Network(chapter2)

prepared by : آلاء نصر الدين أبو بكر

يعني إيه network app؟

- هو أي **برنامج بيشتغل على أجهزة طرفية (End Systems)** وبيتواصل مع برامج تانية من خلال الشبكة.
- **أمثلة:**

- متصفح الإنترنت (Browser) بيتواصل مع السيرفر اللي عليه موقع ويب.
- تطبيق بريد إلكتروني بيتواصل مع سيرفر البريد زي **IMAP**

مش محتاج تكتب كود للأجهزة اللي جوه الشبكة!

- الأجهزة اللي بتكون **في قلب الشبكة (Network Core Devices)**، زي الروترات والسويتشات، **مش بتشغل تطبيقات المستخدم**.
- كل التطبيقات بتشتغل **على الأجهزة الطرفية** زي الكمبيوتر أو الموبايل، وده بيخلي **تطوير التطبيقات أسرع وأسهل**.

مثال لتوضيح الفكرة:

لو أنت بتستخدم متصفح زي **Google Chrome** علشان تدخل على موقع **YouTube**:

1. المتصفح (Chrome) هو التطبيق اللي شغال **على جهازك**.
2. السيرفر اللي عليه موقع **YouTube** هو التطبيق اللي بيشغل الفيديوهات وبيبعت البيانات لجهازك.
3. **الراوترات والسويتشات اللي في النص** وظيفتهم بس **ينقلوا البيانات بينك وبين السيرفر**، مش بيشغلوا المتصفح ولا الفيديوهات.

ليه ده مهم؟

- لأنك **كمطور تطبيقات شبكة، مش هتكتب كود يشتغل على الراوتر أو السويتش**، انت بتكتب كود للتطبيقات اللي بتشتغل على الأجهزة الطرفية زي الموبايل أو الكمبيوتر .

خلاصة Slide 6 :

1. تطبيقات الشبكة بتشتغل على الأجهزة الطرفية، مش على أجهزة الشبكة الداخلية.
2. الطبقة العليا (**Application Layer**) هي المسؤولة عن التطبيقات اللي بنتعامل معاها.
3. النظام **Client-Server** هو الأكثر شيوعًا، حيث السيرفر دائم التشغيل والعميل بيتواصل معاه وقت الحاجة.
4. فيه بروتوكولات مشهورة زي **HTTP, IMAP, FTP** بتستخدم النموذج ده.

- **HTTP:** لتحميل صفحات الويب.
- **IMAP:** لاستقبال الإيميلات.
- **FTP:** لتحميل الملفات.

التواصل بين العمليات (Processes Communicating)

العمليات (Processes) هي **برامج شغالة جوه الجهاز (Host)**، ولو عاوزين يتواصلوا مع بعض، بيكون عندهم طريقتين:

1 التواصل داخل نفس الجهاز (Inter-Process Communication - IPC)

- لو فيه **عمليتين** شغالين على **نفس الجهاز**، بيتواصلوا باستخدام **نظام التشغيل (OS)** اللي بيحدد إزاي البيانات تتحرك بينهم.
- **أمثلة:**

- **Shared Memory:** بيكون فيه جزء من الذاكرة المشتركة بين العمليتين.
- **Message Passing:** العمليتين بيتبادلوا رسائل مباشرة.

2 التواصل بين جهازين مختلفين (Processes on Different Hosts)

- لما تكون العمليتين **على أجهزة مختلفة**، بيتبادلوا **الرسائل عبر الشبكة**.
- العملية اللي تبدأ الاتصال اسمها **Client Process**.
- العملية اللي تستنى الاتصال اسمها **Server Process**.
- في بعض التطبيقات **(P2P - Peer-to-Peer)**، العملية ممكن تكون **عميل وسيرفر في نفس الوقت**

بوابة التواصل بين العمليات – (المنافذ) Sockets

السوكيت (Socket) هو **واجهة (Interface)** بتستخدمها العملية علشان تبعت أو تستقبل رسائل.

🔹 إزاي السوكيت بيشتغل؟

- تخيل إن **السوكيت زي الباب** اللي العملية بتبعت منه البيانات:
  1. **العملية المرسلة (Sending Process)** بتبعت الرسالة من خلال السوكيت الخاص بيها.
  2. **البنية التحتية للنقل (Transport Infrastructure)** في الشبكة بتنقل البيانات للطرف التاني.
  3. **العملية المستقبلة (Receiving Process)** بتستلم الرسالة عند السوكيت الخاص بيها.

🔹 لازم يكون فيه Socket على كل طرف:

✅ **في الجهاز اللي بيبعت البيانات Socket.**
✅ **في الجهاز اللي بيستقبل البيانات Socket.**

📌 (Addressing Processes)

كيف تستقبل العملية الرسائل؟

- أي **عملية (Process)** لازم يكون عندها **معرّف (Identifier)** علشان تقدر تستقبل الرسائل.
- الجهاز المضيف (Host) عنده **عنوان IP مميز (32-bit IP Address)**، لكنه **مش كفاية** لوحده لتحديد العملية!

❓ هل عنوان IP وحده كافي لتحديد العملية؟

❌ **لا!** لأن الجهاز الواحد ممكن يكون عليه **أكتر من عملية** شغالة في نفس الوقت.
✅ الحل: لازم نستخدم **معرّف مزدوج (IP Address + Port Number)**

🔹 ما هو رقم المنفذ (Port Number)؟

هو **رقم يحدد العملية** اللي المفروض تستلم الرسالة على الجهاز.
port number for **HTTP Server** : 80 , **Mail Server - SMTP** : 25

مثال :

لو عاوز تبعت رسالة **HTTP** لسيرفر **gaia.cs.umass.edu**:

- **عنوان الـ IP** للسيرفر: `128.119.245.12`
- **رقم المنفذ**: `80` (لأنها خدمة ويب HTTP)

📌 (Application-Layer Protocols)

إيه اللي بيحدد البروتوكول بتاع أي تطبيق؟

أنواع الرسائل اللي بتتبادلها العمليات:

- زي Request/Response في HTTP.
- هيكلة الرسائل (Message Syntax): إيه هي البيانات اللي بتكون في الرسالة وإزاي بتترتب ؟
- معنى البيانات (Message Semantics): كل حقل في الرسالة معناه إيه ؟
- القواعد اللي بتحكم إرسال واستقبال الرسائل: إمتى العملية تبعت رسالة؟ وإمتى ترد عليها ؟

Types of protocols

(Open Protocols): أي حد يقدر يستخدمها لأنها موثقة في RFCs.,
- **HTTP** (لتحميل صفحات الويب).
- **SMTP** (لإرسال الإيميلات).
(Proprietary Protocols):- **مش متاحة للعامة، ومملوكة لشركة معينة.**

- **Skype
- Zoom

التطبيقات بتحتاج خدمات نقل مختلفة زي الدقة، السرعة، التوقيت، والأمان. , (data integrity , timing , throughput , security)

🚀(Internet Transport Protocols Services)

🔹 TCP (Transmission Control Protocol)

✅ مميزات TCP:

1. **نقل موثوق (Reliable Transport):**
   - يضمن وصول البيانات بشكل **كامل وبالترتيب الصحيح**.
2. **التحكم في التدفق (Flow Control):**
   - يمنع المرسل من **إغراق** المستقبل ببيانات أكتر من قدرته على المعالجة.
3. **التحكم في الازدحام (Congestion Control):**
   - بيبطّئ المرسل **لما الشبكة تكون مزدحمة** لتقليل الضغط.
4. **اتصال موجه (Connection-Oriented):**
   - لازم يكون فيه (Connection) بين العميل والسيرفر قبل تبادل البيانات.

❌ حاجات مش موجودة في TCP:

- ❌ **ما بيضمنش التوقيت (Timing Guarantee).**
- ❌ **ما بيوفرش حد أدنى للسرعة (Minimum Throughput).**
- ❌ **ما بيوفرش أمان (Security) بشكل افتراضي.**

🔹 UDP (User Datagram Protocol)

✅ مميزات UDP:

1. **سريع وخفيف (Lightweight & Fast):**
   - **بيبعث البيانات فورًا بدون ما يستنى تأكيد الاستلام.**
2. **مناسب للتطبيقات الحقيقية (Real-Time Apps):**
   - زي **المكالمات الصوتية (VoIP)** والبث المباشر، حيث السرعة أهم من التأكيد.

❌ إيه اللي UDP مش بيضمنه؟

- ❌ **ما فيش ضمان لوصول البيانات (Unreliable).**
- ❌ **ما فيش تحكم في التدفق (No Flow Control).**

- ❌ **ما فيش تحكم في الازدحام (No Congestion Control).**
- ❌ **ما فيش اتصال مسبق (No Connection Setup).**
- ❌ *ما فيش أمان مدمج (No Security).*

❓ ليه نستخدم UDP بدل TCP؟**

- 🔹 **لأن بعض التطبيقات محتاجة سرعة أكتر من الدقة!**

✅ في تطبيقات زي **الألعاب، البث المباشر، والمكالمات الصوتية**، لو **حزمة بيانات ضاعت، مش مشكلة**، المهم إن باقي البيانات توصل بسرعة.

✅ **TCP** بينما **بطيء** لأنه بيعمل إعادة إرسال للبيانات المفقودة، **بيسمح للبيانات تمشي بأي طريقة أسرع UDP.**

🔒 تأمين بروتوكول TCP (Securing TCP)

🚨 المشكلة في TCP و UDP العاديين (Vanilla TCP & UDP Sockets):

- ❌ **لا يوجد تشفير (No Encryption):** البيانات بتتبعت **كـ نص واضح (Cleartext)**.
- ❌ **كلمات المرور مكشوفة (Passwords in Cleartext):**
  - لما تبعت كلمة سر عبر **TCP أو UDP العادي**، أي حد على الشبكة يقدر يشوفها!
  - يعني **هاكر ممكن يعترض البيانات ويسرق المعلومات.**

🔐 الحل: استخدام TLS (Transport Layer Security)

✅ **TLS بيضيف أمان على TCP** عن طريق:

تشفير البيانات (Encryption):

- يحمي البيانات بحيث حتى لو حد اعتراضها، مش هيقدر يفهمها.

سلامة البيانات (Data Integrity):

- يضمن إن البيانات **ما تمش التلاعب بيها أثناء النقل.**

التحقق من الهوية (End-Point Authentication):

- يضمن إنك **بتتواصل مع السيرفر الصح** مش مع موقع مزيف.

📌 HTTP و Web

- صفحة الويب تتكون من **عناصر (Objects)**، وكل عنصر ممكن يكون **مخزن في سيرفر مختلف.**
- العناصر ممكن تكون **ملفات HTML، صور JPEG، تطبيقات Java، ملفات صوتية... إلخ.**
- الصفحة تتكون من **ملف HTML أساسي (Base HTML-file)** ويحتوي على **عدة عناصر مرتبطة (Referenced Objects).**

- كل عنصر له **عنوان URL** يحدد مكانه، مثل:

```
www.someschool.edu/someDept/pic.gif
```

- **Host Name:** اسم السيرفر اللي بيستضيف الصفحة (مثلاً someschool.edu).
- **Path Name:** المسار إلى العنصر داخل السيرفر (مثلاً someDept/pic.gif).

🔹 (HTTP Overview)

- **HTTP (Hypertext Transfer Protocol): بروتوكول طبقة التطبيق (Application-Layer Protocol)** هو المستخدم في الويب.
- **نموذج العميل/الخادم (Client/Server Model):**
  - **العميل (Client):** المتصفح (مثل Firefox أو Safari) اللي بيطلب ويستلم ويعرض بيانات الويب.
  - **الخادم (Server):** السيرفر اللي بيخزن ويرسل بيانات الويب استجابة لطلبات المتصفح.
  - العميل بيبعت **HTTP Request** إلى السيرفر، والسيرفر بيرد بـ **HTTP Response** يحتوي على البيانات المطلوبة.

🔹 كيفية عمل HTTP مع TCP

- **HTTP بيستخدم TCP** لنقل البيانات:

1. العميل (المتصفح) **ينشئ اتصال TCP مع السيرفر** على port **80**.
2. السيرفر يقبل اتصال **TCP** من العميل.
3. يتم تبادل **رسائل HTTP** (الطلبات والاستجابات).
4. يتم **إغلاق اتصال TCP** بعد الانتهاء من نقل البيانات.

🔹 (Stateless) بروتوكول HTTP بدون حالة

- **السيرفر لا يحتفظ بأي بيانات عن العميل بعد انتهاء الطلب.**
- كل طلب **يُعامل كأنه مستقل** ولا يعتمد على الطلبات السابقة.
- **البروتوكولات التي تحتفظ بحالة العميل تكون معقدة،** لأنها تحتاج إلى:
  - تخزين **تاريخ الجلسات والطلبات السابقة**.
  - التعامل مع **عدم التوافق إذا تعطل السيرفر أو العميل**.

HTTP connections two types :

1- Non-persistent HTTP (الاتصال غير المستمر)

- يفتح اتصال TCP جديد لكل طلب.

- يتم إرسال **عنصر واحد فقط** في كل اتصال.
- بعد إرسال العنصر، يتم **إغلاق اتصال TCP**.
- لتحميل **عدة عناصر (مثل الصور والملفات المرتبطة)**، يجب فتح اتصالات TCP متعددة.

2- Persistent HTTP (الاتصال المستمر)

- يتم **فتح اتصال TCP واحد** بين العميل والسيرفر.
- يتم إرسال **عدة عناصر عبر نفس الاتصال** دون الحاجة إلى فتح اتصال جديد لكل عنصر.
- يتم **إغلاق اتصال TCP** بعد الانتهاء من نقل البيانات المطلوبة.

🔹 وقت استجابة Non-persistent HTTP

📌 **RTT (Round Trip Time):** هو الزمن الذي يستغرقه **باكيت صغيرة** للانتقال من العميل إلى السيرفر والعودة.

📌 **وقت استجابة HTTP لكل عنصر:**

1. **RTT واحد** لفتح اتصال TCP.
2. **RTT آخر** لإرسال طلب HTTP واستلام أول بايت من الاستجابة.
3. **زمن نقل الملف/العنصر نفسه**.

Non-persistent HTTP response time = 2RTT+ file transmission time

🔹 الحل: Persistent HTTP (HTTP 1.1) 🏆

📌 **مميزات Persistent HTTP:**

✅ **السيرفر يُبقي الاتصال مفتوحًا** بعد إرسال الاستجابة.

✅ **الطلبات التالية تُرسل عبر نفس الاتصال** بدلاً من فتح وإغلاق اتصالات جديدة.

✅ **تقليل وقت الاستجابة** إلى **RTT واحد فقط لكل العناصر المرتبطة**، مما يحسن السرعة بشكل كبير.

💡 **النتيجة:**

وقت الاستجابة ينخفض إلى النصف تقريبًا !

📌 **مكونات طلب HTTP:**

- **يُكتب بصيغة ASCII** ليسهل قراءته.
- **يتضمن سطر الطلب (GET, POST, HEAD)** لتحديد نوع الطلب.
- **يحتوي على فاصل carriage return + line feed** لتحديد نهاية الهيدر .

🔹 تنسيق عام لرسالة طلب HTTP (HTTP Request Message) 📨

📌 تتكون رسالة الطلب من:

1 **سطر الطلب (Request Line)**: يحتوي على:

- **طريقة الطلب (Method)** مثل `GET` أو `POST` .
- **عنوان URL** المطلوب.
- **إصدار HTTP** المستخدم.

2 **رؤوس الطلب (Header Lines)**: تحتوي على معلومات إضافية مثل نوع المحتوى (Content-Type) والطول (Content-Length).

3 **الجسم (Body)**: يحتوي على بيانات الطلب (يُستخدم في `POST` و `PUT`).

🔹 (HTTP Methods) 🛠️

📌 GET Method:

✅ تُستخدم لجلب البيانات من السيرفر.
✅ البيانات تُرسل ضمن **عنوان URL** بعد `?` .
✅ لا تُستخدم لإرسال بيانات حساسة لأنها **ظاهرة في الـ URL**.

📌 POST Method:

✅ تُستخدم لإرسال بيانات إلى السيرفر)مثل بيانات النماذج Forms).
✅ البيانات تُرسل في **الجسم (Body)** وليس في الـ URL، مما يجعلها أكثر أمانًا.

📌 HEAD Method:

✅ تُستخدم لجلب **الرؤوس فقط** دون استرجاع المحتوى الفعلي.
✅ تُفيد في معرفة ما إذا كان المورد متاحًا أو حجم المحتوى بدون تحميله.

📌 PUT Method:

✅ تُستخدم لتحميل (رفع) ملف جديد أو استبدال ملف موجود على السيرفر.
✅ البيانات تُرسل في **الجسم (Body)** تمامًا مثل `POST` .

🔹 رسالة استجابة (HTTP Response Message)

📌 **تحتوي على:**

1 **سطر الحالة (Status Line):**

- يحتوي على **إصدار HTTP + رمز الحالة (Status Code) + الوصف**
- مثال:

```
HTTP/1.1 200 OK
```

2 **رؤوس الاستجابة (Header Lines):**

- تحتوي على معلومات مثل **نوع المحتوى، تاريخ الإنشاء، الكوكيز، والتخزين المؤقت**

3 **جسم الاستجابة (Body):**

- يحتوي على **البيانات المطلوبة** (HTML، JSON، صور...)

📌 مفهوم الـ Cookies ودورها في الحفاظ على حالة المستخدم 🔄🍪

🔹 HTTP هو بروتوكول عديم الحالة (Stateless Protocol)

- لا يوجد مفهوم **للتفاعل متعدد الخطوات** في HTTP.
- كل **طلب HTTP مستقل تمامًا** عن الآخر.
- السيرفر والعميل لا يحتاجان إلى **تتبع حالة المستخدم** بين الطلبات.

🔹 كيف تحافظ الـ Cookies على حالة المستخدم؟

المواقع الإلكترونية تستخدم **ملفات تعريف الارتباط (Cookies)** لتذكر حالة المستخدم بين الطلبات المختلفة.

📌 **أجزاء الـ Cookie الأساسية:**

1 **سطر رأس (Header) في استجابة HTTP** يحتوي على `Set-Cookie` لإرسال الكوكيز إلى العميل.
2 **سطر رأس (Header) في الطلبات المستقبلية** من العميل يحتوي على `<Cookie: <value>` ليتم إرسالها للسيرفر.
3 **ملف الكوكيز (Cookie File)** المخزن على جهاز المستخدم بواسطة المتصفح.
4 **قاعدة بيانات في السيرفر** تحتفظ بمعلومات المستخدم وربطها بمعرّف الكوكيز.

🔹 كيف تعمل الكوكيز مع السيرفر؟

📌 **التفاعل بين العميل والسيرفر باستخدام الكوكيز:**

1 **العميل يرسل طلب HTTP عادي.**
2 **السيرفر يرد بإضافة** `Set-Cookie` **في الاستجابة (مثلاً:** `Set-Cookie: 1678` **).**
3 **المتصفح يخزن الكوكيز في ملف محلي.**
4 **في الطلبات اللاحقة، المتصفح يرسل الكوكيز تلقائيًا (** `Cookie: 1678` **).**
5 **السيرفر يقرأ الكوكيز، يتعرف على المستخدم، ثم ينفذ الإجراءات المناسبة بناءً على حالته.**

📌 **بعد أسبوع:**

- يزور العميل الموقع مجددًا، ويرسل المتصفح نفس الكوكيز (`Cookie: 1678`).
- السيرفر يتحقق من المعرّف في قاعدة البيانات **ويسترجع معلومات المستخدم دون الحاجة لإعادة تسجيل الدخول

- ### 🔹 ما هي استخدامات الـ Cookies؟

الكوكيز تلعب دورًا هامًا في تحسين تجربة المستخدم، ومن أبرز استخداماتها:

✅ **التوثيق (Authorization):**

- تُستخدم لتذكر حالة تسجيل الدخول للمستخدمين.

✅ **عربات التسوق (Shopping Carts):**

- تساعد في حفظ المنتجات المختارة عند التنقل بين الصفحات.

✅ **التوصيات (Recommendations):**

- تعتمد المواقع على الكوكيز لتقديم اقتراحات مخصصة بناءً على نشاط المستخدم.

✅ **حالة جلسة المستخدم (User Session State):**

- مثل خدمات البريد الإلكتروني التي تتطلب الاحتفاظ بجلسة المستخدم بعد تسجيل الدخول

كيف يمكن الحفاظ على حالة المستخدم؟

🔹 **هناك طريقتان رئيسيتان:**

1 **عبر نقاط نهاية البروتوكول (Protocol Endpoints):**
(مثل **السيرفر (Server)** والمتصفح أو التطبيق **(Client)**.)

- يتم الحفاظ على حالة المستخدم في **السيرفر والعميل** عبر عدة معاملات.

مثال: عندما تسجل الدخول إلى موقع ويب، يتم **حفظ حالة الجلسة (Session State) في السيرفر**، وكل مرة تطلب فيها صفحة جديدة، يقوم السيرفر **بمطابقة الجلسة مع بياناتك المسجلة مسبقًا**. 2 **عبر الرسائل (In Messages):**

- يتم تضمين الكوكيز في **رسائل HTTP** لحمل بيانات الحالة.

الطريقة	كيف تُحفظ البيانات؟	مثال
نقاط نهاية البروتوكول	يتم الاحتفاظ بالبيانات داخل السيرفر أو العميل ويتم تحديثها باستمرار	تسجيل الدخول إلى موقع ويب دون الحاجة لحفظ بيانات في جهاز المستخدم
الكوكيز	يتم تخزين البيانات في ملف صغير على جهاز المستخدم وإرسالها مع كل طلب	تذكر بيانات تسجيل الدخول تلقائيًا عند العودة إلى الموقع

💡 متى تُستخدم كل طريقة؟

🔹 **استخدام Protocol Endpoints** يكون أكثر أمانًا لأنه لا يعتمد على تخزين البيانات في جهاز المستخدم.
🔹 **استخدام الكوكيز** يكون مناسبًا للحالات التي تتطلب تذكر المستخدم عبر زيارات متفرقة للموقع.

📌 تتبع سلوك المستخدم باستخدام الكوكيز

الكوكيز يمكن استخدامها لتسجيل وتحليل **سلوك المستخدم أثناء تصفح المواقع**، ويوجد نوعان رئيسيان:

1 كوكيز الطرف الأول (First Party Cookies):

- تستخدمها المواقع لتتبع نشاط المستخدم داخل نفس الموقع.
- مثال: عند دخولك إلى موقع تسوق، يتم تخزين بيانات المنتجات التي شاهدتها لتحسين التوصيات لاحقًا.

2 كوكيز الطرف الثالث (Third Party Cookies):

- تُستخدم لتتبع المستخدم عبر **مواقع مختلفة**، غالبًا بواسطة شبكات الإعلانات أو أدوات التحليل.
- يتم تحميلها من مواقع خارجية لم يزرها المستخدم مباشرة، مما يجعل التتبع غير مرئي له.
- مثال: إذا بحثت عن منتج معين على موقع، ثم رأيت إعلانات عنه على موقع آخر، فهذا غالبًا بسبب كوكيز الطرف الثالث.

🔹 هل التتبع مرئي للمستخدم؟ ليس دائمًا !

- يمكن أن يتم التتبع بدون عرض إعلان واضح، عن طريق **روابط غير مرئية** أو طلبات HTTP GET ترسل معلومات المستخدم إلى الخادم .

🔹 الحماية من تتبع الطرف الثالث؟

- **تم تعطيل كوكيز الطرف الثالث افتراضيًا في متصفحي Firefox وSafari.**
- **كان من المقرر تعطيلها في Google Chrome في 2023**

📌 اللائحة العامة لحماية البيانات (GDPR) والكوكيز

✅ الكوكيز التي يمكنها تحديد هوية المستخدم تعتبر **بيانات شخصية** وتخضع لقوانين حماية البيانات مثل **GDPR (General Data Protection Regulation)**.

📌 **وفقًا لـ GDPR، يُعتبر أي معرف عبر الإنترنت مثل:**

- عناوين الـ IP
- معرفات الكوكيز
- أي بيانات يمكن استخدامها لتحديد المستخدم

كبيانات شخصية، مما يعني أنها تخضع للقوانين التي تتطلب موافقة المستخدم على جمعها ومعالجتها.

✅ **لذلك، معظم المواقع الآن تعرض نافذة تطلب منك الموافقة على استخدام الكوكيز قبل تخزينها في جهازك.**

Web Caching (Proxy Servers)

✅ **تلبية طلبات العميل دون الحاجة إلى التواصل مع الخادم الأصلي.**

🔹 كيف يعمل التخزين المؤقت للويب؟

1 يقوم المستخدم بإعداد المتصفح لاستخدام **ذاكرة التخزين المؤقت (Web Cache)**.
2 المتصفح يرسل جميع طلبات HTTP إلى ذاكرة التخزين المؤقت.
3 **إذا كان العنصر موجودًا في ذاكرة التخزين المؤقت** → يتم إرجاعه مباشرةً إلى العميل
4 **إذا لم يكن العنصر موجودًا** → يتم جلبه من الخادم الأصلي، ثم تخزينه في الكاش، وإعادته للعميل

- **تتصرف ك server** عند تلبية الطلبات من الكاش.
- **تتصرف ك client** عند طلب المحتوى من الخادم الأصلي.

✅ يستخدم الخادم **رأس استجابة (Response Header)** لتحديد سياسة التخزين المؤقت:

- `Cache-Control: max-age=<seconds>` → يحدد المدة التي يمكن فيها تخزين العنصر مؤقتًا.
- `Cache-Control: no-cache` → يمنع التخزين المؤقت للعنصر.

في هذه الحالة، يتم تخزين الاستجابة مؤقتًا، لكن كل مرة يحتاج فيها العميل إلى هذا المحتوى، يجب أن يطلب تأكيدًا من الخادم

📌 لماذا نستخدم التخزين المؤقت للويب؟

✅ **تقليل زمن الاستجابة** للمستخدم، لأن ذاكرة التخزين المؤقت **أقرب إليه من الخادم الأصلي**.
✅ **تقليل الحمل على الشبكة** داخل المؤسسات والشركات.
✅ **تحسين توزيع المحتوى**، مما يساعد مزودي المحتوى الضعفاء على تقديم محتواهم بشكل أكثر كفاءة.

📌 كيف يساعد الكاش مزودي المحتوى الضعفاء؟

- لما يكون عندك **موقع بسيط** بإمكانيات محدودة، ممكن يحصل ضغط كبير على السيرفر لو عدد الزوار زاد.
- لكن لو المحتوى تم تخزينه في **كاشات قريبة من المستخدمين** (زي CDNs أو بروكسي سيرفرات)، المستخدم هيقدر يحمل المحتوى بسرعة **بدون الحاجة للوصول للسيرفر الأصلي كل مرة**.
- ده بيقلل الضغط على السيرفر الأساسي، وبيحسن أداء الموقع حتى لو موارده ضعيفة.

💡 مثال عملي:

- موقع عنده صور ومنشورات ثابتة، بدل ما كل طلب يروح للسيرفر الأساسي، الصور دي **تتخزن في كاشات** قريبة من المستخدمين.
- كده حتى لو السيرفر الأصلي ضعيف، المستخدم يقدر يشوف الصور بسرعة بدون الحاجة للاتصال بالسيرفر كل مرة.

Browser caching : Conditional GET

Goal: don’t send object if browser has up-to-date cached version

- الفكرة الأساسية: لو المتصفح عنده نسخة مخزنة (cached) من الصفحة، مش لازم ينزلها تاني إلا لو تغيرت.
- ده بيتم باستخدام **HTTP Conditional GET**، اللي بيستخدم الهيدر `If-Modified-Since` في الطلبات.

💡 كيف يعمل؟

1 **المتصفح (client)** يرسل طلب **GET** إلى السيرفر، ويضيف فيه هيدر `If-Modified-Since: <date>`، اللي فيه آخر تاريخ عنده للنسخة المخزنة.
2 **السيرفر (server)** يراجع إذا كان المحتوى **تم تعديله بعد هذا التاريخ أم لا**:

- **لو المحتوى لم يتغير** → السيرفر يرد بكود **304 Not Modified** بدون إرسال البيانات، والمتصفح يستخدم النسخة القديمة من الكاش.
- **لو المحتوى تغير** → السيرفر يرد بكود **200 OK** ويرسل النسخة الجديدة للمتصفح.

📌(HTTP/2) Key goal : decreased delay in multi-object HTTP requests

🔹 **المشكلة في HTTP/1.1:**

- كان بيعتمد على **pipelined GETs** في نفس الاتصال.
- السيرفر كان بيرد **بالترتيب** (FCFS - First Come, First Served).
- المشكلة إن لو فيه **ملف كبير**، هيمنع الملفات الأصغر من الوصول بسرعة (Head-of-Line Blocking).

🔹 **الحل في HTTP/2:**

- HTTP/2 قدم **مرونة أكبر** في إرسال البيانات.

- بيتم إرسال الملفات بناءً على priority، مش بالضرورة بالترتيب.

- السيرفر يقدر **يدفع (push) بيانات غير مطلوبة مسبقًا** للعميل لتحسين السرعة.
- تقسيم الملفات إلى **Frames** وجدولتها بشكل ذكي، لتجنب تأخير الملفات الصغيرة بسبب الملفات الكبيرة.

🔹 **HTTP/2 Mitigating HOL Blocking:**

- بدل ما الملفات تنتظر بعضها، HTTP/2 بيجزّأ البيانات ويبعثها **بالتوازي**.
- الملفات الصغيرة تقدر **تتسلم قبل انتهاء تحميل الملف الكبير**، مما يحسن الأداء بشكل كبير.

🚀 **النتيجة:** HTTP/2 أسرع وأكثر كفاءة، وبيحل مشكلة **تأخير الصف الأول (HOL Blocking)** اللي كانت في HTTP/1.1.

HTTP/2 to HTTP/3

🔹 مشاكل HTTP/2:

- بيعتمد على **اتصال TCP واحد** لكل الجلسة، لكن لو حصل فقدان في البيانات **كل النقل بيتعطل** لحين استرجاع الحزمة المفقودة.
- عشان يحلوا المشكلة، المتصفحات **بتفتح عدة اتصالات TCP متوازية**، لكن ده بيستهلك موارد أكتر.
- مفيش **أمان افتراضي** مع TCP العادي، فلازم إضافة TLS يدويًا.

🔒 ما هو TLS؟

TLS (Transport Layer Security) هو بروتوكول أمان يُستخدم لتشفير البيانات المنقولة بين الأجهزة على الإنترنت، مثل متصفحك والخوادم.

🛡️ ليه بنستخدم TLS؟

1. **التشفير (Encryption):** بيمنع أي حد من التنصت على البيانات أثناء نقلها.
2. **التأكد من الهوية (Authentication):** يضمن إن الموقع اللي بتتصل بيه هو الموقع الحقيقي، مش موقع مزيف.
3. **سلامة البيانات (Data Integrity):** بيضمن إن البيانات ما تمش التلاعب بيها أثناء النقل.

🔄 الفرق بين TLS و SSL؟

- **SSL (Secure Sockets Layer)** لأنه أكثر أمانًا **TLS** هو الإصدار الأقدم، وتم استبداله بـ.
- معظم المواقع حالياً تستخدم **TLS 1.2** أو **TLS 1.3**.
  ✅ **باختصار:** TLS هو اللي بيخلي المواقع تبدأ بـ **"https://"** بدل **http://** وده دليل على إن الاتصال مشفر وآمن. 🔐

🔹 الحل الجديد: HTTP/3

- بيعتمد على **بروتوكول QUIC (قائم على UDP)** بدل TCP، وده بيسمح **بنقل بيانات متوازٍ بدون تعطيل** لو حصل فقدان للحزم.
- **كل كائن (Object) له تحكم مستقل** في الخطأ والازدحام، وده بيقلل التأخير.
- **الأمان مدمج افتراضيًا** (TLS 1.3 جزء من QUIC، مش محتاج تفعيل يدوي زي TCP).
  ✅ **النتيجة:** HTTP/3 أسرع، أكثر استقرارًا، وبيعالج مشاكل HTTP/2 مثل التأخير الناتج عن فقدان الحزم، مع تحسينات في الأمان والتوازي. 🔥

📌 E-mail

1. **مكونات البريد الإلكتروني الأساسية:**
   - **User Agent : زي Outlook، Gmail، إلخ.
   - **Mail Server : مسؤولة عن إرسال واستقبال البريد.
   - **SMTP (Simple Mail Transfer Protocol)**: مسؤول عن إرسال الإيميلات بين السيرفرات.
2. **كيف يتم إرسال الإيميل؟**
   - كل **Mail Server** عنده **mailbox** (لحفظ الرسائل الواردة) و**message queue** (لإرسال الرسائل).
   - يستخدم SMTP لنقل الرسائل بين **Mail Servers**
   - السيرفر المرسل يعمل كـ **Client** والسيرفر المستلم يعمل كـ **Server**.
3. **SMTP RFC 5321:**
   - يستخدم **TCP** لنقل الرسائل بأمان على **Port 25**.
   - يتم النقل في **3 مراحل:**
     1. **SMTP Handshaking** (التعارف بين السيرفرات).
     2. **SMTP Transfer** (نقل الإيميلات).

3. **SMTP Closure** (إنهاء الاتصال).

- يتبع أسلوب **command/response** زي HTTP، حيث يستخدم أوامر نصية **ASCII** مع ردود بحالة العملية.

💡 ملحوظة:

- **يستخدم SMTP فقط لإرسال البريد، لكن لاستقبال البريد نحتاج بروتوكولات أخرى مثل IMAP أو POP3**
- **يستخدم TCP لضمان وصول البريد بدون فقدان البيانات.**

📊 جدول مقارنة بين HTTP و SMTP

العنصر	HTTP (HyperText Transfer Protocol)	SMTP (Simple Mail Transfer Protocol)
الوظيفة	نقل صفحات الويب والموارد من السيرفر إلى المتصفح	إرسال البريد الإلكتروني بين السيرفرات وعملاء البريد
نموذج العمل	Client Pull (المستخدم يطلب البيانات)	Client Push (المرسل يدفع البيانات إلى السيرفر)
طريقة التفاعل	الطلبات (Requests) يرسلها العميل، والاستجابات (Responses) يرسلها السيرفر	المرسل يدفع البريد إلى السيرفر، ثم يتم نقله إلى المستقبل
عدد الكائنات في الرسالة	كل كائن في HTTP يُرسل بشكل منفصل في استجابة مستقلة	يمكن إرسال عدة كائنات في رسالة واحدة (multipart message)
الاتصال	غير مستمر (stateless) – يتم إنشاء اتصال لكل طلب جديد	مستمر (persistent) – يبقى الاتصال مفتوحًا أثناء نقل البريد
نوع البيانات	HTML, JSON, XML, CSS, JavaScript, Images...	نصوص (ASCII) فقط، ويحتاج إلى ترميز (MIME) لإرسال المرفقات
التشفير والأمان	HTTPS (باستخدام TLS/SSL)	SMTPS (باستخدام TLS/SSL)
المنفذ الافتراضي	🔹 80 (HTTP) 🔹 443 (HTTPS)	🔹 25 (بدون تشفير) 🔹 465/587 (مع التشفير)
الاستخدام الرئيسي	تصفح الإنترنت ونقل البيانات بين المتصفح والسيرفر	إرسال البريد الإلكتروني بين المستخدمين والسيرفرات

- **HTTP** يُستخدم لتصفح الويب وتحميل الصفحات والمحتوى التفاعلي.
- **SMTP** يُستخدم لنقل البريد الإلكتروني بين السيرفرات والمستخدمين.
- الفرق الجوهري أن **HTTP يعتمد على الطلب والاستجابة**، بينما **SMTP يعتمد على الدفع التدريجي للرسائل**.

DNS: Domain Name System

مسؤول عن **تحويل أسماء المواقع (Domains) إلى عناوين IP** حتى يتمكن الكمبيوتر من تحديد موقع السيرفر المطلوب على الإنترنت.

- قاعدة بيانات موزعة (Distributed Database) تنظمها مجموعة من (Name Servers)**.
- يعمل كبروتوكول في (Application Layer) ليساعد في تحويل **الأسماء إلى عناوين IP**.
- يعتبر من الوظائف الأساسية للإنترنت، لأن كل معاملة تقريبًا تحتاج إلى DNS.

📄 خدمات DNS

- 🔹 **ترجمة أسماء النطاقات إلى عناوين IP**
- 🔹 **الـ Host Aliasing**: تعيين أكثر من اسم لنفس العنوان
- 🔹 **الـ Mail Server Aliasing**: تحديد أسماء سيرفرات البريد الإلكتروني
- 🔹 **تحميل متوازن (Load Balancing)**: استخدام عدة عناوين لنفس الموقع لتوزيع الضغط

لماذا لا نجعل الـ DNS مركزيًا؟

🚫 **المشاكل التي ستحدث إذا كان مركزيًا:**

- **نقطة فشل واحدة** (Single Point of Failure)
- **حجم ضخم من الطلبات** لا يمكن تحمله
- **زيادة التأخير (Latency) بسبب بعد السيرفر المركزي**
- **صعوبة الصيانة والتحديثات**

✅ **الحل:** توزيع الـ DNS عالميًا، مما يجعله أكثر كفاءة وأمانًا

مميزات الـ DNS

- 🔹 **يخدم تريليونات من الطلبات يوميًا**
- 🔹 **أداء عالي لأن كل اتصال بالإنترنت يحتاج إلى DNS**
- 🔹 **موزّع تنظيميًا وجغرافيًا** بحيث لا يوجد تحكم مركزي
- 🔹 **يُعتبر "bulletproof" أي مقاوم للفشل بسبب آلية التوزيع**

💡 الخلاصة

الـ **DNS هو العمود الفقري للإنترنت**، حيث يجعل التصفح سهلاً للمستخدمين عبر تحويل الأسماء إلى عناوين رقمية يمكن للأجهزة فهمها. ولأنه نظام موزع، فإنه أكثر أمانًا وكفاءة، ويمنع حدوث مشاكل الأداء والانهيار في حال وجود عدد كبير من الطلبات. 🚀

🔎 شرح الـ DNS Hierarchy & Root Name Servers

1️⃣ الـ DNS كنظام هرمي (Hierarchical Database)

الـ DNS يعمل كنظام **موزّع وهرمي**، حيث يتم تقسيم المسؤوليات بين عدة أنواع من الخوادم:

📌 **الجذر (Root DNS Servers):**

- النقطة الأولى في البحث، وهي مسؤولة عن توجيه الاستعلامات إلى الـ **Top-Level Domains (TLDs)** مثل `.com`، `.edu` ، `.org`

📌 **خوادم النطاقات العُليا (TLD Servers):**

- مثل `com DNS servers.` ، مسؤولة عن معرفة الـ DNS الخاص بالمواقع التي تنتهي بـ `.com` مثل `amazon.com`

📌 **خوادم الأسماء الموثوقة (Authoritative DNS Servers):**

- مثل `amazon.com DNS server` ، وهو الذي يملك عنوان IP النهائي لموقع `www.amazon.com`

2️⃣ كيف يعمل الـ DNS عند البحث عن موقع؟

لنفترض أن العميل يريد معرفة عنوان IP لموقع `www.amazon.com` :
✅ **الخطوة 1:** يرسل الجهاز استعلامًا إلى **Root DNS Server** ليعرف من المسؤول عن `.com`
✅ **الخطوة 2:** يرسل الاستعلام إلى **com DNS Server.** ليحصل على عنوان **amazon.com DNS Server**
✅ **الخطوة 3:** يرسل الاستعلام إلى **amazon.com DNS Server** ليحصل على **عنوان IP النهائي**

3️⃣ خوادم الجذر (Root Name Servers)

📌 **ماذا تفعل؟**

- تعتبر **الملاذ الأخير** في حال لم يستطع أي سيرفر DNS حل اسم النطاق
- مسؤولة عن توجيه الاستعلامات إلى خوادم TLD المناسبة

📌 **أهميتها:**

- الإنترنت لا يمكن أن يعمل بدونها!
- تدعم **DNS Security Extensions (DNSSEC)** للحماية من الهجمات
- تديرها منظمة **ICANN**

📌 **عددها:**

- يوجد **13 خادمًا منطقيًا حول العالم**، لكن كل واحد منهم **يُكرر عدة مرات** ليصل العدد إلى **حوالي 200 خادم في الولايات المتحدة وحدها**

💡 الخلاصة

الـ **DNS يعمل كنظام موزّع وهرمي**، حيث يمر الاستعلام بعدة مراحل منRoot DNS Servers إلى **TLD Servers** ثم إلى Authoritative DNS Servers لحل الاسم إلى عنوان IP.

وتعتبر Root DNS Servers أهم مكونات الـ DNS لأنها البداية لكل استعلام وهي المسؤولة عن توجيهه إلى المكان الصحيح. 🚀

📌 الخوادم المحلية (Local DNS Name Servers)

✅ عندما يطلب جهاز اسم نطاق، يتم إرسال الطلب إلى **الخادم المحلي (Local DNS Server)**
✅ يقوم الخادم المحلي إما:

- بالرد مباشرة إذا كان لديه **Cache (ذاكرة مؤقتة)**
- أو **تمرير الطلب** إلى التسلسل الهرمي للـ DNS
  - 🔹 يمكنك معرفة الـ DNS المحلي على جهازك عبر:
- **MacOS:** استخدام الأمر `scutil --dns`
- **Windows:** استخدام الأمر `ipconfig /all`

⚠ **الخوادم المحلية ليست جزءًا رسميًا من التسلسل الهرمي للـ DNS ولكنها تساعد في تسريع العمليات**

📌 استعلامات DNS التكرارية (Iterated Queries)

✅ عندما يبحث مضيف في **engineering.nyu.edu** عن عنوان **gaia.cs.umass.edu**، يتم اتباع الخطوات التالية:
1 يرسل الاستعلام إلى (dns.nyu.edu) **Local DNS Server**
2 إذا لم يكن لديه الإجابة، يستفسر من **Root DNS Server**
3 الجذر يوجهه إلى **TLD DNS Server**
4 الـ TLD يوجهه إلى **الخادم الموثوق (Authoritative DNS Server)**
5 يتم إرجاع **عنوان الـ IP النهائي** إلى الجهاز الطالب

📌 **في هذا النوع من الاستعلامات، كل خادم يُخبر الجهاز أي خادم يجب أن يسأل بعده، حتى يتم العثور على عنوان الـ IP المطلوب.**

📌 الاستعلام التكراري العكسي (Recursive Query)

✅ في هذا النوع، يقوم **(Local DNS Server)** بتحمل مسؤولية العثور على العنوان المطلوب، حيث:
1 يقوم الجهاز بإرسال طلب إلى **الخادم المحلي**
2 إذا لم يكن لديه الإجابة، يقوم **الخادم المحلي** بالسؤال بدلاً من الجهاز عبر التسلسل الهرمي للـ DNS
3 يتم توجيه الطلب إلى **Root DNS Server** ثم إلى **TLD Server** ثم إلى **Authoritative DNS Server**
4 بمجرد العثور على عنوان الـ IP، يتم إرجاعه إلى الجهاز

⚠ **عيب هذا النوع؟**

- يؤدي إلى heavy load at upper levels of hierarchy حيث تتحمل الخوادم المحلية مسؤولية البحث بالكامل

تخزين الكاش (Caching DNS Information) ⚡

- **عشان نقلل عدد الاستعلامات ونسرّع البحث، DNS بيخزن النتائج لفترة معينة.**
- كل نتيجة لها **TTL (Time To Live)** يحدد مدة صلاحية الكاش.
- لو الـ **IP Address للموقع تغير قبل انتهاء الـ TTL، المستخدم هيشوف IP قديم!**

📌 (DNS Records)

🔹 **السجلات المخزنة في قاعدة بيانات DNS تأتي بأشكال مختلفة، أشهرها:**

✅ 📌 **سجل (type=A)**

- بيربط اسم الDomain بـ IP Address

مثال:

```
example.com → 192.168.1.1
```

✅ 📌 **سجل (type=NS)**

- بيحدد الـ Name Server المسؤول عن الDomain

مثال:

```
example.com → ns1.example.com, ns2.example.com
```

✅ 📌 **سجل (type=CNAME)**

- بيعمل Alias لاسم دومين إلى اسم دومين تاني
- مثال:

```
www.example.com → example.com
```

✅ 📌 **سجل (type=MX)**

- يحدد **خادم البريد SMTP** المسؤول عن استقبال البريد الإلكتروني لهذا النطاق
- مثال:

```
example.com → mail.example.com
```

(DNS Protocol Messages) 📩**

أي استعلام DNS بيتم إرساله في **رسالة DNS Protocol**، والرسائل دي لها **تنسيق موحد**:

🔹 **Header (16 بت) يحتوي على:**

- **Identification**: للتمييز بين الطلبات.
- **Flags** (query or reply) :لتحديد نوع الطلب
- **additional “ helpful” info that may be used**
  - **Body (البيانات الفعلية):**

📌 كيفية تسجيل الدومين وإدخاله في نظام DNS

🔹 1 ما هو DNS Registrar؟

🔸 **DNS Registrar** هو مزود خدمة مسؤول عن **تسجيل أسماء النطاقات (Domain Names)** وربطها بخوادم DNS.

🔸 من أشهر المسجلين:

✓ GoDaddy
✓ Namecheap
✓ Google Domains
✓ Cloudflare Registrar

⏳ **ماذا يحدث عند تسجيل دومين جديد؟**

1. تقوم باختيار اسم الدومين، وليكن **networkutopia.com**.
2. تشتري الدومين من أحد الـ **DNS Registrars**.
3. يُطلب منك إدخال معلومات خوادم الـ **DNS الخاصة بك**.
4. يتم إدراج اسم الدومين في **قاعدة بيانات DNS العالمية** عن طريق خوادم **TLD الخاصة بـ "com."**.

🔹 2 إدخال الدومين إلى DNS – كيف يتم؟

📌 **بعد التسجيل، يتم إضافة سجلات DNS في خوادم TLD الخاصة بـ "com."**.

🔸 السجلات التي يتم إدراجها:

📍 NS Record (Name Server Record)

- يحدد **الخادم المسؤول عن إدارة النطاق**.
- مثال

```
networkutopia.com → dns1.networkutopia.com (NS)
```

📍 A Record (Address Record)

- يربط **الدومين بعنوان IP** الخاص بالموقع.
- مثال:

```
dns1.networkutopia.com → 212.212.212.1 (A)
```

📌 لماذا هذا مهم؟

✅ بدون تسجيل النطاق وإدخاله في DNS، لن يكون الموقع متاحًا.
✅ يعتمد الإنترنت بالكامل على DNS لترجمة **أسماء النطاقات إلى عناوين IP**.
✅ أي خطأ في إعداد DNS قد يؤدي إلى عدم قدرة الزوار على الوصول إلى موقعك.

🔹 (DNS Security) 🛡️

(DDoS Attacks) 🚨 **

- 🔸 **مهاجمة سيرفرات الـ Root** بإرسال كمية ضخمة من الطلبات، لكن حتى الآن لم ينجح ذلك بسبب:
  - وجود **فلترة للمرور**.
  - **الكاش المحلي للسيرفرات يقلل من الضغط على السيرفرات الجذرية**.
  - 🔸 **مهاجمة سيرفرات TLD** مثل `com.` أو `org.`، ودي أخطر لأنها تؤثر على ملايين المواقع.

(Spoofing Attacks) 🎭 **

- 🔸 يتم اعتراض طلبات DNS وإرسال **ردود مزورة**.
- 🔸 من أشهر الهجمات:

- **DNS Cache Poisoning**: إدخال بيانات خاطئة في كاش DNS.
- **DNSSEC (RFC 4033)**: بروتوكول يضيف توقيعات رقمية لمنع التزوير.
  - 🔹 **كيف تحمي نفسك؟**
  - ✔️ استخدام **DNSSEC** لحماية بيانات DNS.
  - ✔️ تجنب استخدام **خوادم غير موثوقة**.
  - ✔️ تحديث **أنظمة الحماية** باستمرار.
