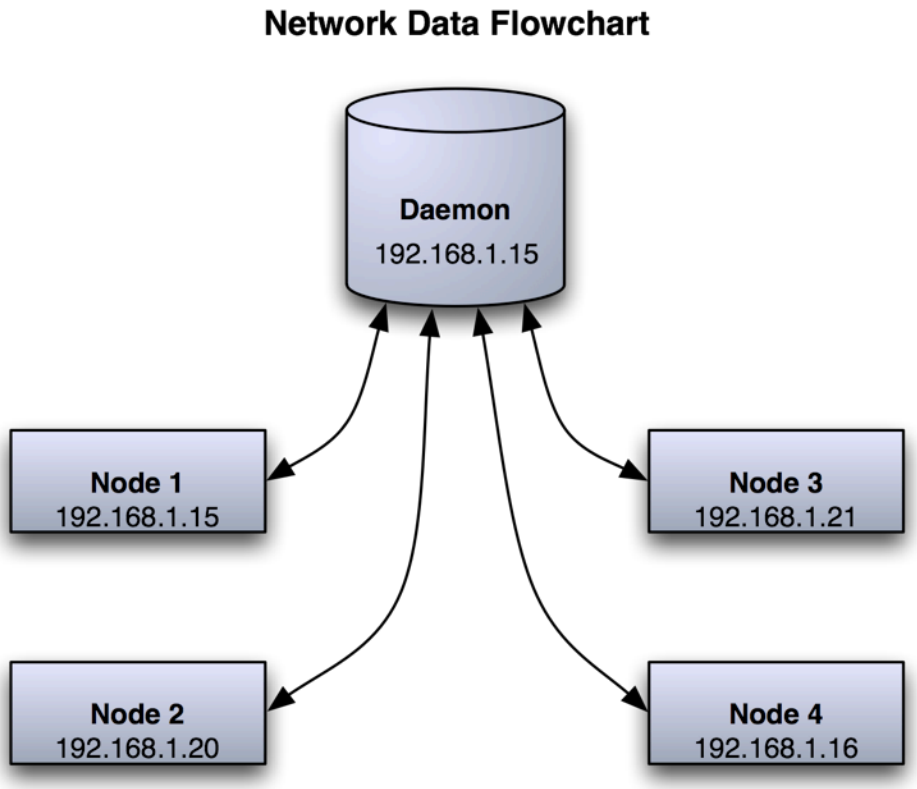
Client/Server Data Sharing Strategy

Overview:
This document describes a generic strategy for creating a distributed client/server architecture for sharing application data amongst multiple clients simultaneously.

Details:
In any client/server architecture, there are always 2 parts: The client and the server. In our case, the Client Server Application (hereafter CSA) will be broken down into the user application (hereafter “node”) and the database daemon (hereafter “daemon”) The principle job of the daemon is to provide a permanent repository for node data, and share data between nodes as changes are made on each node.

Diagram:
Here is a basic diagram that illustrates one possible network setup between nodes and the daemon. A single subnet is assumed. Static IPs are not required.

Note that Node 1 and the Daemon are on the same IP address (i.e., same physical computer). The daemon can be installed either alongside or independent of a node.



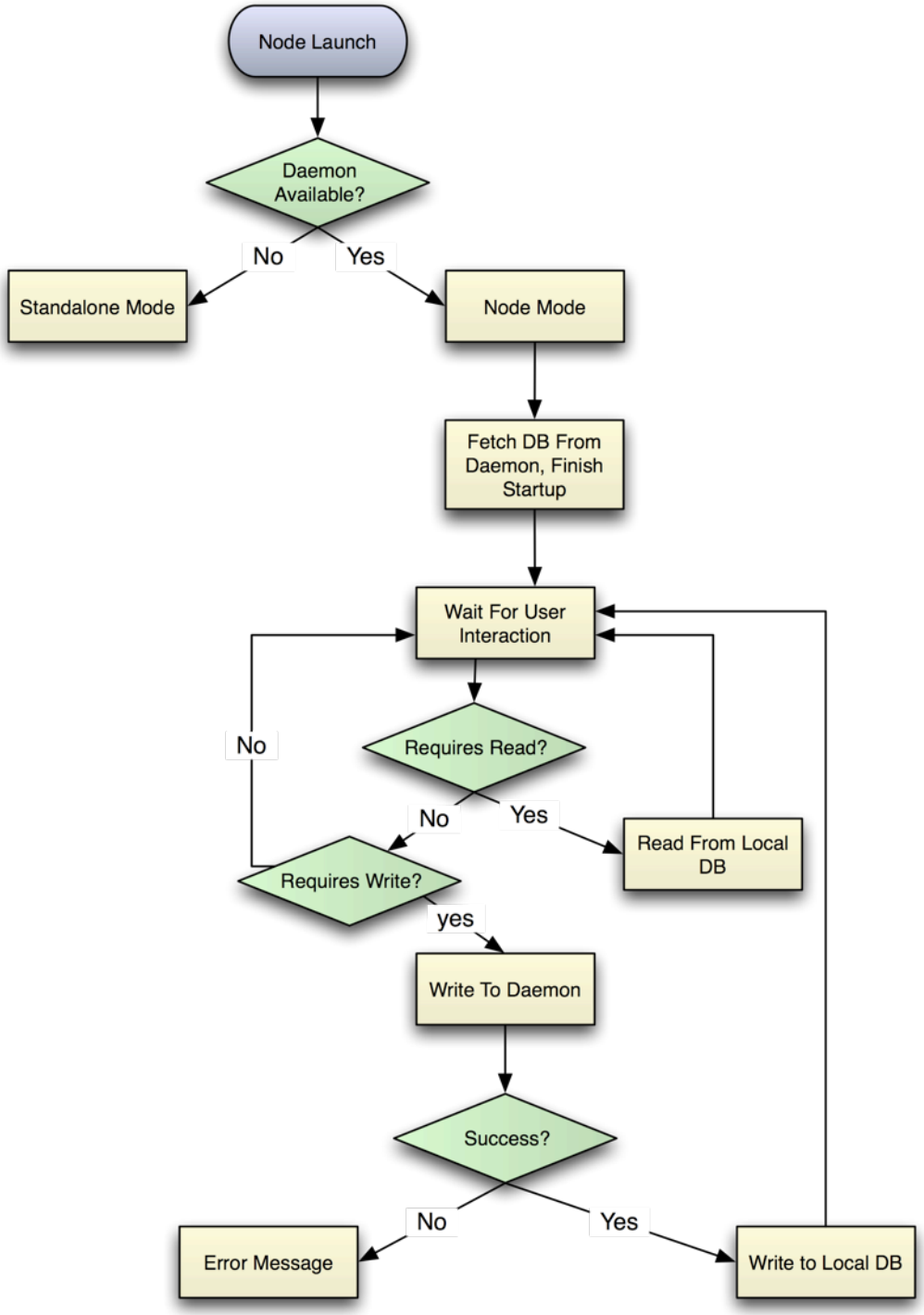
Data Processing - Node:
The Node will not keep a permanent copy of all the CSA data. Upon startup, it will fetch a current full copy of the CSA data from the daemon, which will be used for all read operations from the database. This local copy will persist after the node exits in case the next time the node is invoked there is no daemon available and the node must degrade to standalone operation. The local copy will be updated by the node anytime a node is notified from the daemon of changes on other nodes. Write operations to the database will be written from the node to the daemon first, and if successful then also to the node's local copy of the database. The daemon will then be responsible for propagation of the change to all other nodes.

Data Processing - Daemon:
The daemon is the warehouse for all application data, and also acts as a middleman between nodes. At node startup, the daemon will serve a copy of the full application database to the node. Whenever a node needs to save data to the database, it is sent to the daemon first, which saves it in the global database. Upon success the daemon informs all other nodes that new data is available, and informs the sending node that the save was successful. If unsuccessful, an error is returned to the node and handled there. The daemon will be able to respond to heartbeat requests from all nodes independently of any other data processing requests. This will be used by the node to ensure continued connectivity to the daemon.

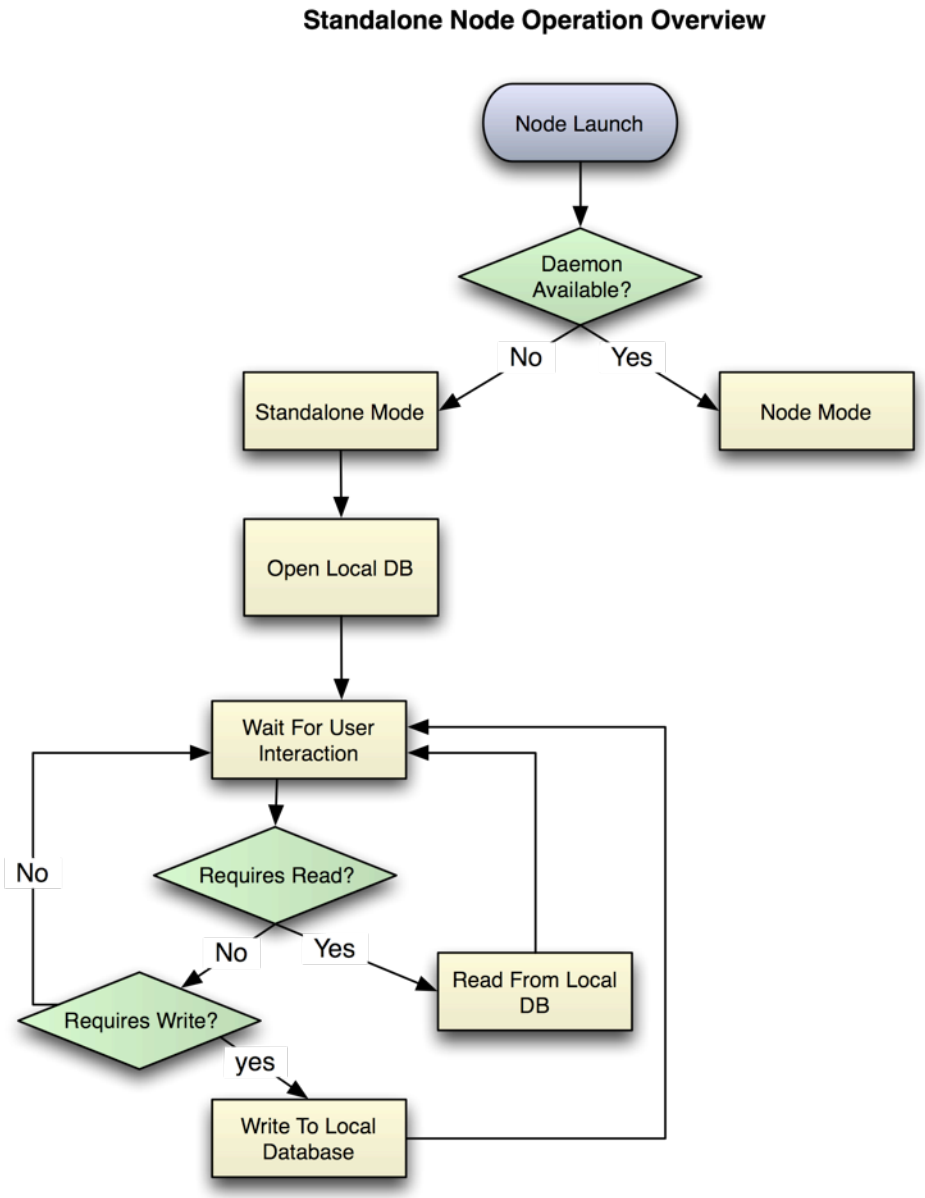
Changes to Traditional Application Behavior:
Traditionally, applications store all their data in a local database. With this design, the node needs to be able to operate in 2 modes: Node, and Standalone. As a node, it must locate and fetch a database from a daemon on the local subnet. As standalone, it will use the local copy of the database that was last fetched from the daemon, and keep track of any changes that are made so that when it is later connected to the daemon the changes can be added to the master database.

CSA "Node" Operation Flow and Diagram:
When a copy of CSA is started, it must first search for a daemon on the local subnet. If a daemon is found, it moves into "node" operating mode. In this mode it fetches a copy of the database from the daemon. All read operations happen on this local copy of the database. All writes are sent to the daemon first, and if the daemon reports success then they are applied to the local database as well. When other nodes write to the daemon, the daemon will notify all other nodes of the write operation. At any time write notifications can arrive from the daemon, so the node must always be able to process incoming write requests from the daemon. When the user is finished with CSA, the node simply shuts down, leaving the on-disk database file intact in case the next launch will be in standalone mode.

Network Node Operation Overview



CSA "Standalone" Operation Flow and Diagram:
When a copy of CSA is started, it must first search for a daemon on the local subnet. If none is found, it moves into "standalone" operating mode. In this mode, it locates a local copy of the database as left over from the previous time it was run (either as a node or standalone). If this database is not found, an appropriate error is produced and CSA exits. If it is found, this database is opened and used for this session. CSA displays a prominent notice on the interface that the user is operating in a non-networked node, and all data added will not be saved to their main database until they reconnect. When a standalone CSA is shut down, it leaves the on-disk database file intact in case the next launch will also be in standalone mode.



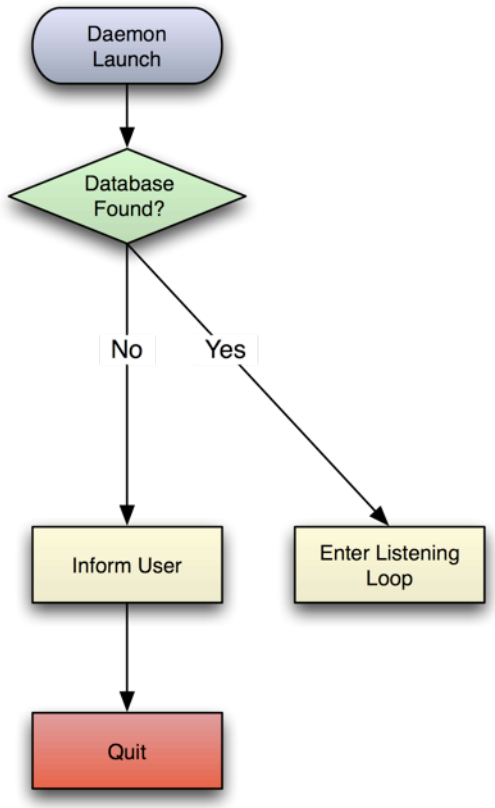
Daemon Startup Operation Flow and Diagram:
When the daemon starts up, it will attempt to locate the CSA database. If no database is found, the daemon must inform the user and offer corrective options. If a database is found, the daemon will open and use it.

Once a database is found and determined to be valid, the daemon moves out of startup mode and into its listening loop. It listens for the following events:

1. Node connects to network
2. Node leaves network
3. Node requests full database
4. Node publishes database update
5. Heartbeat from each currently connected node
6. Node publishes database update recorded when it was running standalone

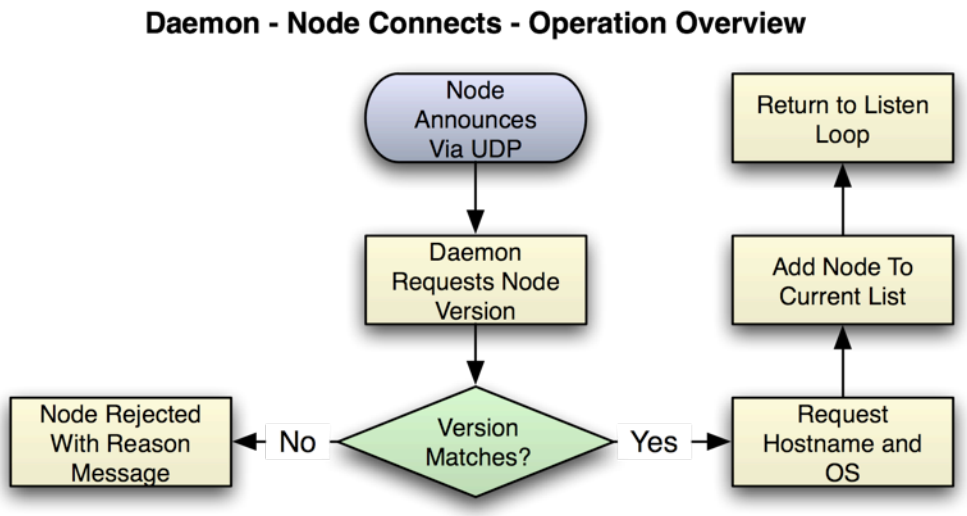
Each of these events triggers specific events in the daemon, described in detail below:

Daemon Startup Operation Overview



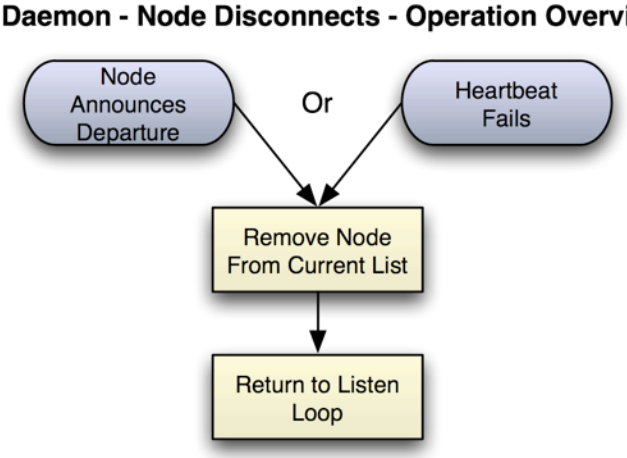
Node Connects To Network
When a node connects to the network, the daemon will receive a UDP broadcast message announcing the arrival of the node. The daemon will then request the version number of the node. If the version does not match the daemon version, the node is notified that it cannot connect due to version problems, and ignored until it announces again. If the version does match, the daemon requests the hostname from the node,

stores the result, adds the node to the list of currently connected nodes and returns to listen loop.



Node Leaves Network

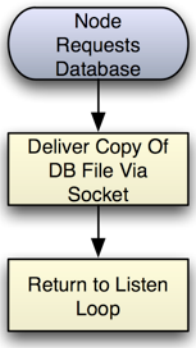
When a node leaves the network, there is little for the daemon to do but to remove the node from the list of currently connected nodes (i.e., pending updates no longer matter). It will be assumed that if a node is announcing that it is leaving, it is because it is being shut down. If a heartbeat fails, the node will not be removed until 30 seconds of no heartbeat has elapsed.



Node Requests Full Database

After a node first connects, and it has been added to the list of active nodes by the daemon and processed any pending standalone transactions, the next thing that it should do is request a full copy of the current database from the daemon. The daemon should respond by sending a full copy of the database file to the node as quickly as possible, as the node has to wait to finish startup until this operation is completed.

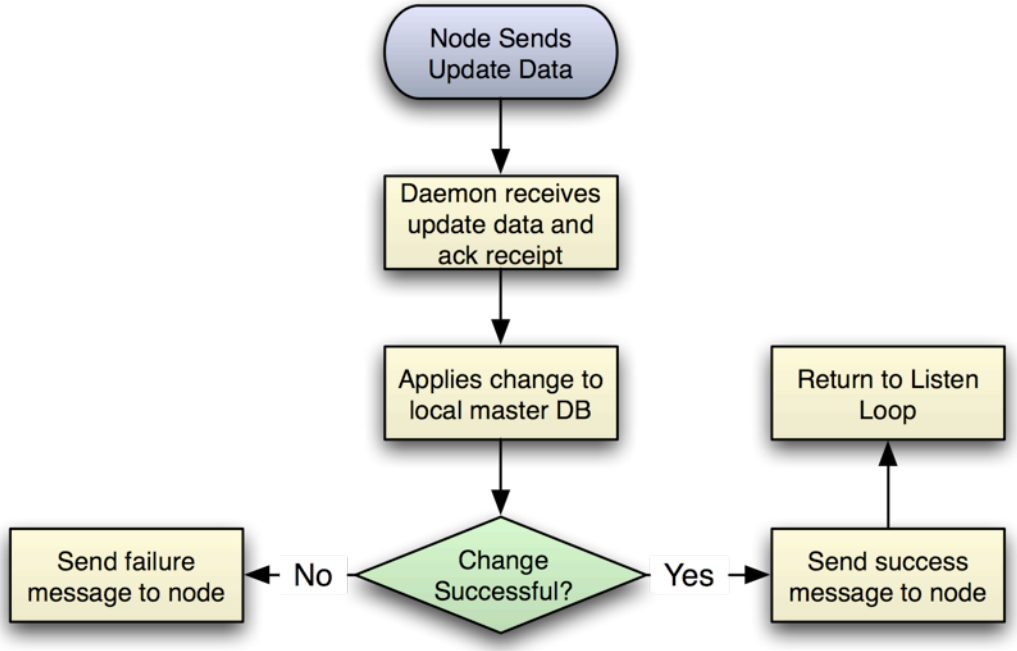
Daemon - Node Requests Full Database - Operation Overview



Node Publishes Database Update

As noted above, whenever a node needs to write data to its own database, it must first send the update information to the daemon. The daemon will then apply the change to the master copy of the CSA database, and send confirmation of a successful change or failed change back to the node. The node will then either make the change locally or report the error from the daemon, if one occurred.

Daemon - Node Sends Update - Operation Overview



Heartbeat:

Periodically, the nodes will send a small message to the server to indicate that they are still connected. It is suggested that this be sent every 10 seconds. If the server does not receive a heartbeat when it was expecting one, it is assumed that the client has become disconnected. The server will continue to listen for up to 30 seconds after the last heartbeat before it removes the node from the list of active nodes. If the node is

unable to get a response from the server for a heartbeat, the node assumes it has lost connection to the server immediately and warns the user, then shuts down.

Node Publishes Standalone Database Update:
When a copy of CSA is started and there is no connection available to the daemon, the node runs in standalone mode. This can occur when a satellite office is disconnected, a user takes a laptop traveling, etc. When CSA runs in standalone mode, instead of sending each database update to the daemon before executing on the local database, all transactions are run against the local database and a record of which rows were affected in which tables is kept. When the node next connects to the daemon, before it can request a full database, it must publish these changes to the daemon. The node will provide the database table name, row id, and row data (including a timestamp to the second from when it was last modified) for each modified row. The server will have to analyze this data and make some decisions. Specifically, the server will have to determine if the data presented will require an insert of new data or an update to existing data. If the data requires an insert, it is simply inserted. If it requires an update, the server checks the timestamp of the master copy of the data, and uses the most recent one. The logic for determining if the data presented requires an update or insert is not included here, as the rules for what should be updated will depend on your particular application and use cases.