

## SpaghettiNodes

This is the design for SpaghettiNodes. SpaghettiNodes is a peer-to-peer file sharing system. The system will be distributed, with one node per computer in the network. A node is a self-sufficient entity in the form of an executable that can send and receive files to / from other nodes.

The node-to-node connections are controlled by being based on a pre-existing social network. That is, the owners of the individual nodes must first know one another, whether before the setup of their node or as a result of connections made via the SpaghettiNodes' existing social network.

Every node contains the following classes. The term “class” is used liberally, in that the object-oriented aspect is not necessary insofar as the program remains modular with respect to the modules:

- A “sender” class – must be able to transmit a packet (or a vector of packets) via UDP.
- A “reciever” class – must be able to receive a packet (or a vector of packets) via UDP.
- A “packet factory” class – must be able to accept data of any acceptable format (i.e. file data, strings for chat, etc.) and construct a valid UDP packet for transfer.
- A “packet deconstructor” class – must be able to accept a valid UDP packet and deconstruct it, retrieving the data originally placed inside by the factory in its proper form.
- NOTE: Transmissions in SpaghettiNodes are encrypted.
- A “file library” class – must be able to store what files / folders have been added to the shares list, and properly translate between some form of file / share identifier and the file's contents.
- A “file disassembler” class – must be able to break down a file of arbitrary size into a set of uniform-length “chunks” for transfer.
- A “file assembler” class – must be able to accept chunks of a file of arbitrary size in an arbitrary order and be able to reassemble those chunks into a file identical to the file originally disassembled.
- A “hook” class – must be able to store the address, authentication pair (username / password or username / 256-bit key), and level of trust for a particular connection.
- A “hook list” class – must be able to store a list of hooks the user has created, as well as the status of the hooks: unlocked, locked, and connected.
- A “controller” class – must be able to accept valid commands from the user, handle invalid commands, and act as an interface between non-coupled modules of the system.

An overview of the life of a node. A newly created node is referred to as a child node. A child node can add/remove hooks to/from the hook list, attempt to connect to an existing node's hook, and add/remove files and folders to/from the node's share list.

Here's how the nodes will become connected:

- 1 - An existing node creates a hook with an authentication pair – a username and password that the two people have previously agreed upon.
- 2 - The other node (whether it has other connections or not) sends a connection attempt with the same authentication pair.

3 - A corresponding hook is created in the attempting node's hook list, and both hooks are locked, preventing future change of the authentication pair by either user.

4 - Instead, a new (far longer) passkey is exchanged over the new connection, so that successive connections will use the new pair.

5 - Addresses are added to the hooks – this adds a second layer of identity. If a node's IP changes, the other node is alerted – and must verify the change before the nodes can once again be connected.

6 - The level of trust value for each hook is set to 1, being a nominal connection. This is the most pliable of a hook's values, and can be changed at any time.

When a node is shutdown, the other is left with the locked hook, albeit disconnected. When a node starts up again, it sends connection attempts to all nodes corresponding to a locked hook on that node's list.

Now that the node has at least one connection, it is an adult node. In addition to the functionality of the child nodes, adult nodes can disconnect connected hooks, remove locked hooks, ask a connected node for a file list, ask a connected node for a file, and ask a connected node for an abstracted list (i.e. not all data corresponding to) of the nodes that node is connected to. This abstracted list is also maintained by the user of the node, and is a subset of the hooks that the node contains represented by names, etc. that the node's user provides.

Finally, these actions are controlled via the following levels of trust. Note that any “requests” made must be verified by the receiving node before the action succeeds.

0 - Either the user is not in the node's hook list, or the node has been lowered to 0, effectively blacklisting the node. A node with a level of trust of 0 will not receive any responses for messages and requests, cloaking the other node from the blacklisted node.

1 - The node's users can request file lists, but can make no other requests.

2 - The node's users can see file lists without verification and make file / node list requests.

3 - The node's users can see file lists, read files, and see the connected node list without need for verification.