Document date: 04/15/09. For most current version see: https://seattle.cs.washington.edu/wiki/NodeManagerDesign.

Node Manager Design Document

This document details the design and implementation of the Node manager. The purpose of the node manager is to manage the different sandboxed programs (called vessels) that are running on a computer. The node manager stores information about the vessels it controls and allows vessels to be started, stopped, combined, split, and changed. This document describes vessels, the node manager interface for manipulating them, advertisement of vessels, and how secure communication is provided.

There are several issues that are punted to V0.2. These include communication when behind a firewall or NAT, malicious users overwriting advertisement information, transfer / translation of restrictions when splitting and merging vessels, and the hooks necessary to build a resource trading system (Bellagio, SHARP, BACKS, Tycoon, etc.) on top of this. I believe it's best to get the basics working and solve any problems there.

Vessels

A vessel is a controlled environment for running code (implemented using the repy sandbox). Programs that run in vessels are prevented from performing unsafe actions or consuming undue resources. A single node manager typically manages (i.e. controls the resources assigned to) many vessels at the same time. Vessels have well defined boundaries that prevent them from interfering with one another (for example, different vessels may be provided their own disjoint set of network ports). Each vessel has a restrictions file, a stop file, and a log associated with it. The restrictions file lists what the vessel can and cannot do (enforced by repy). The stop file allows the node manager to stop the vessel (by creating a file with that name). The log is a circular logging buffer that the vessel writes to which can be read by the vessel owner.

A common use of vessels would be that a researcher obtains a vessel (let's suppose upon the installation of software). They then may decide to run a program within their vessel. To do this, they add the program they want to run to their vessel (along with any data files). They then may start the program running in the vessel. They can monitor the status of their program by looking at a status indicator (coarse grained monitoring) or by downloading information about the program from its circular log buffer (fine grained monitoring). They can also stop their vessel (if the need arises). If their vessel provides a useful service to other vessels, they can indicate this by entering information into the node manager about the service. The researcher can locate the vessels they control because their vessels advertise their availability in OpenDHT. The researcher can also prevent the vessels from advertising as a way to prevent their limited space for advertisements from being overwritten.

A more complex example is that a party (let's say an instructor) begins with a vessel on a node and decides to split it into separate vessels (so as to allow different students to run programs). They split the vessel (perhaps multiple times) and assign the vessels to

different students in the class. The students are allowed to work in groups and so some of the students decide (once groups are formed) to combine their vessels on the node so as to get more resources in a single vessel (note that there is nothing parallel or distributed about the node manager, it only worries about the node it runs on).

Node Manager Interface

The node manager provides an interface that allows manipulation of vessels and information gathering about the host computer. The node manager keeps a list of the vessels it manages. Each vessel has the following information:

vesselname
A string that uniquely identifies the vessel. These strings are assigned by the node manager and are unique to the vessel. Splitting and joining vessels result in new vessel names, but stopping and starting a vessel do not.

ownerkey
A public key that is allowed to change the ownership or use of the vessel. The owner can perform any action on the vessel

userkey(s)
A list of public keys that are allowed to issue a subset of the API calls

resourcefile
This is the resources and restrictions file for the vessel

status
What the execution status of the vessel is. Allowed values are: Fresh (has never been started), Stopped (was running but stopped by NM command), Started (has been started and is running when last checked), Terminated (the vessel stopped of its own volition, possibly due to an error), or Stale (it last reported a start of "Started" but significant time has elapsed, likely due to a system crash)

ownerinformation
This contains opaque data about the vessel that the owner defines. This information is flushed anytime the vessel's owner key changes. The information field is a field that only the current owner could have set and may be used to advertise a service, etc.

advertise
This boolean indicates whether or not the vessel should be advertised in announce services like OpenDHT. It is set to true when the owner changes.

logfile
This is the circular log file that the vessel uses

stopfile
This is a file that the vessel checks and if it exists, the vessel exits. This is used to stop a vessel.

oldmetadata This is needed for replay attack, freeze attack, etc. prevention

There is also a special set of resources called the offcut resources. The offcut resources are the amount of capability lost when a vessel is split into two vessels and the amount of capability gained when two vessels are joined. It accounts for the management overhead of monitoring a vessel.

The interface is:

GetVessels() -- public
Returns the vesselname, owner key, advertise flag, status, user key(s), and ownerinformation for every vessel (including vessels which do not advertise in OpenDHT)

GetVesselResources(vesselname) -- public
Returns the resource file for a vessel

GetOffcutResources() -- public
Returns the offcut resources

StartVessel(vesselname, args) -- private to owner, user
Begins executing a vessel with a set of arguments (including the command name).

StopVessel(vesselname) -- private to owner, user
Stops the execution of a vessel. Does not clean up the state for the vessel.

AddFileToVessel(vesselname, filename, filedata) -- private to owner, user
Create (overwrite if it exists) a file called "filename" in the vessel with contents "filedata". This operation can fail if the file system of the vessel is too small.

RetrieveFileFromVessel(vesselname, filename) -- private to owner, user
Returns the contents of a file in the vessel.

DeleteFileInVessel(vesselname, filename) -- private to owner, user
Deletes a file in a vessel.

ReadVesselLog(vesselname) -- private to owner, user
Returns the vessel's log.

ListFilesInVessel(vesselname) -- private to owner, user
Returns a list of files (space separated) in the vessel.

ResetVessel(vesselname) -- private to owner, user
Removes all files in a vessel's file system, resets the log, and stops the vessel if it's running. The advertise status is not changed.

ChangeOwner(vesselname, newpublickey) -- private to owner
Change the owner of a vessel to a different key. Also resets the ownerinformation to the empty string.

ChangeUsers(vesselname, listofpublickeys) -- private to owner
Change the list of public keys selected for the vessel (the list may only have a small number of entries)

ChangeOwnerInformation(vesselname, informationstring) -- private to owner

Sets the information string to a specific value. There is a limit on the value and longer strings will be truncated.

ChangeAdvertise(vesselname, boolean) -- private to owner
Should this vessel be advertised in OpenDHT?

SplitVessel(vesselname, resourcedata) -- private to owner
Splits a vessel into two smaller vessels. The resource data determines the size of one of the new vessels (the other is the original - the offcut). Both vessels are considered new and are given new vesselnames. The owner key is copied from the existing vessel. The restrictions are all removed and must be readded by the owner. The vesselname, filesystems and logs of the vessels are newly created. Returns the names of the new vessels (separated by a space).

JoinVessels(vesselname1, vesselname2) -- private to owner
Merge two vessels into two one larger vessel. The resource information is determined by adding the resources in the two vessels along with the offcut resources. A new vessel is created with a new vesselname. The restrictions are written so that any action that either could have performed can be performed by the created vessel. The created vessel is considered new and is given a new vesselname. The owner key must have previously been identical on both vessels and will be copied to the new vessel. The filesystem and log of the new vessel is freshly made. Returns the new vesselname.

SetRestrictions(vesselname, restrictiondata) -- private to owner
Sets the restrictions for a vessel. The restrictions may only contain "deny" entries.

Advertising Vessels

Every minute, the node manager inserts a key / value pair into OpenDHT in order to allow parties to find the nodes where they control vessels. The key that is inserted is the owner's public key and the value is the local computer's IP address. This means that a party can lookup their public key and find the nodes with vessels they control without needing to search for nodes.

However, there exist several unsolved problems that are punted to v0.2:

1) clients may not be contactable if they are behind a firewall or NAT
A good solution to this is likely to be non-trivial because of the different NAT implementations / quirks

2) preventing a malicious user from overwriting the information in openDHT.
This seems easy to fix but may require changes to openDHT

Secure Communication

After digging into this, I decided to "roll my own" because an XMLRPC client is going to be a huge headache to port to the sandbox. I've already ported RSA and SHA so I'm not too far away. I've built a "signeddata" module that handles many types of attacks (replay, freeze, misdelivery, and out of order).