SYNCX

Outline

- Problem Statement
- Approach
- Implementation
- Reinforcement Learning
- Empirical Analysis: Comparison with State of the Art Technique
- Timeline
- Future Research

Problem Statement

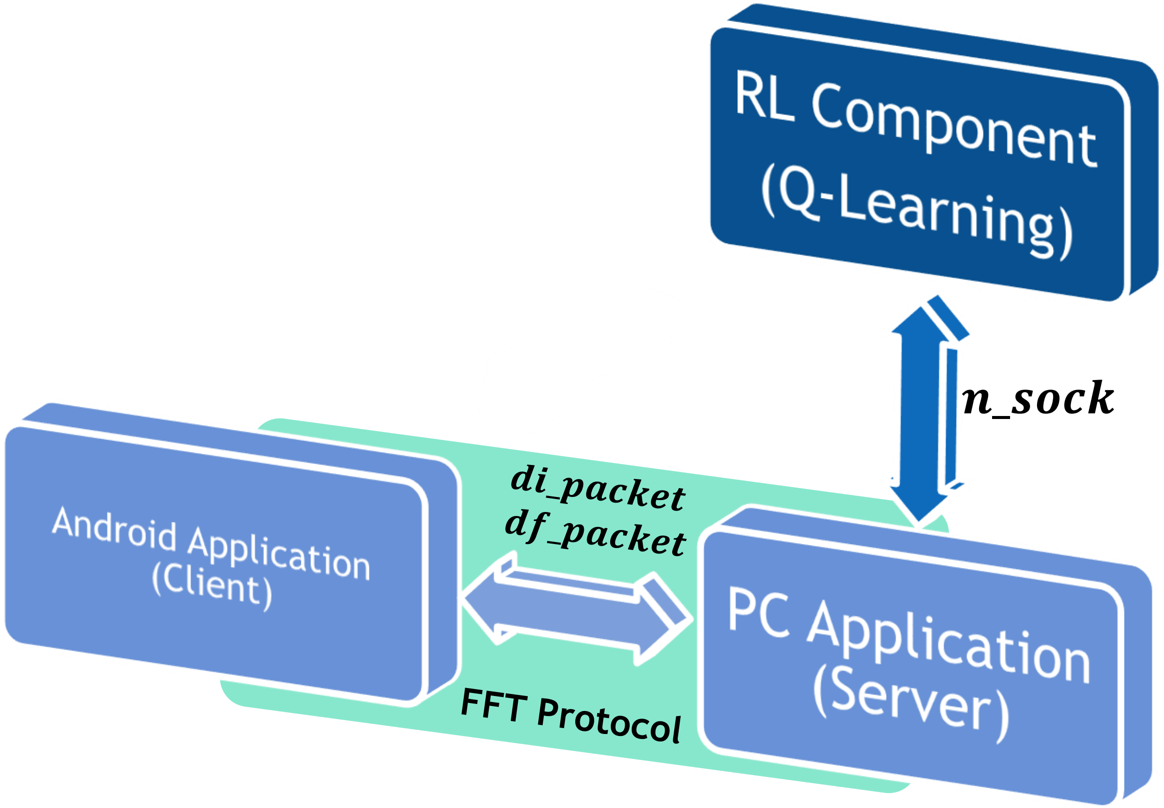
Optimizing the file transfer throughput between two end systems using Reinforcement Learning and a customized network protocol

Most of the existing file sharing software's like "***ShareIt***" and "***Xender Web***" don't provide the optimal speed for transfer using parallel sockets (depending on system resources) because of the dynamically changing resources and standardization of file transfer protocols.

Approach

- Since **FTP** doesn't support multiple sockets to transfer a single file, we designed our own underlying protocol (***FFT***) to transfer files between two end systems using **TCP** sockets. Our protocol supports multiple sockets for file transfer.
- Our approach was to implement RL as a managing element. We built a RL model to dynamically learn the optimal TCP sockets to transfer a payload. This allowed us to utilize maximum available system resources.
- An Android application and a corresponding Server End on PC to transfer files interchangeably employing the ***FFT*** protocol with optimal number of sockets.

Implementation



Client End

Native Android Application built using **AndroidStudio** – JAVA.

- The Application receives the optimal sockets for transfer after interaction with the server end through a permanent connection.
- It splits the payload into chunks equal to the number of sockets and sends them over a reliable link to the server application to which it is connected.
- The application is built as a native interface with minimum computational overhead.
- The server end records the transfer information that is used further to improve the RL model.

The FFT Protocol

- Same as **FTP** (File Transfer Protocol), there is a permanent **TCP** connection between the server and client, which is used to send the data-exchange information($\boldsymbol{di\_packet}$).
- The sending entity will first exchange the file information – file name, file size over a non-permanent connection.
- After exchanging the file information, the sending entity starts sending the file by truncating it into $\boldsymbol{n}$-different chunks and send all the chunks parallely using multiple sockets.
- The number $\boldsymbol{n}$ is determined by our RL algorithm running on the server (PC Application).

FFT Protocol

Client End: di_packet

Fields	Size(in bytes)	Description
is_request	1	• 1 – When a sender is asking receiver to send a file, the packet is called $\boldsymbol{di\_request}$ packet. • 0 – When receiver is granting permission to sender, the packet is called $\boldsymbol{di\_response}$ packet.
name_size	4	Length of the name of the file
name	*name_size*	Name of the file(in byte utf-8 encoding)
file_size	8	Size of file in bytes
id	4	Unique Identifier for a single file transaction
n_sock	4	Value currently not set

FFT Protocol

Server End : di_packet

Fields	Size(in bytes)	Description
is_request	1	• 1 – When a sender is asking receiver to send a file, the packet is called $\boldsymbol{di\_request}$ packet. • 0 – When receiver is granting permission to sender, the packet is called $\boldsymbol{di\_response}$ packet.
name_size	4	Length of the name of the file
name	*name_size*	Name of the file(in byte utf-8 encoding)
file_size	8	Size of file in bytes
id	4	Unique Identifier for a single file transaction
n_sock	4	Optimal number of sockets as determined by server

FFT Protocol

- When a sender(Server/Client) wants to send a file, it needs to send a $\boldsymbol{di\_packet}$ to the receiver with $\boldsymbol{is\_request}$ set as 1 and receives a $\boldsymbol{di\_packet}$ with $\boldsymbol{is\_request}$ set as 0.
- The $\boldsymbol{n\_sock}$ field value is decided by the server. In case the server wants to send a file, the value of $\boldsymbol{n\_sock}$ is already set for both $\boldsymbol{di\_request}$ and $\boldsymbol{di\_response}$ packet as determined by the RL model.
- However, on the other hand, if the client wants to send a file, then the value of $\boldsymbol{n\_sock}$ is randomly selected for the $\boldsymbol{di\_request\ packet}$. The random value is discarded by the server and $\boldsymbol{di\_response\ packet}$ contains the optimal value for $\boldsymbol{n\_sock}$.
- Server cannot initialize connection itself so if server wants to send a file then it will send $\boldsymbol{di\_request}$ packet the client and after sending $\boldsymbol{di\_response}$ packet client will also initialize $\boldsymbol{n\_sock}$ parallel connection to the server and on each connection it will send empty $\boldsymbol{df}$ header packet($\boldsymbol{df\_packet}$ with no payload) then server send $\boldsymbol{df\_packet}$ to client with payload.

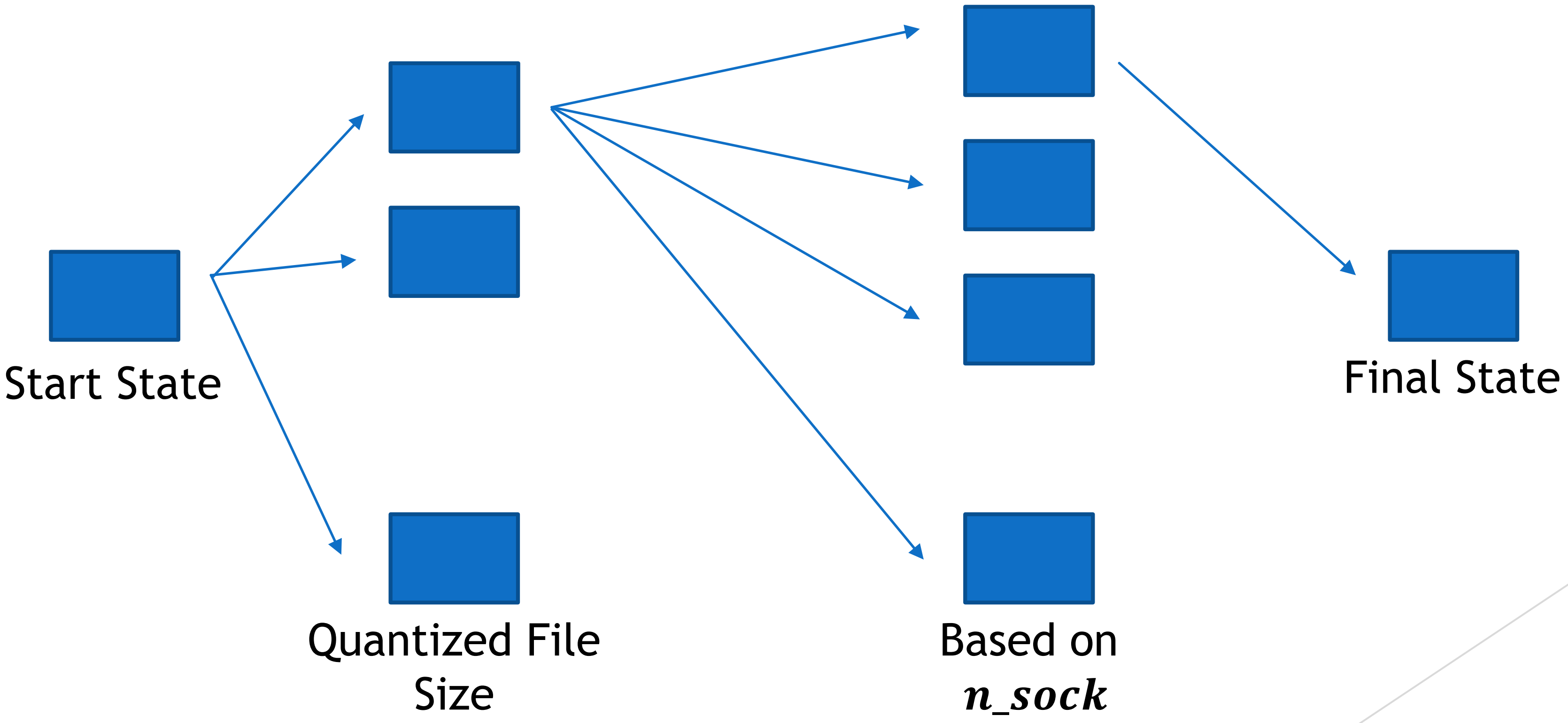
FFT Protocol : df_packet

Fields	Size(in bytes)	Description
data_id	4	Unique Identifier for a single file transaction
starting_point	8	Offset from the beginning of the file
data_size	8	Size of the payload
data	data_size	Payload

Reinforcement Learning

For the reinforcement learning model, we have used **Q-value** algorithm to determine the value of n_sock for a given file size.

The MDP of the problem is given below:



Reinforcement Learning

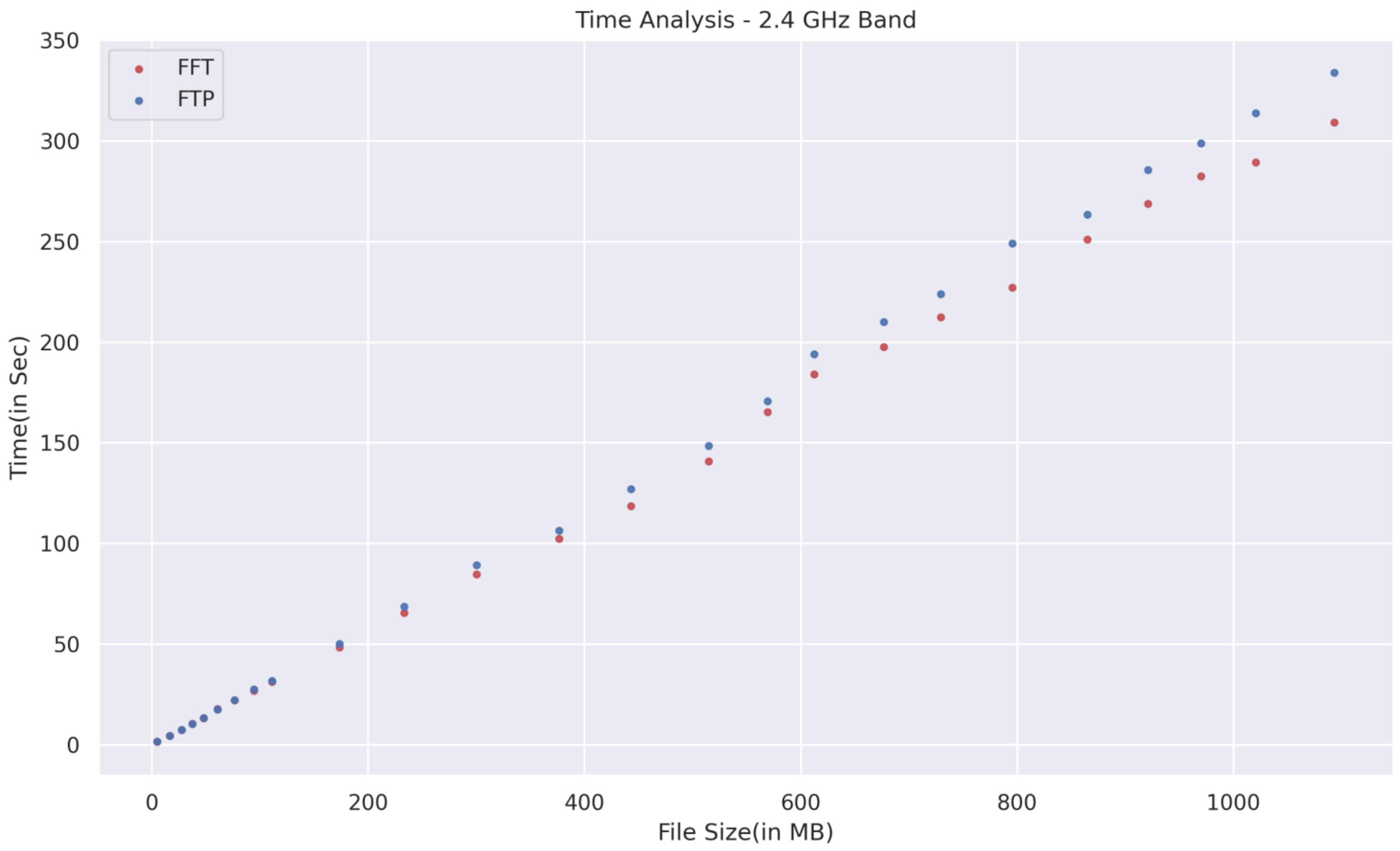
- For construction of discrete states from file sizes, we quantize the file size into **10** states, having linear interval of **20**MB, for size **0** to **200**MB. For file sizes **200**MB to **1**GB, **10** states are chosen such that they follow a geometric progression.

- Initially, we had also chosen an exploration rate of **0.99** and an exploration decay rate of **0.99**. After every iteration, the exploration rate was decreased.

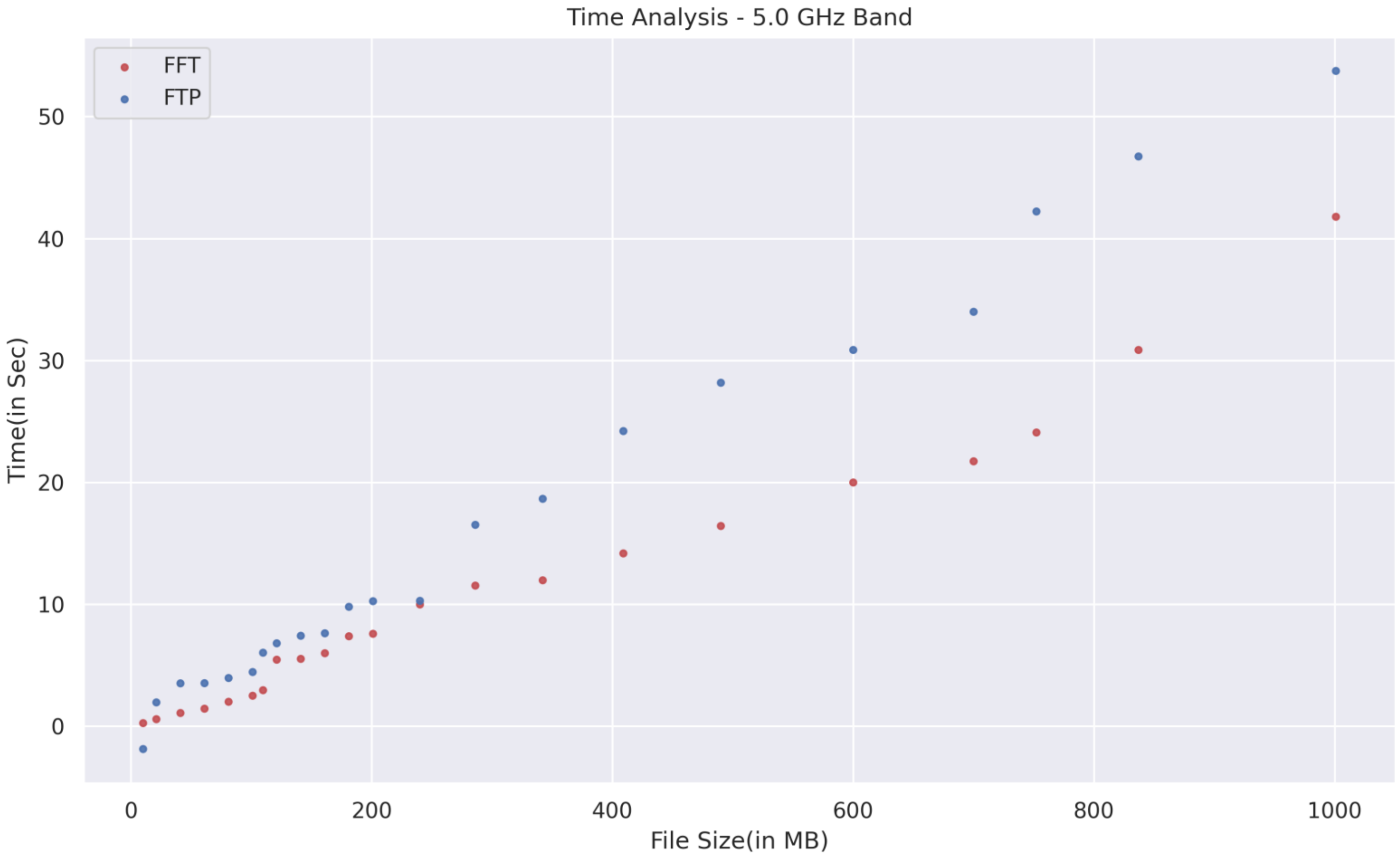
Reinforcement Learning

For abstraction, the Reinforcement Learning model had implemented 2 functions, ***predict*** and ***learn***. The '***predict***' function takes the input as file size, perform the corresponding *state transition* and return the $\boldsymbol{n\_sock}$ value. Later, the learn function would the reward for the action(dependent on the bandwidth that was obtained) and would make the Q-value updation.

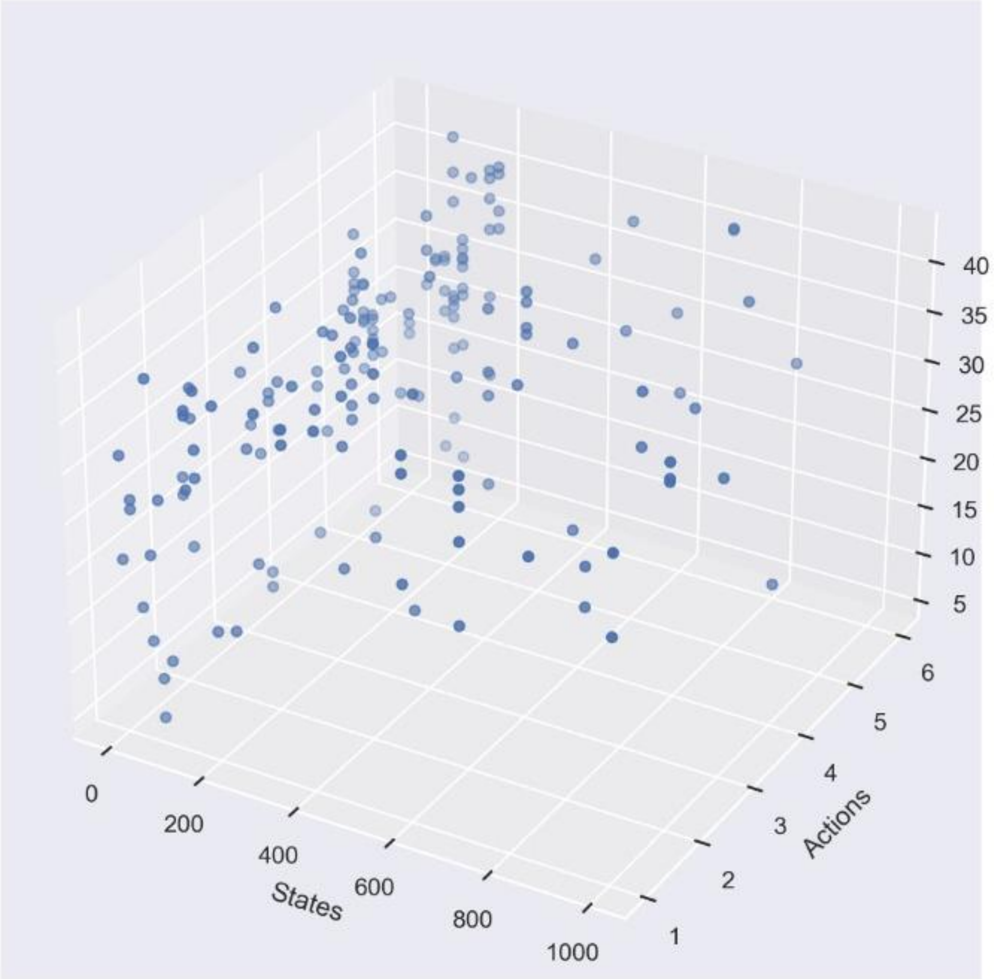
Empirical Analysis : Time



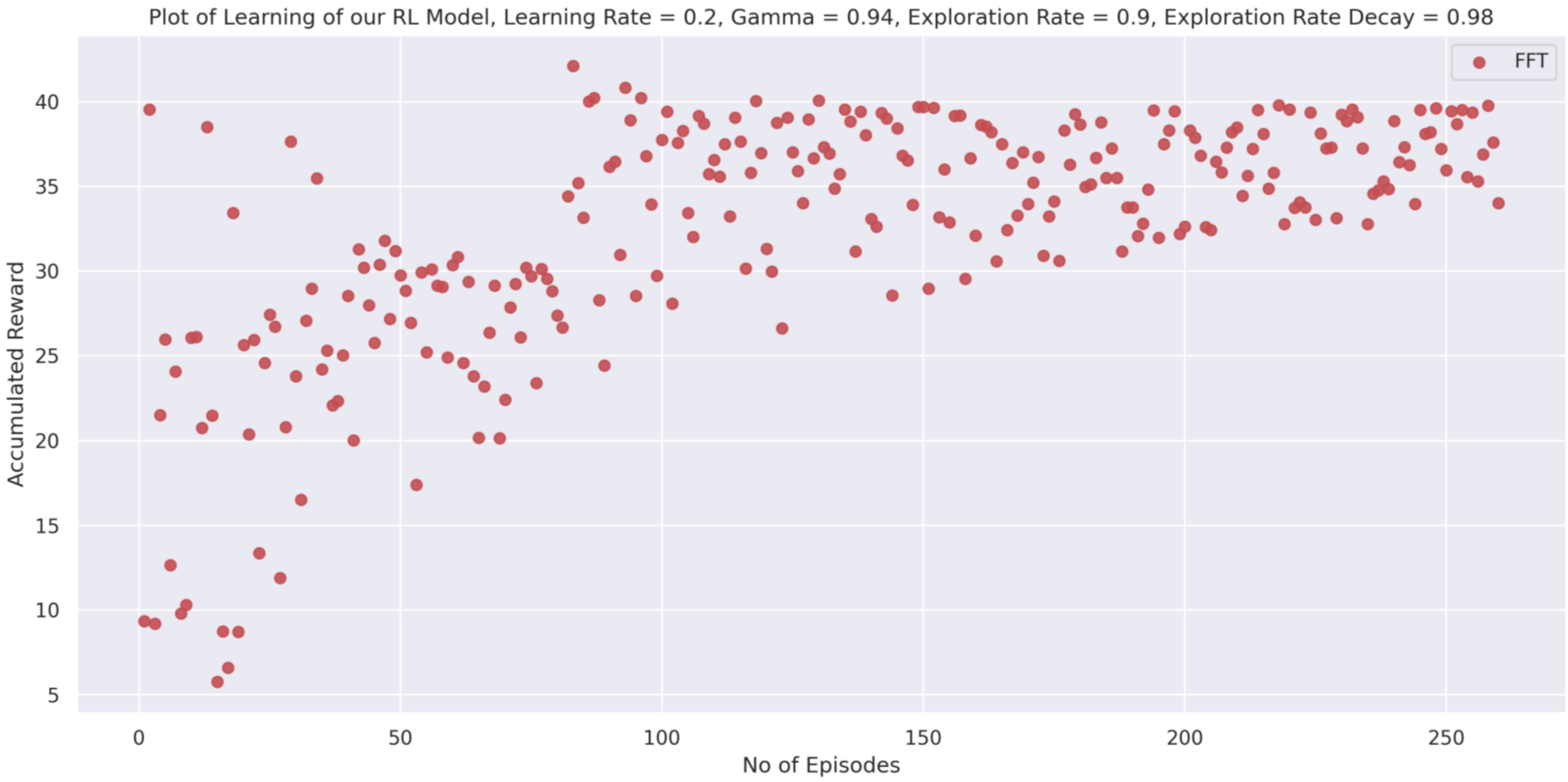
Empirical Analysis : Time



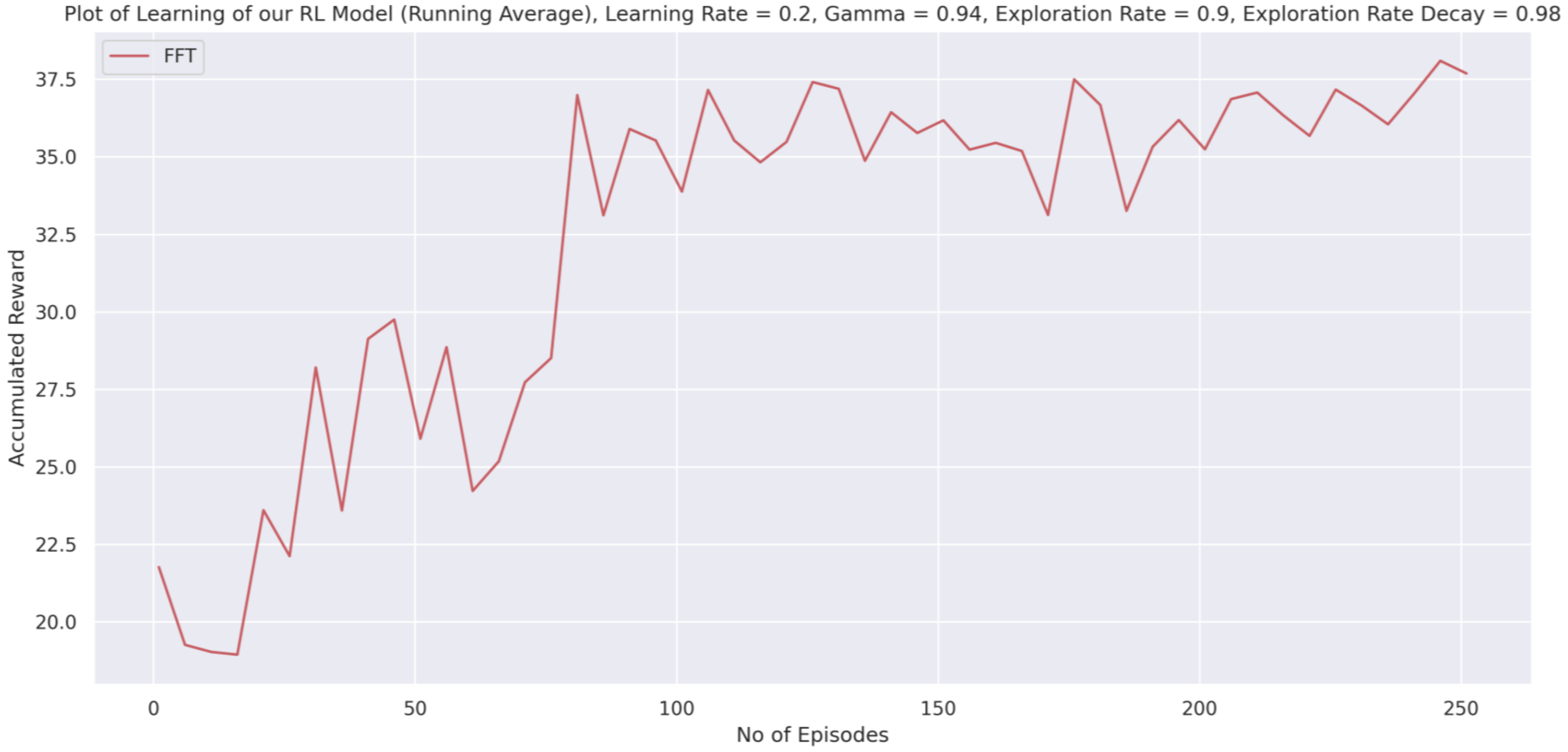
Empirical Analysis: Rewards v/s State Action Pair



Empirical Analysis: Learning



Empirical Analysis: Learning



Future Research & Scope of Improvement

- We can include additional parameters in our MDP, which take into account the network strength, battery status etc.
- Raw Sockets implementation instead of TCP which will ensure a better control over file transfer.
- Quantization of file size into states can be more fine grained if large no. of examples can be found. Also, the Q-table maintenance can be centralized and to have optimal starting values across servers.
- Logging options to improve the performance of the model further, including a centralized computational unit which receives the transaction logs of all users.

TIMELINE

FFT Protocol Design
& Implementation

Mobile Application
Development (Client End)

Using RL to
improve the overall throughput + Testing

11 Days	14 Days	12 Days	6 Days	23 Days

PC Application
Development (Server End)

Using the state-of-art techniques
to achieve seamless file transfer

TEAM SYNCX

Apurv Purohit
CSE Undergrad’22, IIT Jammu

Android Client Application Development

Navdeep Singh
CSE Undergrad’22, IIT Jammu

Protocol and Server Application Development

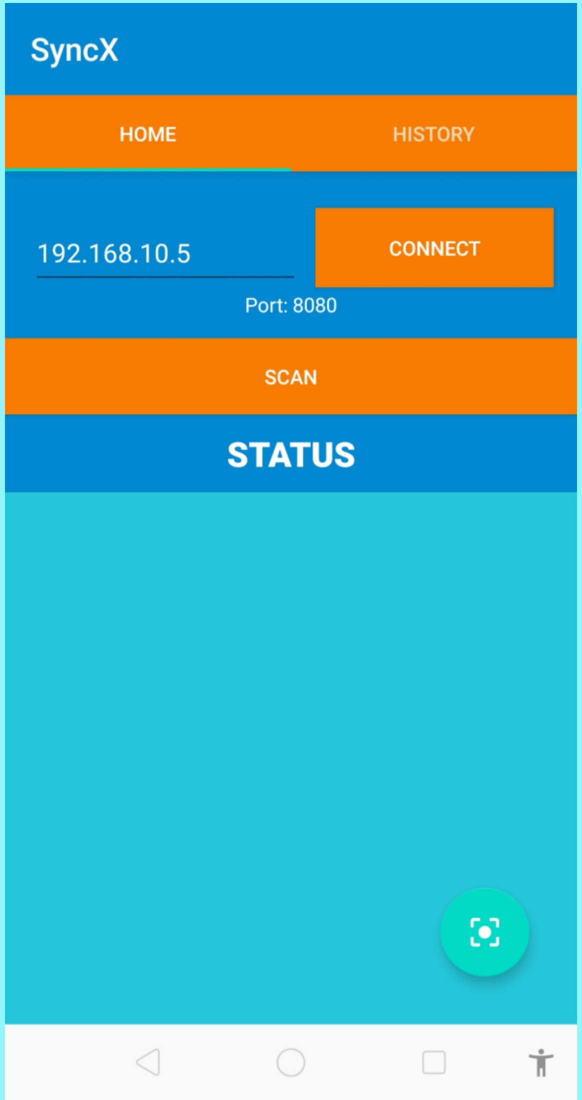
Pradyumn Sharma
CSE Undergrad’22, IIT Jammu

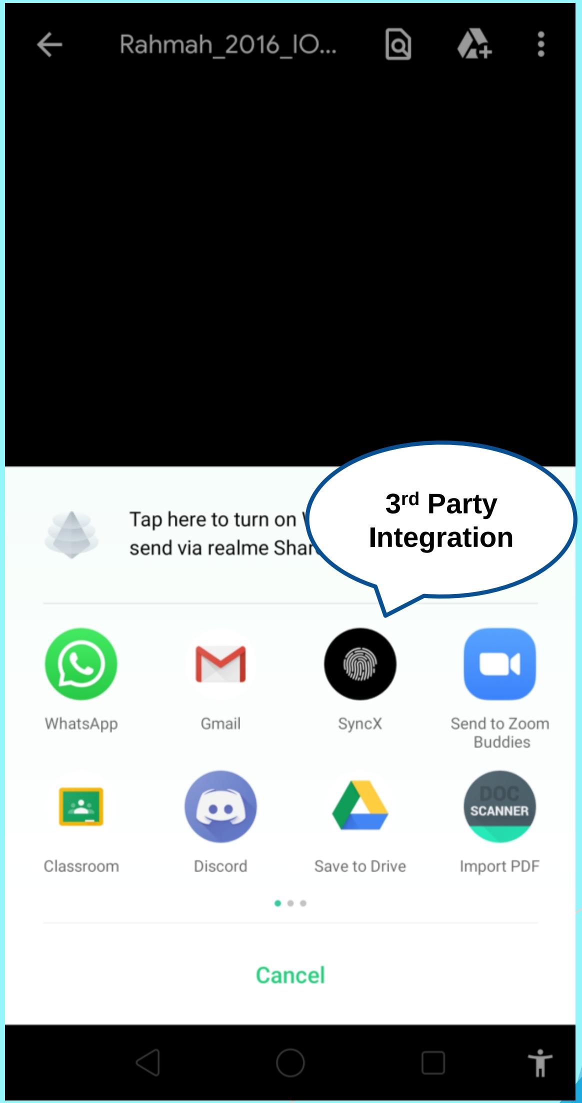
Reinforcement Learning Model Development

**The protocol design was discussed and worked upon by the whole team.*

GALLERY


SyncX


SyncX
HOME
HISTORY
192.168.10.5
CONNECT
Port: 8080
SCAN
STATUS


Rahmah_2016_IO...
Tap here to turn on
send via realme Shar
3rd Party Integration
WhatsApp
Gmail
SyncX
Send to Zoom Buddies
Classroom
Discord
Save to Drive
Import PDF
Cancel

THANK YOU

QUESTIONS?