (\$currentTime > (\$lastTime + (60 * $updateMin))){
  \$systemStatus = “#FF0000”;
  \$message = “<H1>Status Update Failed</H1>”;
}

# if this is cron running the script
if (\$currentUser =~ “$webServUname”){
  # send email if status is down & logFile doesn’t exist
  &sendEmail() if (
     (\$systemStatus =~ “#FF0000”) && !(-e “$logFileName”)
  );
  # delete log file if everything is up
  unlink(“$logFileName”) if (
    (!(\$systemStatus =~ “#FF0000”)) &&
    (-e “$logFileName”)
  );
}

#else apache is accessing the page (its a web request)
else{
  #print the page
  print header();
###########################
#   start of html output  #
###########################
  print <<WEB_OUTPUT;
  <HTML>
  <HEAD>
    <TITLE>$instName Wireless Status</TITLE>
    <STYLE>
      BODY      {text-align:center;
                 color:#FFFFFF;
                 font-family:Verdana,Helvetica,sans-serif}
      H1        {font-weight:bold;
                 font-size:26;
                 text-align:center;}
      TABLE     {width:300px;
                 border:2px solid #222222;
                 margin-left:auto;
                 margin-right:auto}
      TD        {border:1px solid #222222;
                 font-size:16}
      TD.up     {width:70px;
                 background-color:#00DD00;
                 font-weight:bold;
                 text-align:center}
      TD.down   {width:70px;
                 background-color:#FF0000;
                 font-weight:bold;
                 text-align:center}
      TD.header {text-align:center;
                 font-weight:bold}
    </STYLE>
  </HEAD>
  <BODY BGCOLOR="\$systemStatus">
    <H1>$instName Wireless Status</H1>
    \$message
    <TABLE>
      <TR>
        <TD class="header">Access Point</TD>
        <TD class="header">Status</TD>
      </TR>
      $allRouterStat
    </TABLE>
    <BR>
    <TABLE>
      <TR>
        <TD class="header">Daemon</TD>
        <TD class="header">Status</TD>
      </TR>
      $allDaemonStat
    </TABLE>
    <BR><BR>Last updated $currentTime<BR>
    </CENTER>
  </BODY>
  </HTML>
WEB_OUTPUT
##########################
#   end of html output   #
##########################
}#end else

sub sendEmail {
  my \$mailer = Mail::Mailer->new("sendmail");
  \$mailer->open({From      => '$fromEmail',
                   To      => [\$toEmail],
                   Subject => "Wireless Problem"});
  my \$message = "The wireless system has failed to "
                 ."it's status.\n\n$webOutputUrl\n";
  print \$mailer \$message;
  \$mailer->close();
  open(FILE, ">>$logFileName");
  print FILE "Failed to Update system.";
  close(FILE);
}

OUTPUT_FILE_FOR_REMOTE_HOST
#######################################################
#               end of script output block            #
#######################################################


#write output code to the file
my $perlOutputFile = "status.cgi";
open (OUT, ">$perlOutputFile");
print OUT $perlOutput;
close (OUT);
chmod 0755, $perlOutputFile;

#send email is necessary
&sendEmail($diff, $webOutputUrl, $fromEmail, $toEmail)
  if ($diff);

#send perl file to webserver
&scpFile($perlOutputFile, $webServUname, $webServPass,
         $webServUrl,     $webServTarg);


###############################################
#                                             #
#     END MAIN CODE BLOCK, START FUNCTIONS    #
#                                             #
###############################################


# given the name and status of something (ap or
# daemon), this returns a string for the table
# row for displaying the status of the ap/daemon
sub printStatus {
  my ($service,    $status, $oldStatus,
      $owner,      $toEmail,$oldStatusFile,
      $currentTime                          ) = @_;
  my $msg = "";
  my $statusLine =
    "\n      <TR><TD>$service</TD><TD class=\"";
  # if current status is up
  if ($status){
    ### -- NOTE: This comment holds the previous  -- ###
    ###           status & status previous to that -- ###
    $statusLine .=
      "up\">UP<!--($service)-$status-$oldStatus-->";
    # if last two status' were down
    if ($oldStatusFile =~ m/\($service\)-0--->/){
      $msg = "$service back up at $currentTime\n";
      # if service has owner & not already in mail list,
      # add owner to mail list
      $toEmail .= ", \'$owner\'"
        if ($owner && (!($toEmail =~ $owner)));
    }
  }
  #else current status is down
  else{
    $statusLine .=
      "down\">DOWN<!--($service)-$status-$oldStatus-->";
    # if last status was down & before that status was up
    if ($oldStatusFile =~ m/\($service\)-0-1-->/){
      $msg = "$service down at $currentTime\n";
      # if service has owner & not already in mail list,
      # add owner to mail list
      $toEmail .= ", \'$owner\'"
        if ($owner && (!($toEmail =~ $owner)));
    }
  }
  $statusLine .= "</TD></TR>";
  return ($statusLine, $toEmail, $msg);
}#end printStatus function

# checks the status for the given daemon
# takes in IP, port to check, daemon name, and protocol
# (tcp/udp). if given port=0 it checks for local daemon
sub checkDaemon {
  my ($ip, $port, $daemon, $proto) = @_;
  my $dStat = 0;
  if ($proto !~ /LOCAL/){
    #sU checks for udp ports
    my $com = ($proto =~ "TCP")
            ? ("nmap -p $port $ip | grep $port")
            : ("nmap -sU -p $port $ip | grep $port");
    open(TMP, "$com|");
    my $comOut = <TMP>;
    close(TMP);
    if ($comOut =~ /open/){
      $dStat = 1; #if port is open, status is up
    }
  }
  else{
    $daemon =~ s/ +.*//g;
    #\l lowercases the first letter of $daemon
    my $com = "which \l$daemon";
    open(TMP, "$com|");
    my $comOut = <TMP>;
    close(TMP);
    $com = "ps aux | awk '{print \$11}' | grep $comOut";
    open(TMP, "$com|");
    $comOut = <TMP>;
    close(TMP);
    $dStat = 1 if ($comOut);
  }
  return $dStat;
} # end checkDaemon function
# send the output perl status file to the webserver
sub scpFile {
  my ($filePath,   $webServUname, $webServPass,
      $webServUrl, $webServTarg                 ) = @_;
  my $command = "scp $filePath $webServUname"
              ."\@$webServUrl:$webServTarg";
  my $exp1 = Expect->spawn ($command);
  # the first argument "30" may need to be adjusted
  # if your system has very high latency
  my $ret = $exp1->expect(30, "word:");
  print $exp1 "$webServPass\r";
  my $ret = $exp1->expect(undef);
  $exp1->close();
} # end scpFile function

# send an email to the admin & append error to log file
sub sendEmail {
  my ($errorList,  $webOutputUrl, $fromEmail,
      $toAddresses                             ) = @_;
  my $mailer = Mail::Mailer->new("sendmail");
  $mailer->open({From    => "$fromEmail",
                 To      => [$toAddresses],
                 Subject => "Wireless Problem"});
  $errorList .= "\n\n$webOutputUrl";
  print $mailer $errorList;
  $mailer->close();
} # end sendEmail function
```

APPENDIX D. Script output page



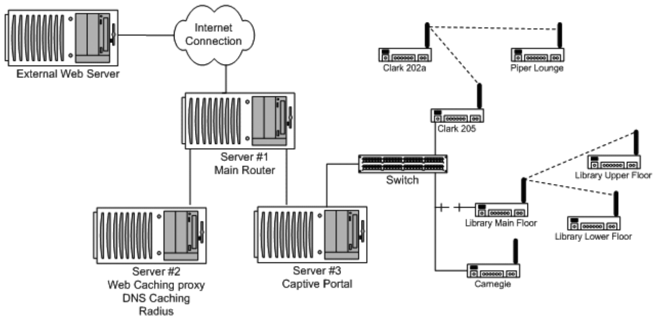
McKendree College Wireless Status

Access Point	Status
Clark 205	UP
Clark 202a	UP
Library Main Floor	UP
Library Upper Floor	UP
Library Lower Floor	DOWN
Carnegie	UP

Daemon	Status
Apache - Web Server	UP
Chilli - Captive Portal	UP
Dnsmasq - DNS Server	UP
Radius - Authentication	UP
Squid - Web Cache	UP

Last updated on Wed Feb 28, 2007 at 10:45

APPENDIX E. Diagram of network



Index to Advertisers

LITA cover 2, cover 3, cover 4