$\boxed{\mathcal{AIPS}\text{ Memo 99}}$

Multiple TV Servers on a Single Host

Patrick P. Murphy
National Radio Astronomy Observatory
Charlottesville, Virginia, USA

May 29, 1998

Abstract

Since the network-related overhaul of $\mathcal{AIPS}$ in 1992, it has not been possible to have more than one instance of the "TV" (image display) server or its ancillary servers on a single host. This has proven to be an inconvenience in many environments, and debilitating for $\mathcal{AIPS}$ users where X terminals or their equivalent are widely deployed. With recent changes in the `15OCT98` version of $\mathcal{AIPS}$, this restriction has now been lifted. This memo describes how the use of Unix domain sockets enables one to have multiple instances of the TV and other servers on a single host, displaying either to a single X11 display or multiple displays.

1 Background

When the first $\mathcal{AIPS}$ screen servers were written to take advantage of workstations and their high resolution screens, there were at least two modes of communication between the $\mathcal{AIPS}$ tasks and the server. One of these involved the use of Unix domain sockets, the other used INET or Internet domain sockets. As the latter could be used easily to take advantage of networks where the screen server and `AIPS` task were on different hosts, INET sockets were the logical choice for use in the network overhaul of the $\mathcal{AIPS}$ infrastructure which occured in 1992. Figure 1 illustrates the relationship between local and remote `AIPS` sessions and the INET version of the servers.

Unix domain sockets had been used before this in $\mathcal{AIPS}$, often on a routine basis. As they preclude the network transparency that was a goal of the overhaul, their use was deprecated. The use of INET sockets was built into the newer shell scripts surrounding $\mathcal{AIPS}$ and its tasks, even to the extent of attempting automatic detection of remote logins and subsequent execution of a remote shell back to the user's original host to start the TV and other servers locally. One negative side effect of this reliance on INET sockets was caused by their nature: one can only open one INET socket on a given port number on a single host at a time. Therefore it was only possible to run one instance of each server (*e.g.*, `XAS`) on a given computer at any one time.

However, the ability of the low level routines within $\mathcal{AIPS}$ to use Unix domain sockets was retained, and were added to the newer servers (`MSGSRV`, `TEKSRV`, and `TVSERV`). As these sockets manifest themselves as special files on local disk, there is no immediate restriction on their number other than system resources used by the tasks using these sockets. The shell script infrastructure of $\mathcal{AIPS}$ has now been modified to take advantage of Unix domain sockets without compromising or affecting in any way the existing functionality of the INET socket based scheme already in use.

2 Usage

From an end-user's viewpoint, multiple "TV"s are enabled by a slight change in use of one of the existing arguments to the `AIPS` command. If one now starts it up with:

```
aips tv=local
```

then the TV and other servers will start up *on the local host* using Unix based sockets. The keyword "local" is meant to remind the user that this option is meant for running things on the same host as the `AIPS` session. The servers will be started as `XAS1`, `MSSRV1`, `TKSRV1`, and `TVSRV1` respectively; the last one being the TV lock dæmon for `XAS1`. Any subsequent `AIPS` session started from this same host *with the same* `tv=local` *argument* will connect to this same set of servers for image display, *etc*.

These servers are *completely independent* of the traditional INET based servers. They have different names (`XAS` *vs.* `XAS1`, `TKSRV1` *vs.* `TEKSRV.EXE`, *etc.*) as well as using different types of sockets. There is no interaction between INET and UNIX ("`local`") servers and both can in fact be running — independently — on the same host at the same time.

If the user wishes to start another `AIPS` session on the same host, this time with a second instance of the TV and other servers, the syntax of the command is then:

```
aips tv=local:0
```

or one can specify the specific "slot":

```
aips tv=local:2
```

This will start up `XAS2`, `MSSRV2`, `TKSRV2`, and `TVSRV2` alongside the four servers associated with "slot" 1 which have already been started. Repeating this command (presumably in other windows) will generate successive new instances of the TV and associated servers. In the event that a user has, *e.g.*, three TV's currently running and wishes to connect a fourth session to the second TV, the command is:

```
aips tv=local:2
```

This would start an `AIPS` session that used the servers for "slot" 2, but as the servers were already running, it would not need to start up any instances of `XAS`, *etc*.

A total of 35 individual Unix-domain socket TV's may be started on a single host, provided that host has sufficient resources to accomodate them. Figure 2 shows the relationships between the various servers and `AIPS` sessions when two instances of the servers are present and being used by three different `AIPS` sessions. Note that the remote `AIPS` session now has no access to the local TV servers (though if an INET based TV is also running on the local host, the remote servers can still access it as they also can the `TPMON` dæmons).

3 Different X Displays

In a typical X-terminal environment, all that is necessary for the system to work is to have the DISPLAY environment variable pre-defined to whatever is the correct value for that system. Use of the `TV=LOCAL` option will then either start a set of Unix-domain TV servers, or use an existing running set if the `DISPLAY` basename (whatever is on the left side of the colon, *e.g.*, `orangutan` for `orangutan:0.0`) matches.

Currently the only problem with this approach is with utilities such as the secure shell (`ssh/slogin`) where the `DISPLAY` environment variable as set on the compute server does not contain the name of the X terminal,

Figure 1: AIPS Network Servers — INET Sockets

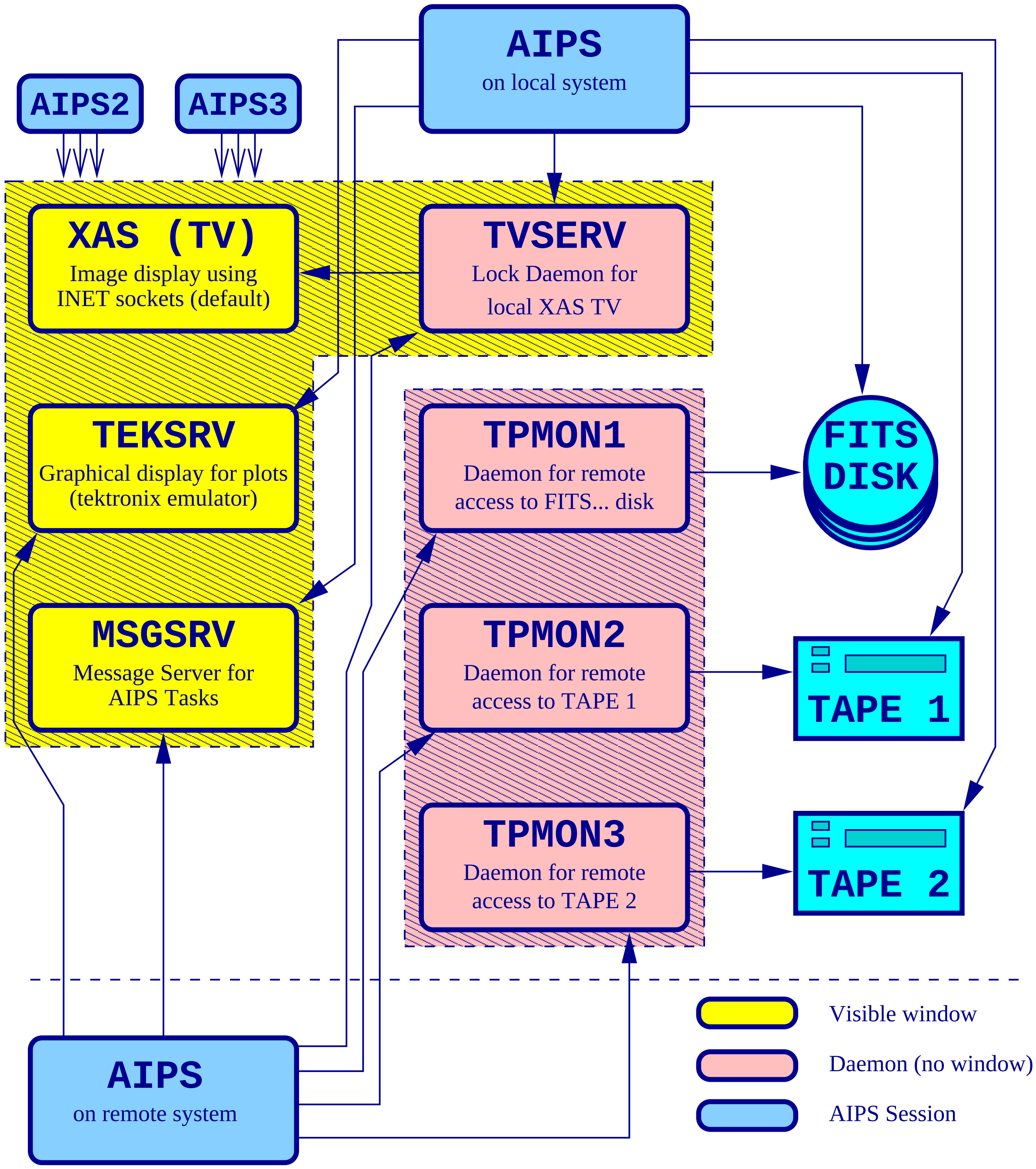
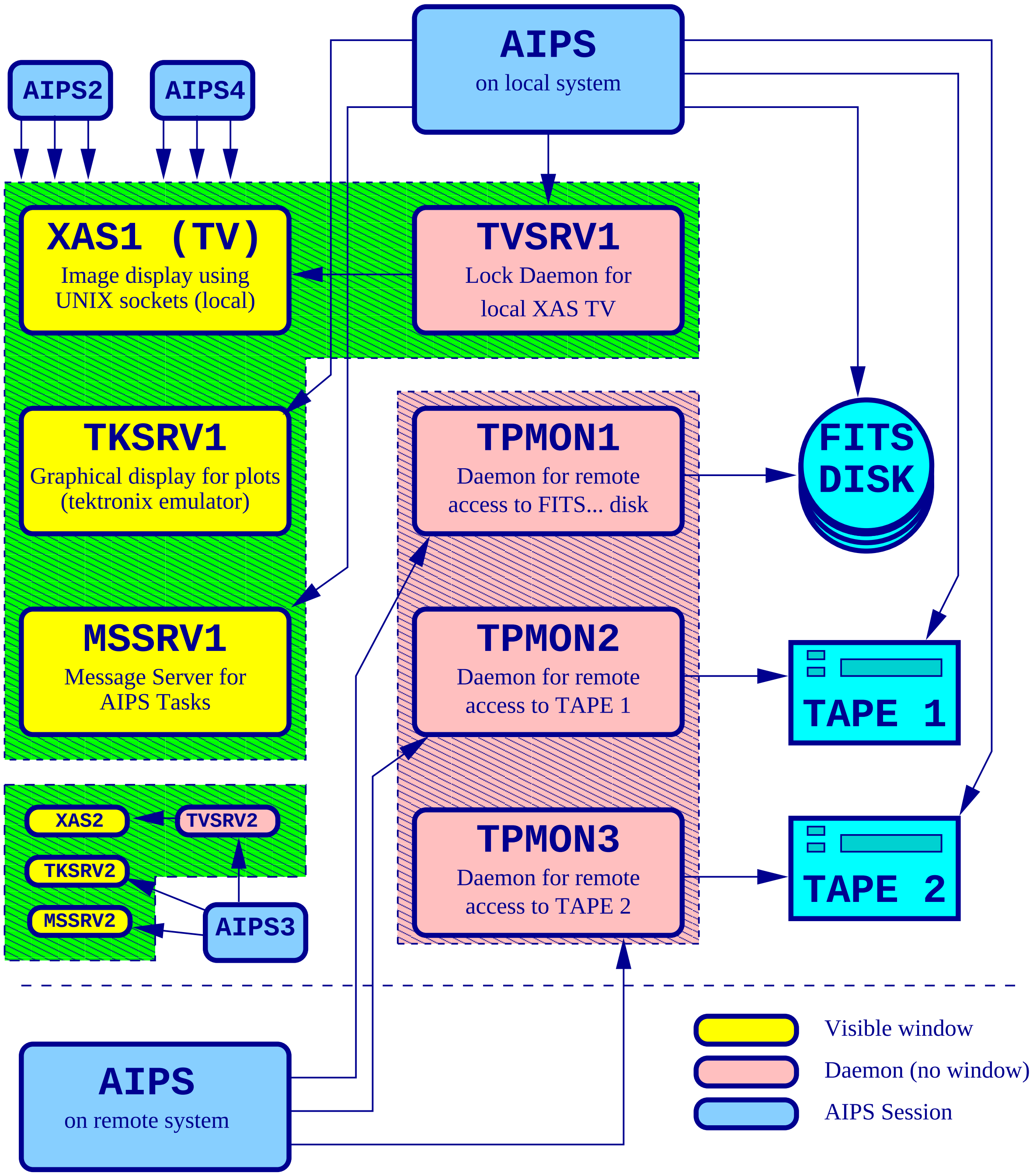


Figure 2: AIPS Network Servers — UNIX Sockets



PC or workstation on which the X server is running. In such cases, the *AIPS* scripts will only take whatever is on the left side of the colon when checking for already running servers. This may result in one user accidentally getting the TV that is already running and displaying on another user's screen.

For example, a secure login session from a system called `rebo` to a compute server called `zooty` will set the environment variable `DISPLAY` to something like `zooty:10.0`, instead of the more traditional `rebo:0.0` that one would manually set on a regular remote login session with `rlogin` or `telnet` (these last two do not set the variable automatically). There are at least two ways of providing a workaround for this; one is to manually set the variable to `rebo:0.0` before starting `AIPS`; the other is to configure the secure shell either globally or on a per-user and/or per-host basis to not forward or "tunnel" X11 through the secure encrypted connection. This is usually done in the `.ssh/config` file.

4 Unix or INET?

The choice of whether to use a "local" set of servers or the more traditional INET based servers depends on the needs of the astronomer. If it is important to have `AIPS` sessions from more than one computer all displaying to the same TV or Tek server, or perhaps to have the messages from tasks on many systems appear in the same message server, then the use of INET sockets is required. Conversely, if remote access is not relevant, and/or if multiple instances of the TV are necessary or desirable, then Unix sockets, *i.e.*, a "local" TV, is clearly indicated.

Theoretically, use of Unix sockets might result in a gain in performance over INET sockets for image-intensive operations such as `TVMOVIE` where a data cube is being displayed plane by plane in sequence. Initial tests reveal little difference in performance between the two, however; on a Pentium II dual processor system (Linux kernel version 2.0.32, 128 megabytes of memory) there was less than a 5% difference in favour of the Unix socket TV when running `TVMOVIE` on a $512 \times 512 \times 127$ channel data cube. Therefore the decision on which flavour of socket to use should be based on details of a given site's systems.

One detail that is vitally important for users and *AIPS* managers to realise is that an instance of an INET based set of servers can easily co-exist with one or more instances of Unix based servers on the same host. At one point in the testing phase, the author had one compute server running the INET based TV and other servers locally (displaying to his workstation) and at the same time running three sets of Unix based servers, also displaying on the same workstation. Use of a virtual window manager, and organisational skills on the part of the user, are obviously prerequisites for this sort of use. Plentiful system resources are also necessary, especially if the 24-bit mode of the `XAS` TV is used with the shared memory feature of the X server.

5 Implementation

The `HOSTS.LIST` file specifies the normal set of "TV Hosts", *i.e.*, systems capable of running *AIPS* and a set of TV servers. As *AIPS* itself relies on a TV "number" internally, the 35 potential local or Unix-domain-based TV servers for a given host are assigned numbers after these conventional INET-socket based TV's. So in an *AIPS* site with 25 workstations, on any given workstation the TV number assigned to the first Unix-domain TV will be 26. There is no conflict with TV 26 on another host, as these local TV's are purely local and cannot accept input from other hosts.

The `/tmp` directory, or the area pointed to by environment variable `$TMPDIR`, is used as a repository for the Unix socket files. Each of the four servers has its "device name" defined as a file within this directory. The first instance of `XAS` running to a display `rebo:0.0` will thus be `/tmp/XAS.1.rebo`; likewise the corresponding instance of the Tektronix server will use `/tmp/TKS.1.rebo` as its socket name. If such a local TV were the first in the previous example (TV number 26), the environment variable `TVDEV` would be set to `TVDEV0Q`, and `TVDEV0Q` would be set to `/tmp/XAS.1.rebo`.

A second user on the same host, using a different `DISPLAY` setting (*e.g.*, `zooty:0.0`), will be given slot 2 as the *AIPS* startup scripts will detect the presence of the `/tmp/xxx.1.*` socket files. There is a possibility of a collision if two people start up `AIPS` on the same system with different displays almost simultaneously, as the detection/creation of the socket files is far from atomic. In fact, the socket "files" do not appear until the servers themselves have started. It would be possible to perform some form of locking via use of hard links to prevent this from occuring; use of such a technique is being investigated and may be implemented before the release of the `15OCT98` version of *AIPS*.

Finally, it may be desirable to encode the whole of the string in the environment variable `DISPLAY` in the Unix socket name. For example, `/tmp/XAS.1.rebo:0.0` would be used in place of `/tmp/XAS.1.rebo`. This would have the advantage of avoiding the incorrect assumption the scripts currently make after discarding the display number to the right of the colon.

6 Conclusions

The desire to run multiple instances of the TV and ancillary servers on a single *AIPS* host has been stated over the years by a variety of users. The changes described herein are an attempt to address the need behind these requests, while leaving intact the current infrastructure and having no impact on the mode of operation to which *AIPS* users are currently accustomed.