

# CS7270 Homework 1, Problem 4

*Due date: 11:59pm Thursday, Feb 28*

## 1 Overview

In this part of the assignment you will write the code for a limited-functionality Chord node. We will provide the executable for a network of Chord nodes, and your node will interoperate with the provided network. You will also write a small program to control the operation of your Chord node.

The Chord node and the control program will communicate via a protocol we have devised. The objectives are for you to further familiarize yourself with the basic Chord algorithm while honing your low-level network programming skills.

## 2 Node and network operation

Extract the tarfile and cd into the “cs7270chord” directory. Note that we are only providing Solaris sparc binaries, so you are limited to this platform for running the “canned” network. You will start four chord nodes with the following commands:

```
./7270chord -c node1 -d doc1
./7270chord -c node2051 -d doc2051
./7270chord -c node3075 -d doc3075
./7270chord -c node3588 -d doc3588
```

Your node will be the 5th node, started with:

```
./chord -c node2050 -d doc2050
```

The argument of “-c” is the name of the config file that is used to set up the ID and finger table for the node and the argument of “-d” is the name of the file containing the node’s initial documents. The formats of these files are described later.

Node ID’s are in a 12-bit space, so they can run from 0 to 4095. Keys are mapped into this space with the modulus operator.

The network will initialize itself with your node already added to the finger tables of the other nodes. When you start your node it will read a configuration file we provide. Using this

configuration file you will create a local finger table with the information for the other nodes. Your node will also read a file that consists of the documents your node is responsible for.

At this point your node is “joined” with the Chord network. It will loop, reading packets and reacting to packets. To simplify the design of your node, the protocol does not require that the node remember the reason why it sent certain messages. When it receives responses to the GetSuccessor query in the form of Successor messages, all the information necessary to continue will be contained in the message itself.

You will use your control program to direct the Chord node to take specific actions described below. The control program can (and should!) be run from machines with different architectures (sparc, ix86). I’ll be doing this with my control program to check your byte ordering when I test your node.

### 3 File Formats

Your executable should be named *chord*, and invoked as follows:

```
./chord -c <configfile> -d <docfile>
```

The configuration file format is shown in Table 1. Items in the same row in different cells are separated by a single space.

nodeId		nodePort
1st finger nodeId	hostname	nodePort
2nd finger nodeId	hostname	nodePort
.	.	.
.	.	.
.	.	.
Last finger nodeId	hostname	nodePort

Table 1: Configuration file format.

The document file format is shown in Table 2. Again, items in the same row in different cells are separated by a single space.

### 4 Algorithms

The two core algorithms are **find\_successor** and **closest\_preceding\_node**. Note that there is conflicting information in the paper about which node is responsible for a key. The interpretation chosen for this project is that *successor(k)* is responsible for key *k*, and that the *successor* of a node is the node indicated in the first finger. So key 10 is not stored at node 10, but at *successor(10)*. This differs from the pictures in the paper!

#### 4.1 Find\_successor

The pseudocode for `find_successor` is:

key1
document1 (may consist of multiple lines, 1024 characters max)
END_DOC
key2
document2 (may consist of multiple lines, 1024 characters max)
END_DOC
.
.
.
Final key
Final document (may consist of multiple lines, 1024 characters max)
END_DOC

Table 2: Document file format.

```

n.find_successor(id)
  if (key ∈ [n, n.successor))
    return n.successor;
  else
    n' = closest_preceding_node(id);
    return n'.find_successor(id);

```

## 4.2 Closest\_preceding\_node

The pseudocode for `closest_preceding_node` is:

```

n.closest_preceding_node(id)
  for i = m downto 1
    if (finger[i] ∈ (n, id))
      return finger[i];
  return n;

```

## 5 Messages, message formats and protocol

When your node is executed and has configured itself, it should listen for TCP connections on the port specified in its configfile. Each message in the protocol is transmitted by connecting to the well-known socket for the node, writing the message and closing the TCP connection. There are two types of messages: node messages and control messages. Node messages are sent among nodes while executing the Chord algorithm. Control messages are sent from outside the network to cause a node to take some action.

## 5.1 Node messages

- **GetSuccessor:** Execute the `find_successor` algorithm. If you *know* the successor of the indicated key, send a Successor message to the initiator of this message, otherwise execute the `closest_preceding_node` algorithm and forward the message.
- **Successor:** This message indicates that you previously sent a GetSuccessor message. Check the reason code and perform the indicated operation. Operations that you may perform are InsertDoc, GetDoc and DeleteDoc (values are given later).
- **InsertDoc:** You are getting this message because the system has determined that a document belongs in your node. Read and store the document in your node with the indicated key.
- **GetDoc:** You are getting this message because the system has determined that if this document exists, it will be stored in your node. Send the document with the indicated key to the initiator of this message via a Doc message.
- **DeleteDoc:** You are getting this message because the system has determined that if this document exists, it will be stored in your node. If the document exists in your node, silently delete it. If the document does not exist in your node, silently ignore this message.
- **Doc:** You are getting this message because you previously sent a GetDoc message. Read the document, print information about which node sent the document and print the document. The preferred output is described later.

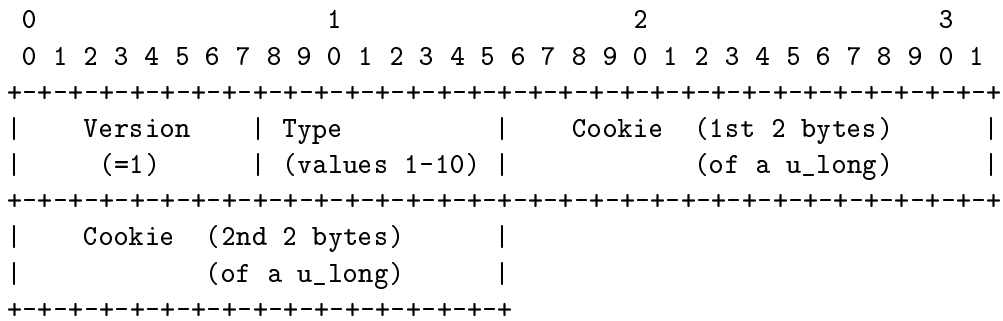
## 5.2 Control messages

- **AddDoc:** This is a message sent from the control program. Determine where the key/document must be stored and send the messages necessary to store it.
- **PrintDoc:** This is a message sent from the control program. Determine where the key/document is stored and send the messages necessary to cause the responsible node to send you a Doc message.
- **RemoveDoc:** This is a message sent from the control program. Determine where the key/document would be stored and send the messages necessary to delete it.
- **Exit:** This is a message sent from the control program. Clean up and exit.

The message formats are given in the following diagrams. (A description of this style of diagram can be found in RFC1035 if necessary.)

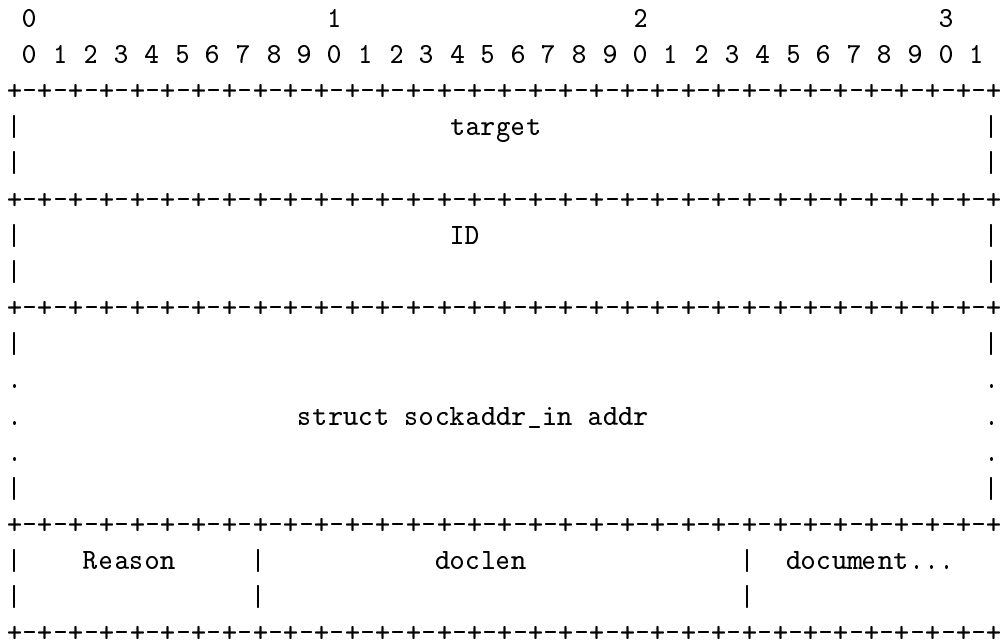
## 5.3 Header

All messages will begin with a 6-byte header. The *version* is 1, The *type* will indicate the format of part of the message following the header. Type will take on values from 1 to 10, indicating GetSuccessor, Successor, InsertDoc, GetDoc, DeleteDoc, Doc, AddDoc, PrintDoc, RemoveDoc and Exit, respectively. The *cookie* is 64206<sub>10</sub>. You can use the cookie to check that you're doing byte ordering correctly.



### 5.4 Type 1, GetSuccessor

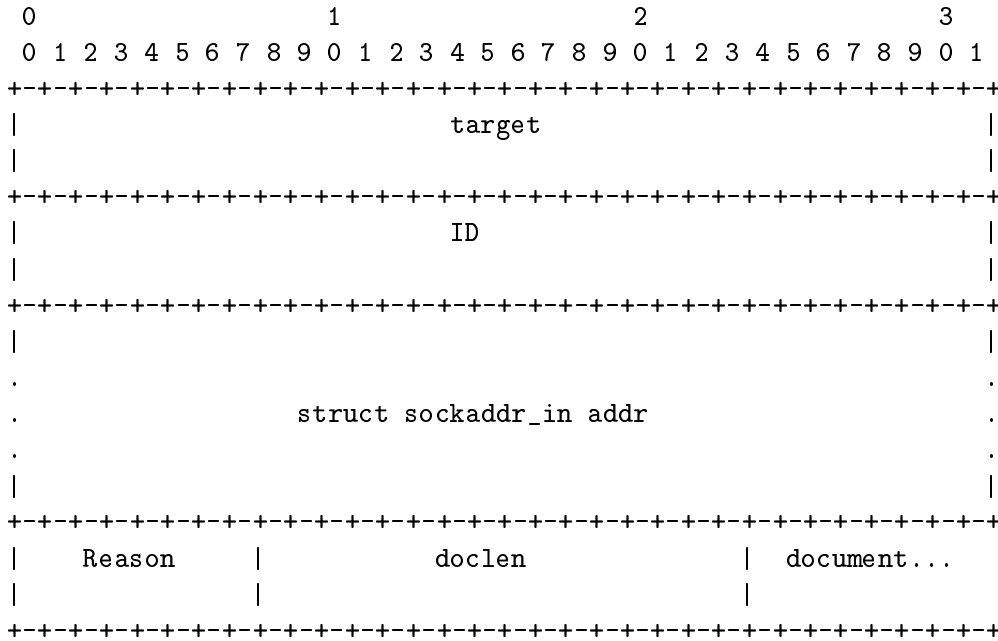
*Target* indicates the ID or key whose successor is being sought. *ID* and *addr* indicate the node to whom the eventual answer will be sent. *Reason* is either 1, 2 or 3, indicating InsertDoc, DeleteDoc and GetDoc, respectively, as the reasons why the successor is being sought. Following this message are *doclen* bytes which comprise the document.



### 5.5 Type 2, Successor

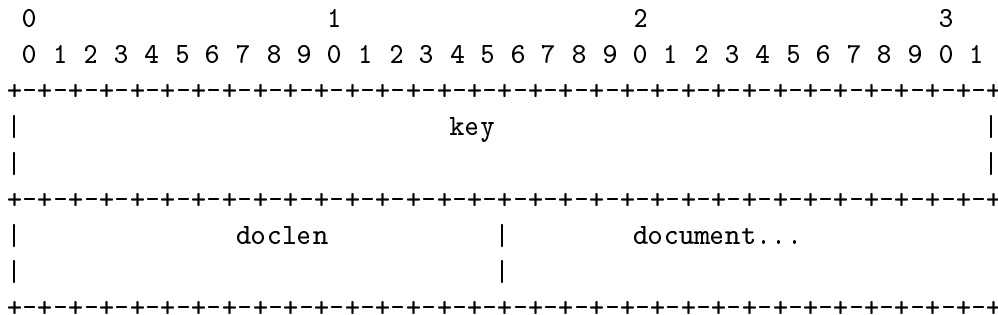
*Target* indicates the key or ID whose successor was being sought. *ID* and *addr* indicate the successor of *target*. *Reason* is either 1, 2 or 3, indicating InsertDoc, DeleteDoc and GetDoc, respectively, as the reasons why the successor was being sought. Following this message are *doclen*

bytes, which comprise the document. These values are passed unchanged from the GetSuccessor message.



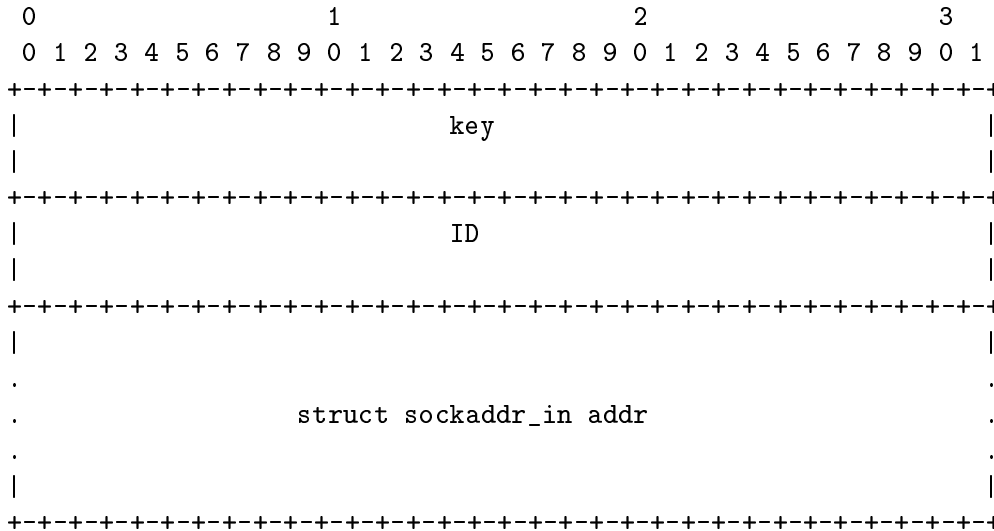
### 5.6 Type 3, InsertDoc

*Key* indicates the key of the document being inserted. Following this message are *doclen* bytes, which comprise the document.



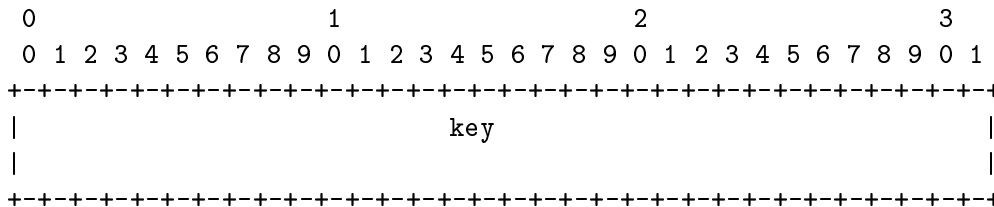
### 5.7 Type 4, GetDoc

*Key* indicates the key of the document to send. *ID* and *addr* indicate the initiator of the request, to whom the document must be sent.



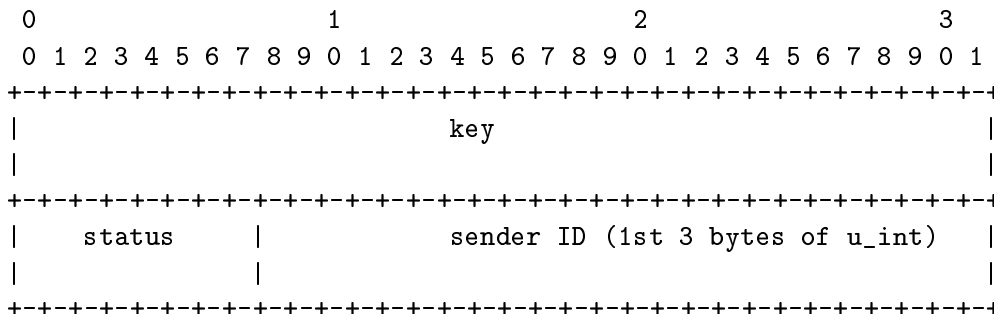
### 5.8 Type 5, DeleteDoc

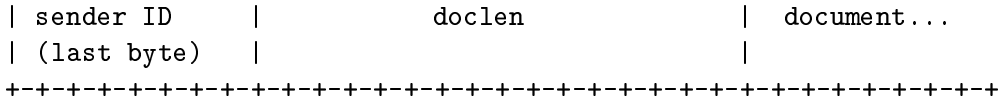
*Key* indicates the key of the document to delete.



### 5.9 Type 6, Doc

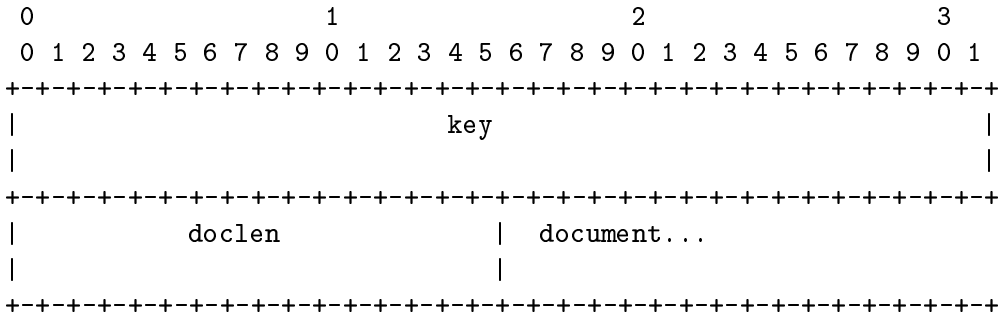
*Key* indicates the key of the document in the message. *addr* indicate the initiator of the request, to whom the document must be sent. *Status* is either 1, indicating the document was found, or 2, indicating the document was not found. If the document was found, *doclen* bytes, which comprise the document, follow the message.





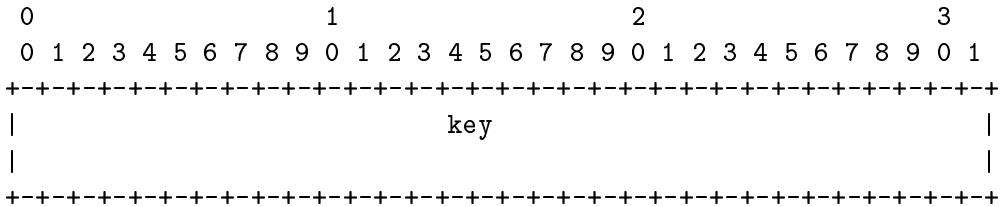
### 5.10 Type 7, AddDoc

*Key* indicates the key of the document to add (wherever it belongs). *Doclen* bytes, which comprise the document, follow the message.



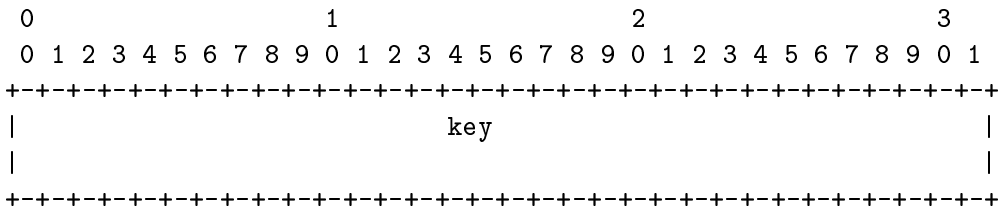
### 5.11 Type 8, PrintDoc

*Key* indicates the key of the document to locate and print.



### 5.12 Type 9, RemoveDoc

*Key* indicates the key of the document to locate in the system and delete.



### 5.13 Type 10, Exit

No need for anything else here - when you read the header and determine the type, just cleanup (close files, free memory) and exit.

## 6 Output

Your node must print out enough information for me to tell that it is working properly. When you read a message, print “recv: <messagetype>”, where <messagetype> is one of GetSuccessor, Successor, etc. When you send a message, print “send: <messagetype> to node <nodeId>”, where <messagetype> is as above, and <nodeId> is the node to which you are sending the message. When you receive a document, print the key and the node at which it was found (or where it was not found if that is the case), and the text of the document, clearly delineated.

Here’s an example of what your node should print out (or something really, really, really close to it) when I send messages from my control program to your node:

```
recv: PrintDoc
send: GetSuccessor to node 3075
recv: Successor
>>>>
>>>> Key 302026777 found at node 2050
>>>>
==>StartDoc
Text of document w/key 302026777.
==>EndDoc
recv: PrintDoc
send: GetSuccessor to node 3075
recv: Successor
>>>>
>>>> Key 9642089 found at node 2050
>>>>
==>StartDoc
Text of document w/key 9642089.
==>EndDoc
recv: RemoveDoc
send: GetSuccessor to node 3075
recv: Successor
recv: PrintDoc
send: GetSuccessor to node 3075
recv: Successor
<<<<
<<<< Key 302026777 not found at node 2050
<<<<
```

This was the output from my node 2050 to which I first sent a PrintDoc message for key

302026777, a PrintDoc message for key 9642089, a RemoveDoc message for key 302026777, then another PrintDoc message for key 302026777. Since node 2050 discovered it was the one responsible for the key, no GetDoc message was sent and no Doc message was received. Generally this will not be the case. Note that this is all done in the node - not in the control program which is just used to send the messages to your Chord node.

## 7 Other

Some final notes:

- It shouldn't take you 3 weeks to implement. You have the benefit of knowing exactly the correct algorithm, packet formats and system requirements. Besides, you get only 2 weeks.
- You should loop reading from socket until you get everything. However, if you send less than you claim you're going to, you'll cause my node to hang. Sorry. I won't require that your node handles this either, but it would be fine if you want to add a timeout mechanism.
- The default ports specified in the config files may already be in use on the machine you're working on (probably because someone else is working on this project on the same machine). If so, you can edit the config files to use other ports.
- Be careful not to change the key as you are passing messages around.
- Don't forget byte ordering!
- Be conservative in what you send and liberal in what you receive.
- Does your node work properly when you're the only node?
- Hint: don't give your TA a headache while he's grading your program.
  - Make sure it compiles properly.
  - Be precise with your output format. Don't print anything that's not required. I don't plan to spend a lot of time to figure out where the juicy information is within a bunch of output statements.
- Use a default of 12 bits for keys/IDs. Feel free to add other flags to the command line like "-b <numbits>" to override any defaults as you wish, but make sure the command line given above works properly.
- Regular logging was removed from the node we're providing. If you see them printing any messages, you should probably pay attention to them as they will be indicating either a problem with my code (nah...) or with yours.
- My office hours will be in the CoC Commons Wednesdays 9am-11am during the two weeks this assignment is out.

- There are no bugs in my code. However, if you find any :- ) send me email (liston@cc.gatech.edu) precisely describing the problem so that I will be able to reproduce it. Include any config and doc files that are needed other than the ones we supplied. I'll post a note on the newsgroup when I confirm the problem, then again when I've got a fix and where to find the fix. Make sure to check the newsgroup BEFORE sending me a bug report so you don't tell me something I already know. I'll give up to 1 point for each bug identified, depending on severity of the bug (other than anything already mentioned in this description), but I will be somewhat testy with bug reports that are \*clearly\* not a bug with my code, so try to make absolutely certain the problem is with my code before complaining! Those who work on this early will have a better chance at getting points for bug reports.
- You can post interesting (but correct!) topologies in the form of config files, and docfiles on the newsgroup. I'll give 1 point for each correct topology posted, up to a maximum of 5 points for any individual.
- Watch the newsgroup for instructions on how to turn this in. Source files that compile cleanly under Solaris and Linux with a correct makefile will probably be part of this picture.
- Start now.