# DarkChat: a light weight, [almost] server-free, push-based robust p2p network with a chat payload

Nathan Griffith and Ahmet Aktay

May 12, 2010

## 1 Abstract

The purpose of this project is to create a protocol for clients on a network to function with very little dependence on a server. In the case of *DarkChat*, the clients function as 'chat' clients. The clients are able to communicate with each other information such as 'online'/'offline' state and lists of contacts, with little-to-no interaction with the server.

While *DarkChat* relies on a multi-threaded approach, and also makes no use of authentication, implementation decisions such as this have been made independently of the underlying protocol. As such, the protocal could be implemented in any number of ways, and this is meant as merely one example.

## 2 Clients and Servers

The 'Client' class has been built as a wrapper for the messaging and interface classes, putting them all together into one usable chat client. Note that the term 'client' is slightly misleading, as with the inclusion of a simple flag, a client can function as a server with very few implementation changes. It is possible that this could be reduced even further, such that servers could simply act as 'super-clients' which store information about the entire netowrk, instead of about only their nearby edges and nodes. In either case, the only purpose of the sever is to act as a secondary or backup repository for storing information about the network. Once the network is fully operational, the server is not needed, as long as enough clients remain online that one cluster of edges does not become separated from any other.

### 2.1 Usage: Parameters

When executing the client, a number of parameters can be passed. WIth the exception of 'username' ('-u'), all parameters will revert to default values if none is specified:

```
-p [port number, for accepting incoming communication]
-t [the default number of listening threads]
-u [the username of the client]
-sip [ip address of known server]
-sp [port number of known server]
-s (server flag, specifying that this instance of DarkChat is a server)
```

Note that there is currently no reliable centralized server, so in the actual case of starting a network running from scratch, it would be best to ensure that a server is running, and specify the server ip and

port for each client. Once the clients are online and have contacted each other, one need only use a macro to contact a currently online client.

### 2.2 Usage: Macros

Once the client has been started, a number of macros have been made available to give the user control:

```
\help
 -Display this dialog
\chat <username>
 -Switch to conversation with <username>
\users
 -See a list of KNOWN users (online or off)
\online
 -See a list of known, ONLINE users
\offline
 -See a list of known, OFFLINE users
\add <username>
 -Attempt to add username to list of known users
\ping <ip> <port>
-Attempt to ping the ip at the given port
-If no port is specified, the current instance's incoming will be used
\quit or \exit
 -Leave the program gracefully
```

Note that these are made available before the client has necessarly made successful contact with a server, allowing one to use the '\ping' macro to manually connect to another client or server.

Once a connection has been made, information about offline and online users should propogate across the network and to the client, at which point the macros '\online', '\offline', and '\users' will provide useful information.

To communcate with an online user, one need only use the '\chat' macro (followed by the user's name) to start a new chat session with that user. From that point on, any line of text that is entered and doesn't start with a macro '\' will be sent to *all online sessons* associated with the reciever's username.[1]

## 3 The Algorithm

While the details of the 'Chat' client is useful in understanding some aspects of this algorithm, the true implementation has been abstracted from the end-user. Therefore, to understand the intent behind this project, a deeper explanation of the algorithm is required.

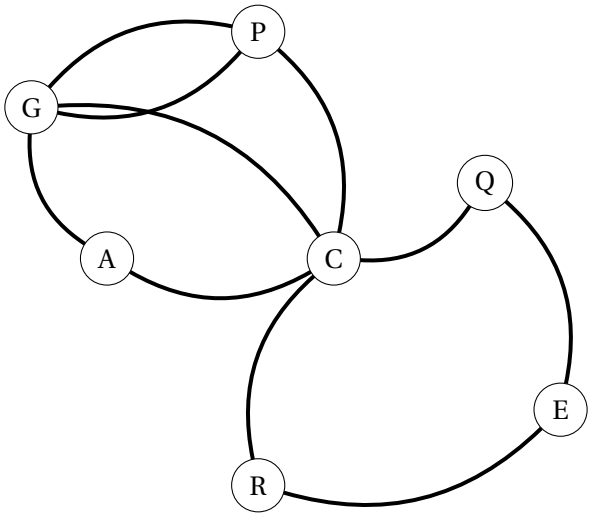
### 3.1 The Network

Instead of picturing the network in terms of 'users' and 'buddies' and 'servers,' as one might with some types of chat implementations, one can also see it very simply as an undirected graph composed of nodes and edges. Each node represents a 'client,' and each edge represents a direct, two-way association between two clients.[2]

[1] Currently there is no way to chat with *only one* session if a user is logged into multiple.

[2] In the context of 'chat,' adjacent nodes might represent "buddies," but it's important to think of them abstractly.

Figure 1: An example network configuration




As shown in **Figure 1**, a network graph might look like one or more clusterings of nodes[3]. In the *DarkChat* implementation, each node is required to keep track of all nodes that are within two edge traversals.

When, for instance, a node is to add or update information about itself, it sends a notification to all nodes in its list of relevant nodes. This means that if, for instance, *Node R* and *Node E* have lost their connection to each other[4], they could rely on either *Node C* or *Node Q* to provide one of the required locations[5].

### 3.2 The Status

prune sessions – compromising between age and online status

### 3.3 The Protocol

Clients make use of a limited number of message types to recieve the information they need.

## 4 Use Cases

NOT for high bandwidth cases – if optimizing bandwidth is not as useful as increasing redundancy and synchronicity

Failsafes and backups – gmail collapse – the backups themselves are more of the 'same' – failure mode is untested, anything causing original to go down causes backup. In this case, the failure mode is being tested every day.

[3]This could be particularly true in the case of a 'chat' client.

[4]This might happen if they have both changed IP addresses since their last communication

[5]Note of two nodes, only one needs to know the other's location for a connection to be established. In the case of *E* and *R*, as soon as *Node E* sends a message to *Node R*, the location of *Node E* has been revealed.

coordiation in case of an emergency

## 5 Possible Optimizations

scaling for bandwidth
message identifiers
combined requests

## References

[1] Atul Adya, John Dunagany, Alec Wolman, *PRS: A Reusable Abstraction for Scaling Out Middle Tiers in the Datacenter*. Microsoft Research; Microsoft Corporation, 2008.