



Fall 2009

Dr. Kevin Hamlen



# CS 4485: CS PROJECT

# Stage 3



# New Features

- Keyword lookup
  - instead of searching for files by name, search by keyword
  - keywords specified by file publisher (trusted)
  - (To make it untrusted, we'd need a voting system.)
- Leave
  - how to do a graceful exit
  - dealing with non-graceful exits
- Stage 3 completes the core functionality of the project!
  - Stage 4 is devoted to fixing bugs, making it look nice, documenting it, and cool security features

# Publishing by Keyword


- Publish:
  - hash file CONTENT instead of filename to determine file's key-holder
  - Tell key-holder: file hash, my IP, my Port
- Retract: same as before (but with new keys)
- Associate:
  - GUI allows user to associate zero or more keywords with any shared file
  - Hash each keyword to obtain keyword-id
  - Tell keyword-holder: keyword, file hash, filename
- Unassociate: same as retract but use keyword-id

# Downloading by Keyword

- Search:
  - User enters keyword to search for
  - Hash keyword to obtain keyword-id
  - Ask keyword-holder: Which files match this keyword?
  - Keyword-holder returns list of filenames & file-hashes
- Lookup:
  - Use file-hash to contact desired file's key-holder
  - Ask key-holder: Who has file <file hash>?
  - Key-holder returns list of IP:Port's of publishers
- Download: same as before



# Note about Key-transfer

- During join protocol, when Pred gives some keys to New, that now includes both file hash keys and keyword hash keys!
- 

# Predecessor Pointers

- In order to deal with non-graceful exits, we will make use of predecessor info.
- Recall that during join...
  - Pred->New: Pred's finger table,  $ID_{Pred}:\text{Port}_{Pred}$
  - New->Succ:  $ID_{New}:\text{Port}_{New}$
  - New and Succ update their predecessor pointers
- Predecessor pointer NOT used for any forwarding.

# Graceful Leave Protocol

- Me->Pred: I'm leaving. Succ is your new successor. Here are all my keys (file-keys and keyword-keys).
- Me->Succ: I'm leaving. Pred is your new predecessor.
- Retract any published files (but not keywords)
- Do not wait for responses and ignore connection failures.
- Pred and Succ update successor/predecessor pointers, and Pred incorporates new keys.

# Problem

- Some nodes still have me in their finger tables.
- They will try to forward to me and fail.
- We could hunt them down and correct this, but since we need to deal with non-graceful exits anyway, we might as well just have a **forwarding failure recovery algorithm**.

# Forwarding Failures

- I try to forward to  $P$  and get connection denied
  - This does not deal with message-dropping
  - This does not deal with suboptimal routing paths
- If  $P$  is not my successor:
  - Find the first non- $P$  entry in my finger table before  $P$  appears. Call it  $X$ .
  - Replace all  $P$ 's in my finger table with  $X$ .
  - Forward the message via  $X$ .
  - Trigger a finger table update for the changed entries.

# Forwarding Failures (cont)

- If P is my successor:
  - Drop the pending message (sad, but it's for the best)
  - Find the finger table entry that immediately follows all the P's in my table. Call it Y.
  - Ask Y who his predecessor is. Call it Z.
    - If Y can't be contacted either, give up. Do a leave and then rejoin the network as a new node.
  - While Z is not me and is "between" me and Y...
    - Ask Z who his predecessor is.
    - If Z cannot be contacted, stop loop.
    - Otherwise, set  $Y:=Z$  and  $Z:=Z$ 's predecessor and repeat.
  - At end of loop: Choose Y as my new successor. Contact Y and tell him I'm his new predecessor.



# Forwarding Failures: Thread Safety

- The find-my-successor loop should run in a separate thread so that other activity can continue
- Only one find-my-successor loop should run at a time! Be sure to synchronize it!
  - Other forward-to-successor failures that happen during the loop will just drop the message without spawning another (redundant) loop.