# Programming Assignment 2
# Maintaining file consistency in your Gnutella-style P2P system
# CS550- Advanced Operating System

Harsh Singh (A20398109)

## Design:

### Introduction:

Gnutella is a protocol for distributed search. It is a decentralized Peer-to-Peer model. Though it's based on client-server model, in the absence of central server, each peer themselves acts as a server. When any peer issues a query, a peer, acting as a server, will receive the request, process it and propagate through the network (by sending request to its neighbor). This system is highly fault tolerant. A subset of peer leaving the network wouldn't fail the system.

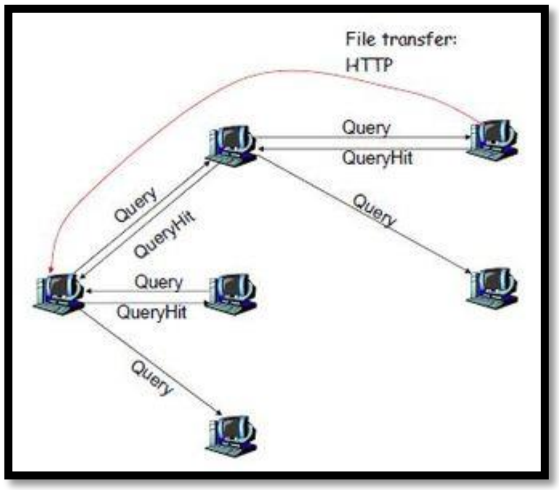


*Figure 1 GNUTELLA P2P system*

To maintain file consistency, either cached file holder could prompt original file owner or original file owner could flood the network about any change. These methods are often use to maintain file consistency in distributed systems.

### Brief Details:

System Requirement:

1. Linux based OS enabled Computer
2. Python 3.0+ compiler.
3. At least 10 open local ports.

Running the code:

1. Navigate to directory in 10 different terminals.
2. Use *make peer_x* where x is (1, 2 ... 10)
3. Use params.py to manipulate ports, if default ports are not free.
4. Also use params.py, to configure PUSH/pull.

**Design:**
Underlying architecture of the code follows core guidelines of Gnutella protocol (v0.4). Each peer acts as server and client. When user enters a query, the peer acting as a client sends request to another peer acting as server. The message is processed and forwarded accordingly.

Since Gnutella doesn't specify about initial neighbors, one neighbor is specified by default in the code. Initially, each peer connected to next peer in circle configuration. (fig 2)

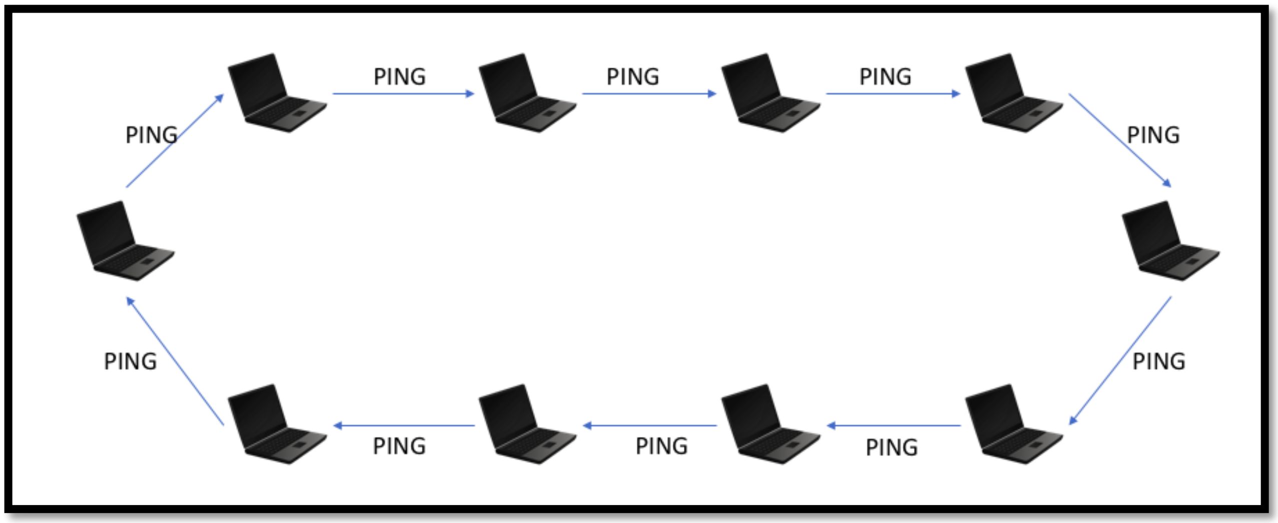
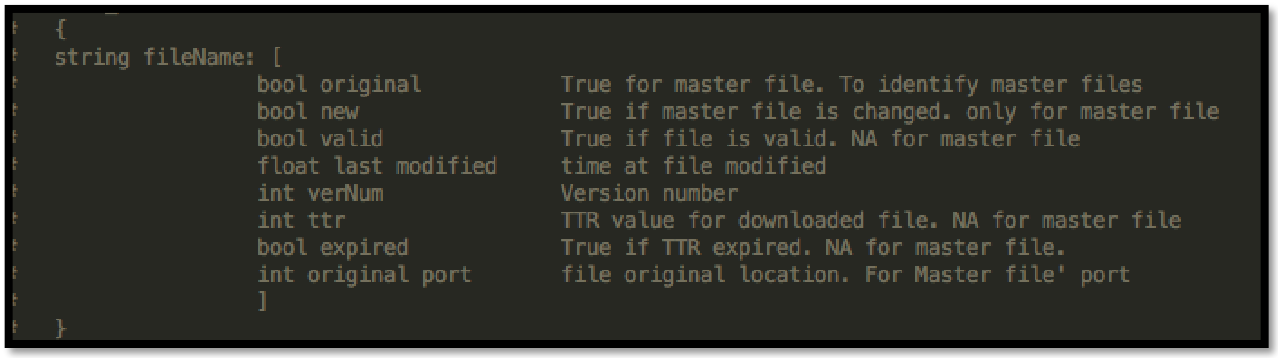


*Figure 2 Initial PINGs*

>GNUTELLA implementation:

- A file list is maintained containing the names of all files in the shared directory.
- Query forward, PING etc. are designed on top of a Network abstractions to minimize communication redundancy and failure.
- Query and query forward message contains original port and previous port information, to reduce communication redundancy. Along with them, Hops value are also propagated to attain similar goal.
- QUERYHIT message contains port of file owner and other information for direct connection for downloading the required file. (fig 1)
- Whenever a query message is generated and passed, query flag is set for certain time to acknowledge any incoming QUERYHIT.

To check file consistency, file list was updated to a structure (python dictionary) as shown in figure 3. Also, additional parameter PUSH was added to configure between PUSH and PULL.



```
{
string fileName: [
                bool original          True for master file. To identify master files
                bool new               True if master file is changed. only for master file
                bool valid             True if file is valid. NA for master file
                float last modified    time at file modified
                int verNum             Version number
                int ttr                TTR value for downloaded file. NA for master file
                bool expired           True if TTR expired. NA for master file.
                int original port      file original location. For Master file' port
                ]
}
```

*Figure 3 File list struct*

>PUSH implementation:

A separate thread is added to monitor any changes in master (owned by a peer) files. File last modified parameter is used to ascertain version number. When a file changed is detected, an invalidation message is broadcasted over the network. It follows QUERY mechanism of Gnutella protocol. Message format is *"INVALIDATE:<Origin_port>:<current:port>:<Hop>:<vesrsionNumber>:<pushID>:<filename>". "*pushID" is used to identify invalidate message. It's generated by appending port number to current time. Making it always unique.

Upon receiving the Invalidation message, a peer checks for the file. if available and version number doesn't match then, application will remove. Also invalidation message is propagated.

>PULL implementation:

Similar to PUSH, a file change monitor thread is added but the thread just updates the version number and does not use network abstractions. A separate thread is used to update the TTR value of a downloaded file. This thread updates the "expired" member of structure upon TTR reaching to 0. Another thread is used to detect if the "expired" member of structure is marked True. PULLPOLL message is sent to owner of the file to request its current version number. If current version number matches with the saved version number then, the TTR value is updated. Otherwise, file is marked invalid by setting valid to False.

>USER commands:

Client side of peer works whenever user request a file query, REFRESH a file etc.

- SEARCH
  It enables user to query a file across the network. QUERY message is created of format *"QUERY:<origin_port:,<current_port>: <hop>:< filename>"* and forwarded to all neighbors and their corresponding neighbors. After receiving none, one or multiple QUERYHITs, user is prompted to select the peer. Then download message is sent to the selected peer of format *"DOWNLOAD:<port>:fileName".*
- UPDATE

It enables user to force update file list if he/she changes anything in shared directory. The directory is read again, and the file list is updated accordingly.

- REFRESH
  (Only works when PUSH is set to False in params.py) It enables user to force check and if needed, download a cached copy of file. “valid” member of file list is checked. If valid is False, the new version of file is requested from the owner

**Tradeoffs:**

1. The system is designed with the intent of every peer gets the message. Sometimes, it introduces redundant communication. E.g. broadcasting invalidate message.

**Improvements:**

1. “Master” peers could be assigned, (dynamically) acting as central indexing server and slave peers could connect on them. This will lead to less network usage.
2. Currently, renaming of file is not supported. A symbolic link could be used to tackle the problem.
3. Partial matches of the search could be used.
4. Hybrid of PUSH and PULL could be developed and implemented
5. UI could be improved.