Implementation of Bittorrent-like algorithem Using UDP with Reliability and Congestion Control

CS:6250 Computer Networks - Project Final Report

Team 11: Mansour Alharthi, Jiahao Cui, Peicheng Hua, Hao Fu, Nan Zhang

For: Dr. Mostafa Ammar

Georgia Institute of Technology

Introduction

BitTorrent, one of the most widespread file sharing P2P applications, has recently been updated to eliminate use of TCP by introducing an application-level congestion control protocol [1]. This new protocol aims to efficiently use the available link capacity while avoiding interference with other user traffic (e.g., Web, VoIP, and gaming) sharing the same access link. In this project, we aim to simulate this protocol in a virtual network environment. Because of the high complexity of Bittorrent, a full Bittorrent protocol implementation is not feasible; instead, we will implement a Bittorrent-like protocol to demonstrate its functionality.

Design

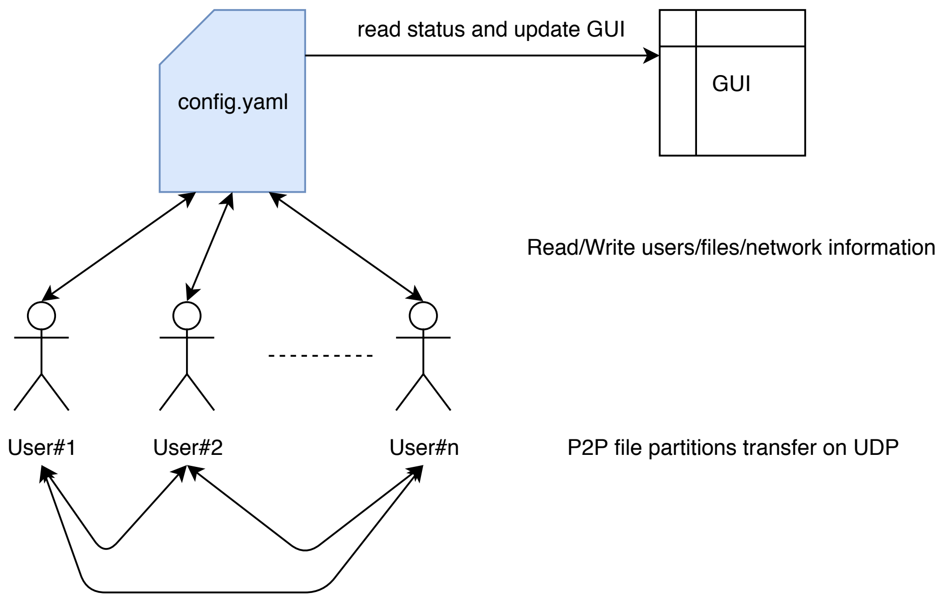

read status and update GUI
config.yaml
GUI
Read/Write users/files/network information
User#1
User#2
User#n
P2P file partitions transfer on UDP

Figure 1: The design of Bittorrent-like protocol

Our protocol design objectives are as follows:

1. **Fairness among users.** We ensure fairness by dividing available bandwidth among users by equal shares.
2. **Fairness to network traffic.** We ensure fairness to network traffic by detecting congestion control and sensing available network bandwidth.
3. **Link utilization.** Our design satisfy link utilization requirements by dividing available bandwidth only to active users.

4. **Transparent network status.** Our design includes an easy-to-read Graphical User Interface; allowing the user to monitor the network status.

The design of our protocol is shown in Figure 1. When a user wishes to download a file, it will first query a central `config.yaml` file that contains information about available users, existing files in the peers network, and the network status. Then, the user will initiate a file request from peers that own the requested file; keeping in mind the available bandwidth allocated to him in `config.yaml`. The new user will also update the `config.yaml` to add his own information, and update the network status information. A Graphical User Interface is updated periodically to reflect changes made in the `config.yaml`. An example of `config.yaml` is shown in Listing 1.

```
available: 138
file_info:
  filename1:
    file_size: 60
    hostsWithFile:
    - {ip: 127.0.0.1, port: 8080}
    - {ip: 127.0.0.2, port: 8081}
    - {ip: 127.0.0.3, port: 8082}
  filename2:
    file_size: 100
    hostsWithFile:
    - {ip: 127.0.0.1, port: 8080}
    - {ip: 127.0.0.2, port: 8081}
    - {ip: 127.0.0.3, port: 8082}
lock: false
network_info:
  activeUsers:
  - {ip: 127.0.0.1, port: 8080}
  - {ip: 127.0.0.2, port: 8081}
  - {ip: 127.0.0.3, port: 8082}
  available: 138
  bandwidth: 100
  threshold: 80
  total_time: []
total_time: [4.031458854675293, 4.0014543533325195,
    3.9738223552703857, 3.9723050594329834,
  3.9719390869140625, 3.972656726837158, 3.9447362422943115,
      3.9431591033935547, 3.9433462619781494,
  3.9440770149230957, 3.9153740406036377]
```

Listing 1: Example `config.yaml` file

Implementation

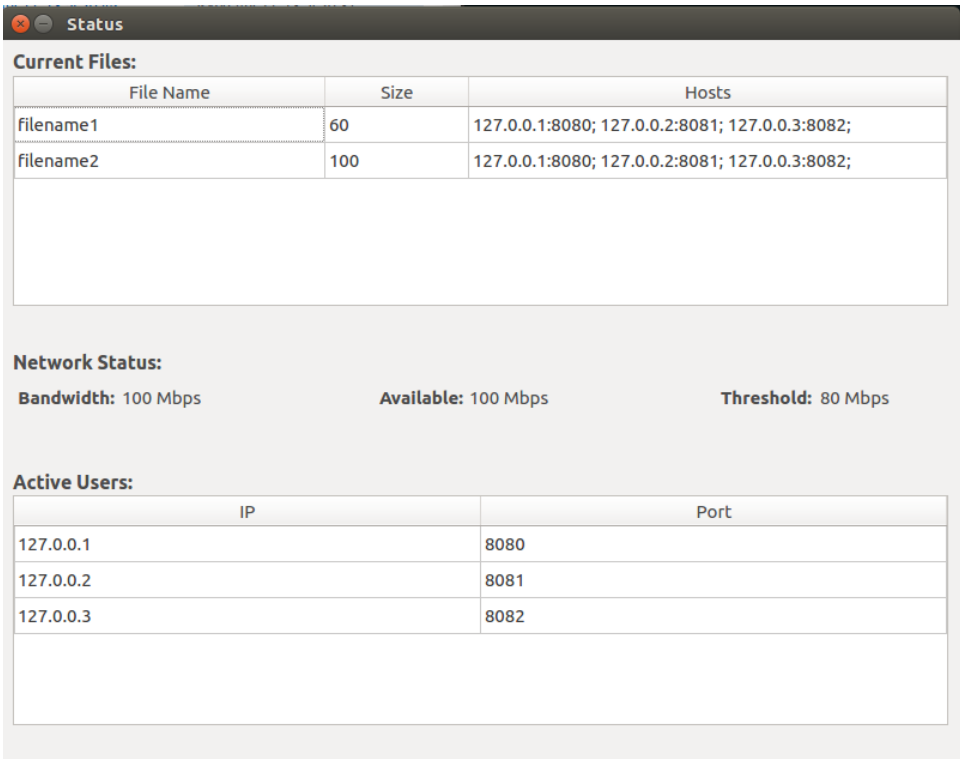
Part 1

Part 2

Part N

Graphical User Interface

The Graphical User Interface should reflect current peer network status based on information read from `config.yaml`. Given the frequency the configuration file changes with, the GUI content will expire every second, and a content update will occur. Figure 2 shows our GUI design. We implemented the GUI code using `Python`, utilizing the `PyQt4` library.



Current Files:

File Name	Size	Hosts
filename1	60	127.0.0.1:8080; 127.0.0.2:8081; 127.0.0.3:8082;
filename2	100	127.0.0.1:8080; 127.0.0.2:8081; 127.0.0.3:8082;

Network Status:

Bandwidth: 100 Mbps **Available:** 100 Mbps **Threshold:** 80 Mbps

Active Users:

IP	Port
127.0.0.1	8080
127.0.0.2	8081
127.0.0.3	8082

Figure 2: A GUI showing current peer network status

Evaluation

Concurrent Download

Show the speed of concurrent download in our implementation. Maybe simulate packet loss to test error detection?

Comparison with TCP

Compare our results with concurrent download using TCP. Simulate packet loss.

Discussions

Equal Sharing. Our design divides the available bandwidth among active users in equal shares. Although this ensures fairness among users, it also may lead to inefficient link utilization if the user bandwidth demand is less than equal share.

Other points to discuss?

Conclusion

We designed and implemented a peer-to-peer file transfer protocol that ensures fairness to users and network traffic. We tested our implementation against concurrent TCP connection downloads. Our results show that our implementation provide XXXX increase in download speed over concurrent TCP connections download.

References

[1] Arvid Norberg, *uTorrent transport protocol*. `http://www.bittorrent.org/beps/bep_0029.html`