eecs489 Computer Networks

# Lecture 11: PA2 Walk-through

## Lab 3: `imgdb`

Image database server

Communicate with `netimg` client over an image socket

Default `images` folder under working directory

All instances of `imgdb` share the same `images` folder

Each instance serves up only images whose names are within the instance's ID range, `(beginID, endID]`

```
% imgdb [ -b <beginID> -e <endID> ]
```

## ID

Computed from SHA1 of image name:

```
unsigned char ID = 0;
unsigned char md[SHA1_MDLEN];  // message digest

SHA1((unsigned char *) fname, strlen(fname), md);

for (i = 0; i < SHA1_MDLEN; i++) {
  ID ^= md[i];    // XOR all the unsigned chars,
                  // assuming 8 bit ID
}
```

Folding up the 160-bit SHA1 value increases the probability of the IDs colliding

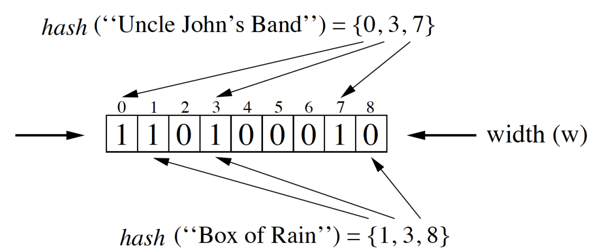
On Windows you need to install and link with the `openssl` library (see *Building Socket Programs* course note)

## Bloom Filter

When an image is loaded, it's also entered into a 64-bit Bloom Filter (`bf`)

Three hash functions:

- each computes an index in [0,63] from a random offset of the image name's SHA1 value
- `bf` bit at the computed index is set to 1



# Lab3

Task 1: become familiar with modulo arithmetic, compute `ID_inrange(ID, begin, end)` and populate the Bloom Filter (`bf`) on image addition (2 lines)

Task 2: become familiar with SHA1 computation, ID generation, and Bloom Filter operation (8 lines)

Be sure you really understand what you're doing, not just filling in the blanks

# Assumptions

ID is 8 bits

Image database can hold only `IMGDB_MAXDBSIZE` number of images

Once loaded or cached, images are never removed

Only one image is read into memory at a time

# Lab 4: `dhtn`

The first instance assumes the whole ID ring

Subsequent instances join the DHT by contacting the provided node:

```
% dhtn [ -p <node>:<port> -I <ID> ]
```

Each node's ID is computed from its address and port number and is on the same space as the image IDs

Node ID can be statically assigned using the `–I` option

- useful for testing ID collision
- and for testing node addition order and scenarios

# `dhtn`

As with PA1, the DHT socket used for inter-node communication is different from the image socket used for client communication

Use the node IDs to differentiate nodes

# DHTM Packet Formats

| | | |
|---|---|---|
| vers | type | ttl |
| rsvd | ID | port |
| addr | | |
| rsvd | ID | port |
| addr | | |

dhtmsg_t

dhtwlcm_t

Defined in `dhtn.h`

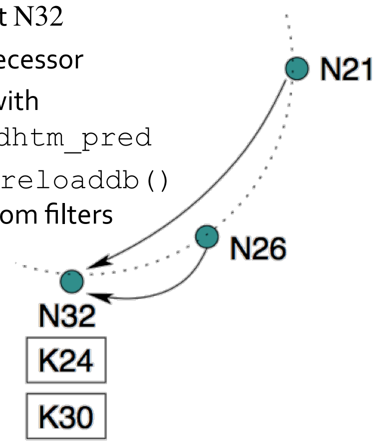
`dhtm_type`: `DHTM_JOIN` ⇒ `dhtm_node`: joining node
`dhtm_type`: `DHTM_REID` ⇒ `dhtm_node`: not used
`dhtm_type`: `DHTM_WLCM` ⇒ `dhtm_node`: successor node
`dhtm_pred`: predecessor node
`dhtm_type`: `DHTM_RDRT` ⇒ `dhtm_node`: new successor

# Join Handling

`DHTM_JOIN`:
- `handlepkt()` usually closes DHT socket immediately upon receiving a packet,
- but if packet is a join packet, it passes the DHT socket to `handlejoin()`
- `handlejoin()` must close DHT socket as soon as possible, to avoid deadlock

# Join Handling: Case 2

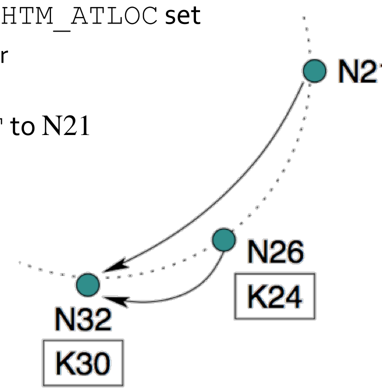
A correct spot has been found on the identifier ring for the joining node
- for example: N26's join request at N32
- N32 accepts N26 as its new predecessor
- N32 sends `DHTM_WLCM` to N26 with N32 in `dhtm_node`, and N21 in `dhtm_pred`
- N32 and N26 both call `imgdb::reloaddb()` to reload their databases and Bloom filters




[Stoica+'03]

# Join Handling: Case 3

When the sender's successor has become inconsistent:
- for example, after N26 joins the network, let N24 joins at N21
- N21 still thinks that N32 is its successor, so it forwards N24's join request to N32 with `DHTM_ATLOC` set
  - `DHTM_ATLOC`: you're my successor and this ID should be in your range
- N32 sends back a `DHTM_REDRT` to N21 with N26 in `dhtm_node`
- N21 corrects its successor info (`finger[0]`) and forwards N24's join request to N26




[Stoica+'03]

## Join Outcome at the Joining Node

`DHTM_REID`: ID collision (Case 1), `reID()` and `join()` again

`DTHM_WLCM`: store successor in `fingers[DHTN_SUCC]` and predecessor in `fingers[DHTN_PRED]` (`DHTN_SUCC` == `0` & `DHTN_PRED` == `DHTN_FINGERS`)

## Lab 4

All `dhtn`'s may share the same `images` folder, but each may serve up only images within its purview

We don't implement image search in Lab4

Entering '`p`' prints out successor and predecessor info

- newly joined node must have both correct
- all nodes must have predecessor info correct at all times (can be used to reconstruct the ring)
- successor info may become inconsistent after node additions
- ('`p`' doesn't work on Windows)

## More Assumptions

No node departure

Node join does not fail

No concurrent joins

Single message per connection, except for node redirect

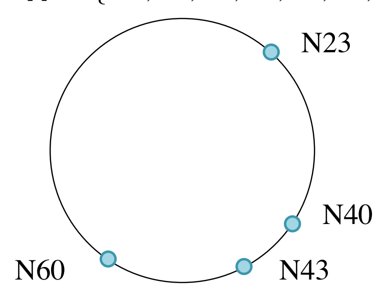
## PA2: Search with Finger Table

Initialize all fingers to point to `self`

May be useful to keep a lookup table `fIDs[]` at each node to keep the $\mathrm{ID}+2^i$ values, $0 \leq i < n$, $n = 8$ in PA2

Example: let current node ID be 23

- `fIDs[]` $= \{\, 24, 25, 27, 31, 39, 55, 87, 151 \,\}$




N23's finger table:

| | | |
|---|---|---|
| 0 | 24 (successor) | 40 |
| 1 | 25 | 40 |
| 2 | 27 | 40 |
| 3 | 31 | 40 |
| 4 | 39 | 40 |
| 5 | 55 | 60 |
| 6 | 87 | 23 |
| 7 | 151 | 23 |
| 8 | predecessor | 60 |

# Join/Search Example

Let `targetID` (joining node or image ID) 42 arrives at node 23

Which node shall it be forwarded to?

Find the largest index `j` for which `fIDs[j]` is in the range `(nodeID, targetID]`

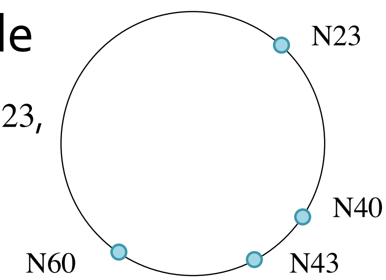
| index | fIDs[] | fingers[] |
|---|---|---|
| 0 | 24 (successor) | 40 |
| 1 | 25 | 40 |
| 2 | 27 | 40 |
| 3 | 31 | 40 |
| 4 | 39 | 40 |
| 5 | 55 | 60 |
| 6 | 87 | 23 |
| 7 | 151 | 23 |
| 8 | predecessor | 60 |

In this example, `nodeID` $= 23$, `targetID` $= 42$;
`j` $= 4 \Rightarrow$ `fIDs[j]` $= 39 \in (23, 42]$
forward to `fingers[j]` $= 40$

N40 further forwards to N43, where ID 42 "belongs"
Forwarding to N60 would have overshot

# Join/Search Example

Another example, `nodeID` $= 23$, `targetID` $= 44$; `j` $= 4$
$\Rightarrow$ `fIDs[j]` $= 39 \in (23, 44]$
forward to `fingers[j]` $= 40$




Summary: `fingers[j]` contains the node that immediately precedes `targetID` in the finger table (though not necessarily immediate precedent of `targetID` on the ring, e.g., `targetID` $= 44$ is forwarded to N40 not N43)

# Join/Search Example

If `targetID` is expected to be in forwarded node's range, set `DHTM_ATLOC`

For example: `nodeID` $= 23$, `targetID` $= 56$; `j` $= 5$
$\Rightarrow$ `fIDs[j]` $= 55 \in (23, 56]$
forward to `fingers[j]` $= 60$ with `DHTM_ATLOC` set

| | | |
|---|---|---|
| 0 | 24 (successor) | 40 |
| 1 | 25 | 40 |
| 2 | 27 | 40 |
| 3 | 31 | 40 |
| 4 | 39 | 40 |
| 5 | 55 | 60 |
| 6 | 87 | 23 |
| 7 | 151 | 23 |
| 8 | predecessor | 60 |

If N58 has joined, N60 returns `DHTM_RDRT`

If `DHTM_RDRT` received, correct `fingers[j]` (not just correcting successor as in Lab4)

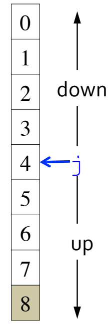
# Updating the Finger Table

If `DHTM_RDRT` received, correct `fingers[j]`

Upon `DHTM_WLCM`, set `fingers[DHTN_SUCC]` and `fingers[DHTN_PRED]`

Every time a finger table entry (at index `j`) is modified, call `fixup(j)` and/or `fixdn(j)`

# `fixup()` and `fixdn()`




`fixup(j)`:

for each $\texttt{k}, \texttt{j} < \texttt{k} < \texttt{DHTN\_FINGERS}$

if $\texttt{fIDs[k]} \in (\texttt{nodeID}, \texttt{fingers[j]}\text{'s ID}]$,

update `fingers[k]` with `fingers[j]`

otherwise stop the walk

`fixdn(j)`:

inclusive!

for each $\texttt{k}, \texttt{j} > \texttt{k} \geq 0$

if $\texttt{fingers[j]}\text{'s ID} \in [\texttt{fIDs[k]}, \texttt{finger[k]}\text{'s ID}]$,

update `fingers[k]` with `fingers[j]`, stop the walk if `fIDs[k] == fingers[k]`

# DHT Search

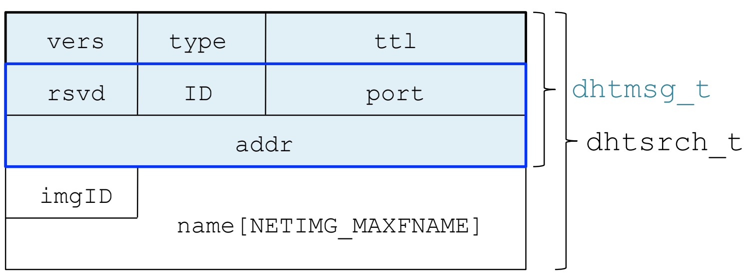
When a client or another node queries for an image, first check local database and cache (Bloom filter) for image

If found, send image to client or send `DHTM_RPLY` to search originator node

If not found and image is in node's ID range, replies with `DHTM_MISS`

If not found and image is not in node's ID range, sends out a `DHTM_SRCH` packet

# DHTM Search Packet Format




Defined in `dhtn.h`

`dhtm_type`: `DHTM_SRCH` ⇒ `dhtm_node`: originator node

`dhtm_type`: `DHTM_RPLY` ⇒ `dhtm_node`: not used

`dhtm_type`: `DHTM_MISS` ⇒ `dhtm_node`: not used

# Search Forwarding

`DHTM_SRCH` packets are forwarded like `DHTM_JOIN`s

- including use of `DHTM_ATLOC` and `DHTM_RDRT` to fix the finger table

When you send back a `DHTM_RPLY` or `DHTM_MISS` packet, you don't forward the search packet further and consequently do not need to fix any existing finger table inconsistencies

Unlike PA1, `DHTM_RPLY` doesn't transfer an image, it's only a "permission" to load the search originator's database and cache (Bloom filter) with the queried image name

# Even More Assumptions

Once loaded or cached, images are never removed, but when the ID range of a node changes, its whole image database is reloaded, its cache flushed, and its Bloom filter reinitialized

Only one outstanding search request per `dhtdb`