StrongDC++ DHT

This document describes one of many versions of P2P network based on Distributed Hash Table. The main idea was taken from Kademlia including some tweaks, mainly from eMule's KAD network.

Protocol

StrongDC++ DHT implementation uses ADC protocol over UDP interface. Default UDP port is 6245. Complete description of ADC commands is at the end of this document.

Node

This network consists of clients running StrongDC++ or its derivation. These are called nodes and they communicate with each other and store information for each other. Every node consists of CID, IP and UDP port. CID is a 192-bit unique identifier used in ADC network.

Metric

Everything (nodes, files…) in the network is identified by 192-bit number. Algorithm to generate these numbers is Tiger Tree Hash. Operations in the network are based on exclusive OR. The distance between two IDs is defined as:

$$distance(x, y) = x \oplus y$$

Routing table

Current routing table consist only from one bucket, which can contain up to 1920 nodes (CID size * K). This bucket is sorted by the time when each node has been seen active for last time.

Inserting nodes

Each time the client receives communication from another node, it updates its own routing table. When node is new (i.e. it doesn't exists in routing table), it is added at the end of the bucket. When the bucket is full, nothing happens. When node is already contained in the routing table, it is moved to the end and set it as alive. This ensures that active nodes are at the end, while inactive nodes are at the beginning.

Node expiration

When new node is inserted into routing table it is marked as it will expire very soon. When node is being set alive (i.e. another packet received from him), the node expiration is set according to following table:

Node is known for	Node will expire in	Node type will be set to
Just created	Now	3
Less than 1 hour	60 minutes	2
1 – 2 hour	90 minutes	1
More than 2 hours	120 minutes	0

Removing nodes

Nodes are checked every minute. The oldest expired node is pinged (by sending INF) and its type is increased by one. It has 2 minutes to respond else it will be marked as expired. If node's type reaches 4, it will be removed from routing table on next check.

Searching for a node

When a client wants to search for a node (with target ID) in the network, it selects 50 nodes which ID is the closest to the target ID and inserts them to the "possible" list sorted by its distance to the target ID. Then it sends SCH request to 3 closest nodes. These nodes respond (if they are online) with RES command containing 10 nodes from their routing table which are closest to our target ID. Searching client inserts these nodes to "possible" list. This list is checked every 3 seconds when it selects another 3 closest nodes and sends the SCH request to them. It must be ensured that request isn't send twice to the same node(neither CID nor IP). This operation is repeated until the "possible" list is empty or whole search operation passes 45 seconds. Then it waits 15 seconds for possibly delayed results. Now search operation is complete.

Searching for a file

Operation of searching for a file is almost the same as a searching for node. There's only exception when remote node has wanted file in its store, it returns directly file's source list instead of node list. It returns up to 300 sources.

Not implemented idea #1: *When client wants to search for file source, it could look into its own store first. If required file ID is present, it wouldn't have to send search request at all.*

Not implemented idea #2: *When remote client receives search file request. It looks into its store for file ID. It should also look into its own sharelist to provide results faster.*

Publishing a file

When client wants to publish file from its sharelist, it performs normal search request for node which ID is closest to the file ID. But it has 40 seconds to finish operation (instead of 45 seconds). Every node which responds to search request is added to "responded" list. When search operation completes, 10 closest (to file ID) nodes are selected from "responded" list and PUB request is sent to them. Remote nodes respond with STA when the file was correctly stored to their store.

Not implemented: *Currently, it doesn't solve situation when remote node denies publishing our file. It should be ensured that file will always be published to 10 nodes.*

Joining the network

When a new node wants to join the network, it must go through a bootstrapping process. It should know at least one node already participating in the network. The more known nodes we have, the more probability that some of them are still online is. The list of such nodes is stored in file dht.xml. When this file contains zero nodes, the list is downloaded from remote server.

The client sends “GET nodes dht.xml” command to the nodes in the list until one of them responds. (These commands aren’t sent at once, but there’s a delay to give a remote node time to respond.) Remote client responds to this command with SND containing list of 20 random nodes from its routing table. The client stores them. When any remote node sends a packet to us (although it’s not a node we contacted), the bootstrapping is stopped and client starts searching for own ID in 3 minutes.

When client disconnects from the network, it stores up to 50 nodes closest to itself, so it can bootstrap from them next time. These bootstrapping nodes won’t be added to the routing table later because there’s possibility they will be offline.

Compression

Any ADC message can be compressed using ZLIB algorithm. Client must add ADC_PACKED_PACKET_HEADER character before any compressed message. Compression isn’t mandatory and should be used only if compressed message is smaller than uncompressed one.

Encryption

Node A sending a message to remote node B will append UK tag containing its public UDP key. The choice of this key depends on client implementation but it should be ensured that it’s random and different for more nodes. Node B sending a message to node A will encrypt this message with key specified as TTH hash of node-A’s key and node-A’s CID. When key is unknown only CID will be used. Encryption algorithm is RC4.

Following scheme shows encrypted packet:

Any character except of ADC_PACKET_HEADER or ADC_PACKED_PACKET_HEADER	RC4-encrypted part	
	Magic Value 0x5B	(compressed) ADC message containing our public UDP key

StrongDC++ key: *The public UDP key of node A is specified as TTH hash of private UDP key and IP address of node B. Private UDP key is random 32-bit integer valid for whole client.*

Protocol commands

SCH

Searches for object in the network

TR	requested TTH/CID
TO	Token
TY	search type (1 = file, 3 = node, 4 = store)

RES

Response to SCH

TO	token (it must match the token in SCH)
NX	node/source list in XML format. When search type is "file" and node knows sources for searched file, it should directly respond with these sources; in other cases, it should returns

	K nodes closest to requested TTH/CID. <Nodes> <Source CID="user ID" I4="IP address" U4="UDP port" SI="filesize" PF="is it partial file?"/> <Node CID="user CID" I4="IP address" U4="UDP port"/> </Nodes>

PUB

Request for file publishing

TR	TTH root
SI	file size
PF	1 - file is partially downloaded (present in our queue)

INF

Sends our info to node

TY	1 - node must response with its own info
VE	client version
NI	user's nick
SU	features (ADC0, TCP4, UDP4 now supported)

CTM

CTM protocol port token

Connect to me request. Protocol must be ADCS/0.10, ADC/1.0.

PSR

Partial file request

U4	DC UDP port (that one used for classic DC searches, not DHT one!)
TR	file TTH

MSG

Private message (not supported yet)

GET

GET type identifier

Request to get some special data. Only type “nodes” is currently supported. In such case, identifier must be “dht.xml”.

SND

SND type identifier data

Response to GET. Type and identifier must be same as in received GET. The type is "nodes" then data contains node list in XML format.

STA

STA code description

status message

FC	PUB - response to file publish request (TR contains published file TTH) FWCHECK - response to UDP firewall check (I4, U4 - our external IP/port)