AniTMT

Hacker's Guide

Part II

"Network"

Technical description for:

NRP (Network Render Protocoll) Version 1.0

`anitmt-server` Version 1.0

`anitmt-client` Version 1.0

written by:
Jan Theofel, `jan@theofel.de`

Januar 2000

Part I.

Introduction

Contents

Part II.

NRP (Network Render Protocoll)

1. NRP via TCP/IP

1.1. Using IPs

The NRP uses the TCP/IP protocoll for communication. As is does not really need the client IP, it should be run under IPv6 without any problems, but this is not yet tested.

Another important point is that the clients will all get a unique client-ID (CID) which is not depending on the IP. This will allow you also to use Clients behind gateways with ipmasq. (Or should I call it NAT for our Novell people?).

1.2. Default port

Like it or not, but the NRP's default port is **4004**. As far as we know, there are no other important protocolls which use this port.

Of course it is also possible to use any other port which is free on your server. You can do this by changing the configuration file or by calling the server with the option `--server-port port`. But this would not be so fine for public servers of course, because teh user would have to change the port according to the server he wants to connect to.

2. Basic NRP definitions

2.1. Line terminator

The line terminator which is used in the NRP is the default internet one's: First the ascii character 15 followed by the 12.

According to the perl doc[1] this should be fine on most systems as long as you really send `\015\012`. But they are not sure about Macs there.

While hacking the `anitmt-server` and the `anitmt-client` I simply used the standard perl read and write functions on my sockets and wondered that this worked fine. Hope that this is true for most other systems. If not, you can easily but a regex `s/[\015,\012]//g` followed by adding `\015\012`.

2.2. Server code

Each line which the server sends (except for data transmission) is started by a three-digit number. The first one defines the section of the protocoll which is actually used, the second one gives an error code and the last one is the line code. All these codes are explained in the tables 2.1, 2.2 and 2.3.

section code	description
0	Initial server message.
1	Handshaking the server.
2	Status informations are exchanged.
3	Section in which client informations are recieved.
4	Section in which the client gets work.
5	Section in which the client sends his results.
6	Data is exchanged (via internel protocoll)
7 – 8	(reserved)
9	Connection closed

Table 2.1.: List of section codes

These codes are the basic information for the client to understand the protocoll. They tell him if an error ioccured and when he has to wait for more server informations.

[1] `perldoc perlipc`, section "Internet Line Treminators"

line code	description
0	This is a single line message.
1	This line is followed by one or more others.
2	This is the last line.
3	Data follows after this line.
4	Last line of data.
5 – 9	(reserved)

Table 2.2.: List of line codes

error code	description
0	first server message (no error)
1	no error
2	unknown command
3	mal formed command
4	command not allowed here
5 – 9	(reserved)

Table 2.3.: List of errors

Some examples:
The server will first send the line
`000␣anitmt-server␣NRP␣server␣ready␣-␣protocoll␣v0.01`
or something very similar. The `000` tells that the server has sent his initial message, that there are no more lines except for this one and that there was no error.
Later in the connection the client might get
`413␣Command␣not␣allowed!`
`423␣You␣can␣get␣help␣by␣sending␣'help'.`
This tells us that we are in the section when the client wants to get work (section code 4), but that the server does not know the command the client has sent (error code 3, "unknown command"). The first line will be followed by some others (line code 1), the second one is the last (line code 2).

3. Sharing data

4. The NRP commands

4.1. Basic commands

HELO

This is the first command the client should send to the server. It is not needed if you want to enter debug or project mode.

protocol extract:

```
001 anitmtserver s0.03 (build 2000-01-23) - protocol version p0.01
001 PLEASE CONNECT USING "HELO". "BYE" TO QUIT.
002 YOU CAN GET HELP BY SENDING 'HELP' AS COMMAND.
helo
100 HELO 127.0.0.1
```

followed by:

```
SEHE
```

possible errors:

none

SEHE

If a client was already connetced he can send this command followed by his client id (CID).

protocol extract:

```
100 HELO 127.0.0.1
sehe <CID>
101 SEHE 127.0.0.1 AS <CID>
102 ACCEPETD COMMANDS ARE ALL EXCEPT LOGIN.
```

followed by:

```
GET, SEND
```

possible errors:

`190: This is not a valid CID.`
Maybe the server lost his client informations.
`190: SEHE denied! Already connected from <IP>.`
Another client is connected with this CID yet. If this error is permanent resend your information and get a new CID.

BYE

Closes the connection between server and client.

protocol extract:

```
bye
900 CU L8ER!
```

followed by:
none
possible errors:
none

MSG

Most commands set an additional message. It can be read by sending the `MSG` command. Use it for understanding the errors which occur and to tell the user what happend if nessecary.

protocol extract:

```
helo
100 HELO 127.0.0.1
info
100 OK TO SEND INFO FOR 127.0.0.1
msg
800 Next command must be SYS.
```

followed by:
any command
possible errors:
none

4.2. Client informations

The first time a client connects to the server, some information about the client is send to the server. In the following sections the commands to exchange these informations are described. They must be sent in this order.

INFO

This is the first command the client should send to the server. It is not needed if you want to enter debug or project mode.

protocol extract:

```
helo
100 HELO 127.0.0.1
info
100 OK TO SEND INFO FOR 127.0.0.1
```

followed by:
`SYS`
possible errors:
`130 MAL FORMED COMMAND`
This happens if you send additional parameters for `INFO`.
`140 COMMAND NOT ALLOWED HERE`
Occurs if `INFO` is not send immediatly after `HELO`.

SYS

`SYS` sends some information about the system the client is running on. It is one string parameter, which is stored on the server to inform the admin of it which clients are calculating for the server. The parameter may (as the only one in this protocol) contain spaces.

protocol extract:

```
info
100 OK TO SEND INFO FOR 127.0.0.1
sys Linux
180 VALUE ACCEPTED!
```

followed by:
`SPED`
possible errors:
`130 MAL FORMED COMMAND`
This happens if you forget to send one parameters for `SYS`.
`140 COMMAND NOT ALLOWED HERE`
Occurs if `SYS` is not send immediatly after `INFO`.

SPED

Informs about the speed of the client. It will be used in comming protocol versions to share the jobs between the clients more efficently. Anyway it is not yet implemented in the server but must be send from the client.

Parameters are a integer and a string, where the string must be `NRPMIPS`.

The NRPMIPS will be some sort of benchmark results, which can be later detected with a little benchmark program. Send 0 NRPMIPS if this speed is not yet defined / tested.

protocol extract:

```
sys Linux
180 VALUE ACCEPTED!
sped 0 NRPMIPS
180 VALUE ACCEPTED!
```

followed by:
`JOBC`
possible errors:
`130 MAL FORMED COMMAND`
This happens if an incorrect number or wrong parameters are send.
`140 COMMAND NOT ALLOWED HERE`
Occurs if `SPED` is not send immediatly after `SYS`.

JOBC

Defines the amount of disk cache which can be used on the client to store jobs in. Note that this value is not yet used by the server but must be set be the clients!

Parameters are a integer and a string, where the string must decalre the unit used. Exapmle: `JOBC 10 MB`

protocol extract:

```
sped 0 NRPMIPS
180 VALUE ACCEPTED!
jobc 10 mb
180 VALUE ACCEPTED!
```

followed by:
`RESC`
possible errors:
`130 MAL FORMED COMMAND`
This happens if an incorrect number or wrong parameters are send.
`140 COMMAND NOT ALLOWED HERE`
Occurs if `JOBC` is not send immediatly after `SPED`.

RESC

Defines the amount of disk cache which can be used on the client to store jobs in. Note that this value is not yet used by the server but must be set be the clients!

Parameters are a integer and a string, where the string must decalre the unit used. Exapmle: `RESC 1 GB`

protocol extract:

```
jobc 10 mb
180 VALUE ACCEPTED!
resc 1 gb
180 VALUE ACCEPTED!
```

followed by:

ADRR

possible errors:

130 MAL FORMED COMMAND

This happens if an incorrect number or wrong parameters are send.

140 COMMAND NOT ALLOWED HERE

Occurs if RESC is not send immediatly after JOBC.

ADRR

Tells the server which Raytracer / Renderers the client supports. Send a single ADRR <NAME> vor every installed one.

The name is normally formed by adding the program version to the program name.

protocol extract:

```
adrr povray30
111 ADDED AN R/R FOR 127.0.0.1: POVRAY30
112 SEND ANOTHER 'ADRR' OR 'LARR'.
adrr povray31
111 ADDED AN R/R FOR 127.0.0.1: POVRAY31
112 SEND ANOTHER 'ADRR' OR 'LARR'.
larr
```

followed by:

ADRR, LARR

possible errors:

130 MAL FORMED COMMAND

This happens if an incorrect number or wrong parameters are send.

140 COMMAND NOT ALLOWED HERE

Occurs if ADRR is not send immediatly after RESC or another ADRR.

LARR

Tells the server that this client has sent all its Raytracer / Renderers.

protocol extract:

```
adrr povray30
111 ADDED AN R/R FOR 127.0.0.1: POVRAY30
112 SEND ANOTHER 'ADRR' OR 'LARR'.
adrr povray31
111 ADDED AN R/R FOR 127.0.0.1: POVRAY31
112 SEND ANOTHER 'ADRR' OR 'LARR'.
larr
111 NO MORE R/R FOR 127.0.0.1
112 SEND YOUR F/P USING 'ADFP'.
```

followed by:
`ADFP, LAFP`
possible errors:
`140 COMMAND NOT ALLOWED HERE`
Occurs if `LARR` is not allowed when it was send.

ADFP

Tells the server which Filters / Plugins the client supports. Send a single `ADFP <NAME>` vor every installed one.

The name is normally formed by adding the program version to the program name.

protocol extract:

```
adfp nightwatch01
111 ADDED AN R/R FOR 127.0.0.1: NIGHTWATCH01
112 SEND ANOTHER 'ADFP' OR 'LAFP'.
lafp
```

followed by:
`ADRR, LARR`
possible errors:
`130 MAL FORMED COMMAND`
This happens if an incorrect number or wrong parameters are send.
`140 COMMAND NOT ALLOWED HERE`
Occurs if `ADRR` is not send immediatly after `RESC` or another `ADRR`.

LARR

Tells the server that this client has sent all its Filter / Plugins.

protocol extract:

```
adfp nightwatch01
111 ADDED AN R/R FOR 127.0.0.1: NIGHTWATCH01
112 SEND ANOTHER 'ADFP' OR 'LAFP'.
lafp
111 NO MORE F/P FOR 127.0.0.1
112 SELECT DATA EXCHANGE MODE 'DEXM'.
```

followed by:
`DEXM`
possible errors:
`140 COMMAND NOT ALLOWED HERE`
Occurs if `LAFP` is not allowed when it was send.

DEXM

Selects the *data exchange mode*, which will be used to exchange job and result files between the client and the server.

One parameter is needed which is `NFS`, `FTP` or `INTERNAL`.

protocol extract:

```
dexm nfs
180 VALUE ACCEPTED!
```

followed by:

`DEXM`

possible errors:

`190 VALUE DENIED!`
This DEXM is not supported by the server.
`140 COMMAND NOT ALLOWED HERE`
Occurs if `LAFP` is not allowed when it was send.

FIIN

Finishes the information transfer. (FIIN = finish information)

The server sends some values afterwards. Important is the client id (CID) which is generated for this client. It should be stored in a file. After this information exchange, the client should disconnect using `BYE` and return again with `HELO` and `SEHE <CID>`.

protocol extract:

```
fiin
101 FINISHING FOR 127.0.0.1
101 YOUR CID: 127.0.0.1_XXX
102 PLEASE DISCONNECT NOW USING 'BYE'.
```

followed by:

`BYE`

possible errors:

`140 COMMAND NOT ALLOWED HERE`
Occurs if `FIIN` is not allowed when it was send.

DEXM

Selects the *data exchange mode*, which will be used to exchange job and result files between the client and the server.

One parameter is needed which is `NFS`, `FTP` or `INTERNAL`.

protocol extract:

```
dexm nfs
180 VALUE ACCEPTED!
```

followed by:

`DEXM`

possible errors:

`190 VALUE DENIED!`

This DEXM is not supported by the server.

`140 COMMAND NOT ALLOWED HERE`

Occurs if `LAFP` is not allowed when it was send.

DEXM

Selects the *data exchange mode*, which will be used to exchange job and result files between the client and the server.

One parameter is needed which is `NFS`, `FTP` or `INTERNAL`.

protocol extract:

```
dexm nfs
180 VALUE ACCEPTED!
```

followed by:

`DEXM`

possible errors:

`190 VALUE DENIED!`

This DEXM is not supported by the server.

`140 COMMAND NOT ALLOWED HERE`

Occurs if `LAFP` is not allowed when it was send.

4.3. Status login

4.4. Getting work

4.5. Sending results

4.6. Other commands

Part III.

anitmt-server

Part IV.

anitmt-client