CMSC 417 Computer Networks		**Fall 2026**
Programming Assignment 4		
Assigned: October 31	*Due: November 15, 11:59:59 PM.*	*Weight: 1.75x*

1 Introduction

In this project, you will implement Chord [1]. To do so, you must read the paper describing the Chord protocol, algorithms, and implementation, which is available at the following URL:

`http://www.cs.berkeley.edu/~istoica/papers/2003/chord-ton.pdf`

Chord is a peer-to-peer lookup protocol which enables the mapping of a key (or identifier) to a peer (or node) in the network. Chord exposes a single function, "lookup", to higher-layer applications:

lookup(<Key>) → <Host>

Chord uses local information and communicates with other peers to find the host (IP address and port) that a given key is mapped to.

Applications may then be built on top of this service that Chord provides. For example, a distributed hash table (DHT) application may distribute the key-value storage of a hash table across many peers and support the typical operations (i.e., insert, get, and remove). The DHT may be implemented on top of Chord by storing the key-value pairs at the node that Chord mapped to the given key.

2 Protocol

You should read the paper to learn about the design of Chord, prior to starting your own implementation. The implementation in the paper, shown in Figure 5 and Figure 6, is presented as pseudocode. The pseudocode involves both local and remote function call invocations, determined by the node at which a function call is invoked. Note that this pseudocode is extended by Section IV.E.3 "Failure and Replication", with changes to the 'stabilize', 'closest_preceding_node', and 'find_successor' function calls. Additionally, there is a new function call 'get_successor_list' hinted at in the description, which returns a node's list of successors and is used in the extended function calls.

For this project, you do not need to understand the details of the following portions of the paper:

- Section IV.E.4 "Voluntary Node Departures"
  (All node departures will be treated as node failures.)
- Section II "Related Work"
- Section V "Simulation Results"
- Section VI "Future Work"

- Section VII "Conclusion"

As discussed in the paper, there are two ways to implement the protocol: iteratively or recursively. Figure 5 in the paper presents the "find_successor" function using a recursive implementation. However, it may be easier to approach it iteratively, since then each node will be able to respond to any incoming RPCs immediately without blocking to wait on responses from other nodes. The iterative version of Chord works similarly to iterative lookups in DNS. If you have further questions on how to implement, you can ask in office hours or on Piazza.

3 Implementation

You choose whether to use TCP or UDP. The `assignment4b` directory inside the `assignment4` repo on Gitlab contains some starter code; the README.md describes the protobuf structures, command line options, and expected output.

4 Additional Requirements

1. Your code must be submitted as a series of commits that are pushed to the origin/main branch of your Git repository. We consider your latest commit prior to the due date/time to represent your submission.
2. You must provide a Makefile that is included along with the code that you commit. We will run 'make' in the root directory, which must produce a 'chord' executable also located at the top level.
3. You must submit code that compiles in the provided Docker, otherwise it will not be graded.
4. Your code must be -Wall clean on gcc/g++ in the provided Docker, otherwise you may be penalized. Do not ask the TAs for help on (or post to the forum) code that is not -Wall clean, unless getting rid of the warning is the actual problem.
5. You are not allowed to work with the other teams or to copy code from any source.

References

[1] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan. Chord: A Scalable Peer-to-peer Lookup Protocol for Internet Applications. *Networking, IEEE/ACM Transactions on*, 2003.