Web and HTTP

- اپلیکیشن وب یکی از مهم ترین اپلیکیشن ها در اینترنته.
- در اوایل دهه ی ۱۹۹۰ اینترنت فقط برای متخصصان بود. در سال ۱۹۸۹ ، وب توسط آقای Berners-Lee معرفی شد و بعد ها به یه پلت فرمی تبدیل شد که همه میتونستن ازش استفاده کنن.

در اون زمان شبکه هایی که برای اطلاعات استفاده می شدن ، شبکه های broadcast مثل تلویزیون و رادیو بودن . ولی وب ویژگی هایی مثل : ۱- on-demand داره ، یعنی به جای اینکه برنامه توسط شخصی غیر از ما باشه، ما هر زمان بخوایم به اطلاعات و سرویس ها دسترسی داریم و ازشون میتونیم استفاده کنیم.

۲- هرکسی میتونه با قیمت ارزان publisher بشه ، یعنی نظرات خودش رو بیان کنه ، document هاشو در معرض دید سایرین بذاره و صداشو به گوش همه برسونه.

۳-توی وب فایل هایی از قبیل تصاویر و فیلم و ... هست و شبیه به مناظر طبیعی هست و همین حس طبیعی بودن رو به مخاطب القا می کنن.

اصطلاحات :

- Web page : اون مجموعه ای از آبجکت هاست که میتونن داخل وب سرور های مختلف ذخیره بشن .

این آبجکت ها میتونن HTML file ، تصاویر به فرمت های متفاوت(مثل JPEG) ، java applet ، audio file و ... باشن.

معمولا Web page ها یه base HTML-file دارن که این فایل محل هر کدوم از آبجکت های دیگه رو داخل Web page و رفرنس اون ها رو در اختیار ما میذاره . هر آبجکتی یه url داره ، که یه آدرس یونیکه ، و تمام آبجکت ها توسط url مخصوص خودشون قابل دسترسی هستن.

url از دو قسمت host name و path name تشکیل شده.

host name نام اون نودی هست که آبجکت در اون قرار گرفته .

path name مسیر دایرکتوری ای هست که ما باید در نود طی کنیم تا به محلی که توی فایل سیستم سرور قرار گرفته دسترسی پیدا کنیم.

فایل base HTML جایگاه آبجکت ها توی Web page و url مربوط به اون ها رو هم تعیین می کنه. Browser ما وقتی یه base HTML-file رو تقاضا می کنه و بعد قسمت های مختلف اون رو parse و بررسی می کنه ،تگ های مختلف رو ویزیت می کنه ، و اون تگ هایی که متناسب با اون url هستن رو دوباره تقاضای

دریافتشون رو از سرور می کنه و توی اپلیکیشن وب طبق قواعد HTML به کاربر نشون میده.

پروتکل HTTP (Hyper Text Transfer Protocol):

- پروتکل HTTP قلب تکنولوژی وب هست. ما ازین پروتکل برای ارتباط بین کلاینت و سرور استفاده می کنیم.
- به صورت عمده توی پروتکل HTTP پیام ها دو نوع هستن، request و response . request ها توسط کلاینت فرستاده میشن و response ها توسط سرور گرفته میشن.
- پروتکل HTTP هم مثل سایر پروتکل ها دوتا side داره ، یه side سمت سرور و یه side سمت کلاینت پیاده سازی میشه.
- توی اپلیکیشن وب ، کلاینت و browser می تونن به جای همدیگه استفاده بشن چون اون ماژولی که نقش کلاینت رو ایفا می کنه browser هست . browser (طبق قاعده ی HTTP) تقاضا رو برای سرور ارسال می کنه و جواب ها رو به نحو مناسب تفسیر و نمایش میده.
- Server هم منتظر دریافت تقاضاهای HTTP هست و وقتی دریافت کرد ، اگه توی اون تقاضا دنبال فایلی باشیم ،سرور میاد با استفاده از url ای که داخل پیام HTTP request هست اون آبجکت رو توی

فایل سیستم خودش پیدا می کنه و مجددا اون رو طبق قواعد HTTP برای کلاینت ارسال می کنه.

- HTTP از TCP استفاده می کنه. چون کاری که HTTP انجام میده file transferring هست و ما توی این نوع کاربرد نمی تونیم خطا رو تحمل کنیم و باید ارتباط reliable باشه.
- بعد از اینکه request دادن و response گرفتن انجام شد ، ارتباط TCP می تونه بسته بشه.
- پروتکل HTTP یه پروتکل stateless یا بدون حافظه هست . دلیلش هم اینه که به صورت اولیه ما اطلاعاتی از کاربران توی سرور ذخیره نمی کنیم . مثلا اگه یه کلاینتی یه بار یه آبجکتی رو از سرور تقاضا کنه ، و دوباره هم این کارو انجام بده ، این طور نیست که سرور بگه این همون آبجکتی هست که قبلا فرستادم و دوباره طی یه کانکشن دیگه آبجکت رو براش می فرسته.

اینکه ما state یا history کاربرانمون رو داشته باشیم باعث پیچیده شدن پروتکل میشه ، هم باید اطلاعات رو ذخیره و محافظت کنه و هم اگه یکی از دو طرف client/server کرش کنن ، ممکنه اون state ای که از هم دارن با دیگری متفاوت باشه و مجبور بشیم یه مرحله ی هماهنگ کردن state هم داشته باشیم که باعث پیچیده شدن پروتکل میشه . (چون HTTP از اول قرار بوده ساده باشه دیگه ویژگی stateless رو داره)

- بر حسب این که HTTP از چندتا کانکشن TCP استفاده می کنه ، نسخه های مختلف HTTP متفاوت عمل می کنن.

۱-نسخه ی non-persistent (ناپایدار) : برای دریافت هر آبجکت یه کانکشن TCP ایجاد می کنیم و بعد از دریافت هم اون کانکشن بسته میشه. (به ازای هر آبجکت یه کانکشن داریم)

۲- در نسخه ی persistent یک کانکشن TCP ایجاد می کنیم، و بعد از اینکه فایل base HTML رو دریافت کردیم و متوجه شدیم چندتا آبجکت دیگه احتیاج داریم تا کل web page رو داشته باشیم، میایم در قالب همون کانکشن TCP سایر آبجکت ها رو تقاضا می کنیم و از سرور دریافت می کنیم ، و بعد از دریافت همه ی آبجکت ها اون کانکشن TCP بسته میشه .

- مثال : مثلا یه web page رو می خوایم دانلود کنیم ، شامل base HTML و ۱۰ تا تصویر . اگه نسخه ی HTTP ، non-persistent باشه ، ابتدا یه کانکشن TCP ایجاد می کنیم و برای ایجاد این کانکشن باید یه سیگنالینگی بین کلاینت و سرور رخ بده.

کلاینت یه پیام TCP از جنس syn به سرور می فرسته . برای فرستادن پیام احتیاج به ۳ تا سیگنال داریم : ۱- سیگنالی که به سرور می فرستیم syn نام داره. ۲- جوابی که سرور میده syn-ack

نام داره . ۳- سیگنال بعدی ای که کلاینت به سرور می فرسته هم ack نام داره .

به این رد و بدل سیگنال three-way handshaking میگن .

بعد از این پیام های کنترلی می تونیم بگیم ارتباط TCP شکل گرفته.

- مثلا در سیگنال اولی که کلاینت برای سرور می فرسته ، مشخص می کنه که سگمنت هایی که قراره بفرسته با چه عددی شروع می کنه به شماره گذاری کردن. (به خاطر اینکه ارتباط TCP یه ارتباط reliable هست و نباید ترتیب بسته ها به هم بریزه) به این کار، میگن initial sequence number که در سیگنال اول انجام میشه.
- بعد ازینکه سرور سیگنال اول رو دریافت کرد ، initial sequence number ای که خودش انجام میده به همراه یه سری اطلاعات دیگه در سیگنال دوم برای کلاینت می فرسته .
- در آخر هم یه سیگنال ack از کلاینت به سرور فرستاده میشه که ارتباط TCP شکل بگیره.
- در دوتا سیگنال اول ، فقط سیگنال های کنترلی رو و بدل میشه و توی قسمت دیتا شون چیزی نیست ، ولی توی این سیگنال سوم داخل قسمت دیتا ،پروتکل های لایه اپلیکیشن می تونه وجود داشته باشه و ما در قالب هدر اون بسته ، ارتباط TCP رو در سیگنال سوم

ایجاد می کنیم. برای همین اصطلاحا میگن سیگنال سوم ، pig-back میشه!

- برای دریافت اولین آبجکت که همون base HTML هست ، ابتدا دوتا سیگنال فرستاده میشه (اول از کلاینت به سرور و بعد از سرور به کلاینت) سیگنال سوم هم در قسمت داده اش ، HTTP request قرار داده میشه و مشخص می کنه که باید به عنوان اولین آبجکت فایل base HTML باید برای کلاینت فرستاده بشه. بعد از دریافت سیگنال سوم هم سرور میاد با توجه به url این آبجکت ، محلش رو پیدا می کنه و در پیام چهارم فایل base HTML رو برای کلاینت میفرسته.
- بعد از پیام چهارم ، ارتباط TCP بسته میشه و از بین میره.
- توی مرحله ی پنجم ، بر اساس اون فایل base HTML ای کلاینت دریافت می کنه ، می تونه بفهمه چه آبجکت های دیگه ای در اون web page وجود داره . مثلا توی این مثال ۱۰ تا تصویر به فرمت jpeg داریم . با توجه به url ای که از طریق base HTML دریافت کردیم ، می تونیم دوباره برای دریافت هر کدوم از این آبجکت ها همین روند رو که برای دریافت base HTML انجام دادیم تکرار کنیم.(برای هر تصویر به نوبت این کار انجام میشه)

- تاخیری که این فرایند برای دریافت کل web page با تمام آبجکت هاش احتیاج داره ، کافیه تاخیر دریافت هر آبجکت رو ضرب در تعداد آبجکت ها کنیم .

RTT(Round Trip Time) به زمانی میگن که طول می کشه یه بسته ی کوچک مسیر بین کلاینت و سرور و بره و برگرده.

برای این میگیم بسته ی کوچک(small packet) ، چون می خوایم تاخیر transmission رو در نظر نگیریم و فقط تاخیر مسیر رو در نظر بگیریم . توی سیگنال های اول و دوم و سوم چون فقد سیگنال های کنترلی داریم طول بسته کوچک هست و میشه از تاخیر transmission صرف نظر کرد . اما توی سیگنال چهارم که داره فایل ارسال میشه ، بسته بزرگ هست و ممکنه تقسیم بشه به چندتا سگمنت و سرور این ها رو به کلاینت بفرسته، و اینجا تاخیر transmission هم مقدار قابل ملاحظه ای میشه و متناظر با حجم بسته تقسیم بر throughput مؤثری هست که بین کلاینت و سرور وجود داره .

پس کل زمان دریافت یه آبجکت میشه :

2RTT + file transmission time

file transmission time = length of file / throughput

در نهایت برای زمان دریافت کل صفحه ی وب، باید این زمان رو ضرب در تعداد کل آبجکت ها بکنیم.

- توی نسخه ی non-persistent اتلاف منابع زیاده چون به ازای هر آبجکت یه RTT داریم و سیستم عامل باید برای هر کانکشن TCP یه سری منابع رو در اختیار سوکت قرار بده.این منابع از جنس بافر و تایمر هستن و اینا باعث اتلاف منابع تو سیستم عامل سمت سرور میشه.
- معمولا تو نسخه ی non-persistent میان از روش parallel TCP connection استفاده می کنن ، یعنی وقتی base html رو توی رو توی transaction اول دریافت کردیم ، بعد همه ی اطلاعات راجع به سایر آبجکت ها رو در اختیار داریم ،مثل url . به جای اینکه این ها رو به صورت تک تک از سرور درخواست کنیم ، به صورت موازی در خواست می کنیم و کانکشن های TCP به صورت موازی شکل می گیرن . این باعث میشه در خواست های TCP ، overlap پیدا کنن و تاخیر کمتر بشه.

بنابراین کل تاخیر برابر میشه با 4 RTT به اضافه ی تاخیر transmission کانکشن اول ، به اضافه ی ماکزیمم تاخیر transmission بقیه ی کانکشن ها . البته گاهی وقت ها مواردی پیش میاد که به جای ماکزیمم ، مجموع همه ی این تاخیر ها رو حساب می کنیم.(چه مواردی؟ -> سوال تالار گفتگو)

- نسخه ی persistent :

- نسخه ی HTTP 1.1 ازاین نوع هست.
- توی persistent HTTP وقتی یه کانکشن TCP ایجاد کردیم برای دریافت فایل HTML، اون کانکشن از بین نمیره و سرور اون کانکشن رو باز نگه می داره ،و بعد browser وقتی اون فایل base HTML رو دریافت می کنه ، و متوجه میشه چه آبجکت های دیگه ای در اون web page هست، اون ها رو از طریق همون کانکشن TCP تقاضا می کنه و دریافت می کنه و دیگه three-way handshaking نداریم.
- درسته که توی نسخه ی persistent ، برای هر آبجکتی تقاضای یه ارتباط جداگانه نمی کنیم و overhead تقاضای جدید نداریم ، ولی همین sequential رفتار کردن باعث تاخیر زیادی میشه .
مکانیزمی که اینجا استفاده می کنیم برای کم کردن تاخیر ، pipe line نام داره .
- روش pipe line به این شکله که بعد از دریافت فایل base-html به تعداد بقیه ی آبجکت ها به صورت back-to-back درخواست میدیم و دیگه منتظر جواب اون ها از سرور نمی مونیم و بعد از همه ی درخواست ها ، سرور میاد یکی یکی جواب همشون رو میده. این پیام ها که از جنس request هستن پیام های کوچکی هستن و زمان ارسالشون رو توی فرمول میتونیم نادیده بگیریم ، پس 3 تا

RTT داریم و به تعداد آبجکت ها transmission delay که جمع این ها ، از مقدار تاخیر در حالت عادی ای که نسخه ی persistent داره کمتره.

- لازم نیست همه ی آبجکت هایی که روی یک صفحه ی وب هستن ، روی یه سرور هم باشن ، بلکه این آبجکت ها url دارن و این url ها قسمت host name شون لزوما مثل هم نیست و ممکنه برای آبجکت های مختلف ارتباط های TCP با سرور های مختلف برقرار کنیم.

- همونطور که گفتیم دو نوع پیام HTTP داریم ، request و response .
  - HTTP request message
- پیام های HTTP به فرمت ASCII هستن و human-readable هستن. دلیلش هم اینه که HTTP جزو پروتکل های قدیمی هست و اون موقع استفاده بهینه از پهنای باند مطرح نبوده و اینکه تا اون جایی که میشه هدر ها کوچک باشن و بعضا افراد خودشون باید این پیام ها و هدر ها رو ایجاد می کردن و به همین دلیل هم راحت تر بوده که جوری طراحی بشن که یه کاربر انسانی هم بتونه اون ها رو بخونه، راحت تایپ کنه و کار مشکلی نداشته باشه.

- هر پیام HTTP request یه request line داره که خط اول این پیام هست. ابتدای پیام یک متد رو بیان می کنیم که معروف ترین متد GET هست که در اون ما تقاضای دریافت یه resource یا آبجکت رو می کنیم. بعد از اون ما باید قسمت path name رو قید کنیم (نه host name) که با اسلش شروع میشه و بعد از اون یه فاصله میذاریم ، کلمه ی HTTP و دوباره اسلش ، و بعد ورژن HTTP رو مشخص می کنیم. بعد از ورژن اینتر رو میزنیم که این اینتر خودش متناظر با دوتا کاراکتر هست :carriage return و line feed . (\r\n)
- بعد از این، قسمت header lines رو داریم ، که طولش میتونه متغیر باشه .حداقل header line ای که می تونیم داشته باشیم خطیه که اولش کلمه ی Host رو قید کردیم. (بقیه ی header line ها ضروری نیستن و اگه نباشن bad request دریافت نمی کنیم) . بعد از کلمه ی Host ، یه علامت : و یه فاصله قرار میدیم و host name رو می نویسیم و مجددا با \r\n این header line رو هم ختم می کنیم. (انتهای هر header line ای، یه \r\n وجود داره.)
- هدر لاین دیگه ای به اسم User-Agent وجود داره که نوع browser کلاینت رو تعیین می کنه تا اگه صفحه وب با فرمت

خاصی مخصوص اون browser ، در سرور وجود داره ،همون به کلاینت ارسال بشه.

- فرمت هایی که browser میتونه قبول کنه توی هدرلاین Accept میاد .
- توی هدر لاین Accept-Language هم اون زبان هایی که browser ترجیح میده محتوای صفحه ی وب از سمت سرور بهش ارسال بشه آورده شده.
- توی هدر لاین Accept-Encoding ، Encoding های قابل قبول رو ذکر می کنیم که می تونن decode بشن.
- توی هدرلاین connection هم عبارت keep-alive وجود داره که متناظر با عبارت persistent عه (البته این مربوط به مثال کتابه که نسخه ی HTTP، 1.1 هست)

  اگه به جای عبارت keep-alive، عبارت close رو داشته باشیم ، متناظر با non-persistent هست .
- بعد از همه ی هدرلاین ها ، یه اینتر دیگه هم میزنیم (درواقع یه عبارت \r\n دیگه هم وجود داره) که انتهای پیامی که می خوایم به سرور بفرستیم مشخص بشه.
- بعضی از request ها بعد از قسمت هدرلاین ها ، یه entity body دارن که داده هاشون در اون قرار می گیره (مثل POST و PUT) . البته متد GET داده نداره و این قسمتش خالیه.

- ما قبل از اینکه پیام HTTP request بفرستیم، یه ارتباط TCP برقرار می کنیم و آدرس سرورمون مشخصه.دلیل اینکه دوباره آدرس host name رو توی هدرلاین Host می نویسیم چیه؟(تالار گفتگو)

- Other HTTP request message

- POST method : بعضی صفحات وب هستن که کارکرد اون ها مبتنی بر گرفتن ورودی از کلاینته (تحت عنوان form) و اون کلاینت باید اون فرم رو پر کنه و برای سرور ارسال کنه و مبتنی بر اون مقادیری که توی فرم پر کرده، یه محتوایی براش ارسال میشه. شاخص ترین این صفحات وب ،موتور های جستجو(research engines) هستن. داده هایی که توسط کلاینت جستجو می شن توی قسمت entity body اون در خواست HTTP قرار می گیره.
- GET method هم که گفتیم به طور اولیه برای درخواست یه فایل یا آبجکتی از سرور استفاده میشه . اما میتونه کاری که متد POST می کنه رو هم از طریق مفهومی به اسم extended url انجام بده. این متد میتونه مقادیری که در بعضی از فرم ها هستن (عنوان فیلد فرم + چیزهایی که کاربر وارد کرده) توی قسمت url توسط علامت هایی مثل & یا ? مشخص کنیم وبرای سرور بفرستیم.
- HEAD method : بیشتر برای دیباگ کردن استفاده میشه. مثل متد GET عه ولی وقتی با استفاده از این متد به سرور پیام می

فرستیم ، سرور فقط هدر پیام رو برای ما می فرسته و آبجکت رو نمی فرسته.

- PUT method : برای web publishing استفاده میشه ، میتونیم از کلمه ی PUT استفاده کنیم و بعد یه url ای رو مشخص کنیم و بعد چیزی که توی body پیام ما هست رو میره جایگزین url توی صفحه ی وب ما می کنه و داده های قسمت body به طور کامل با فایلی که توی url مشخص شده جایگزین میشه.

- HTTP response message :

- این پیام ها هم ASCII و human readable هستن.هدر لاین دارن. قبل از قسمت هدرلاین ، یه خطی هست که خط اول این پیامه و بهش status line گفته میشه. توی این خط اول ، ابتدا ورژن HTTP رو میگیم و بعد یه کدی رو میگیم ، و بعد معنی کد رو .البته این معنی کد اضافه است و چون موقع ساخت پروتکل HTTP ملاحظات صرفه جویی توی هدر در نظر گرفته نمی شده ، این معنی کد هم توی خط اول ذکر میشه. مثلا اگه کلاینت یه فایلی رو از سرور درخواست کرده باشه ، اگه درخواست موفقیت آمیز باشه کد 200 ارسال میشه و بعد اطلاعاتی راجع به آبجکت در قالب هدر لاین ها و خود اون فایل درخواست شده توی قسمت body پیام ارسال میشه.

- هدر لاین ها کلا یه عنوان دارن، یه«:» ، یه اسپیس ، و یه ولیویی که متناسب با اون عنوان هست.
- هدر لاین Date تاریخ رو مشخص می کنه.
- هدرلاین Server ، نوع سرور رو مشخص می کنه .
- هدرلاین Last-Modified زمان آخرین آپدیت آبجکت رو مشخص می کنه.
- Content-Length طول پیام رو مشخص می کنه.
- Content-Type هم نوع محتوا رو مشخص می کنه (نوع فایل + encoding)
- در آخر هم هدرلاین ها با یه \r\n از قسمت body جدا میشن.

- HTTP response status codes :

- کد 200 OK : موفقیت آمیز
- کد 404 Not Found : سرور نمی تونه آبجکت درخواست شده رو توی فایل سیستم هاش پیدا کنه.
- کد 301 Moved Permanently : آبجکت از روی اون صفحه ی وب منتقل شده. البته توی این موقعیت صفحه وب ، آدرس جدید آبجکت رو داره و طی هدر لاین جدیدی به اسم Location آدرس جدید اون آبجکت رو برای کلاینت می فرسته.

- کد 400 Bad Request : وقتی سینتکس پیاممون با پروتکل مطابقت نداره .
- 505 HTTP Version Not Supported :وقتی ورژن HTTP کلاینت و سرور باهم فرق کنه.
- ابزار هایی هست که میشه با اون ها پیام های HTTP رو در عمل دید. مثل netcat که برخلاف telnet کانکشن UDP هم با سرور ها ایجاد می کنه.