OBSIDIAN 0.1 DOCUMENTATION

Clement Game
clement(at)digi-nation(dot)com

Keywords: Peer-to-Peer,Fast Rplication,mirroring,Dexp,Linux

Contents

1 Introduction

This is the documentation of the Obsidian Peer-to-Peer mirroring Software, version 0.1. This paper is released under CC-by-SA licence.

In ancient times, People was using a volcanic stone, which amount of

2 Overview

Obsidian is a fast-mirroring, Peer-to-Peer software which was specially designed to maintain large file repositories synchronized with each others, featuring very fast changes propagating.

To achieve this, Obsidian use 2 core technologies: Filesystem Events Listening , and a new protocol specially designed for the occasion: DEXP, acronym for Documents EXchange Protocol. With DEXP, Change Notifications are sent directly to the mirroring peers, so they don't have to poll periodically for file changes.

To illustrate how fast is Obsidian compared to a classical multiple rsync-based system, we can do the math:

Let's consider a chain of n peers, each node being synchronized to the next with a periodical rsynch check (T_c).The time taken by each node to download a change is T_d. In this case, the worst-case propagating time will be:

$T_r = \sum_0^n T_d(n) + T_c(n)$

As we can see in this case (rsync) , the more nodes we have, the more time it loses, because of the accumulation the T_c components.

But in the other case (Obsidian) , the calculation of the maximum propagating time between n nodes is:

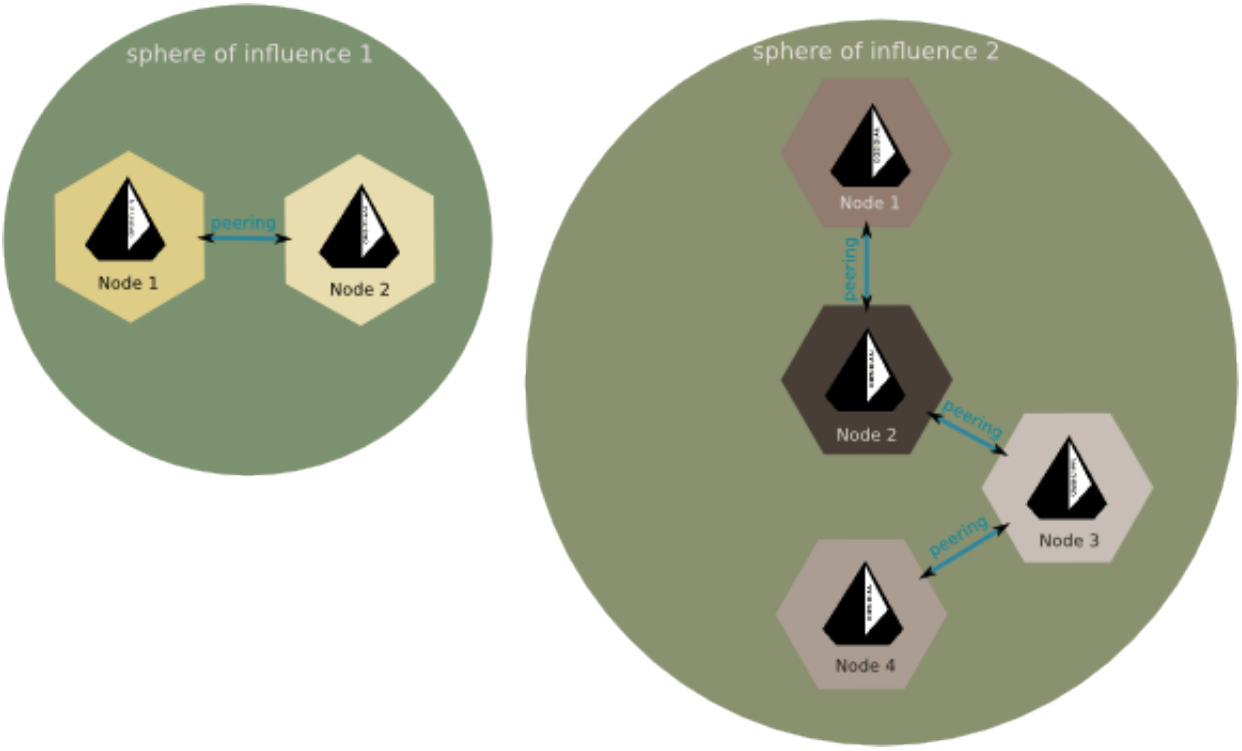
$T_r = \sum_0^n T_d(n)$

So with obisidan, The more nodes you haves, the more efficient it is compared to a periodic-poll based method.If we now consider a T_c constant for each node, we can evaluate the speed gain with:
$T_g = nT_c$
So the bigger is n, the bigger is the gain.

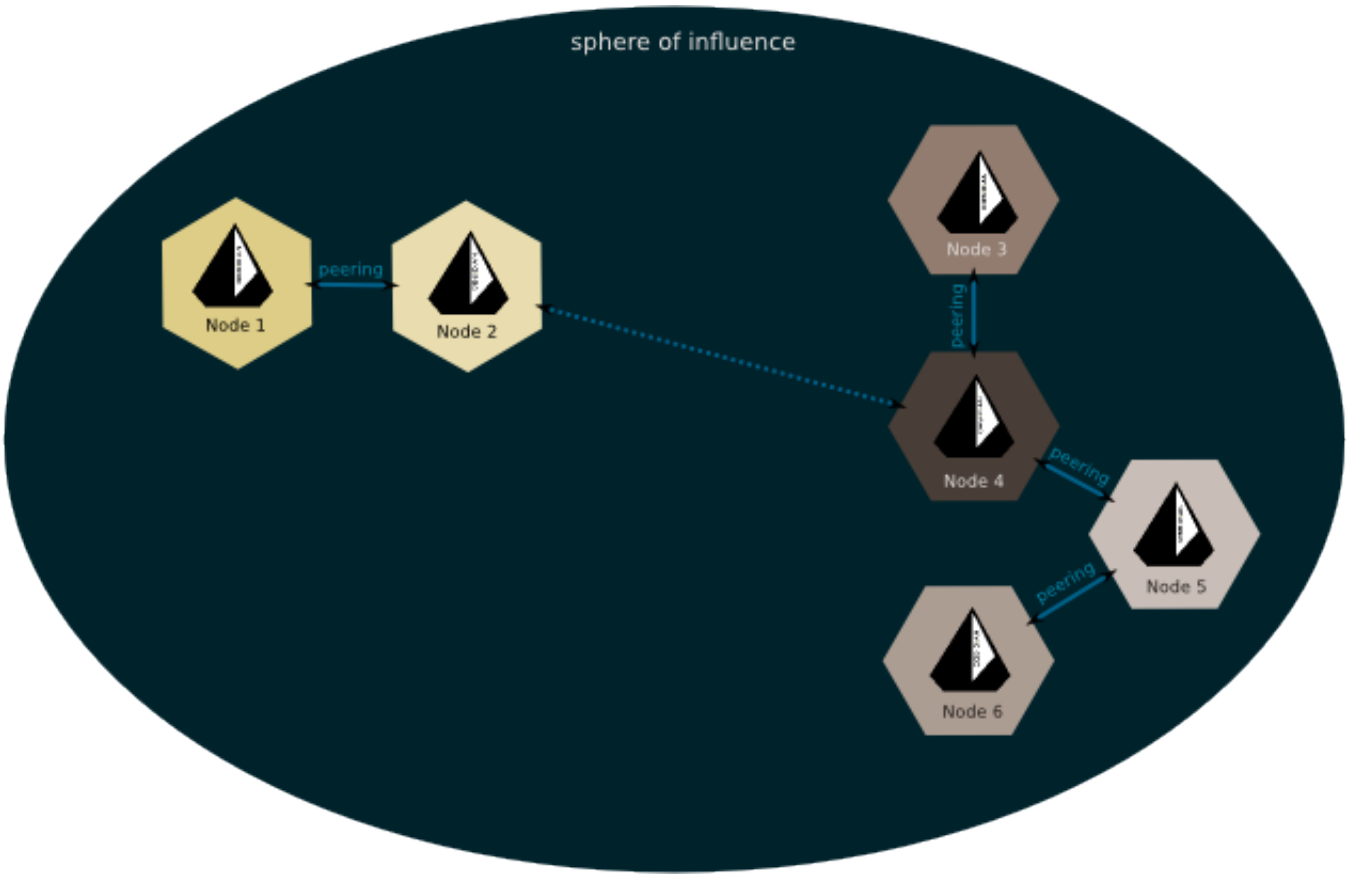
3 Global Architecture

This section explains how Obsidian works from a global point of vew:

Obsidian nodes are logically grouped by what we call 'Spheres of influence'. Each Obsidian node inside a sphere of influance receive the same announces, and all the nodes will end-up hosting the same files database.
2 distinct spheres of Influence appear when nobody belonging to sphere 1 has a peering connection to anybody Belonging to sphere 2



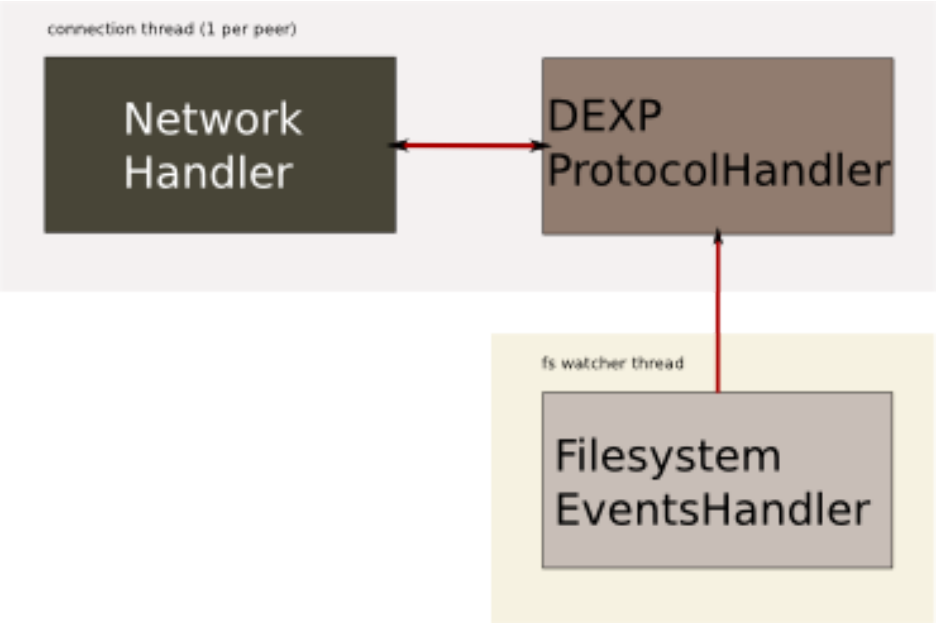
Now let's make the model evolve. Node 2 from sphere 1 peers with node 2 from sphere 2: Since These 2 nodes will share announces, the 2 spheres will merge.



From an ensemblist point of view, to describe the nodes behaviours while peering, we can write the following axioms:
$n1 \epsilon S1$
$n2 \epsilon S2$
$peering(n1, n2) => n1 \epsilon (S1 \cup S2); n2 \epsilon (S1 \cup S2)$

4 Software Architecture

The Software architecture behind Obsidian is quite easy, and can be described with the following scheme:



Obsidian uses a multi-threaded architecture along with blocking sockets. This is probably the less efficient part of the software, because this model is very expensive, but this way it was was really quick to prototype something that works.
A futur version of obsidian will surely come with a fully asynchronous model, at least for connections handling, as soon as we'll have time to redesign it.

Obsidian makes use of two other threads: a watchdog for current connections, and the filesystem events handler.

5 Getting Started with Obsidian

This section describes in detail how to get started with Obsidian, step-by-step and from scratch.

5.1 prerequisites

In order to build and run Obsidian, you'll need:

- A GNU/Linux Oprating System, with kernel 2.6.19 or higher.
- The OpenSSL Library plus its headers (libssl-dev) , 0.9.8 or higher.
- Cross-platform make (cmake 2.6 or higher).
- The GNU C compiler (gcc-4.X).
- Git DVCS(to retrieve the source code, if not done yet)

5.2 Fetching the sources

Here we're assuming that that you haven't fetched the obsidian source code yet. To do that, you will need the GIT DVCS, a well-known .
Once you have git installed on your system, just type:

```
git clone git://github.com/digination/obsidian.git
```

5.3 Compilation

This step explains how to build Obsidian from sources, with the use of cmake and gcc.

5.3.1 Producing the Makefile

This is cmake's duty to help you create a correct Makefile, suitable for your platform (even though up to know, obsidian is only compatible with Linux :P).
In order to produce your Makefile, you'll just have to use the following command at the root of obsidian's project tree :

```
cmake -DCMAKE_INSTALL_PREFIX=/usr/local/obsidian .
```

Also note that you can of course change the directory where obsidian will be installed, by modifying the CMAKE_INSTALL_PREFIX variable in the command above.

5.3.2 Let's roll, baby !

Now that the Makefile for obsidian has been created, we can lunch the actual compilation step, with the command:

```
make
```

5.4 Installation

Ok now that the compilation process is ok, let's install obsidian on the Systme, according to the PREFIX variable declared earlier.To install Obsidian, Nothing is more simple. From the Obsidian project Root Directory, just type the following:

```
make install
```

5.5 Configuration

6 Advanced Usages

6.1 Setting up TLS on your obsidian Server

6.2 Running multiple sessions of Obsidian on the same server

In order to circumvence the zero-dir limitation, you can choose to run multiple instances of obsidian at the same time on your server, so your peers can replicate data from the multiple instances.

7 More on DEXP

This section explains in details how the underlying protocol created for obsidian,DEXP, actually works.

7.1 A few notes to start

7.1.1 Client/Server relationship

Like every Peer-to-Peer protocol, Obsidian acts both as a server for some peers, and as a client for others, and so there is no distinction between binaries, there's only "obsidian".

7.1.2 Structure

DEXP is mainly a text-based protocol, as can be HTTP, FTP, IMAP and many more; meaning that it's based on a specific, human-readable vocabulary.

7.2 Steps

Here we will describe in a detailed way the differents step of a DEXP session, from the beginning.

7.2.1 Connection

7.2.2 Capacities

Once the TCP session has been established between two obsidian servers, the peer acting as a server will sent its capacities, consisting of a set of informations allowing the session between the two peers to be correctly negotiated.
These informations are:

7.2.3 Session Negotiation

Once the server sent its capacities to the client, the client will then choose which protocol version to use, comparing the received protocols list to its own one, choosing the higher matching one. The the protocol version is choosen, the client will send a "NEGOTIATE" command followed by the protocol version it choose, as argument.

7.2.4 TLS Negotiation (optionnal)

If the server announced TLS capacities, the client will systematically try to negotiate a TLS session with its peer, using a "STARTTLS" command (like many other protocols, doing the same). If TLS is correctly configured on the server, the negotiation will completed and the trafic between the two peers will then be ciphered.

7.2.5 Catalogs Exchange and files Synchronisation (optionnal)

Once the session and security layer have been negotiated, the client will start sending it's own files catalog to the host

7.2.6 Announces propagation

7.2.7 Files transferts

7.2.8 Announces delaying

If an announce has to be sent to a peer but this peer is currently busy sending or receiving data, the announce is placed in a queue and sent once the peer becomes available again. a peer notices its availability by sending a "READY" command to the server it was previously sending/receiving data to/from.

8 Known Bugs, Limitations

There is no knwon bug at this time, but the situation will change for sure.
Also the main limitation of Obsidian the "zero-dir" issue, meaning that it's filesystem events handler cannot detect changes located in subdirectories located inside the data_dir. That's why, for now, Only the root of the data directory will be replicated. (subdirectories are simply ignored).
This is the main change expected for version 0.2: A recursive fs events handler.

9 Special Thanks

My special thanks go to "Le Loop" Hackerspace in Paris and their fellow members, for providing me their support, advices and a place to crash and work on my project.