Simplified File Transfer Protocol

Examining Networking Capabilities of Haskell, Java, and the Event-driven Python Library Twisted

Coy Humphrey and Dustin Pfeiffer

First paragraph.

Second paragraph.

Third paragraph.

```haskell
-- Dummy Haskell code
main = do
    putStrLn "Hello"
    return ()
```

```java
// Dummy Java code
for (int i = 3; i >= 0; --i){
    System.out.println (i);
}
```

Why choose these languages?

We chose Java because we already had an understanding of Java and it made for a good, prototypical imperative language. Similarly, we chose Haskell because we were learning it in class, and we feel it represents functional programming languages. After seeing how similar our Java and Haskell implementations were, we chose Twisted to explore event-driven programming.

Project Outline

Our goal was to create an FTP server in different languages to compare the networking capabilities of each language, as well as the ease of programming and understanding. We wanted our server to have basic FTP functionality, such as `get` and `put`. We decided to simplify the implementation, using just ports rather than an IP address. We decided to use Java, Haskell, and Twisted, a Python library.

Handling Clients

```java
for (;;) {
    Socket clientSocket = serverSocket.accept();
    (new Thread(new ClientHandler(clientSocket))).start();
}
```

In Java, we had a main loop that would constantly accept clients and start a new thread to handle each new client.

```
loop serv_sock = do
    (h,host,_) <- accept serv_sock
    forkIO $ handler h host ""
    loop serv_sock
```

The Haskell implementation follows the same structure as the one in Java. Notably, while Haskell does not have an infinite for loop as in Java, it can still loop endlessly by using tail recursion. Additionally, Haskell can start a new thread using only a function, rather than creating a Thread object as in Java.

Handling Files

```
while ((int d = in.read()) != -1){
    fileout.write(d);
}
```

Our Java implementation is naive method of reading and writing files. It reads and writes a single byte at a time without buffering, until it reaches the end of the file. This affects speed of the transfer, but is still suitable for our goals in this project.

```
withFile file ReadMode (\handle -> do
    contents <- B.hGetContents handle
    B.hPut sock contents)
```

Our Haskell implementation takes advantage of Haskell's laziness. Conceptually, we read in the contents of the entire file, then write everything to the socket. Haskell will handle any buffering that needs to be done.

```
def connectionMade(self):
    fs = FileSender()
    fs.beginFileTransfer(self.factory.fp, self.transport)
```

Twisted provides a class called FileSender for sending files.

Language	Lines	Center
Java	210	Hello
Python	130	Dustin
Haskell	90	Hi