

Chat

Refs: RFC 1459 (IRC)

Multi-user chat program

- Functional Issues
 - Message types.
 - Message destinations (one vs. many groups)
 - Scalability (how many users can be supported)
 - Reliability?
 - Security
 - authentication
 - authorization
 - privacy

Message Types

- Some options:
 - text only
 - audio
 - images
 - anything (MIME)?

Scalability

- How large a group do we want to support?
- How many groups?
- What kind of service architecture will provide efficient message delivery?
- What kind of service architecture will allow the system to support *many* users/groups?

Message Destinations

- Each message goes to a group (multi-user chat).
 - Can we also send to individuals?
 - Should we support more than one group?
 - Are groups dynamic or static?
 - What happens when there is nobody in a group?
 - Can groups communicate?
 - Can groups merge or split?

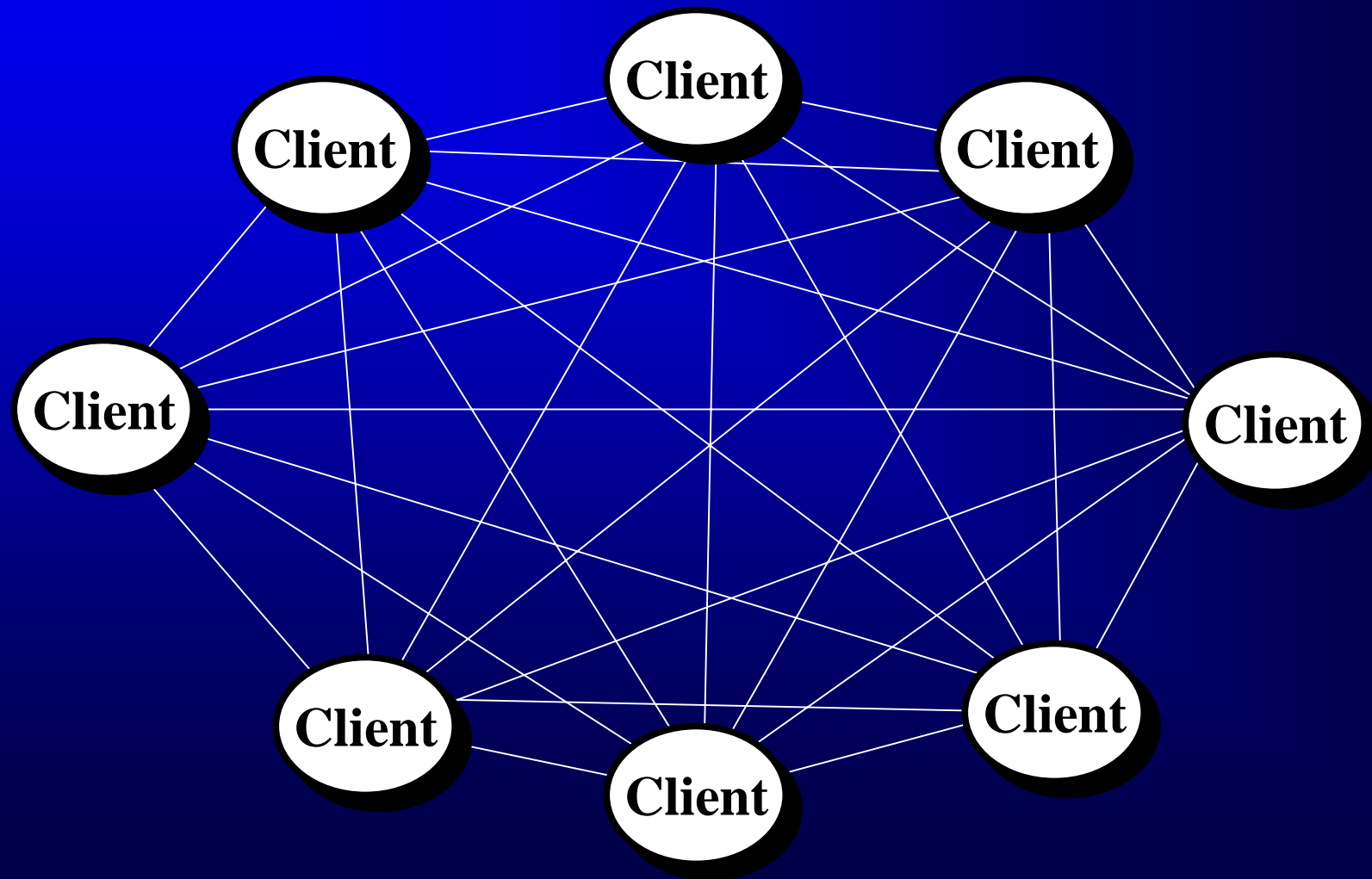
Reliability

- Does a user need to know (reliably) all the other users that receive a message?
- What happens if a message is lost?
 - resend? application level or at user level?
- What happens when a user quits?
 - Does everyone else need to know?

Security

- Authentication: do we need to know who each user is?
- Authorization: do some users have more privileges than others?
- Privacy:
 - do messages need to be secure?
 - Do we need to make sure messages cannot be forged?

Peer-to-Peer Service Architecture

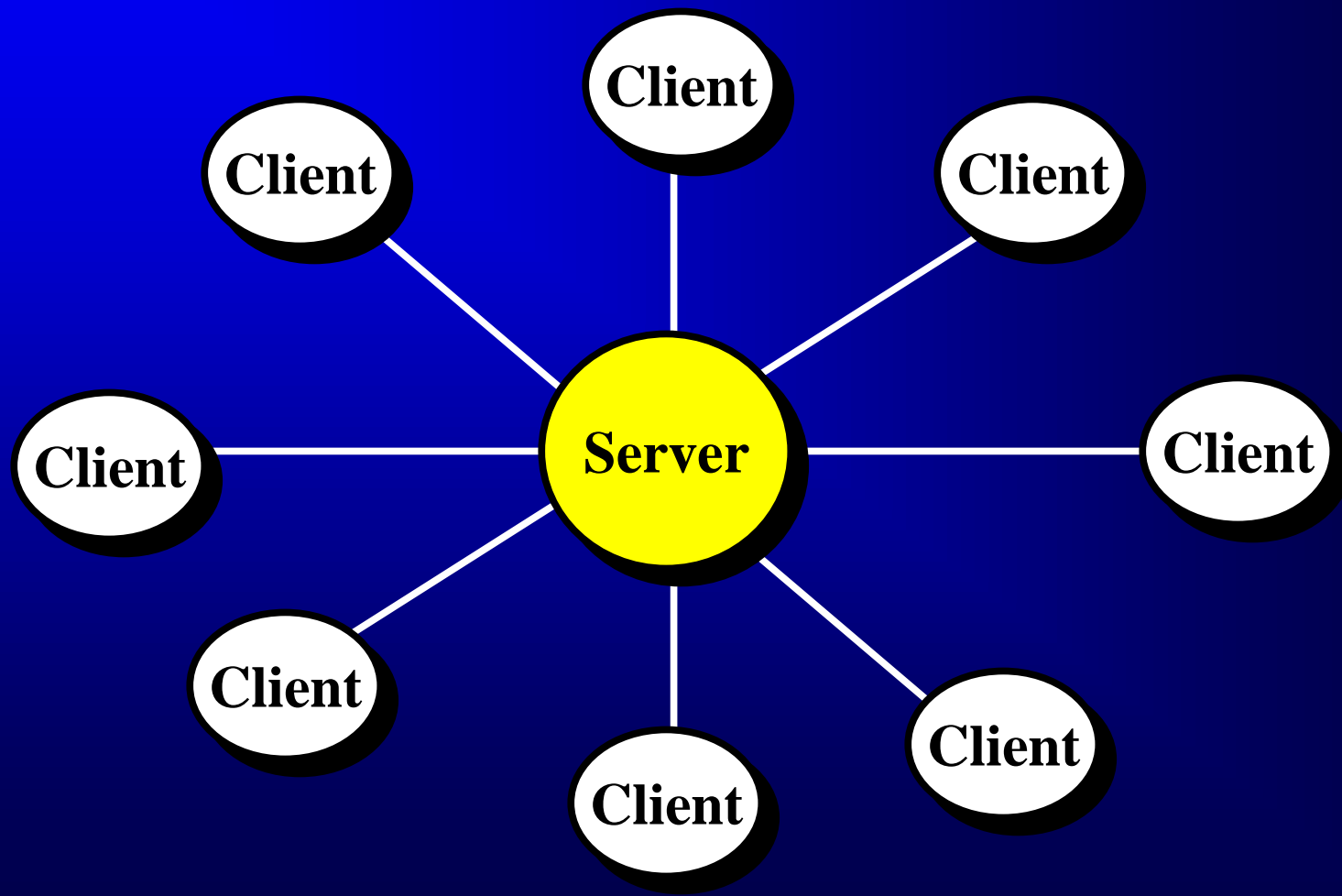


Peer-to-Peer Service Architecture

Each client talks to many other clients.

- Who's on first? Is there a well known address for the service?
- How many peers can we keep track of?
- If 2 peers (clients) are on the same machine, do we need to send a message to the machine twice?

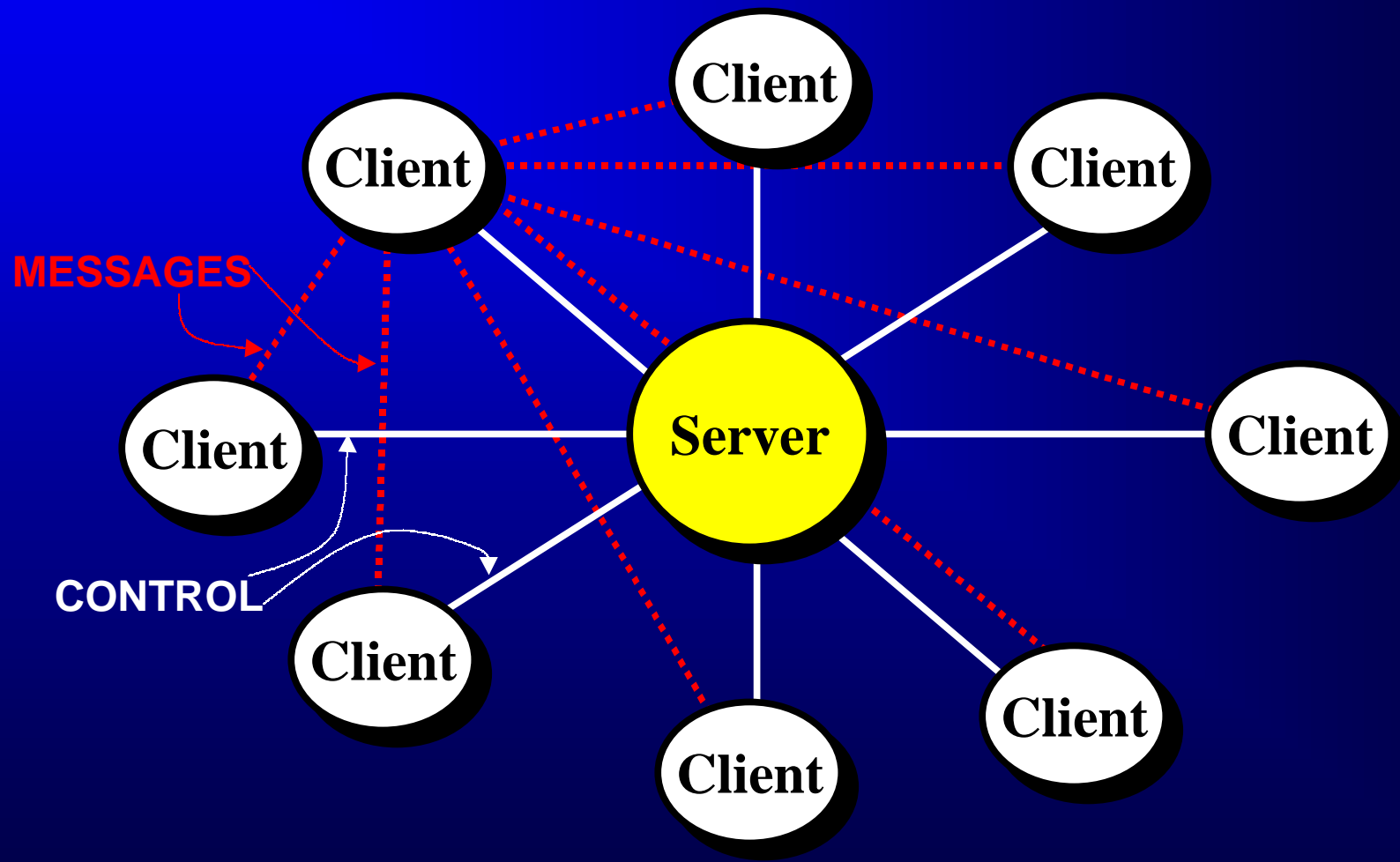
Client/Server



Client/Server

- Server is well known.
- Life is easier for clients - don't need to know about all other clients.
- Limited number of clients?
- Security is centralized.
- Server might get overloaded?

Hybrid Possibility



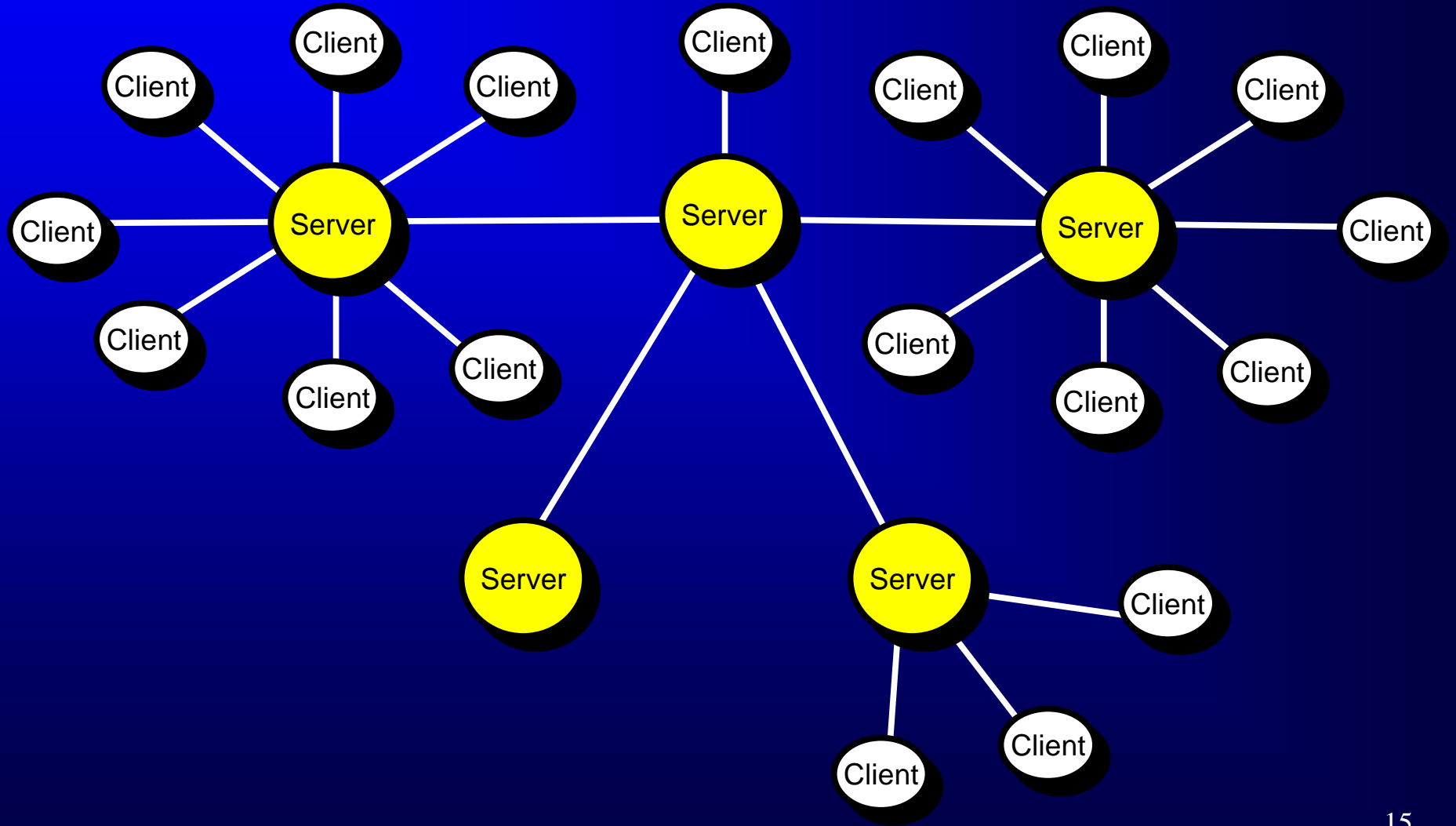
Hybrid

- Clients connect to server and gather control information:
 - List of other clients.
 - List of chat groups.
- Messages are sent directly (not through server).
 - Could use connectionless protocol (UDP or transaction based TCP).

Internet Relay Chat

- IRC is a widely used multi-user chat system.
 - Supports many chat groups (channels).
 - Extensive administrative controls.
 - Distributed service architecture.
 - Still in use today, although WWW based chat is now more common.

IRC Architecture



Server Topology

- Servers are connected in a spanning tree
 - Single path between any 2 servers.
 - New servers can be added dynamically
 - support for preventing cycles in the server graph.
- A collection of servers operates as a unified system, users can view the system as a simple client/server system.

Server Databases

- Each server keeps track of
 - all other servers
 - all users
 - all channels (chat groups)
- Each time this information changes, the change is propagated to all participating servers.

Clients

- A client connects to the system by establishing a TCP connection to any server.
- The client registers by sending:
 - (optional) password command
 - a nickname command
 - a username command.

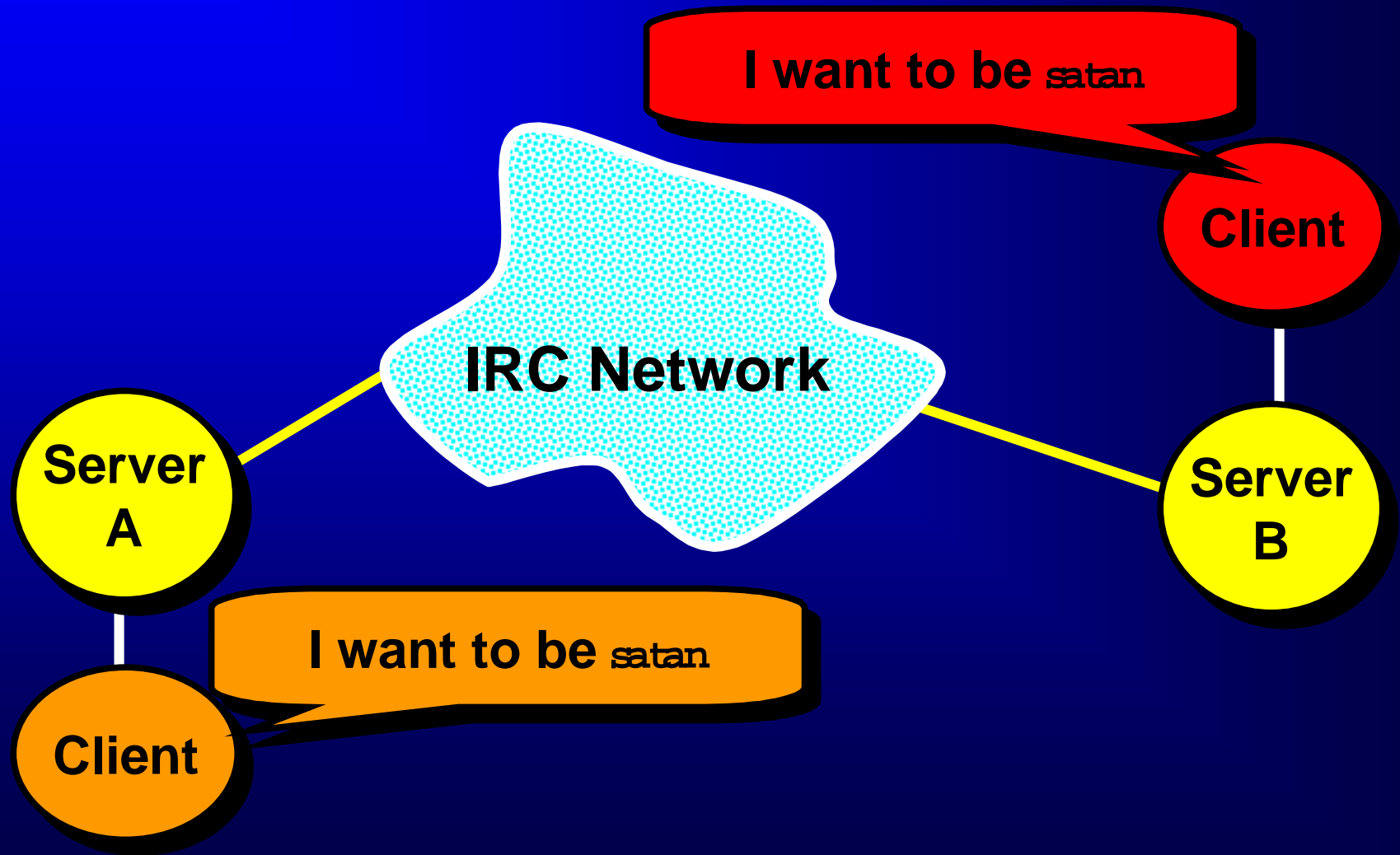
Nicknames and user names

- A nickname is a user supplied identifier that will accompany any messages sent.
 - Wizard, kilroy, gargoyle, death_star, gumby
- The username could be faked, some implementations use RFC931 lookup to check it.
- Users can find out the username associated with a nickname.

Collisions

- If a client requests a nickname that is already in use, the server will reject it.
- If 2 clients ask for the same nickname on 2 different servers, it is possible that neither server initially knows about the other.
- In this case both requests for the nickname are rejected.

Nickname Collision



Nickname Propagation

- The command used to specify a nickname is forwarded from server to all other servers (using the spanning tree topology).
- The command is the same, but extra information is added by the original server:
 - server name connected to client with nickname.
 - Hop count* from the server connected to the client.

*hop count is IRC server count (not IP!)

Channels

- 2 kinds of channels
 - local to a server - start with ‘&’ character
 - global, span the entire IRC network -start with the ‘#’ character.
- Users can JOIN or PART from a channel.
- A channel is created when the first user JOINS, and destroyed when the last user PARTS.

Channel Operators

- The user that creates a channel becomes the channel operator and can set various channel properties (modes):
 - invite-only
 - moderated
 - private
 - secret

Channel Op commands

- A Channel Op can:
 - give away channel op privileges
 - set channel topic (just a string)
 - kick users out of the channel.
 - Invite a client to a channel
 - change channel mode

Messages

- All messages are text.
- A message can be sent to nicknames, channels hosts or servers.
- There are two commands for sending messages:
 - PRIVMSG: response provided.
 - NOTICE: no response (reply) generated.
Avoids loops when clients are automatons

Other Stuff

- Special class of users known as Operators.
 - Operators can remove users!
- Servers can be told to connect to another server (operators create the spanning tree).
- The tree can be split if a node or network fails - there are commands for dealing with this.

Problems

- Scalability: works well with quite a large IRC network, but needs to be changed to get much bigger.
 - Currently every server needs to know about every other server, every channel and every user.
 - Path length is determined by operators, an optimal tree could be generated automatically.

Problems

- Supporting a cyclic network (instead of a tree) could minimize disruptions.
- Need a better scheme for nicknames, too many collisions (everyone wants to be *satan!*)
- Current protocol means that each server must assume neighbor server is correct. *Bad guys* could screw things up.