Using *rq2proxy*

Richard Watts
University of Cambridge Computer Laboratory
`Richard.Watts@cl.cam.ac.uk`

December 22, 1998

Abstract

This document explains how to configure and use rq2proxy to proxy Quake II games through a firewall. The latest version of *rq2proxy* is always available from `http://epona.ucam.org/rrw/qproxy.html`.

Contents

Chapter 1

Introduction

rq2proxy is a UDP-over-TCP tunnelling proxy suitable for proxying Quake II packets. It consists of two components:

- A client, *rq2pc*, which sits behind a firewall, accepting packets from a TCP connection and sending them to Quake II client machines. It also accepts packets from Quake II clients, and sends them down the TCP connection.
- A server, *rq2ps*, which sits on the outside of a firewall, accepting packets from Quake II servers and passing them to *rq2pc* via TCP, and accepts packets from the TCP connection and passes them to the Quake II servers.

Neither component requires privelege to run. Both can run as any user, though they take advantage of memory locking (using `mlockall()`) and POSIX real-time scheduling if they are available.

rq2proxy has the following features:

Many-to-many proxying —multiple clients can play on multiple servers through the proxy.

Per-origin routing —you can route different players to different servers based on the machines they are using, as well as the machine they are connecting to.

Programmed delay —you can create programmed delays for paths through the proxy, so different players can have different pings and you can equalise LAN players' pings with those of modem players.

Works with xqf —the proxy works fine with game selectors like `xqf` and `gamespy`. I mention this because the previous version didn't...

rq2proxy can be used for a number of tasks, including:

- Tunneling through firewalls to avoid fascist packet filtering restrictions or provide controlled access to servers. Administrators within JANET may want to use the proxy to allow access only to UK Quake II servers without having to write router rules.
- Providing control over your ping: LAN players can now play with more or less any ping they like.
- To provide 'best-of' services: players can connect to a single address and be proxied to today's CTF/DM server (but see the performance notes).
- To provide connection monitoring and packet accounting.

Note that this proxy is only tested with Quake II. Stock Quake I probably will not work, as its networking code is rather more primitive: I have no idea about QuakeWorld, Hexen II or any other game—if you've tried them and found them to work/not work/destroy small south american states, please drop me a line.

1.1 Support

rq2proxy is currently supported by `Richard.Watts@cl.cam.ac.uk`. It has a home page at `http://epona.ucam.org/rrw/qproxy.html`, from which the latest version is always available.

There is also an electronic mailing list. To subscribe, send a human-readable e-mail to `Richard.Watts@cl.cam.ac.uk` with the phrase 'Subscribe rq2proxy' in the subject line.

Chapter 2

Building and installing *rq2proxy*

rq2proxy uses GNU autoconf, so you should just be able to do

```
$ ./configure
$ make
$ make docs
$ make install
```

For more details about how to install the program, see the `INSTALL` file in the distribution directory and the output of `./configure --help`.

The `make docs` stage simply rebuilds the documentation: it shouldn't be necessary, so you can omit it if you don't have LaTeX, `dvips` and `latex2html`.

If that doesn't work, try doing:

```
$ ./configure
$ make lexclean
$ make
$ make docs
$ make install
```

To kill off the stale bison and flex output files.

You may need to use GNU `make` to build the system on some platforms (eg. OSF/1), since the native make doesn't understand how to regenerate the dependency files.

You can remove build files by doing `make clean`, and attempt to return the distribution to a pristine state by doing `make distclean`.

If `configure` can't find `yacc` or `lex`, don't despair—pre-built copies of `y.tab.c` and `lex.yy.c` are provided. This does, however, mean that you won't be able to change the parser and scanner and recompile. Free replacements for `yacc` and `lex` are available as `bison` and `flex`, respectively (these are the programs *rq2proxy* was developed with), available from the GNU project. Likewise, if `libfl` is not available (eg. because flex is not installed on your machine), you will have to remove `lex.yy.c` and have your copy of lex rebuild it.

Note that this program probably will not work on 64-bit systems such as the DEC Alpha, where `sizeof(unsigned long) == sizeof(long) == sizeof(int) == 4`. In the next version...

If you want to mess about with the internals of the program, take a look at `config.h.in`—it contains a few defines you can tweak to use different routing policies. Note that tweaking most of these will result in variously nasty effects—see §7.

2.1 Known builds

rq2proxy is known to build successfully on:

- *Linux 2.0.34* and *libc5.4.44*—development system. Works fine.
- *Linux 2.1.127* and *glibc2*.
- *Solaris 2.4* with *gcc*—Solaris has no `vsnprintf()`. You need to pass `--enable-vsprintf` to `configure` and accept that there will probably be some logging buffer overflows.
- *OSF/1 v3.2* with *gcc* and *gnumake*, but it won't run because of incompatible type sizes. Needs `--enable-vsprintf`.
- *Irix 6.4* with *gcc* and *gnumake*. Needs `--enable-vsprintf`.

Please email me if you manage to build *rq2proxy* on any other machine, especially if you had to do something out of the ordinary to make it happen.

Chapter 3

Quick start

Suppose you have a machine, P, inside a firewall, which you want to proxy to an external Quake II server, Q. Further suppose you have a machine inside the firewall, A (A can be the same as P), and a machine outside the firewall, B (B can be the same as Q).

To proxy P to Q, first write a configuration file on B:

```
cookie Hello_World
listen 17000
client allow A

# Access control
global allow P

# A single redirect
redirect A port 12000 to Q port 27910

# S'all.
```

Call it `myconfig.conf`. Now start the server on B:

```
B$ rq2ps -f my(20416)
pid 20416 created socket 17000. Waiting for client to connect...
```

And start the client on A:

```
A$ rq2pc B 17000 Hello_World
Attempting to connect to B (XXXXXXXX), port 17000...
Connection successful. Authenticating and downloading ACLs and proxy options..
Connection successful. Creating listening sockets ...
Opened file descriptor 4 to proxy port 12000, client YYYYYYYY
(to server ZZZZZZZZ, 27910)
```

Now, just start Quake II on P and issue:

```
] connect A:12000
```

... and that's it. You can also use game selectors like `gamespy` or `xqf`.

The `Hello_World` string above is a cookie used to provide some additional level of security: the client has to present a cookie identical to the cookie in the server's configuration file before it will be allowed to connect to the server.

Don't worry about any `shutting down rogue file descriptor` messages you may see after using `xqf` or `gamespy`: they simply indicate that the client port (in this case, the port the program opened momentarily to check the server) has gone away unexpectedly, and are harmless (though too many of them will cause performance problems).

Chapter 4

Command line options

Most command line options are common to both client (`rq2pc`) and server (`rq2ps`). A summary is included in the manual pages supplied, but here's a reference:

Option	`rq2pc` ?	`rq2ps` ?	Description
`--log-all-packets`	Y	Y	Log all packets.
`--help`	Y	Y	Show brief help.
`--no-mlock`	Y	Y	Don't attempt to lock pages in memory.
`--no-rt`	Y	Y	Don't attempt to use POSIX real time scheduling.
`--use-mlock`	Y	Y	Attempt to lock pages in memory.
`--use-rt`	Y	Y	Attempt to use POSIX RT scheduling.
`--version`	Y	Y	Report version number and exit.
`--do-fork`	N	Y	fork() on each client connecton.
`--from-inetd`	N	Y	Run from inetd. Implies `--no-fork`.
`--no-fork`	N	Y	Don't fork() on each client connecton.
`--test-only`	N	Y	Just print out the config file and exit.
`-f filename`	N	Y	Read configuration from this file.
`-p portname`	N	Y	Listen on this port.

By default, the proxy will try to lock pages in memory and set real-time scheduling on startup: this reduces proxy latency, but ties up roughly 1Mbyte/program of real memory and may interrupt other time-critical tasks, and is only available to proxies running as super-user on many operating systems (eg. Linux).

By default the server runs just once: it accepts a single client connection, and then dies. If you prefer, you can use `--do-fork`, which causes the server to fork off another process for each client connecting: this allows for multiple clients per server, but uses more memory and makes debugging difficult.

Alternatively, you can run the server from inetd by putting something like the following in your `/etc/inetd.conf`:

```
17003 stream tcp nowait root /usr/sbin/tcpd /homes/rrw/q2/server
   -f /homes/rrw/q2/example.conf --from-inetd
```

Chapter 5

Configuration file format

The configuration file specifies what the server should proxy. It consists of a sequence of statements, which can be of four forms:

1. Redirect commands (beginning with `redirect`). These specify redirections: redirections are processed in order of declaration.
2. Access control commands (beginning with `global`). These specify access control list elements: ACL elements are processed in order of declaration.
3. Miscellaneous commands: these simply set various characteristics of the server, and the last such command fixes the characteristic, unless overridden by a command-line option.
4. Comments: introduced by `#` and last until the end of the line.

Options supplied by the configuration file override built-in defaults and are overridden by command-line options.

Whenever an IP address (*NOT* a netmask) is specified, a hostname can be used instead. Beware, however: DNS cache poisoning attacks can cause you to trust people you don't think you're trusting, and all names are looked up at the server, so if your internal names are invisible from outside your firewall you will need to use IP numbers (and indeed, you're probably better off doing this anyway).

If you want to know exactly what is permitted under certain circumstances, you are referred to the `parser.y` and `lexer.l` files in the source distribution.

5.1 Access control commands

Access control commands govern which machines are allowed to send packets to the client side of the proxy (`rq2pc`): the server side of the proxy will ignore any packets not sent by the right server.

Access control commands consist of the keyword `global` followed by an access control specification of the form:

```
[allow|deny] address </netmask>
  <(<from portfrom> <-|to> <portto>|port [port])>
```

The `[allow|deny]` indicates whether hosts matching the succeeding address specification are to be allowed or denied access. A packet from host `A`, sent from port `p` matches an ACL command if `(address)&netmask == (A&netmask) && (p>=portfrom && p<=portto)`.

If not given, the netmask is taken to be `255.255.255.255`, `portfrom` is 0 and `portto` is 65535. If `port [port]` is specified, `portfrom = portto = port`.

Examples of ACL commands:

```
allow 192.168.1.1 port 23-45
allow 192.168.1.2/255.255.255.240 from 23-
allow 192.168.1.3
deny 192.168.1.4 from - 45
allow foo.bar.baz
```

ACL commands are processed in the order given in the configuration file: the result of the first matching entry (allow or deny) is the result of the match - hence, in a configuration file containing:

```
allow 192.168.1.1
deny 192.168.1.2/255.255.255.128
allow 192.168.1.24
```

`192.168.1.1` would be allowed, and `192.168.1.2` and `192.168.1.24` denied.

If no rules match, the packet is rejected.

ACL processing occurs before redirect matching, so ACLs associated with redirects which specify hosts that don't match the access control commands are pointless and might as well not exist.

5.2 Redirect commands

Redirect commands are of the form:

```
[on <address>] redirect <address> port <port> to
   <address> <port> [acl] [delay <num>(s|us|ms)<+<num>(s|us|ms)>]
```

Where each address is either a host name or an IP number, the ports are port numbers, and the `acl` is an ACL expression of the same form as one following a `global` command (ie. `allow` or `deny` followed by an address specification).

A preceeding `on <address>` should be used for multi-homed hosts: clients identify which redirects pertain to them by comparing the redirect address with their host address, as obtained by `getaddrbyname(gethostname())` and by examining the `dest_address` of the IP packet carrying data from server to client. Unfortunately, firewalls and other multi-homed hosts typically have different names on the internal and external networks, so one might easily specify a redirect for which the client connects, looks at the redirect address, discovers that it isn't an address for the default name of this host, and so drops it, despite having a secondary interface bound to the redirect address. You can avoid this problem by putting the default name of the client that should handle a redirect in the address field of an `on <address>` clause preceeding the redirect.

A delay clause indicates that packets should be delayed at the proxy to give an artifical high ping. The delay specified is additional to the inherent delay of the proxy (see §7). Either one or two components may be specified: if two components are specified, the first is the outgoing delay (from Quake client to Quake server), and the second is the incoming delay (from Quake server to Quake client). Hence, the delay clause below indicates that packets should be delayed by 100ms from client to server, and by 50 from server to client. Thus you can simulate the effects of asymmetric routing.

If only a single delay value is specified, it is divided by two, and the outgoing and incoming delay set to this value (since the pings measured by Quake II are round-trip times).

For example, a redirect of the form:

```
on 1.2.3.4 redirect 5.6.7.8 port 12000 to 9.10.11.12 port 14000 allow
 13.14.15.16/255.255.0.0 from 13000-18000 delay 100+50
```

Indicates that a client whose primary hostname is bound to the address `1.2.3.4` should accept packets destined for `5.6.7.8` (which is presumably bound to another interface), on port 12000, and proxy them to the Quake II server `9.10.11.12`, port 14000, but only if the machine originating the packets matches the hostname/netmask pair `13.14.15.16/255.255.0.0`, and the originating port is between 13000 and 18000 inclusive. All packets proxied will be given an induced delay of 150ms (100ms outgoing, 50ms back), in addition to the delay inherent in the proxy, which will typically result in a ping of about 200.

As with ACL commands, redirect commands are processed in the order given, and the first match is used. A redirect with an ACL which doesn't permit connection by a given Quake II client is treated as if it didn't exist, and the search continues.

If no redirect is found for a packet, the packet is silently dropped.

5.3 Miscellaneous commands

There are a number of miscellaneous commands:

5.3.1 `set`

The `set` command toggles various options. Its syntax is:

```
set [option name] (on|off)
```

Options are roughly equivalent to command-line switches. Available options are:

Option	Description
`log_all_packets`	When set, logs all packets to stdout.
`use_mlock`	Attempt to lock pages in memory on startup.
`use_rt`	Attempt to use POSIX real-time scheduling.
`do_fork`	Fork a new server for each client.
`from_inetd`	Run as if from inetd (implies `do_fork no`).
`test`	Don't actually run: just print the config file and stop.
`verbose`	Be verbose.

5.3.2 `client`

The `client` command adds an entry to the set of machine/port pairs that are allowed to be clients for this server. The syntax is:

```
client [acl]
```

eg.

```
client allow 192.168.1.1/255.255.0.0
```

Where `[acl]` is an ACL entry as described above, beginning with `allow` or `deny`. As for the global ACL list, this list is processed in order of declaration.

5.3.3 `cookie`

The `cookie` command specifies the cookie the client must present in order to demonstrate its authenticity. The cookie may be any text string (but if it contains spaces, it must be wrapped in double quotes). The syntax is:

```
cookie [cookie]
```

eg.

```
cookie Hello_World_foo_bar_24624
```

5.3.4 `listen`

The last `listen` declaration in a configuration file sets the port on which the server will listen for connections (and is ignored if `from_inetd` is set). The syntax is:

```
listen [portno]
```

eg.

```
listen 17003
```

5.4 Comments

Are introduced by `#` and last until the end of the line.

5.5 An example configuration file

Here's an example configuration file which shows off some of the more complex features of the file format:

```
# Cookies'n'stuff.
cookie Hello_World
listen 17003
client allow stkitts.cl.cam.ac.uk
set do_fork on

#Acls
global allow 128.232.0.0/255.255.240.0 from 1200 - 1600
global  allow 128.232.0.0 netmask 255.255.240.0
global allow epona.ucam.org
global allow maui.al.cl.cam.ac.uk
global allow maui.nt.cl.cam.ac.uk
global allow uist.cl.cam.ac.uk


# Redirects
#redirect stkitts.cl.cam.ac.uk port 12000 to 131.111.129.173
#    port 27910 deny maui.nt.cl.cam.ac.uk
redirect stkitts.cl.cam.ac.uk port 12004 to
   quake-1.games.group.cam.ac.uk port 27910 allow uist.cl.cam.ac.uk delay 90
#redirect stkitts.cl.cam.ac.uk port 14000 to
   quake-1.games.group.cam.ac.uk port 27910
on epona.ucam.org redirect stkitts.cl.cam.ac.uk port 12004 to
  131.111.129.174 port 27910
redirect stkitts.cl.cam.ac.uk port 12000 to 131.111.129.173 port 27910
redirect stkitts.cl.cam.ac.uk port 12001 to 131.111.129.173 port 27910
 delay 210
redirect stkitts.cl.cam.ac.uk port 12002 to 131.111.129.173 port 27910
 delay 90+90
redirect stkitts.cl.cam.ac.uk port 12003 to 131.111.129.173 port 27910
 delay 90
redirect stkitts.cl.cam.ac.uk port 14000 to quake2.demon.co.uk port 27910

redirect stkitts.cl.cam.ac.uk port 16000 to quake2.demon.co.uk port 27910
    deny maui.al.cl.cam.ac.uk/255.255.240.0 from 0 - 1024
```

Chapter 6

How it works

The server, *rq2ps*, contains most of the intelligence: the client is more or less a dumb proxy. The server keeps track of temporary associations between Quake II client machine,port pairs, file descriptors, *rq2ps* port numbers and Quake II server machine, port pairs via. a pair of routing hash-tables.

On startup, the server reads its configuration file, including the options set on the command line. If not running from inetd, it then opens a server socket and listens for connections. If running from inetd, this chapter is skipped, and a synthetic accepted connection is synthesised from fd 0 (which is how inetd passes us the socket we're connected to) (`server.c`).

The server is responsible for resolving names into IP addresses.

When a client connects, the first thing it does is to send the cookie (in clear). It then reads the global access control list and routing tables from the server (`ft.c`).

The client then discards any routing entries which don't pertain to it (ie. which don't have one of the IP numbers associated with its primary name or the IP number the packet came in on as their `on` parameter (or redirect-from IP if they don't have an `on` parameter)), and builds a file descriptor for each port it's required to proxy, putting them in a hash table keyed by (client name, port) and back-annotates the routing table to contain fds (`client.c`).

Both sides now go into proxying mode. Each does a `select()` on all relevant fds, then loops over them, proxying for each that is ready for reading, exchanging encapsulated UDP packets.

When the client recieves a packet from a client, it does a linear search of the global ACL, then hashes the incoming file descriptor to find the routing table entry it needs to get `[clnt ip]` and `[clnt port]` (yes, this is necessary - the target address of the packet might not be the same IP address as specified in the routing entry). The client sends encapsulations of the form:

```
[length] [orig ip] [clnt ip] [clnt port] [orig port] [data]
```

All the above (except for `[data]`) are 4-byte words, and `[length]` is in bytes, counting everything except itself (so the actual length of the data packet is `[length]-16`).

The server looks up a route in its routing table, using the quadruple (orig ip, orig port, clnt ip, clnt port) as a key. If it doesn't find an existing route, it creates one, opening a file descriptor to send UDP packets to the Quake II server (it uses linear search through the routing table to find out who to proxy to) (`server.c`, `route.c`). It then attaches the packet to the incoming packet queue and gives it a time to send.

When the server finds a packet to proxy back to the client, it looks up a routing entry using the triple (QII server ip, QII server port, file descriptor) as a hash key into a second table. It then sends an encapsulation of the same type (including `[orig ip] and [orig port]`, just as the client sent them: it knows them because it squirreled them away in its routing table) (`server.c`, `route.h`, `route.c`). It then stores this packet (ready-encapsulated) in the outgoing packet queue, with the appropriate time-to-send.

Each server route has one incoming and one outgoing packet buffer. If these buffers are already full when a packet comes in (or out), the convention is to throw away the old packet (you can change this by defining `INDUCED_LATENCY_DROP_POLICY` to `0` in `config.h.in`).

At various times in the loop, if a packet on the input or output queue is ready to send, it is sent. This is a cheap test because the packet queues are held as pointer arrays sorted by increasing time to send. At the end of the loop over `select()`d fds, any remaining due packets are cleared to prevent them building up in the packet queue and causing confusion when they are eventually cleared.

When the client recieves a packet from `rq2ps`, it does a linear search of the relevant routing table entries to find who it should send the packet to (the routing table having been back-annotated with the appropriate fds earlier).

If the server detects that a connection hasn't been used for 10s (the value of `PRUNE_AFTER` in `config.h.in`), either with no packet sent from either direction if `DROP_CONN_ASYMMETRIC` is 0, or with one side having not sent a packet in `PRUNE_AFTER`s (though the other side may have) if `DROP_CONN_ASYMMETRIC` is `1`, the server shuts down the connection, returns the packet buffers and per-route data to a free pool for re-use and closes the file descriptor.

`DROP_CONN_ASYMMETRIC` should always be set to 1. This is because some servers refuse to acknowledge the destruction of a client, and carry on sending data until the port they're talking to closes. If `DROP_CONN_ASYMMETRIC` is 0, the port will never be closed and this situation will continue indefinitely.

Chapter 7

Performance Issues

7.1 Resource usage

rq2proxy doesn't handle significant amounts of data (typically 2k/player/s), so doesn't use significant amounts of CPU. If POSIX real-time scheduling is used, *rq2proxy* will use more CPU than with real-time scheduling disabled, but this is still no more than about 5% of the CPU time of a 486DX2/66.

More important is memory usage: to avoid double-free()ing bugs, *rq2proxy* doesn't free() nearly as much memory as it should do—all temporary buffers allocated at parse time, for example, are never freed. The result is that with memory locking on, each instance of either `rq2ps` or `rq2pc` uses about 1Mbyte of unswappable, real memory. This isn't nearly as nasty as it used to be, given the recent dramatic fall in memory prices, but it's much worse than it should be: the next version hopes to clean this up—if you email me saying you need this feature, it will be bumped up the priority queue.

7.2 Quake II latency

rq2proxy has been tweaked to provide good performance. With one player, you can get negligible packet lag. The author regularly uses the program to artifically bump up his ping to 150, and has observed no problems.

Preliminary experiments suggest that lag increases by c. 5-11ms for each additional player on the same server (players playing on different servers seem to be more independant of each other: increases of lag of more like 2-7ms have been observed). Unfortunately, the only systems I have to test this on are four rather old SGIs, which aren't really ideal (or sufficient in number), so if you're intending to proxy more than 6-8 games through one server, (i) tread carefully, and (ii) please write in and tell me how it went (or better stil, send me 16 PII/400s with 3Dfx cards and 100Base-T ethernet :-)).

The internal structure of the proxy is rather odd because of its need to get packets in and out with low latency. The Quake II server appears to have some kind of priority queue structure for emitting world packets, and the server and client can only proxy in some defined order. Unfortunately, since the server keeps shuffling its priority queue, this results in the nasty pathology that high-priority packets get trapped (or tossed) behind low-priority packets, which not only causes momentary freezing, but also seems to cause the server to do horrible things to its priority queue, resulting in higher average lag for all players. Again, if anyone notices anything odd (or if anyone at id is feeling helpful), please e-mail me.

Notably, Quake II seems to possess no rate adjustment, and no effective discard - it copes extremely badly with out-of-order packets (freezes and the network icon), so extreme measures have to be taken to avoid sending 'old' packets (ie. to avoid buffering). *rq2proxy* tries to avoid this problem by tossing packets at every opportunity.

Please feel free to play around with the source code (specifically, `config.h.in` contains a lot of useful `#define`s: remember to re-run `configure` after changing it), but note that an awful lot of apparently 'good' changes (like extra buffering) will simply result in a few seconds' play, until the packet stream gets segmented into 'back-packets' which consistently get stacked behind 'forward-packets', the client becomes terminally confused at receiving what are effectively alternate, interleaved histories, displays the network icon and crashes.

The distribution for version 1 of the proxy, which only supported proxying one client to one server, is provided in the `version1` directory: it's not very pretty, but may be easier to play with, since it doesn't have any of the complexities of dynamic route allocation or queued packet buffers to worry about.

Chapter 8

Security

It would be nice to have some, wouldn't it ?

DoS attacks on this code are many and varied. If you're not on the global ACL or client ACL, there's relatively little you can do, but if you have an account on the client machine, you can spoof the server if you know the cookie (and it's easy to find, since it's sent in the clear). If `do_fork` is on, spoofing the server is utterly trivial, and all access control is done at the client...

The moral of this story is 'don't put insecure machines in the client ACL'—at least, until version 2.1, which will have md-5 based authentication.

If you are in the global ACL, you can cause DoS attacks by:

- Connecting to the client vast numbers of times, running it out of fds.
- Repeatedly connecting and disconnecting: setting up and tearing down connections is complex and time-consuming.

And if you aren't, sending a UDP flood should cause CPU usage to go up nicely. The client and server are written in such a way that I believe an actual lock-up (no processes other than them can run) with RT scheduling is impossible, but you can certainly make the client and server machines thrash.

There is also a social engineering attack: be a moron on as many servers as possible via the proxy, and the proxy's IP will be banned.

I hope to do something about some of these in the next release.

Chapter 9

Further Work

The obvious extension is to add profiling capabilities and try to work out some of the Quake II network protocol. Some sort of automated charging infrastructure might be neat for JANET sites.

The current encapsulation protocol is quite inefficient. This doesn't matter much on ethernet (MTU 1500), but is probably pushing it for ISDN and certainly dodgy for modems. Something more compact should be arranged (IP option fields ?).

Cryptographic authentication for clients would be nice: possibly something MD5-driven.

64-bit-cleanness would be nice, so it can work on Alpha.

The server should set an alarm, and boot the client if it hasn't sent its cookie in some fixed time.

Occasionally, servers from whom the proxy has disconnected carry on sending packets, giving errors as they do so. I've no idea why, but I suspect it's because QII isn't checking the return code from recvfrom().

One possibility is for the proxy to act like an infinite improbability shield: detecting world data from the server and sending back move commands so you stay out of the way of anything anyone shoots at you. I'm not sure how difficult this might be, though (before anyone asks, yes you probably should use a game DLL...).

Chapter 10

Licencing

rq2proxy is licenced under the terms of either the GNU Library General Public Licence, version 2 or above, or under the terms of Larry Wall's Artistic Licence, at your option. A copy of both licences may be found in the distribution directory under the names `LGPL-2.txt` and `Artistic.txt` respectively.

Note that the files `config.sub` and `config.guess` are part of GNU autoconf.

The latest distribution of *rq2proxy* can always be obtained from `http://epona.ucam.org/rrw/qproxy.html`.

rq2proxy is Copyright (c) Richard Watts (`Richard.Watts@cl.cam.ac.uk`), 1998.