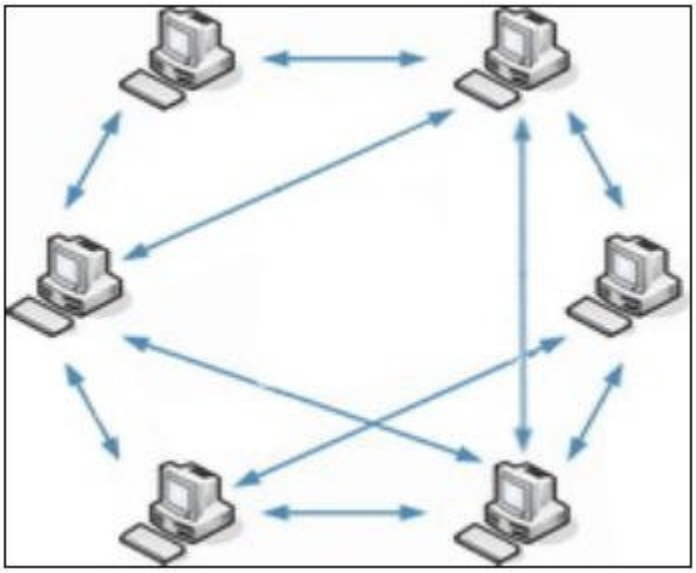
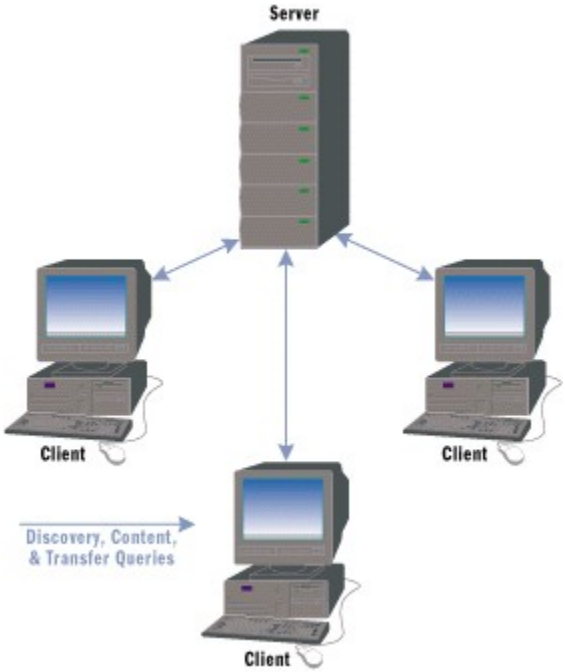
P2P Exposed

فى شبكات ال P2P بتكون الأجهزة مترابطة مع بعض بصورة معقدة شوية كالتالى مثلا



وبصراحة ال Model دا دراسته شيقة جدا بس للأسف هنتكلم عن ال Model دا :D



Server-Based Network وهو بيعتمد إن يكون فى Server يعمل بدور وسيط بين الأجهزة وبنطلق عليه Discovery Server او Indexer لأنه بيكون على Database

بتتخزن فيها معلومات كل Peer من حيث ال Shared Files و ال Alias و .. إلخ

ملحوظة: ال Model الثانى مش يعتبر P2P بصورة كاملة

اولا هنعمل Module بسيطة نعمل بيها Encapsulation للحاجات دى

```
import marshal

class PeerInfo(object):

    def __init__(self, alias, sharedFiles, listeningPort):
        self.alias=alias
        self.sharedFiles=sharedFiles
        self.listeningPort=listeningPort

    def __str__(self):
        sb="Alias: " + self.alias
        sb += "\nFiles: " + str(self.sharedFiles)
        sb +="\nListens At:" + str(self.listeningPort)
        return sb


def serialize(obj):
    '''Serialize an object to a string...'''

    return marshal.dumps(obj)

def deserialize(objString):
    '''Deserialize an object string...'''

    return marshal.loads(objString)


if __name__=="__main__":
    p=PeerInfo("ahmed", [1, 2, 3 ,4], 80)
    print p
    print "alias: ",p.alias
    print "files: ", p.sharedFiles
    print "port : ", p.listeningPort
```

ال peerInfo class هو class هنخزن فيه بيانات المستخدم عشان تكون اسهل فى التعامل والإستعلام وهى ال alias, sharedFiles, ListeningPort
هنعمل 2 methods بسيطين جدا لهندلة ال Objects اللى هتتبعت على السوكيت وهما serialize و deserialize

بصورة مبدأية لما نفكر فى ال Discovery Server لازم يكون فيه شوية مميزات
1- ان يتم تسجيل بيانات اى حد يعمل Connect والبيانات دى بتشمل Alias, SharedFiles, ListeningPort
2- نقدر نستعلم عن الملفات الموجودة على ال Clients الآخرين

3- كل Client يقدر يعدل على بياناته
4- يقدر يعرض كل ال Clients الآخرين
5- يقدر يعدل على ال SharedFiles
6- اقصى عدد يقدر يتصل بالسرفر
7- وطبعا لازم يقدر يتعامل مع اكتر من Client فى نفس الوقت
8- .....
وهكذا
ملحوظة: ممكن تعمل Clients ليهم صلاحيات اعلى "مثلا اللى مشترين pro للمنتج بتاعك"

اولا إحنا مش هنعمل DB لأننا مش منتج كبير .. احنا يدوب بنعرض الفكرة .. فال DB بتاعتنا هتكون عبارة عن Dictionary او HashTable او غيرهم حسب اللغة اللى إنت جاى منها :D

{'clientIP:port' : peerInfo Object}
اولا هنعمل socket object ونعمل set لل options بتاعته "اهم شئ REUSEADDR وهى بتسمحلك بإستخدام ال port مرة اخرى مباشرة فى حال إنك قفلت السرفر لسبب ما "وغالبا هنا الإختبار للبرنامج"
ملحوظة : مع إننا هندعم ال Chat بين المستخدمين ولكننا هنستخدم TCP مش UDP -لأننا هنتعامل مع نقل لملفات-

```
self.listener=socket.socket(socket.AF_INET, socket.SOCK_STREAM)
self.listener.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
```

ال Class هيبدأ بالصورة دى .. وهنحدد فيه ال port اللى هيشتغل عليه السرفر ..

واقصى عدد للمستخدمين والأوامر المدعمة
/register : لتسجيل ال client
/setSharedFiles : لتعديل ال ملفات اللى معمولها share
/setNick : لتغيير ال Nick او ال alias
/showall : لعرض جميع المستخدمين
/query : للإستعلام عن ملف ما

```
class DiscoveryServer(object):
   '''Indexer...'''
   def __init__(self, port, maxPeers=5):
      self.port=port
      addr=('', self.port)
      self.maxPeers=maxPeers
      eslf.supportedCommands=["/register", "/setNick", "/setSharedFiles",
"/showall", "/query"]

      self.listener=socket.socket(socket.AF_INET, socket.SOCK_STREAM)
      self.listener.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
      self.listener.bind(addr)

      self.tListening=threading.Thread(target=self.listeningHandler, args=[])
      self.tListening.start()
      self.alSocks=[]
     # {clientAddr:peerInfo Object}
      self.db={}
      self.log=[]
      self.BUF=2048
```

لاحظ مش هسجل log فى البرنامج .. هسيبهالك واجب او فرصة إنك تعدل شئ لشئ احسن وهكذا .. بحيث إنك تبقة مشارك
self.alSocks هى List هنضيف فيها كل ال Sockets المفتوحة "هتعرف بعد شوية"

لاحظ فى السطر دا

```
      self.tListening=threading.Thread(target=self.listeningHandler, args=[])
```

إننا انشأنا thread للميثود listeningHandler وال args اللى هنباصيها [] List فاضية لأنها مش بتاخد args
ولبدأ ال thread بنستخدم start method

```
      self.tListening.start()
```

ال implementation الخاص بال method دى

```
def listeningHandler(self):
    self.listener.listen(self.maxPeers)
    print "Server Started..."
    while True:
        clientSocket, clientAddr=self.listener.accept()
        print "Gotta a connection from", clientAddr
        tClientHandling=threading.Thread(target=self.clientHandler,
args=[clientSocket])
        tClientHandling.start()
    clientSocket.close()
```

بنخلى السرفر يبدأ فى عملية ال listening
وبعد كدا بنعمل forever loop بنتعامل فيها مع اى client بيعمل connect وبمجرد
مايعمل Connect نهندله فى thread جديد بإستخدام method بإسم clientHandler

```
tClientHandling=threading.Thread(target=self.clientHandler,
args=[clientSocket])
```

ملحوظة accept method بتدى return ب tuple مكونة من ال clientSocket, clientAddress

```
def clientHandler(self, clientSocket):
    self.alSocks += [clientSocket]
    formatedAddress=clientSocket.getpeername()
[0]+":"+str(clientSocket.getpeername()[1])
    objString=""
    try:
      while True:
          objString=clientSocket.recv(self.BUF)
          if not objString:
              break
          data=deserialize(objString)
          #print data
          tAnalyzeData=threading.Thread(target=self.analyzeData, args=[data,
clientSocket])
          tAnalyzeData.start()
```

```
            objString=""

        except Exception, e:
            print "E: ", e
            print clientSocket.getpeername(), " closed.."
            self.alSocks.remove(clientSocket)
            del self.db[formatedAddress]
```

ال method دى خاصة بالتعامل مع ال Client وكل اللى بتعمله فى الحقيقة هى إنها بتضيف ال ClientSocket إلى ال alSocks list اللى بال server class وتبدأ thread جديد تحلل فيه الداتا الجاية من ال Client
وطالما هتحلل data وفى thread جديد يبقة لازم نعمل method تبقة مسؤلة عن عملية ال تحليل او ال parsing لل داتا واكيد ال method دى هتاخد argument وهى ال data واللى ارسل الداتا وهو ال clientSocket

```
 tAnalyzeData=threading.Thread(target=self.analyzeData, args=[data,
clientSocket])
            tAnalyzeData.start()
```

جميل جدا .. تحليل الداتا فى ال analyzeData method

```
def analyzeData(self, data, clientSocket):
    formatedAddress=clientSocket.getpeername()
[0]+":"+str(clientSocket.getpeername()[1])
    try:
        if isinstance(data, tuple): #registering...
            pInfo=PeerInfo(data[1], data[2], data[3]) #(register, alias, files, port)
            print "Registering: ", pInfo.alias
            print pInfo
            self.db[formatedAddress]=pInfo #peerInfo object..
            print self.db

        if isinstance(data, list):
            try:
                #split the sender's alias..
                #recvd=['tina: /showall']
                recvd=data[0].split(": ")[1]
                cmd=recvd.split(" ")[0]
                # test cmd...
```

```
            if not cmd in self.supportedCommands:
                self.sendToAll(data, clientSocket)
            else:
                if cmd=="/showall":
                    self.showAllTo(clientSocket)
                if cmd=="/query":
                    fileName=recvd.split(" ")[1]
                    self.queryFile(fileName, clientSocket)
                if cmd=="/setNick":
                    self.setNick(formatedAddress, recvd.split(" ")[1])

        except Exception,e :
            print "Error: ", e


    except Exception, e:
        print "Data: ", data
        print "Error: ", e
        self.alSocks.remove(clientSocket)
```

انا اتبعت تكنيك بسيط شوية هنا وهو إنى بعت تسجيل البيانات على صورة tuple مكونة من

(“/register”, files=[], listeningPort)

وحولتها ل peerInfo object

وبعد كدا شوية إختبارات لل command نفسه لو كان query نعمل كذا لو كان كذا نعمل كذا وإذا مش كان موجود فى الأوامر المدعمة بال Server تبقة مجرد رسالة تتبعت لكل الأهل والأحباب :D

فى حال إن حصل اى ايرور يتم حذف ال Client من ال alSocks لنعرف إنه غير active او متصل بالسرفر حاليا

طب فى حال لو إن الداتا اللى إتبعتت مجرد مسج عادية ؟ يعنى هنحتاج نبعتها لكل المستخدمين ماعدا بالطبع اللى ارسلها مش كدا ؟

نجيب كل المستخدمين منين ؟

اها تمام من ال alSocks اللى بتعبر عن كل ال Clients المتفاعلين مع السرفر حاليا قشطة

```
def sendToAll(self, msg, clientEx):
    print "Message Recieved: ", msg
    try:
        for sock in self.alSocks:
            if not sock==clientEx:
```

```
            sock.send(serialize(msg))
        else:
            pass
    except Exception, e:
        print "Error: ", e
```

جميل جدا.. فى حال لو المستخدم عايز يستعرض المتسخدمين الموجودين "اكيد رد السرفر هيكون ليه لوحده مش كدا ؟"super
فهتكون ال showallTo method كالتالى

```
def showAllTo(self, clientSocket):

    data="\n--------------------------------------\nOnline Users:\n"
    for addr, pInfo in self.db.items():
        data += pInfo.alias + " -> " +addr +"\n"
    data +="\n--------------------------------------\n"
    print data
    clientSocket.send(serialize(data))
```

ملحوظة انا فضلت اعمل serialize لكل الداتا اللى هتتبعت حتى لو strings عشان مش اقعد اتعب نفسى فى ال de-bugging واعملهم de-serialize فى الطرف التانى

لهندلة امر تغيير ال Nick وهو /setNick

```
def setNick(self, to, newNick):
    self.db[to].alias=newNick
    print "Nick Changed..."
    print self.db[to]
```

جميل جدا ناقص ال Querying files
اللى هيطلب الإستعلام هو واحد مش كل الناس فلازم نباصى ال socket بتاعه فى ال method + نبعتله ال داتا

```
def queryFile(self, fileName, clientSocket):
    print "Querying: ", fileName
    data=""
    for addr, pInfo in self.db.items():
        if fileName in pInfo.sharedFiles:
            data += "\n"+addr + " | " + pInfo.alias + " => " + fileName
            data += "\n\t" + pInfo.alias + " Listens at: "+ str(pInfo.listeningPort)
    print data
```

```
clientSocket.send(serialize(data))
```

كدا انهينا ال Discovery Server او ال Indexer

super

ننقل على ال Client او ال Peer
ال Peer لازم يتفاعل مع ال Server و يقدر يحمل ملفات من ال Peers التانين و ان يكون فى peers يقدرو يحملو منهم بردو فكدا هنعمل client ليتعامل مع السرفر ونعمل internal server يكون مسئول عن عملية ال دونلود او ال fetching

ال Constructor بتاعه هياخد
1- alias
2- list of sharedFiles
3- ال endPoint الخاصة بالسرفر
ال endPoint هى ال Server IP + port

بمجرد مايتم ال connection مع ال server نهندله فى thread جديد

```
def __init__(self, alias, serverAddr=(), sharedFiles=[]):

    self.alias=alias
    self.serverAddr=serverAddr
    self.sharedFiles=sharedFiles
    self.tcpClient=socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    self.tcpClient.connect(self.serverAddr)
    self.BUF=1024
    self.listeningPort=rnd.randint(8081, 10000)
    self.pInfo=PeerInfo(self.alias, self.sharedFiles, self.listeningPort)


    print "\nConnected to server..."
    self.tClientToServer=threading.Thread(target=self.clientToServerHandler,
args=[])
    self.tClientToServer.start()
```

رائع جدا
بس بردو لازم نجهز لل Peers اللى هيحاولو يعملو fetch ل files من عندنا فهنعمل server object وبردو نشغله فى thread جديد قشطة؟

```
#listen for connections in background..
```

```
        self.addr=('', self.listeningPort)
        self.listener=socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.listener.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
        self.listener.bind(self.addr)

        self.tListening=threading.Thread(target=self.listeningHandler, args=[])
        self.tListening.start()
```

جميل بما إنى على جهاز واحد فهتحصل مشكلة انى مش هقدر اعمل set ل port معين اربطه بالسرفر لأن بكل بساطة كل ال Peers هيحاولو يربطو على السرفر دا .. فبكل بساطة هنستخدم randint method الموجودة ب random module ونجيب اى random int بس لاحظ إنه يكون فوق ال 1024 وبردو نبلغ بيه ال Discovery Server

```
self.listeningPort=rnd.randint(8081, 10000)
```

اولا ال registerAtServer Method

```
    def registerAtServer(self):
        msg=("/register", self.alias, self.sharedFiles, self.listeningPort)
        self.tcpClient.send(serialize(msg))
```

وهى اول ميثود هيتم إستدعائها بمجرد إنى اعمل connect على السرفر

```
def clientToServerHandler(self):

        print "Start Chatting.."
        #first register the files...
        self.registerAtServer()
        while True:
            tServerToClient=threading.Thread(target=self.serverToClientHandler,
args=[])
            tServerToClient.start()
            data=raw_input()
            if not data: continue
            if data.lower=="exit": exit()

            if data.split(" ")[0]=="/fetch":
                fileneeded=data.split(" ")[1]
                addr=data.split(" ")[2]
                tFetchFile=threading.Thread(target=self.fetchFile, args=[addr,
fileneeded])
                tFetchFile.start()
            else:
```

```
        msg=self.alias+": "+data
        self.tcpClient.send(serialize([msg]))
```

بما إن ال client مش ليه غير وظيفة واحدة بس اللى مش هتتعامل مع ال server وهى ال fetching فهنختبرها الأول إذا هى المطلوبة او لأ.. فى حال آه نبدأها فى thread جديد خاص بالتعامل معاها :D لو لأ تبقة مجرد رسالة او امر زى إستعلام او تغيير بيانات فهنبعته للسرفر والسرفر يحلله

نهدل رسايل ال server فى thread هيستخدم ال serverToClientHandler method

```
def serverToClientHandler(self):
    while True:
        data=deserialize(self.tcpClient.recv(self.BUF))
        if not data: break

        if isinstance(data, list): #data ['tina: hi']
            print data[0]
        else:
            print data
```

عملية ال fetching بكل بساطة هنتعامل معاها كالتالى

```
def fetchFile(self, addr, fileneeded):
    #addr is formated => addr:listeningPort
    endPoint=addr.split(":")[0], int(addr.split(":")[1])
    fetchTCPClient=socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    fetchTCPClient.connect(endPoint)
    fetchTCPClient.sendall(serialize(("/fetch", fileneeded)))
    tDownloadFile=threading.Thread(target=self.downloadFile,
args=[fetchTCPClient, fileneeded])
    tDownloadFile.start()
```

ونبدأ فيها thread جديد لل download بيستخدم ال downloadFile method و هيباصى ال tcpClient, الفايل المطلوب ك args ليها

```
def downloadFile(self, fetchTCPClient, fileneeded):

    f=file(fileneeded, "wb")
    while True:
        try:
```

```
            buf=fetchTCPClient.recv(self.BUF)
            if not buf:
                break
            f.write(buf)
        except EOFError, eofErr:
            print "EOFError: ", eofErr
        except Exception, e:
            print "Error: ", e
            break
    print "File Downloaded!"
    f.close()
    fetchTCPClient.close()
```

جميل جدا احنا كدا شبه خلصنا .. ناقص شئ واحد بس وهو التعامل مع ال clients اللى عايزين يحملو فايلات مننا -خليك فاكر إننا already مشغلين thread خاص بال listening للطلبات :)

```
    self.tListening=threading.Thread(target=self.listeningHandler, args=[])
    self.tListening.start()
```

جميل ال thread دا بيستخدم listeningHandler method وال method دى مش هتاخد args

```
  def listeningHandler(self):
    self.listener.listen(5)
    while True:
        clientSocket, clientAddr=self.listener.accept()
        tClientHandling=threading.Thread(target=self.clientHandler,
args=[clientSocket])
        tClientHandling.start()
```

محتاجين إنها تكون Multi-threaded عشان مش نربط نفسنا مع مستخدم واحد بس فهنعمل thread جديد يهندل كل client يعمل connect على ال internal server

```
  def clientHandler(self, clientSocket):
    rcvd=clientSocket.recv(self.BUF)
    data=deserialize(rcvd)
    if isinstance(data, tuple):
        if data[0]=="/fetch": #go on..
```

```
        fileneeded=data[1] #(/fetch, fileneeded, from)
        print "File Request: ", fileneeded
        f=file(fileneeded, "rb")
        while True:
            try:
                buf=f.read(self.BUF)
                if not buf:
                    break
                clientSocket.send(buf)
            except Exception, e:
                print "Error: ", e
                break

        f.close()
        clientSocket.close()
        print "Copied!"
```

وبس كدا :D

```
if __name__=="__main__":
    alias=raw_input("Alias: ")
    sharedFiles=os.listdir(os.getcwd())
    peer=Peer(alias, ('localhost', 8080), sharedFiles)
```

طبعا تقدر تظبطها بحيث إنك تباصى ال addr الخاص بال server من ال command line او حتى من prompt :)

Regards,
Ahmed Youssef
Programming-Fr34ks.NET Administrator