Protocol layers, service models

- شبکه های کامپیوتری ، خیلی پیچیده ان و مسئله ی مهم اینه که چطور بتونیم تحلیل و بررسی شون کنیم. آیا ساختاری هست که بتونیم مبتنی بر اون شبکه رو مطالعه و طراحی یا حتی پیاده سازی کنیم؟ بله:)
- اینکه ساختار ها رو به یه سری کارها و گام های کوچکتر بشکنیم ، میتونیم بهش modularity ، یا layering یا شکست به زیرکار ها بگیم.
- **مثال اول** : مهم ترین تکنولوژی توی حوزه ی مهندسی کامپیوتر ، طبیعتا خود کامپیوتر هست . و خود ساختاری که کامپیوتر داره ، ساختاریه که توسط آقای «وان نیومن» پیشنهاد شد که ما سخت افزار و نرم افزار رو از هم تفکیک می کنیم و یه API تحت عنوان Instruction Set Architecture(ISA) یا زبان ماشین سطح پایین بین این دوتا داریم.

مادامی که این ISA یکسان باشه ، مهندسین سخت افزار جدای از نرم افزار میتونن ماژول های سخت افزاری رو پیاده سازی کنن و مهندسین نرم افزار هم مستقل ازینکه چه جزئیاتی در سخت افزار به کار رفته رو اپلیکیشن خودشون کار کنن و زبان مشترک بین این دو ، همون ISA هست که سخت افزار ساپورت می کنه و ما اگه برنامه هامون رو در قالب

زبان های ماشین بنویسیم می تونیم از اون تکنولوژی سخت افزار استفاده کنیم تا برنامه مون انجام بشه.

- **مثال دوم** : یه برنامه نویس برای نوشتن برنامش(چه ساده چه پیچیده) ، از یه سری ماژول های آماده استفاده می کنه. در واقع task ما اینجا برنامه نویسی هست و sub-tasks های ما فانکشن هایی هستن که برای نوشتن برنامه استفاده می کنیم ، زبان مشترکمون هم آرگومان هایی هست که رد و بدل می کنیم و مقادیر بازگشتی ای که از فانشکن ها دریافت می کنیم.
- **مثال سوم** : توی مسافرت هوایی ، task ما سفر کردنه ، sub-tasks های ما رفتن به آژانس مسافرتی، check-in کردن ، رفتن به گیت های بازرسی ، برخاستن از باند ، مسیریابی هواپیما و... هست. زبان مشترک یا interface هم ، پاسپورت ، بلیط ، شماره سریال و ... هست که توی قسمت های مختلف نشون میدیم یا تحویل می گیریم که مجوز ورود ما به قسمت بعدیه.
- مثال هواپیما خیلی شبیه به شبکه های کامپیوتریه ازین حیث که به جای حمل و نقل انسان ها ، حمل بسته ها رو داریم.
- هر layer یه کاری انجام میده و یه سرویسی میده و مبتنی بر کاری که لایه ی بالایی انجام داده ما میتونیم به لایه ی پایینی بریم. هر لایه ای هم فقط با لایه ی بالایی و پایینیش در ارتباطه.
- حالا مزایای ساختار layering چیه؟

۱-به ما اجازه ی فهم و طراحی بهتر سیستم رو به ما میده.
۲- باعث میشه آپدیت و نگه داری سیستم راحت بشه ، چون اگر قرار شد یک چیزی داخل یه قسمت عوض بشه ، مادامی که API اش با لایه بالایی و پایینی تغییر نکنه ، بقیه ی قسمت های سیستم نیازی نیست تغییر کنن.
اگه به دید پیاده سازی هم به این قضیه نگاه کنیم ،این ساختار لایه ای ، باعث میشه که تغییرات منتشر نشن و بتونیم هنگام طراحی و پیاده سازی ، علاوه بر عمل کردن به طور ماژولار، هنگام آپدیت شدن نگران بقیه ی قسمت های سیستم نباشیم.

- پروتکل هایی هم که در شبکه های کامپیوتری هستن ، در قالب لایه به لایه طراحی شدن و به مجموعه ی اون ها Protocol layers یا stack میگن.
- 5 لایه در شبکه های کامپیوتری داریم : ۱-application ۲-transport ۳-network ۴-link ۵- physical
- چرا بهش میگیم stack ؟ به خاطر اینکه حالت پُشته ای داره ،یعنی اینکه پروتکل های هر لایه فقط با لایه ی بالا و لایه پایین در ارتباطن و از طریق رد و بدل کردن اطلاعات با لایه های مستقیم بالا و پایین خودشون کار می کنن.(سرویس از لایه پایین می گیرن و به لایه ی بالا

سرویس میدن) . ارتباط گرفتن با یه لایه ای که مستقیما با لایه ی فعلی در ارتباط نیست ، غیر مجازه!

- یه نکته ی کلی اینه که لایه های مختلف که باید کارهای خاص خودشون رو انجام بدن ، توسط اضافه کردن هدر به بسته هایی که دریافت می کنند،اون اعمالی که باید انجام بدن رو با طرف مقابل هماهنگ می کنن.

مثلا اون لایه ی اپلیکیشن که در مبدا هست میاد و به پیامی که میخواد ارسال کنه یه هدر اضافه می کنه ، و اون لایه ی اپلیکیشنی که درمقصد قرار داره با داشتن هدر، اون درخواست هایی که باید انجام بده رو بررسی کنه و اطلاعاتی که لازم داره رو به دست بیاره.

- Application layer : برنامه های کاربردی مختلف رو پشتیبانی می کنه و پروتکل هایی که توی این لایه ان برنامه های مختلف رو پشتیبانی می کنن، مثلا HTTP برای اپلیکیشن وب هست ، IMAP و SMTP برای اپلیکیشن ایمیل هستن ، و DNS برای اپلیکیشن ترجمه ی Domain name یا IP هست.

- وب سایت www.example.com یه وب سایته که با پروتکل http کار می کنه و تعامل باهاش ساده ست. برای اینکه بتونیم محتوای این وب سایت رو توی کامند لاین ببینیم ، می تونیم یه ارتباط TCP توسط اپلیکیشن telnet با وب سرور برقرار کنیم، برای برقراری این ارتباط باید پورت رو هم مشخص کنیم(مثلا پورت 80) :

telnet www.example.com 80

بعد از این دستور میتونیم با سرور ارتباط برقرار کنیم و صحبت کنیم و این صحبت باید در قالب پروتکل http باشه.

برای دریافت یه url ، از متد Get استفاده می کنیم ، بعد از اون آدرس home اون وب سرور رو وارد می کنیم(/index.html) ، بعد از اون باید نسخه ی پروتکل http رو مشخص کنیم(HTTP/1.1) ، و در خط بعد باید آدرس host اون وب سرور رو مشخص کنیم :

```
GET /index.html HTTP/1.1
Host: www.example.com
```

و بعد دوبار اینتر رو میزنیم.

بعد ازین کار ، محتوای اون سایت در قالب html برای ما ارسال میشه.

قسمت اول پیامی که دریافت می کنیم در قالب http هست و اطلاعاتی مثل وضعیت در خواست ، فرمت اطلاعات ، طول پیام ،زمان آخرین تغییر و ... .

به طور کلی یه سری meta data توسط http و هدری که توسط سرور اضافه شده ، دریافت می کنیم.

کاری که مرورگر می کنه اینه که اون قسمت هایی که مربوط به پروتکل http هست رو ازش استفاده می کنه و بعد اونو دور میریزه و فایل html رو به ما نشون میده.

- بنابراین در پروتکل http یه هدری ، با یک فرمت خاصی به پیامی که کلاینت قراره به سرور بفرسته اضافه میشه ، بعد متد درخواست مشخص میشه (فاصله) ، بعد url مشخص میشه (فاصله) بعد ورژن مشخص میشه (فاصله) و بعد carriage return و line feed (متناظر با دوتا اینتر) وارد میشن. و بعد هم سایر قسمت ها پر میشن. مثلا ما توی بعضی از متد های http ، داده هم داریم، مثلا وقتی متدمون POST باشه و از وب سرور می خوایم پیامی رو داخل url قرار بده ، درونِ body پیاممون خالی نیست و حاوی اون داده ایه که میخوایم ارسال کنیم.
- توی هدر ، یه url داریم که یه application layer address عه. مثلا توی مثال قبل ، url ما www.example.com/index.html بود. این نوع آدرسه که در سطح اپلیکیشن اون resource ای که ما می خوایم مشخص کنیم ، توسط این آدرس مشخص می کنیم.
- یه نکته : اون پیامی که مربوط به برنامه ی کاربردی مون هست ، داخل قسمت دیتا قرار می گیره و ما هدر http رو بهش اضافه می کنیم، برای اینکه http protocol سمت مقصد به meta data و فرامینی که احتیاج داره از طرق هدر بتونه دسترسی پیدا کنه.
- این بسته هایی که توی لایه ی اپلیکیشن هستن و متشکل از قسمت های دیتا و هدر یه پروتکل http ان ، به کل این بسته، Message

میگیم. توی لایه های مختلف ، اسم های مختلفی روی این بسته میذاریم تا بتونیم توی هرلایه واضح تر راجبش صحبت کنیم.

- پروتکل هایی که داخل application layer هستن ، فقط در داخل end system ها هستن. یک بسته برای اینکه از مبدا به مقصد برسه از روتر ها و سوییچ ها و .. هم عبور می کنه ولی این پروتکل های application layer به طور پیش فرض توی این تجهیزات وجود نداره و اون ها این message ما رو به شکل چندتا بیت می بینن و براشون قابل مفهوم نیست که برن هدر رو بررسی کنن و ببینن چه فرامینی داخلشون وجود داره.

به طور کلی این پروتکل های application layer برای مبدا و مقصد و end system ها طراحی شدن.

- Transport layer:

- برای انتقال پیام از application layer مبدا به مقصد ، از سرویسی که لایه پایینی یعنی transport layer میده استفاده می کنیم. این پیام دو تکه هست : ۱- Data ۲-Ha (header)
- یکی از سرویس هایی که transport layer به پروتکل های application layer میده اینه که در داخل یک end system ، ممکن هست اپلیکیشن های زیادی ران بشن و متناسب با هرکدومشون هم ما یک پروتکل اپلیکیشن داریم.

- لایه ی transport باید برای بسته هایی که از application layer دریافت می کنه اعداد متفاوتی در نظر بگیره که بهش میگن port number و با قرار دادن این عدد در هدر لایه ی transport end system طرف مقابل ، اطلاعات لازم برای اینکه بسته رو به اپلیکیشن مربوط به خودش برسونه ، انتقال بده.
- کار دیگه ای که لایه ی transport انجام میده اینه که بیاد کانال نا امنی که توسط اینترنت بین گیرنده و فرستنده ایجاد شده رو به یه کانال امن تبدیل کنه. همونطور که گفتیم اینترنت حالت Best Effort داره و تضمینی نمیده که بسته ها توی راه گم نشن و ترتیبشون بهم نریزه.
پروتکل TCP در لایه ی transport ، میاد یه مکانیزم هایی رو در داخل هدر تعبیه می کنه که توسط اون مکانیزم ها با این چالش ها مقابله کنه.
(یه پروتکل معروف دیگه به نام UDP هم داخل لایه transport داریم)
- شکل اسلاید ۱۰۵ ، ساختار پروتکل TCP هست.
ابتدا Source port و Dest port رو داریم که آدرس لایه ی transport هستن و به درد multiplexing و Demultiplexing بسته های اپلیکیشن های مختلف میخوره.
یه سری فیلد هم داخل این ساختار داریم که برای مقابله با چالش های کانال نا امن اینترنت هستن ، مثل sequence number . مثلا ما

بسته ها رو با ترتیب خاصی شماره گذاری می کنیم و اگه داخل شبکه ترتیب این بسته ها به هم خورد ، TCP ای که داخل end system گیرنده هست میاد و این ترتیب بسته ها رو بررسی می کنه و اگه به هم خورده بود، مرتبشون می کنه.

یا مثلا Acknowledgement number رو داریم که میاد اطلاعات بسته دریافتی رو بررسی می کنه و اگه این اطلاعات رو دریافت نکرد، بنا رو براین میذاره که بسته گم شده و دوباره بسته رو ارسال می کنه تا به این شکل از گم شدن بسته ها جلوگیری کنه .

قسمت Data هم شامل html file و HTTP هست که خودش ، encapsulate شده ی داده ی برنامه ی کاربردی هست که هدر HTTP بهش اضافه شده.

- حالا به کل این پیام که هدر TCP بهش اضافه شده (توی لایه ی transport) segment گفته میشه.
- پروتکل های لایه ی transport ، فقط توی end system ها وجود دارن و توی روتر ها و سوییچ ها و ... وجود نداره و این بسته ها و هدر ها براشون غیرقابل مفهوم هست.
- برای رد و بدل کردن اطلاعات بین پروتکل های مبدا و مقصدِ لایه ی transport، ما به پیاممون یه هدر اضافه کردیم که این هدر پیامی که از لایه ی قبلی گرفتیم رو encapsulate می کنه.

- Network layer :

- وظیفه ی اصلی این لایه اینه که بسته هایی که قراره از مبدا به مقصد برسن رو درست مسیریابی کنن تا نهایتا بسته ها به مقصد برسن.
- پروتکل هایی هم که داخل این لایه هستن برای مسیریابی درست بسته ها و پر کردن مقادیر جدول forwarding که توی روتر ها هستن، به کار میرن و بهشون routing protocols گفته میشه.
- یه پروتکل معروف توی لایه ی Network که پروتکل غالب هم هست، IP (Internet Protocol) نام داره. حتی بعضی مواقع به جای اینکه بگن لایه شبکه میگن لایه ی IP .

در این پروتکل ، فرمت هدر هایی که باید به segment های لایه ی transport بشه ، تعریف میشه و از طریق اطلاعاتی که در داخل این هدر ها قرار می گیره ، روتر ها میتونن بفهمن که بسته ها رو به کدوم یکی از پورت های خروجیشون ارسال کنن تا نهایتا بسته به مقصد برسه، و مثلا end-system مقصد هم با نگاه کردن به اطلاعاتی که در داخل هدر هست،میتونه بفهمه بسته مال خودش هست یا نه.

- اسلاید صفحه ی ۱۰۸ ساختار داخلی پروتکل IP رو نشون میده.

مجددا داخل هدر IP ،دوتا آدرس source و destination داریم که Network layer addresses هستن و جنسشون با url و port number که در داخل لایه های بالاتر بود فرق می کنه و وظیفه ی این آدرس ها اینه که host های مبدا و مقصد رو به صورت یکتا داخل

شبکه مشخص کنه. ماهم با نگاه کردن به این آدرس های IP توی روتر ها و matching ای که با یکی از رکورد های جداول forward اتفاق میفته متوجه میشیم که این بسته رو باید به کدوم یکی از پورت ها ارسال کنیم تا بسته به مقصد برسه .

قسمت Data هم در این ساختار متشکله از دیتای اصلی که داشتیم + هدر لایه ی اپلیکیشن + هدر لایه ی transport (مثلا TCP) . توی این مثال اسلاید ۱۰۸ فرض می کنیم دیتای اصلی مون فایل html هست.

یعنی اینجا مجددا encapsulation رخ داده و ما روی این بسته ای که دریافت کردیم یه هدر قرار دادیم تا بسته به مقصد برسه.

اسم جدید بسته ها در داخل لایه ی شبکه ، Datagram هست.

- برخلاف دوتا لایه ی قبلی ، پروتکل های لایه ی شبکه ، هم در داخل end system ها هستن و هم داخل روتر ها ، و اصلا روتر ها با توجه به اینکه داخل هدر پروتکل ها چی هست، میتونن هدایت بسته ها رو انجام بدن.(ولی در داخل سوئیچ ها نیستن)
- اون بسته هایی که از لایه ی transport سمت مقصد دریافت می کنیم که حاوی Message و هدر اون لایه هستن، در داخل لایه ی شبکه ، هدر این لایه رو هم بهش اضافه می کنیم و دیگه مثل لایه های قبل ، تجهیزات بین راه مبدا تا مقصد رو نادیده نمی گیریم و روتر ها مخاطب پیام هایی هستن که در داخل این هدر ها قرار دارن و با توجه

به IP Address های گیرنده و فرستنده تشخیص میدن که باید چه کاری رو انجام بدن. حتی ممکنه بسته هایی که دریافت می کنن ، هدر لایه ی network قدیم رو دور بندازن و هدر جدید جایگزین کنن. البته بعضی فیلد ها مثل IP Address مبدا و مقصد تغییری نمی کنن و سراسری هستن ، اما بعضی فیلد ها ممکنه آپدیت بشن، مثل فیلد Time-to-live . که هر روتری ttl بسته ای که دریافت می کنه رو یکی ازش کم می کنه و اگه صفر شد اون رو دور میریزه و اگه صفر نشد مقدار قبلی رو با مقدار جدید جایگزین می کنه ، بنابراین هروقت بسته روتر رو ترک کنه یه ttl جدید پیدا می کنه.

- Link Layer :

- وظیفه ی این لایه انتقال بسته ها بین نود های همسایه هست .
- مسیر از مبدا به مقصد ،تشکیل شده از یه سری نود و لینک هایی که این نود ها رو به هم متصل می کنه و بسته ی ما از طریق انتقال گام به گام بین این نود ها هست که به مقصد می رسه .لایه link بسته ها رو از این نود ها به نود همسایه شون انتقال میده.
- به خاطر این بهش link گفته میشه چون پروتکل های این لایه به لینکی که یه نودی رو به نود همسایه ش متصل می کنه خیلی وابسته هستن. مثلا اگه از زوج سیم یا فیبر نوری استفاده می کنیم ، link layer میتونه Ethernet باشه . اگه مثلا داریم از لینک رادیویی

استفاده می کنیم ، لایه لینک میتونه 802.11 ، WiFi ، یا LTE توی سلولار باشه .

- شکل اسلاید صفحه ۱۱۱ ، فرمت هدر و Ethernet ِtrailer رو نشون میده.

برخی از فیلد ها توی هدر Ethernet که به بسته هایی که از لایه ی لینک دریافت کرده اضافه میشن ، از جنس آدرسن.اما این آدرس ها جنسشون با port number ، IP Address و url متفاوته و برای مقصد دیگه ای هم هستن. این آدرس ها ، بهشون آدرس فیزیکی یا MAC هم میگن . این MAC Address ها گیرنده و فرستنده ی دو سر لینک رو مشخص می کنن. بنابراین مقادیر Dest Address و Source Address توی این لایه وقتی بسته از یه نود عبور می کنه ، تغییر می کنه.(الان مقادیر MAC Address ۴۸ بیت هستن)

- فیلد Data ،مقادیر encapsulate شده ی دیتای اصلی توسط یه سری هدر پروتکل هایی هست که قبل از اینکه بسته به لایه ی لینک برسه ، بهش اضافه شدن. پس اینجا Data شامل فایل html + هدر HTTP + هدر TCP + هدر IP هست.
- به بسته هایی که داخل لایه ی لینک هستن و هدر این لایه رو دارن و قسمت Data ی این بسته ها شامل دیتای اصلی + هدر لایه های قبلیه، Frame میگن.

- بعد از فیلد Data ، یه دنباله ای (Trailer) به بسته مون اضافه می کنیم که شامل یه سری بیت به نام CRC هست که وظیفه ی این بیت ها اینه که داخل این Frame، Error checking انجام بدن .
البته این Trailer در تمام پروتکل های لایه ی لینک وجود نداره .
- Link layer در تمام device ها و تجهیزات شبکه وجود داره ، هم در روتر ها هم در سوئیچ ها و هم در end-system ها. چون همه این ها باید بتونن بسته رو برای همسایه ی خودشون ارسال کنن.
- الان شکل encapsulation ما ، (اسلاید ۱۱۳) توی لایه ی لینک ، هدر مربوط به این لایه هم بهش اضافه شده و علاوه بر روتر ها ، توی سوئیچ ها هم این لایه وجود داره و این سوئیچ ها ، هدر ها و MAC Address ها رو میفهمن و میتونن بفهمن که بسته ها رو روی کدوم پورت ارسال کنن.

- Physical layer :

- وظیفه ای لایه اینه که بیت ها رو با شکل موج مناسب داخل اون مدیایی که device بهش متصل هست هدایت کنه. برای انتقال بیت هایی که دریافت می کنیم از پروتکل های لایه ی Physical استفاده می کنیم.

- توی یه پروتکل لایه ی Physical ، میتونه اینجوری تعریف شده باشه که مثلا توی یه فرکانس و پهنای باند خاصی، از امواج الکترومغناطیسی استفاده کنه. و مثلا ازین modulation استفاده کنه که اگه می خوایم ۱ بفرستیم شکل موج با سطح مثبت و اگه می خوایم صفر بفرستیم شکل موج با سطح منفی بفرستیم .(Binary encoding)
یا مثلا توی یه پروتکل دیگه به جای Binary encoding ، از Manchester encoding استفاده کنه ، به این شکل که به جای یه پالس، دوتا پالس می فرستیم ، اگه ۱ باشه ، تغییر(transition) از سطح مثبت به منفیه و اگه صفر باشه تغییر از سطح منفی به مثبت.
مزیت Manchester encoding نسبت به Binary encoding اینه که self clock هست. یعنی اینکه به ازای هر بیتی که داریم میفرستیم یه transition توی اون بازه ای که داریم پالس میفرستیم وجود داره و گیرنده ای که اون سمت لینک قرار داره با همین پالس میتونه clock بخوره و ببینه این پالس جهش به سمت پایین کرده یا جهش به سمت بالا ، و جهت این جهش، مشخص می کنه که ما صفر فرستادیم یا یک .
- در نهایت این بسته ای که تمام هدر های لایه های سمت مبدا بهش اضافه شده ، از طریق physical layer مبدا به physical layer مقصد برده میشه و بعد از اون ، به هرلایه ای که میره ، (به سمت بالای stack) از اطلاعات هدر لایه ی خودش استفاده می کنه و بعد چون

لایه ی بالایی به هدر لایه ی پایین احتیاجی نداره، اونو دور میریزه – که به این کار de-encapsulation گفته میشه – و بعد بسته رو به لایه ی بالایی انتقال میده. این روند به همین ترتیب ادامه پیدا می کنه و در نهایت فایل اصلی در اختیار برنامه ی کاربردی قرار می گیره.

- چند نکته راجب Layered Internet Protocol Stack :

- هرچی در پروتکل stack به سمت پایین تر حرکت می کنیم پیاده سازی ها به سمت سخت افزاری نزدیک تر میشه و هرچی به سمت بالاتر حرکت کنیم پیاده سازی ها بیشتر در نرم افزار انجا میشه. مثلا physical layer کاملا سخت افزاریه ، link layer و network layer حالت hybrid دارن یعنی یه قسمتایی سخت افزاری و یه قسمتایی نرم افزاریه ، transport layer و application layer هم به صورت ۱۰۰ درصد نرم افزارین.
- NIC(Network Interface Card) و WiFi dongle (یا LTE dongle) ماژول هایی هستن که لایه ی link و physical رو پیاده سازی می کنن. البته این ماژول ها بعضا CPU هم دارن که کارای نرم افزاری این لایه ها رو توش انجام میدیم.
- در داخل end system ها ،(kernel space) operating system داریم که پروتکل های لایه های transport و network توی operating system پیاده سازی میشن.

- اپلیکیشن ها و پروتکل های این لایه هم توی user space پیاده سازی میشن.
- این که پردازنده ها اجازه بدن چه دستوراتی روشون اجرا بشه در محیط های مختلف متفاوته ، مثلا اگر نرم افزار در kernel space باشه ، این قدرت رو داره که همه ی دستورات رو اجرا کنه . ولی نرم افزار هایی که توی user space هستن یه مقدار limited هستن ، و اجازه ی اجرای بعضی دستورات رو ندارن . این کار به خاطر این انجام میشه که اگه خواستیم resource رو در اختیار چندتا برنامه قرار بدیم ، ممکنه یه برنامه monopoly کنه ، و اون resource نتونه در اختیار بقیه ی برنامه ها قرار بگیره ، پس باید یه کنترلی روی برنامه ها داشته باشیم.
- برنامه ی Wireshark ابزاری هست که ما با استفاده از اون میتونیم بسته هایی که توسط کارت شبکه ی کامپیوترمون ارسال یا دریافت می کنیم رو capture کنیم و بعد packet inspection انجام بدیم، یعنی داخل بسته ها رو بررسی کنیم و هدر های اون ها رو ببینیم.