APPLICATION PROTOCOL DESIGN

Protocol

- Set of rules:
  - Message format
  - Message sequence
  - Process message
- Goals
  - Everyone must know
  - Everyone must agree
  - Unambiguous
  - Complete

Example: POP session

```
C: <client connects to service port 110>
S: +OK POP3 server ready <1896.6971@mailgate.dobbs.org>
C: USER bob
S: +OK bob
C: PASS redqueen
S: +OK bob's maildrop has 2 messages (320 octets)
C: LIST
S: +OK 2 messages (320 octets)
S: 1 120
S: 2 200
S: .
C: QUIT
S: +OK dewey POP3 server signing off (maildrop empty)
C: <client hangs u>
```

Example: FTP authentication

```
>   ftp 202.191.56.65
C: Connected to 202.91.56.65
S: 220 Servers identifying string
User: abcd (C: USER abcd)
S: 331 Password required for tungbt
Password:(C: PASS)
S: 530 Login incorrect
C: ls
S: 530 Please login with USER and PASS
C: USER abcd
S: 331 Password required for abcd
Password:(C: PASS)
S: 230 User abcd logged in
```

Steps in design

1. Define services
2. Choose application model(client/server, P2P,…)
3. Establish the design goals
4. Design the message structure: format, fields, types of messages, encoding, …
5. Protocol processing
6. Interaction with environment (DNS, DHCP…)

Design Goals

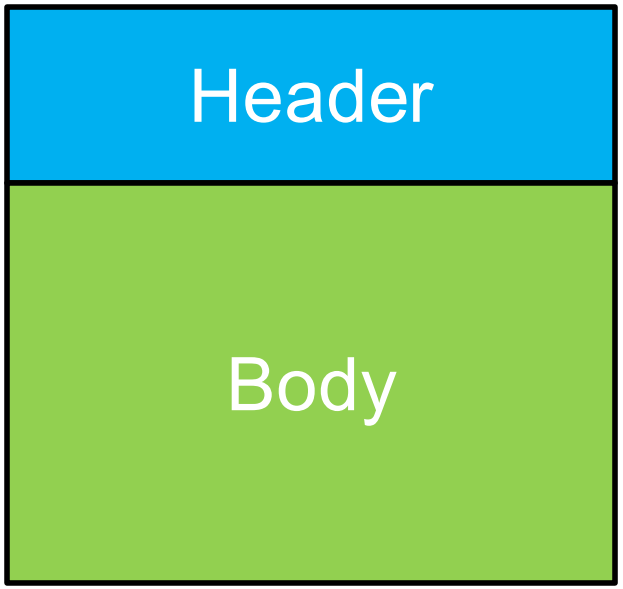
- Do we need reliable exchanges?
- How many types of parties are involved? Can they all communicate to each other?
- Is the authentication of parties needed
- How important is the authentication of parties?
- Is the transferred data confidential? What degree of authorization is needed?
- Do we need complex error handling?

Design Issues

- Is it to be stateful vs stateless?
- Is the transport protocol reliable or unreliable?
- Are replies needed?
  - How to respond to lose replies?
- Is it to be broadcast, multicast or unicast?
  - Broadcast, multicast: must use UDP Socket
- Are there multiple connection?
  - How to synchronize?
- How many types of parties are involved? Can they all communicate to each other?
- Session management
- Security: authentication, authorization, confidential…

Designing the Message

- Header: contains structured fields describing the actual data in the message, such as
  - message type
  - command
  - body size
  - recipient information
  - sequence information
  - retransmission count…
- Body: the actual data to be transmitted:
  - the command parameters
  - the data payload



The simplest formats:

- Type – Length – Value(TLV)
- Type – Value

Control Messages

- Define the stages of the dialogue between the parties
- Control the dialogue between the parties
  Address various communication aspects:
  - communication initiation or ending
  - describe the communication stage (e.g. authentication, status request, data transfer)
  - coordination (e.g. receipt confirmation, retry requests)
  - resource changes (e.g. requests for new communication channels)
- Usual format:

Command	Parameter

  - Command: SHOULD has fix length or use delimiter
  - Example: USER, PASS, PWD (FTP),

Data transfer

- Messages that carry data over the network
- They are usually sent as a responses to specific commands
- Data is usually fragmented in multiple messages
- Header describe:
  - the type of the binary data format
  - clues for the layout of the structured data (when the structure is flexible/dynamic)
  - data size, offset or sequence information
  - type of the data block: last / intermediary
  -

Message Format

Byte oriented

- The first part of the message is typically a byte to distinguish between message types.
- Further bytes in the message would contain message content according to a pre-defined format
- Advantages: compactness
- Disadvantages: harder to process, debug or test
- Example: DHCP, DNS

Data Format

Text-oriented

- A message is a sequence of one or more lines
- The start of the first line of the message is typically a word that represents the message type.
- The rest of the first line and successive lines contain the data.
- Advantage:
  - easy to understand, monitor
  - flexible
  - easy to test
- Example: HTTP, FTP, email protocols

- Disadvantage
  - may make the messages unjustifiably large
  - may become complex

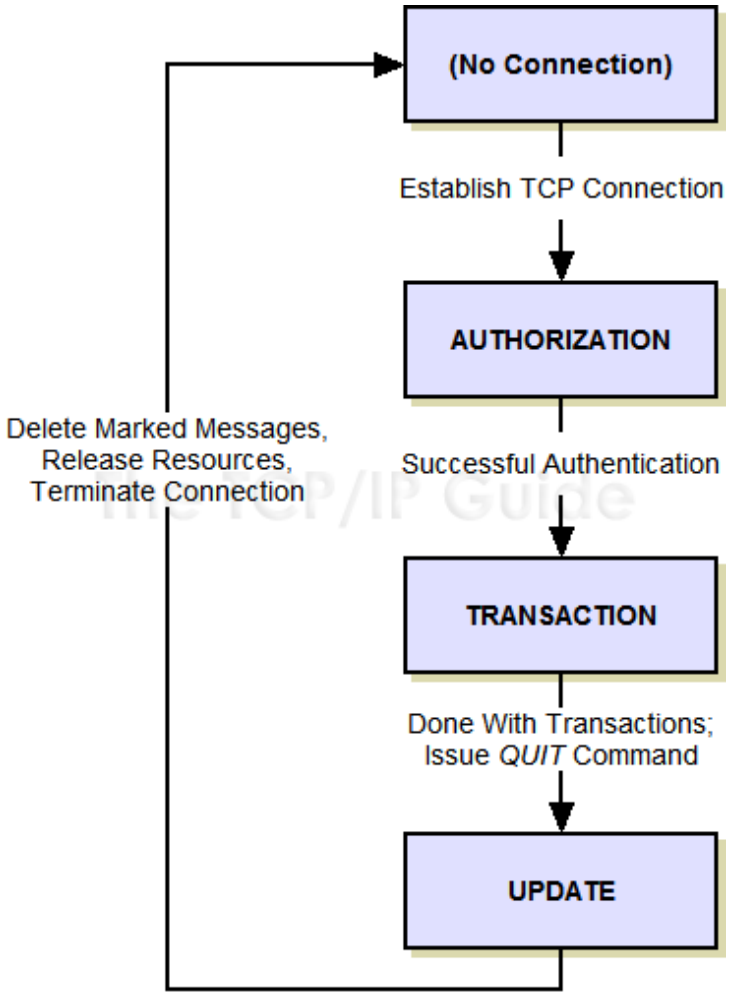
Protocol Processing

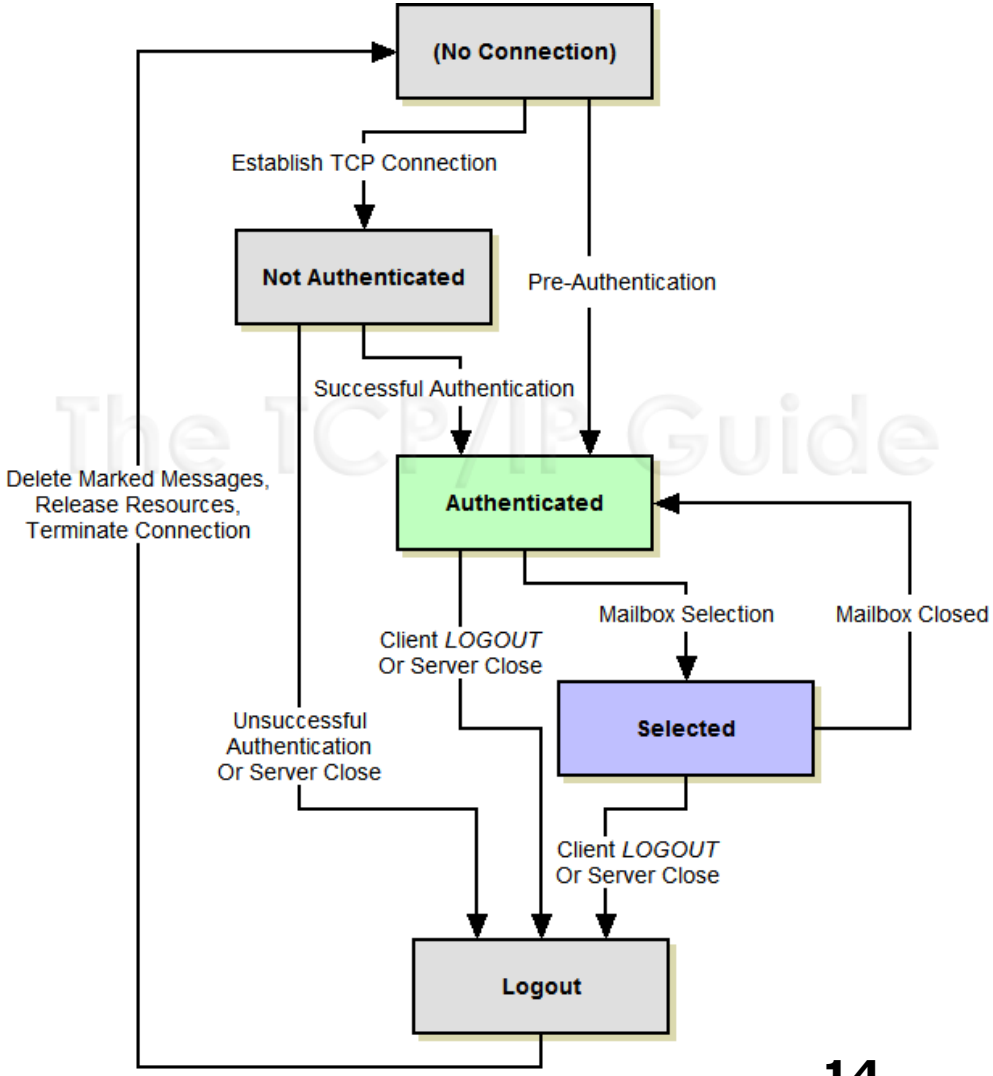
- Describe the sequences of messages, at each and all the stages in the of each communication scenario, for all parties in the system
- Finite State Machine is mandatory:
  - State: 
  - Transaction: 

  - Choose: 
- And/ Or use state Table

Current state	Transaction		Next state
	Receive	Send	

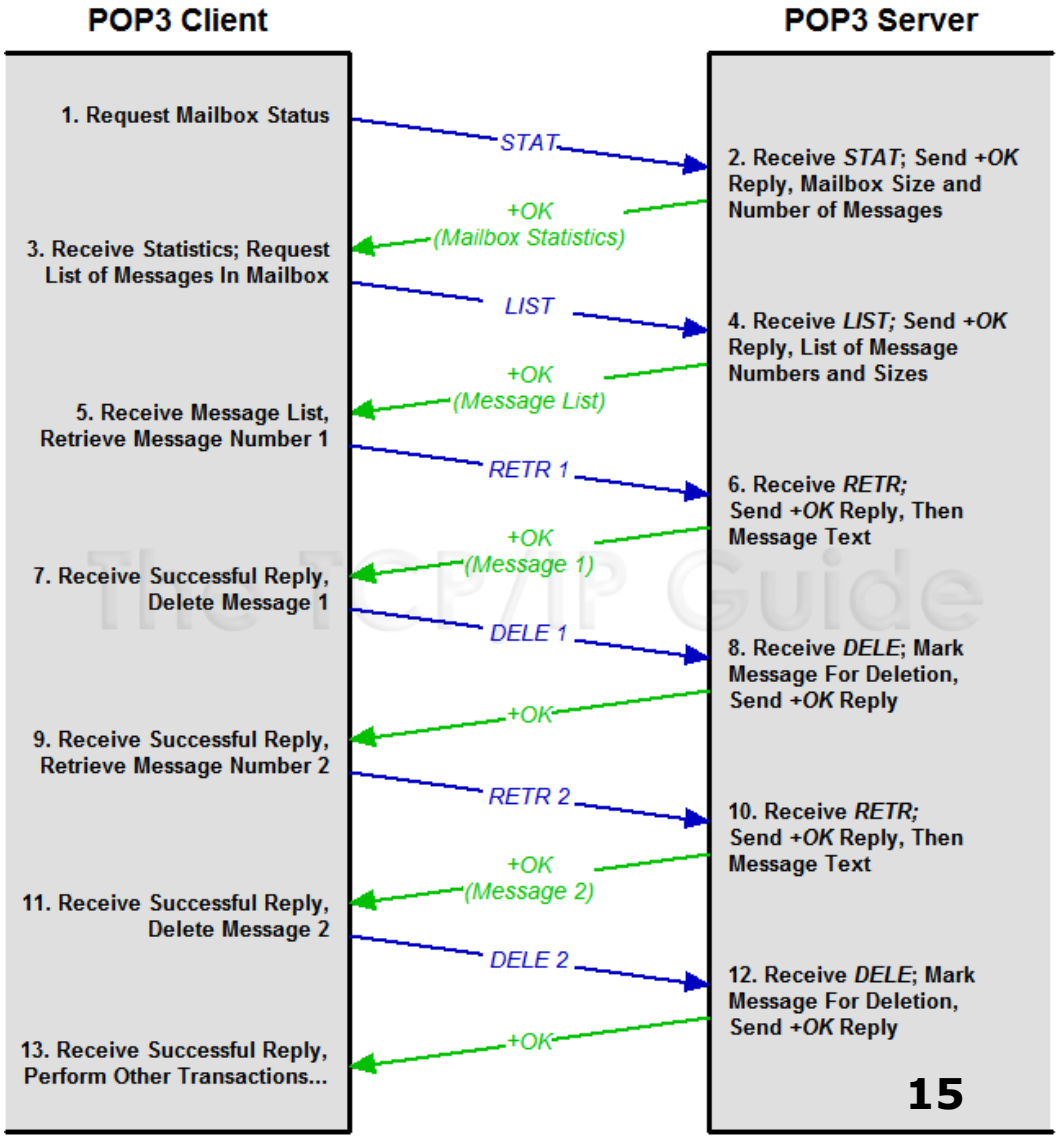
Example: POP3 and IMAP4 session





Message Transaction Diagram

- Represents the sequence of message transaction
- Example: POP3



Implementing an Application Protocol

- Type of message
  - Use integer: `enum msg_type {…}`
  - Use string
- Data structure
  - Use `struct`. Example:

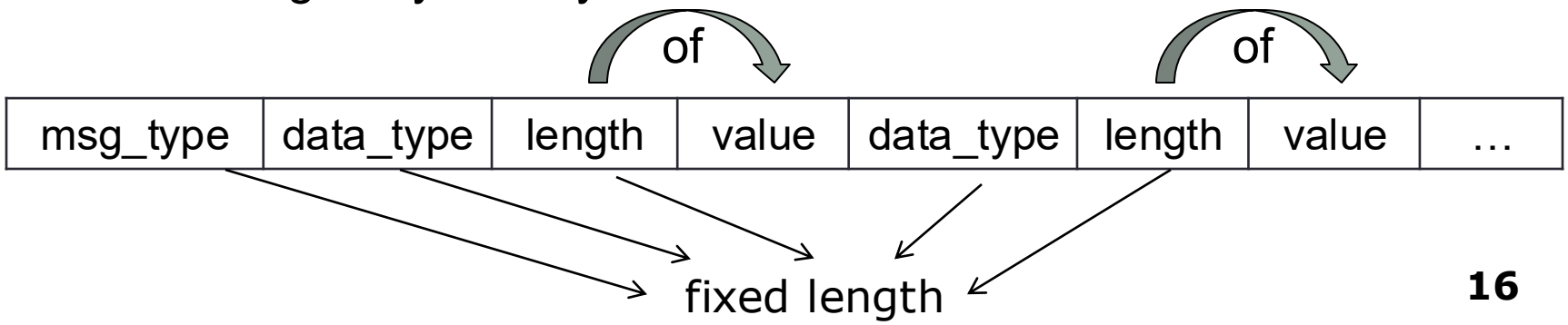
```
struct message{
        char msg_type[4];
        char data_type[8];
        int value;
}
```

or

```
struct msg_payload{
        int id;
        char fullname[30];
        int age;
        //...
}
```

```
struct message{
    msg_type type;
    struct msg_payload payload;
};
```

  - Use string or byte array



Implementing an Application Protocol

- Message handler(pseudo code)

```
//handle message
switch (msg_type){
    case MSG_TYPE1:
         {
            //...
         }
    case MGS_TYPE2:
         {
            //...
            if(data_type == DATA_TYPE1)
            //...
         }
    //...
}
```

Bài tập

- Tạo chương trình UDP / TCP Client / Server hoạt động như sau:
  - Client có thể gửi đến máy chủ tên đăng nhập (login name) hoặc một bản tin
  - Nếu client gửi login name thì server đồng ý kết nối và nhớ login name này
  - Nếu client gửi một tin nhắn văn bản, server tin nhắn vào một file log. Mỗi client sẽ có một file log riêng.
- Cần làm:
  - Xác định message type cho login và text message
  - Xác định các trường trong mỗi message và độ dài cũng như kiểu của từng trường
  - Xác định xem client/server xử lý các message như thế nào
  - Vẽ protocol state machine
  - Coding