DirectConnect TNG

A call for a new protocol

Draft, version 0.1 May 3rd, 2003

By Jan Vidar Krey <janvidar@extatic.org>

1. Introduction

1.1. Background

I have been developing QuickDC for over a year, written it from scratch using packet sniffers and poking through other implementations from projects like javaDC, dctc and DC++.
During this period both me and other developers have found and discussed numerous problems in the current Direct Connect protocol. We have suggested and implemented many extensions to the protocol, like the "Extendedprotocol" (implemented first in DC++), "Quicklist" and many more.
They are all "hacks" to improve some flaws in the system, but in the long run it will just make things more complicated.

1.2. Status of this document

This document does not specify a implemented protocol in any way.
These are just suggestions proposed by me and other parties who have contacted me about some of these things. There are many issues that needs to be worked out.
The idea here is to get the ball rolling and create a standard protocol sometime in the future.
This document doesn't address the client to client protocol at all.

1.3. Some of the faults

Here is a short list of the problems me and others have found (in no particular order).

- The protocol doesn't handle error conditions (If something went wrong, just ignore it, thats the implementation).
- Implementations doesn't handle different states of the protocol, command scopes and order is undefined in most cases and implemented differently between projects.
- The protocol doesn't escape special characters, like: |, $, <, >, character 0x5 or even spaces in some cases (the only exception here is in the $Lock), it doesn't support 16bit unicode.
- The whole network relies on a simple nickname, this is problematic when somebody has taken your name, or if you use the same nickname from different computers, or if someone uses a different nickname on different hubs (thus you are unable to download).
- Searches are difficult to track (which search result belongs to which search request). This makes automated searches difficult.
- No implementation of hashes to uniquely identify files.
- The default ports are 411 and 412 which are privileged ports on Unix and commonly blocked by firewalls
- The protocol doesn't support multisource downloads (without hacks).
- The protocol doesn't handle secure communications (like SSL)
- The protocol doesn't offer support for IPv6
- Hubs aren't uniquely identified (bug in search results if the hub changes it's name)
- The Client-Client protocol is uneccesary complex, with too much interactions, which gives poor performance. A stateless protocol is prefered.
- No good support for multiple hubs

1.4. A proposed solution

Instead of adding bloat to the current protocol, which seems to be the trend these days, I propose creating a totally new system around on the DC model (clients and hubs), which should be reasonable easy to implement in current open source DC clients.
In short, this means a brand new protocol, which addresses most of the annoying parts in the current Direct Connect protocol.

1.5. Executive summary

In general we remove the nickname dependencies, add hashes and make a more statefull and secure protocol. For the client-client protocol we are moving towards a (semi-)stateless protocol.

2. New protocol

2.1. The basics

First of all, the protocol is text based, like most other protocols (FTP, SMTP and HTTP).
Commands are (unlike the current DC protocols) transmitted as lines separated by the common Internet standard; CRLF (carriage return and linefeed), which means it's easier to debug implementations with a simple telnet session.
Special characters like are escaped C-style with backslashes "\", ASCII 0x5C (examples; \\, \n, \r, \t etc "\ " for space).

A username can be strictly alpha numeric and additionally contain the following characters "[](){}$&%@#"

The lock/key challenge response system is gone. It was ment to "protect the network" from unauthorized clients, but it didn't work well did it?

2.2. Client -Server Protocol

2.2.1. The basics

Ok, the big difference here is the new GUID concept, but here are two basic facts;

- The server does not use any "default" TCP port by specification (up to the implementation or administrator).

- All address are in this format;
    - IPv4: "x.x.x.x:port"
    - IPv6: "[x:x:x:x:x:x:x:x]:port" (follows the RFC 2732 guidelines).
    - DNS: "fully.qualiyfied.hostname:port"

- The connecting party (client) will always talk first when connected (this to make automated scans difficult).

- All error/info strings from special commands should not exceed 50 bytes.

2.2.2. New concepts

A new concept GUID, which is short for "Globally Unique ID". Similar functionality is in the recent Gnutella protocol aswell.

Each node (client and server) has a unique ID. All references in the protocol should use this ID instead of the nickname. This way we can have a network that doesn't rely on the nickname.
You can join multiple hubs with multiple nicknames, and still be interpreted as one user.

The GUID is calculated as a SHA-1 (160 bit) hash of "ip:port" IPv4 or IPv6 doesn't matter.
This GUID is usually represented as a 40 byte text string hexadecimal style, but that might be too large. It is possible to "squeeze" this size by using some other encoding (20 bytes is minimum nevertheless).

The GUID is created the first time the client is ran, and it will be stored and used for each later session.
The user will be identified by this ID accross different hubs and names. All login systems (registered users, etc) will use the GUID to identify a user.

-

2.2.3. An example handshake between client and hub

Client connects to server.
Client: DCTNG [protocol version]
If protocol mismatch (did not get DCTNG first)
    Server: (closes connection without any error message)

Server: DCTNG [supported protocol version]

if the client or server has any protocol mismatch the one not understanding the other disconnects. The client may retry with another protocol.

Server: HubName [text]
Optional: Server->ServerVersion [name, version, etc]

If the server is full:
    Server: HubFull
    Server: Redirect address (optional)
    Server closes the connection.

Or if your IP is banned, or you are otherwise not permitted to log in;
    Server: AccessDenied [reason]
    Server: Redirect address (optional)
    Server closes the connection.

For acceptance:
Server: Login

Client: UserAgent [string] (optional)
Client: Info <GUID> <nick> <shared bytes> <freeslots/totalslots> <hubs> <mode> <description>

If you have no access the server will reply:
    Server: AccessDenied [reason]
    Server: Redirect address (optional)
    Server closes the connection.

Otherwise it *may* request a password for registered users (note users are identified by the GUID not the nick).
    Server: Password [challenge]
    Client: Password [response] (See howto calculate the response).

    If not OK, repeat the process *or*
        Server: AccessDenied [reason]
        Server: Redirect address (optional)
        Server closes the connection.

If authentication went well, or no authentication is required the server will go on sending your "Info" data to everybody on the hub, including yourself (which indicates you are logged in).
The "Info" for all other users are sent aswell...
.

2.3. Command reference

2.3.1. DCTNG

Syntax: DCTNG *version*

Parameters:

version a positive integer representing the protocol version supported by the client

Valid: Client to server and server to client, during initial handshake

Errors: [disconnect]

UserAgent

Syntax: UserAgent *string*

Parameters:

string describing the client software. This string should **not** exceed 50 bytes.

Valid: Client to server, during initial handshake only.

Errors: none

See also: ServerVersion

Example:

```
HubName QuickDC 0.1.0 Linux/IA32 DCTNG
```

2.3.2. HubName

Syntax: HubN*ame string*

Parameters:

string name of the hub. This string should not exceed 50 bytes.

Valid: Server to client, during handshake, or at any time after successfully logged in.

Errors: none

Example:

```
HubName Rob's cool hub.
```

2.3.3. ServerVersion

Syntax: ServerVersion *string*

Parameters:

string describing the server software. This string should not exceed 50 bytes.

Valid: Server to client, during initial handshake only.

Errors: none

See also: UserAgent

Example:

```
ServerVersion NextGenDC HUB 0.0.2 (build 5) FreeBSD 4.7
```

2.3.4. HubIsFull

Syntax: HubIsFull

Parameters: none

Valid: Server to client, during initial handshake only.

Errors: none

See also: Redirect

2.3.5. Redirect

Syntax: Redirect *address*

Parameters: *address to redirect to*

Valid: Server to client at any time.

Errors: none

See also: HubIsFull

Example:

```
Redirect some.dc.server.com:15019
Redirect another.server.com:9258
```

2.3.6. AccessDenied

Syntax: AccessDenied string

Parameters:

string A reason why your access was denied. May not exceed 50 bytes.

Valid: Server to client during initial handshake or login

Errors: none

See also: Redirect

Example:

```
$AccessDenied Your IP is banned.
$AccessDenied Too many login attempts.
```

2.3.7. Login

Syntax: Login

Parameters: none

Valid: Server to client to prompt for userdata

See also: Password

Errors: AccessDenied

2.3.8. Password (server to client)

Syntax: Password

Description: Request a password from a user. Can be sent multiple times if "Pass" fails.

Parameters: *challenge*

Valid: Server to client after the Login request have been answered

See also: Login, Password

Errors: none

2.3.9. Password (client to server)

Syntax: Password *password*

Description: Send password to server. The password is sent as a response to a server challenge. To prevent playback attacks, the server will send a password request containing some random data. These data will be concatinated (appended) to the password and the whole string is MD5 digested and sent back to the hub for verification.

Parameters: *password* The password challenge hash.

Valid: Client to server after the optional Password-request

See also: Login, Password

Errors: Password (meaning: please try agan), AccessDenied

Example:

```
Server: Password this_is_supposed_to_be_random
Client: Password b2133e9a8b93bc75d55518ff583d0ab2
```

How this was calculated:

password	*= "swordfish"*
challenge	*= "this_is_supposed_to_be_random"*
concatenated	*= "swordfishthis_is_supposed_to_be_random"*
hashed	*= "b2133e9a8b93bc75d55518ff583d0ab2"*

2.3.10. Info

Syntax: Info *GUID nick bytes slots hubs mode flags description*

Description: This command is used to provide the user data for a client

Parameters:

GUID	Globally unique ID for client
bytes	Bytes shared (64bit integer)
nick	Nickname for user
slots	Slot status on the format "freeslots/totalslots"
hubs	Number of hubs client is connected to
mode	One character mode: { 1=Active, 2=Passive, or 4=Proxy }
desc	User specified description. Must not exceed 50 bytes.

2.3.11. Search

Syntax: Search *Destination ID Category Mode Metadata Pattern*

Description: CSend password to server

Parameters:

Destination The destination of the search result, can be two things:
Hub:GUID for passive searches
Address notation is used for active searches (replied with UDP).

ID A 1 byte, up to an 8 byte alphanumeric ID which should be used by clients to track results and which search request it is a reply to.

Category

0 = Any
1 = Audio
2 = Compressed
3 = Document
4 = Executable
5 = Picture
6 = Video
7 = Directory
8 = Regexp

Mode The search mode:

0 = any size
1 = size atleast
2 = size atmost
3 = exact size
4 = match HASH

Metadata *This field depends on search mode it can be the size for searchmodes 1-3, a hash for search mode 4 (example: SHA1:f2a394e49ac933602cb46897b780425ed73ed6f4) or simply 0 for search mode 0.*

pattern *Search pattern, special characters are escaped.*

Valid: To both client and server after sucessfull login

See also: SR

Examples:

```
Search Hub:a88c9fe4842d0b19729d49b9232b5b4c664c1fde X862abC9 0 0 0 Madonna\ mp3
Search 192.168.51.12:12825 1280a1 6 1 64000000 Some\ Cool\ movie.avi
```

2.3.12. SR

Syntax: *SR [destination] ID size data*

Description: Send result via HUB for passive requests.
For active search requests the same should be sent as an UDP packet, without the destination (as it is specified by the UDP address).

Parameters:

destination should be the GUID where the search reply belongs

ID The search tracking ID

size Size of the search result file

data *Filename with full path (should be escaped)*

Valid: To both client and server after sucessfull login

See also: SR

Examples:

```
Search Hub:a88c9fe4842d0b19729d49b9232b5b4c664c1fde X862abC9 0 0 0 Madonna\ mp3
SR a88c9fe4842d0b19729d49b9232b5b4c664c1fde X862abC9 1208123781 Madonna\ -\ Virgin.mp3
```

2.3.13. Connect

Syntax: Connect *GUID address*

Description: Request client with specified *GUID* to connect to *address*

Parameters:

GUID client identifier

address *TCP address or IP with port to connnect to...*

2.3.14. ReqConnect

Syntax: ReqConnect *GUID1 GUID1*

Description: Request client A with GUID1 to ask client B (with GUID2) to connect to A.
This is used if client A is firewalled and cannot receive incomming TCP connections.
Client B will issue a "Connect" command to initiate a connection, if B also is firewalled it will send a ReqConnect back (which means both parties are firewalled and cannot talk.)

Parameters:

GUID1 client identifier

GUID2 client identifier

Example:

```
ReqConnect a88c9fe4842d0b19729d49b9232b5b4c664c1fde 73c1a57735f9e8cc6afbc2d3a7d700233cf7fb42
Connect 73c1a57735f9e8cc6afbc2d3a7d700233cf7fb42 192.168.0.82:19414
```