



Spring 2009

Dr. Kevin Hamlen



CS 4485: CS PROJECT

Stage 2

Stage 2 (due October 16th)

- Two main new features:
 - join an existing network
 - file upload/download
- Demo
 - network starts with one node
 - join several new peers, one at a time
 - exchange a few files
 - automated tests of the above

Join operation

- Initial contact
 - each client starts with a list of one or more peers likely to be in the network
 - list is NOT necessarily complete
 - some peers in the list might NOT actually be listening
 - can reuse initialization text file format from Stage 1
- Two different start-up modes:
 - Mode 1: One peer gets launched in a special mode that ignores the initialization file and starts its own one-peer network
 - all finger table entries are me and I'm my own successor
 - Mode 2: Contact each peer in the list with a "join request" message until one says "okay".
 - order of attempts is up to you

Join protocol

- New->Inductor: Join request [IP_{new} , $Port_{new}$]
- Inductor may respond “no” (e.g., if it is still trying to complete its own join procedure)
 - New then tries another peer in its list
- Otherwise:
 - Inductor->New: “Okay”
 - Inductor computes ID_{new} and forwards the join request to the peer nearest ID_{new} rounded down
 - Call this peer “Pred” (because it’s the new peer’s predecessor)
 - Pred->New: Pred’s finger table (and key database, to be discussed later)
 - Pred replaces its first one or more finger table entries with New

Update protocol

- Pred's finger table is pretty close to what New's should be, but not exactly. We need a finger table update procedure.
- New sends out N messages using Pred's finger table, one for each $0 \leq n < N$
 - New contacts the peer whose ID is closest to $(ID_{\text{new}} + 2^n) \bmod 2^N$, rounded down
 - Message asks: Who is your successor?
 - New replaces its n th finger table entry with the peer given in response to the message

Pointsto Correction

- Problem: Nobody other than Pred knows that New exists. Some of them might need to update their finger tables to include New.
- Three possible solutions:
 - Just tell everyone to run Update every few minutes (probably unnecessarily expensive)
 - Have New find each and announce itself (also expensive and potentially hard to implement)
 - Augment the forwarding protocol to discover and correct these small errors on the fly (best solution)

New Forwarding Step

- Overlay message content
 - All overlay messages already say “please send to peer whose identifier is nearest x , rounded down”
 - Add to each message two more fields: the current forwarder’s IP and Port
- Detecting forwarding inefficiencies
 - Upon receiving a message to be forwarded, first forward it as normal.
 - Next, compute the last forwarder’s ID from his IP and Port (now stored in the message).
 - If I’m not the best peer for this job, send last-forwarder a message suggesting a better peer.

Forwarding Pseudocode

$ID_{last} = \text{hash}(IP_{last}:IP_{port})$

$dist = ID_{me} - ID_{last}$

if ($dist < 0$) $dist = dist + 2^N$

$hop = 2^{\text{floor}(\log_2 dist)}$;

$target = (ID_{last} + hop) \bmod 2^N$;

if (target not "between" my predecessor and me)

 Send " $IP_{pred}:Port_{pred}$ exists" message to $IP_{last}:Port_{last}$

New Predecessor Field

- Each client will track who its predecessor is
 - Predecessor info is NOT part of the finger table
 - Do NOT forward any messages to it
 - It will only be used in our points-to correction protocol (coming up)
- One-node network: I'm my own predecessor
- When I join
 - I find out from Pred who my predecessor is
 - I tell my successor, "I'm your new predecessor"

File Sharing

- Four major operations:
 - Publish: I have file “filename”.
 - Retract: I no longer have file “filename”.
 - Lookup: Who has file “filename”?
 - Download: Please send me file “filename”.
- File Identifiers
 - Each file gets a unique identifier (just like peers)
 - For now: hash the file’s name (assume names are unique)
- Peer whose ID is nearest a file’s ID, rounded down, is called the file’s “key holder”

Publish/Retract

- Sender
 - Hash the filename to get ID_{file}
 - Send "I have file ID_{file} " message to peer whose ID is nearest ID_{file} , rounded down (key holder)
- Receiver
 - Each peer keeps a database of file ID's and the IP:Port's of peers claiming to have those files
 - Upon receiving publish message, add info to database
- Retract is exactly the same, but remove info from database instead of adding

Lookup

- Sender
 - Hash filename to get ID_{file}
 - Contact peer whose ID is nearest ID_{file} , rounded down (key holder)
 - Ask “Who has file ID_{file} ?”
- Receiver
 - Responds with list of IP:Port’s in its database for that file id
 - Empty list if no such file in database

Download

- Sender
 - Obtained list of IP:Port's from key holder
 - Just contact one directly and request file by filename
- Receiver
 - Has a set of files to share in a reserved folder
 - Streams the named file in that directory to the requestor (or sends a failure message if the file doesn't exist or isn't sharable)

Join+Share Interaction

- Addition to join protocol
 - Recall: When a new peer joins, its predecessor sends it its initial finger table.
 - Also send it the key holder database for all files whose ID's fall "between" the new node and it's successor
 - (Recall that the new node's successor is the predecessor node's old successor.)

Stage 2 Summary

- Dynamic Network Support
 - Initialization (two different modes)
 - start my own one-node network
 - ask an existing peer to join me to the network
 - Respond to join requests
 - send my finger table
 - split my key database
 - Full finger table updates (send N messages)
 - Respond to “Peer Exists” messages by updating my finger table
 - Points-to correction (respond to forwards with “peer exists” messages)
- File Sharing
 - File lookup by ID
 - Respond to file lookup requests with peer lists
 - Request file downloads by filename
 - Respond to file download requests by streaming the file