Appendix A

The Network Probe Daemon

NPD (Network Probe Daemon) is a framework for probing paths through the Internet by tracing the routes corresponding to the paths, and by sending TCP packets along the paths and tracing the arrivals of both the packets and their acknowledgements. NPD consists of a daemon (`npd`) that services authenticated requests for tracing and generating probes, and a control program (`npd_control`), which is run only at the site conducting the probe experiments.

The following sections discuss the daemon's operation (§ A.1) and the steps taken to address security concerns (§ A.2).

A.1 Daemon operation

A site participates in the network probe experiment by running the network probe daemon `npd` on a Unix workstation connected to the Internet. The workstation does not need any special location in the network topology (e.g., it does not need to be located on the wide-area gateway network).

The `npd` process is run by Internet services daemon `inetd` whenever a connection appears for the "npd" service (TCP port 7504, by default). This means that installing the daemon requires editing */etc/services* to add the "npd" service, and */etc/inetd.conf* to add the service with the given port number.

Once running, `npd` responds to the following requests:

`trace-route` *X*

Run the `traceroute` utility [Jac89] to measure the path to host *X* and send back the results.

`begin-trace` *X Y*

Begin tracing "discard" or `npd`-to-`npd` packets and their acknowledgements between hosts *X* and *Y*.

`terminate-trace`

Stop the trace and send back the results.

`sink` *s*

Accept a connection on the "npd" port, using a socket receive buffer of *s* bytes, and read from it until the connection is closed.

`source` *X p n s*

Send *n* bytes to the discard or "npd" port (as indicated by *p* being "discard" or "npd") of host *X*, using a socket send buffer of *s* bytes.

`npd` sources and sinks always use a local TCP port of 7505 (that they both do has security benefits, as discussed in § A.2 below). If the bytes are sent to the "discard" port, then no remote `npd` need run; the `inetd` process on the remote machine will instead handle discarding the data packets itself.

`restart-log`

Mail the current log to a preconfigured address and, upon success, clear it.

`self-test`

Perform a self-test and report the results.

`quit` Terminate the connection.

On some operating systems, the packet filter cannot capture traffic generated by the same host that is running the filter. In particular, Sun workstations using SunOS and the stock "NIT" (Network Interface Tap) interface do not capture their own outbound traffic. Because SunOS is quite popular, it was necessary to accommodate this deficiency. For the `traceroute` experiment it makes no difference, but for the packet dynamics (*probe*) experiment it is crucial that the TCP traffic comprising the probe be recorded at both endpoints. NPD can thus be configured at a site to run on two workstations, a *source/sink* host that sources or sinks TCP probes, and a *trace* host that runs `traceroute` or `tcpdump`, depending on the experiment. For a given site A, we refer here to these machines as A_s (source) and A_t (trace) respectively. For many sites, $A_s = A_t$, as summarized in Table XIV.

To conduct a `traceroute` experiment measuring the route from site A to site B, the NPD master program (`npd_control`) connects to the `npd` daemon at host A_t and (after authentication) issues:

`trace-route` *B*

`quit`

and reads back the `traceroute` output, if successful. To conduct a *probe* experiment of b bytes between A and B, using send and receive buffer sizes of s and r, `npd_control` executes the following steps (assuming each preceding step is successful):

1. Send the request `begin-trace` A B to A_t and B_t, and wait for them to indicate they are ready.

2. Send the request `sink` r to B_s and wait for it to indicate it is ready.

3. Send the request `source` B `npd` b r to A_s.

4. Wait for A_s and B_s to indicate they have finished sourcing/sinking the data stream.
5. Wait two more seconds, to allow any packets still traveling inside the network to arrive at the endpoints.
6. Send the request `terminate-trace` to A_t and B_t.
7. Receive the trace and error files from A_t and B_t.
8. Send the request `quit` to A_s and B_s, and to A_t and B_t if different.

A.2 Security issues

Allowing a program to originate and trace network traffic at an Internet site naturally raises important security issues. To this end, we took a number of steps to make NPD secure:

- A host attempting to make NPD requests must first authenticate itself, as explained below.
- `npd` does not need to be installed with any privilege, other than being able to exec `tcpdump` and `traceroute`. A site can also configure it so it can only run a special, restricted version of `tcpdump` (`rtcpdump`; see below).
- `npd` is hardwired to only be able to trace TCP "discard" traffic, or traffic between two `npd`'s. This is done by constructing a `tcpdump` filter of

  ```
  (RESTRICTION) and (XXX)
  ```

  whenever `npd` is asked to trace traffic using the filter `XXX`, where `RESTRICTION` is:

  ```
  (tcp port 9) or (tcp src port 7505 and tcp dst port 7505)
  ```

  i.e., only allow traffic involving either the TCP `discard` port, or both an `npd` sender and receiver. (TCP port 7505 is the well-known port used by `npd` for sourcing and sinking traffic; see § A.1.)
- `npd` logs all of its connections and activity. If writing to the log fails, or if `npd` cannot lock the log for exclusive access, `npd` exits.
- The log file can only be reset if `npd` first succeeds in mailing the previous log to a preset Internet mail address. Sites can configure this address to include a local address.
- The only files created by `npd` (other than the log file) are temporary files created using the Unix `tmpfile(3)` library routine, which are guaranteed to disappear when `npd` exits, and also to be unreadable by other local processes.
- When executed, `npd` forks a child process that sleeps for a fixed amount of time (10 minutes). When the child process wakes up, it kills its parent process. This mechanism acts as a crude "fail-safe." Normally, after `npd` successfully completes its requests, it kills the child process prior to exiting itself. But if for any reason `npd` fails to do so (for example, if the network connection between `npd` and `npd_control` is lost), the fail-safe guarantees that `npd` will at some point cease consuming resources on the host.

A.2.1 Using `rtcpdump` instead of `tcpdump`

The NPD sources include `rtcpdump`, a version of `tcpdump` that is restricted to capturing TCP discard packets (or `npd`-to-`npd` packets, as described above). `rtcpdump` can only capture live, restricted packets (it cannot read existing trace files), and only writes to *stdout*, which is under the full control of `npd`.

Thus, a site can safely give `rtcpdump` "setgid" or "setuid" privilege to the Unix "group id" or "user id" necessary for packet capture on the tracing host, without needing to give the tracing group-id or user-id to `npd` itself.

`rtcpdump` terminates whenever its *stdin* is closed, which happens automatically when `npd` exits.

A.2.2 NPD authentication

An important aspect of NPD security is the use of fairly strong authentication to restrict use of `npd` at a site to only authorized remote sites. `npd` authenticates a remote site in the following manner:

1. The IP address of the remote host must translate to a hostname that in turn translates back to the given IP address. To illicitly pass this test, an attacker must subvert a Domain Name System (DNS) name server [MD88] (which, unfortunately, is possible [Be95]).

2. As part of the authentication procedure, the host must identify itself using a DNS hostname. The host's claimed identity must then translate to the host's IP address. Like the previous step, this step requires that an attacker subvert a DNS name server.

3. The host's claimed identity must appear in `npd`'s directory of secret keys. For an attacker to pass this test, they must successfully subvert a DNS name server authoritative for one of sites appearing in the directory of secret keys; more difficult than the subversions above, but still possible.

4. `npd` challenges the remote host to prove its identity by sending it a random bit-string. The remote site must successfully xor this bit-string with the secret key and send to `npd` the MD5 checksum [Ri92] of the result. `npd` then verifies that the result matches its own local computation of what the checksum should be. If so, then the remote site is presumed to know the secret key and is authenticated.

   For an attacker to successfully pass this test essentially requires that they know the secret key, since MD5 checksums take on $2^{128} \approx 10^{38}$ possible values. Since the secret key never crosses the network,[1] to acquire the secret key requires either subverting the `npd_control` site or the `npd` site, or computing the key by observing previous authentication exchanges as they crossed the network. This latter attack is believed infeasible due to the presumed non-invertibility of MD5 [Ri92].

[1] Except when distributing the NPD sources to a remote site; or if `npd` retrieves the key using NFS.