CM2040: Databases, Networks and the Web

Summary

Arjun Muralidharan

6th September 2020

Contents

List of Figures

List of Algorithms

List of Listings

1 Three-tier web applications

Key Concepts

- ✓ Intro to module and lab
- ✓ Static vs. dynamic web applications
- ✓ 3-tier web applications architecture

Learning Outcomes

- ✓ Recognize the tools available for this module to edit a Node file and run it
- ✓ Describe what static and dynamic web applications are
- ✓ Describe what a 3-tier web application architecture is

1.1 Static vs. Dynamic Web Applications

A **web application** is a client-server software application in which the user interface runs in a web browser. It could be a computer program which allows a user to submit and retrieve data to or from a database over the Internet using their preferred browser. It is an application in which all or some parts of the software are downloaded from the web each time it is run.

Static web applications are static internet resources with little to no interaction with the user. **Dynamic web applications** have more user interactions, and users can input, change and manipulate data.

Differences between web applications and desktop applications are shown in Table 1.

Desktop	**Web**
Accessed through OS	Accessed through browser
Different appearance/experience in each OS	Consistent appearance/experience across platforms
Access to system resources; fast	Less access to system resources; slow
Lower risk of data loss	Higher risk of data loss
Different version for each OS	Same version across all platforms
Multiple updates required	Single update for all users

Table 1. *Desktop vs. Web Applications*

1.1.1 Lifetime of a web request

When a (static) web application is accessed, the following steps occur in order.

1. A **URL** is typed into the browser
2. Browser cache is checked; if the application is already available, the procedure skips to the last step
3. **DNS** lookup finds the **IP** address of the server
4. Browser initiates a **TCP** connection with the server
5. Browser sends an **HTTP** request to the server
6. Server handles the incoming request
7. Browser receives the **HTTP** response
8. Browser displays the **HTML** content

1.1.2 Accessing remote resources with HTTP

HTTP is a *client-server application protocol*. The client sends a request and a server provides a response.

HTTP requests have a fixed format, which consists of three parts: a **request line** , some optional message **headers** , and the **request body** (also optional). The request line is a formatted string, which consists of three parts; the HTTP method (e.g. `GET` or `POST`), the URL of the requested resource, and the protocol version. For example, the following request line:

```
GET /unbound/flashbks/computer/bushf.htm HTTP/1.1
```

A server responds with an HTTP response message, which is structured in three parts: a **status line** , a set of optional **headers** , and a **message body**.

The status line consists of the protocol version followed by a numeric status code and its associated message, e.g.

```
HTTP/1.1 404 Not found
```

HTTP is **stateless**. Each call is treated independently and there is no information maintained between two successive requests. HTTP cannot maintain a user session.

HTTP is **pull-based**. Only the client can initiate a request-response action, the server cannot contact the client.

Header Types There are four kinds of **headers**:

1. **General headers**: These apply to both the request and response, e.g. Date
2. **Request headers**: Specific to requests, e.g. Accept-Language, which sends which response languages are acceptable

3. **Response headers**: Specific to responses, e.g. WWW-Authenticate which returns if a user is authorized or not

4. **Entity headers**: Apply to the content of the request or response body, e.g. Content-Length specifies the length (in bytes) of the body

However, these notions are valid as of HTTP1.1 and may not be valid in HTTP/2 and newer versions.

1.2 Three-tier application architecture

The core principle of **three-tier architectures** is to provide an intermediate layer between the client and the data tier, which centralizes middleware services and the business logic of the application. Three-tier architectures offer a higher degree of scalability than two-tier con- figurations, thanks to better network utilization and to the virtually unlimited replication and load distribution capabilities of the middle tier. A comparison is shown in Figure 1.

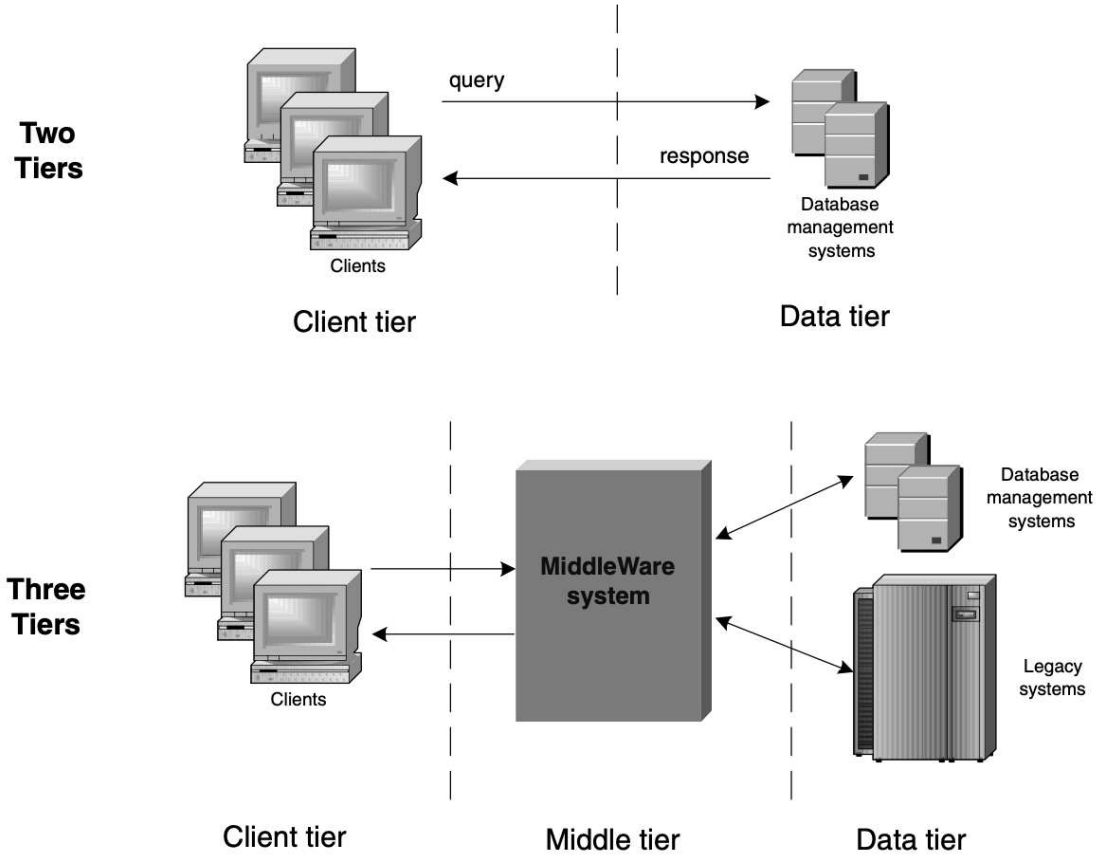

Figure 1. *Two-tier vs. three-tier architecture*

2 Building simple web servers

Key Concepts

- ✓ Web server and web hosting
- ✓ How to build a simple web server with Node
- ✓ Building an Express web server

Learning Outcomes

- ✓ Explain what a web server is
- ✓ How to build a simple web server with Node.js
- ✓ Building an Express web server

2.1 Web servers

A **web server** is a program that uses HTTP to serve the files that form web pages to users, in response to their requests, which are forwarded by their computers' HTTP clients.

A basic web server architecture includes:

- Hardware
- Operating system
- HTTP server
- Database *(optional)*
- Scripting Language *(optional)*

The **HTTP server** compiles the results from databases and scripting language, and controls what options are available to the end user via **configuration files** .

2.2 Node.js

Node.js is an asynchronous, event-driven Javascript runtime environment. It is not a programming language or development framework.

In an **asynchronous** environment, events can happen at different times, i.e. the timing of events doesn't matter. Node.js responds to any request immediately, as opposed to synchronous environments such as PHP, that handles one request at a time, and subsequent requests wait. The differences are shown in Table 2.

Node.js is suitable for **I/O-intensive applications** such as social media websites, but not suitable for **CPU-intensive applications** such as CAD software or games.

Node.js	PHP
Asynchronous	Synchronous
Single-threaded	Multi-threaded
Non-blocking	Blocking
Highly scalable	Less scalable
Memory-efficient	Less memory-efficient
Able to share code between browser and server	—

Table 2. *Node.js vs. PHP*

3 Generating web pages from data using templates

Key Concepts

- ✓ Routing in dynamic web applications
- ✓ Separation of Concern (SoC)
- ✓ Rendering html files and Templating

Learning Outcomes

- ✓ Describe and apply routing in web server development
- ✓ Describe and apply Separation of Concern principle of programming in your web application development
- ✓ Describe and apply templating in web application

3.1 Routes

Routes are URL schema, which describe the interfaces for making requests to your web app. Combining an HTTP request method (a.k.a. `HTTP verb`) and a path pattern, you define URLs in your app.

Each route has an associated route handler, which does the job of performing any action in the app and sending the HTTP response.

```
app.get("/about", (req, res) => res.send("<h1>This is the about page</h1>"));
```

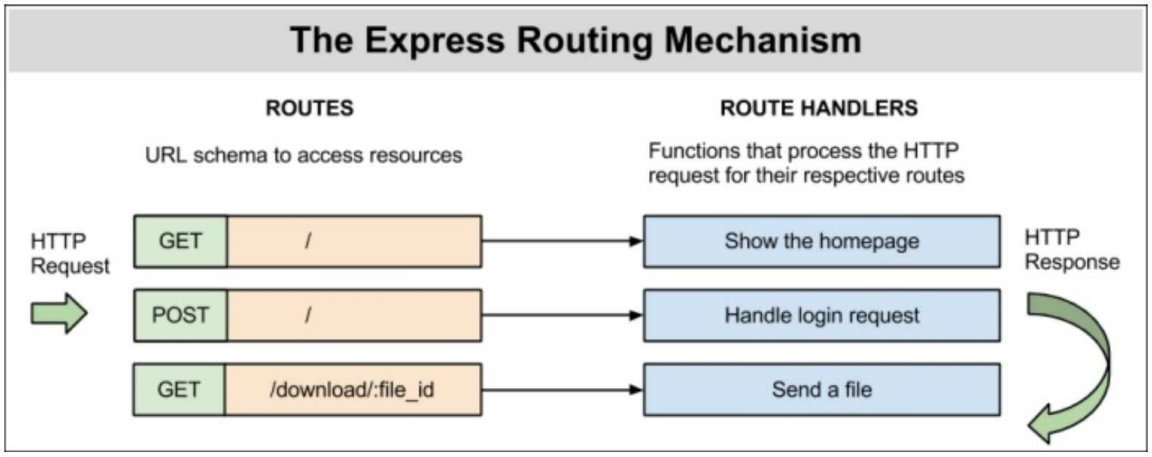

Figure 2. *Express Routing Mechanism*

3.2 Separation of Concerns

Separation of Concerns is a *design principle* for an application such that each module or layer in an application should only be responsible for one thing and should not contain code that deals with other things.

SoC is achieved by

1. constructing a modular application through encapsulation
2. layering functionality in the application, e.g. such as a three-tier architecture which is loosely coupled

The objective is to remove as many dependencies as possible between the individual parts.

Benefits of SoC include:

1. Reduces complexity
2. Helps **DRY** ("Don't repeat yourself")
3. Improves portability
4. Improves maintainability and testability

3.3 Templating

A **template engine** can help produce a dynamic HTML page by **inserting variables** into the final output, and **running programming logic** at run-time before sending the final HTML output to the browser.

This helps keep the logic in the HTML file at a minimum. Popular template engines include **EJS**, **Pug** and **Mustache**.

EJS template files use HTML and EJS tags:

<%= %>

used to output the return value of an expression into the document.

<% %>

used to evaluate an expression, but not add the return value to the document. An example:

```
<h1><%=title%></h1>
<ul>
        <%for (var i=0; i < supplies.length; i++) {%>
                <li>
                        <a href="supplies/<%=supplies[i]%>"></a>
                </li>
        <%}%>
</ul>
```

The parameters, such as `title`, can be set in a JavaScript file that define the routes. These are usually located in the *main.js* file in *routes*.

4 Handling forms to input data

Key Concepts

- ✓ Form handling and HTTP GET and POST methods
- ✓ HTTP GET method, form handling and collecting form-data
- ✓ HTTP POST method, form handling and collecting form-data

Learning Outcomes

- ✓ Describe and apply form handling in their web application
- ✓ Describe and apply GET and POST request methods in dynamic web application
- ✓ Describe and run the code to retrieve form data in middleware

4.1 GET and POST methods

The `GET` HTTP method is used to request a resource on a server, while `POST` sends data to a resource on the server.

`GET` may never manipulate any data. `POST` is commonly used with web forms to send data to a server or to retrieve sensitive data (by providing credentials).

4.2 GET

Routes for an HTTP `GET` method can be invoked with the following Express command. The method requests the resources from the server but can also pass form input fields via the URL. This is useful for simple data that doesn't need to be stored and is not sensitive.

```
app.get("/search-result", function (req, res) {
        //searching in the database
        res.send(
                "This is the keyword you entered: " +
                        req.query.keyword +
                        "<br />" +
                        "This is the result of the search:"
        );
});
```

5 Representing data in databases, relational databases

Key Concepts

- ✓ Introduction to databases
- ✓ Relational databases
- ✓ MySQL shell

Learning Outcomes

- ✓ Describe what a list of data is
- ✓ Describe modification problems related to lists of data
- ✓ Describe what a relational database is

5.1 Relational Databases

The **database approach** to managing and accessing data has **key characteristics**:

Self-describing nature A database system contains a complete definition of the database structure and constraints, stored in the **DBMS catalog** and called **meta data**. NOSQL systems are designed to not requires a catalog.

Insulation between programs and data, and data abstraction With a DBMS, the structure of the data can be extended or even changed **without programs having to change** (e.g., adding a

new attribute). Some databases provide an **object-oriented system** , allowing to define an interface describing the data names and types, and the implementation contains the actual operation, which can change without breaking the interface.

Support for multiple views of the data Data can be served in subsets tailored for specific users, and data can be **virtual data**, which is derived from the stored data but not actually stored (e.g. a calculated field).

Sharing of data and multiuser transaction processing A DBMS includes **concurrency control** software, ensuring multiple users can interact with the data at the same time. A *transaction* is a process with one or multiple database accesses. The **isolation** property ensures that a transaction appears to execute in isolation from other transactions. **Atomicity** ensures that either all database operations in a transaction are executed, or none are.

The DBMS approach further reduced **redundancy** by **normalizing data**, i.e. storing a logical data item in only a single place.

5.2 MySQL Shell

The MySQL shell can be accessed and quit with

```
# Invoke the mysql shell with with user root and ask for password
mysql -u root -p

# Exit the mysql shell
exit
```

with the following basic SQL commands available.

```
--Create a database names "myRestaurantMenu"
CREATE DATABASE myRestaurantMenu;

--Show all databases on the system
SHOW DATABASES;

--Switch to a specific database named "myBookShop"
USE myBookShop;

--Show the available tables
SHOW TABLES;
```

5.3 Creating Tables

SQL tables are created using the `CREATE TABLE` command specifying the data types and constraints for each field. The most common constraints to use are `NOT NULL` to specify that an attribute may not be `NULL`, `DEFAULT` i to specify a default value of i and `AUTO_INCREMENT` to specify integer values that automatically increment as rows are added.

```
CREATE TABLE dishes
(
id INT AUTO_INCREMENT,
name VARCHAR(50),
price DECIMAL(5, 2) unsigned,
is_vegetarian BOOLEAN,
is_vegan BOOLEAN,
PRIMARY KEY(id)
);
```

5.4 Inserting data

Inserting data is done using the `INSERT` command and specifying values with the `VALUES` keyword.

```
INSERT INTO
        dishes (name, price, is_vegetarian, is_vegan)
VALUES
        ('pizza margherita', 10.99, 1, 0),
        ('soya burger', 9.50, 1, 1);
```

5.5 Querying data

Basic SQL queries follow the `SELECT-FROM-WHERE` structure.

```
SELECT name, price
FROM dishes
WHERE price > 1;

--Using wildcards and limits
SELECT *
FROM dishes
LIMIT 2;
```

This structure is also used to perform deletion und updates of data.

```
--Delete rows
DELETE FROM
        dishes
WHERE
        price > 1;

--Update rows
UPDATE
        dishes
SET
        price = 2
WHERE
        price = 1;
```

6 Basic database operations, providing access to databases from middleware

Key Concepts

- ✓ Basic SQL, review create, select and insert into and introduce update and delete
- ✓ Primary keys
- ✓ Access to database from middleware

Learning Outcomes

- ✓ Describe and apply statements in SQL to do basic database operations.
- ✓ Describe and apply primary key and foreign keys in relational databases
- ✓ Recognize and apply access to databases from middleware code in Node to build dynamic web applications

6.1 Primary Keys

A primary key is a field or combination of fields in a table that **uniquely** identifies each row or record. It is declared when creating a table using the `PRIMARY KEY` directive.

Mostly, a primary key is a **numerical** field and is set to automatically increment (`AUTO_INCREMENT`).

6.1.1 Integrity Constraints

Table fields can be defined with constraints such as `UNIQUE`, `NOT NULL` and `DEFAULT` to support the structural integrity of the database. Primary keys have some built-in constraints:

- Every table must have a primary key field or combination of fields.
- This is equivalent to `UNIQUE` and `NOT NULL` constraints.

6.2 Passing variables to the back end

Data from an application can be passed to the back end and stored in the database by constructing an SQL query in code and using the appropriate function to invoke that query with the variable passed in. In Express, placeholder question marks (`()?`) are used in the SQL statement, which can then be filled up by a variable. This variable can be a single value or an array of values.

```
app.post("/bookadded", function (req,res) {

// saving data in database
let sqlquery = "INSERT INTO books (name, price) VALUES (?,?)";

// execute sql query
let newrecord = [req.body.name, req.body.price];
db.query(sqlquery, newrecord, (err, result) => {
        if (err) {
                return console.error(err.message);
                }
        else res.send(" This book is added to database, name: "
                                        + req.body.name
                                        + " price "
                                        + req.body.price);
                });
});
```

The input to the query can be sanitized using a sanitizer to clean up data and prevent malicious code injection.

7 Building a dynamic web application

Key Concepts

✓ Passing variables from middleware to front- end (templates)

✓ Passing variables from middleware to backend (database)

✓ Review what you have learned so far for development of a dynamic web application

Learning Outcomes

✓ Create a complete dynamic web application performing basic database operations

✓ Describe and apply passing variables from middleware to front-end (templates)

✓ Describe and apply passing variables from middleware to backend (database)

The web applications built with Express, Node and SQL Server are an example of **event-driven programming** as opposed to sequential programming. The flow of the program is determined based on events, such as user inputs or sensor outputs.

8 Database schema, ERD

Key Concepts

✓ Database schema

✓ Junction (bridge) tables

✓ SQL Join

Learning Outcomes

✓ Explain the context behind the relational model and describe the process of schema development

✓ Explain the context behind the junction (bridge) tables in relational model and describe the process of schema development

✓ Explain and apply foreign keys in relational databases

✓ Perform queries on databases using multiple related tables by JOIN queries

8.1 Database schema

To develop a **conceptual schema** of a database, we first identify the **entities**, **attributes**, and **data types** in the database.

Next, decide the **primary key** for each entity, **relationships** between entities and the respective **foreign keys** and, finally, identify the **relationship cardinalities** to proceed to refining the databse design.

Entity relationship diagrams are denoted using the notation shown in Figure 3.

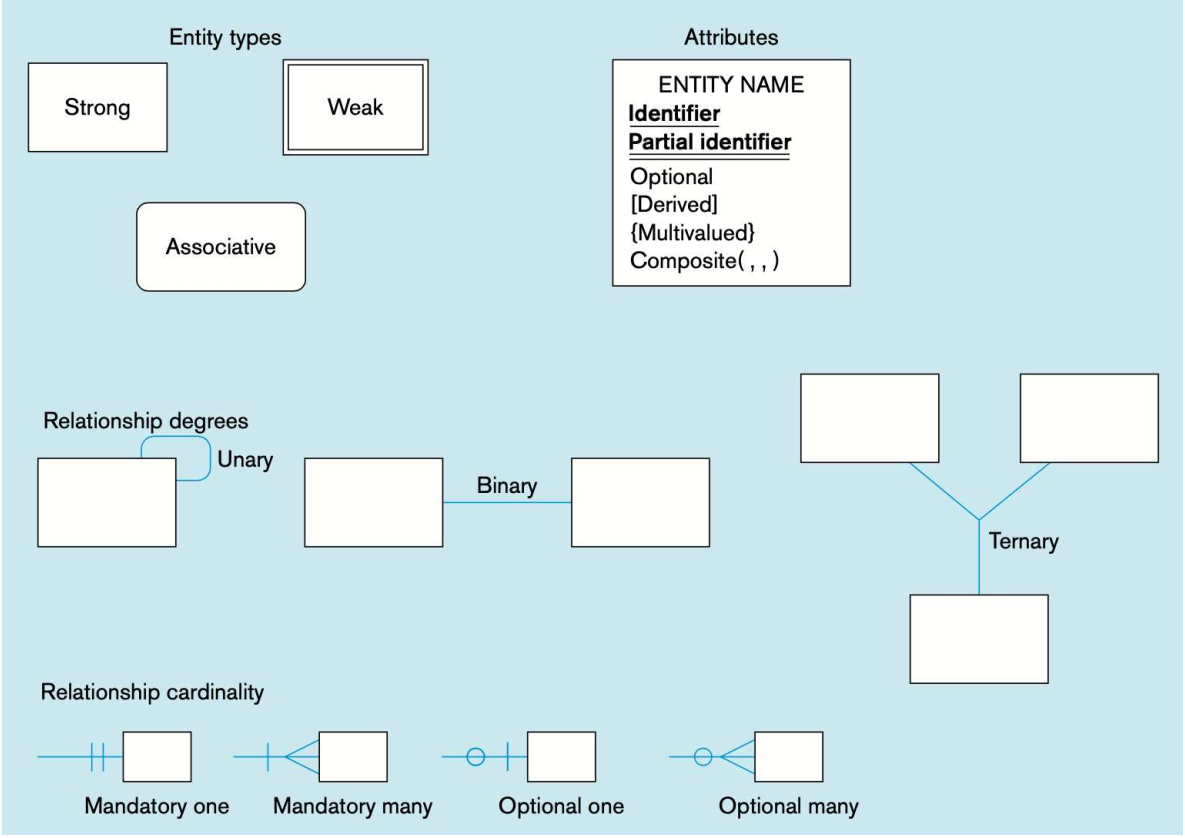

Figure 3. *ER Diagram Notation*

Alternative notations use the concept of **total relationships**, denoting that an entity's existence depends on the presence of one or more relationships, or **partial relationships**, stating that entities can exist with or without the relationship. These denoted with single or double lines between the entity and the relationship.

8.2 Foreign Keys and SQL Join

A **foreign key** is an attribute of an entity that is also the *primary key* of another entity that is related to the original entity.

Foreign keys are declared with the respective constraint and ensure the referential integrity of the data by **limiting the content of the foreign key** to the values of the referenced primary key.

To combine attributes from two tables, we can use a `JOIN` statement.

```
SELECT * FROM table1
JOIN table2
ON table1.attribute = table2.attribute
WHERE table1.anotherAttribute = criterion;
```

In the example above, the attribute from table 1 is the foreign key, while the attribute from table 2 is usually the primary key, but not restricted to it.

9 Querying a database (advanced)

Key Concepts

- ✓ Aggregate functions in SQL
- ✓ Left and right joins in SQL
- ✓ Nested select in SQL

Learning Outcomes

- ✓ Perform advanced queries on databases using aggregate functions
- ✓ Perform advanced queries on databases using multiple related tables by LEFT and RIGHT JOIN queries
- ✓ Perform advanced queries on databases using multiple related tables by nested SELECT queries

9.1 Aggregate Functions

Aggregate functions allow summarizing tuples of data into a single-tuple summary. In other words, aggregate functions query a database and summarize the data into a single row.

Common aggregate functions are `SUM`, `COUNT`, `MIN`, `MAX` and `AVG`, and they do exactly what one would expect them to do.

9.2 LEFT and RIGHT JOIN

A normal `JOIN` statement returns only records at the intersection $A \cap B$ of two tables A and B. This is also sometimes called `INNER JOIN`.

A `LEFT JOIN` returns all the records from A, completed with matching attributes from B. A `RIGHT JOIN` does the opposite.

9.3 Nested SQL Queries

The argument to a `WHERE` clause can itself be an SQL subquery returning a value for comparison.

```
SELECT name, price FROM food WHERE price=(SELECT MAX(price) FROM food)
```

10 Networking concepts

Key Concepts

- ✓ Basic concepts of computer networking
- ✓ TCP/IP model and network protocol

Learning Outcomes

- ✓ Explain and apply the basic concepts of computer networking
- ✓ Describe TCP/IP model and layers in the model
- ✓ Identify network protocols in each layer

10.1 Packet switching

In **packet switching** , unlike in circuit switching, each data unit to be transferred by the network doesn't know the entire path to the destination. It only knows the destination address.

Intermediate devices in the network locally decide the path and forward to the next known node that is closer to the destination.

10.2 The TCP/IP model

The **Transmission Control Protocol (TCP)/Internet Protocol (IP)** , which is an Internet protocol stack to perform communications between two computers, or hosts, through the Internet. It is a collection of different protocols. A *protocol* is a set of rules that controls the way data is transmitted between hosts.

The open systems interconnection reference model provides an abstraction of networking, as shown in Figure 4.

10.2.1 Application Layer

The **application layer** is the highest layer in the OSI stack and closest to the user. Example protocols are HTTP, FTP, SMTP, SSH, DNS, DHCP and Telnet.

This layer defines the type of messages exchanged, such as request/response, defines the fields in the messages and the meaning of fields.

SSL (older) and TLS (newer) are additional layers used to encrypt data at the application layer.

10.2.2 Transport Layer

The main protocol at the **transport layer** is **TCP**, which handles establishing of connections, checking their reliability, error checking and a three-way handshake process:

1. Client host sends `TCP SYN` segment to server.

2. Server host receives `SYN`, replies with `SYNACK`.

3. Client receives `SYNACK`, replies with `ACK`, which may also contain data.

An alternative to TCP is UDP, or **user datagram protocol** which does not provide error-checking or handshaking, making it suitable for high-speed applications such as live broadcasting, gaming or video conferencing. TCP is more suited to transfering documents and data reliably.

In addition to an **IP address**, a **port** defines which application receives specific data.

Multiplexing allows collecting multiple data segments from various ports and sending them out. **De-multiplexing** does the opposite, checking each received datagram and delivering it to the correct port.

10.2.3 Network Layer

The main protocol at the network layer is the **Internet Protocol** or **IP**. IPv4 addresses are 32-bit numbers, divided into four 8-bit sections ranging from 0 to 255.

IPv6 addresses are 128-bit numbers divided into eight 16-bit sections, each represented by 2 hexadecimal octets.

Routing is a mechanism carrying packers from source to destination, using the IP address, by a hop-by-hop process.

To prevent packets from being passed on infinitely, a TTL (time to live) value is initialised in the packet, and each time the packet is forwarded, the value is reduced by 1. When it reaches 0, the packet has lived too long and gets destroyed.

10.2.4 Link Layer

The link layer gets data across one hop, using protocols based on the medium at the physical layer, such as wire, Wi-Fi, fibre-optic and ethernet.

Frames use hardware-specific addressing such as media access control **MAC addresses**.

Layer			Protocol data unit (PDU)	Function[19]
Host layers	7	Application	Data	High-level APIs, including resource sharing, remote file access
	6	Presentation		Translation of data between a networking service and an application; including character encoding, data compression and encryption/decryption
	5	Session		Managing communication sessions, i.e., continuous exchange of information in the form of multiple back-and-forth transmissions between two nodes
	4	Transport	Segment, Datagram	Reliable transmission of data segments between points on a network, including segmentation, acknowledgement and multiplexing
Media layers	3	Network	Packet	Structuring and managing a multi-node network, including addressing, routing and traffic control
	2	Data link	Frame	Reliable transmission of data frames between two nodes connected by a physical layer
	1	Physical	Bit, Symbol	Transmission and reception of raw bit streams over a physical medium

Figure 4. *OSI reference model*