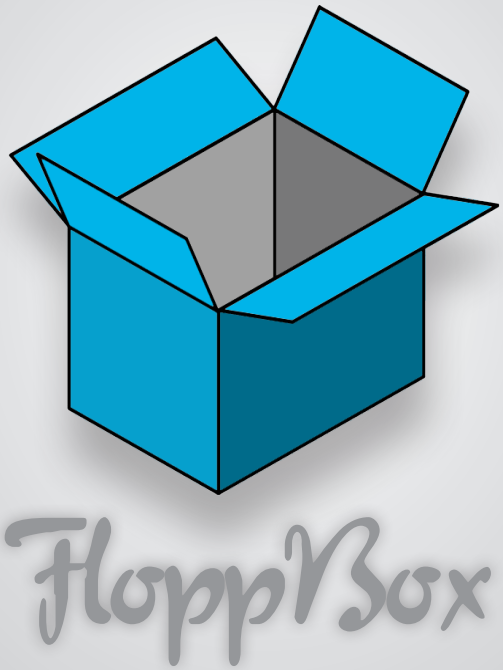


Rechnernetze und Netzwerkprogrammierung
Sommersemester 2012

# Dokumentation zur Projektabgabe “FloppBox“

**Marcel Bechtold, Soeren Kroell, Marc Maaß, Tina Schedlbauer**

## 1 | Einleitung

Im Rahmen des Moduls '**Rechnernetze und Netzwerkprogrammierung**' soll als Praktikumsleistung eine **Datei Synchronisationslösung** entwickelt werden, welche ein vorgegebenes Verzeichnis (<gruppenname>) im Home-Directory regelmäßig mit dem gleichen Verzeichnis auf einem anderen Client im lokalen Netzwerk abgleicht.

Auf dieser Grundlage basiert die hier vorliegende Applikation, welche folgende **Features** beinhaltet:

- Die Applikation kann mit folgendem Aufruf gestartet werden:
  python floppbox.py <gruppenname>
- Dateien egal welches Typs können synchronisiert werden
- Ordner/ Unterordner können **synchronisiert** werden
- Dateien/ Ordner können **gelöscht**, **editiert** oder **verschoben** werden
- Alle Aktivitäten werden in der Datei '<gruppenname>.log' **geloggt**

Im folgenden werden Ablauf und Funktionen der Applikation im Einzeln erläutert.

## 2 | Allgemeiner Ablauf beim Starten der FloppBox

Auf einem Client wird die Applikation 'FloppBox' gestartet. Im lokalen Netz existieren weitere Clients auf denen die Applikation bereits läuft. Umgangssprachlich findet folgende Kommunikation statt:

<An Alle> Hallo, ich bin da?

<An Alle> Hallo, ich bin auch da!

<An Alle> Hallo, ich bin auch da!

Na gut, dann nehm ich den Grauen und baue eine direkte Verbindung via TCP zu ihm auf.

Oh, eine Verbindung zu mir. Hier hast du meine Ordnerstruktur!

Die gleich ich doch gleich mal mit meiner ab .... Ohhh, schickst du mir bitte mal die Dateien aus dem Ordner z, die fehlen mir noch!

Hier die gewünschten Files!

Super, wird direkt gespeichert! Machs gut und danke für den Fisch! Verbindungsende...

## 2.1 | Erstellen von neuem Inhalt

Auf allen Clients im LAN läuft die Applikation 'FloppBox'. Wird nun ein neuer Inhalt auf Client 1 erstellt (Datei oder Ordner), sendet dieser via Broadcast eine Benachrichtigung darüber an alle anderen Clients im Netz. Diese registrieren die Benachrichtigung und 'beantragen' über TCP die Ordnerstrukturt von Client 1. Da dieser jedoch nur die Verbindung zu einem anderen Client aufnehmen kann, wird die Anfrage des Ersten angenommen und eine Liste der kompletten Ordnerstruktur übertragen. Der Empfänger gleicht nun diese mit der eigenen ab und lässt sich gegebenenfalls die fehlenden Inhalte schicken. Während der kompletten Kommunikation bleibt die TCP Verbindung offen.

Ist der Abgleich abgeschlossen wird die TCP Verbindung geschlossen und die Anfrage des nächsten Clients wird von Client 1 bearbeitet.

## 2.2 | Löschen von Inhalt

Auf allen Clients im LAN läuft die Applikation 'FloppBox'. Löscht nun Client 1 Inhalt informiert er via Broadcast alle anderen Clients im Netz darüber. Diese bauen jeweils eine TCP Verbindung zu Client 1 auf, lassen sich die neue Ordnerstruktur schicken und schließen nach Empfang die TCP Verbindung wieder. Im Anschluss führt jeder Client ein Abgleich mit seiner eigenen Ordnerstruktur durch und passt diese der neuen Struktur an, sofern neuere Inhalte als vorhanden verfügbar sind.

## 2.3 | Verschieben von Inhalt

Beim Verschieben von Inhalt wird als erster Schritt der Inhalt gelöscht (siehe dazu 2.2) und im Anschluss ein neuer Inhalt erstellt (siehe dazu 2.1).

## 2.4 | Editieren von Inhalt

Beim Editieren von Inhalt findet der gleiche Ablauf wie beim Erstellen von neuem Inhalt (siehe dazu 2.1) statt.

## 3 | Source-Listing

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

from decimal import Decimal
from socket import *
from operator import itemgetter
import pickle
import select
import sys, time, os, os.path
import logging
import random

filelist = []
dirlist = []
aktdirlist = []
aktfilelist = []
gruppenname =„“
neu = True
logger = None

BPORT = 1111 # ERSTLLE BROADCAST PORT
SPORT = 1112 # ERSTELLE TCP PORT
REFRESHTIME = 15 - random.uniform(0.0,6.0) # SELECT REFRESH

def get_dir_list():
  ,“Gibt ein set des aktullen Ordnerverzeichnisses zurueck“‘
  dl = []
  for root, dir, files in os.walk(gruppenname):
    dl.append(root)
  return set(dl)

def get_file_list():
  ,“Gibt ein set des aktullen Dateiverzeichnisses zurueck“‘
  filist = []
  fidic = {}
  for root, dir, files in os.walk(gruppenname):
    for fi in files:
      fileurl = os.path.join(root,fi)
      mtime = os.path.getmtime(fileurl)
      fidic[fileurl] = int(Decimal(mtime))
    #sortiert Filedic
    filist = sorted(fidic.items(), key=itemgetter(1), reverse = False)
  return set(filist)

def refresh_lists():
  ,“aktualisiert Ordner- und Dateiliste“‘
  global aktdirlist,aktfilelist
```

```
    aktdirlist = get_dir_list()
    aktfilelist = get_file_list()
    sorted(aktdirlist,reverse=False)

def makedirs(dirpathes):
    ,"erstellt Ordner"'
    for path in dirpathes:
        if not os.path.exists(path):
            logger.info(„MAKE DIR:„ + path +„ TIME:„ + time.ctime(time.time()))
            os.makedirs(path)

def sync(addr):
    ,"Diese funktion synconiesert zwei Rechner.
    Dazu wird eine TCP Verbindung zu dem Rechner aufgebaut der die neuen Ordner oder Dateien besitzt.
    Als erstes werden die Datei- und Ordnerliste vom angesprochen Rechner uebermittel.
    Die werden mit den eigenen Listen verglichen und fehlende Datein werden einzeln abgefragt und
empfangen."'
    global aktdirlist, aktfilelist, dirlist, filelist, gruppenname
    #Verbindung zu dem Rechner, der eine neue Informationen hat, wird aufgebaut
    s = socket(AF_INET, SOCK_STREAM)
    s.setsockopt(SOL_SOCKET, SO_REUSEADDR, True)
    s.connect((addr[0], SPORT))
    s.settimeout(3.0)
    logger.info(„TCP connect to:„ + addr[0] +„ TIME:„ + time.ctime(time.time()))

    info =„"
    recv = 1
    #die Datei- und Dirliste werden in dieser Schleife zusammengesetzt
    try:
        while recv:
            recv = s.recv(1024)
            info += recv
            #wenn die letzten drei Zeichen der Listen sind $$$, ist diese zu Ende
            if recv[-3:] ==„$$$":
                recv = 0
    except:
        s.close()
        logger.WARNING(„INCOMPLETE TRANSMISSION TIME:„ + time.ctime(time.time()))
        runSynchronisation()

    picklelist = info.split(„**")
    logger.info(„TCP recv filelist FROM:„ + addr[0] +„ TIME:„ + time.ctime(time.time()))

    #Listen des Fremdrechners werden geladen und gespeichert
    dirlist = pickle.loads(picklelist[0])
    filelist = pickle.loads(picklelist[1])
    #eigene Listen werden aktuallisiert
    refresh_lists()
```

```
    #fehlende Datein werden gefiltert
    miss_dirs = set(dirlist) - set(aktdirlist)
    miss_files = set(filelist) - set(aktfilelist)

    #fehlende Ordner werden zuerst erstellt, um Fehler beim Speichern der Datei zu verhindern
    if miss_dirs:
        makedirs(miss_dirs)
    #fehlende Dateien werden gespeichert
    if miss_files:
        #Schleife geht alle fehlenden Dateien durch
        for file_pathe, mtime in miss_files:
            #falls eine Datei nicht existiert oder aelter ist, wird sie ersetzt
            if not os.path.exists(file_pathe) or mtime > os.path.getmtime(file_pathe):
                #die fehlende Datei wird angefragt
                s.send(file_pathe)
                rec_data = s.recv(1024)
                #abfangen falls die Datei in der Zwischenzeit geloescht wurde
                if int(rec_data[:10]) !=0:
                    #an den ersten zehn stellen steht die groesse der Datei
                    filesize = int(rec_data[:10])
                    size = int(rec_data[:10])
                    #Temporaere Datei fuer Zwischenspeicherung wird geoeffnet
                    writeFile = open(gruppenname+"/temp.temp","w")
                    #Laenge des ersten recv wird von Datei groesse abgezogen
                    filesize -= len(rec_data)-10
                    #ersten Daten werden in die Datei gespeichert
                    writeFile.write(rec_data[10:])
                    logger.info("TCP recv FILE:" + file_pathe + " FROM:" + addr[0] +" TIME:" + time.ctime(time.
time()))

                    #Datei wird solange geschrieben, bis sie komplett empfangen ist oder die andere Seite ab-
bricht
                    try:
                        while filesize:
                            rec_data = s.recv(1024)
                            writeFile.write(rec_data)
                            filesize -= len(rec_data)
                        writeFile.close()
                        #Wenn die Datei komplet uebertragen wurde, wird die Temporaere Datei umbenannt
                        if os.path.getsize(gruppenname+"/temp.temp") == size:
                            os.rename(gruppenname+"/temp.temp",file_pathe)
                            #Zeit der Datei wird gleich der mit gesendeten Zeit gesetzt
                            os.utime(file_pathe,(os.path.getatime(file_pathe),mtime))
                            logger.info("TRANSMISSION COMPLED:" + file_pathe +" TIME:" + time.ctime(time.time()))
                        #Wenn die Datei fehlerhaft ist, wird diese geloescht
                        else:
                            os.remove(gruppenname+"/temp.temp")
                            logger.WARNING("INCOMPLETE TRANSMISSION:" + file_pathe +" TIME:" + time.ctime(time.
time()))
```

```
            except:
                s.close()
                logger.WARNING(„INCOMPLETE TRANSMISSION: „ + file_pathe + „ TIME: „ + time.ctime(time.
time()))
                runSynchronisation()
    s.close()
    logger.info(„Connection closed TO: „ + addr[0] + „ TIME: „ + time.ctime(time.time()))

def sendContent(ts):
    ,"Funktion die anefragte Dateien verschickt"'
    global aktfilelist, aktdirlist, dirlist, filelist
    #angefragte Verbindung wird akzeptiert
    s, addr = ts.accept()
    logger.info(„TCP connection accept FROM: „ + addr[0] + „ TIME: „ + time.ctime(time.time()))
    #Listen werden aktualisiert
    refresh_lists()
    #Listen werden als String verpackt
    liststring = pickle.dumps(aktdirlist)+"**"+pickle.dumps(aktfilelist)
    #Liste wird in 1024 Bloecken verschickt
    von = 0
    bis = 1024
    while von < len(liststring):
        s.send(liststring[von:bis])
        von += 1024
        bis += 1024
    #Liste schliesst mit $$$ ab
    s.send(„$$$")
    logger.info(„Send filelist TO: „ + addr[0] + „ TIME: „ + time.ctime(time.time()))

    file_pathes = 1
    #Schleife, die fuer alle angefragten Dateien den Inhalt schickt
    while file_pathes:
        file_pathes = s.recv(1024)
        #wenn die Datei exsitiert, wird sie gesendet
        if os.path.exists(file_pathes):
            fil = file(file_pathes)
            filesize = os.path.getsize(file_pathes)
            s.send(„%10d"%filesize)
            data = 1
            while data:
                data = fil.read(1024)
                s.send(data)
        #ansonsten wird die groesse 0 geschickt
        else:
            filesize = 0
            s.send(„%10d"%filesize)
        logger.info(„Send FILE: „ + file_pathes +" TO: „ + addr[0] + „ TIME: „ + time.ctime(time.time()))
    dirlist = aktdirlist
    filelist = aktfilelist
```

```
def del_sync(addr):
  ,"Diese funktion synchonisiert zwei Rechner.
  Dazu wird eine TCP Verbindung zu dem Rechner aufgebaut, der einen Ordner oder Dateien geloescht
hat.
  Als erstes werden die Datei- und Ordnerliste vom angesprochen Rechner uebermittelt.
  Diese werden mit den eigenen Listen verglichen und die ueberschuessigen Dateien oder Dateien ge-
loescht'"
  global aktdirlist, aktfilelist, filelist, dirlist
  #Verbindung zu dem Rechner, der eine neue Information hat, wird aufgebaut
  s = socket(AF_INET, SOCK_STREAM)
  s.setsockopt(SOL_SOCKET, SO_REUSEADDR, True)
  s.connect((addr[0], SPORT))
  s.settimeout(3.0)
  logger.info(„TCP connect to:„ + addr[0] +„ TIME:„ + time.ctime(time.time()))

  info =„"
  recv = 1
  #die Datei- und Ordnerliste werden in dieser Schleife zusammengesetzt
  try:
    while recv:
      recv = s.recv(1024)
      info += recv
      #wenn die letzten drei Zeichen der Listen ,$$$' sind, ist diese zu Ende
      if recv[-3:] ==„$$$":
        recv = False
  except:
      s.close()
      logger.WARNING(„INCOMPLETE TRANSMISSION TIME:„ + time.ctime(time.time()))
      runSynchronisation()

  picklelist = info.split(„**")
  s.close()
  logger.info(„TCP recv filelist FROM:„ + addr[0] +„ TIME:„ + time.ctime(time.time()))
  logger.info(„Connection closed TO:„ + addr[0] +„ TIME:„ + time.ctime(time.time()))

  #Listen des Fremdrechners werden geladen und gespeichert
  dlist = pickle.loads(picklelist[0])
  flist = pickle.loads(picklelist[1])
  #eigene Listen werden aktualisiert
  refresh_lists()

  #ueberfluessige Ordner und Dateien werden ermittelt
  d_dirs = set(aktdirlist) - set(dlist)
  del_files = set(aktfilelist) - set(flist)
  #Liste wird sortiert, damit die Ornder in der richtigen Reihenfolge geloescht werden
  del_dirs = sorted(d_dirs,reverse=True)
```

```
    #Dateien werden zuerst geloescht, um Fehler beim Löschen der Ordner zu vermeiden
    if del_files:
        for del_file, mtime in del_files:
            index = 0
            if os.path.exists(del_file) and mtime >= os.path.getmtime(del_file):
                os.remove(del_file)
                logger.info(„Remove FILE: „ + del_file + „ TIME: „ + time.ctime(time.time()))
                #Datien werden aus der Liste geloescht
                filelist.remove((del_file,mtime))
    #Ordner werden geloescht
    if del_dirs:
        for del_dir in del_dirs:
            logger.info(„Remove DIR: „ + del_dir + „ TIME: „ + time.ctime(time.time()))
            os.rmdir(del_dir)
            #Ornder werden aus der Liste geloescht
            dirlist.remove(del_dir)

def abgleich(bs, data, addr):
    ,"Funktion die die empfangen Broadcasts verwaltet"'
    global neu
    #Falls ein neuer Rechner in das System kommt
    if data[0] ==„$":
        logger.info(„Recv Broadcast From new joined: „+addr[0] +" TIME: „ + time.ctime(time.time()))
        #Broadcast fuer den neuen Rechner
        bs.sendto(„#",(„<broadcast>", BPORT))
        logger.info(„Broadcast hello new one. TIME: „ + time.ctime(time.time()))

    ,"Wenn ein neuer Rechner den Broadcast empfaengt reagiert er
    auf den ersten der Antwortet alle weitern werden dann ignoriert.
    Alle bisher laufenden Rechner ignoriern die Antwort auch."'
    if data[0] ==„#":
        if neu:
            sync(addr)
            neu = False

    #Broadcast, falls eine Datei neu ist
    if data[0] ==„%":
        sync(addr)

    #Broadcast, falls eine Datei geloescht wurde
    if data[0] ==„!":
        del_sync(addr)

def checkRoot(bs):
    ,"Funktion ueberwacht die Ordnerstrucktur des Rechners."'
    global filelist, dirlist, aktdirlist, aktfilelist

    #Listen werden aktuallisiert
    refresh_lists()
    sorted(filelist,reverse=False)
```

```
    sorted(dirlist,reverse=False)

    #Ueberprueft, ob eine Datei geloescht wurde
    if set(dirlist) - set(aktdirlist) or set(filelist) - set(aktfilelist):
        logger.info(„Found deleted File. TIME: „ + time.ctime(time.time()))
        #Broadcast an alle
        bs.sendto(„!“,(„<broadcast>“, BPORT))
        logger.info(„Broadcast has deleted File. TIME: „ + time.ctime(time.time()))

    #Uerbprueft, ob eine Datei geloescht wurde
    if set(aktdirlist)-set(dirlist) or set(aktfilelist)-set(filelist):
        logger.info(„Found new File. TIME: „ + time.ctime(time.time()))
        #Broadcast an alle
        bs.sendto(„%“,(„<broadcast>“, BPORT))
        logger.info(„Broadcast has new File. TIME: „ + time.ctime(time.time()))


def runSynchronisation():
    ,“Chef von allem, kann alles!“‘
    global dirlist, filelist
    bs = socket(AF_INET, SOCK_DGRAM) #erstelle Broadcast socket
    bs.setsockopt(SOL_SOCKET, SO_BROADCAST, True)
    bs.setsockopt(SOL_SOCKET, SO_REUSEADDR, True)
    bs.bind((„<broadcast>“, BPORT))

    ts = socket(AF_INET, SOCK_STREAM)  #erstelle TCP socket
    ts.setsockopt(SOL_SOCKET, SO_REUSEADDR, True)
    ts.bind((„“,SPORT))
    ts.listen(0)

    #Ordner- und Dateistruktur wird geladen
    dirlist = get_dir_list()
    filelist = get_file_list()
    #Broadcast an alle: ich bin neu (FloppBox wurde soeben gestartet)
    bs.sendto(„$“, („<broadcast>“, BPORT))
    logger.info(„Broadcast I am new. TIME: „ + time.ctime(time.time()))
    #Verwaltung der empfangenen TCP-Anfragen und Broadcast Mitteilungen
    while 1:
        a, b, c = select.select([bs, ts], [], [], REFRESHTIME)
        #TCP Anfrage ist immer eine Datei- und Ordneranfrage
        if ts in a:
            sendContent(ts)
        #Broadcast wird empfangen
        if bs in a:
            (data, addr) = bs.recvfrom(1000)
            myip = gethostbyaddr(gethostname())[2]
            #Faengt ab, ob der Broadcast nicht von einem selbst stammt
            if addr[0] != myip[0]:
                #Methode um die unterschiedlichen Broadcasts zu verwalten
                abgleich(bs, data, addr)
```

```
        #Wenn nichts los ist (kein Empfangen oder Senden), wird die eigene Ornderstrucktur ueberwacht
        if not bs in a:
            checkRoot(bs)

if __name__ ==,__main__‘:
    if len(sys.argv) == 2:
        gruppenname = sys.argv[1]
    else:
        print „FEHLER - FloppBox ausführen mit: python“, sys.argv[0], „<Ordnername>“
        sys.exit(1)

    if not os.path.exists(gruppenname):
        print „Passender Ordner“, gruppenname, „nicht vorhanden“
        sys.exit(1)
    #Logger wird erstellt
    logpath = gruppenname + „.log“
    logging.basicConfig(filename = logpath, level = logging.INFO)
    logger = logging.getLogger(„FloppBox:„ + gruppenname)
    logger.info(„Run FloppBox. TIME:„ + time.ctime(time.time()))
    try:
        runSynchronisation()
    finally:
        logger.info(„FloppBox SHUTDOWN:„ + time.ctime(time.time()))
        logging.shutdown()
```

# 4. | Bekannte Fehler

Bei Entwicklungsschluss bekannte Fehler werden anschließend aufgeführt.

## 4.1 | Leere Datein

Datein ohne Inhalt werden nicht Syncronisiert.

## 4.2 | Anlegen von neuen Dokumenten mit 'Rechts-Klick'

Wird innerhalb des synchronisierten Gruppenordners per 'Rechts-Klick' ein neues Dokument erstellt, so kann es passieren, dass diese fehlerhaft sind und nicht mehr richtig verwaltet werden können.

Werden neuen Dateien ausserhalb erstellt und via Drag-n-Drop bzw. kopieren in den Gruppenordner eingefügt, so treten diese Probleme nicht auf.

## 5 | Erklärung

Hiermit erklären wir, dass das Projektergebnis vollständig in Eigenleistung erstellt bzw. verwendete Teile aus fremden Arbeiten deutlich als solche gekennzeichnet wurden.

_____

Marcel Bechtold Wiesbaden, 29.06.2012

_____

Soeren Kroell Wiesbaden, 29.06.2012

_____

Marc Maaß Wiesbaden, 29.06.2012

_____

Tina Schedlbauer Wiesbaden, 29.06.2012