Chapter 3 : Transport Layer

- چیزایی که یاد می گیریم:

multiplexing , demultiplexing , reliable data transfer , flow control , congestion control

transport layer protocols : UDP & TCP

- ترتیب موضوعات :

- Transport-layer services
- Multiplexing and demultiplexing
- Connectionless transport: UDP
- Principles of reliable data transfer
- Connection-oriented transport: TCP
- Principles of congestion control
- TCP congestion control

• Transport services and protocols

- در مورد سرویسی که لایه ی حمل و نقل به لایه ی اپلیکیشن میده ، می تونیم بگیم که لایه ی حمل و نقل ، یک ارتباط logical بین process هایی که توی لایه ی اپلیکیشن هستن برقرار می کنه.

منظور از logical بودن، اینه که بین دوتا process که توی لایه ی اپلیکیشن دوتا سیستم مختلف هستن، ارتباط مستقیمی وجود نداره ، بلکه ارتباط فیزیکی از طریق لینک های مختلفی که توسط end system ها و روترها(که در مسیر وجود دارن) به وجود اومده، شکل می گیره . اما این جزئیات بر process ها (موقعی که دارن از سرویس های لایه ی حمل و نقل استفاده می کنن) پوشیده ست، و وقتی یه process در یه end system پیام خودشو تحویل لایه ی حمل و نقلِ اون end system میده ، می تونه فرض کنه که یه ارتباط مستقیم بین اون process و process طرف دیگه وجود داره و پیام هاش در نهایت در اختیار process ِ end system طرف مقابل قرار می گیره.

- پروتکل های لایه ی transport ،توی end system ها وجود داره و روتر ها و دستگاه های میانی به صورت اولیه پروتکل های این لایه رو ندارن.
- فرستنده ی لایه transport، پیام هایی که تحویل می گیره رو به یه سری segment می شکنه و براشون هدر قرار میده، بعد این segment ها رو تحویل لایه ی زیرینش ، ینی لایه ی شبکه میده.
- در سمت گیرنده ، وقتی segment ها دریافت میشن، لایه ی transport ِ گیرنده این segment ها رو کنار هم قرار میده (reassemble) و بعد در اختیار لایه ی اپلیکیشنِ گیرنده می ذاره.

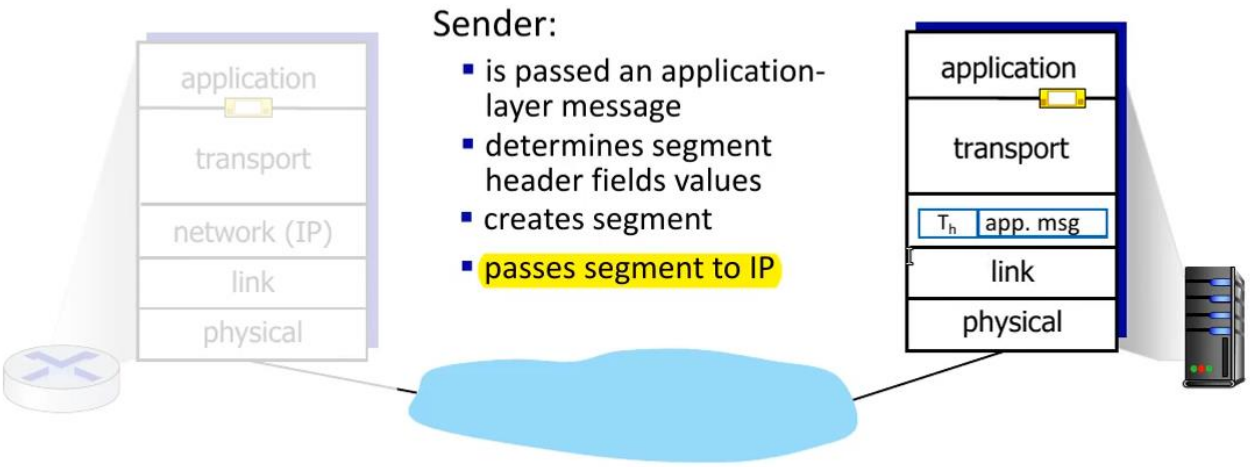
- ارتباط لایه ی شبکه و لایه ی حمل و نقل

- مثال : یک آپارتمان رو در نظر می گیریم که در این آپارتمان ۱۲ تا بچه هستن. این ۱۲ تا بچه خدمتکاری به اسم Ann دارن. یه آپارتمان دیگه هم داریم که ۱۲ تا بچه توشه و خدمتکارشون Bill نام داره. قراره بچه های آپارتمان اول به بچه های آپارتمان دوم نامه بنویسن. به این شکل که هر کدوم از بچه ها برای هر بچه ای بخواد نامه می نویسه و نامه شو تحویل Ann میده، Ann مشخصات فرستنده و گیرنده و آدرسشون رو نامه ها می نویسه. بعد نامه ها رو تحویل اداره ی پست میده. وقتی پست چی نامه ها رو به آپارتمانِ Bill رسوند، Bill هر نامه رو به دست بچه ی مربوطه می رسونه.
- شباهت این مثال و رابطه ی لایه ی شبکه و حمل و نقل به این شکله که عملکرد لایه ی شبکه شبیه به اداره ی پست و پست چی هاست، یعنی یه ارتباطی بین دوتا host ایجاد می کنن. کاری هم که خدمتکار ها انجام میدن شبیه کاریه که لایه ی transport انجام میده.(ارتباط بین process ها) یعنی در واقع لایه ی transport کاری که لایه ی شبکه کرده بود رو داره گسترش میده.
- لایه ی transport علاوه بر عملکرد اصلیش ، می تونه سرویسی که لایه ی شبکه بهش داده بود رو ارتقا هم بده. چطور ؟ ممکنه لایه ی شبکه بسته هایی که بهش می رسه رو گم کنه و نتونه تحویل لایه ی شبکه ی گیرنده بده. اینجا لایه ی transport می تونه از مکانیزم re-

transmission استفاده کنه و انقد این کارو تکرار کنه که بسته به لایه ی transport ِ گیرنده برسه.

البته این سرویس ِ enhance ، یه کار اضافه ست و مثلا پروتکل UDP این کارو انجام نمیده ولی TCP انجام میده.

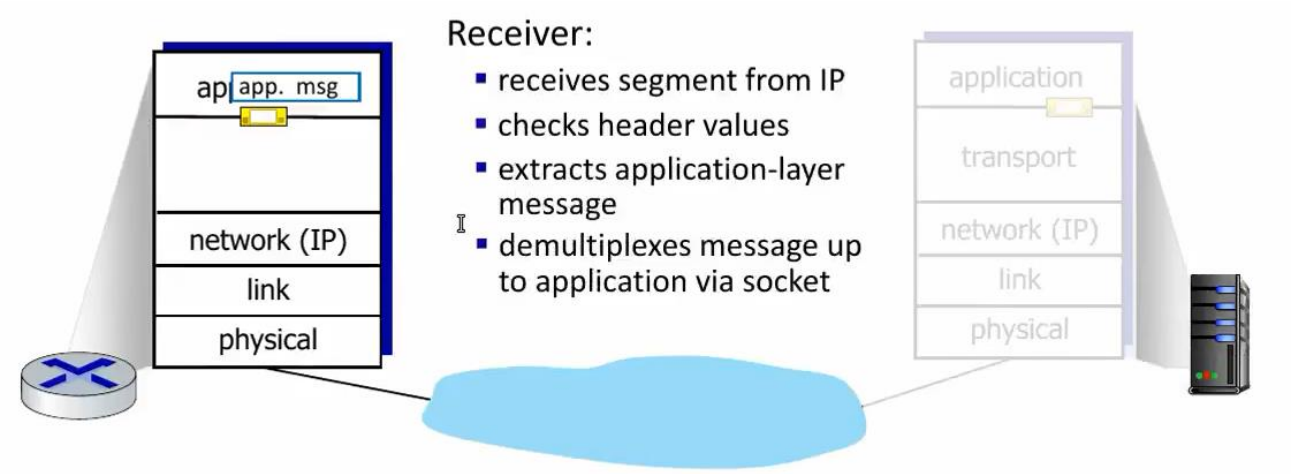
- عملکرد لایه ی transport (طبق توصیحاتی که دادیم) در سمت فرستنده :



- عملکرد لایه ی transport در سمت گیرنده :

- لایه ی transport در سمت گیرنده بسته رو از لایه ی IP (شبکه) دریافت می کنه ، هدر رو به دو دلیل چک می کنه: ۱- خطا رخ داده یا نه . ۲- توسط ID هایی که داخل هدر قرار داده شده ، متوجه بشه که این بسته مربوط به کدوم process هست و بسته توسط سوکتی که بین process و لایه ی transport ایجاد شده، در اختیار process قرار می گیره . البته قبل از اینکه پیام در اختیار process قرار بگیره،

لایه ی transport هدر بسته رو دور می ریزه و بسته رو extract می کنه و پیامی که توسط لایه ی اپلیکیشنِ فرستنده ارسال شده بوده تحویلِ process لایه ی اپلیکیشنِ گیرنده قرار می گیره.



- پروتکل های لایه ی transport

- TCP و UDP
- هر دوی این پروتکل ها ، سرویس multiplexing و de-multiplexing رو ارائه میدن. یعنی در کنار پروتکل IP که بسته ها رو از host ای به host دیگه منتقل می کنه ، ما توسط لایه ی transport میایم بسته ها رو بین process ها انتقال می دیم.
- در کنار عملکرد اصلی ای که لایه ی transport داره، اینکه TCP و UDP چه کارهای مازادی انجام میدن بستگی به شرایط داره.

UDP تقریبا به جز mux و demux (و بحث error checking که بعدا راجع بهش صحبت می کنیم) کار دیگه ای انجام نمیده .

اما TCP خیلی کارای بیشتری انجام میده. مثل :

1 - Reliable, in order delivery : یعنی اگه به هر دلیلی بسته ها داخل اینترنت گم بشن یا ترتیبشون بهم بریزه، TCP جبرانش می کنه و درنهایت بسته ای که به دست process لایه اپلیکیشن گیرنده می رسه ترتیبش حفظ شده و بسته ای گم نشده.

2 - Congestion control : در مواقعی که شبکه شلوغه، TCP به صورت داوطلبانه سرعت ارسال خودش رو پایین میاره.(اگه همه ی TCP ها این کارو انجام بدن اوضاع شبکه بهتر میشه ولی این موضوع ربطی به ارتباط بین end system های بین دوتا کانکشن TCP نداره)

3 - Flow control : TCP سمت فرستنده مطابق با حجم بافر TCP سمت گیرنده بسته ها رو ارسال می کنه .

4 - Connection setup : قبل از اینکه مبادرت به ارسال داده ها کنیم ، بین دوتا کانکشن TCP یه سری پیام های کنترلی باید رد و بدل بشه و در صورت موفقیت آمیز بودنش داده ها ارسال میشن.

- این کار ها توی UDP انجام نمیشن و UDP یه extension of IP "best-effort" هست ، یعنی اگه بسته ها گم بشن کاری انجام

نمیده و فقط سرویسِ host-to-host مربوط به لایه ی شبکه رو به process-to-process ، extend می کنه.

- یه سرویسایی هستن که حتی TCP هم اون ها رو ارائه نمی کنه ، مثل delay و bandwidth .

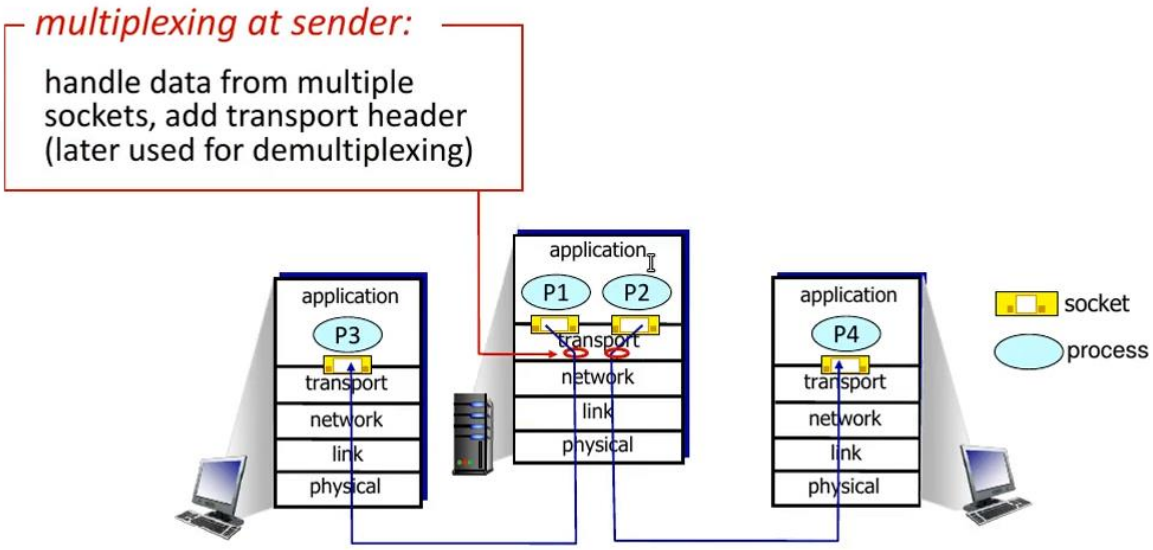
مثلا TCP نمی تونه تضمین کنه که تاخیر ارسال تا دریافت بسته از یه حدی کمتر بشه؛ چون این موارد در اختیار end system ها نیست و بعضی از این تاخیر ها مربوط به تاخیر صف در روتر ها هستن.

همچنین راجب پهنای باند هم نمیتونه ضمانتی بده. به خاطر این که سرعت ارسال بسته ها تابعی از شرایط شبکه و پهنای باند لینک های شبکه هست، و توسط کارهایی که توی end system ها انجام میدیم ، نمی تونیم ضمانتی برای سرعت ارسال بسته بکنیم.

- Multiplexing/demultiplexing

- مثال : دوتا host دارن با یه سرور ارتباط برقرار می کنن. فرض شده که process مربوط به hostِ اول اسمش P3 هست و یه سوکت برای ارتباط با لایه یtransport خودش داره. process مربوط به hostِ دوم هم اسمش P4 هست و اون هم یه سوکت برای ارتباط با لایه ی transport داره. روی وب سرور هم دوتا process به نام های P1 و P2 هست که سوکت های خودشون رو دارن و فرض شده که P1 با P3 و P2 با P4 در ارتباطه.

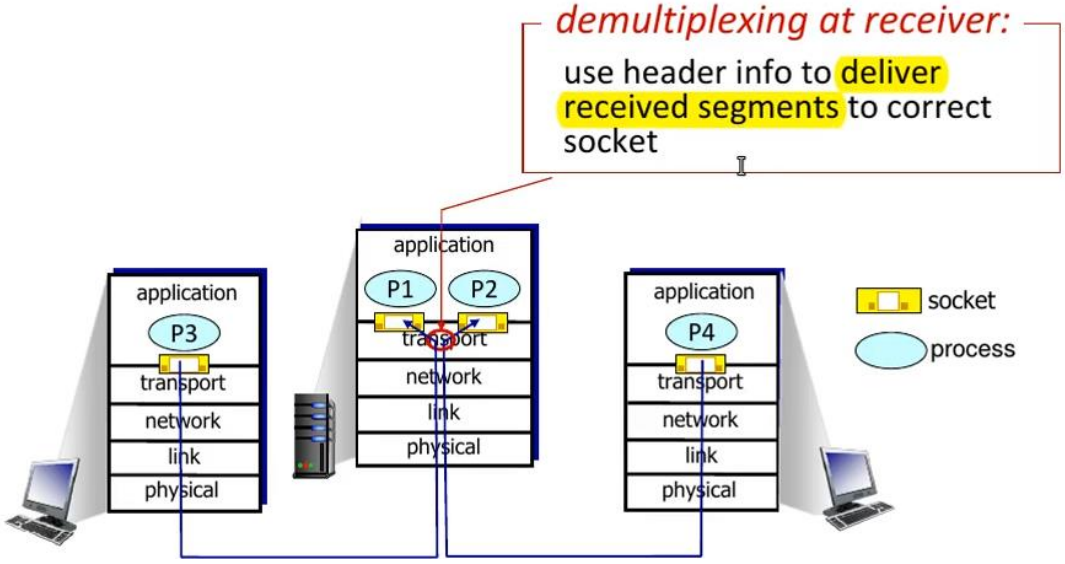
- عمل multiplexing در فرستنده انجام میشه و به این معنیه که وقتی process های مختلفی داخل یه end system داریم و این ها پیام های خودشون رو تحویل لایه ی transport میدن، لایه ی transport اون ها رو تکه تکه می کنه و هدر خودش رو بهشون الحاق می کنه و یک segment ایجاد میشه و بعد این segment ها رو ارسال می کنه. به این عمل تحویل گرفتن پیام ها از process های مختلف، اضافه کردن هدر به اون ها، ارسالشون به لایه ی شبکه ، multiplexing (MUX) گفته میشه.



- عمل demultiplexing در گیرنده انجام میشه. با استفاده از اطلاعاتی که داخل هدر segment ها هست، لایه ی transport گیرنده بسته ها رو به process مربوط به خودشون می رسونه. مثلا توی این مثال بسته هایی که از P3 ارسال شده ، در لایه ی transport با توجه به

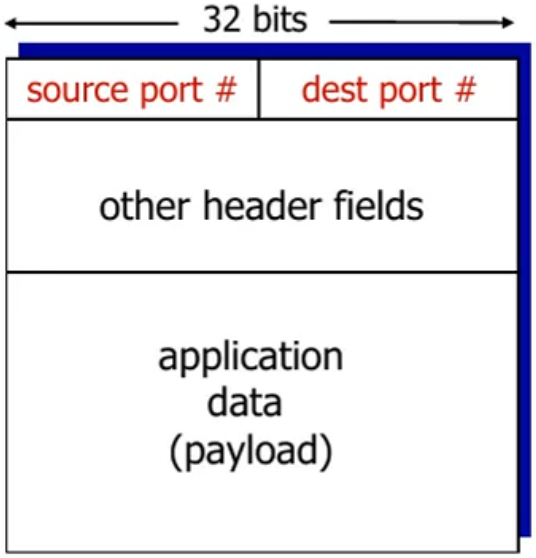
هدر بسته ها، برای P1 ارسال میشه. همچنین بسته هایی که از سمت P4 ارسال میشه بعد از اینکه در سمت لایه ی transport دریافت شدن، به P2 تحویل داده میشه.

پس demultiplexing یعنی استفاده از اطلاعات داخل هدر ها برای اینکه segment ها به socket متناظر خودشون تحویل داده بشن.



- برای انجام عمل demultiplexing فقط آدرس IP کفایت نمی کنه. IP Address مشخصات host رو تعیین می کنه اما برای اینکه ما بتونیم بسته رو به socket متناظرش برسونیم ،احتیاج به ID دیگه ای داریم که این ID توسط پروتکل های لایه ی transport (مثل TCP یا UDP) در داخل هدر لایه ی transport قرار می گیره، که به این ID ، میگن port number .

در نهایت این عمل demultiplexing می تونه مبتنی بر IP Address ها یا Port number هایی که داخل هدر بسته ها هستن انجام بشه. IP Address ها داخل هدر لایه ی شبکه و Port number ها داخل هدر لایه ی transport قرار دارن. شکل روبرو ، یه شمای کلی از یه segment ِTCP یا UDP عه :



TCP/UDP segment format

هردو پروتکل به طور مشترک فیلد هایی برای source port number و destination port number دارن.source port number داره به port number فرستنده اشاره می کنه و destination port number مربوط به process ِ گیرنده ست. هرکدوم ازین فیلد ها دو بایت هستن بنابراین مقداری که میتونن داشته باشن از 0 تا 65500 تغییر می کنه. از 0 تا 1023 رو بهشون میگن port number های well known یا معروف.(administrative port numbers هم بهشون میگن.) این port number ها برای سرویس های به خصوصی کنار گذاشته شدن. مثلا اگه یه end system ای قراره سرویس وب رو ارائه بده، باید port number اش 80 باشه تا کلاینت ها برای ارسال بسته هاشون بدونن چه port number رو قرار بدن.

سایر سرویس های معروف مثل ایمیل ، DNS و ... سرور هاشون وقتی میخواد راه اندازی بشه ، بهشون یه عدد مشخص تخصیص داده میشه و لیست ای ن اعداد در سایت IANA موجود هست.

به خاطر این به این port number ها administrative میگن که اگه یه end system ای یه process ای داخلش ران بشه که ازین port number ها استفاده کنه، باید اون کاربر، کاربرِ root باشه و اولویت super-user رو داشته باشه که پسورد بتونه وارد کنه و سیستم عامل بعد از احراز اصالت(که این کاربر، کاربر اصلی و ادمینه) اجازه ی استفاده از این port number ها رو میده.

- اطلاعاتی که تا الان گفتیم 4 تا بودن(IP Address گیرنده و فرستنده و Port number گیرنده و فرستنده) . آیا ما از همه ی این اطلاعات داخل هدر برای demultiplexing استفاده می کنیم؟

این موضوع چیزیه که شرایطش توی TCP و UDP باهم فرق می کنه.

- عمل demultiplexing در UDP

- پروتکل UDP برای عمل demultiplexing ، به IP Address و Port number ِ گیرنده توجه می کنه.(همین اطلاعات از فرستنده، اصلا مورد توجهش قرار نمی گیره) به عبارتی همه ی بسته هایی که از سمت فرستنده ارسال میشن ، اگه port number و IP Address گیرنده شون یکسان باشه ، همه تحویل یک process داده میشن.

این قضیه در عمل هم اتفاق میفته. وقتی که برای ایجاد یه سوکت UDP کدی می نوشتیم ، فقط IP Address و Port number خودمون رو به API ای که سوکت رو ایجاد می کرد می دادیم.(اگه هم نمی دادیم سیستم عامل خودش مقادیر پیش فرضی رو برای IP Address و Port number لوکال در نظر می گرفت)

بعد توسط این سوکت ما میتونستیم با process های مختلف که سوکتشون از جنس UDP بود هم صحبت کنیم. کافی بود همراه پیامی که میخواستیم برای اون process بفرستیم ، source & destination port number رو هم بفرستیم.

در سمت گیرنده هم وقتی پیام های UDP رو تحویل می گیریم، فقط به IP Address و Port number گیرنده توجه می کنیم و سوکت متناظر باهاش رو پیدا می کنیم و بسته رو تحویل میدیم.(یعنی فرستنده های مختلف بسته هاشون می تونه به یک گیرنده برسه)

- وقتی یه بسته ای به لایه ی transport رسیده یعنی بسته مال همین host بوده،چرا ما مجدد میایم از IP Address گیرنده به عنوان یک ID در کنار port number استفاده می کنیم تا سوکت رو تشخیص بدیم؟

جواب به این سوال اینه که بعضی از host ها چندین IP Address ممکنه داشته باشن یا به عبارتی multi-homing باشن. یعنی به ازای IP Address های مختلف می تونن سوکت های مختلفی داشته باشن.

بنابراین اگه یه host ای ، دوتا IP Address داشته باشه، میتونه یک port number یکسانِ UDP متناظر با این دوتا IP داشته باشه. پس از زوج IP Address و Port number به صورت یکتا یک سوکت رو مشخص می کنن.

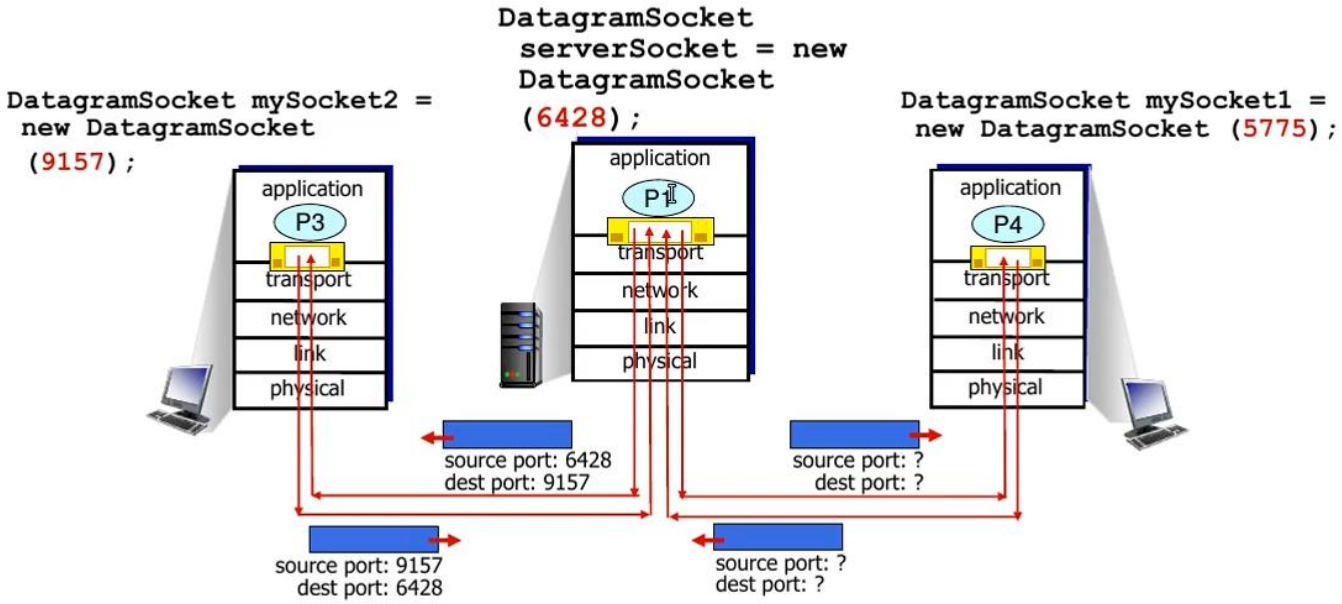
مثلا اگه یه لپ تاپ هم به WiFi و هم به شبکه ی Ethernet متصل باشه ، دوتا IP Address برای دریافت بسته ها داره و در این مورد باید هم به IP Address و هم به Port number برای تشخیص یه سوکت UDP به طور یکتا توجه کرد.

- مثال : در یه شرکتی که یه سرویسی از جنس UDP راه اندازی شده و اومدیم برای process وب سرور، یه سوکت UDP قرار دادیم. چون سرویسمون جزو سرویس های معروف نیست و لوکال هست، port number اش(6428) هم جزو Port number های معروف بین 0 تا 1023 نیست.

کارمندای این شرکت میتونن با این اپلیکیشن و با Port numberِ 6428 ارتباط برقرار کنن.

توی این مثال فرض کردیم سرور فقط یه IP Address داره و کفایت می کنه که فقط به Port number گیرنده توجه کنیم.

در واقع یک سوکت UDP داریم ، اما فرستنده های مختلف میتونن بسته هاشون رو به همون سوکت و همون process تحویل بدن.



این قضیه به طور برعکس هم میتونه اتفاق بیفته ، یعنی از یک سوکت UDP ، به گیرنده های مختلف بسته بفرستیم. نکته ی مهمش اینه که همراه پیام ، حتما destination port number و destination IP Address رو بفرسته. در گیرنده هم با توجه به destination port number بسته به process مربوطه تحویل داده میشه.

- عمل demultiplexing در TCP (یا به اصطلاح Connection-oriented demultiplexing)

- در UDP وقتی یه سوکت ایجاد می کردیم، اطلاعات طرف مقابل رو مشخص نمی کردیم و فقط IP Address و Port number محلی خودمون رو مشخص می کردیم و توسط این سوکت ، با هر process UDP دیگه ای می تونستیم صحبت کنیم.

این قضیه رو می تونیم تشبیه کنیم به لوله ای که فقط یک سرش فیکس شده و سر دیگه می تونه به جاهای دیگه وصل بشه . برای همین میگن UDP برای انجام کارش به ایجاد کانکشن نیازی نداره. چون وقتی میگیم یه کانکشن ایجاد شده که دو طرف ارتباط مشخص باشن. برای همین به عمل multiplexing از طریق UDP ، Connectionless demultiplexing هم میگن.

- بر خلاف UDP ، TCP ، Connection oriented عه. یعنی وقتی میخوایم یه کانکشن از جنس TCP ایجاد کنیم علاوه بر اطلاعات محلی خودمون ، اطلاعات process ای که میخوایم باهاش صحبت کنیم رو هم مشخص کنیم. (هم IP Address و Port number فرستنده، هم گیرنده)

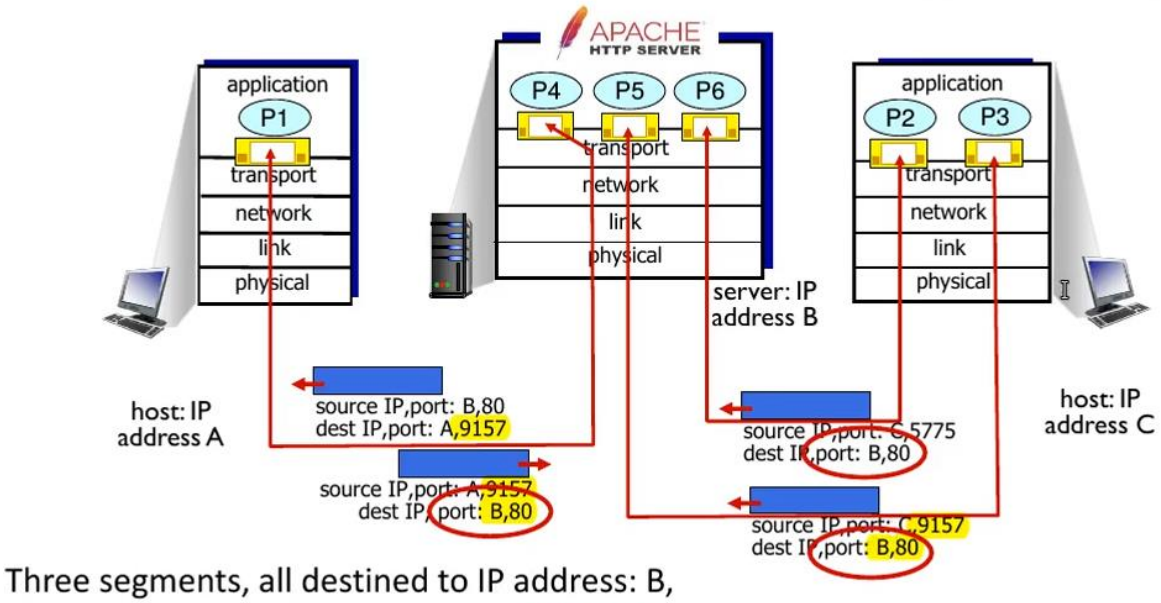
برای demux هم برخلاف UDP که مهم نبود اطلاعات فرستنده چیه، توی TCP اطلاعات فرستنده هم مهمه.

- حالا سوالی که پیش میاد اینه که چرا در ارتباط TCP ، برای هر فرستنده ای باید یه ارتباط TCP جداگونه داشته باشیم، یا به عبارت دیگه ، چرا هر دو طرف ارتباط باید مشخص باشن؟

چون در TCP ما state رو نگه داری می کنیم، یعنی به ازای هر ارتباطی باید بررسی کنیم بسته هایی که ارسال شدن به مقصد رسیدن یا نه، و اگه نرسیدن دوباره re-transmission انجام بدیم. به خاطر همین باید به ازای هر ارتباطی ، یه سوکت مجزا داشته باشیم.

- یک سرور TCP ، که داره به تعداد زیادی کلاینت سرویس میده، باید به ازای هر کلاینت یه سوکت(یا یک process) ایجاد کنه و ارتباطی که نهایتا بین کلاینت ها و سرور ها شکل می گیره از طریق همون سوکت یا process ایجاد شده.

- مثال از Connection oriented demultiplexing : یه وب سروری داریم که داره به سه تا process سرویس میده. (یکی در یک host و دوتای دیگه در یک host دیگه) هر بسته ای که میخواد از process کلاینت به process متناظرش در سرور بره، حتما اون ۴ تا مولفه بررسی میشه .



Three segments, all destined to IP address: B,
dest port: 80 are demultiplexed to *different* sockets

توی این مثال ، مولفه ای که باعث میشه مثلا TCP بدونه P3 باید به کدوم process بره و P1 به کدوم process ، source IP address توی بسته هاشونه. بسته هایی که از P1 ارسال میشن آدرس IP مبدا شون A عه و باید به سوکت P4 تحویل داده بشن ، ولی بسته هایی که از P3 ارسال میشن آدرس IP مبداشون C عه و باید به سوکت P5 تحویل داده بشن. به این ترتیب ما می تونیم بسته های این دوتا process رو از هم تفکیک کنیم.

- خلاصه :

- Mux و demux براساس اطلاعاتی که توی هدر های segment ها و datagram (بسته های لایه ی شبکه) ها وجود دارن ، انجام میشن.
- توی UDP برای عمل demultiplexing فقط با IP Address و port number گیرنده رو مورد توجه قرار میدیم.
- توی TCP برای عمل demultiplexing هم IP Address گیرنده و فرستنده و هم port number گیرنده و فرستنده مورد توجه قرار می گیرن.

- نکته : توی ساختاری که برای encapsulation و de-encapsulation بیان کردیم این طور بود که هر لایه ای که بسته

رو به لایه ی بالاتر تحویل می داد، هدر مربوط به خودش رو دور می ریخت و چیزی که باقی می موند رو تحویل لایه ی بالاتر می داد.
الان می بینیم که ضمن عمل multiplexing و demultiplexing این قضیه تا حدودی داره نقض میشه. چون به اطلاعات هدرها برای demux کردن احتیاج داریم. در واقع encapsulation به اون شکل آرمانی که ما انتظار داشتیم در عمل رخ نمیده.

- نکته: mux و demux به جز لایه ی حمل و نقل توی لایه های دیگه که لایه ی بالاترشون چندتا پروتکل داره هم رخ میده و فقط مختص لایه ی حمل و نقل نیست. مثلا تو لایه ی شبکه سمت گیرنده برای این که بفهمیم الان بسته رو کدوم پروتکل لایه ی حمل و نقل باید تحویل بدیم(TCP یا UDP) باید یه ID در هدر بسته ها داشته باشیم که اینو تشخیص بدیم.(به این ID میگن protocol number که برای پروتکل TCP مقدارش 6 و برای UDP مقدارش 17 عه)
- سوال تالار اعلانات : آیا ممکنه که دوتا process مختلف توی یک host ، port number های یکسانی داشته باشن؟