How to build, install and run the APE testbed

APE project, Uppsala University

November 8, 2002

Contents

1 Introduction

The APE testbed is a project started at Uppsala University, Sweden and has been partly funded by Ericsson AB. The main concept of the APE testbed is to provide an environment which greatly simplifies the process of performing complex real world tests of ad hoc routing protocols. More specifically, APE focus on smooth deployment, high ability of customization and ability to easily run several protocol implementations for comparisons. APE also try to make testruns reproducible by using scripted scenarios with movement instructions to test-participants during the actual testruns. These scenarios also define traffic generated during the tests, so that most aspects of the tests are automated and easily re-produced. The only interaction necessary from test participants is to follow the on-screen movement instructions. Node setup is automated and IP-addresses of nodes are defined in the scenario files. Log files are generated on every node participating in the test and are easily uploaded to a “collect node” at the end.

APE also include a MAC-filter called **mackill** which can force different connectivity setups using stationary nodes. The MAC-filter can be used in a scenario, so that topology configurations can be simulated dynamically. This can be extremely useful for testing the routing logic of protocol implementations.

For evaluation, APE includes performance and network characterization metrics, and tools to analyze them. These metrics are based on data gathered during tests and aim to be protocol independent, so that they can be evaluated no matter what protocol implementation is used.

APE introduce a new mobility metric for real world tests called “virtual mobility” (vM), which is based on SNR changes in links between nodes. SNR is mapped to a “virtual distance”, then vM is calculated as the changes in “virtual distance” between two consecutive time intervals. vM is mainly used for two purposes: one is to estimate the degree of similarity between two testruns (in a mobility and movement sense, but also in a connectivity sense, since vM is dependent on inter-node connectivity), and to provide a way to evaluate how it affects other metrics, for example packet loss.

This document will give basic introduction to installing the binary APE distribution, typically called ape-<version>.zip, and installing and compiling the APE source code for building your own APE distributions. It will also briefly explain the basics of the APE distributions and how to perform customizations.

1.1 Terminology

To avoid confusion about the different parts of APE, this section will explain some of the terminology used.

APE Testbed General term describing the testbed as a whole.

APE See APE Testbed.

APE distribution This is the execution environment in which test runs are performed. More specifically it is a small Linux distribution which usually comes as a .zip file and installs on a Linux or Windows system. Also available as a bootable CD iso image.

APE sources Software package which contains all the source code and build scripts to customize and build your own APE distribution. Also contains the APE analysis tools.

1.2 The APE distribution

The major part of the APE testbed consists of a stripped down Linux distribution based on Red Hat Linux. The purpose is to have an easy to install test environment. The APE distribution easily installs on Windows 98 (or other DOS enabled Windows systems) and Linux without the need for repartitioning or other pre-installation preparations. It is also possible to run APE from a bootable CD-ROM, which eliminates the requirement to have a compatible host operating system installed on the computer.

The APE distribution and related tools and test scripts are tailored towards users unfamiliar with the details of performing test runs. The idea is that it should be simple to make use of students, colleagues and other people to be able to perform large scale tests. Therefore, instructions and test scripts are written with this in mind.

The APE distribution's root file-system is kept within an image file which makes it possible for it to be installed on a host file-system as a regular file. Currently Linux ext2, ext3 and reiserfs along with Windows FAT16/32 are supported as hosting file-systems. NTFS should be possible too, but Linux write support for NTFS is still experimental, so proper functionality can not be guaranteed.

1.3 Data gathering

A modification of the wireless extensions "spy mode" of the IEEE 802.11 WaveLAN drivers, called *"superspy"*, enables logging of signal level and signal noise for all packets picked up by a wireless interface (the interface is put in promiscuous mode). This ethernet level data is the basis of the vM metric, but can also be used for several other purposes, such as estimating the density of the network or the connectivity.

APE scenarios currently use the standard Ping tool for basic IP-level performance metrics. Events (sending/reception of packets) are timestamped and logged. Things which can be examined using ping are round trip time (RTT), actual routes taken by packets, basic connectivity, packet loss and more. However, any type of traffic generating program can be used and scripted in APE scenarios.

To be able to time synchronize logs after testruns, APE have the Time Stamp Broadcaster (**TSB**) which runs on every node during testruns. It regularly broadcasts the local node's time to other nodes. All nodes can then gather the information necessary to time synchronize logs against one of the other nodes.

1.4 Analysis

The APE testbed includes several analysis tools to analyze the data gathered during testruns. They work either at ethernet level or IP level and are located in the src/analysis directory. The tools are written in Perl and are command line based, but there is also a GUI wrapper available. See section 9.2 for more information.

The APE testbed also have APE-view, a Java-based node movement visualization tool, which is driven by the log files from testruns.

2 Getting started

To get started on performing testruns and doing analysis please consult the QUICK_START guide in the source package. Also carefully read the requirements section below to make sure you have all the hardware and software necessary to make full use of APE. If you want to start modifying the APE distribution and add customizations, this document is the best source of information. Other parts of the APE testbed, such as the analysis tools, are documented in their respective directories. If you want to know about documented problems or bugs with the APE testbed, consult the KNOWN_ISSUES file in the source package.

2.1 Requirements

2.1.1 Participant nodes

- Hardware:
  i386 compatible computers (preferably laptops) equipped with APE supported IEEE 802.11 WaveLAN cards. Make sure to use a recent firmware to avoid trouble.
- OS: Windows 95/98 or Linux. Alternatively a computer which can boot from CD-ROM.
- Software: An APE distribution.

2.1.2 Collect node

This computer will receive log-files from client machines using rcp (remote copy) or scp (secure copy) and typically does **not run the APE distribution** .

- Hardware:
  i386 compatible computers (preferably a laptop) equipped with a APE supported 802.11 WaveLAN device (see section 2.1.3), running in ad hoc mode with same network configuration as participant nodes.
- OS: Linux.
- Software:
  rsh or ssh server running so that log files can be received (see README).
  Should be configured to run with a wireless interface in ad hoc mode using the same network configuration as participant nodes. Default network setup in APE is 192.168.5.0/255.255.255.0.

2.1.3 Supported WaveLAN devices

- Cards supported by either the **orinoco_cs** or alternatively the **wavelan2_cs** drivers. (ORINOCO IEEE 802.11 WaveLAN cards. Firmware 6.06 or higher strongly recommended).

2.1.4 Computer to build the APE distribution on

Linux machine to compile the APE sources on. You need root-access on this machine to be able to perform all steps necessary to create your own APE distribution. Recommended setup is an i386-compatible machine running a recent version of Red Hat Linux. Other distributions might work as well, but the reference and development system is Red Hat.

3 Installation instructions

This document makes a distinction between a binary APE distribution package and the source code, which is used to build the binary package. A binary distribution package is required in order to install the APE testbed on a host computer and perform tests, while the source code is necessary to modify and build customized APE distributions.

3.1 Installing the binary distribution on Windows

Installing on Windows 98 is the easiest option, and requires least steps to get the APE system up and running. The kernel is booted using the DOS utility “LoadLin”, which makes installation as simple as extracting the contents of the binary APE zip-archive to the root of the Windows file-system (eg. c:\). The system can then be booted by clicking on the shortcuts available in c:\APE.

3.2 Installing the binary distribution on Linux

Linux does not allow another operating system to take over the computer resources while running, as a DOS-based system does. Therefore, a utility like LoadLin can not be used. Instead booting the APE distribution is done via the Linux bootloader (either lilo or grub) if available. An install script, performing any necessary modifications to the bootloader setup, is available in the installation folder of the binary distribution, usually /APE. The installation can be summarized in the following steps:

1. Extract the contents of ape-<version>.zip to /

```
> cd /; unzip /path/to/ape-<version>.zip
```

2. Run /APE/install-linux.sh

```
> cd /APE; ./install-linux.sh
```

3. Reboot and select APE at the bootloader prompt.

Any changes made by the “install-linux.sh” script can be reversed by running “uninstall-linux.sh” located in the same directory as the install script.

4 Booting and running the APE distribution

To boot the APE distribution after installing on a supported Windows operating system, simply navigate to `c:\APE` and double click on the “Start APE!” icon. If the distribution has problems booting, you can try clicking on any of the other icons which match the memory configuration of the computer.

After installing on a Linux based host OS, the computer must be restarted and APE selected at the bootloader prompt.

4.1 Booting from the APE CD-ROM

If you use an APE bootable CD-ROM, make sure your computer is set to boot from CD-ROM in your BIOS. Then insert the APE CD and restart the computer.

4.2 Keyboard configuration

If the keyboard does not match the default keymap configuration in APE you can load a suitable keymap located in `/usr/lib/kbd/keymaps/i386/`. Example command:

```
> loadkeys se-latin1
```

4.3 Running the APE test script

To start an APE test run, execute the “`start_test`” script in the home directory of “root”.

```
> ./start_test
```

Follow the on-screen instructions and select scenario to run, node number and protocol to evaluate.

4.3.1 Synchronized test start

The testscript will halt before the testrun is started. At this point the test run can be started by any of the nodes by pressing the **“Enter”** key. NOTE: The node that press “Enter” will send out a “start packet” to all other nodes within range and starting them automatically, so that the start is synchronized.

4.3.2 Test data uploading

After a test, log files can be uploaded to a “collect node” by running the “`upload.sh`” script (collect node need to be configured and accessible).

```
> ./upload.sh
```

4.4 Log files

Log files are written to `/var/log/` during tests, but the path can be configured in the **ape.conf** file (see section 7.2). Log files are also gzip-compressed when written to disk to save space.
NOTE: After tests, log files are packaged into a tarball and copied to the host OS filesystem or whatever storage path is defined in the ape.conf file. If APE is run from a bootable CD-ROM, there is no host OS, therefore log files are copied to /tmp (a ramdisk).

ping.log - Ping output with APTS timestamps.

superspy.log - Ethernet level signal quality and connectivety log.

tsb.log - Timestamp log.

Custom log files with names that end with "`.apelog`" and are written to the APE log write path defined in **ape.conf**, will automatically be uploaded to the collect node if the `upload.sh` scipt is used.

4.5 Streaming mp3 music

If you have compiled and configured sound for your APE nodes, you may try some mp3 streaming over the ad-hoc network. APE is configured for standard Sound Blaster compatible sound by default.

APE include a small mp3 streaming application called **mp3stream**. It acts both as sender and receiver and uses UDP. To send an mp3 stream to a node, run:

```
> mp3stream <music.mp3> <node IP>
```

You must have access to a mp3 file, often one can be available on the host OS filesystem, accessible through `/mnt/`. <node IP> is the IP-address of the node which will receive the mp3 stream. The receiver must run:

```
> mp3stream | amp -
```

to receive the music (you may have to do "`modprobe sound`" first).

You can also try some of the mp3-streaming example scenarios.

4.6 Switching between ad-hoc and basestation mode

A convenience script called **netswitch** is available for switching between ad-hoc mode and base station mode (Access Point mode without encryption).

Going from AP-mode to ad-hoc, simply run:

```
> netswitch
```

When going to AP-mode you may also specify the network name (ESSID) unless correctly configured in ape.conf:

```
> netswitch [ ESSID ]
```

Currently there is no way to (automatically) set encryption keys to access encrypted APs.

5 Short guide for building an APE distribution

These are the steps necessary to build a binary distribution package. First download and install the APE testbed source code. Then perform the following steps from the APE source directory:

1. Download kernel source from `ftp.kernel.org` or favorite mirror. A 2.4.x based kernel is recommended. Extract to the `kernel/` directory of ape source tree. Rename extracted linux directory to `linux-<version>`. Symlink new dir to `kernel/linux` (You may want to chose a kernel version that have a matching mackill-patch in `kernel/patches`).

   ```
   > cd kernel; tar -zxvf linux-<version>.tar.gz
   > mv linux linux-<version>; ln -s linux-<version> linux
   ```

   It is not necessary to configure or compile the kernel at this point. See README in the `kernel/` directory for more information.

2. Download PCMCIA source which have a matching patch in `kernel/patches` from `http://pcmcia-cs.sourceforge.net/ftp/`. Extract to `kernel/` directory and symlink to `kernel/pcmcia-cs`.

   ```
   > cd kernel; ln -s pcmcia-cs-<version> pcmcia-cs
   ```

   There is no need to perform any configuration of the pcmcia package.

3. **OPTIONAL:** Alternatively, to use the ORINOCO wavelan2_cs driver v6.16: Download driver from `ftp://ftp.orinocowireless.com/pub/software/ORiNOCO/PC_Card/linux`. Install into the pcmcia-cs directory.

4. Apply patches to the PCMCIA wireless drivers (superspy support) and Linux kernel (MAC-kill support).

   ```
   > cd kernel; ./patch-o-matic
   ```

   See `kernel/README` for more information about patching.

5. **OPTIONAL:** Download and install supported protocol implementations to be included. See section 6 below.

6. Configure `dist/configuration/etc/ape.conf` to your preference.

7. Kernel setup in root of ape-sources tree (root permissions required). This will make sure /usr/src/linux points to the APE kernel.
   ```
   > make kernel_setup
   ```

8. Compile all source code (kernel, pcmcia, APE-source code...) (**NOTE***: You must save the kernel config file when you exit the kernel configuration program.*)
   `> make`

9. Build a .zip package or bootable cd image (root permissions required).
   `> make zip` or `> make bootcd`

The resulting binary APE distribution package or bootable CD-ROM iso image will be created in the top-level source directory. More specific make targets, modifications and options will be discussed further on throughout this document.

5.1 APE kernel compilation and issues

The kernel is also compiled in the default make target. For the kernel to compile properly on your system, some symbolic links has to be setup. On Red Hat systems /usr/src/linux usually is a symbolic link pointing to the currently used kernel source tree. This link has to be updated to point to the kernel source tree of the APE kernel. This is automatically done when doing a “make” or “make all”. However, one must have root-permissions to update the links, so it is suggested that a “make kernel_setup” is issued as root before compilation. Any changes to the system is stored in the `.system_restore` file in the root of the ape-sources tree. A “make restore” reverses the effect of a “make kernel_setup” and is automatically done if the kernel is compiled with root permissions.

See section 7.1.1 about changing the default configuration options in the kernel setup.

5.1.1 Common errors

- If you get errors during kernel compilation, make sure you selected save configuration when leaving the kernel configuration program.

6 Including routing protocol implementations

The APE testbed is prepared for integration of a number of protocol implementations available on the Internet. This means that Makefiles and APE build scripts have been prepared so that installation and compilation of these implementations is easy, and that they can be used in scripted tests. Supported protocol implementations are:

- Uppsala University AODV implementation. This is our own implementation and is included in APE. Also available at `http://www.docs.uu.se/docs/research/projects/scanet/aodv/`

- LUNAR - Lightweight Underlay Network Ad hoc Routing. A lightweight protocol for small ad hoc networks. `http://www.docs.uu.se/selnet/lunar/`

Third party protocols

- Madhoc AODV implementation v1.0 (included in APE distribution) available from `http://mad-hoc.flyinglinux.net`.
- Kernel AODV implementation. `http://w3.antd.nist.gov/wctg/aodv_kernel/`
- IMEP/TORA implementation from `http://www.cshcn.umd.edu/tora.shtml`. Requires kernel 2.2.x.
- OLSR implementation v1.0 from `http://menetou.inria.fr/olsr/`.
- DSR kernel implementation from University of Queensland `http://piconet.sourceforge.net/`.

Some of these may come included as part of the APE package, but others may not. When including protocols for which there are no preparations done in the testbed, read section 7.1.4.

6.1 Protocol installation and issues

Including a “supported” protocol implementation is done by following the installation instructions below. The compilation of the protocol implementation is then automatically done when building the testbed.

6.1.1 Uppsala University AODV:

Installation: Included in APE distribution.

Issues: No known issues.

6.1.2 LUNAR:

Installation: Included in APE distribution.

Issues: No known issues.

6.1.3 Madhoc AODV:

Installation: Included in APE distribution.

Issues: No installation issues. However, we have experienced some problems with this version (1.0) of the implementation, e.g. with route set-up. Consult Madhoc web page for further information and future updates.

6.1.4 IMEP/TORA (UMD):

Installation: Extract package to src/. Directory should be called “umd-manet”.

Issues: Requires a 2.2.x kernel. This version (by August 2001) can have trouble compiling on some systems, due to bad ANSI-C conformity. If you get a complaint about no return type of function at line 78 of imep_arp.cc, just add void as return value to that function. Our experiences with this implementation are that the logic seems OK, but it seems to be a bit unstable and sometimes core-dumps. Consult IMEP/TORA web page for further information and future updates.

6.1.5 OLSR (INRIA):

Installation: Extract package to src/. Directory should be called “olsrd”.

Issues: Seem to be missing an include of <time.h>. If it does not compile and complains about implicit declaration of “ctime”, add #include <time.h> to the defs.h header file. Also have a lot of functions that may complain about return values. For some reason the code is compiled using the C++ compiler (g++) as default, while the code seem to be exclusively C. Change to gcc in the Makefile to avoid that the binary is linked to the C++ libraries. Otherwise, there may be a version mismatch between the libraries on the compiling system and those in the APE distribution which will stop the OLSR daemon from running. Consult OLSR web page for further information and future updates.

6.1.6 DSR (University of Queensland):

Installation: Extract package to src/. Directory should be called “dsr-uq”.

Issues: To build for the x86 platform do a “make dsrx86.o”, but this is also taken care of by the main APE build script. During compilation, gcc may complain about *too few arguments to function ‘ip_select_ident’*. To fix change line 401 of `dsr_output.c`:

```
ip_select_ident( (*skb)->nh.iph, (*skb)->dst);
```

to

```
ip_select_ident( (*skb)->nh.iph, (*skb)->dst, (*skb)->sk);
```

Has problems with route set-up, due to ARP failures.

7 Customizing the APE distribution

The APE distribution is not entirely built from sources, it would simply be too much to handle easily. Therefore a basic file-system is available in `dist/fs-base/rootfs-base.tgz`, with the most common and essential libraries and executables necessary for a minimal system. These files are from a standard RedHat 6.2 installation and are used to create the APE file-system when doing a `make zip` or `make bootcd`. However, as

of APE version 0.4 the default behavior when building an APE distribution is to “rip” binaries and libraries from the build system and incorporating them into the APE distribution. This will make sure that binaries and libraries are always up to date. This section will describe how to add both binary components and components built from source code.

7.1 Software installation

To minimize the size of the APE distribution its sources does not include some of the software components that are easily downloaded from the Internet, such as the Linux kernel source and the pcmcia driver package. This section contain a more detailed description of how to install software components necessary to compile the APE distribution.

7.1.1 Linux kernel source

Installing the Linux kernel source code should almost be as simple as extracting a new kernel source package into the kernel directory. The new source directory should be named “linux-<version>”. The linux kernel source directory must then be symlinked to kernel/linux. A standard APE-preconfigured .config-file which holds the kernel configuration is available in kernel/ape-config, named config-<kernel version>. This file is used as a basic configuration for the kernel. This way no further configuration is necessary unless special options are needed or an incompatible kernel version is used.

MAC-filter kernel support

To make use of the APE MAC-filter it is necessary to apply some small modifications to the kernel source. See the README file in `src/mackill`. Alternatively check the `kernel/patches` directory for a suitable patch, matching the kernel version you plan to use.
Apply with the “Patch-O-Matic” script or alternatively (from `kernel/` directory):

```
> patch -p0 < patches/mackill-linux-X.X.X-<date>.patch
```

7.1.2 PCMCIA

Installing the PCMCIA package is similar to installing the kernel source code. Simply extract the new package to the kernel directory. The new PCMCIA source directory should be named pcmcia-cs-<version> and then symlinked to kernel/pcmcia-cs. To add support for the APE superspy mode to the supported WaveLAN drivers, run the “Patch-O-Matic” script in the `kernel/` directory. See the `kernel/README` file for more information.

To use the the ORINOCO `wavelan2_cs` driver not included in the standard PCMCIA distribution, the ORINOCO driver package (v6.16) can be downloaded separately and installed into the PCMCIA directory. Patch the driver by executing the “Patch-O-Matic” script.

7.1.3 WaveLAN drivers

APE uses modified versions of wavelan drivers. See section 2.1.3 for a list of supported drivers and cards. Patches for drivers are available in the kernel/patches directory. In the modified drivers, code sections that have been changed or added are marked with `/*******APE_BEGIN*******/` at the start, and `/*******APE_END*******/` at the end. It should be fairly straightforward to merge these changes with any new version of the drivers, unless there have been major changes to a driver between releases.

7.1.4 Adding a new routing protocol

The APE testbed have a system for using different routing protocols in APE scenarios. This system makes sure that all pre-test configurations necessary for a specific protocol are done properly and that the protocol will be available as an option when running a test. To make the system recognize the new protocol, a setup script must be created in `src/runtime/protocol-setup`. The scripts that are already there may be used as templates for a new script.

The setup script will take three arguments, the first is a command string, which is either “start” or “stop”. Depending on the command received the setup script should perform either protocol initialization or cleanup. The second argument is the name of the interface the protocol is supposed to run on. Note that given the “start” command, the protocol implementation should also be started, and then killed when given the stop command. The third argument is the IP address to be used during the test-run.

In the script header there should also be a `# PROTOCOL=<name>` line which declares the name of the protocol. This name will be shown in a menu where you select protocol to evaluate during test startup.

Also do not forget to include all necessary parts of your protocol implementation in the APE distribution.

7.2 APE configuration file (ape.conf)

The runtime configuration of APE nodes can be customized by editing the central APE configuration file in `dist/configuration/etc/ape.conf`. The idea of this file is that it should not be necessary to edit any other files to change the behavior of APE nodes. Some of the things which can be changed from ape.conf:

- APE network configuration.
- Collect node configuration.
- Data gathering.
- Data uploading.

7.3 Wireless configuration

The default setup of the wireless network interface can be changed by editing the `dist/configuration/etc/pcmcia/wireless.opt` file. This is a standard PCMCIA package configuration file.

7.4 Adding libraries and files

A routing protocol or other APE specific software for which there is source code available, is typically installed in the `src/` directory and compiled and installed from the local Makefile. A Makefile variable called `$(INSTALL_PATH)` is exported in the Makefiles throughout the APE source tree. This variable can be used to install software into the APE distributions whenever a `make install` is done from the top level Makefile.

If there is a program (binary) on the system APE is built on which should be incorporated into APE, then add an entry to the `dist/rootfs_files` file. Programs and files listed there, will automatically be searched for on the build system and installed into the APE distribution with the indicated path and permissions.

7.5 APE file-system images

The file-systems of the binary APE distribution is kept in so called image files. The image files are created when doing a “make dist”. A basic file-system tree with binaries are located in `dist/fs-base` and are extracted to the root file-system image file during the distribution compilation process. The binaries are from a redhat 6.2 distribution and are stripped of debugging information to save space. As of APE version 0.4 binaries and libraries are ripped from the build system. See section 7.4.

7.5.1 Increasing the size

If additional space is required for the root file-system, the image file size can be adjusted in the `mk_images script` located in the `dist/` folder.

7.5.2 Mounting an image file

Before mounting a file-system image file, it has to be associated with a loop device (eg. /dev/loop0). This is done in Linux by issuing the command.

```
> losetup /dev/loop{0,1,2...} /path/to/imagefile
```

Consult the losetup man page for more information. The file-system can then be mounted as any other mountable block device with:

```
> mount /dev/loop{0,1,2...} /path/to/mountpoint
```

The two steps above can be replaced with the following:

```
> mount /dev/loop{0,1,2...} /path/to/mountpoint -o loop
```

The “-o loop” option makes mount automatically perform the loop device setup. Consult the mount man page for more information.

7.6 The APE boot process

To be able to make make changes to the APE-distribution or fix any unforeseen incompatibilities with computers, it may be necessary to understand the special boot process when using a loopback root file-system.

The binary distribution typically contain the following components:

- initrd.gz - Initial RAM-disk image (gzipped).
- rootfs.img - Root filesystem image.
- swap.img - Swap filesystem image.
- vmlinuz - Linux kernel.
- Loadlin - DOS based Linux boot utility.

Usually, initial RAM-disks are used to load any driver modules needed before the main root filesystem can be mounted. An example would be SCSI-driver modules necessary to operate the SCSI-harddisk containing the root filesystem. In the APE distribution the root filesystem is kept within an image file and to be mounted it must first be associated with a loop device (/dev/loop). This is done through the “losetup” command (see losetup manual page). Unfortunately this can not be done before a root filesystem is mounted, where the losetup binary can be accessed.

The problem is solved by using an initial RAM-disk (initrd) which is mounted as a temporary filesystem kept in memory. This filesystem holds a losetup binary which enables the root filesystem image file to be mounted. When the initial RAM-disk is mounted, the kernel automatically executes a program or script named linuxrc in the root of the filesystem (/linuxrc), if available. In APE this is a shell script that automatically searches commonly used partitions (hda1, hda2, hdb1 and so on) for a supported filesystem (vfat, ext2, ext3, reiserfs) holding the root filesystem image file (rootfs.img). See dist/files/initrd/linuxrc. If the root filesystem image file is found it is associated with a loopback device (default is /dev/loop7) as described above. After “linuxrc” has finished executing, the kernel resumes normal boot procedures and mounts the root filesystem as defined by the kernel boot option which in APE is /dev/loop7. Since this device is now associated with the root filesystem image file, the root filesystem is mounted as normal. The RAM-disk mount-point is automatically transferred to /initrd, but only if that directory is available in the newly mounted root filesystem. See the initrd manual page for more information. This makes the initial RAM-disk accessible through /initrd. Also the filesystem holding the root filesystem image should be accessible in /initrd/mnt (if that was its mount point), unless unmounted in the linuxrc script.

8 Conducting tests

8.1 Creating scenario files

To make use of scenario choreography and automatic traffic generation a scenario file has to be created. Scenario files define the movement and traffic of every node participating in the test. Example scenario files are located `src/scenarios` and they should

```
choreography.scenario.title=Lost n found
choreography.total.nodes=4
choreography.startup.command.0=startup
choreography.shutdown.command.0=copy_files
choreography.shutdown.command.1=pack_files

# node 0   ------------------------------------------------------------

node.0.ip=193.10.133.10
node.0.ipmask=255.255.255.128
node.0.action.0.msg=Test is starting...
node.0.action.0.command=start_spyd
node.0.action.0.duration=1
node.0.action.1.command=ping_node 1 580
node.0.action.1.msg=Stay at this location. You are pinging node 1. (30sec).
node.0.action.1.duration=30
```

Figure 1: Example from a scenario file

make a good starting point for creating new scenario files. There are also a number of scenario generating scripts in `src/scenarios/scenario_generators` that may simplify the process. To make sure the new scenario file is included when building an APE distribution, place the scenario file in `src/scenarios` with a .ape file extension.

Shown below in Figure 1 are some lines from an example scenario file with instructions for node 0.

8.1.1 Scenario setup instructions

Every scenario file starts with instructions beginning with “choreography”. These are instructions that apply to all nodes participating in the scenario. First we have

```
choreography.scenario.title=Lost_n_found
```

which define the title of the scenario. The title will be shown at a scenario selection screen when the test is started. It is important that the title does not contain any white-spaces.

```
choreography.total.nodes=4
```

This instruction defines the total number of nodes participating in the test.

```
choreography.startup.command.0=startup
```

This instruction define a command to run at startup, before the test is started. It is possible to define more startup commands by giving them separate numbers, like 0, 1, 2, 3. . . and so on. They will also execute in that order. Note that at least the startup command shown above should be available in a scenario file.

In a similar way it is possible to define shutdown commands which are executed after the test is finished:

```
choreography.shutdown.command.0=copy_files
```

8.1.2 Node instructions

After the scenario instructions that apply to all nodes, there are node specific instructions. These instructions define the movements and traffic generated by individual nodes. Node instructions generally have the form “node.<node number>.<type>”. Every node should set an IP-address and netmask:

```
node.0.ip=193.10.133.10
node.0.ipmask=255.255.255.128
```

Then we have “action” instructions which are executed during the actual test. “Action” instructions are numbered and executed in order. There are three types of instructions that can be specified for each “action”:

- **msg:** A message to be displayed on the screen of the computer.
- **command:** A command to be executed. Basically a standard shell command, or one of:
  - **start_spyd** Start the superspy logging.
  - **ping_node** Start a ping session. Takes three arguments: 1. Node number to ping 2. Number of packets to send 3. Duration between pings (in seconds).
- **duration:** The duration until next “action” (in seconds).

An “action” must include at least the “msg” instruction. Command and duration (defaults to 0) are optional.

NOTE: It is important that “start_spyd” is executed at the beginning of the test if signal levels of packets are to be logged. Also make sure a scenario file ends with a newline character, else the scenario reading script may have problems with running grep on the file.

8.1.3 Scripting connectivity with mackill

It is possible to create scenarios for stationary nodes, where mackill is used to simulate different connectivity configurations. Nodes can then be positioned close to each other where radio-connectivity is guaranteed, and the scenario can be played without any movement. This can be extremely helpful when testing the routing logic of a protocol implementation.

During the test initialization phase, nodes must build an IP to MAC translation table, so that they will recognize the hardware addresses to block during the test. This is done by putting the following instruction in the scenario file:

```
choreography.startup.command.1=generate_ip2mac
```

Nodes can then be blocked with an instruction like:

```
node.0.action.2.msg=Blocking nodes 1 and 2
node.0.action.2.command=mac_disable 1 2
```

And later enabled again with the following instruction:

```
node.0.action.5.msg=Enabling nodes 1 and 2
node.0.action.5.command=mac_enable 1 2
```

8.2 Configuration of “collect node”

When performing testruns, one computer should be configured as a “collect node”. This machine will act as a central repository for log files created during tests. The “collect node” should be running a wireless interface in ad hoc mode and should be accessible by the other nodes (i.e. same network configuration). Uploading is done using rcp (remote copy) or scp (secure copy) to a predefined user account on the “collect node”. Here are the necessary steps to configure the “collect node”:

1. Configure the wireless interface so that it matches the collect node setup in the **ape.conf** file and the setup in `dist/configuration/etc/pcmcia/wireless.opts`. Make sure IP-address does not clash with those used by the test nodes, defined in the scenario files.

2. Make sure a rsh server (or ssh server) is running. NOTE: Uploading with scp require password authentication, unless it is configured to use key-based authentication.

3. Create the upload account in accordance to the configuration in **ape.conf**.

4. Create a `.rhosts` file in the home directory of the upload account and add one line for each node participating in the test in the form “<ip> root”, i.e. each line contains an ip-address of a node and then the username which is “root”. The ip-address and username is separated by a whitespace. The `.rhosts` file enables password-less access to the upload machine, so if the “collect node” is a production machine, do not forget to secure it after the test is finished.

   A script called `mk_rhosts.sh` in `src/tools/` is available to simplify the creation of the `.rhosts` file. Simply pipe a scenario file into this script and redirect the output to `.rhosts`.

5. Create the upload directory on the upload account as specified in **ape.conf**.

The `upload.sh` script on the test nodes can now be used to upload the log files.

8.3 Collecting log files

Our upload system can upload log files from several testruns at once. Log files from different nodes are then synchronized using timestamps, and log files from the same test are sorted into the same directory. However, this feature is experimental and not thoroughly tested. It is therefore recommended to upload the log files after each testrun and sort them into different directories manually.

9 Post testrun analysis

This section will briefly explain what can be done with the data collected from testruns, using the APE analysis tools.

9.1 Unpacking and synchronizing log files

When all files have been uploaded there should be one .a and one .z file and a number of log files from each node. The .a and .z file are for roughly synchronizing the different timestamps from the clocks on respective laptops. This enables the different log files to be sorted according their creation time and therefore easily separated into different testruns. Just run `src/analysis/utilities/timesync.sh` in the directory where the uploaded files are. Even if this sorting method is not used the files have to be synchronized in order for them to get the right names.

When the synchronizing is done, `src/analysis/utilities/unpack.sh` can be used to unpack the logs in different directories. The directories will be named with the scenario title and time of when the testrun was started. Newly created directories will contain compressed log files. Before starting the analysis, the log files must be decompressed with the `src/analysis/utilities/decompress.pl` script, by executing it in the directory with the compressed log files.

9.2 Analysis scripts

The analysis scripts in `src/analysis` are described in this section. For more information about how to run the analysis, view the documentation in the directory where the scripts are located. It is recommended that the analysis scripts are installed on a fast computer with lots of memory so that it can handle the large amount of data that may have to processed. Depending on the maturity of the different analysis scripts, not all scripts listed here may be available in the APE release you are using.

9.2.1 Managing log files

Tsb.pm - Perl module for doing time synchronization of logs.

mk_apeview_edge.pl - Make log for driving APE-view visualization tool.

mk_eth_log.pl - Combine superspy (ethernet) log files into one file.

mk_ip_log.pl - Combine ping logs into one file.

mk_step.pl - Step-divide (average SNR over interval) combined ethernet log file.

9.2.2 vM-related

vm_diff.pl - Calculates differences between adjacent intervals in a step-divided log file.

vm_multihop.pl - Calculates multi-hop virtual distances.

vm_network.pl - Averages nodes vM to a network vM value per step.

vm_node.pl - Calculates vM for each node in a step-divided log file.

vm_plot.pl - Plots vM using gnuplot.

vm_plotstats.pl - Plots statistical vM using gnuplot.

vm_stats.pl - Calculates statistical vM (vM-low, vM-high and vM-average).

vm_truncate.pl - Truncates log files to a specified interval.

vm-3.sh - Wrapper script for running vM calculation.

vm-simple.sh - Wrapper performing all necessary steps for a complete vM analysis.

analysis-gui - Graphical user interface for doing analysis on APE logs.

9.2.3 Ethernet level

eth_connectivity.pl - Average number of neighbors per node.

eth_link_change.pl - Number of changes in “connectivity” per interval.

eth_route_change.pl - Still experimental...

9.2.4 IP-level

ip_hop_count.pl - Calculates distribution of path-lengths for ping packets.

ip_path_optimality.pl - Examines how optimal paths taken are.

ip_pingsuccess.pl - Calculates ping success ratio for a test.

ip_pingsuccess_plot.pl - Plots ping success ratio.

ip_delayplot.pl - Creates a graph showing distribution of RTT from Ping.

10 Additional resources of information

- APE webpage `http://apetestbed.sourceforge.net`
- APE support list: apetestbed-support@lists.sourceforge.net
- The Loopback Root Filesystem HOWTO at `http://www.linuxdoc.org`
- The Linux “initrd” and “losetup” man pages.
- AODV-UU webpage `http://www.docs.uu.se/docs/research/projects/scanet/aodv/`