Cam-View Documentation

2020-11-07

Contents

1 Introduction

Cam-View is an open source system for capturing video from one or more network-attached video cameras and for displaying, storing locally and archiving remotely captured images.

Cam-View can run on inexpensive PC hardware and draw images from inexpensive cameras. It was developed on Ubuntu but should be easily portable to any Linux distribution and with minor effort any Unix-based system.

Cam-View is suitable for capturing, storing and viewing security video footage in a small setting such as a house or apartment up to a medium to large setting such as a multi-floor office building. A modestly equipped PC (sub $1000, no GPU) should be able to capture and process image data from at least 20 or 30 cameras concurrently.

Key features of Cam-View include:

1. Inspecting each captured image to determine whether it should be stored or discarded.
2. Only storing images if either motion is detected or a threshold time interval has elapsed since the last stored image.
3. Displaying stored images via a web UI – all user interaction, once the system is installed, is via a web browser.
4. Migrating image files across three storage tiers:
    - A local, high performance, non-persistent ramdisk. This is where images are initially captured.
    - A local persistent disk (solid state SSD or spinning HDD). This is where interactive viewing or video downloads come from.
    - A remote archive. This ensures image retention even if the local storage server is lost or damaged.
5. Dealing gracefully with inexpensive, unreliable and poorly connected cameras, for example by periodically attempting to reconnect.
6. Support for multi-tenancy, in both the physical building sense and logical access sense of the word. One process can capture video data from multiple cameras in a building. Different URLs are used to present video to different tenants – each from a distinct subset of cameras.
7. Interactive and off-line viewing. Interactive via a web UI, which allows a viewer to navigate through captured images (i.e., those where motion was detected or time had elapsed) or to download a video file or image archive consisting of the same files.

2 Using Cam-View

Once Cam-View has been installed and configured, it works as follows:

1. A process – `cam-monitor` runs in the background.
2. `cam-monitor` launches one command per camera to generate image files. The usual command is `ffmpeg`.
3. The image generator produces image files on a RAMDISK. This is normally mounted in `/mnt/ramdisk`.
4. The `cam-monitor` process inspects images as they are generated and when it either detects motion or a timeout has passed, it stores images in a backup location, such as `/data/security`.
5. Periodically, `cam-monitor` may also run a command to copy recent images from the backup location to a remote location, for example to protect against damage to or loss of the video surveillance system.
6. A web server is configured on the video surveillance system. This is normally Apache2.
7. Users can view recently backed up images using a web browser. The usual URL for this is http://localhost/cam-bin/cam-view.
8. Users can view the status of the system at http://localhost/cam-bin/cam-status.
9. With either `cam-view` or `cam-status`, users can click on a single image to zoom in, and click on arrow icons on either side of a zoomed-in image to step forward or backwards through stored images. Clicking anywhere else in the zoomed in image closes the zoom-in popup view.
10. Users can download recorded images or video at http://localhost/cam-bin/download-recordings.
11. A number of shell scripts are scheduled to run periodically:
    - `/usr/local/bin/cam-view-clean-download.sh` cleans old files from the download folder.
    - `/usr/local/bin/cam-view-health-check.sh` checks that `cam-monitor` is running and, if not, starts it.
    - `/usr/local/bin/cam-view-archive.sh` cleans up old files in the backup folder. First, `.jpg` image files older than two days are stored in a `.tar` archive. Second, files older than 21 days are deleted. Finally, any log files older than 10 days are removed.
12. Command-line utilities are also included. These provide usage instructions via the `-h` argument:
    - `cam-archive` – archive old `.jpg` images into `.tar` archives.
    - `cam-config` – test parsing of configuration files.
13. Static web documents – including image files and JavaScript libraries – are normally placed in `/var/www/html/cam-v` and accessed at http://localhost/cam-view/.
14. CGI programs and their configuration files are normally placed in `/var/www/cam-view-bin` and accessed at http://localhost/cam-bin/.
15. HTML documents for the UI are constructed by the CGI programs from template files, also installed in `/var/www/cam-view-bin`.
16. The web CGI programs, the `cam-monitor` process, the image capture commands and the scheduled jobs all run in the security context of an (unprivileged) user, normally `camview`.

3 Configuration

Cam-View is configured using a text configuration file. Conventionally, this is called `config.ini`. In the CGI directory, this is the mandatory filename.

The configuration file specifies both global parameters and a list of cameras, each with its own parameters. The file format supports comments, file inclusion and simple macro expansion.

3.1 Syntax

- Items in the configuration file are written as single lines of text, in the form *variable*=*value*. Values can include spaces and punctuation marks, and are ended at the end of each line.
- If the combination of a variable and value is very long, a configuration line can be split over two or more lines by ending all but the last line of text with a `\` character.
- Comments are lines that begin with a # character.
- If a *variable* is not one that the system understands, it is stored and can be referred to later in the file by the expression *$variable*.
- Secondary configuration files are incorporated into the main one via the `#include "`*filename*`"` directive.

The following block of text illustrates this syntax:

```
CONFIG_NAME=My Video Surveillance System
EPHEMERAL=/mnt/ramdisk/security
PERSISTENT=/data/security
BASE_DIR=$EPHEMERAL
BACKUP_DIR=$PERSISTENT
DOWNLOAD_DIR=$PERSISTENT/download
LOG_FILE=$EPHEMERAL/monitor.log
CGI_LOG_FILE=$EPHEMERAL/cgi.log
DEFAULT_MINIMUM_CAPTURE_INTERVAL=600
BACKUP_COMMAND=/bin/tar cf - %FILES%\
  | /usr/bin/ssh backup@my-backup-service.org "cd /data/security; tar xpf -"
DEFAULT_MINIMUM_CAPTURE_INTERVAL=600
DEFAULT_MOTION_FRAMES=3

CAPFILE=test-image-%08d.jpg
# Use the same command for every camera of this type.  Just
# substitute the IP address or hostname, user and password
# in the RTSP URL:
BESDER_CAPTURE=/usr/bin/ffmpeg\
  -y\
  -i rtsp://$IPADDR:554/user=$USER_password=$PASS_channel=1_stream=1.sdp\
  -r 1\
```

```
  -f image2\
  -vframes 99999999\
  $CAPFILE

# Here is the first camera:
CAMERA=front-entry
USER=admin
PASS=admPASS
IPADDR=192.168.1.20
COMMAND=$BESDER_CAPTURE

# Second camera:
# uses the same creds as the front camera so no need to update those.
CAMERA=back-entry
IPADDR=192.168.1.21
COMMAND=$BESDER_CAPTURE
```

3.2 Global parameters

Each configuration file must specify the following parameters:

- `CONFIG_NAME` – The name of the system. This is displayed in the web UI.
  Example usage: `CONFIG_NAME=My Video Surveillance System`
- `BASE_DIR` – The directory where real time image capture will take place. It is recommended that a RAM disk be created and mounted in `/mnt/ramdisk` and the base directory be defined as `/mnt/ramdisk/security`.
  Example usage: `BASE_DIR=/mnt/ramdisk/security`
- `BACKUP_DIR` – The directory to which images are copied if motion is detected or time has elapsed. The `/data/security` directory is suggested.
  Example usage: `BACKUP_DIR=/data/security`
- `DOWNLOAD_DIR` – A directory where archives or movie files are temporarily placed, before being downloaded to an authorized browser. Since these can be quite large and a RAM disk is usually small, do not use the value set for `BASE_DIR`. A sub-folder of `BACKUP_DIR` is recommended.
  Example usage: `DOWNLOAD_DIR=/data/security/downloads`
- `LOG_FILE` – The full path to a filename where debug and event log information will be written. Since this can be quite a busy file whose contents need not be retained for a long time, a folder in `BASE_DIR`, on a RAM disk, is a good idea.
  Example usage: `LOG_FILE=/mnt/ramdisk/security/monitor.log`
- `CGI_LOG_FILE` – Like `LOG_FILE` but used by the web UI of the system, rather than the video capture process. Keeping this in the RAM disk also makes sense.
  Example usage: `CGI_LOG_FILE=/mnt/ramdisk/security/cgi.log`

The following parameters are not mandatory but can also be specified:

- DEFAULT_MOTION_FRAMES – If a camera does not specify a different value, copy this many image files (frames) from the BASE_DIR folder to the BACKUP_DIR folder after detecting motion.
  Example usage: DEFAULT_MOTION_FRAMES=3
- DEFAULT_MINIMUM_CAPTURE_INTERVAL – Even if no motion is detected, an image will be copied from the BASE_DIR location to the BACKUP_DIR location periodically. The period is specified in seconds using this directive.
  Example usage: DEFAULT_MINIMUM_CAPTURE_INTERVAL=600
- BACKUP_COMMAND – If there is any concern about the video monitoring server itself being damaged or stolen, images can be copied from the BACKUP_DIR directory to another location on the network, such as a VM in the cloud. This is the command used to copy a set of files from the video server to the off-site backup location.

  The %FILES% macro is expanded into a list of filenames of the form *cameraID/imagefile-with-timestamp.jpg*.

  The backup command is executed with the current working directory being BACKUP_DIR.

  Example usage:
  ```
  BACKUP_COMMAND=/bin/tar cf - %FILES% \
    | /usr/bin/ssh backupUser@backupSystem "cd /data/security; tar xpf -"
  ```
  Note that in the above example, an SSH trust relationship should exist, to eliminate password prompts.
- FILES_TO_CACHE – It would be inefficient to run BACKUP_COMMAND once per image. This parameter tells Cam-View how many files (approximately) it should collect before copying images to the remote server.
  Example usage: FILES_TO_CACHE=50

3.3 Defining cameras

Each camera is introduced in the configuration file with at least the following two commands:

- CAMERA – The name of the camera. It is recommended that camera names consist only of letters, digits, periods and the dash (-) character. All subsequent parameters relate to this camera.
  Example usage: CAMERA=My-first-camera
- COMMAND – The command that Cam-View should run to fetch image files from this camera. This is normally the ffmpeg command acquiring a video stream from an RTSP service on the camera. For example, this works for many "BESDER" branded cameras:
  ```
  COMMAND=/usr/bin/ffmpeg -y\
    -i rtsp://ip-address:554/user=UU_password=PP_channel=1_stream=1.sdp\
    -r 1\
    -f image2\
    -vframes 99999999\
    test-image-%08d.jpg
  ```

> **Note:** Cam-View requires that image files generated by the capture process begin with `text-image-` and end in `.jpg`.

The parameters that follow a CAMERA line pertain to just that camera. In addition to the mandatory COMMAND line, there are optional parameters that can be included:

- `MINIMUM_CAPTURE_INTERVAL` – same as `DEFAULT_MINIMUM_CAPTURE_INTERVAL` but just for the current camera.
- `MOTION_FRAMES` – same as `DEFAULT_MOTION_FRAMES` but just for the current camera.
- `DEBUG` – set to `true` or `false` – increases logging for the capture and image processing performed for this camera. Can generate a lot of data so do not run with this for a long time – the RAM disk can fill up.

3.4 Debugging configuration files

Because of multi-line inputs, variable expansion and file inclusion, it is easy to make mistakes in the configuration file. A mechanism is provided to help diagnose such problems. Just add the following directive at the end of the main configuration file:

`#print "`*`debug-output-filename`*`"`

When the `cam-monitor` command is run using a configuration file that ends in this directive, the configuration file, with all includes, long lines and macros expanded out, to be printed to the file *filename*. The monitor will then exit rather than trying to connect to cameras. Run the monitor as follows to do this:

`cam-monitor -c `*`main-config-filename`*

You can then inspect the contents of *debug-output-filename*. Just don't forget to remove this line from your configuration file before trying to move your setup to production, as this command terminates the monitor after printing the diagnostic file.

4 Installation

There are a number of elements required to run Cam-View:

1. A user and group in whose security context the software runs.
2. A video acquisition program which connects to each camera, fetches a video stream and writes image files on the filesystem. The installer is able to install ffmpeg, which is commonly used.
3. A RAMDISK where image files are stored, to minimize I/O cost.
4. A backup folder, where images of interest are stored.
5. A web server, used to present all user interfaces. The installer is able to install and configure Apache2.
6. A variety of CGI programs that run under the web server, to (a) display the most recently stored image from each camera, (b) to rewind to previous images, (c) to check on the status of the system and all cameras and (d) to download video files or archives that contain individual image files.
7. A variety of scripts and binary programs for archiving and cleaning up image files and log files.

4.1 Simple installation

There is a fairly extensive installation script, which can configure all of the above items. Like the rest of Cam-View, this script was developed and tested on an Ubuntu Linux system. It may work on other systems, with minor modifications, but has not been tested outside of this platform.

To configure the system, first create a configuration file as described in section 3 on Page 3. The example provided is a good starting point. The installer will, by default, look for a configuration file called `config.`*`hostname`* where *hostname* is the output of the `uname -n` command on your system. To keep things simple, use this filename for now.

You must specify at least one camera in your configuration file. Video capture is typically performed using the `ffmpeg` command, which in turn requires an RTSP URL for your camera.

Some tips for configuring cameras:

1. Most cameras are activated using a smart phone app, so start there, to attach your camera to your wired or WiFi network.
2. If you can reach the camera with an Ethernet cable, do so. This usually yields better image quality and more consistent results than WiFi, which is vulnerable to range issues and interference.
3. Do set a user ID and password on your camera. Consider using the same credentials on all your cameras, for simplicity. Note these credentials, as they will be incorporated into the RTSP URL later.
4. If the camera uses DHCP for its network address, adjust your DHCP server, which might be on the video surveillance server itself (`isc-dhcp-server` works well) or possibly on your WiFi router to assign the camera a static IP address. Otherwise, the system will stop working whenever the camera gets a new address!

5. Refer to your camera's documentation or search the web for a suitable URL for your brand and model of camera. This is a good resource for some cameras: https://security.world/rtsp/.
6. Formulate your RTSP URL with your camera's IP address (or hostname if you've added it to /etc/hosts or a DNS service), port number, login ID and password.
7. Test the RTSP URL with a program such as ffmpeg (command-line) or vlc (GUI). For example: `ffmpeg -y -i rtsp://Your-URL test.mp4` – this should generate a video file in test.mp4 (quit with Ctrl-C) or error messages.
8. Make sure you have a working RTSP URL before incorporating it into the configuration file.

Once you have written your configuration file and placed it in the configuration folder, test it:

```
cd src
make
cd ../config
../src/cam-config -c config.servername
```

If the configuration file is good, this will print nothing. If there are problems, you will see an error message.

Once you have a good camera RTSP URL and a good configuration file, install the software:

```
cd src
make
cd installer
./install.sh
```

You need to do this as a user with permission to run commands as `root` using the `sudo` command and you will be prompted for your password, as the installer needs to perform various actions as `root`.

This will:

- Install any missing packages, such as apache2 or ffmpeg.
- Create a user called `camview`.
- Install binaries from the `src` folder to `/usr/local/bin`.
- Create a backup folder and a download folder.
- Copy the configuration file to `/usr/local/etc/cam-view/`.
- Create and mount a RAMDISK in `/mnt/ramdisk`, where image files will be captured.
- Configure Apache2 to support CGI programs and create a CGI URL for the software at http://localhost/cam-bin. The files for this are in `/var/www/cam-view-bin`.
- Install shell scripts in `/usr/local/bin/cam-*.sh` and schedule some to run periodically in the security context of the `camview` user, for example to clean up old images and log files.

You can reverse all this and uninstall the software easily:

```
cd installer
./uninstall.sh
```

4.2 Troubleshooting the system

After creating a configuration file and running the installer, check that everything works:

1. Was the `camview` user created?
   `tail -f /etc/passwd`
2. Is the `cam-monitor` process running?
   `ps -ef | grep cam-monitor | grep -f grep`
3. Is there one `ffmpeg` process running per camera?
   `ps -ef | grep ffmpeg | grep -f grep`
4. Is the `/mnt/ramdisk` RAMDISK mounted?
   `df -h /mnt/ramdisk; ls -l /mnt/ramdisk`
5. Is there at least one camera folder under `/mnt/ramdisk/security` and are image files being written there?
   `find /mnt/ramdisk/security -name '*jpg'`

4.3 Web view of captured images

To check on the status of the system, you should be able to open this URL:

http://localhost/cam-bin/cam-status

To view "live" images (just those that merit being stored in the backup folder, as they are written there), use this URL:

http://localhost/cam-bin/cam-view

To download a video file or `.tar` archive of individual stored images, use this URL:

http://localhost/cam-bin/download-recordings

4.4 Remote backups

If you want to protect captured images against theft or damage of the capture server, you can configure a command to be run periodically to copy images elsewhere over the network – for example to a VM in the cloud.

The installer will check for a folder under the `config` folder called `.ssh.`*`hostname`* where the latter part is the output of `uname -n`. If this exists, it will copy files from there to `/home/camview/.ssh/` once the `camview` login account is created.

A good approach to doing this is to create an SSH private and public key-pair for the `camview` user, create an account on the remote system to which you will copy images and configure the target account to trust the `camview` user's public key:

- Login to the video capture server.
- `sudo bash` – *enter your password*.
- `su - camview`
- `ssh-keygen` – *press Enter at the prompts.*
- `mkdir /tmp/camview-ssh`
- `cp .ssh/* /tmp/camview-ssh`
- `exit` – i.e., log out of the `camview` account.
- `exit` – i.e., this time, log out of the `root` shell.
- `mkdir` *`packagedir`*`/config/.ssh.`‘`uname -n`‘
- `cp /tmp/camview-ssh/*` *`packagedir`*`/config/.ssh.`‘`uname -n`‘

You can run uninstall and install again, to verify that the installer will now create and populate `/home/camview/.ssh`.

Next, you have to create an SSH trust relationship. Copy the file `/home/camview/.ssh/id_rsa.pub` to the target account on the remote system and append its contents to that account's `authorized_keys` file:

- `scp /tmp/camview-ssh/id_rsa.pub` *`backupaccount@remotesystem`*`:/tmp/` – *enter the password when prompted.*
- `ssh` *`backupaccount@remotesystem`* – *enter the password when prompted.*
- `ssh-keygen` – *only do this if you have never done so on this account.*
- `cat /tmp/id_rsa.pub » .ssh/authorized_keys`.

Test that a trusted SSH login works from the video capture server to the backup server:

- `sudo bash` – *enter your password*.
- `su - camview`
- `ssh` *`backupaccount@remotesystem`*

You should get a login shell with no password prompt. If this does not work, check the SSH configuration. For example, on the remote system, in `/etc/ssh/sshd_config`, there should be an entry like `PubkeyAuthentication yes`.

Once you have a working mechanism for the local `camview` user to SSH to a remote account and system, you can add remote backup to the configuration file:

```
FILES_TO_CACHE=50
BACKUP_COMMAND=/bin/tar cf - %FILES%\
| /usr/bin/ssh backupaccount@remotesystem "cd security; tar xpf -"
```

In the above, the first parameter tells the system to send images to the remote backup system once every 50 captured frames. The second parameter specifies the command to run to send images over.

4.5 Activating changes to the configuration file

If you change the configuration file, for example by adding remote backup as described in the previous section, you need to install the new configuration file into the running system. There are two ways to do this:

1. Uninstall and (re)install the software. This is easiest but also the most disruptive.

   Note: Be sure to deploy the SSH key material in the config folder under `.ssh.`*`hostname`* so that it gets installed in the new `camview` user's home directory.

2. Change the configuration in place and stop/restart processes:

   - Copy the new configuration file to `/usr/local/etc/cam-view/config.ini`.
   - Stop the cam-monitor process:
     ```
     sudo bash
     /usr/local/bin/cam-view-stop.sh
     ```
   - Either wait for the process to restart automatically (there is a health checking job that runs every 5 minutes) or do it manually:
     ```
     sudo bash
     su - camview
     /usr/local/bin/cam-view-health-check.sh
     ```

4.6 Requiring passwords for web access

The default, simple installation does not require user passwords to access any of its URLs. To add passwords, you can create a password file. The installer will incorporate it into the Apache configuration

- `cd` *basedir*
- `cd config`
- `htpasswd -c htpasswd.`` `uname -n` `` *username*
- Enter the password you would like to use.

Uninstall and reinstall Cam-View and check that the URLs now require a password. The password file gets installed to `/etc/apache2/cam-view.htpasswd` and you can use the `htpasswd` command (which is a part of Apache2) to add users.

The web configuration in `/etc/apache2/conf-enabled/cam-view.conf` specifies the password file to use.

To use something more sophisticated, for example `libapache2-mod-auth-mellon` for SAML authentication via trust of an existing identity provider, you can modify `/etc/apache2/conf-enabled/cam-view.conf` directly but you are probably better off editing the relevant configuration file in `installer/web-server` so that your changes take effect on the next installation.

4.7 Multi-tenant deployments

In some cases, you want to capture video from many cameras but permit different users to view or download content from only a subset of cameras. This creates a small conflict:

1. The system should have a single configuration file for the `cam-monitor` process to use to fetch images from all cameras.
2. Each tenant or zone should have a unique URL, with its own, distinct configuration and password files, representing credentials used by people in that tenant and cameras in the tenant space.

This means we need both a global configuration and per-tenant configurations. Ideally we can do this without having to write the same configuration data in duplicate locations.

To address this, we first have to understand how the various Cam-View binary programs, including `cam-monitor` and the CGI user interfaces, find their configuration files. They all use the same logic:

1. In the current working directory, in a file called `config.ini`.
2. If that fails, in `/usr/local/etc/camview/config.ini`.

With this information, we are ready to write multi-tenant configurations:

- Create one configuration file with all parameters for all configuration files. In our example, we'll call this file `config.params`. This should set `BACKUP_DIR`, `BASE_DIR`, etc. but not define any cameras.
- Create one configuration file per tenant or zone. For example, call these `config.zone1`, `config.zone2`, etc. Near the top of each of each zone configuration file, write `#include "config.params"`.
- Create a "master" configuration file called `config.ini` in `/usr/local/etc/camview` which includes each of the zones.
- Create one CGI folder for each zone or tenant.
- In each CGI folder:
  - Copy that zone's configuration file to a local file called `config.ini`.
  - Place a copy of the `config.params` file.

  Consider using symbolic or hard links instead of copies to minimize the risk of later duplication.

Because a multi-tenant configuration is more complex than the simple case described earlier in subsection 4.1 on Page 7, the installer needs a bit of help to configure Apache2. To do this:

1. Create your own configuration file for Apache2. Create a file called `apache2-config.`*`hostname`* in the `config` directory, where *hostname* is the output of the command `uname -n`.
2. Place the Apache2 configuration in this file. See an example below.

3. Create *zone*`.htpasswd` files for each tenant or zone that will require password protected logins, also in the `config` folder. This was illustrated in subsection 4.6 on Page 11.

Example multi-tenant Apache configuration file:

```
ScriptAlias /zone1/ /var/www/zone1/
<Directory "/var/www/zone1">
  AuthType Basic
  AuthName "Restricted Content"
  AuthUserFile /etc/apache2/zone1.htpasswd
  Require valid-user
  AllowOverride None
  Order deny,allow
  Options +ExecCGI -MultiViews +SymLinksIfOwnerMatch
</Directory>

ScriptAlias /zone2/ /var/www/zone2/
<Directory "/var/www/zone2">
  AuthType Basic
  AuthName "Restricted Content"
  AuthUserFile /etc/apache2/zone2.htpasswd
  Require valid-user
  AllowOverride None
  Order deny,allow
  Options +ExecCGI -MultiViews +SymLinksIfOwnerMatch
</Directory>
```

Be sure to name the configuration and password files in the `config` folder as follows:

- Main configuration file that includes all zones: `config.`*hostname*.
- Configuration file for a single zone: `config.`*zone*.
- Password file for a single zone: *zone*`.htpasswd`.

Name zones using `ScriptAlias`, `Directory` and `AuthUserFile` directives in the Apache2 configuration file as shown in the example.

The installer should be able to handle things from there.

4.8 Internet access to local video

In a typical deployment, Cam-View is installed on a local network, which is not reachable from a public Internet address. If you want to access the Cam-View web UI at a public URL, and if you have a VM running on a cloud platform such as AWS or Azure, you can readily use port forwarding to accomplish this.

First, ensure that the cloud-hosted VM with a public, static IP address will accept inbound connections and forward them to ports on an SSH client via a tunnel. To do this:

- Lets call the video capture server `video` and the cloud VM `cloud`.
- Ensure that the SSH service is running on `cloud`:
  `ps -ef|grep sshd`
  If you don't see an SSHD process, install the package:
  `sudo apt install ssh`
- Edit `/etc/sshd_config`
- Add the following line:
  `GatewayPorts yes`
- Restart the SSHD service:
  `sudo /etc/init.d/sshd reload`

Next, create an SSH trust relationship:

1. Create a login account on `video`. Lets call it `client`.
2. Create a login account on `cloud`. Lets call it `server`.
3. Login as `client` on `video` and create a key-pair using the `ssh-keygen` command.
4. Login as `server` on `cloud` and create a key-pair using the `ssh-keygen` command.
5. Copy the public key from `client@video` to `server@cloud` and append it to the `.ssh/authorized_keys` file:
   - *client@video: $* `scp .ssh/id_rsa.pub server@cloud:/tmp`
   - *server@cloud: $* `cat /tmp/id_rsa.pub » .ssh/authorized_keys`
6. Verify that it is now possible to login from `client@video` to `server@cloud` without a password:
   - *client@video: $* `ssh server@cloud`

Next, write a shell script that runs as `root` on `video` which establishes a "reverse" SSH tunnel: Lets call this `tunnel.sh` and place it in the `/root` folder on `video`:

```
#!/bin/sh

while :
do
  echo "Opening ssh tunnel"
  echo '' | ssh -M -R '*:8881:localhost:80' -i /home/client/.ssh/id_rsa server@video "sleep
  sleep 1
done
```

Finally, write a shell script that is periodically run as root by the cron scheduler, which checks to see if a tunnel is running and starts it if not:

```
#!/bin/sh

PROCESS=`ps -ef|grep '8881:localhost:80'|grep -v grep`

if [ -z "$PROCESS" ]; then
  /root/tunnel.sh &
fi
```

Put this in `/root/tunnel-check.sh`.

Finally, schedule this job to run every few minutes, to ensure that even if the tunnel process failed for some reason, it gets restarted:

- Edit the cron table:
  `crontab -e`
- Add this line of text:
  `*/10 * * * * /root/tunnel-check.sh`

You should now be able to access the video system at this URL:

http://cloud:8881/cam-bin/cam-view

In the above, as noted earlier, `cloud` is the hostname or IP address of the cloud VM that terminates the reverse SSH tunnel.

5 Tuning

Cam-View is quite configurable, in particular in relation to the motion detection algorithm. To understand the parameters, it is important to first understand the algorithm.

5.1 Motion detection overview

Cam-View compares sequential pairs of images, which it reads from JPEG files on the filesystem (typically on a RAM disk). For each such pair, it tries to determine whether there has been motion, by seeing if the images are sufficiently different.

The challenge with motion detection is that images come in via a lossy file format (JPEG) and have noise from other sources, including RTSP network transport over (lossy) UDP, changing light levels, optical artifacts and natural small movements, such as moving shadows or rustling leaves in outdoor images. No two images are identical, even if they encode two scenes that are essentially the same.

The motion detection algorithm is designed to flag image pairs where there has been substantial change, while minimizing "false negatives" (failed to detect actual motion) and "false positives" (flagged motion where there was just noise).

The motion detection algorithm also has to be quite fast. Consider a deployment with just 10 cameras, each of which produces a "full HD" color image – i.e., 1920 pixels wide, 1080 pixels tall, 3 bytes per pixel. This means that comparing just two images requires processing 1920 x 1080 x 3 x 2 = 12,441,600 bytes of data. Comparing a pair of images across each of 10 cameras means processing 120MB of data. If images are acquired at just 1 image/second, this is 120MB/second (7.2GB/minute). This is all in-memory, as the raw image files are JPEG, typically about 10x smaller than the raw bitmaps.

5.2 Motion detection algorithm

Motion detection is a multi-step process, which proceeds as follows:

1. **Increase brightness of dark images:**

   First, check if either image is too dark. If so, increase the luminosity of both images by the same amount. Luminosity is increased by calculating the logarithm of each of the red, green and blue components of each pixel, multiplying by a factor and taking the exponent of the result.

2. **Subtract the images to generate a gray-scale:**

   Create a gray-scale image, where each pixel is a measure of how far the colors of the corresponding pixels in the two images are from one another. There are details in how this comparison is calculated that make it more useful in noisy images:

   - Color distance = the sum of the difference between red, green and blue colors of the pixels. All difference values are absolute (i.e., no negative values).
   - If the difference between any of the red, blue or green components of two pixels is less than a threshold amount, it is assumed to be "near enough" to zero and ignored.

- Color distance is compared to each neighboring pixel as well. In other words, each pixel in one image is compared to nine pixels in the other image. If the images are noisy, a visual element may appear in one pixel in one image and a nearby pixel in the other image, and such differences can be safely ignored. The smallest difference between a pixel in one image and its neighbors in the other image is used for the gray-scale output.
- To minimize computational cost, the 1-pixel edges of the input images are assumed to be identical and their differences are set to "0." This is not actually true, but we don't care enough to incur the extra computational expense of treating edge pixels, which have fewer than 8 neighbors, differently than "inside" pixels.

3. **Remove light or bright "speckles" from the difference:**

   The difference image may still be a bit noisy, with occasional bright pixels surrounded by dark pixels or dark pixels surrounded by white ones. Such one-off pixels are just noise and are removed. If the average brightness of neighboring pixels is above a light threshold and the pixel is below a dark threshold, then the pixel brightness is made the same as the average brightness of its neighbors. Similarly, if the average brightness of a pixel is below a dark threshold while the neighbors are above a light threshold, it is made the same as the average brightness of its neighbors.

4. **Reduce the gray-scale to a smaller black and white checkerboard:**

   The gray-scale image is reduced by a factor of N, where each NxN gray-scale pixels are mapped to a single pixel which is either black or white. For each of the NxN pixels in the gray-scale, a value of 1 or 0 is assigned based on whether the brightness of that pixel is above a threshold. Next, corresponding the checkerboard pixel is set to 1 if the total number of NxN pixels set to 1 is above a second threshold. Essentially, a checkerboard pixel is white if a sufficient number of the NxN gray-scale pixels are above a brightness threshold.

5. **Count the white pixels:**

   Finally, the number of white pixels in the resulting checkerboard is counted. If they represent at least a threshold percent of the image area, then the image pair is considered to have motion. If they are below the threshold, then the pair of images are considered to be static.

5.3 Motion detection parameters

The algorithm described above mentions various parameters. Each parameter has a built-in default plus the configuration file can specify either global default values or per-camera values for each parameter. The parameters are as follows:

1. **Increase brightness of dark images:**

Parameter	The pixel value of 'dark' below which image brightness will be increased. Values are out of 255(red) + 255(green) + 255(blue) = 765.
Built-in default	40
Global default	`DEFAULT_COLOR_DARK`
Per-camera default	`COLOR_DARK`

Parameter	How much to boost the brightness of dark pixels. New brightness (per color) = exp(log(old brightness) * factor).
Built-in default	1.5
Global default	`DEFAULT_DARK_BRIGHTNESS_BOOST`
Per-camera default	`DARK_BRIGHTNESS_BOOST`

2. **Subtract the images to generate a gray-scale:**

Parameter	Minimum meaningful difference between the red, green or blue components of corresponding pixels in two images.
Built-in default	40
Global default	`DEFAULT_COLOR_DIFF_THRESHOLD`
Per-camera default	`COLOR_DIFF_THRESHOLD`

3. **Remove light or bright "speckles" from the difference:**

Parameter	Maximum brightness of pixel in an image, out of 765 (R+G+B), if the pixel is to be considered "dark."
Built-in default	30
Global default	`DEFAULT_DESPECKLE_DARK_THRESHOLD`
Per-camera default	`DESPECKLE_DARK_THRESHOLD`
Parameter	Minimum average brightness of neighboring pixels, out of 765, for those pixels to be considered "bright."
Built-in default	200
Global default	`DEFAULT_DESPECKLE_NONDARK_MIN`
Per-camera default	`DESPECKLE_NONDARK_MIN`
Parameter	Minimum brightness of pixel in an image, out of 765 (R+G+B), if the pixel is to be considered "light."
Built-in default	140
Global default	`DEFAULT_DESPECKLE_BRIGHT_THRESHOLD`
Per-camera default	`DESPECKLE_BRIGHT_THRESHOLD`
Parameter	Maximum average brightness of neighboring pixels, out of 765, if they are to be considered "dark."
Built-in default	60
Global default	`DEFAULT_DESPECKLE_NONBRIGHT_MAX`
Per-camera default	`DESPECKLE_NONBRIGHT_MAX`

4. **Reduce the gray-scale to a smaller black and white checkerboard:**

Parameter	Size of a square in the difference gray-scale image that will be mapped to a single pixel in the black-and-white checkerboard. Measured in pixels (both width and height).
Built-in default	8
Global default	`DEFAULT_CHECKERBOARD_SQUARE_SIZE`
Per-camera default	`CHECKERBOARD_SQUARE_SIZE`
Parameter	Minimum brightness (out of 255) of a pixel in the gray-scale difference image that constitutes a "light" pixel.
Built-in default	100
Global default	`DEFAULT_CHECKERBOARD_MIN_WHITE`
Per-camera default	`CHECKERBOARD_MIN_WHITE`
Parameter	Minimum number of "light" pixels (as above) in the NxN square area of the gray-scale difference image, before a pixel in the "checkerboard" black and white image should be set to "on." Note that this is out of a maximum of CHECKERBOARD_SQUARE_SIZE squared.
Built-in default	33
Global default	`DEFAULT_CHECKERBOARD_NUM_WHITE`
Per-camera default	`CHECKERBOARD_NUM_WHITE`
Parameter	Percentage of the checkerboard pixels that are lit up as "white" using the above method before the two images that formed the original difference gray-scale are considered to be different enough to indicate motion. For example, 0.02% in a 1920x1080 image which is sub-sampled into 8x8 blocks means 0.0002 $\times$ 1920 $\times$ 1080 / (8 $\times$ 8) = 6 pixels.
Built-in default	0.02%
Global default	`DEFAULT_CHECKERBOARD_PERCENT`
Per-camera default	`CHECKERBOARD_PERCENT`

6 Command-line utilities and shell scripts

Cam-View includes a handful of command-line programs and shell scripts which are useful in the operation of the system. These are described below.

6.1 cam-archive

Cam-View accumulates large numbers of JPEG image files in the backup folder. Over time, these will fill all available drive space. Moreover, on most filesystems, very large folders have poor runtime performance.

It makes sense to (a) collect large numbers of small image files into a few large archive files and (b) delete older archive files. These two tasks are the job of cam-archive:

`cam-archive` must be supplied with a path to the Cam-View configuration file:

```
cam-archive -config CONFIGFILE
```

All other arguments are optional. They include:

- `-camera` *`ID`* – operate on a single, specified camera. Otherwise, `cam-archive` operates on all configured cameras.
- `-days` *`N`* – operate on a data from N days in the past. The default is 0, meaning today.
- `-range` *`N1 N2`* – operate on a data from multiple days, starting with N1 days ago and until N2 days ago. For example `-range 0 5`.
- `-count` – print the number of image files per camera.
- `-tar` – create one or more ".tar" archive files of images matching the above conditions. There will be one archive file per day. The file9s) will be placed in camera-specific backup folders and named after the date.
- `-remove` – delete `.jpg` files that were archived.

6.2 cam-view-archive.sh

A simple shell script that:

- Archives images that are 2 or more days old, for each configured camera.
- Deletes archive files that are 21 or more days old.
- Deletes log files that are 10 days or older on the RAM disk.

This shell script is normally scheduled to run periodically as the `camview` user.

6.3 cam-config

Used to diagnose configuration files and to extract the values of configuration variables.

The sole mandatory argument is `-c configfile` - which specifies the configuration file on which the program should operate.

The two optional arguments, used separately (not together) are:

- `-listincludes` – prints a list of files included by the main configuration file. This is useful in scripts that copy configuration files, such as at installation time, as it tells the script what other files are required.
- `-printvar` – prints the value of a single variable, including any expansions. This is useful to locate folders, such as the `BASE_DIR` or `BACKUP_DIR`, for use in scripts such as `cam-view-archive.sh`.

This program will also print errors if a configuration file cannot be parsed. If the configuration file includes the `#print` directive, it will also cause the fully expanded configuration to be printed out to the specified file.

6.4 cam-view-clean-download.sh

When users download archives or movie files of captured images, the downloaded material is stored in the `downloads` folder, under the `BACKUP_DIR` folder specified in the configuration file. This shell script is used to find and remove older download files – by default those that are at least 30 minutes old.

This shell script is normally scheduled to run periodically as the `camview` user.

6.5 cam-view-health-check.sh

This is used to check that `cam-monitor` is running and, if not, to stop any running ffmpeg jobs and restart it. This is also how `cam-monitor` is usually started - by a scheduled cron job that runs this script every few minutes as the user `camview`.

6.6 cam-view-stop.sh

This is used to stop any running `ffmpeg` and `cam-monitor` jobs. It is normally only used in the context of system diagnostics or Cam-View software development.

This page intentionally left empty.