

GOTORRENT

Gossip-Based BitTorrent

Josep Sugrañes Riera
Sergi Canet Vela

INDEX

1. Especificacions del projecte.....	3
1.1. Requeriments específics	4
1.2. Tasques a realitzar i mètode d'avaluació.....	4
2. Disseny	4
2.1. Gràfiques de rendiment.....	5
3. Annex. Codi en Python.....	7

1. Especificacions del projecte

Aquesta pràctica es basa en implementar un prototip simplificat de BitTorrent mitjançant les funcions de la llibreria proporcionada PyActor treballada a classe (<https://github.com/pedrotgn/pyactor>). Pel que fa a la disseminació de fitxers, es demana utilitzar el protocol de comunicació de tipus gossip, el qual s'utilitza en molts sistemes distribuïts moderns degut a les seves propietats de simplicitat, robustesa i control descentralitzat.

La idea de funcionament del protocol BitTorrent és que l'arxiu a compartir es divideix en chunks, i els peers van descarregant l'arxiu tant del seed (peer que té la còpia completa de l'arxiu) com de la resta de peers.

El swarm d'un fitxer torrent el forma el conjunt de peers que està descarregant i compartint aquell mateix fitxer. El tracker s'encarrega de rastrejar tots els peers del swarm i mantenir-los o excloure'ls d'aquest.

Per unir-se al swarm, primer el client ha de descarregar-se un metafitxer que conté la URL del tracker, el tamany del fitxer, el nombre de chunks i altres metadades. D'aquesta manera, el client estableix connexió amb el tracker i, posteriorment, entra al swarm.

Per a dur a terme la comunicació gossip, de manera periòdica (cada cicle gossip) es seleccionen un o més peers aleatòriament del nombre total de peers al swarm i es du a terme una operació de comunicació. Aquesta comunicació pot ser de tipus push (on els nodes que tenen una peça d'informació la disseminen entre la població), pull (on els nodes seleccionen peers de manera aleatòria per demanar informació) o híbrida (push i pull alhora), depenent de l'estil de gossip.

Se'ns demana, primerament, implementar un tracker que tindrà dos mètodes: `announce` i `get_peers`. El mètode `announce` rep per paràmetre el ID del fitxer torrent i la referència del peer que s'anuncia. Els peers hauran d'anunciar la seva presència periòdicament, i quan un peer no s'hagi anunciat en un període de 10 segons el tracker l'eliminarà del swarm. El mètode `get_peers` rep per paràmetre el ID del fitxer torrent i ens retornarà un nombre fixat de peers que participen a la descàrrega seleccionats aleatòriament del swarm.

Segonament, també es requereix que els peers utilitzin comunicació gossip per a distribuir els chunks, utilitzant la funcionalitat intervals de la llibreria PyActor, la qual serveix per permetre que un actor periòdicament faci una acció. Indiferentment de l'estil gossip escollit, cada node selecciona un nombre fixat de peers de manera aleatòria al principi de cada cicle gossip amb el mètode `get_peers` del tracker. Aquest mètode rep per paràmetre el ID del chunk i el contingut d'aquest per poder-lo enviar. En el cas de comunicació push, els peers seleccionen un chunk aleatòriament i l'envien als peers que ha retornat el mètode `get_peers`. En el cas de comunicació pull, els peers demanaran un chunk als peers que ha retornat el mètode `get_peers`. Aquest mètode rep per paràmetre el ID del chunk el qual vol rebre. Es necessari tenir en compte que podria ocórrer que un peer no té el chunk que se li ha demanat.

1.1. Requeriments específics

- a) El sistema ha d'estar format per 1 tracker, 5 peers i 1 seed.
- b) La durada d'un cycle gossip serà de 1 segon.
- c) Inicialment, només el seed contindrà el fitxer i els peers no tindran cap informació.
- d) El sistema haurà de distribuir almenys la següent cadena de text: "GOTORRENT", on cada caràcter serà un chunk (9 chunks en total). Aquesta cadena de text serà llegida a partir d'un fitxer, on l'ID del chunk serà la posició dels caràcters de la cadena.
- e) Un cop feta la disseminació del fitxer, tots els peers hauran de tenir una còpia completa del fitxer.
- f) Cada peer haurà d'anunciar-se al tracker cada 10 segons. Abans de que la disseminació del fitxer comenci, cada peer, inclòs el seed, s'haurà d'haver anunciat.
- g) El mètode `get_peers` retornarà 3 peers a random dels 6 disponibles al swarm.

1.2. Tasques a realitzar i mètode d'avaluació

- a) Tracker (30%). Implementació del tracker i comprovacions necessàries per verificar el correcte funcionament d'aquest.
- b) Comunicació push (35%). Implementació del protocol de comunicació gossip de tipus push. Per comprovar la eficàcia d'aquest mètode construir una gràfica on l'eix d'abscisses sigui el nombre de cicles gossip i l'eix d'ordenades la llargada de la cadena de text.
Respondre: *Explicar que podeu veure a la gràfica. Hi ha molts missatges redundants?*
- c) Comunicació pull (25%). Implementació del protocol de comunicació gossip de tipus pull. Fer l'experiment anterior per obtenir una nova gràfica.
Respondre: *Explicar que podeu veure a la gràfica i comparar-la amb la del push. Quina de les tècniques és més eficient? Per què?*
- d) Comunicació híbrida (5%). Combinar les dues implementacions anteriors per obtenir comunicació push-pull (híbrida) i obtenir la gràfica corresponent.
Respondre: *Explicar que podeu veure a la gràfica. Hi ha molta diferència amb la gràfica del pull?*
- e) Repositori github (5%). Crear un repositori personal amb el codi i la documentació. És recomanable incloure proves de code coverage i unitary tests.

2. Disseny

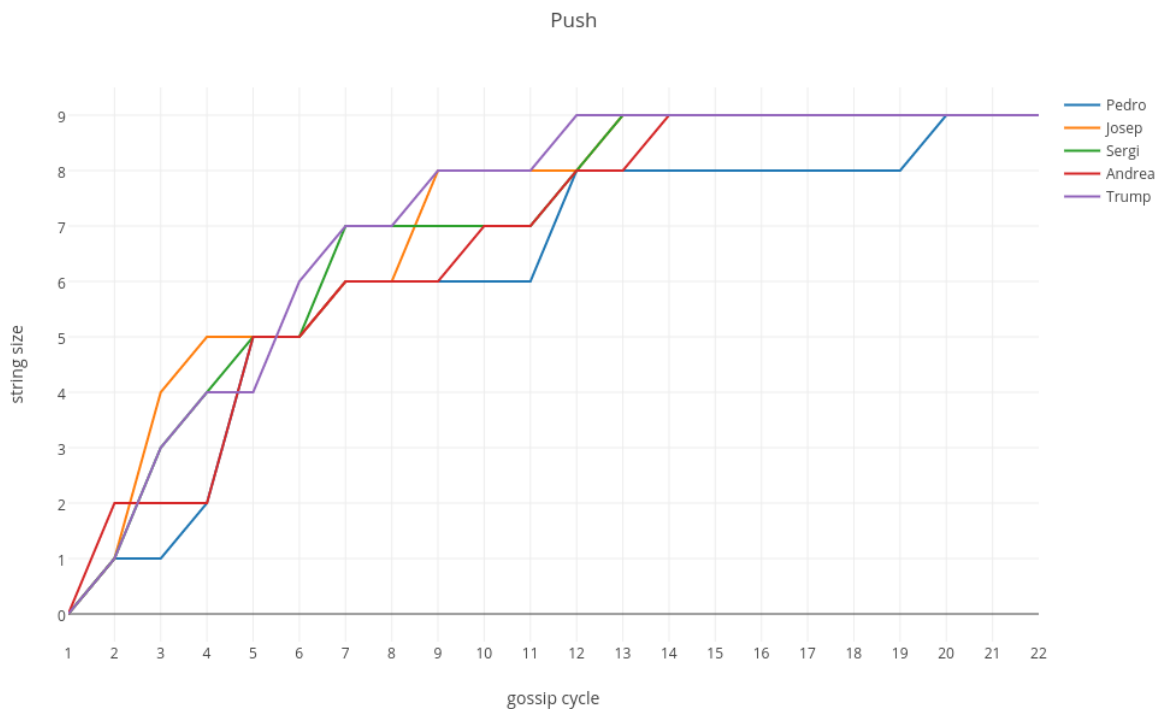
Hem decidir crear 2 classes: Tracker i Peer. Addicionalment tenim 2 classes més: Print (per imprimir per pantalla) i Main (on fem les proves per comprovar que el projecte funciona correctament).

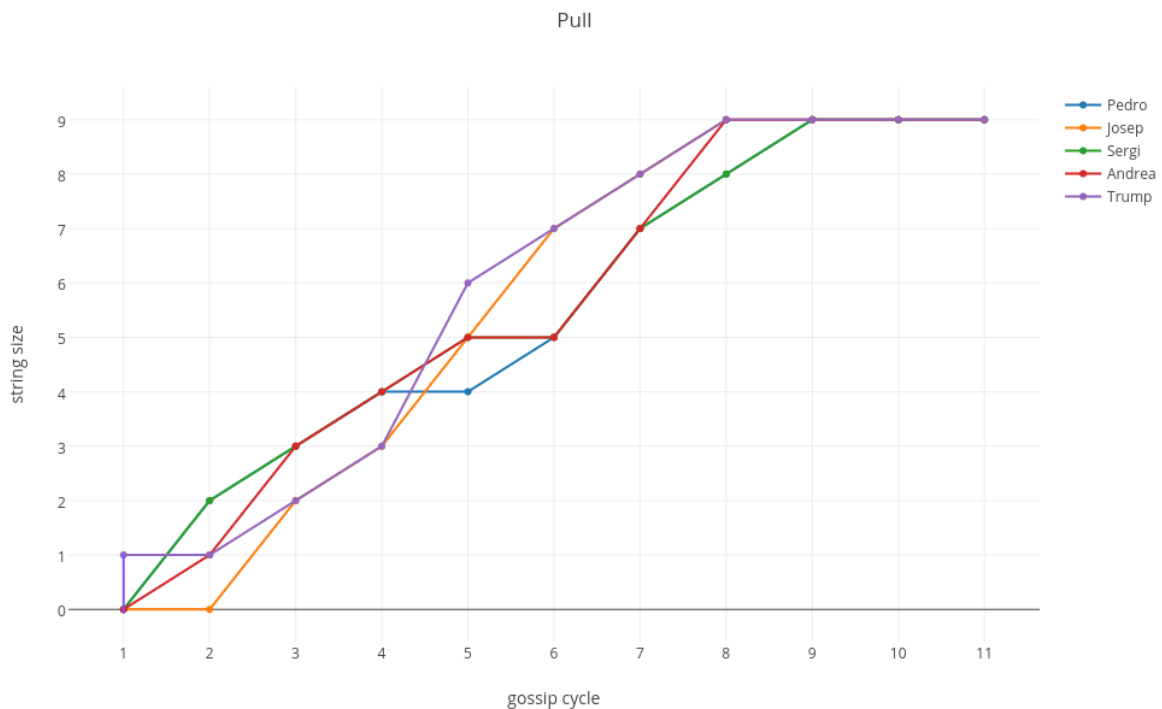
La classe Tracker conté els mètodes `announce` i `get_peers` (el qual és un mètode síncron, ja que esperem que ens retorni 3 peers) i s'encarrega de cada segon comprovar que no hagin passat més de 10 segons desde l'últim `announce` per cada peer que hi ha al swarm amb el mètode `càlcul_time`. En cas contrari, l'elimina. Està formada per un diccionari que conté l'ID del torrent i en el valor un altre diccionari amb l'ID del peer i l'últim temps que ha anunciat. També tenim el mètode `init_intervals`, el qual s'encarrega de fer les crides cada segon al mètode `càlcul_time`.

La classe Peer conté els mètodes attach (per unir-los al tracker), announce_peer (que crida al mètode announce implementat a la classe Tracker), seed_fitxer (el qual llegeix el fitxer amb la cadena de text a disseminar), init_start (el qual cada 3 segons demana fer un announce al peer i cada segon (és a dir, cada cicle gossip) farà la crida a una comunicació gossip del tipus que s'hagi rebut per paràmetre. Per tant, aquesta classe també implementa els mètodes donar_lletra i demanar_lletra que s'encarreguen de cridar als mètodes push i pull, respectivament.

Finalment, en quant a la observació que es detallava a les especificacions que podria ocórrer que un peer no tingui el chunk que se li està demanant, nosaltres hem decidit crear una excepció al mètode demanar_lletra la qual permet continuar al programa en cas que no es pugui executar correctament el mètode pull de manera que no hàgim rebut res. En canvi, això no pot passar amb el mètode donar_lletra, ja que sempre donarem lletres que tenim, però sí que pot passar que la lletra que donem ja la té l'altre peer.

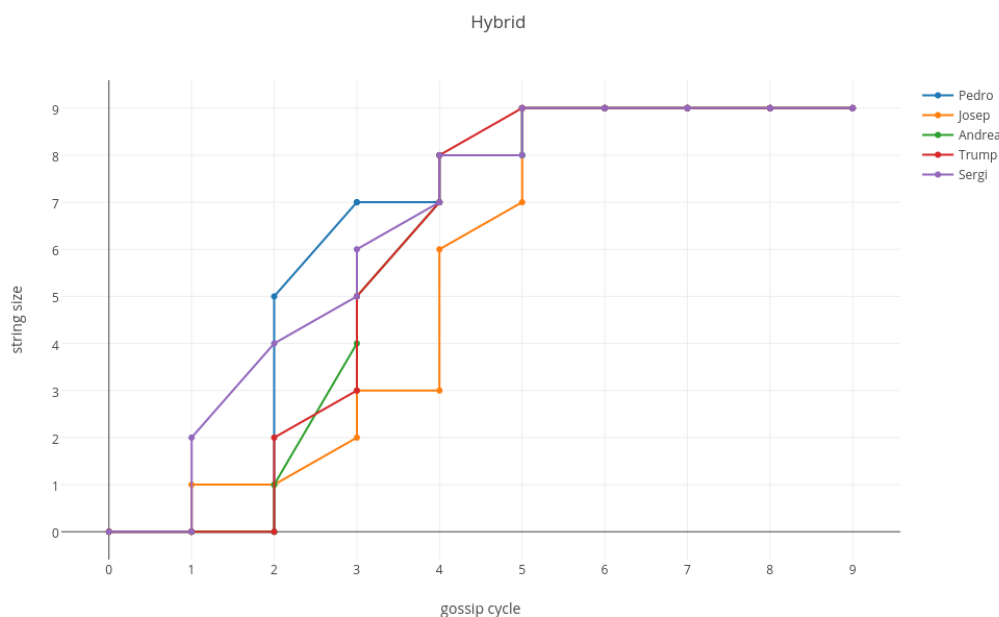
2.1. Gràfiques de rendiment



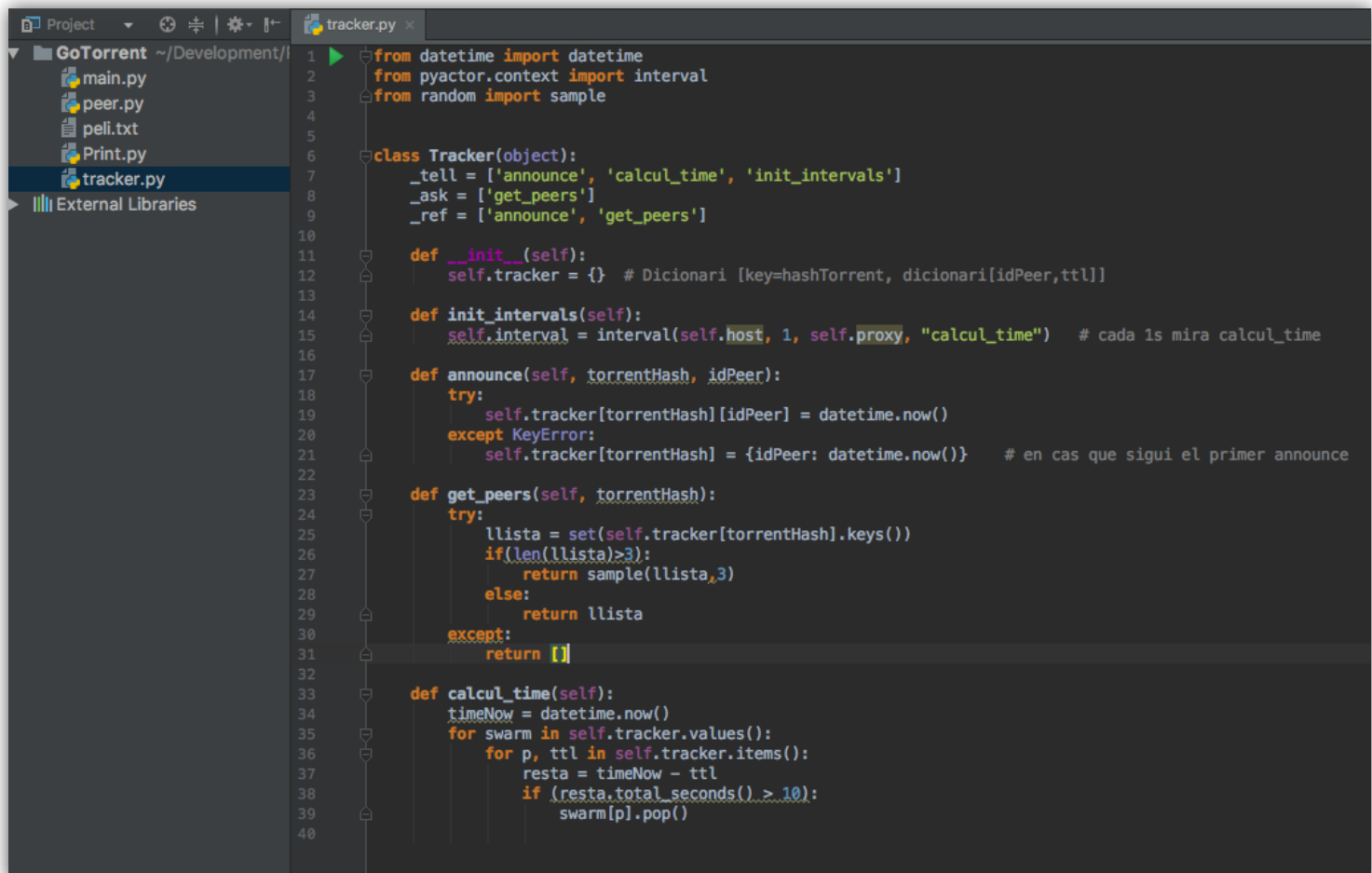


Com podem observar, la comunicació pull és més eficient, ja que amb 9 cicles enlloc de 20 tots els peers ja tenen la cadena de completa. Però cal notar que durant els primers cicles és més eficient el push (ja que la majoria de vegades que un peer dona un chunk al principi, el més probable és que els altres peers encara no el tinguin), però amb l'avanç del temps es torna ineficient perquè cada vegada hi haurà més comunicacions redundants. Llavors es quan la comunicació pull ens resulta més apropiada ja que es demana un chunk específic (i quants més cicles gossip han passat, és més probable que els altres peers tinguin aquest chunk en concret). En el cas de la comunicació pull, doncs, els missatges redundants es troben durant els primers cicles.

Per tant, la conclusió teòrica és que el mètode híbrid serà el més eficient de tots ja que obtindrem els millors temps dels dos mètodes i, tal com podem comprovar a la següent gràfica, es compleix.



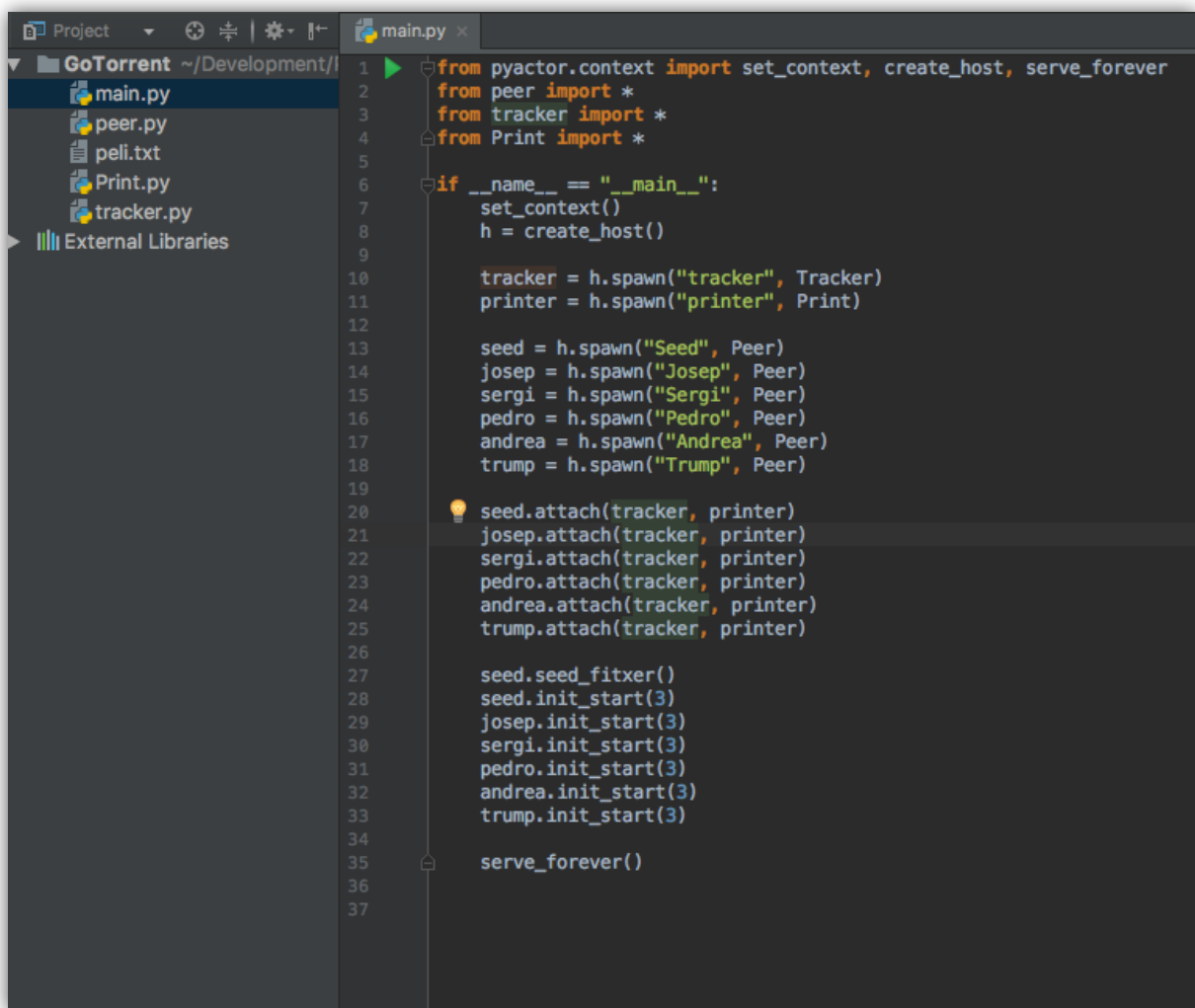
3. Annex. Codi en Python



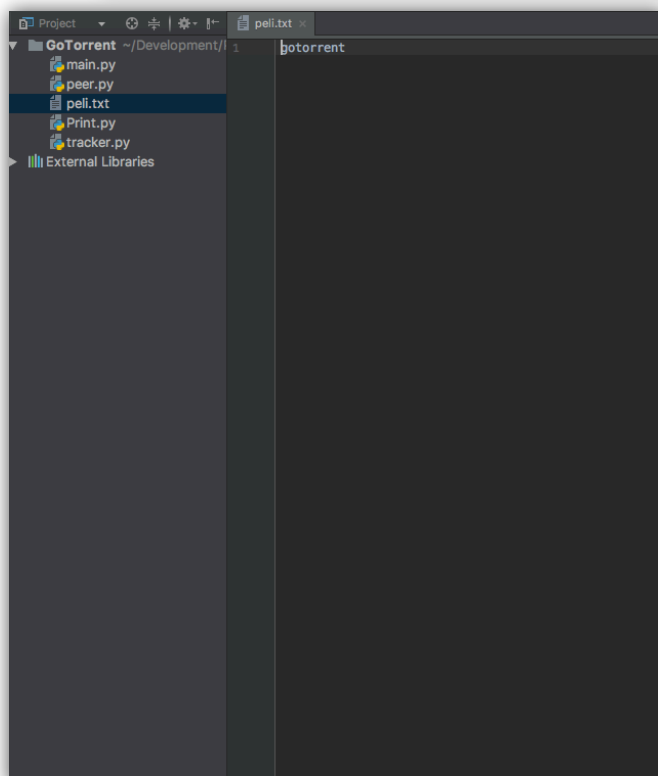
```
1 from datetime import datetime
2 from pyactor.context import interval
3 from random import sample
4
5
6 class Tracker(object):
7     _tell = ['announce', 'calcul_time', 'init_intervals']
8     _ask = ['get_peers']
9     _ref = ['announce', 'get_peers']
10
11     def __init__(self):
12         self.tracker = {} # Dicionari [key=hashTorrent, dicionari[idPeer,ttl]]
13
14     def init_intervals(self):
15         self.interval = interval(self.host, 1, self.proxy, "calcul_time") # cada 1s mira calcul_time
16
17     def announce(self, torrentHash, idPeer):
18         try:
19             self.tracker[torrentHash][idPeer] = datetime.now()
20         except KeyError:
21             self.tracker[torrentHash] = {idPeer: datetime.now()} # en cas que sigui el primer announce
22
23     def get_peers(self, torrentHash):
24         try:
25             llista = set(self.tracker[torrentHash].keys())
26             if len(llista) > 3:
27                 return sample(llista, 3)
28             else:
29                 return llista
30         except:
31             return []
32
33     def calcul_time(self):
34         timeNow = datetime.now()
35         for swarm in self.tracker.values():
36             for p, ttl in self.tracker.items():
37                 resta = timeNow - ttl
38                 if (resta.total_seconds() > 10):
39                     swarm[p].pop()
40
```

```
Project ~ /Development/
GoTorrent
  main.py
  peer.py
  peli.txt
  Print.py
  tracker.py
  External Libraries

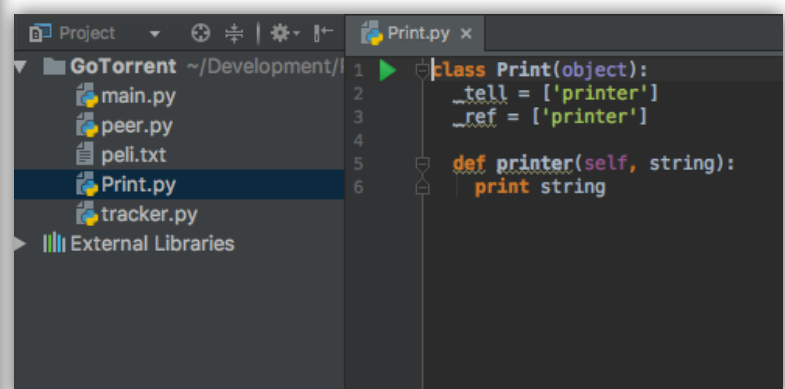
peer.py
1 from random import choice
2 from pyactor.context import interval
3
4 llargada_paraula = 0
5
6 class Peer(object):
7     _tell = ['attach', 'announce_peer', 'seed_fitxer', 'init_start', 'donar_lletra', 'push', 'demanar_lletra']
8     _ask = ['pull']
9     _ref = ['attach', 'init_start', 'push']
10
11 def __init__(self):
12     self.paraula = {}
13     self.torrent_hash = "peli.txt"
14
15 def attach(self, tracker, printer):
16     self.tracker = tracker
17     self.printer = printer
18
19 def announce_peer(self):
20     self.tracker.announce(self.torrent_hash, self)
21
22 def seed_fitxer(self):
23     with open(self.torrent_hash) as f:
24         for idx, ch in enumerate(f.readline()):
25             self.paraula[idx] = ch
26     global llargada_paraula
27     llargada_paraula = len(self.paraula)
28
29 def init_start(self, opcio):
30     self.interval = interval(self.host, 3, self.proxy, 'announce_peer')
31     if (opcio == 1 or opcio == 3):
32         self.interval1 = interval(self.host, 1, self.proxy, 'demanar_lletra')
33     if (opcio == 2 or opcio == 3):
34         self.interval2 = interval(self.host, 1, self.proxy, 'donar_lletra')
35
36 def donar_lletra(self):
37     for peer in self.tracker.get_peers(self.torrent_hash):
38         try:
39             lletra = choice(self.paraula.items())
40             peer.push(lletra[0], lletra[1])
41         except:
42             pass
43
44 def push(self, pos, lletra):
45     self.paraula[pos] = lletra
46     self.printer.printer(str(self.id)+str(self.paraula))
47
48 def demanar_lletra(self):
49     for peer in self.tracker.get_peers(self.torrent_hash):
50         for i in range(llargada_paraula):
51             if (i not in self.paraula.keys()):
52                 try:
53                     self.paraula[i] = peer.pull(i)
54                     self.printer.printer(str(self.id)+str(self.paraula))
55                 except:
56                     pass
57                 break
58
59 def pull(self, i):
60     return self.paraula[i]
61
```

```
1 from pyactor.context import set_context, create_host, serve_forever
2 from peer import *
3 from tracker import *
4 from Print import *
5
6 if __name__ == "__main__":
7     set_context()
8     h = create_host()
9
10    tracker = h.spawn("tracker", Tracker)
11    printer = h.spawn("printer", Print)
12
13    seed = h.spawn("Seed", Peer)
14    josep = h.spawn("Josep", Peer)
15    sergi = h.spawn("Sergi", Peer)
16    pedro = h.spawn("Pedro", Peer)
17    andrea = h.spawn("Andrea", Peer)
18    trump = h.spawn("Trump", Peer)
19
20    seed.attach(tracker, printer)
21    josep.attach(tracker, printer)
22    sergi.attach(tracker, printer)
23    pedro.attach(tracker, printer)
24    andrea.attach(tracker, printer)
25    trump.attach(tracker, printer)
26
27    seed.seed_fitxer()
28    seed.init_start(3)
29    josep.init_start(3)
30    sergi.init_start(3)
31    pedro.init_start(3)
32    andrea.init_start(3)
33    trump.init_start(3)
34
35    serve_forever()
36
37
```



```
1 btorrent
```



```
1 class Print(object):
2     __tell = ['printer']
3     __ref = ['printer']
4
5     def printer(self, string):
6         print string
```