**DartSync**

**Xiankai Yang, Luyang Li, Shanzae Nadeem Khan, Khizar Hussain**

**Team C-Port**

**Final Project**

Culminating Project for Networks Class offered by Professor Xia Zhou in Spring, 2016

Abstract

In this project, a completely different paradigm for synchronization of personal data across different devices is explored. Rather than relying on the centralized cloud, we enable local file exchange and synchronization across devices. Devices can directly talk to each other to obtain the latest files, and all files are stored locally. The resulting benefits are twofold. First, it naturally protects user's data privacy, since all data are stored locally and controlled by users, and no copies are made on centralized cloud. Second, it constraints traffic within a local network, which not only reduces the network traffic burden to remote cloud servers, but also leads to faster file synchronization by accessing local networks. This design paradigm is towards our vision of a "decentralized cloud"

**DartSync**

DartSync runs on a tracker (server) and multiple peers (clients). These can be run with arguments: ./DartSync -tracker or ./DartSync –peer

1. **The Tracker**

The tracker collects file information from all peers, maintains records of all files, and notifies peers upon any file updates. Note that the tracker does not store files, but rather only keeps file information of all peers. Specifically, the tracker has the following functionalities:

* *Maintaining file records*. The tracker collects file information from all peers and keeps the information in a file table. Each entry in the table consists of file information, the peer IP address, and the timestamp. The tracker periodically handshakes with peers to receive their file information. The tracker then compares the tracker-side and peer-side file information and see whether there are updates to broadcast to all peers. The tracker always knows which device has the newest file.
* *Monitoring online/alive peers*: The tracker should always know whether a peer is online or alive, by receiving a heartbeat (alive) message every ten minutes (or any time interval you define) from every peer. The tracker maintains a peer table for the list of alive peers, and updates the table based on the heartbeat messages it receives. If the tracker does not receive any heartbeat message from a peer for ten minutes, this peer will be deleted in the peer table

1. **Peers**

A peer node monitors a local file directory, communicates with the tracker, and updates files if necessary

* *Monitoring a local file directory*. Users define a local file directory that contains all the files users want to synchronize. This file directory is similar to the Dropbox root directory. The peer node monitors this folder and sends out handshake messages to the tracker of any updates
* *Communicating with the tracker*. A peer node communicates with the tracker by handshake messages. The handshake message from the tracker contains the timestamps of the latest files and the list of IP addresses that own these files. A peer parses the message to know whether it needs to download any files from other peers
* *Downloading and uploading files*. Peers upload and download the newest files from other peers using Peer-to-Peer (P2P) connections. Each peer has a thread that keeps listening to messages from other peers, and another thread that creates P2P connections to upload or download files. To avoid duplicated downloads, the peer maintains a peer-side peer table that tracks all its existing P2P download threads. When multiple other peers have the latest file, the peer can request different file pieces from these peers concurrently

**Requirements**

1. C implementation that can run in any Linux system
2. TCP connection and Data Transfer
3. Local File Monitoring
4. Synchronization of Multiple files by comparing file time stamps
5. File replacement when updating files
6. Fetching data from multiple peers
7. Synchronization of multiple file folders

Here is a list of advanced features that we have added to our project:

**Design**

1. Threads:

Figure below shows the major threads running on the tracker and peers and their interactions.

* + 1. Tracker Threads:
* **Main thread:** Listen on the handshake port and create a Handshake thread when a new peer joins
* **Handshake thread:** receive messages from a specific peer and respond if needed, by using peer-tracker handshake protocol:



* **Monitor alive thread**: monitor and accept alive message from online peers periodically, and remove dead peers if timeout occurs:



* + 1. Peer Threads:
* **Main thread**: after connecting to the tracker, receive messages from tracker, and then create P2PDownload Threads if needed.
* **P2P listening thread**: listen on the P2P port; when receiving a data request from another peer, create a P2PUpload Thread:



* **P2PDownload thread:** download data from the remote peer:



* **P2PUpload thread:** upload data to the remote peer:



* **File monitor thread:** monitor a local file directory; send out updated file table to the tracker if any file changes in the local file directory:



* **Alive thread:** send out heartbeat (alive) messages to the tracker to keep its online status:



1. Peer-Tracker Protocol:

Peer-Tracker Protocol (PTP) defines the structure of packets sent between peers and the tracker. A peer needs to inform the tracker when it logs in the system for the first time. It also needs to send keep-alive messages periodically, and send local file table when necessary. On the other hand, the tracker provides information for peers to set up, and broadcasts latest file table. The most important part here is how to define and send the file table, whose structure is left for you to design. In this section, we will cover the protocol in details.

* **Peer to Tracker:**

**Protocol Length:** protocol length.

**Protocol Name:** protocol name.

**Reserved**: reserved space for the further extensions.

**Request Type:** type of the packet: 0 –REGISTER 1 –KEEP ALIVE 2 –FILE UPDATE.

**Peer IP:** IP address of the peer.

**Listening Port**: p2p listening port number.

**Files:** file table of the peer.

* **Tracker To Peer:**

**Interval**: interval between two consecutive keep-alive messages.

**Piece Length**: length of a piece of the file

**Files**: file table of the server.

When a peer first logs in the system, it should send a REGISTER packet to the tracker to notify its existence. Files field in a REGISTER packet should be null. Upon receiving a REGISTER packet, the tracker sends back a packet informing the Interval and Piece Length for the peer to set up. A peer needs to send a KEEP ALIVE packet periodically according to the interval returned by the tracker. Moreover, a FILE UPDATE packet must be sent upon local file changes. Once there is a file change/update in the local directory, the peer constructs a packet containing the updated local file table and sends it to the tracker. Once the tracker receives a FILE UPDATE packet from a peer, it compares the received file table with the one it already has. If an update is necessary, the tracker will broadcast the updated file table to all peers currently alive.

1. **Data Structures:**

Following Data Structures are being utilized in our DartSync system:

* Tracker side Peer Table:



* Main File Table: ~Some explanation goes here for why we chose this~



* Peer File Table: ~Same sort of explanation will be necessary~



1. **P2P File Transfer:**

While peers communicates with the tracker to obtain information about the latest versions of files, the actual file transfer happens among peers themselves. During a P2P file transfer, a peer is both an uploader and a downloader:

1. **Uploading:**
2. **Downloading:**
3. **File Transfer:**

Requesting file data from multiple peers can boost download speed. In order to achieve this, we need to partition a file into pieces, and allow a peer to download different pieces from multiple peers and finally combine these pieces into a whole file. Specifically a piece represents a range of data. The downloader will send a request to the uploader with a piece number. Then, the uploader will choose the corresponding range of data from its memory and send that data back. The downloader will order the received pieces based on their piece numbers and store them in the buffer. When all pieces are received, the data in the buffer is exactly the same as the requested file. Recall that the piece length is retrieved from the server. The last piece might not be a full piece.

1. **Local File Monitor:**

The File Monitoring module monitors file system activities. We have used the provided open source lib inotify to monitor the local directory for changes that will then dictate the actions taken by DartSync. The implementation of the file monitor thread is given above under Peers’ File Monitor Thread.

**Difficulties Faced During the Project Work:**

References:

Last Name, F. M. (Year). Article Title. *Journal Title*, Pages From - To.

Last Name, F. M. (Year). *Book Title.* City Name: Publisher Name.

Appendix

1Any appendix we may have goes here …