# RemoteDictionary Task

## Overview

You should implement a client-server application that uses a minimalist remote dictionary access protocol, defined bellow.

## Protocol

The protocol defines a set of operations over the remote dictionary object *RemoteDictionary*.

The data types used are specified only at the requirements level. You may implement them as you wish, as long as the implementation meets the requirements.

### Networking

The protocol must be implemented using TCP.

### Request-response model

The client sends a request

The server handles it

The server passes a response to the client.

### Serialization

Any serialization method may be used as long as the *RemoteDictionary* requirements are met.
Possible options are: FlatBuffers, Protocol Buffers, Json, etc.

### RemoteDictionary

In this section we define the *RemoteDictionary* requirements per operation.

#### RemoteDictionary::get(key: string) → GetResponse

Lookup the specified key and pass back a GetResponse.
The GetResponse type must be able to convey either success or failure.
The successful case includes the value found in the dictionary at the specified key.
The failure case includes the reason behind it.

#### RemoteDictionary::set(key: string, value: string) → SetResponse

Store the value in the dictionary at the specified key.
The SetResponse type must be able to convey either success or failure.
In case of failure the reason behind it must be included.

#### RemoteDictionary::stats() → StatsResponse

Get dictionary statistics, should include:
- total number of get operations
- total number of successful get operations
- total number of failed get operations

## Server

The server should implement the *RemoteDictionary* data structure and the network exposed access protocol with client handling.

The implementation should be non-blocking. Open source networking libraries may be used (examples: libuv or boost).

**Optionally** is may have the following features:

| Optional Task | Description |
|---|---|
| Bloom filter optimization | Avoid accessing the dictionary directly on get operations.<br>This feature should be configurable(enable/disable) at server startup. |
| Benchmark | Design a benchmark capable to measure the latency and throughput of the server. |

## Client

A client should also be implemented in order to test the server.

The client should be a basic library that allows concurrent requests to the server.