15-440 Project 1

15-440项目1

Project 1: Distributed Bitcoin Miner

项目1:分布式比特币矿工

1 Overview

1概述

This project will consist of the following two parts:

该项目将包括以下两个部分:

• Part A: Implement the Live Sequence Protocol, a homegrown protocol for providing reliable communication with simple client and server APIs on top of the Internet UDP protocol.Part A is due on Monday, February 24 at 11:59pm.

第一部分:实现实时序列协议，这是一个自主开发的协议，用于在互联网UDP协议的基础上提供与简单客户端和服务器应用程序接口的可靠通信。甲部分应于2月24日星期一晚上11:59提交。

• Part B: Implement a Distributed Bitcoin Miner.Part B will be due on Monday, March 3 at 11:59pm.

第二部分:实现分布式比特币矿工。乙部分将于3月3日星期一晚上11:59到期。

The starter code for this project is hosted as a read-only repository on GitHub.For instructions on how to build, run, test, and submit your server implementation, see the README.md file in the project's root directory.To clone a copy, execute the following Git command:

该项目的起始代码在GitHub上作为只读存储库托管。有关如何构建、运行、测试和提交服务器实现的说明，请参见项目根目录中的README.md文件。要克隆副本，请执行以下Git命令:

git clone https://github.com/cmu440/p1.git

git克隆https://github.com/cmu440/p1.git

We will begin by discussing part A of this project, in which you will implement the Live Sequence Protocol.In your implementation, you will incorporate features required to create a robust system, handling lost or duplicated Internet packets, as well as failing clients and servers.You will also learn the value of creating a set of layered abstractions in bridging between low-level network protocols and high-level applications.

我们将从讨论这个项目的第一部分开始，您将在其中实现实时序列协议。在您的实现中，您将结合创建健壮系统所需的功能，处理丢失或重复的互联网数据包，以及出现故障的客户端和服务器。您还将了解在低级网络协议和高级应用程序之间建立一套分层抽象的价值。

2 Part A: LSP Protocol

2第一部分:物流服务协议

The low-level Internet Protocol (IP) provides what is referred to as an "unreliable data-gram" service, allowing one machine to send a message as a packet to another, but with the possibility that the packet will be delayed, dropped, or duplicated.In addition, as an IP packet hops from one network node to another, its size is limited to a specified maximum number of bytes.Typically, packets of up to 1, 500 bytes can safely be transmitted along any routing path, but going beyond this can become problematic.

低级互联网协议提供所谓的“不可靠的数据图表”服务，允许一台机器以数据包的形式向另一台机器发送消息，但数据包可能会被延迟、丢弃或复制。此外，当一个数据包从一个网络节点跳至另一个网络节点时，它的大小被限制在指定的最大字节数。通常，最多1，500字节的数据包可以安全地沿任何路由路径传输，但超出此范围可能会有问题。

Very few applications use IP directly.Instead, they are typically written to use UDP or TCP:

很少有应用程序直接使用知识产权。相反，它们通常是使用UDP或TCP编写的:

1

1

15-440 Project 1

15-440项目1

UDP: The "User Datagram Protocol."Also an unreliable datagram service, but it allows

UDP:“用户数据报协议”也是一种不可靠的数据报服务，但是它允许

packets to be directed to different logical destinations on a single machine, known as ports.This makes it possible to run multiple clients or servers on a single machine.This function is often called multiplexing.

数据包将被定向到一台机器上的不同逻辑目的地，称为端口。这使得在一台机器上运行多个客户端或服务器成为可能。这种功能通常被称为多路复用。

TCP: The "Transmission Control Protocol" offers a reliable, in-order stream abstraction.

传输控制协议:传输控制协议提供了可靠的、有序的流抽象。

Using TCP, a stream of arbitrary-length messages is transmitted by breaking each message into (possibly multiple) packets at the source and then reassembling them at the destination.TCP handles such issues as dropped packets, duplicated packets, and preventing the sender from overwhelming both Internet bandwidth and the buffering capabilities at the destination.

使用TCP，通过在源端将每个消息分成(可能是多个)包，然后在目的端重新组装，来传输任意长度的消息流。TCP处理诸如丢弃数据包、重复数据包等问题，并防止发送方占用互联网带宽和目的地的缓冲能力。

Your task for Part A of this project is to implement the Live Sequence Protocol (LSP).LSP provides features that lie somewhere between UDP and TCP, but it also has features not found in either protocol:

本项目第一部分的任务是实现实时序列协议。LSP提供了介于UDP和TCP之间的特性，但它也有两种协议都没有的特性:

• Unlike UDP or TCP, it supports a client-server communication model.

与UDP或TCP不同，它支持客户机-服务器通信模型。

• The server maintains connections between a number of clients, each of which is iden-tified by a numeric connection identifier.

服务器维护多个客户端之间的连接，每个客户端都由数字连接标识符标识。

• Communication between the server and a client consists of a sequence of discrete messages in each direction.

服务器和客户端之间的通信由每个方向的一系列离散消息组成。

• Message sizes are limited to fit within single UDP packets (around 1,000 bytes).

消息大小被限制在单个UDP数据包内(大约1，000字节)。

• Messages are sent reliably: a message sent over the network must be received exactly once, and messages must be received in the same order they were sent.

消息发送可靠:通过网络发送的消息必须准确接收一次，并且消息必须按照发送的顺序接收。

• The server and clients monitor the status of their connections and detect when the other side has become disconnected.

服务器和客户端监控其连接状态，并检测对方何时断开连接。

The following sections will describe LSP in-depth.We begin by describing the protocol in terms of the messages that flow between the server and its clients.

以下部分将深入描述LSP。我们首先根据服务器和客户端之间的消息来描述协议。

2.1 LSP Overview

2.1 LSP概述

In LSP, communication between a server and client consists of a sequence of discrete messages sent in each direction.Each message sent over an LSP connection is made up of the following four values:

在LSP中，服务器和客户机之间的通信由一系列向各个方向发送的离散消息组成。通过LSP连接发送的每条消息由以下四个值组成:

2

2

15-440 Project 1

15-440项目1

Message Type: An integer constant identifying one of three possible message types:

消息类型:标识三种可能消息类型之一的整数常数:

Connect: Sent by a client to establish a connection with the server.Data: Sent by either a client or the server to transmit information.

连接:由客户端发送以建立与服务器的连接。数据:由客户端或服务器发送以传输信息。

Ack: Sent by either a client or the server to acknowledge a Connect or Data message.

确认:由客户端或服务器发送以确认连接或数据消息。

Connection ID: A positive, non-zero number that uniquely identifies the client-server connection.

连接标识:唯一标识客户机-服务器连接的非零正数。

Sequence Number: A positive, non-zero number that is incremented with each data message sent, starting with the number 0 for the initial connection request.

序列号:一个正数，非零值，随着发送的每条数据消息而递增，从初始连接请求的数字0开始。

Payload: A sequence of bytes, with a format determined by the application.

有效载荷:字节序列，格式由应用程序决定。

In the sections that follow, we will use the following notation to indicate the different possible messages that can be sent between a client and a server (note that both Connect and Ack messages have payload values of nil):

在接下来的章节中，我们将使用以下符号来表示客户端和服务器之间可能发送的不同消息(请注意，连接和Ack消息的有效负载值都为零):

• (Connect, 0, 0): Connection request.It must have ID 0 and sequence number 0.

(连接，0，0):连接请求。它必须具有识别码0和序号0。

• (Data, id, sn, D): Data message with ID id, sequence number sn, and payload D.

(数据、标识、序列号、数据):具有标识、序列号和有效载荷数据的数据消息

• (Ack, id, sn): Ack message with ID id, and sequence number sn.

(Ack，id，sn):带有ID id和序列号sn的Ack消息。

2.1.1 Establishing a connection

2.1.1建立连接

Before data can be sent between a client and server, a connection must first be made.The client initiates a connection by sending a connection request to the server.In response, the server generates and assigns an ID that uniquely identifies the new client-server connection, and sends the client an acknowledgment message containing this new ID, the sequence number 0, and a nil payload.Figure 1 illustrates how a connection is established.

在客户端和服务器之间发送数据之前，必须先建立连接。客户端通过向服务器发送连接请求来启动连接。作为响应，服务器生成并分配唯一标识新的客户机-服务器连接的标识，并向客户机发送包含该新标识、序列号0和零有效载荷的确认消息。图1说明了如何建立连接。

Your server may choose any scheme for generating new connection IDs.Our implementa-tion simply assigns IDs sequentially starting with 1.

您的服务器可以选择任何方案来生成新的连接标识。我们的实现简单地从1开始依次分配标识。

2.1.2 Sending & acknowledging data

2.1.2发送和确认数据

Once a connection is established, data may be sent in both directions (i.e. from client to server or from server to client) as sequences of discrete data messages.Figure 2 gives an example of a normal communication sequence between the server and a client.The figure illustrates the transmission of three data messages from the client to the server, and

一旦建立了连接，数据可以作为离散数据消息序列在两个方向上发送(即从客户端到服务器或从服务器到客户端)。图2给出了服务器和客户端之间正常通信序列的示例。该图说明了三条数据消息从客户端到服务器的传输

3

3

15-440 Project 1

15-440项目1

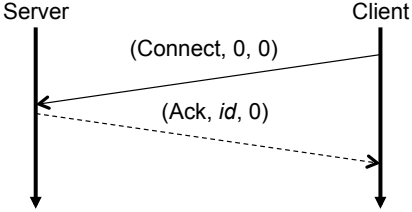


Figure 1: Establishing a connection.A client sends a connection request to the server, which responds to the client with an acknowledgment message containing the connection's unique ID.The vertical lines with downward arrows denote the passage of time on both the client and the server, while the lines crossing horizontally denote messages being sent between the two.

图1:建立连接。客户端向服务器发送连接请求，服务器用包含连接唯一标识的确认消息来响应客户端。带有向下箭头的垂直线表示客户机和服务器上时间的流逝，而水平交叉的线表示两者之间正在发送的消息。

one data message from the server to the client.Note that all messages are acknowledged, and that the client and server maintain their own series of sequence numbers, independent of the other.Also note that it is entirely possible for one side to receive data messages from the other as it is waiting for these acknowledgments—the client and server operate asynchronously, and there is no guarantee that packets arrive in the same order they are sent.

一条从服务器到客户端的数据消息。请注意，所有消息都被确认，并且客户端和服务器保持它们自己的一系列序列号，彼此独立。还要注意，在等待这些确认时，一方完全有可能从另一方接收数据消息—客户端和服务器异步运行，并且不能保证数据包以相同的发送顺序到达。

Like TCP, LSP includes a sliding window protocol.The sliding window represents an upper bound on the range of messages that can be sent without acknowledgment.This upper bound is referred to as the "window size," which we denote with the symbol ω.For example, if ω = 1, every message must be acknowledged before the next message can be sent.If ω = 2, up to two messages can be sent without acknowledgment.That is, a third message cannot be sent until the first message is acknowledged, a fourth message cannot be sent until the first and second messages are acknowledged, etc.

像TCP一样，LSP包括滑动窗口协议。滑动窗口表示无需确认即可发送的消息范围的上限。这个上限称为“窗口大小”，我们用符号ω表示。例如，如果ω = 1，则必须先确认每条消息，然后才能发送下一条消息。如果ω = 2，最多可以发送两条消息，而无需确认。也就是说，在第一消息被确认之前不能发送第三消息，在第一和第二消息被确认之前不能发送第四消息，等等。

Note that the range of messages that are allowed to be sent without acknowledgment is fixed.If the oldest message that has not yet been acknowledged has sequence number n, then only messages with sequence numbers n through n + ω − 1 (inclusive) may be sent.Only when the oldest (leftmost) messages are acknowledged can the window slide over to the right, possibly allowing more recent data messages to be sent.

请注意，允许在没有确认的情况下发送的消息范围是固定的。如果尚未确认的最早消息的序列号为n，则只能发送序列号为n至n+ω1(含)的消息。只有当最早的(最左边的)消息被确认时，窗口才能滑动到右边，可能允许发送更新的数据消息。

In addition to maintaining a sliding window for non-acknowledged data messages, both clients and servers will also need to maintain a sliding window for their ω most recently sent acknowledgments, as they might need to be re-sent at a later time (you may assume that both the client and server will have the same window sizes).The reasons for this additional book-keeping will become clear in the next section, in which we discuss epoch events.

除了为未确认的数据消息维护一个滑动窗口之外，客户端和服务器还需要为它们最近发送的确认维护一个滑动窗口，因为它们可能需要在以后重新发送(您可以假设客户端和服务器将具有相同的窗口大小)。这种额外记账的原因将在我们讨论时代事件的下一节中变得清楚。

4

4

15-440 Project 1

15-440项目1

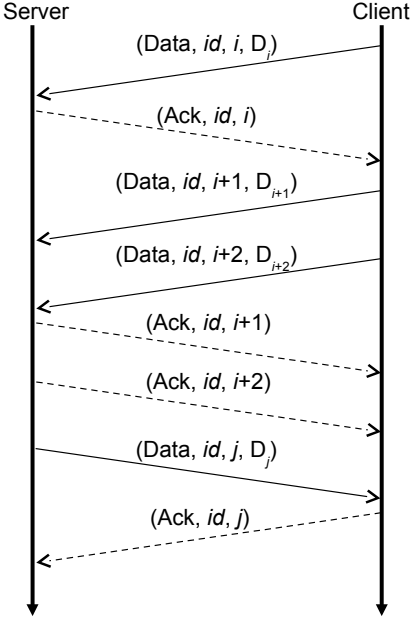


Figure 2: Sending & acknowledging data.Data may be sent from both the client or the server, and all data messages must be acknowledged.

图2:发送和确认数据。数据可以从客户端或服务器发送，所有数据消息都必须得到确认。

2.1.3 Epoch events

2.1.3新纪元事件

Unfortunately, the basic protocol described so far is not robust.On one hand, its sequence numbers makes it possible to detect when a message has been dropped or duplicated.However, if any message—connection request, data message, or acknowledgment—gets dropped, the linkage in either one or both directions will stop working, with both sides waiting for messages from the other.

不幸的是，迄今为止描述的基本协议并不健壮。一方面，它的序列号使得检测消息何时被丢弃或复制成为可能。但是，如果任何消息(连接请求、数据消息或确认)被丢弃，一个方向或两个方向的链接将停止工作，双方都在等待另一方的消息。

To make LSP robust, we incorporate a simple time trigger into the servers and clients.Timers fire periodically on both the clients and servers, dividing the flow of time for each into a sequence of epochs.We refer to the time interval between epochs as the epoch duration, which we denote with the symbol δ.Our default value for δ is 2, 000 milliseconds, although this can be varied.

为了使LSP健壮，我们在服务器和客户端中加入了一个简单的时间触发器。计时器会在客户端和服务器上定期触发，将每个客户端的时间流划分为一系列时期。我们称各时期之间的时间间隔为时期持续时间，用符号δ表示。我们的缺省δ值是2000毫秒，尽管这可以变化。

When the epoch timer fires, a client takes the following actions:

当纪元计时器触发时，客户端会采取以下操作:

5

5

15-440 Project 1

15-440项目1

• If the client's connection request has not yet been acknowledged by the server, then resend the connection request.

如果服务器尚未确认客户端的连接请求，则重新发送连接请求。

• If the connection request has been sent and acknowledged, but no data messages have been received, then send an acknowledgment with sequence number 0.

如果连接请求已经发送并确认，但没有收到数据消息，则发送序列号为0的确认。

• For each data message that has been sent but not yet acknowledged, resend the data

对于已发送但尚未确认的每条数据消息，重新发送数据

message.

信息。

• Resend an acknowledgment message for each of the last ω (or possibly fewer) distinct data messages that have been received.

为每个最后接收到的ω(或可能更少)不同数据消息重新发送确认消息。

The server performs a similar set of actions for each of its connections:

服务器对其每个连接执行一组类似的操作:

• If no data messages have been received from the client, then resend an acknowledg-ment message for the client's connection request.

如果没有从客户端收到数据消息，则为客户端的连接请求重新发送确认消息。

• For each data message that has been sent, but not yet acknowledged, resend the data

对于已发送但尚未确认的每条数据消息，重新发送数据

message.

信息。

• Resend an acknowledgment message for each of the last ω (or possibly fewer) distinct data messages that have been received.

为每个最后接收到的ω(或可能更少)不同数据消息重新发送确认消息。

Figure 3 illustrates how the epoch events make up for failures by the normal communication.We show the occurrence of an epoch event as a large black oval on each time line.In this example, the client attempts to send data Di , but the acknowledgment message gets dropped.In addition, the server attempts to send data Dj , but the data message gets dropped.When the epoch timer triggers on the client, it will send an acknowledgment of data message j − 1, the last data message received, and it will resend data Di .

图3说明了纪元事件如何弥补正常通信的失败。我们将一个时代事件的发生显示为每个时间线上的一个大的黑色椭圆形。在这个例子中，客户端试图发送数据Di，但是确认消息被丢弃。此外，服务器试图发送数据Dj，但数据消息被丢弃。当epoch定时器在客户端触发时，它将发送数据消息J1的确认，这是接收到的最后一条数据消息，并且它将重新发送数据Di。

Assuming the server has an epoch event around the same time (there is no requirement that they occur simultaneously), we can see that the server will send an acknowledgment for data Di , and it will resend data Dj .

假设服务器大约在同一时间有一个纪元事件(不要求它们同时发生)，我们可以看到服务器将发送数据Di的确认，并且它将重新发送数据Dj。

We can see in this example that duplicate messages can occur: for example, the server might receive two copies of Di .For most cases, we can use sequence numbers to detect any duplications: Each side maintains a counter indicating which sequence number it expects next and discards any message that does not match the expected window range of sequence numbers.One case of duplication requires special attention, however: it is possible for the client to send multiple connection requests, with one or more requests or acknowledgments being dropped.The server must track the host address and port number of each connection request and discard any for which that combination of host and port already has an established connection.

在这个例子中，我们可以看到可能会出现重复的消息:例如，服务器可能会收到两个Di副本。对于大多数情况，我们可以使用序列号来检测任何重复:每一侧都有一个计数器，指示它期望下一个序列号，并丢弃任何与序列号的预期窗口范围不匹配的消息。但是，有一种情况需要特别注意:客户端可能会发送多个连接请求，其中一个或多个请求或确认会被丢弃。服务器必须跟踪每个连接请求的主机地址和端口号，并丢弃主机和端口的组合已经建立连接的任何请求。

6

6

15-440 Project 1

15-440项目1

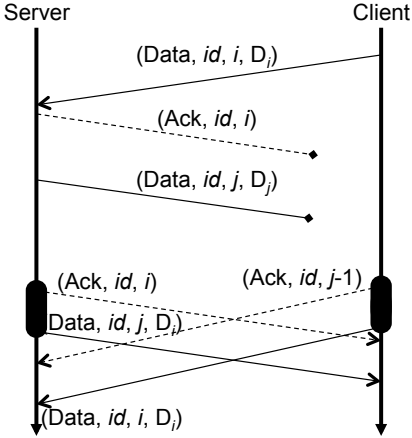


Figure 3: Epoch events.Both sides resend acknowledgments for the most recently received data, and (possibly) resend any unacknowledged data.

图3:新纪元事件。双方重新发送对最近接收到的数据的确认，并(可能)重新发送任何未确认的数据。

One feature of this epoch design is that there will be at least one message transmitted in each direction between client and server on every epoch.As a final feature, we will track at the endpoint of each connection the number of epochs that have passed since a message (of any type) was received from the other end.Once this count reaches a specified epoch limit, which we denote with the symbol K, we will declare that the connection has been lost.Our implementation uses a default value of 5 for K;thus, if nothing is received from the other end of an established connection over a total period of K δ seconds, then the connection should be assumed lost.

这种时代设计的一个特征是，在每个时代，客户端和服务器之间至少有一条消息在每个方向上传输。最后一个特性是，我们将在每个连接的端点跟踪从另一端接收到消息(任何类型)以来经过的时期数。一旦该计数达到指定的纪元限制(用符号K表示)，我们将声明连接已丢失。我们的实现对K使用默认值5；因此，如果在总共K δ秒的时间段内没有从已建立连接的另一端接收到任何信息，则应该假设该连接丢失。

2.2 LSP API

2.2 LSP原料药

We will now describe LSP from the perspective of a Go programmer.You must implement the exact API discussed below to facilitate automated testing, and to ensure compatibility between different implementations of the protocol.

我们现在将从围棋程序员的角度来描述LSP。您必须实现下面讨论的确切的应用编程接口，以便于自动化测试，并确保协议的不同实现之间的兼容性。

The LSP API can be found in the lsp package (included as part of the starter code).This file defines several exported structs, interfaces, and constants, and also provides detailed documentation describing the intent and expected behavior of every aspect of the API.Consult it regularly!

lsp应用编程接口可以在LSP包中找到(作为起始代码的一部分)。该文件定义了几个导出的结构、接口和常数，并且还提供了描述应用编程接口每个方面的意图和预期行为的详细文档。定期咨询！

7

7

15-440 Project 1

15-440项目1

2.2.1 LSP Messages

2.2.1 LSP消息

The different LSP message types are defined as integer constants below:

不同的LSP消息类型定义为以下整数常数:

type MsgType int

类型MsgType int

const (

常量(

MsgConnect MsgType = iota // Connection request from client.MsgData // Data message from client or server.MsgAck // Acknowledgment from client or server.

MsgConnect MsgType = iota //来自客户端的连接请求。MsgData //来自客户端或服务器的数据消息。MsgAck //来自客户端或服务器的确认。

)

)

Each LSP message consists of four fields, and is declared as a Go struct:

每个LSP消息由四个字段组成，并声明为Go结构:

type Message struct {

键入消息结构

Type MsgType // One of the message types listed above.ConnID int // Unique client-server connection ID.SeqNum int // Message sequence number.Payload []byte // Data message payload.

键入MsgType //上面列出的消息类型之一。唯一的客户端-服务器连接标识。SeqNum int //消息序列号。有效负载[]字节//数据消息有效负载。

}

}

To send a Message over the network, you must first convert the structure to a UDP packet by marshalling it into a sequence of bytes.This can be done in Go using the Marshal function in the json package.

要通过网络发送消息，您必须首先通过将该结构编组为字节序列，将其转换为UDP数据包。这可以在Go中使用json包中的封送函数来完成。

2.2.2 LSP Parameters

2.2.2 LSP参数

For both the client and the server, the API provides a mechanism to specify the epoch limit K, the epoch duration δ, and the sliding window size ω when a client or server is first created.These parameters are encapsulated in the following struct:

对于客户端和服务器，应用编程接口都提供了一种机制，用于在首次创建客户端或服务器时指定时段限制K、时段持续时间δ和滑动窗口大小ω。这些参数封装在以下结构中:

type Params struct {

Params结构类型{

EpochLimit int // Default value is 5.EpochMillis int // Default value is 2000.WindowSize int // Default value is 1.

EpochLimit int //默认值为5。EpochMillis int //默认值为2000。窗口大小int //默认值为1。

}

}

2.2.3 LSP Client API

2.2.3物流服务提供商客户端应用编程接口

An application calls the NewClient to set up and initiate the activities of a client.The function blocks until a connection with the server has been made, and returns a non-nil error if the connection could not be established.The function is declared as follows:

应用程序调用新客户机来设置和启动客户机的活动。函数会一直阻塞，直到与服务器建立连接，如果无法建立连接，则返回非零错误。该函数声明如下:

8

8

15-440 Project 1

15-440项目1

func NewClient(hostport string, params \*Params) (Client, error)

func NewClient(主机端口字符串，参数\*参数)(客户端，错误)

The LSP client API is defined by the Client interface, which declares the methods below:

LSP客户端应用编程接口由客户端接口定义，该接口声明了以下方法:

ConnID() int

ConnID() int

Read() ([]byte, error) Write(payload []byte) error Close() error

读取()([)字节，错误)写入(有效负载[)字节)错误关闭()错误

The Client interface hides all of the details of establishing a connection, acknowledging messages, and handling epochs from the application programmer.Instead, applications simply read and write data messages to the network by calling the Read and Write methods respectively.The connection can be signaled for shutdown by calling Close.

客户端界面对应用程序程序员隐藏了建立连接、确认消息和处理时期的所有细节。相反，应用程序只需分别调用读取和写入方法，即可向网络读取和写入数据消息。可以通过调用关闭来发出关闭连接的信号。

A few other details are worth noting:

其他一些细节值得注意:

• Read should block until data has been received from the server and is ready to be returned.It should return a non-nil error if either (1) the connection has been explicitly closed, or (2) the connection has been lost due to an epoch timeout and no other messages are waiting to be returned by Read.

读取应该被阻止，直到从服务器接收到数据并准备返回。如果(1)连接已被显式关闭，或者(2)连接因纪元超时而丢失，并且没有其他消息等待读取返回，它应该返回非零错误。

• Write should not block.It should return a non-nil error only if the connection to the server has been lost.

写入不应被阻止。只有当与服务器的连接丢失时，它才应该返回非零错误。

• Close should not forcefully terminate the connection, but instead should block until all pending messages to the server have been sent and acknowledged (of course, if the connection is suddenly lost during this time, the remaining pending messages should simply be discarded).

关闭不应该强制终止连接，而是应该阻止，直到发送和确认所有到服务器的挂起消息(当然，如果在此期间连接突然丢失，剩下的挂起消息应该被丢弃)。

• No goroutine should be left running after Close has returned (this will cause our Autolab tests to hang).

关闭返回后，任何goroutine都不应继续运行(这将导致我们的自动拉布测试挂起)。

• You may assume that Read, Write, and Close will not be called after Close has been called.

您可以假设在调用“关闭”后不会调用“读取”、“写入”和“关闭”。

For detailed documentation describing the intent and expected behavior of each function and method, consult the client\_api.go and client\_impl.go files.

有关描述每个函数和方法的意图和预期行为的详细文档，请参考client\_api.go和client\_impl.go文件。

2.2.4 LSP Server API

2.2.4 LSP服务器应用编程接口

The API for the server is similar to that for an LSP client, with a few minor differences.An application calls the NewServer to set up and initiate a LSP server.However, unlike

服务器的应用编程接口类似于LSP客户机的应用编程接口，只是有一些小的不同。应用程序调用新闻服务器来设置和启动LSP服务器。然而，不像

9

9

15-440 Project 1

15-440项目1

NewClient, this function should not block.Instead, it should simply begin listening on the specified port and spawn one or more goroutines to handle things like accepting incoming client connections, triggering epoch events at fixed intervals, etc.If there is a problem setting up the server, the function should return a non-nil error.The function is declared as follows:

NewClient，这个函数不应该阻塞。相反，它应该简单地开始监听指定的端口，并产生一个或多个goroutines来处理诸如接受传入的客户端连接、以固定的时间间隔触发纪元事件等事情。如果设置服务器有问题，函数应该返回一个非零错误。该函数声明如下:

func NewServer(port int, params \*Params) (Server, error)

函数新闻服务器(端口int，参数\*参数)(服务器，错误)

The LSP server API is defined by the Server interface, which declares the following meth-ods:

LSP服务器应用编程接口由服务器接口定义，该接口声明了以下方法:

Read() (int, []byte, error)

读取()(int，[)字节，错误)

Write(connID int, payload []byte) error CloseConn(connID int) error Close() error

写(连接符int，有效负载[]字节)错误关闭连接(连接符int)错误关闭()错误

Similar to the Client, the Server interface allows applications to both read and right data to its clients.Note, however, that because the server can be connected to several LSP clients at once, the Write and CloseConn methods take a client's unique connection ID as an argument, indicating the connection that should be written to or that should be closed.

与客户端类似，服务器接口允许应用程序读取数据并将其正确地发送给客户端。但是，请注意，因为服务器可以同时连接到多个LSP客户端，所以Write和CloseConn方法将客户端的唯一连接标识作为参数，指示应该写入或应该关闭的连接。

A few other details are worth noting:

其他一些细节值得注意:

• Read should block until data has been received from some client and is ready to be returned.It should return a non-nil error if either (1) the connection to some client has been explicitly closed, or (2) the connection to some client has been lost due to an epoch timeout and no other messages from that client are waiting to be returned by Read.This method should not return data from a connection that was explicitly closed by a call to CloseConn.

读取应该被阻止，直到从某个客户端接收到数据并准备返回。如果(1)到某个客户端的连接已被显式关闭，或者(2)到某个客户端的连接由于纪元超时而丢失，并且没有来自该客户端的其他消息等待读取返回，则它应该返回非零错误。此方法不应从通过调用CloseConn显式关闭的连接返回数据。

• The Write and CloseConn methods should not block, and should both return a non-nil error value only if the specified connection ID does not exist.

写入和关闭连接方法不应阻塞，并且仅当指定的连接标识不存在时，两者都应返回非零错误值。

• Close should block until all pending messages to each client have been sent and acknowledged (of course, if a client that still has pending messages is suddenly lost during this time, the remaining pending messages should simply be discarded).

关闭应该会被阻止，直到发送和确认到每个客户端的所有未决消息(当然，如果在此期间仍有未决消息的客户端突然丢失，剩余的未决消息应该被丢弃)。

• No goroutine should be left running after Close has returned (this will cause our Autolab tests to hang).

关闭返回后，任何goroutine都不应继续运行(这将导致我们的自动拉布测试挂起)。

• You may assume that after CloseConn has been called, neither Write nor CloseConn will be called on that same connection ID again.You may also assume that no other Server methods calls will be made after Close has been called.

您可以假设在调用CloseConn之后，无论是写连接还是CloseConn都不会在同一个连接标识上再次被调用。您还可以假设在调用关闭后不会进行其他服务器方法调用。

10

10

15-440 Project 1

15-440项目1

For detailed documentation describing the intent and expected behavior of each function and method, consult the server\_api.go and server\_impl.go files.

有关描述每个函数和方法的意图和预期行为的详细文档，请参考server\_api.go和server\_impl.go文件。

2.3 Starter Code

2.3启动器代码

The starter code for part A of this project can be found in the p1/src/github.com/cmu440/ directory, and is organized as follows:

本项目第一部分的起始代码可在p1/src/github.com/cmu440/目录中找到，组织如下:

• The lsp package contains the API, tests, and the starter code you will need to complete:

lsp包包含您需要完成的应用编程接口、测试和起始代码:

The client\_api.go, server\_api.go, message.go, and params.go files col-lectively define the LSP API.To facilitate automated testing, you must not modify these files.

客户机应用程序接口、服务器应用程序接口、消息应用程序接口和参数应用程序接口文件共同定义了LSP应用程序接口。为了便于自动化测试，您不得修改这些文件。

The client\_impl.go file contains a skeletal implementation of the Client that you will write.

client\_impl.go文件包含您将编写的客户端的框架实现。

The server\_impl.go file contains a skeletal implementation of the Server that you will write.

服务器文件包含您将要编写的服务器的框架实现。

The \*\_test.go files contain the tests that we will run on Autolab to test and evaluate your final submission.

\*\_test.go文件包含我们将在Autolab上运行的测试，以测试和评估您的最终提交。

• The lspnet package contains all of the UDP operations you will need to complete this assignment.Under-the-hood, the lspnet package provides some additional func-tionalities that allow us to more easily grade the robustness of your implementation.

lspnet包包含完成此任务所需的所有UDP操作。实际上，lspnet包提供了一些额外的功能，使我们能够更容易地对您的实现的健壮性进行分级。

• The srunner (server-runner) and crunner (client-runner) packages each provide sim-ple executable programs that you may use for your own testing purposes.Instruc-tions on how these programs can be used are posted in the project's README.md file on GitHub.

srunner(服务器运行程序)和crunner(客户端运行程序)包都提供了简单的可执行程序，您可以将它们用于自己的测试目的。关于如何使用这些程序的说明发布在GitHub上项目的README.md文件中。

We have also provided pre-compiled executables of the srunner and crunner programs discussed above that you can use for testing.The binaries were compiled against our reference LSP implementation, so you might find them useful in the early stages of the development process (for example, if you wanted to test your Client implementation but haven't finished implementing the Server yet, etc.).Instructions on how these programs can be used are posted in the project's README.md file on GitHub.

我们还提供了上述srunner和crunner程序的预编译可执行文件，您可以使用它们进行测试。二进制文件是根据我们的参考LSP实现编译的，因此您可能会发现它们在开发过程的早期阶段很有用(例如，如果您想要测试您的客户机实现，但是还没有完成服务器的实现，等等)。)。关于如何使用这些程序的说明发布在GitHub上项目的README.md文件中。

In addition to the starter code we provide, you may create your own utility files if you wish.For example, it might be a good idea to create a common.go file and use it to store code that can be shared between both your Client and Server implementations.

除了我们提供的起始代码之外，如果您愿意，您可以创建自己的实用程序文件。例如，创建一个common.go文件并使用它来存储可以在客户机和服务器实现之间共享的代码可能是个好主意。

11

11

15-440 Project 1

15-440项目1

3 Part B: Distributed Bitcoin Miner

3第二部分:分布式比特币矿工

In part B of this project, you will implement a simple distributed system using your LSP implementation you wrote in part A. Your system will harness the power of multiple processors to speed up a compute-intensive task and will be capable of recovering from sudden machine failures.

在本项目的第二部分，您将使用您在第一部分中编写的LSP实现来实现一个简单的分布式系统。您的系统将利用多个处理器的能力来加速计算密集型任务，并且能够从突然的机器故障中恢复。

3.1 Bitcoin Overview

3.1比特币概述

Bitcoin is a decentralized digital currency.At the heart of the Bitcoin protocol is a replay prevention mechanism that prevents participants from double spending Bitcoins.The replay prevention mechanism uses a crypotographic proof-of-work function that is designed to be computationally hard to execute.Clients compete to be the first to find a solution to the proof-of-work in order to get their signature attached to a sequence of transactions.If a client wins, they are rewarded with Bitcoins.The process of finding a solution to the proof-of-work is called mining.

比特币是一种分散的数字货币。比特币协议的核心是一种防止重放的机制，防止参与者双倍消费比特币。重放防止机制使用一个密码学的工作证明函数，该函数被设计为难以计算执行。客户竞相成为第一个找到工作证明解决方案的人，以便将他们的签名附加到一系列交易中。如果客户赢了，他们会得到比特币奖励。找到工作证明解决方案的过程称为挖掘。

In this project, we will spare you the details of the real Bitcoin protocol.You will instead implement a mining infrastructure based upon a simplified variant: given a message M and an unsigned integer N, find the unsigned integer n which, when concatenated with M, generates the smallest hash value, for all 0 ≤ n ≤ N. Throughout this document, we will refer to each of these unsigned integers as a nonce.

在这个项目中，我们将省去你真正比特币协议的细节。相反，您将基于一个简化的变量实现一个挖掘基础结构:给定一个消息M和一个无符号整数N，找到无符号整数N，当它与M连接时，为所有0 ≤ n ≤ N生成最小散列值。在本文中，我们将把这些无符号整数中的每一个都称为随机数。

As an example, consider a request with string message "msg" and maximum nonce 2.We can determine the desired result by calculating the hash value for each possible concate-nation.In this case, nonce 1 generated the least hash value, and thus the final result consists of least hash value 4754799531757243342 and nonce 1.Note that you need not worry about the details of how these hash values are computed—we have provided a Hash function in the starter code for you to use to calculate these hashes instead.

例如，考虑一个带有字符串消息“msg”和最大随机数2的请求。我们可以通过计算每个可能连接的哈希值来确定所需的结果。在这种情况下，随机数1生成最小哈希值，因此最终结果由最小哈希值4754799531757243342和随机数1组成。请注意，您不必担心如何计算这些哈希值的细节——我们在起始代码中提供了一个哈希函数，供您用来计算这些哈希值。

• bitcoin.Hash("msg", 0) = 13781283048668101583

比特币。哈希(“msg”，0) = 13781283048668101583

• bitcoin.Hash("msg", 1) = 4754799531757243342

比特币。散列(“msg”，1) = 4754799531757243342

• bitcoin.Hash("msg", 2) = 5611725180048225792

比特币。哈希(“msg”，2) = 5611725180048225792

One simple approach to completing this task is to perform a brute-force search, in which we enumerate all possible scenarios across multiple machines.For tasks that require searching a large range of nonces, a distributed approach is certainly preferable over executing the compute-intensive task on a single machine.As an example, our measurements show that a typical Andrew Linux machine can compute SHA-256 hashes at a rate of around 10,000

完成这项任务的一个简单方法是执行暴力搜索，在这种搜索中，我们列举了多台机器上所有可能的场景。对于需要搜索大量随机数的任务，分布式方法当然比在单台机器上执行计算密集型任务更可取。例如，我们的测量显示，典型的安德鲁Linux机器可以以大约10，000的速度计算SHA-256散列

12

12

15-440 Project 1

15-440项目1

per second.Running sequentially, a brute force approach would require around 28 hours to try all possible hashes consisting of 9 decimal digits.But, if we could harness the power of 100 machines, then we could reduce this time to around 17 minutes.

每秒钟。按顺序运行，强力方法需要大约28小时来尝试所有可能的由9个十进制数字组成的散列。但是，如果我们可以利用100台机器的能量，那么我们可以把这个时间减少到17分钟左右。

Your task for this project is to implement a simple distributed system to perform this task.We discuss our proposed system design in the next section.

这个项目的任务是实现一个简单的分布式系统来执行这个任务。我们将在下一节讨论我们提出的系统设计。

3.2 System Architecture

3.2系统架构

Your distributed system will consist of the following three components:

您的分布式系统将由以下三个组件组成:

Client: An LSP client that sends a user-specified request to the server, receives and prints the result, and then exits.

客户机:一个LSP客户机，它向服务器发送用户指定的请求，接收并打印结果，然后退出。

Miner: An LSP client that continually accepts requests from the server, exhaustively

Miner:一个LSP客户机，它不断地、详尽地接受来自服务器的请求

computes all hashes over a specified range of nonces, and then sends the server the final result.

计算指定随机数范围内的所有散列，然后向服务器发送最终结果。

Server: An LSP server that manages the entire Bitcoin cracking enterprise.At any time,

服务器:管理整个比特币破解企业的LSP服务器。在任何时候，

the server can have any number of workers available, and can receive any number of requests from any number of clients.For each client request, it splits the request into multiple smaller jobs and distributes these jobs to its available miners.The server waits for each worker to respond before generating and sending the final result back to the client.

服务器可以有任意数量的可用工作人员，并且可以从任意数量的客户端接收任意数量的请求。对于每个客户端请求，它将请求分成多个较小的作业，并将这些作业分配给可用的挖掘者。服务器在生成最终结果并将其发送回客户端之前，会等待每个工作人员做出响应。

|  |  |  |  |
| --- | --- | --- | --- |
| Type | Fields | From-To | Use |
| Join  Request Result | Data Lower Upper Hash Nonce | M-S  C-S, S-M M-S, S-C | miner's join request send job request report job's final result |

|  |  |  |  |
| --- | --- | --- | --- |
| 类型 | 域 | 从-到 | 使用 |
| 加入  请求结果 | 数据低位上限哈希随机数 | M-S  C-S，M-S，南卡罗来纳州 | 矿工加入请求发送作业请求报告作业的最终结果 |

Figure 4: Predefined message types in the Bitcoin distributed system.In the "From-To" column, 'M' denotes a miner, 'S' denotes the server, and 'C' denotes a request client.

图4:比特币分布式系统中的预定义消息类型。在“从到”列中，“M”表示矿工，“S”表示服务器，“C”表示请求客户端。

Each of the three components communicate with one another using a set of predefined messages.Figure 4 shows the types of messages that can be sent between the different system components.Each type of message is declared in the message.go file as a Go struct.As in part A, each message must first be marshalled into a sequence of bytes using Go's json package.

三个组件都使用一组预定义的消息相互通信。图4显示了可以在不同系统组件之间发送的消息类型。每种类型的消息都在消息. go文件中声明为Go结构。正如在第一部分中一样，必须首先使用Go的json包将每个消息封送到一个字节序列中。

13

13

15-440 Project 1

15-440项目1

3.2.1 Sequence of events

3.2.1事件顺序

The overall operation of the system should proceed as follows:

该系统的整体操作应如下进行:

• The server is started using the following command, specifying the port to listen on:

使用以下命令启动服务器，指定要监听的端口:

./server port

。/服务器端口

• One or more miners are started using the following command, specifying the server's address and port number separated by a colon:

使用以下命令启动一个或多个挖掘器，用冒号分隔指定服务器地址和端口号:

./miner host:port

。/miner主机:端口

• When a miner starts, it sends a join request message to the server, letting the server know that it is ready to receive and execute new job requests.New miners may start up and send join requests to the server at any time.

矿工启动时，会向服务器发送一条加入请求消息，让服务器知道已经准备好接收和执行新的作业请求。新矿工可以随时启动并向服务器发送加入请求。

• The user makes a request to the server by executing the following command, speci-fying the server's address and port number, the message to hash, and the maximum nonce to check:

用户通过执行以下命令向服务器发出请求，指定服务器的地址和端口号、要哈希的消息以及要检查的最大随机数:

./client host:port message maxNonce

。/客户端主机:端口消息maxNonce

The client program should generate and send a request message to the server, speci-fying lower nonce "0" and maximum nonce "maxNonce":

客户端程序应该生成并向服务器发送请求消息，指定较低的随机数“0”和最大的随机数“最大随机数”:

[Request message 0 maxNonce]

[请求消息0最大当前值]

• The server breaks this request into more manageable-sized jobs and farms them out to its available miners (it is up to you to choose a suitable maximum job size).Once the miner exhausts all possible nonces, it determines the least hash value and its corresponding nonce and sends back the final result:

服务器将该请求分成更易于管理的作业，并将其分发给可用的挖掘者(由您选择合适的最大作业大小)。一旦挖掘器用尽所有可能的随机数，它就确定最小散列值及其对应的随机数，并发回最终结果:

[Result minHash nonce]

[结果闵哈什随机数]

• The server collects all final results from the workers, determines the least hash value and its corresponding nonce, and sends back the final result to the request client.

服务器从工作人员那里收集所有最终结果，确定最小哈希值及其对应的随机数，并将最终结果发送回请求客户端。

The request client should print the result of each request to standard output as follows (note that you must match this format precisely in order for our automated tests to work):

请求客户端应该按如下方式将每个请求的结果打印到标准输出中(注意，您必须精确匹配这种格式，以便我们的自动化测试能够工作):

14

14

15-440 Project 1

15-440项目1

• If the server responds with a final result, it should print,

如果服务器以最终结果作出响应，它应该打印，

Result minHash nonce

结果minHash随机数

where minHash and nonce are the values of the lowest hash and its corresponding nonce, respectively.

其中minHash和nonce分别是最低散列值及其对应的nonce。

• If the request client loses its connection with the server, it should simply print

如果请求客户端失去与服务器的连接，它应该简单地打印

Disconnected

不连贯的

3.2.2 Handling failures

3.2.2处理故障

We will assume that the server operates all the time, but it is quite possible that a request client or some of the miners can drop out.You should take the following actions when different system components fail:

我们将假设服务器一直在运行，但是请求客户端或一些挖掘者很可能会退出。当不同的系统组件出现故障时，您应该采取以下措施:

• When a miner loses contact with the server it should shut itself down.

当矿工失去与服务器的联系时，应自行关闭。

• When the server loses contact with a miner, it should reassign any job that the worker was handling to a different worker.If there are no available miners left, the server should wait for a new miner to join before reassigning the old miner's job.

当服务器与矿工失去联系时，它应该将工人正在处理的任何工作重新分配给不同的工人。如果没有可用矿工，服务器应该等待新矿工加入，然后再重新分配旧矿工的工作。

• When the server loses contact with a request client, it should cease working on any requests being done on behalf of the client (you need not forcibly terminate a job on a miner—just wait for it to complete and ignore its results).

当服务器与请求客户端失去联系时，它应该停止代表客户端处理任何请求(您不需要强制终止矿工的作业，只需等待作业完成并忽略其结果)。

• When a request client loses contact with the server, it should print Disconnected to standard output and exit.

当请求客户端与服务器失去联系时，它应该打印“断开连接”到标准输出并退出。

Your server will need to implement a scheduler to efficiently assign workers to incoming client requests.You should design a scheduler that balances loads across all requests, so that the number of workers assigned to each outstanding request is roughly equal.Your code should contain documentation on how your scheduler achieves this requirement.

您的服务器将需要实现一个调度程序来有效地将工作人员分配给传入的客户端请求。您应该设计一个调度器来平衡所有请求的负载，以便分配给每个未完成请求的工作人员数量大致相等。您的代码应该包含关于您的调度程序如何实现这一要求的文档。

3.3 Starter code

3.3启动器代码

The starter code for part B of this project can be found in the bitcoin/ directory, and is organized as follows:

本项目第二部分的起始代码可以在比特币/目录中找到，组织如下:

• The message.go file defines the message types you will need to implement your

message.go文件定义了实现您的

system.

系统。

15

15

15-440 Project 1

15-440项目1

• The hash.go file defines a Hash function that your miners should use to compute uint64 hash values.

散列. go文件定义了一个散列函数，矿工应该使用它来计算uint64散列值。

• The client/client.go file is where you will implement your request client program.

客户机/客户机. go文件是您将实现请求客户机程序的地方。

• The miner/miner.go file is where you will implement your miner program.

矿工/矿工. go文件是您实施矿工计划的地方。

• The server/server.go file is where you will implement your server program.

服务器/服务器. go文件是您将实现服务器程序的地方。

We have also provided three binaries (ctest, mtest, and stest) for you to use to test your implementation.These binaries can be found in the project's bin/ directory.For instructions on how to compile and test your client, miner, and server programs, please consult the README.md file on GitHub.

我们还提供了三个二进制文件(ctest、mtest和stest)，供您用来测试您的实现。这些二进制文件可以在项目的bin/目录中找到。有关如何编译和测试您的客户机、矿工和服务器程序的说明，请参考GitHub上的README.md文件。

4 Project Requirements

4项目要求

As you write your code for this project, also keep in mind the following requirements:

在为这个项目编写代码时，还要记住以下要求:

• The project must be done individually.You must not use any code that you have not written yourself.As with all projects in this course, we will be using Moss to detect software plagiarism (including comparisons with student submissions from past semesters).

项目必须单独完成。你不能使用任何你自己没有写的代码。如同本课程中的所有项目一样，我们将使用Moss来检测软件剽窃(包括与过去学期学生提交的材料进行比较)。

• Your code must not use locks and mutexes.All synchronization must be done using goroutines, channels, and Go's channel-based select statement.

您的代码不得使用锁和互斥体。所有同步都必须使用goroutines、channel和Go的基于通道的select语句来完成。

• You must not use Go's net package for this assignment.Instead, you should use the functions and methods in the lspnet package we have provided as part of the starter code for this project instead.

您不得在本次任务中使用Go的网络包。相反，您应该使用lspnet包中的函数和方法，这些函数和方法是我们为这个项目提供的初始代码的一部分。

• Avoid using fixed-size buffers and arrays to store things that can grow arbitrarily in size.For example, do not use a buffered channel to store pending messages for a particular connection.Instead, use a linked list—such as the one provided by the container/list package—or some other data structure that can expand to an arbitrary size.

避免使用固定大小的缓冲区和数组来存储大小可以任意增长的东西。例如，不要使用缓冲通道来存储特定连接的挂起消息。相反，使用链表(如容器/列表包提供的链表)或其他可以扩展到任意大小的数据结构。

• You may assume that the UDP packets will not be corrupted, and that you do not need to check your messages for proper formatting (unless, of course, you want to defend against your own programming errors).

您可以假设UDP数据包不会被破坏，并且不需要检查消息的格式是否正确(当然，除非您想防止自己的编程错误)。

• You must format your code using go fmt and must follow Go's standard naming conventions.See the Formatting and Names sections of Effective Go for details.

您必须使用go fmt格式化代码，并且必须遵循go的标准命名约定。有关详细信息，请参见“有效开始”的“格式”和“名称”部分。

16

16