Computer Networks

COMP 3670

Application Layer

Sherif Saad

School of Computer Science | University of Windsor

May 28, 2021

Table of Contents

Network Applications Characteristics

What are the main characteristics of network applications?

1. It consists of several distributed components communicate by passing messages.
2. The components have to follow a set of rules (protocols) to communicate.
3. The components could be written in any programming language and technology stacks.
4. The application could be deployed on different platforms.
5. The fact that the application is distributed is hidden or has minimum effect on the end user's experience.
6. The application is always available to the end-users.
7. The end-user experience should not be affected by the concurrent usage of the application by other end-users.

Developing Network Applications

What are the principles of developing network applications?

- Application Architecture.
- Application-Layer Protocols.
- Processes Communications.
- Transport Services.

Application Architecture

Definition

Deciding on software components roles, their interaction, and their placement leads to an instance of a software architecture, also known as a system architecture.

System Architecture could be categorized into:

- Centralized Organizations.
- Decentralized Organizations.
- Hybrid Organizations.

Centralized System Architectures

Basic Client-Server Model

Characteristics:

- There are processes offering services (servers).
- There are processes that use services (clients).
- Clients and servers can be on different machines.
- Communications follow request/reply model with respect to using services.

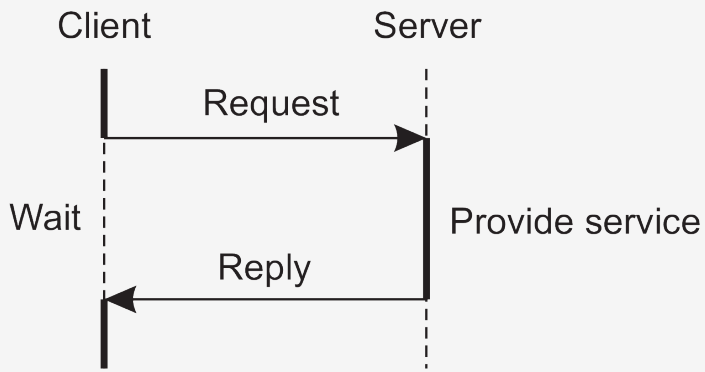


Figure: Basic Client Server

Multi-tiered centralized system architectures

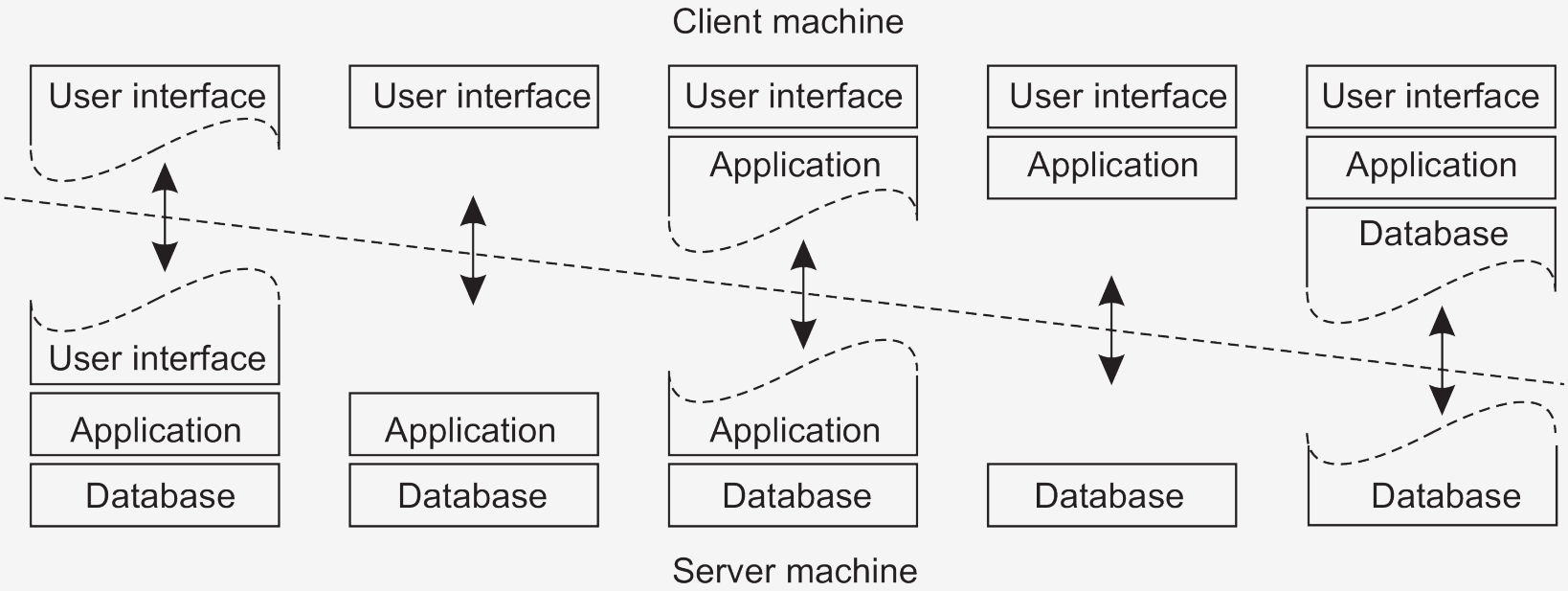


Figure: From Thin to Fat Client

What are the pros and cons of thin and fast clients?

Being client and server at the same time

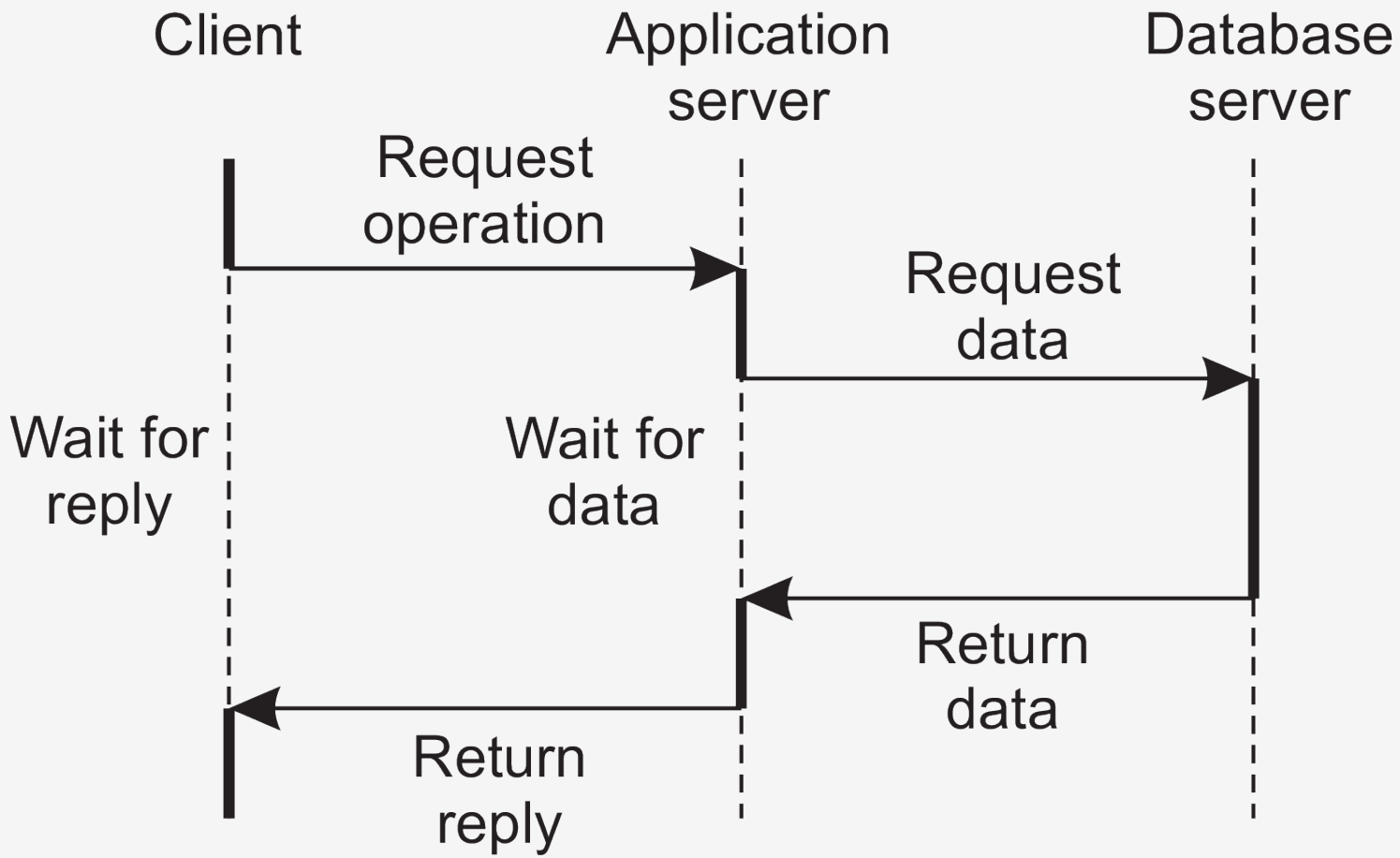


Figure: Three-tiered architecture

Decentralized System Architecture

P2P Architecture

Processes are all equal: the functions that need to be carried out are represented by every process and each process will act as a client and a server the same time (i.e., acting as a servant).

P2P architecture, could be categorized into:

- Structured P2P
- Unstructured P2P

Structured P2P

Key Concepts

- The node are organized into a ring, a binary tree, or a grid.
- This topology is every efficient when executing look-up operation and routing messages.
- Make use of a semantic-free index: each data item is uniquely associated with a key, in turn used as an index.
- Heavily relays on hashing for indexing resources and nodes within the system.
- The index is simply the hash value of the resource name or identifier.

Unstructured P2P

Key Characteristics

- Each node maintains an ad-hoc list of neighbors.
- The resulting overlay resembles a random graph.
- The link between any 2 nodes exist only with a certain probability.
- Look-up operations and routing messages is more complex and less efficient comparing to structured P2P systems.
- Commonly apply flooding or random-walk for Look-up operations and routing messages.

Hybrid System Architectures

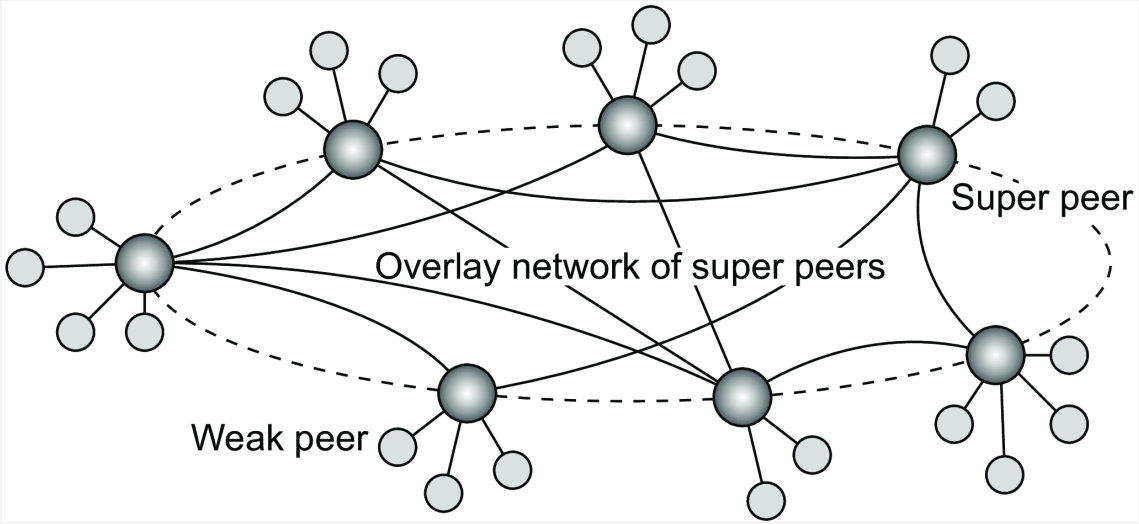


Figure: Hybrid Architecture: Super-Peer Networks

- When searching in unstructured P2P systems, having index servers improves performance.
- Deciding where to store data can often be done more efficiently through brokers.
- How to select the super-peer?

What is Overlay Network?

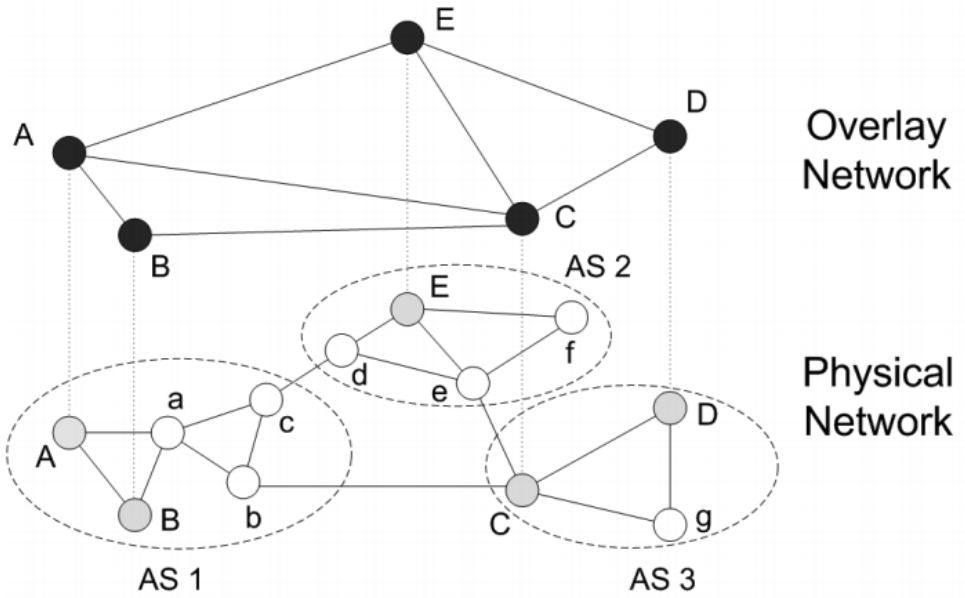
Definition

An overlay network is a communication network constructed on top of another communication network.

The overlay network is a logical or virtual network that is overlaid on a physical network (underlay network). In overlay network:

- The nodes are interconnected using logical connections (overlay topology).
- The logical link usually operates over several hops apart in the underlying network.
- The neighbors nodes are dynamics, the topology of the network possibly unknown.

Overlay Networks



Link overlay	Physical path
A-B	A-B
A-C	A-**a**-b-C
A-E	A-**a**-c-d-E
B-C	B-**a**-b-C
C-D	C-D
C-E	C-**e**-E
D-E	D-C-**e**-E

Figure: Overlay Network Architecture

Fundamentals of Processes Communication

- When developing network applications, we need to understand how the different components deployed at multiple hosts communicate with each other.
- To be specific, distributed network processes are communicating to achieve common goals (e.g. downloading a file, sending an email message, purchase airline tickets, etc).
- Inter-process communication (IPC) refers to the different mechanisms processes use to communicate to manage shared data and resources.
- When processes on the same host communicate, they rely on IPC mechanisms provided by the host operating system (e.g. pipes, shared memory, data buses, etc.)
- Processes running on different hosts communicate by exchanging messages. Processes need to synchronize to coordinate their actions and to share data that they are cooperatively working on

Types of Communication

Synchronous and Asynchronous

- In synchronous, the sender of message **blocks** until the message has been received by the intended recipient.
- In Asynchronous, the sender **continues** execution immediately after sending a message.

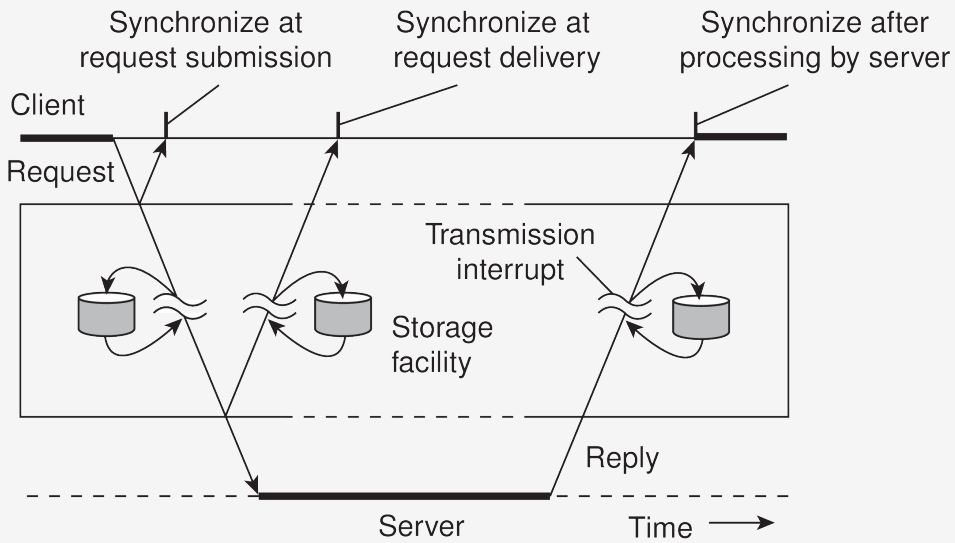


Figure: Synchronous and Asynchronous

Types of Communication

Places for Synchronization

- At request submission
- At request delivery
- After request processing

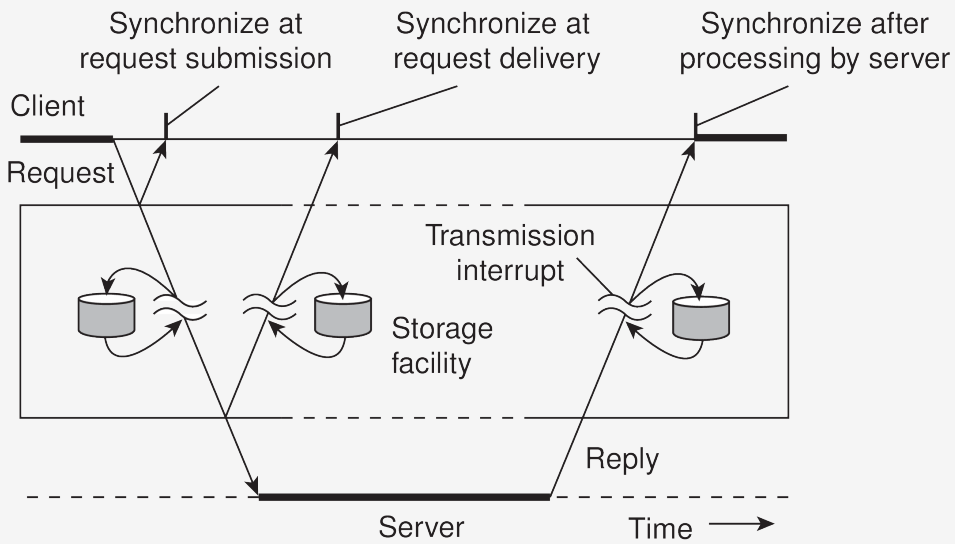


Figure: Synchronous and Asynchronous

Types of Communication

Transient and Persistent

- In transient mode, a message will only be delivered if a receiver is **active**
- In persistent mode, a message will be stored in the system until it can be delivered to the intended **recipient**.

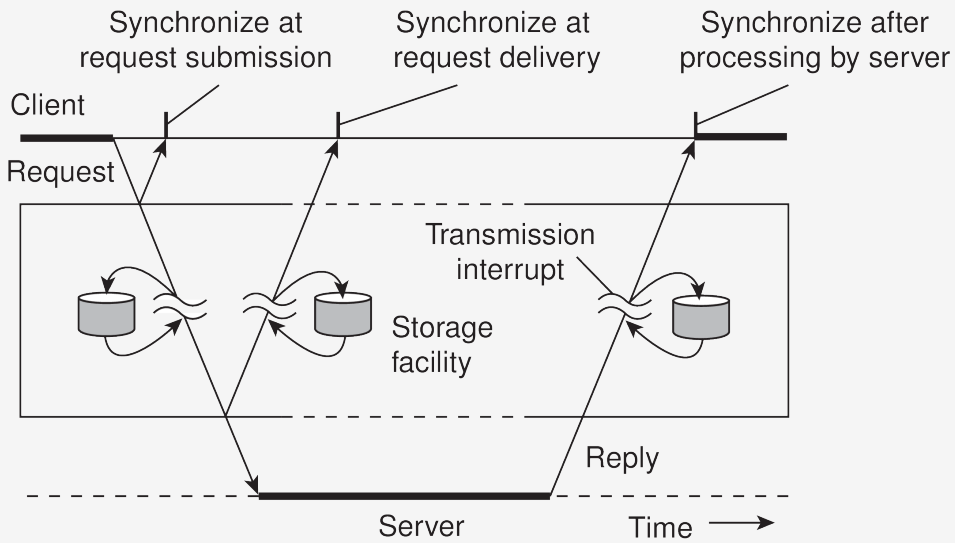


Figure: Transient and Persistent

Types of Communication

Reliable and Unreliable

- In reliable communication errors are discovered and fixed transparently. The processes can assume that a message that is sent will actually arrive at the destination.
- In unreliable communication messages may get lost and processes have to deal with it.

Types of Communication

Ordered Message Delivery

- Provide guarantees about the ordering of messages.
- Guarantee that all messages are received in the same order that they are sent.

Communication Mode

- **One-to-One:** one sender sends to one receiver.
- **One-to-Many:** one sender sends to many receivers.

Types of Communication

Stateless and Stateful

- In stateless communication every message sent by the communicated processes must be self-contained. Session or connection state is not stored by any node
- In stateful communication require at least one node to maintain the session state, usually the server maintain the session state.

Full-Duplex and Half-duplex

- In full-duplex the communicated process are capable of both transmitting and receiving data at the same time.
- In half-duplex communication can only go in one direction at a time, usually the processes take turn when sending messages (e.g. woki toki, http)

Network Socket

- A pair of processes are communicating over a computer network using network sockets.
- One common IPC in computer networks is using network sockets.
- The process sends and receives messages to/from the network using sockets.
- A socket is a communication endpoint to which an application can write data that are to be sent out over the underlying network, and from which incoming data can be read.
- The socket is the bridge between the application layer and the transport layer.
- A network socket is the combination of IP address plus port. Each end of the connection will have a socket.

Why do we need network sockets?

Network Ports

- In computer network a port is a communication end-point.
- A port is a logical construct that identifies a specific process over the network at a given host.
- A port number is a 16-bit unsigned integer, thus ranging from 0 to 65,535.
- A registered port is a network port designated for use with a certain protocol or application.
- Network Ports could be classified into:
  - **System | Well-known Ports:** from 0-1023
  - **Registered | User Ports:** from 1024-49151
  - **Dynamic | Ephemeral Ports:** from 49152–6553

Addressing Processes

- Every host has a unique address over the network, for instance in IP/TCP networks, the IP address is the unique address of the host.
- The most common version of IP address is IP versions 4 and 6. IP address version 4 is a numerical identifier of 32-bits.
- Since there are many concurrent processes running on the same host, we need an approach to distinguish between these processes when they connect to computer network.
- Network sockets are the unique identifiers to distinguish between concurrent processes of the same host that are connected to the network.

Working with Socket

Berkeley Sockets

Is a set of APIs for network prgramming that was developed by the University of California, Berkeley for BSD Unix OS. All modern operating systems implement a version of the Berkeley socket interface.

Operation	Description
socket	Create a new communication end point
bind	Attach a local address to a socket
listen	Set the maximum number of pending connection requests
accept	Block caller until a connection request arrives
connect	Actively attempt to establish a connection
send	Send some data over the connection
receive	Receive some data over the connection
close	Release the connection

Working with Socket

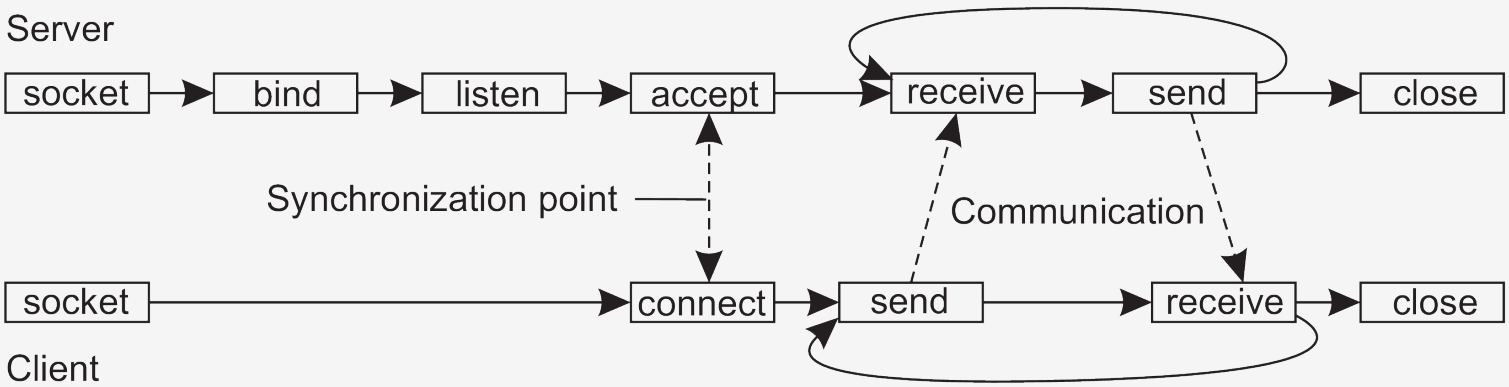


Figure: Connection Oriented Using Socket

Working with Socket

Server

```
1  from socket  import *
2  s = socket(AF_INET, SOCK_STREAM)
3  s.bind((HOST, PORT))
4  s.listen(1)
5  (conn, addr) = s.accept()  # returns new socket and addr. client
6  while True:                # forever
7    data = conn.recv(1024)   # receive data from client
8    if not data: break       # stop if client stopped
9    conn.send(str(data)+"*") # return sent data plus an "*"
10 conn.close()               # close the connection
```

Client

```
1  from socket  import *
2  s = socket(AF_INET, SOCK_STREAM)
3  s.connect((HOST, PORT)) # connect to server (block until accepted)
4  s.send('Hello, world')  # send same data
5  data = s.recv(1024)     # receive the response
6  print data              # print the result
7  s.close()               # close the connection
```

Figure: Connection Oriented Using Socket in Python

Transport Services for Network Application

- The network application developers don't need to implement the transport services to delivery the messages over the network. However, they need to understand how the transport services work and how the different transport services could affect their applications.
- Simply, the application send messages by pushing these messages through the network sockets.
- There are different transport services and each service is implemented at the transport layer and use specific transport layer protocol.

Which transport layer protocol or service we should use?

Selecting Transport Service or Protocol I

Reliable Data Transfer

- Guarantee the delivery of the data in their intend order and the correctness of the delivered data (e.g data are error free, data loss).
- What if your application is a loss-tolerant application (e.g. multimedia applications)
- If the transport service or protocol doesn't provide reliable data transfer, then the application developer must make sure that his application could tolerate data loss and data corruption.

Selecting Transport Service or Protocol II

Throughput

- In a computer network, throughput is the rate at which the ending process can deliver bits to the receiving process. Note that several processes on the same host could send data simultaneously, which means the available bandwidth is shared between these concurrent connections.
- Some application are bandwidth-sensitive application (e.g. has a specific throughput requirements, such IPTV and VOIP application)
- Some other applications are considered elastic application with respect to the throughput (e.g file transfer, web, and email)

Selecting Transport Service or Protocol III

Timing

- The transport service or protocol can also provide timing guarantees.
- The timing puts restrictions on the network latency or delay, such that the message will arrive at the receiver within a time window of 100 msec.
- This is common in multimedia applications and online gaming.

Selecting Transport Service or Protocol IV

Security

- Transport service or protocol can also provide security services to the application.
- Common security services include confidentiality, integrity, authorization, none repudiation, etc

Why not implementing the security services in the application?

The Need for Application Layer Protocols

What is an application layer protocol

Is a communication protocol that enable IPC between processes placed at different hosts. It defines howt these distributed process exchange messages to share data and manage resources.

What are the functions of application layer protocol?

- The types of messages the processes exchange.
- The syntax of the messages.
- The semantic of messages.
- The order of messages and expected responses.

Why do we need more than one application protocol?

Types Application Protocols

- Network applications that share the same communication behaviors, and provide similar services, could use the same application layer protocols.
- Application layer protocols could be categorized into open protocols and proprietary protocols.
- Open protocols are defined in RFCs (request for comments), and to receive approval from the community. They focus on interoperability, and normally become standards application layer protocols (e.g. HTTP, SMTP, FTP, etc).
- Proprietary protocols are not not available to the public domain and usually protected by intellectual property laws (e.ge. Skype, Z-Wave, etc).

Developing Application Protocols

We are building a new network application, and we need a network application protocol, what should we do?

Here are our Options:

- Use a universal communication protocol such as HTTP and design your message exchange model on top of it.
- Find an existing network protocol that fits your application requirements and use it as its.
- Design a new network protocol tailored to your application needs.

Reuse vs Design (New) Application Protocol

- The communication semantics and patterns are usually the main factors that we use to decide if we could use an existing protocol, or we need to create a new one.
- If the cost of modifying an existing protocol exceeds or equals the cost of creating a new one, then maybe we should create a new protocol.
- In addition, to technical factors, it is possible there are other non technical factors, for example you wish to create a proprietary protocol.

Using Universal Application Layer Protocol

What about HTTP?

- In this case we use HTTP to exchange the messages.
- This means HTTP defines the rules (e.g request/response) for exchanging the messages. We need to define the message structure and semantic.
- HTTP characteristics suitable for your application (e.g. stateless, uses a pull communication, half-duplex (with pipelining), etc)
- Another possible alternative is WebSocket protocol, it is state-full, full-duplex, and persistent.

Design a New Protocol

In general, we need to consider the following:

1. The desired communication patterns of the application
2. The design goals of your protocol
3. The message types, format, semantics
4. The communication rules.

Communication Patterns

What is the architecture of your network application?

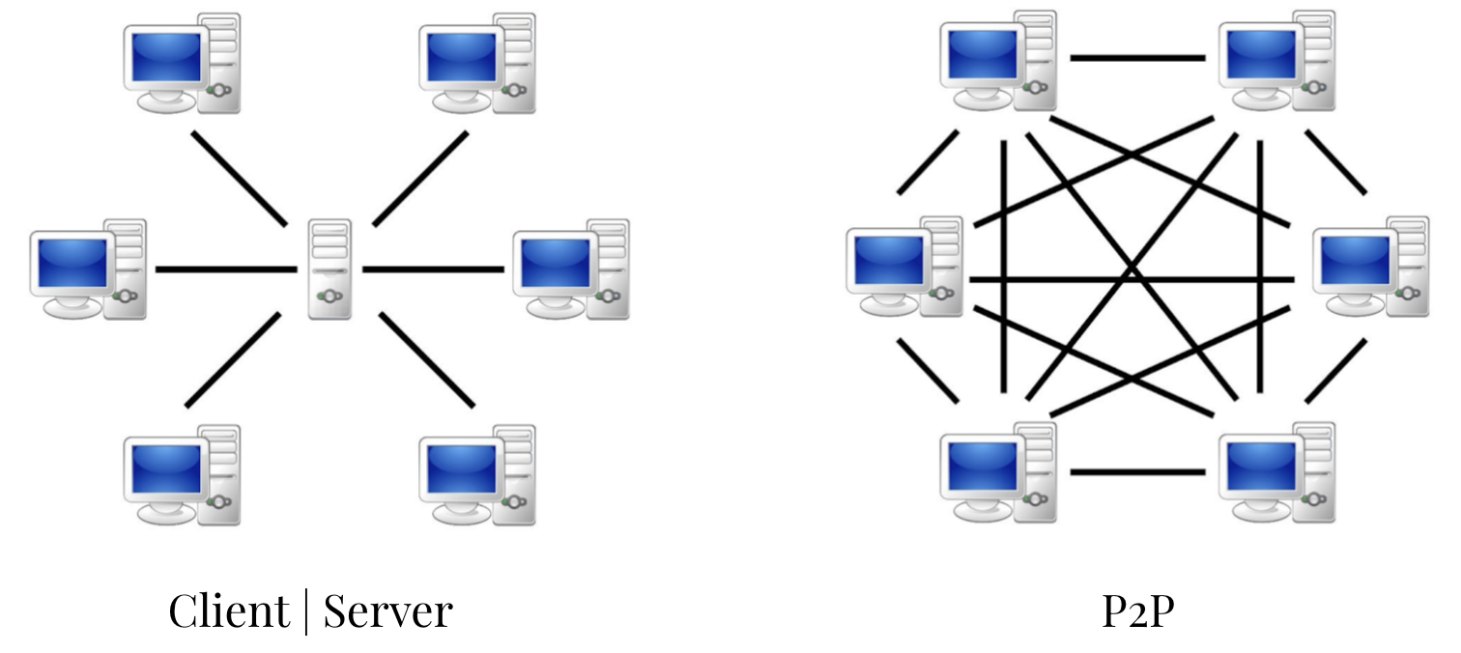


Figure: Possible Application Architecture

Communication Patterns

The communication model

- Always Push
- Always Pull
- Interactive (Push and Pull)

The communication Scope

- Oen-to-One
- Mutlicast
- Broadcast
- All the above

Protocol Design Goals

- The main quality attributes of applications protocols are: simplicity, efficiency, scalability, and extensibility.
- Do we need a reliable message exchange? (Yes | No)
- Is communication security important? (Authentication, Confidentiality, Integrity, Authorization)
- Do we need persistent connections? (Yes | No)
- Do we have any bandwidth or latency restrictions or requirements?
- How do we plan to handle error and failure?

Protocol Design Goals

- Try to make your protocol as simple as possible
  - Do not repeat yourself.
  - Too many messages is a bad practice and inefficient.
- Message size and format is important even if we do not have bandwidth limitations or constraints.
- Extensibility is a key requirement, computer networks is highly dynamic.
- What is your scalability requirements? (size, geographical, administrative)

Message Design

In general protocols could be either text-protocols, or binary-protocols.

Text-Protocols: messages uses human readable characters

- Easy to understand.
- Easy to design and extend.
- Easy to test and debug.
- Could be complex to handle, the parsing code might become complex.
- Reverse engineering the protocol is very easy comparing to binary protocols.
- Require more bandwidth comparing to binary protocols.

Message Design

Binary Protocols: messages are designed using data structure instead of text

- Use less traffic, since the messages are smaller than text protocols
- Easy to handle.
- Not easy to test or debug.
- Data marshalling and unmarshalling.
- Reverse engineering is much harder.

Message Design

```
GET /somedir/page.html HTTP/1.1
Host: www.someschool.edu
Connection: close
User-agent: Mozilla/5.0
Accept-language: fr
```

HTTP Request in Text Format

```
struct HTTP_Reques{
        int requestType;
        int protocolVersion;
        char filePath[1024];
        char host[1024];
        char userAgent[1024];
        Long int langBitMask;
};
```

HTTP Request Binary Representation

Figure: Text Protocol vs Binary Protocol Message Design

Message Design

- The designing of the message (syntax, semantic) is very important aspect that will affect different quality attributes of your protocol (simplicity, efficiency, scalability, and extensibility)
- To design the message syntax is to define the message structure.
- To design the message semantic is to define the messages types and their purpose.

The Message Semantic

The messages could be categorized into two main categories:

- Command and Control Messages.
- Data Messages.

Under each category we could have as many messages as we need. Where each message has different semantic.

Command and Control Messages

- Command and control messages are used to define and control the parties behaviors during communication sessions.
- Command messages define the stage of the communication (e.g. handshake, authentication, data transfer, request, status, etc).
- The communication session could have multiple states or stages, the command messages enable switching between these stages.
- The control messages define the behaviours of the communication parties within the different stages.
- The control messages helps in negotiation, coordination, and shape the interaction between the parties (interruption, reset, cancellation, etc)

The Message Structure

In general it is good to have well-defined structure for your message that enable

- Extensibility.
- Separate application context data from communication data
- Few message structures as possible, and with same design pattern.

Usually the structure of the message contains two main sections, the message header and the message body.

HTTP Message Structure

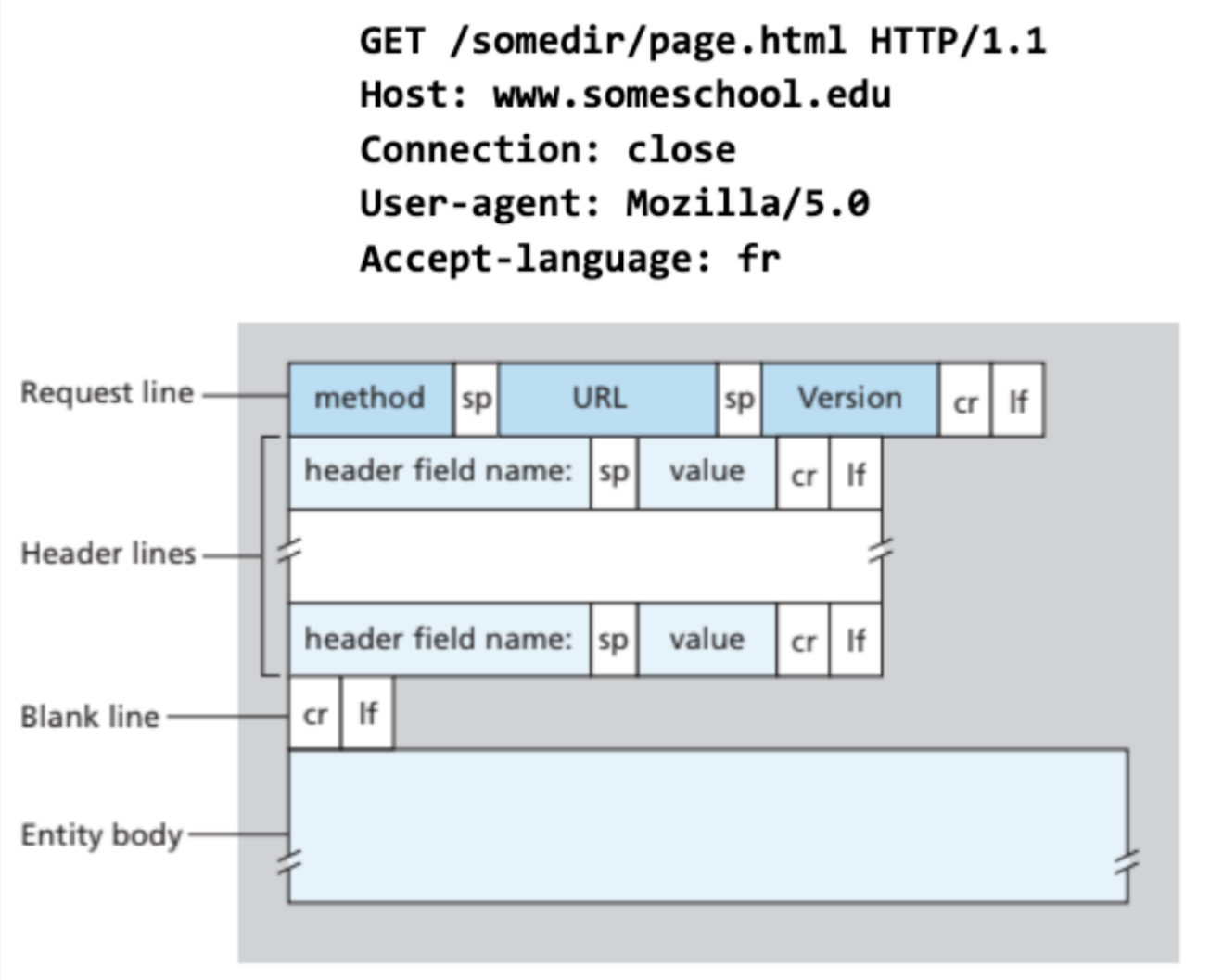


Figure: HTTP Request Message

HTTP Message Structure

```
HTTP/1.1 200 OK
Connection: close
Date: Tue, 18 Aug 2015 15:44:04 GMT
Server: Apache/2.2.3 (CentOS)
Last-Modified: Tue, 18 Aug 2015 15:11:03 GMT
Content-Length: 6821
Content-Type: text/html
(data data data data data ...)
```

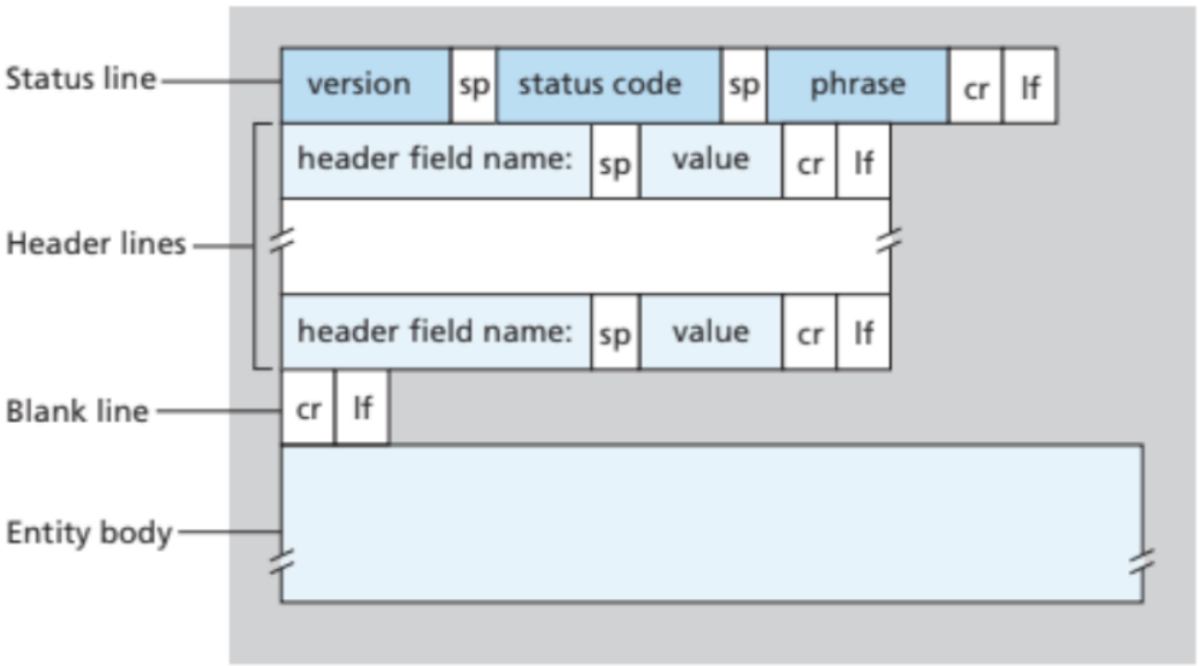


Figure: HTTP Response Message

DNS Message Structure

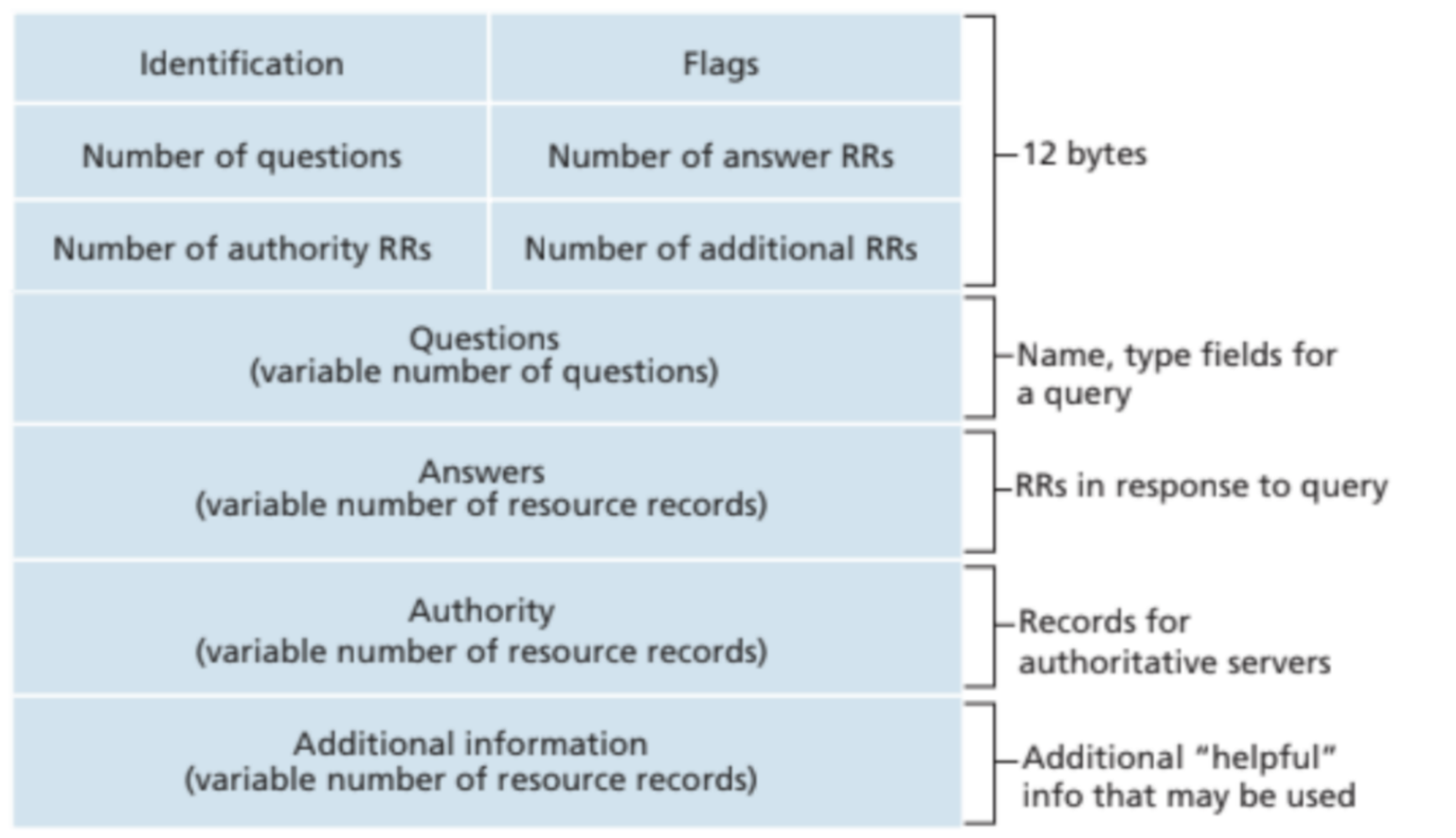


Figure: HTTP Response Message

Communication Rules

- Describe how the communication parties should interact.
- Describe the possible states and how the communication session move from one state to another state.
- It describe the allowed sequence of command and control messages and how they could
- change the state of the connection.
- It define possible communication scenarios between the communication parties.
- Define your protocols states and command sequences, use state and sequence diagram to model/check your communication rules.

SMTP Connection|Session

```
C: <client connects to service port 25>
C: HELO snark.thyrsus.com                  sending host identifies self
S: 250 OK Hello snark, glad to meet you     receiver acknowledges
C: MAIL FROM: <esr@thyrsus.com>            identify sending user
S: 250 <esr@thyrsus.com>... Sender ok      receiver acknowledges
C: RCPT TO: cor@cpmy.com                   identify target user
S: 250 root... Recipient ok                receiver acknowledges
C: DATA
S: 354 Enter mail, end with "." on a line by itself
C: Scratch called.  He wants to share
C: a room with us at Balticon.
C: .                                       end of multiline send
S: 250 WAA01865 Message accepted for delivery
C: QUIT                                    sender signs off
S: 221 cpmy.com closing connection         receiver disconnects
C: <client hangs up>
```

Figure: SMPT Connection between Client and server

What is the plan for next week?

- Transport Layer.
- Lab 2 (Investigating Transport Layer Protocol Behaviours)
- Quiz 2 (3%) cover this week topic "Application Layer"

The End (Questions?)

References I


James Forshaw, *Attacking network protocols: A hacker's guide to capture, analysis, and exploitation*, 1st ed., No Starch Press, USA, 2017.


John S. Gero and Thomas Mc Neill, *An approach to the analysis of design protocols*, Design Studies **19** (1998), no. 1, 21–61.


James F. Kurose and Keith W. Ross, *Computer networking: A top-down approach*, 7 ed., Pearson, Boston, MA, 2016.


M.Rose, *Beep: Building blocks for application protocols.*, 2001.


Rose and Malamud, *Rfc3117:on the design of application protocols*, 2001.


Andrew S. Tanenbaum and Maarten van Steen, *Distributed systems: Principles and paradigms*, 2 ed., Pearson Prentice Hall, Upper Saddle River, NJ, 2007.