3.3 FTP

- File Transfer Protocol (FTP) is first published in 1971 (RFC 114) that were implemented on hosts at MIT.
- In 1985, a standard FTP is published as RFC 959. Several extensions are published since then including anonymous FTP, secure FTP, FTP for IPv6 or NAT etc.
- FTP is used for a local host transfers files from or to a remote host which runs an FTP server and the user has an account on it.
- The user initializes the communication and provides a user identification and a password. After proving this authorization information, the user can transfer files between the local host and the remote host.

FTP uses two parallel TCP connections to transfer file, a control connection and a data connection. The control connection is used for sending control information between the local and remote hosts (such as user identification, password, commands “put”, “get” etc.). The data connection is used to transfer files.

(HTTP basically uses one TCP connection)

- When a user starts an FTP session, the client side of FTP initiates a control TCP connection with the server side on server port 21.
- The client side sends user identity and password and other commands over this control connection.
- When the server side receives a command for a file transfer over the control connection, the server side initiates TCP data connection to the client side.
- FTP sends exactly one file over the data connection and then closes the data connection.
- Usually the control connection remains open throughout the duration of the user session. A new data connection is created for each file transferred within a session.

During an FTP session, the FTP server must maintain the state about the user. For example, the server must keep track of the user's current direction as the user changes directions. Keeping track of this state information for each ongoing user session significantly constrains the number of sessions for a FTP server to maintain simultaneously.

HTTP server uses stateless connections and thus can handle more clients.

FTP commands

- Access control commands, for example:

  `USER` (user name), `PASS` (password), `CWD` (change working directory), `QUIT` (logout).

- Transfer parameter commands, for example:

  `PORT` (data port), `TYPE` (representation type).

- FTP service commands, for example:

  `RETR` (retrieve, i.e., “get”), `STOR` (store, i.e., “put”), `LIST` (list), `HELP` (help).

FTP reply codes Examples:

- `200 Command okay`
- `503 Bad sequence of commands.`
- `221 Service closing control connection.`
- `125 Data connection already open; transfer starting.`
- `226 Closing data connection.`
- `230 User logged in, proceed.`
- `331 User name okay, need password.`
- `425 Can't open data connection.`
- `452 Requested action not taken. Insufficient storage space in system.`

3.4 Electronic mail

An e-mail system has three major components:

- User agents: allow users to read, reply to, forward, save and compose messages.
- Mail servers: contain mailboxes for users for each user. A message is first sent to the sender's mail server and the senders mail server sends the message to the receiver's mail server. If something is wrong for the receiver's server, then the message is held in a message queue and will try to send later.
- Protocols: most important protocol is the SMTP (Simple Mail Transfer Protocol)

SMTP

SMTP (RFC 5321) is the most important protocol of the email system. There are some characteristics made this application different from others.

- Message body uses 7-bit ASCII code only.
- Normally, no intermediate mail servers used for sending mail.
- Mail transmissions across multiple networks through mail relaying.

- Mail servers are listening at port 25.
- The sending server initiates a TCP connection to the receiving mail server.
- If the receiver's server is down, the sending server will try later. If the connection is established, then the client and the server perform some application layer handshaking. The client indicates the e-mail address of the sender and the recipient.
- Then the client sends the message to the server over the same TCP connection.

Some common commands are as follows.

- `MAIL FROM`: The client notifies the receiver of the originating email address of the message.
- `RCPT TO`: The corresponding SMTP for the recipient's address. Each successful reception and execution of a command is acknowledged by the server with a result code and response message.
- `DATA`: The transmission of the body of the mail message is initiated with this command after which it is transmitted verbatim line by line and is terminated with an end-of-data sequence. This sequence consists of a new-line (`<CR><LF>`), a single full stop (period), followed by another new-line.
- `QUIT`: This command ends the session. If the email has other recipients located elsewhere, the client would QUIT and connect to an appropriate SMTP server.

```
S: 220 smtp.example.com
C: HELO send.example.org
S: 250 Hello send.example.org, I am glad to meet you
C: MAIL FROM:<bob@example.org>
S: 250 bob@example.org ... Sender ok
C: RCPT TO:<alice@example.com>
S: 250 alice@example.com ... Recipient ok
C: DATA
S: 354 End data with <CR><LF>.<CR><LF>
C: Date: Tue, 15 January 2008 16:02:43 -0500
C: Subject: Test message
C:
C: Hello Alice.
C: This is a test message with 3 header fields and 4 lines.
C: Your friend,
C: Bob
C: .
S: 250 Message accepted for delivery
C: QUIT
S: 221   smtp.example.com closing connection
```

Some other commands are

- `RSET` (Reset): the current mail transaction will be aborted.
- `VRFY` (Verify): ask the receiver to confirm that the argument identifies a user or mailbox.
- `EXPN` (Expand): ask the receiver to confirm that the argument identifies a mailing list, and if so, to return the membership of that list.
- `HELP` (Help): let the server send helpful information to the client.

Reply codes:

- `220 <domain> Service ready`
- `221 <domain> Service closing transmission channel`
- `250 Requested mail action okay, completed`
- `354 Start mail input; end with <CRLF>.<CRLF>`
- `421 <domain> Service not available, closing transmission channel`
- `451 Requested action aborted: local error in processing`
- `500 Syntax error, command unrecognized (This may include errors such as command line too long)`
- `501 Syntax error in parameters or arguments`

Mail message formats are defined in RFC 5322. The header lines and the body messages are separated by a blank line. Note that the message format is not the SMTP commands.

```
From: "Joe Q. Public" <john.q.public@example.com>
To: Mary Smith <mary@x.test>, jdoe@example.org, Who? <one@y.test>
Cc: <boss@nil.test>,  <sysservices@example.net>
Date: Tue, 1 Jul 2003 10:52:37 +0200
Message-ID: <5678.21-Nov-1997@example.com>

Hi everyone.
```

After the message header, a blank line follows, then the message body (in ASCII) follows.

MIME

SMTP uses ASCII code only, which simplifies the structure but causes the limitations of using languages other than English. People also want to use email to send binary files. Multipurpose Internet Mail Extensions (MIME) was developed for solving these problems.

MIME defines five new message headers:

Header	Meaning
MIME-version:	Identifies the MIME version
Content-Description:	Human-readable string telling what is in the message
Content-ID:	Unique identifier
Content-Transfer-Encoding:	How the body is wrapped for transmission
Content-Type:	Type and format of the content

MIME defines five transfer encoding schemes, plus an escape to new scheme – just in case.

- ASCII characters use 7 bits.
- 8-bit characters, that is, all values from 0 to 255 are allowed.
- Base64 encoding: change binary codes to a form that satisfies the rules of 7 bits (ASCII code).
- Quoted-printable encoding: used for contents mostly in ASCII code, but a small part is not ASCII code.
- Binary.

A user also can specify a user-defined encoding in the `Content-Transfor-Encoding` header.

MIME types were defined in RFC 1521. Each type has one or more available subtypes. Hundreds of subtypes have been added since then.

Type	Example subtypes	Description
text	plain, html, xml, css	Test in various formats
image	gif, jpeg, tiff	Pictures
audio	basic, mpeg, mp4	Sounds
video	mpeg, mp4, quichtime	Movies
model	vrml	3D model
application	octet-stream, pdf, zip	Data produced by applications
message	http, rfc822	Encapsulated message
multipart	mixed, alternative, parallel	Combination of multiple types

The `multipart` type allows a message to contain more than one part, with the beginning and end of each part being clearly delimited.

The `mixed` subtype allows each part to be a different type, with no additional structure imposed.

Many email program allow the user to provide one or more attachments to a text message. These attachments are sent using multipart type.

The `alternative` subtype allows the same message to be included multiple times but expressed in different media. For example, a message could be sent in ASCII, in HTML and in PDF. A properly designed user agent would display it according to user preferences.

The `alternative` subtype can also be used for multiple languages. The `parallel` subtype is used when all parts must be "viewed" simultaneously. For example, movies often have an audio channel and a video channel. Movies are more effective if these two channels are played back in parallel.

The type and subtype are separated by a slash, such as "`Content-Type: video/mpeg`".

An example:

```
From: alice@cs.lakeheadu.ca
To: bob@ee.lakeheadu.ca
MIME-Version: 1.0
Message-Id:<09384756.AA785104@cs.lakeheadu.ca>
Content-Type: multipart/mixed; boundary=frontier
Subject: Try multi-part

This is a message with multiple parts in MIME format.
--frontier
Content-Type: text/plain

This is the body of the message.
--frontier
```

```
Content-Type: application/octet-stream
Content-Transfer-Encoding: base64

PGh0bWw+CiAgPGhlYWQ+CiAgPC9oZWFkPgogIDxib2R5PgogICAgPHA
+VGhpcyBpcyB0aGUgYm9keSBvZiB0aGUgbWVzc2FnZS48L3A+CiAgPC9
ib2R5Pgo8L2h0bWw+Cg==
--frontier--
```

Mail access protocols

POP3 is a simple mail access protocol defined in RFC 1939. POP3 server will listen at port 110. POP 3 progresses through three phases:

- Authentication: the user agent sends a user name and password to authenticate the user.
- Transaction: the user agent retrieves messages; the user agent can also make or remove a mark for a message deletion and get statistics of the user maildrop.
- Update: after the user issued a quit command in the transaction phase, the POP3 server removes all messages marked as deleted.

Let us see an example. At the first phase:

```
S:   +OK POP3 server ready
C:   USER Bob
S:   +OK
C:   PASS some string
S:   +OK user successfully logged on
```

In second phase, a user agent can be configured to "download and delete" or " download and keep" mode. In download-and-delete mode, commands `LIST` (list the mail number and size), `RETR` (download the message) and `DELE` (mark deletion) can be used by the user agent.

```
C:     STAT
S:     +OK 2 320
C:     LIST
S:     +OK 2 messages (320 octets)
S:     1 120
S:     2 200
S:     .
C:     RETR 1
S:     +OK 120 octets
S:     <the POP3 server sends message 1>
S:     .
C:     DELE 1
S:     +OK message 1 deleted
C:     RETR 2
S:     +OK 200 octets
S:     <the POP3 server sends message 2>
S:     .
C:     DELE 2
S:     +OK message 2 deleted
C:     QUIT
S:     +OK  POP3 server signing off (maildrop empty)
C:   <close connection>
S:   <wait for next connection>
```

IMAP (RFC 3501) is another mail access protocol, which has more features than POP3.

An IMAP server will associate each message with a folder. When a message first arrives at the server, it is associated with the recipient's INBOX folder. The recipient can then move the message into a new, user created folder, read the message, delete the message, and so on.

IMAP also provides commands that allow users to search remote folders for messages matching specific criteria.

Another important feature of IMAP is that it has commands that permit a user agent to obtain components of messages. For example, a user agent can obtain just the message header of a message or just one part of a multipart MIME message.

HTTP are now used for Web-based email accessing. Hotmail introduced Web-based access first. Now Web-based emails are provided by many corporations and universities including Google, Yahoo etc. In this approach, the user agent is an ordinary Web browser, and the user communicates with its remote server via HTTP.

3.5 P2P applications

In P2P, each node (called peers) acts somehow as a client and server at the same time. The peers are not owned by a service provider and not supposed to be always listening on the Internet. The peers are dynamic, i.e., some peers will join some peers will leave from time to time.

P2P file distribution

Suppose a server has a large file, which N computers want to download. In the case of client-server architecture, each of the N computers will connect to the server and download a copy of the file to local. For the P2P architecture, a peer is not necessary download a copy from the server. It may download from other peers.

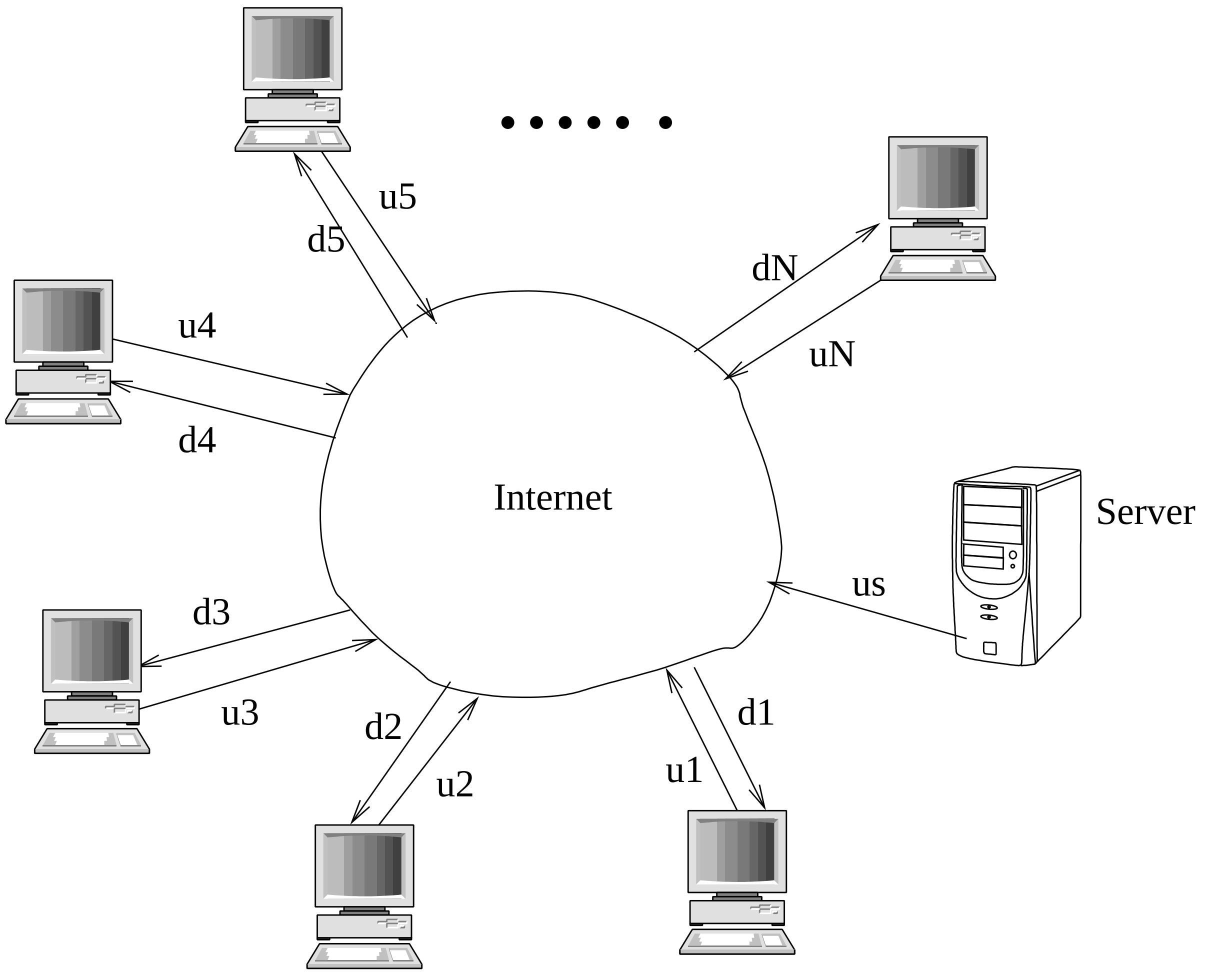


Figure 1: P2P architecture

Suppose the length of the file is F. The upload rate for the server is u_s. The upload rate and the download rate for ith computer is u_i and d_i, respectively.

First let us look at the distribution time for the client-server architecture, that is denoted as D_{cs}.

- The server must transmit one copy of the file to each of the N peers. So the server needs transmit NF bits. Since the server's upload rate is u_s, the distribution time is at least NF/u_s.
- Let d_{min} denote the download rate of the peer with the lowest download rate. The peer with the lowest download rate cannot obtain all F bits of the file in less than F/d_{min}.

Put the above two observations together, we have

$$D_{cs} \geq \max\left\{\frac{NF}{u_s}, \frac{F}{d_{min}}\right\}.$$

Next we can look at the same situation, but using the P2P architecture. Let D_{P2P} denote the distribution time for the P2P mode.

- At the beginning of the distribution, only the server has the file. So the minimum distribution time is at least F/u_s.
- Similar to the situation of client-server architecture, the peer with the lowest download rate cannot obtain all the F bit in less than F/d_{min}. So the minimum distribution time is at least F/d_{min}.

- Observe that the total upload capacity of the system as a whole is $u_{total} = u_s + u_1 + u_2 \cdots + u_N$. Since the system must deliver F bits to each of the N peers, the minimum distribution time is at least $NF/(u_s + u_1 + u_2 \cdots + u_N)$.

So we have

$$D_{P2P} \geq \left\{ \frac{F}{u_s}, \frac{F}{d_{min}}, \frac{NF}{u_s + \sum_{i=1}^{N} u_i} \right\}.$$

To compare the distribution time, we assume that all the upload rate for the peers is u. When N is large, D_{cs} is about $\frac{NF}{u_s}$, while D_{P2P} is about

$$\frac{NF}{u_s + \sum_{i=1}^{N} u} < \frac{F}{u}.$$

The above comparison shows that when N is large, the P2P architecture is not only consuming less distribution time, but also be self-scaling. The distribution time can be independent to the number of peers.

BitRorrent

In BitTorrent protocol, the collection of all peers participating in the distribution of a particular file is called a torrent.

- Peers in a torrent download equal-size chunks of the file from one another, with a typical chunk size of 256 KBytes.
- While a peer downloads chunks it also uploads chunks to other peers.
- Once a peer has acquired the entire file, it may leave the torrent or remain in the torrent and continue to upload chunks to other peers.
- Also, any peer may leave the torrent at any time without download whole file and rejoin the torrent later.

Each torrent has an infrastructure node called tracker. When a peer joins a torrent, it registers itself with the tracker and periodically informs the tracker that it is still in the torrent.

When a new peer, Alice, joins the torrent, the tracker randomly selects a subset of peers from the set of participating peers, and sends the IP addresses of these peers to Alice. Alice then tries to establish concurrent TCP connections with all the peers on this list. We can call all the peers with which Alice succeeds in establishing a TCP connection neighboring peers.

As time evolves, some of these peers may leave and other peers may attempt to establish TCP connections with Alice.

At any given time, each peer will have a subset of chunks from the file, with different peers having different subsets.

Periodically, Alice will ask each of her neighboring peers for the list of chunks they have. So at an instant time, Alice will have a subset of chunks and will know which chunks her neighbors have.

To choose the chunks to download, Alice uses a technique called rarest first. The idea is to determine, from among the chunks she does not have, the chunks that are the rarest among her neighbors and then request those rarest chunks first.

To determine which requests she responds to, Alice uses a trading algorithm.

Alice gives priority to the neighbors that are currently supplying her data at the highest rate. Specifically, for each of her neighbors, Alice continually measures the rate at which she receives bits and determines the four peers that are feeding her bits at the highest rate.

She then reciprocates by sending chunks to these four peers. Every 10 seconds, she recalculates the rates and possible modifies the set of four peers. In BitTorrent lingo, these four peers are said to be unchoked.

Every 30 seconds, she also picks one additional neighbor at random and sends it chunks. In BitTrront lingo, it is said to be optimistically unchoked.

All other neighboring peers besides these five peers are choked peer.

Distributed hash table (DHT)

We can use P2P architecture to form a distributed database.

We consider a simple database, which contains (key, value) pairs. In this architecture, each peer will only hold a small subset of the data. Any peer can query the distributed database with a particular key. The database will then locate the peers that have the corresponding (key, value) pairs and return the key-value to the querying peer.

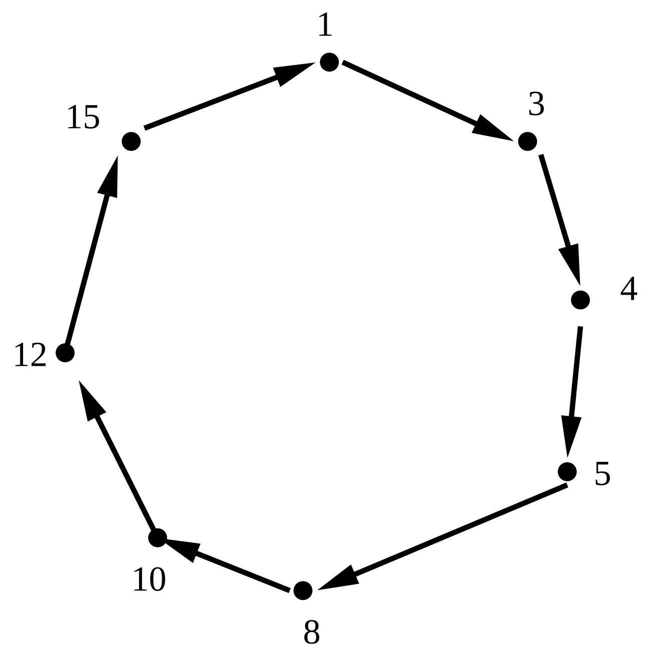
Any peer will also be allowed to insert new key-value pairs into the database.

Such a distributed database is referred to as a distributed hash table (DHT).

To construct the database, we need to use some hash functions. The input of a hash function can be a large number, but the output of the hash function is of fixed size bit string.

The outline of building a DHT is as follows.

- Assign an identifier to each peer, where the identifier is an n-bit string. So we can view the identifier as an integer at the range from 0 to $2^n - 1$.
- For a data pair (key, value), the hash value of the key is computed. We suppose that the hash value is of n-bit (with the same range as the identifiers). Then the data is stored in the peer whose identifier is closest to the key. For convenience, we define the closest peer as the closest successor of the key.
- To insert or retrieve data, first we need to find the appropriate peer. It is not realistic to let the peer to store all of the other peer's identifiers (and the associated IP addresses) or arrange a data center to provide the service (that will damage the P2P architecture). So the DHT uses a circular arrangement. Figure **??** can be used to explain the idea.



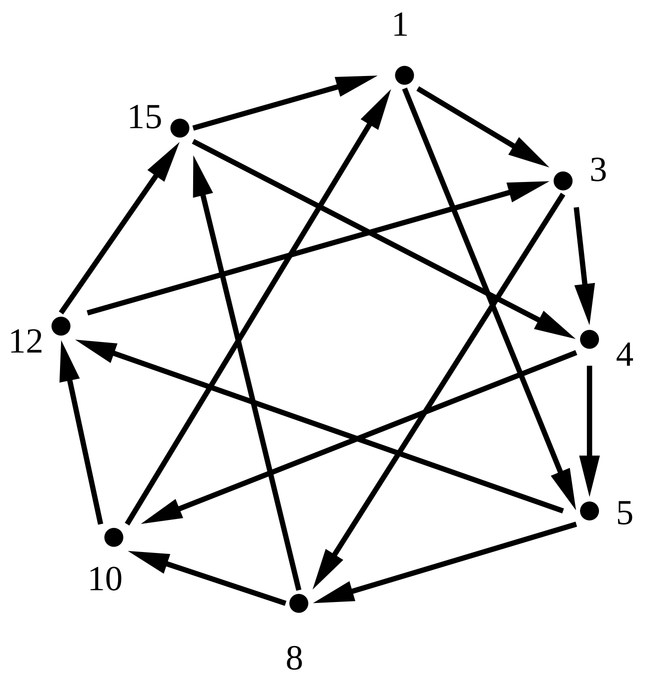


Figure 2: Circular DHT

In this example, the identifiers are range $[0, 15]$ (4-bit strings) and there are eight peers. In the circular arrangement, each peer is only aware of its immediate successor and predecessor (as show at the left side of Figure **??**). So each peer just needs to track two neighbors. When a peer asks for a key, it sends the message clockwise around the circle. For example, the peer 4 sends a message saying "Who is responsible for key 11?". The message will forward through peers 5, 8, 10 and reach peer 12, who determines that it is the closest peer to key 11. At this point, peer 12 can send a message back to the querying peer 4, indicate that it is responsible for key 11. But when the circle is too large, a message may go through a large number of peers to get answer.

There is trade off between the number of neighbors each peer has to track and the number of message that the DHT needs to send to resolve a single query. One method can be used to reduce the query time (but increase neighbors) is add "shortcuts" to the circular arrangement. The right side of Figure **??** explains the idea. In general, researchers provide some methods to determine how to arrange shortcuts for peers. It is proved that the DHT can be designed so that both the number of neighbors per peer as well as the number of messages per query is $O(\log N)$, where N is the number of peers.

- To handle the situation that a peer may join or leave from time to time in a DHT, researchers also proposed various methods. One method for peers' leaving is that a peer keeps track s successors. So when a peer leaves, the other peers still can update their tackers. For example, let $s = 2$ in Figure **??**. Suppose peer 5 left. Since each peer will periodically verify its two successors (by ping its successors, for example), peers 3 and 4 will know that peer 5 departed. Then peer 4 will replace the immediate successor as peer 8 and ask peer 8 for its successor's identifier (in this case, it is peer 10). Peer 3 will ask peer 4 for its updated immediate successor's identifier and update its second successor (Peer 8).

Another case should be considered is if some peer will join the DHT. For example, if the peer 13 wants to connect to the DHT. When it joins, it only knows the peer 1. The peer 13 will send a request "what will be peer 13's predecessor and successor?". This message will get forwarded through the DHT until it reaches peer 12. Peer 12 knows that it is 13's predecessor and peer 15 is 13's immediate successor. So peer 12 will inform peer 13 and peer 13 will update its predecessor and successor. It also inform peer 15 and peer 12 to update their connection tracks.