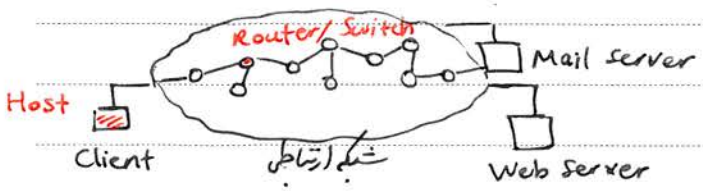


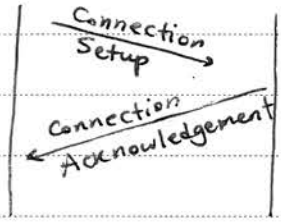
شبکه : از یک سری گره انتهایی تشکیل شده است که از طریق یک شبکه ارتباطی باهم مرتبط هستند. گره های انتهایی به کاربرها و application ها متصل هستند، به همین دلیل گره های انتهایی عموماً به Host (میزبان . app) معروفند.

* وظیفه شبکه انتقال پیام ها به Host انتهایی است.

Protocol : زبان مشترک یا قرارداد : مجموعه قوانینی که مشخص می کند برای انتقال درست داده باید چه کارهایی انجام شود.

Router/Switch : وظیفه انتقال اطلاعات و مسیریابی بر عهده دارند. سعی می کند بهترین مسیر را پیدا کند تا داده ها را منتقل کند. عمل switching (port ورودی را به port خروجی متصل می کند) را نیز انجام می دهد.

TCP: پروتکل لایه حمل



کیفیت سرویس : با delay، خطا و ... مشخص می شود.
اگر یک بیت عوض شود در هنگام انتقال، کل package به درد نمی خورد (هنگام انتقال فایل) ولی اگر در هنگام نمایش یک فیلم عوض شود خیلی به چشم نمی آید. پس کیفیت سرویس برای کاربردهای مختلف متفاوت است.

* بعضی از سرویس ها delay را می توانند تحمل کنند ولی خطا نه و برعکس! ← مسیریابی

(رسانه) medium فیزیکی : مثل هوا و ... که در صد نفوذ noise روی هیچ کدام از آنها صفر نیست ولی اگر noise از یک حدی تجاوز کند، خطا ایجاد می کند. خطا یک پدیده تصادفی است زیرا noise نیز تصادفی است. خطا با یک احتمال بیان می شود.

avg : از هر 10^6 بیت، یک بیت خطاست ← $p = 10^{-6}$

پروتکل های لایه حمل
- سرویس سریع UDP
- سرویس مطمئن TCP

هر سرویس انتقال احتیاج به پروتکل دارد. زیرا بستگی دارد، سرویس سریع باشد یا مطمئن.

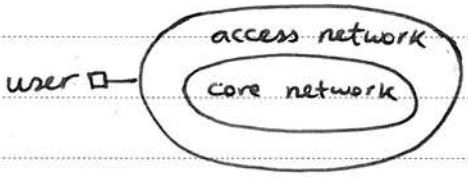
Check Bit : parity یک چک بیت ضعیف است . بعضی از چک بیت ها می‌تواند تشخیص دهند کدام بیت خراب شده است .

1. گیرنده اگر پرنده تایید نگرفت ، مجدداً ارسال می‌کند . زیرا در صورتی که Ack. نرسد یعنی Data خراب بوده است و آن را دور می‌ریزد .
2. بعضی از Check Bit ها علاوه بر تشخیص وجود بیت خطا ، خطا را درست می‌کنند .

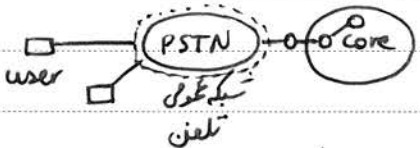
* چه Check Bit خطا داشته باشد ، چه خود داده ، گیرنده داده را دور می‌ریزد .

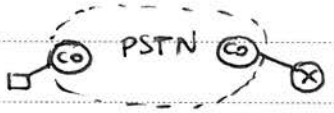
- گره های میانی می‌توانند شکل ظاهری را تغییر دهند ولی محتوای داده را تغییر نمی‌دهند .

شبکه: هسته core ، دسترسی access (wire-less, wired)



. Dial up Network : (wired)

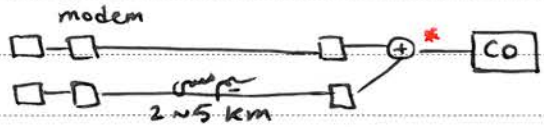




در PSTN چون از شبکه تلفن استفاده می‌شود ، اطلاعات باید مشخصات صوت را داشته باشد ، یعنی شکل data را شبیه به صوت می‌کنند . مشخصات یک سیگنال صوتی را دارد ولی اطلاعات دیجیتال است . به همین دلیل محدودیت سرعت دارد . هم‌چنین نرخ خطای بالایی دارد زیرا در صورت noise خیلی مهم نیست ولی باعث خطا می‌شود .

. Digital Subscriber Line (DSL) :

خطوط دیجیتال دسترسی به شبکه . XDSL (ADSL, HDSL, ...) در این شبکه ها مشترکین از طریق یک زوج سیم مسی با مودم های مخصوص کار عمل مدولاسیون را انجام می‌دهند و router ها وصل می‌شوند .



* router را جایی که control office وجود دارد وصل می‌کنیم ولزومی ندارد که مدولاسیون ما با صوت مطابقت داشته باشد چون وارد شبکه تلفن نمی‌شود ، پس می‌توانیم اطلاعات را با نرخ سریع‌تری ارسال کنیم .

ارسال هم‌زمان: تقسیم زمانی در یک بازه‌ی زمانی کلی ارسال شود ، در بازه زمانی بعدی ، دیگری ارسال شود .

تقسیم فرکانسی ADSL - صوت فرکانس پایین‌تر ، فرکانس بالاتر ⇐ بدون تداخل ارسال می‌شود . (خط تلفن اشغال نمی‌شود)

ADSL :

Asymmetric : نامتقارن (user های خانگی بیشتر request می دهند و اطلاعات را دریافت می کنند. پس این مودم را طوری طراحی کردند که نرخ دریافت بیشتر از نرخ ارسال باشد.)

* معمولاً مرکز تلفن محدوده ۲ تا ۵ کیلومتری خود را پوشش می دهد. هرچه فاصله ی سیم ها بیشتر شود، نرخ بیت کاهش پیدا می کند.

* هرچه پهنای باند بیشتر باشد، نرخ بیت هم بیشتر می شود. پهنای باند فیبر نوری بسیار زیاد است.

Cable Network : به هر user یک سر کابل تعلق می گرفت و روی هر کانال شبکه بندی انجام می شود (تلویزیون) امروزه برای استفاده از اینترنت با پهنای باند زیاد از این شبکه ها استفاده می کنند.

Fiber to the home : اگر شکلی در اتصالات فیبر نوری ایجاد شود، درست کردن آن هزینه ی زیادی در بر دارد.

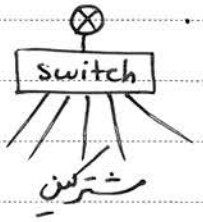
Wireless (Wi-Fi , wi_max)

یک دکل نصب می کنند Wimax ۴
می تواند یک پوشش رادیویی در محدوده بسیار بزرگی ایجاد کند. گیرنده های این امواج معمولاً جدا هستند در wimax که موبایل و ... به آن وصل می شوند.
بسیار ارزان قیمت است. سریع setup می شود و می تواند تا سرعت های بالا را پوشش دهد.
هر دو نوع Wireless به تعداد بستگی دارند، زیرا از یک کانال استفاده می کنند و از کل طیف فرکانسی نمی توانند استفاده کنند (زیرا توسط رادیو و ... اشغال شده است) و به همین دلیل درصورت زیاد شدن تعداد مصرف کنندگان نرخ بیت پایین می آید.

Ethernet

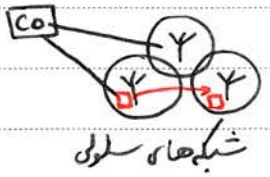
* IEEE : برای شبکه های دسترسی یک سری استاندارد مشخص می کند.

در شبکه Ethernet از یک سری سوئیچ استفاده می شود: مشترکین می توانند از طریق سوئیچ به یکدیگر و به router متصل شوند.



۴ زوج سیم { ارسال / دریافت
۲ زوج دیگر برای تجهیزاتی است که از شبکه استفاده می کنند. (تامین توان و ...)

پروتکل GPRS : شبکه های موبایل به این وسیله به اینترنت وصل می شوند .



BTS : دکل هایی که یک محدوده را پوشش می دهند .

CO : switch می کند user را از یک محدوده به محدوده دیگر !

Media فیزیکی :

. Twisted Pair : هر سیم مسی وقتی جریان از آن عبور می کند ، میدان مغناطیسی در اطرافش ایجاد می شود . اثر noise پذیری با پیچاندن سیم ها به هم کمتر می شود .

1. UTP (Unshield)
2. STP ⏚ noisex

* قطر سیم و طول سیم دو عامل اثرگذار در پهنای باند هستند .

. Coaxial Cable : کابل هم محور

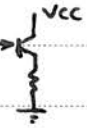
کابل های استفاده شده برای آنتن .

← اثر noise پذیری در کابل خیلی کم می شود :
پهنای باند با noise هم رابطه دارد .
نرخ بیت را تا جایی می توانیم افزایش دهیم که احتمال خطا کم باشد . ↑ پهنا ⇒ noise ↓

. Optical Fiber :

1. Single Mode → یک اشعه
2. Multi Mode → چند اشعه

اطلاعات را به صورت امواج نوری در می آوریم .
اگر یک LED قرار دهیم ، یک روشن و خاموش می شود .

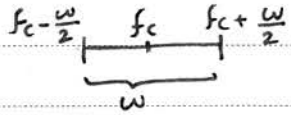


امواج نوری با تاخیرهای متفاوتی به گیرنده می رسند زیرا مسافت های مختلفی را طی کرده اند .

چون در یک نقطه تمرکز دارد می تواند به بافت های بدن آسیب برساند .

1. شبکه مخابراتی → گران تر ، نرخ بالا
2. شبکه محلی → ارزان تر ، نرخ پایین

Wireless از امواج رادیویی استفاده می کند و از رسانه هوا و فضا استفاده می کند و پهنای باند زیادی از چند صد هرتز تا چند تراهرتز دارد . این طیف وسیع را به یک سری باریکه های فرکانسی و باریکه های فرکانسی را به کانال های ریزتر تقسیم می کند تا بتوانند از تمام ظرفیت آن استفاده کنند . هر کانال به صورت زیر بیان می شود :



با استفاده از روش های مدلاسیون در یک بازه فرکانسی خاص می‌ریم و ارسال می‌کنیم. برای انتقال اطلاعات احتیاج به یک بازه‌ی فرکانسی خاص داریم و در wireless lan فرکانس 2.4GHz اختصاص داده شده است.

وقتی فرکانس افزایش می‌یابد، دوره تناوب کاهش می‌یابد: $T = \frac{1}{f}$, $\lambda = v \cdot T$

و هرچه طول موج کوتاه‌تر شود، امواج جهت دارتر می‌شود و احتیاج به دید مستقیم دارند و گیرنده و فرستنده باید در راستای هم باشند. به این امواج ماکروویو می‌گویند و از بشقاب هایی برای متمرکز کردن امواج استفاده می‌کنند.

Chanelize: FDMA, TDMA, CDMA

دو تکنیک برای انتقال اطلاعات در core داریم:

1. Circuit Switching: برای شبکه‌های قدیمی‌تر مورد استفاده قرار می‌گرد. برای برقراری ارتباط از داخل شبکه مسیریابی می‌کنیم و ممکن است از یک مبدأ به مقصد چندین مسیر داشته باشیم. ما بهترین مسیر را می‌یابیم و روی link فیزیکی که ارتباط برقرار کرده است یک کانال رزرو می‌کنیم. (با استفاده از کانال بندی - یکی از سه روش بالا)
این تکنیک در شبکه تلفن هم استفاده می‌شود.

اشکال: اگر نرخ ارسال فرستنده تغییر کند، ظرفیت کانال را باید حداکثر بگیریم تا بتوانیم حداکثر نرخ را داشته باشیم. پس در مواقعی که حداکثر را استفاده نمی‌کنیم داریم منابع را هدر می‌دهیم در نتیجه efficiency را از دست می‌دهد. این روش کارایی خوبی ندارد. در خط تلفن با این که می‌دانیم نرخ ارسال 64Kbps است، باز هم هدر می‌دهیم. چون فقط یکی حرف می‌زند و شنونده کانال خود را هدر می‌دهد.

2. Packet Switching (Store & Forward): اطلاعات را در قالب بسته هایی با یک حداکثر طول در می‌آوریم و اگر پیغامی بزرگتر از یک بسته است، آن را به چند بسته تقسیم می‌کنیم. روی بسته آدرس مقصد است. هیچ ظرفیتی اشغال نمی‌کند. هر گره بسته را می‌گیرد و برای گره‌ی بعدی forward می‌کند.
* بزرگترین حسن این است که از ظرفیت لینک ها به نحو مطلوب استفاده می‌کند.

اشکال: اگر بسته هایی که به یک گره می‌رسد از ظرفیت فیزیکی لینک بیشتر باشد، ازدحام روی می‌دهد. ازدحام یعنی ترافیک از ظرفیت عبوری بیشتر باشد. اگر ازدحام لحظه‌ای باشد مشکلی نیست، اما اگر دائمی باشد ما با شکل ترافیک داریم و buffer گره‌ها سرریز می‌شود و packet lost روی می‌دهد. در این صورت کیفیت سرویس از دید ابتدا به انتها پایین می‌آید و تاخیر نیز افزایش می‌یابد. برای رفع این مشکل باید مدیریت ترافیک صورت

بگیرد. اگر مجموع ترافیک ورودی از کل ظرفیت شبکه بیشتر باشد می‌توان مدیریت ترافیک داشت و ترافیک را کنترل کرد.

packet switching connectionless : در اینترنت داریم. مقصد هیچ اطلاعی از آن دیگر بسته ندارد. بدون ارتباط با گیرنده، packet ها را ارسال می‌کند.

packet connection-oriented switching : می‌توانیم با پیغام‌های کنترلی به گیرنده اطلاع دهیم و اگر گیرنده تأیید کرد، بسته‌ها را ارسال کنیم. مزایای آن نسبت به connectionless این است که گیرنده خودش را آماده می‌کند (با رزرو منابع) و خطا را کاهش می‌دهد. می‌توان هنگامی که پیغام داده می‌شود، مسیریابی کنیم و بسته‌ها را از بهترین مسیر ببریم. در connectionless بسته‌ها به صورت مستقل دیده می‌شوند و هر کدام ممکن است از مسیر جداگانه‌ای بروند.

← در مورد اول باعث ازدحام می‌شود اما در مورد دوم می‌توان پیش‌بینی کرد.

connectionless → Datagram
connection-oriented → Virtual Circuit
} این دو دیدگاه در network هستند.

شبکه را به دو قسمت می‌توان تقسیم کرد:

1. Core Network
2. Access Network

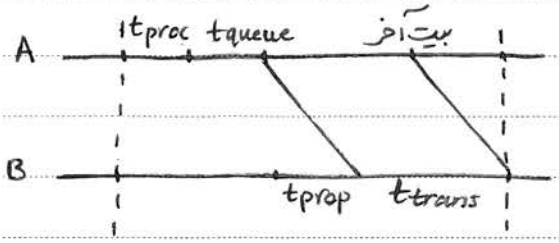
براساس معیارهایی می‌توان کیفیت شبکه را تخمین زد:

1. تأخیر delay : در packet switching از جمع تأخیرها ایجاد می‌شود. یک تأخیر پردازش در ورودی برای مسیریابی داریم. اگر دو بسته هم‌زمان برسند، باید یکی در buffer برود (t_{queue})، برای ارسال بسته ($t_{transmit}$) تأخیر ارسال سرعت لینک R kbps و تعداد بیت‌های بسته np است،

$$t_t = \frac{np}{R}$$

تأخیر انتشار (t_{prop}): این تأخیر به فاصله گره و جنس رسانه استفاده شده بستگی دارد.
اگر طول L متر و سرعت v m/s باشد، داریم:

$$t_{prop} = \frac{L}{v}$$



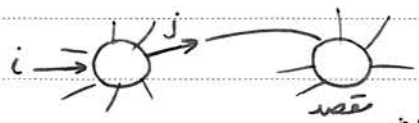
$$t_{total} = t_{proc} + t_{queue} + t_{trans} + t_{prop}$$

صبر کند به ترافیک دارد

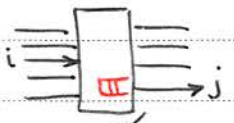
در circuit switching فقط t_{trans} و t_{prop} داریم، t_{proc} و t_{queue} نداریم.
در پارامترهای کارایی در بسیاری موارد میانگین را به لحظه‌ای ترجیح می‌دهیم.

Delay : یکی از پارامترهای کارایی است. تاخیر از گره ی ابتدایی تا انتهایی را حساب می کنیم :

۱) تاخیر پردازش یک بسته : یک گره هربسته ای را دریافت می کند، آدرس مقصد را می خواند و مشخص می کند که از چه پورتی خارج شود :



۲) تاخیر صف بندی : ممکن است برای پورت خروجی بسته های دیگری هم وجود داشته باشد. پس در یک صف برای پورت خروجی قرار می گیرند.



۳) تاخیر ارسال : n بیت حجم بسته است و با نرخ R bps ارسال می شود. پس در چه زمانی ارسال صورت می گیرد؟ $\frac{n}{R}$

۴) تاخیر propagation : اگر سرعت انتشار موج در رسانه ی فیزیکی V m/s باشد. $t_{prp} = \frac{L}{V}$ که L طول است.

Packet Loss : از پارامترهای دیگر کاربردی است. از دلایل از بین رفتن بسته ها، محدود بودن بافرهای خروجی است. اگر به طور لحظه ای بسته هایی که به یک پورت خروجی می روند از ظرفیت بافر بیشتر شوند، از دست می دهیم. اگر از یک حدی بیشتر شود دیگر قابل قبول نیست :

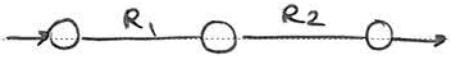
$$\text{Packet Loss} = \frac{\text{تعداد بسته های دور ریخته شده}}{\text{تعداد کل بسته ارسال شده}}$$

که این عدد محاسبه می شود تا چه حدی قابل قبول است.

برای delay، delay های هر گره را حساب می کنیم و سپس برای همه ی گره ها را جمع می کنیم تا delay از ورودی به خروجی بدست بیاید.

برای Packet Loss، باید تعداد گره ها را با هم جمع کنیم.

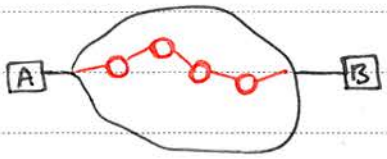
Throughput : می گوید از روی یک لینک، چقدر بسته می توانیم عبور دهیم. نرخ عبوری را به ما می دهد. از ابتدا تا انتهای مسیر که از چندین گره تشکیل شده باشد، می نیم نرخ ها، نرخ کل را مشخص می کند.



$$R_t = \min(R_1, R_2)$$

معماری سیستم : اجزای سیستم چه هستند و ارتباطشان با یکدیگر چگونه است.

برای انتقال یک فایل از A به B : در روش packet switching اگر حجم فایل بیشتر از حجم بسته باشد، آن را به بسته های کوچکتر تقسیم می کنیم. سپس از یک مدیای فیزیکی عبور می دهیم. برای عبور باید آن را به یک موج تبدیل کنیم.

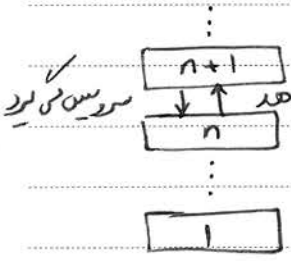


ممکن است موج به علت noise تغییر شکل دهد و ما با خطا مواجه شویم.
هنگامی که به مقصد رسید، بسته های کوچکتر را بازیابی می کنیم تا بسته اصلی تشکیل شود.
اگر رمز دارد باید گیرنده رمزنگاری را هم انجام دهیم. یعنی در فرستنده نحوه نمایش اطلاعات را تغییر دهیم که در بین راه کسی نتواند از اطلاعات استفاده کند.

عملیات زیادی را انجام می دهیم که پیغام را از مبدأ به مقصد منتقل کنیم. از تمام این کارها می توان یک معماری بدست آورد. معماری پیشنهادی باید مقبولیت داشته باشد، زیرا شبکه یک معماری توزیعی دارد و معماری ما باید در دو سیستم ابتدایی و انتهایی یکسان باشد تا بتوانند با هم کار کنند و توزیع را انجام دهند:

ISO
ITU
IETF برای اینترنت
⋮

این معماری ها از ایده ی لایه بندی استفاده می کنند. در معماری لایه ای وظایف را کاملاً تفکیک می کنیم و هر لایه وظایف خودش را انجام می دهد و به سرویس لایه ی بالاتر پاسخ می دهد و جزئیات سرویس ها از دید لایه ی بالاتر مخفی است.



هر کس وظیفه ی خود را انجام می دهد و به لایه ی بالاتر سرویس می دهد:
لایه ی بالایی یک دید abstract دارد. لایه پایینی خود از لایه ی پایین تر ش یک سرویس می گیرد تا بتواند به لایه ی بالایی خود سرویس بدهد. البته این دو سرویس ممکن است به هم مرتبط نباشند. مثلاً یکی مطمئن و دیگری نامطمئن باشد.

*** تغییرات راحت تر:**
اگر پردازش داخل یک لایه را تغییر دهیم، تغییری در لایه های بالاتر به وجود نمی آید. این از مزایای عمده ی لایه بندی است زیرا تغییرات به راحتی امکان پذیر است. لایه ی شبکه، لایه application، لایه ی فیزیکی و... ممکن است تغییر کند (Encapsulation).

* **سادگی پیاده سازی** : یک لایه فقط سرویس هایی را انجام دهد، یک لایه فقط کنترل دگره های انتهایی را انجام دهد و...

* **نگه داری ساده تر** : به راحتی می توان اشکالات را پیدا کرد که ناشی از وظیفه کدام لایه است.

- سربار بالا : اشکال عمده آن است. ممکن است برخی از عملیات لایه‌ی پایینی مورد نیاز لایه‌ی بالاتر نیست و یا مثلاً اگر خود یک لایه، یک سری عملیات را انجام می‌داد، بهینه‌تر بود. (efficient نیست)

← ولی مزیت‌ها بیشتر است، هزینه کم‌تر، پیاده‌سازی بهتر و ...

← تفاوت مدل لایه‌ای با مدل component ای این است که تنها هر لایه با لایه‌ی بالایی و پایینی خود در ارتباط است.

OSI: مدل لایه‌ای Open System Interconnection را پیشنهاد می‌دهد (reference model):
مدل مرجع OSI: شبکه‌های کامپیوتری سیستم‌های باز هستند که از نظر جغرافیایی به هم مرتبط هستند.

: مدل 7 لایه‌ای

7 Application
6 Presentation
5 Session
4 Transport
3 Network
2 Data Link
1 physical

1. وظایف به 7 قسمت تقسیم می‌شوند. پایین‌ترین لایه physical است. وظیفه‌اش این است که بین یک medium فیزیکی که بین دو گره است، اطلاعات را به صورت رشته‌ای از بیت‌ها تحویل بگیرد و در گره‌ی مجاور همان رشته را تحویل بدهد (چگونگی تغییر به امواج الکتریکی، الکترومغناطیسی، نوری و ...)
تمام جزئیات medium فیزیکی در این لایه است و اگر تغییری در آن‌ها ایجاد شود، این لایه تغییر می‌کند.

2. لایه‌ی بعدی پیوند داده است. وظیفه‌ی آن این است که یک رشته بیت را گره از لایه‌ی Network بگیرد و در گره‌ی بعد به Network تحویل بدهد. سنگرون سازی را در رشته بیت انجام می‌دهد و اگر خطایی رخ داده است تشخیص و در صورت لزوم تصحیح کند. (به physical می‌دهد که رشته بیت تبدیل شود و سپس تحویل بگیرد)

3. لایه‌ی شبکه وظیفه دارد یک بسته را از مبدأ می‌برد به مقصد تحویل دهد. وظیفه‌ی مسیریابی را برعهده دارد. مسیرها را انتخاب کرد، گره‌ی بعدی مشخص می‌شود. سپس بسته را به Data Link می‌دهد و سپس تحویل می‌گیرد.

4. لایه‌ی Transport وظیفه‌ی End to End دارد. اگر پیغام بزرگ است به قسمت‌های کوچک‌تر تقسیم می‌شود و در گره‌ی مقصد بازیابی می‌شود. اگر بسته‌ها lost شدند، وظیفه دارد پیدا کند و از فرستنده بخواهد دوباره آن‌ها را ارسال کند. وظیفه جایابی بسته‌ها را هم برعهده دارد، جابجا شده باشند: سرویس 3ل

* گره‌های میانی فقط 3 لایه‌ی پایین را دارند.
* گره‌های انتهایی هر 7 لایه را دارند.

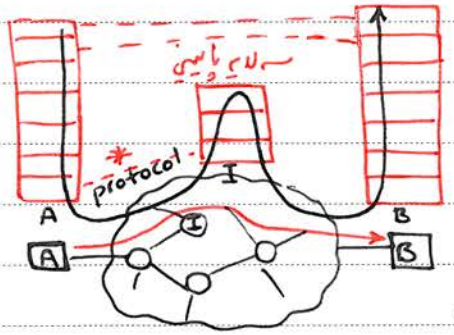
hop-by-hop: گام به گام کنترل خطا انجام می دهد. از وظایف Data Link است.

5. لایه‌ی session (یک ارتباط connection را کنترل می‌کند): ممکن است از گره‌های انتهایی آدرس‌های منطقی را دانسته باشیم نه فیزیکی. این تبدیل آدرس را انجام می دهد. شروع و خاتمه‌ی جلسات را مشخص می‌کند.

6. لایه‌ی ارائه وظیفه‌اش این است که نحوه نمایش اطلاعات را درست انجام بدهد. اگر ما یک coding را منتقل کنیم و در coding مقصد یک زبان دیگر باشد، ارائه اطلاعات بهم می‌خورد. همه‌ی اطلاعات را به یک شکل واحد درمی‌آورد، گیرنده هم همه را به شکل مورد نظر گیرنده درمی‌آورد. وظیفه‌ی رمزنگاری و رمزگذاری را هم دارد.

7. وظیفه‌ی لایه‌ی کاربرد، ارائه سرویس به user است. بستگی به سرویس user است:
سرویس انتقال mail ← application mail
⋮
نیاز user وظیفه‌ی آن لایه است.

* بحث های امنیتی باید ببینیم که intruder کدام لایه‌ها می‌تواند دسترسی پیدا کند. دنبال راه حل می‌گردیم که چگونه از این تهدیدات جلوگیری کنیم. اگر نتوانیم از تهدیدات به طور کامل جلوگیری کنیم. حال اگر نفوذی اتفاق افتاد، چگونه detect کنیم. (Protection → Detection)
پس ممکن است در لایه‌ی Data Link هم رمزنگاری کنیم.



← گره چهار به بالا، وظایف در مورد گره‌های انتهایی است.
Protocol لایه‌ی فیزیکی:
هر لایه‌ای برای این که وظایف اش را به درستی انجام دهد با لایه‌ی متناظر خود توافق می‌کند. در لایه‌ی فیزیکی باید توافق کنیم با چه نرخی بیت ها را ارسال کنیم؟ coding چگونه است؟ هر بیت به چه شکل نوری تبدیل می‌شود و ...؟
← توافق بین دو گره‌ی مجاور
توافق باید روی رنگ زوج سیم هایی صورت بگیرد: مثلاً آبی برای دریافت است و سبز برای ارسال است.

D.L. وظیفه‌ی framing بر عهده دارد: هر بسته از کجا تا کجا را شامل می‌شود. برای این که مرز frame ها را بتوانند مشخص کنند یکسری بیت های کنترلی با pattern خاص در ابتدا و انتهای stream بیت ها قرار می‌دهد. (مثل parity)

ممکن است مدیای فیزیکی ما multi access باشد. در این نوع رسانه باید بفهمیم چه کسی ارسال کرده است، پس frame ها باید آدرس مبدا و مقصد داشته باشند.

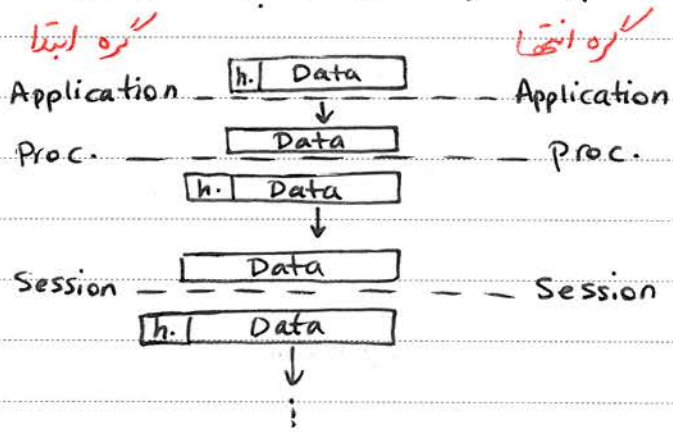
Data Link لایه protocol : header و trailer را به ابتدا و انتها اضافه می‌کند. هرچقدر حجم آنها بیشتر باشد سربار ایجاد می‌کند.

Protocol : توافقی است بین دو لایه‌ی متناظر از دو سیستم که در این پروتکل تعریف می‌شود چه اطلاعاتی به عنوان header داده می‌شود و چه کارهایی باید انجام شود تا پروتکل بتواند سرویسش را فراهم کند. این داده‌ها چگونه پردازش شوند و ...

پروتکل FTP : اگر فایل transfer بخواهد به درستی انجام شود و چه بیت‌های کنترلی مورد نیاز است. (پروتکل لایه Application)

(سرویس* = انتقال فایل)

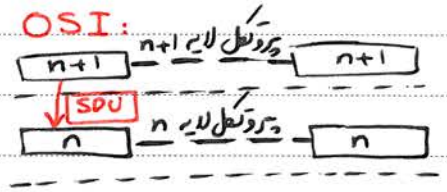
* هر پروتکل سرویسی انجام می‌دهد!



مثال: n لایه، هر لایه h بیت اضافه کند و data خود D بیت باشد، کارایی سیستم:

$$\eta = \frac{D}{D+nh}$$

← دقیق‌تر این است که برای هر لایه سربار را جداجدا حساب کنیم و سپس برای محاسبه‌ی کارایی کل، کارایی هر لایه را در هم ضرب می‌کنیم.

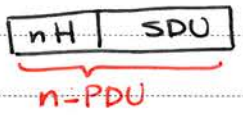


دید دیگر نسبت به دید 7 لایه‌ای:
هر لایه‌ای با لایه‌ی متناظر خودش در ارتباط است:

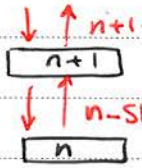
واحد اطلاعات پروتکل : header + data (protocol data unit)

در اینترنت IP : یک بسته‌ی IP درست می‌کند همان واحد اطلاعاتی لایه‌ی IP است.

← برای انتقال protocal data unit باید از لایه‌ی پایین‌تر سرویس بگیرد. لایه‌ی پایین‌تر آن را service data unit می‌گذارد و به آن header خودش را اضافه می‌کند:



سپس آن را به لایه‌ی بالاتر می‌دهد:



Service Data Unit به واحد اطلاعاتی‌ای روی interface می‌گویند.

* هر لایه برای لایه بالاتر می‌تواند سرویس‌هایی (با توجه به کیفیت) انجام دهد.

1. Connection_Oriented Service Model: یک لایه می‌تواند هر دو نوع سرویس را فراهم کند ولی حداقل
2. Connection less " " : یکی از این سرویس‌ها را ارائه می‌دهد.

اگر لایه‌ای روی سرویس‌هایی نوع داشته باشند، لایه‌ی بالایی سرویس مورد نظر خود را انتخاب می‌کند.

1. اگر لایه‌ی بالاتر درخواست ارسال داده کرد (n+1 درخواست ارسال داده کرد برای لایه n+1 تناظر) این داده بلافاصله ارسال نمی‌شود. ابتدا به طرف متناظر درخواست داده می‌شود که این داده می‌خواهد برای او ارسال شود یا نه.

Connection Setup

Data Transfer

Connection Release

* معمولاً سرویس‌های connection-oriented سرویس‌های قابل اطمینان هستند. (به دلیل بالا)

→ buffer‌ها را آزاد می‌کنیم که در اختیار بقیه قرار گیرد.

پس از تایید درخواست ارسال:
لایه‌ی n، لایه‌ی n+1 مبدا را به n+1 مقصد وصل می‌کند = سرویس

پس از رد درخواست ارسال: لایه n به n+1 منتقل می‌کند و او باید تصمیم بگیرد. (می‌تواند مدام درخواست دهد یا نه)

2. هیچ ارتباطی برقرار نمی‌شود و گیرنده هیچ اطلاعاتی در مورد این که ارسالی صورت می‌گیرد ندارد. تنها ارسال صورت می‌گیرد.

* در Connection-Oriented امکان ندارد مقصد اشتباه شود ولی Connectionless ممکن است مقصد را اشتباه کند. ممکن است مقصد خارج شده باشد از شبکه یا خاموش (down) شده باشد.

← در سرویس Connectionless اطمینانی وجود ندارد.

Reliable : مطمئن

Best Effort : بیشترین تلاش = غیر مطمئن (ممکن است انجام نشود، حتی با بیشترین تلاش)

لایه وظیفه اش را به درستی انجام می دهد ولی تضمینی به ما نمی دهد.

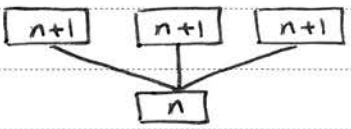
← در صورت پیدا کردن مقصد Data از بین می رود ولی پیغام خطای از دست رفتن Data اگر ما بخواهیم برایمان ارسال می شود.

layer management

— هر لایه علاوه بر سرویس هایی که وظیفه ی تامین آنها را بر عهده دارد، یک سری سرویس های جانبی هم در اختیار لایه ی بالاتر خود قرار می دهد:

- Multiplexing and Demultiplexing

اگر لایه ی n این سرویس را ارائه می دهد، اجازه می دهد به طور همزمان چندین entity از لایه ی n+1 از لایه ی n سرویس بگیرند:

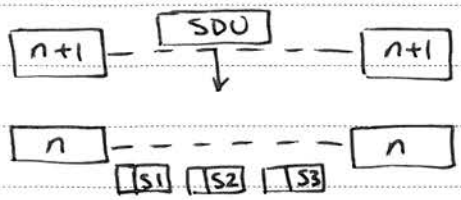


احتیاج به یک ID دارد. عمل multiplexing را با ID انجام بدهیم، زیرا باید بتوانیم داده ی هرکدام از entity ها را تفکیک کنیم به همین دلیل باید در header لایه ی n، ID قرار دهیم. در گیرنده می توانیم این ID ها را هم از هم تفکیک کنیم.

- Segmentation and Reassembly

ممکن است در یک protocol لایه ی n، برای Data Unit محدودیت اندازه بگذاریم. اگر Service Data Unit لایه ی بالاتر اندازه اش بزرگتر باشد چه کار باید بکنیم؟

1. اجازه ندهد Service Data Unit ای بدهد که اندازه اش بزرگتر از حد مجاز باشد.
2. لایه ی n، Service Data Unit را به چند segment تقسیم می کند، در سمت فرستنده. در سمت گیرنده هم Reassembly انجام می دهد و Segment ها را به هم متصل می کند.



← اگر Connection less باشیم و یک segment از بین برود، کل data از بین می‌رود.

← اگر connection oriented باشیم، با روش‌ها و مکانیزم‌های کنترل خطا (معمولاً وقتی داده‌ای را می‌فرستند، در حافظه نگه می‌دارند تا از گرفتن داده توسط گیرنده مطمئن شوند.) جبران می‌کنیم.

- Blocking and Unblocking

در فرستنده چند SDU را به یک PDU تبدیل می‌کنیم و در گیرنده دوباره به SDU تبدیل می‌کنیم.

: Flow Control

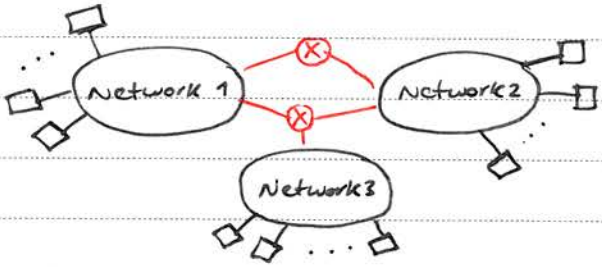
یکی از سرویس‌هایی است که لایه‌ها می‌توانند بدهند که معمولاً در لایه‌ی 2 (Data Link) یا در لایه‌ی 4 (Transport) انجام می‌شوند. به داده‌هایی که برای یک لایه‌ی n هستند و از یک مبدأ به مقصد می‌روند، یک **جریان ترافیکی** می‌گویند. جریان ترافیکی با یک نرخ یا سرعتی در حال گذر است. اگر گیرنده (لایه‌ی n+1) نتواند با سرعتی که داده‌ها می‌رسند، داده‌ها را پردازش کند (سرعت پردازش کمتر از سرعت رسیدن داده‌ها باشد.) دچار تجمع داده می‌شویم. داده‌ها در buffer منتظر می‌شوند و بافرها و صف‌ها سرریز می‌شوند و داده‌ها از بین می‌روند. پس همیشه سرعت رسیدن داده‌ها به گیرنده باید از سرعت پردازش داده‌ها کمتر باشد. به مکانیزمی که این حالت را تنظیم می‌کند flow control می‌گویند.

جریان ترافیکی می‌تواند بین دو گره و یا end to end (مبدأ به مقصد) باشد. در حالت end to end لایه‌ی transport عمل control flow را انجام می‌دهد. اگر بین دو گره باشد hop by hop ، Data Link این کنترل را انجام می‌دهد.

TCP/IP Model:

مدل لایه‌ای که در شبکه‌های اینترنت استفاده می‌شود. یک مدل 4 لایه‌ای است:

TCP/IP	OSI
Application	Application, Presentation, Session
Transport	Transport
Internet	Network
Network Interface	Data Link, Physical



Gateway / Router: گره‌هایی که به طور همزمان در چند شبکه هستند. هر گره‌ای که می‌خواهد به شبکه‌ای وصل شود باید Network Interface داشته باشد. (هر گره‌ای که می‌خواهد به اینترنت وصل شود ← به شبکه وصل شود ← به Network Interface وصل شود.)

* مثلاً وقتی با Dial-up به شبکه وصل می‌شویم، خط تلفن interface است.
* تمام Access layer ها، Network Interface هستند.
* هر گره‌ای که دو شبکه را به هم وصل کند، Network Interface است.

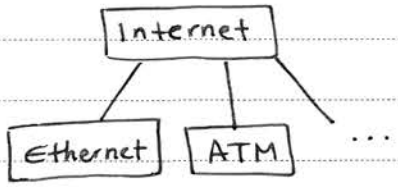
← در مدل TCP برای گره‌های میانی تا لایه‌ی Internet بالا می‌رویم.

استاندارد — رسمی: استانداردهای جهانی مثل ISO و ...
عرفی: استانداردهای defactor که به دلیل مقبولیت عمومی به صورت استاندارد درآمده‌اند. به علت استفاده زیاد از این شبکه‌ها، خودشان را به عنوان استاندارد قالب کرده‌اند در صورتی که توسط هیچ سازمان رسمی‌ای مدون نشده‌اند. مثل استاندارد Internet

* در اینترنت فقط سرویس‌های لایه‌ی Transport, Internet استاندارد هستند. برای Application و Interface خودکار به تصمیم می‌گیرد، یعنی قابل تغییر هستند. رمز موفقیت اینترنت هم همین است.

IETF: مسئول هماهنگ سازی بین کاربران است تا کاربران بتوانند به درستی از این استاندارد استفاده کنند.

* Network Interface لایه‌ی Internet ، هر شبکه‌ای می‌تواند باشد :



پروتکل انتقال فایل : FTP TCP

" " ایمیل : SMTP TCP

" " Real time : RTP UDP

" اینترنت : HTTP TCP

نیاز پروتکل ها برای انتقال متفاوت است . مثلاً FTP به خطا حساس است و اگر یک بیت فایل هم تغییر کند ، به درد نمی خورد . ولی پروتکل های Real time آن قدر دقت ندارند چون اگر بخشی از صوت یا ویدئو از بین برود ، خیلی مهم نیست . زیرا سیگنال صوتی یک سیگنال آنالوگ متغیر با زمان است و می‌توان در تبدیل آن به دیجیتال تخمین زد .

Sampling:

کوانتیزه می‌کنیم که خطا دارد . دگرگونه با یک تبدیل دیجیتال به آنالوگ دوباره به سیگنال آنالوگ تبدیل می‌شود : از یک سیگنال پایین اندرهم عبور می‌دهیم که تیزی ها از بین بروند .

اگر Sample ها را به توقع برگردانیم و زمان بندی را رعایت نکنیم ، صدا تند یا کند می‌شود . یعنی در سیستم های Real time زمان بندی بسیار مهم است .

Application

حساس به دقت ، در لایه‌ی transport به وسیله‌ی پروتکل transmission control protocol (TCP) Reliable می‌شود .

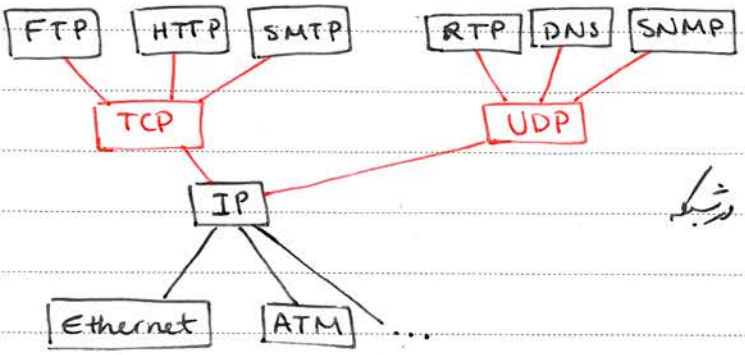
حساس به زمان بندی : TCP یکسری پردازش ها مبدأ و مقصد روی داده انجام می‌دهد تا ترتیب داده ها عوض نشود . که باعث کند شدن سرعت ارسال می‌شود و کم شدن کیفیت سرویس را به دنبال دارد . ← برای applications هایی که به delay حساس هستند یک سرویس دیگر توسط اینترنت اختصاص داده می‌شود :

User Datagram Protocol (UDP)

که مطمئن نیست و تضمین نمی‌کند ، ولی مکانیزم کنترلی ندارد و سربارش کم است پس سرعت بالاتر است .

به DNS query می‌دهید که DOMAIN Name را به IP Address تبدیل کند . این پیغام (query) خیلی کوتاه است و اگر بخواهید از TCP استفاده کنید ، سربار ممکن است بیشتر از حجم خود داده باشد . UDP هم تضمین

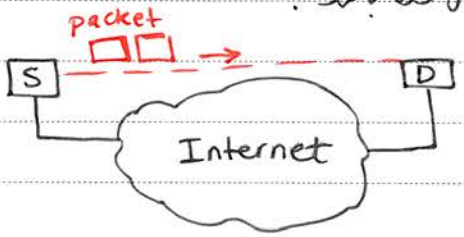
می‌کند. خود DNS منتظر دریافت جواب می‌ماند. پس خود سیستم کنترل دارد و نیازی به تضمین درستی جواب ندارد.
⇐ استفاده از UDP



* وظیفه‌ی IP انتقال بسته‌ها به صورت connectionless در شبکه اینترنت می‌باشد.

TCP: connection oriented, reliable است

و واحد اطلاعاتی که منتقل می‌کند بایت است. وقتی application از TCP استفاده می‌کند باید با مبدأ ارتباط برقرار کند و آدرس گیرنده را بدهد. TCP یک connection بین مبدا و مقصد برقرار می‌کند. بعد از انتقال، Application، bytestream به TCP می‌دهد. مانند این که یک port از مبدأ به مقصد وصل شده باشد:



عمل write: دادن به TCP که به گیرنده منتقل کند. TCP داده‌هایی که از Application می‌گیرد در بافر خود نگه می‌دارد تا به یک اندازه‌ی معقول (Segment) برسد که بتواند آن را ارسال کند. یک Segment حداکثر اندازه‌اش به اندازه‌ی یک پشته است (64 KB)
اگر Application بیشتر از این اندازه به طور یکدفعه داده بدهد، TCP آن را تقسیم می‌کند.

* پس آن چیزی که ما write می‌کنیم، همان را عیناً دریافت نمی‌کنیم ولی TCP ترتیب را به هم نمی‌زند.
* مرزبندی این که از کدام بایت تا کدام بایت یک پیغام است با خود Application است.

— در دو حالت TCP ممکن است Segment ای بفرستد که اندازه‌اش از max segment size کمتر باشد:

1. صبر TCP طولانی می‌شود و داده‌ای از طرف App. نمی‌آید، پس آن را می‌فرستد.
2. App. ، TCP را مجبور می‌کند که داده‌ای را بلافاصله ارسال کند.

Multiplexing: هر Application وقتی می‌خواهد به TCP وصل شود باید یک port number انتخاب کند. هر source برای برقراری connection علاوه بر آدرس گیرنده به port app. نیز نیاز دارد. app. یا خودش port No. را انتخاب می‌کند یا خود سیستم عامل این کار را انجام می‌دهد.
بیت کنترل خطا: دارد.

best effort : UDP
connectionless

واحد اطلاعاتی اش پیغام (message) است. محدودیت سایز آن به اندازه ی بسته IP (64KB) است.
اگر پیغام بزرگتر باشد، خود App باید این پیغام را به قسمت های کوچکتر تقسیم کند. از دید UDP هر پیغام یک موجودیت مستقل است و ترتیب و... با خود App باید مدیریت کند.
مثل TCP سیستم multiplexing را می دهد.

* روی کل پیغام یک بیت کنترل خطا دارد و پیغامی که خطا دارد دور ریخته می شود و به لایه ی بالاتر فرستاده نمی شود.
(optional)

best effort, connectionless : IP
واحد اطلاعاتی packet است.

TCP ارسال مجدد (در صورت خطا) دارد.
UDP پیغام را از دست می دهد. گیرنده خود بازسازی می کند.

در داخل header اش یک پروتکل دارد که مشخص می کند داده را TCP ارسال کرده است یا UDP که در مقصد به TCP بدهیم یا به UDP و سپس با port no. می فهمیم که مال کدام App. است.

Packet sniffing : روی router شبکه رفته و packet ها را گوش کنیم یا کپی کنیم.

* آدرسی که به کره ها می دهیم باید یک آدرس سراسری باشد که شناسند: یک عدد 32 بیتی که برای سادگی خوانش آن را به 4 عدد 8 بیتی تقسیم کرده اند که هر عدد را در مبنای 10 نشان می دهند.

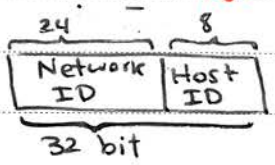
64.128.2.8
01000000 10000000 00000010 00001000

یک محدوده ی عددی خاص به آنها می دهند. (از یک عدد تا یک عدد)
* پیشوند آدرس (prefix) معرف شبکه است.

شبکه اینترنت از اتصال شبکه های کوچکتر به وجود می آید که از طریق Network Interface به هم مرتبط هستند.
آدرسی که به کامپیوترهای روی یک Network اختصاص می دهیم یک range خاص دارد.

prefix شبکه همیشه ثابت است. به قسمتی که ثابت است Network ID و قسمت دیگر Host ID می‌گویند:

مثال: یک شبکه:



78.100.10.0 تا 78.10.100.255

= 78.100.10/24

نشان می‌دهد چند بیت ثابت (prefix) است.

← نمایش دیگر: Network Mask 1111...1 0000...0

بقیه را صفر می‌گذاریم. به جای Network ID، 1 می‌گذاریم

فایده: اگر یک router بخواهد از شبکه، بسته‌ای را دریافت کند که مال یک شبکه باشد، کافی است بسته را با Network Mask، and کنیم. قسمت Host ID حذف می‌شود (چون با صفر and می‌شود). بعد نگاه می‌کند به Network ID چه عددی است. = Masking

* بیت صفرم آدرس خود شبکه است.
* آدرسی که تمام بیت‌های Host ID 1 است، بسته‌ی broad cast است، یعنی اگر به آن آدرس فرستاده شود، به همه‌ی Host ها می‌رسد.

← اگر Host بخواهد بسته‌ای را به Host دیگری بدهد باید از طریق Net بدهد. تمام Host‌هایی که در یک شبکه هستند از دید لایه‌ی اینترنت گره‌ی مجاور محسوب می‌شوند و نیاز به مسیریابی ندارند. برای این که بفهمیم نیاز به مسیریابی داریم یا نه، گره‌ی مبدأ، آدرس خودش و Network Mask اش را نگه می‌دارد. با آدرس گیرنده Mask انجام می‌دهد و آدرس مقصد روی همان Net است نیاز به مسیریابی ندارد ولی اگر روی همان Net نیست باید آدرس را به gateway بدهد تا مسیریابی کند.

اگر داخل همان شبکه بود به Network Interface می‌دهد تا به گره مقصد تحویل بدهد. آدرس‌های Network Interface آدرس IP را نمی‌شناسند چون آدرس‌های شبکه‌ای هستند، آدرس گره‌های روی Net را می‌شناسند.

Physical Add. شرکت سازنده قرار می‌دهد. Ethernet 48b است. معمولاً در مبنای 16 نمایش می‌دهند.

Network Add. IP آدرس.

Network Interface آدرس فیزیکی را می‌شناسد، پس باید تبدیل آدرس (از IP به فیزیکی) انجام شود.

(ARP) Address Reservation Protocol: این تبدیل آدرس را انجام می‌دهد.

در Ethernet ، هر گره‌ای داخل خودش یک جدول درست می‌کند (R-table). هر جدول دو قسمت دارد ، IP Add. و physical Add. (MAC) :

IP Add.	MAC
78.100.10.20	...

هر Host موقع فرستادن بسته به این جدول مراجعه می‌کند تا ببیند آدرس physical را دارد یا نه ، اگر ندارد یک پیغام به نام ARP Request درشبکه broadcast می‌کند. همه‌ی Host ها این پیغام را می‌گیرند. این پیغام می‌گوید که می‌خواهم بدانم کامپیوتری با این IP Add. ، چه physical آدرسی دارد. آن کامپیوتری که IP اش یکی است response می‌دهد و آدرس فیزیکی اش در این جدول قرار داده می‌شود و از این به بعد مشخص است.

Network Security :

تهدیدات اینترنت به دو قسمت تقسیم می‌شوند :

1- تهدیدات روی Host : از شبکه استفاده می‌کنند برای این که روی کامپیوترها انتهایی عملیاتی را انجام دهند. مثل ویروس ، worm ، trojan.
ویروس خود را به packet یا پیغام می‌چسباند و وارد سیستم می‌شود. مثلاً به ایمیل
worm خودشان ، خود را متصل می‌کنند ، port آزاد روی سیستم را پیدا می‌کنند و کپی می‌شوند و خود را اجرا می‌کنند. *

2. تهدیدات روی Network : عملیات Server را مختل می‌کنند و نمی‌گذارند بسته‌ها به مقصد برسند.

1) ممانعت بودن data را تهدید می‌کند و لطمه‌ای نمی‌زند فقط می‌خواهد اطلاعات را ببیند بیاورد.
2) تغییر و دستکاری روی data (integrity)
a) تهدید کننده خود را به جای یک سیستم معتبر جا می‌زند.
b) داده‌هایی که می‌رسند را دستکاری کند
3) از سرویس جلوگیری می‌شود : Hacker (جلوگیری ممانعت از سرویس)

* اسب Trova (Trojan) : خودش را به یک برنامه اجرایی می‌چسباند. اگر برنامه اجرا شود آن هم اجرا می‌شود.

راه های جلوگیری :

1. با شناختن virus های شناخته شده ، فایل‌ها را scan می‌کنیم. برای worm ها از firewall استفاده می‌کنیم (با بستن port های TCP یا UDP)

2. اگر در router ، packet sniffing انجام دهیم می‌توانیم داده را دستکاری کنیم.

3. در لایه‌های مختلف می‌توانند این کار را انجام دهند : از چه راه هایی می‌توان نفوذ کرد. مثلاً در لایه‌ی Transport یک نفر مرتب درخواست برقراری ارتباط دهد. server همه‌ی connection ها را باز کند دیگر نمی‌تواند به سایرین سرویس بدهد. یا در لایه Application می‌توانیم رفتار App. را بسنجیم و سرش را شلوغ کنیم که دیگر نتواند به درخواست های اصلی رسیدگی کند. برای مقابله اگر pattern رفتار attacker را داشته باشیم می‌توانیم جلوگیری کنیم. مثلاً یک port باز می‌کند و یا تعداد درخواست‌ها هم زمان را چک می‌کند.

Application: برنامه‌ای که به لایه‌ی انتهایی پیاده سازی می‌شود (در سیستم های انتهایی) و سرویس هایی را برای کاربران فراهم می‌کند. سیستم های انتهایی Host یا میزبان application ها هستند.

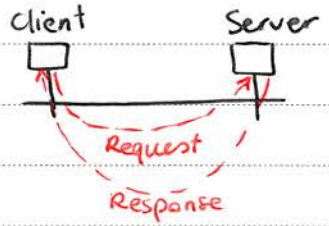
سیستم های انتهایی با هم تبادل اطلاعات برقرار می‌کنند تا سرویس فراهم شود (وظیفه application)

* web client ↔ web server

1. مدل Client-Server:

در این مدل، app. ها دو نقش می‌پذیرند
- Server: سرویس دهنده، source داده و فراهم کننده داده ها
- Client: مشتری، مصرف کننده

server هیچ‌گاه سرویسی را برای client نمی‌فرستد مگر این که client درخواست کند.



2. مدل Peer-to-Peer (P2P):

همتای یکدیگر هستند. یکی client نیست و دیگری server. در این مدل هم از یکدیر هستند: هم client و هم server هستند. یا می‌تواند pure پیاده سازی شود و یا ترکیبی. (Torrent, BTorrent و...)

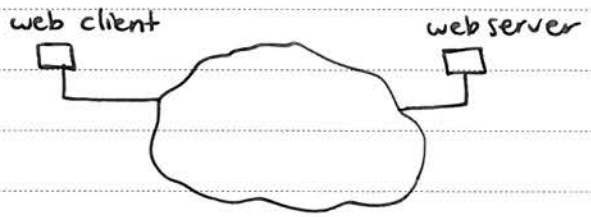
3. مدل Hybrid:

مدل ترکیبی از دو مدل بالا!

* هر App. باید یک مدل از سه مدل بالا را داشته باشد تا پیاده سازی شود. اکثراً Client-Server (اینترنت)

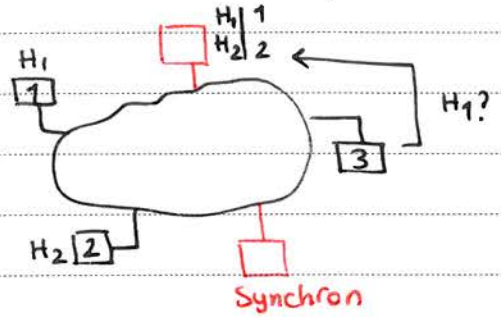
مثال: www (world wide web):

معمولاً server ها آدرس IP ثابت دارند و شناخته شده هستند!



P2P: از نظر آدرس IP، Port No. هیچ محدودیتی ندارند. هرکس در توان خودش اطلاعاتی را جمع کرده است در اختیار بقیه می‌گذارد. این نوع شبکه خیلی reliable نیست چون ممکن است سیستم‌ها بیایند و بروند. وقتی سیستمی خارج می‌شود با تمام اطلاعات اش خارج می‌شود.

Hybrid: هرکس وارد می‌شود اولین Server را پیدا می‌کند. یکسری داده برای به اشتراک گذاری دارد و با استفاده از یک hashtable درست می‌کند. این hashtable ها را به Server می‌دهد:



Server به 3 می‌گوید که به چه سیستمی مراجعه کند.
اگر هرکدام از سیستم ها خارج شوند، یکی یکی به همه مراجعه می‌کند.

* همه‌ی App. ها داده ای را از یک سیستم انتهایی به یک سیستم انتهایی منتقل می‌کند. لایه‌ی Transport این داده را حمل می‌کند.

نیازهای Application:

1. throughput — چقدر تند دهی باید نیاز داریم.
2. Data loss — چقدر انتظار از بین رفتن داده ها را دارد؟ (100% reliable ⇒ 0 Data loss)
3. Timing — آیا App. زمان بندی را رعایت می‌کند؟ (برای App. باید زمان بندی داشته باشیم؟) (داده صوتی با تاخیر 200ms)
4. Security — امنیت داده ها تا چه حد مهم است؟

* بعضی از این ویژگی ها را لایه‌ی transport می‌تواند فراهم کند. براساس نیازمندی ها، App. انتخاب می‌کند چه سرویسی بگیرد.

مثال :	Throughput	Dataloss	Time Sensivity
File Transfer	o (بستگی دارد)	✓	×
E-mail	o	✓	× ! timing مهم نیست * در سرویس offline
Web Document صفحه وب	o	✓	× (in (ms) order)
real-time audio / video	audio: 5kbps ✓ Video: 10kbps - 5Mbps از نظر کیفیت و اندازه صفحه	×	✓ (under 100ms)

real-time : داریم از روی شبکه نگاه می‌کنیم. پس از ۱۱ دیگر real-time محسوب نمی‌شود.

interactive : دوطرفه است. درحالی که interactive نباشد می‌تواند بیشتر از 100ms تاخیر داشته باشد ولی باید تاخیرهای packet ها یکسان باشد.

interactive game	✓	× loss-tolerant	✓
instant message (chat)	o	✓	✓ (بستگی دارد به order بیشتر از آنی نباید باشد!)
Stored audio/video	o	× loss-tolerant	× (few seconds)

شبیه یک file transfer و ...!

* شبکه‌های اینترنت در ابتدا برای انتقال داده طراحی شدند نه برای انتقال اطلاعات multimedia. پس در آن زمان باید انتقال داده نگاه می‌کردند که dataloss برایشان مهم بود پس TCP را گذاشتند که مطمئن بود. UDP برای کاربردهایی بود که حجم کم دارند و نمی‌خواهند سربار TCP را تحمل کنند زیرا TCP دارای connection است.
بعدها که سرویس‌های multimedia اضافه شدند، ترجیح دادند از UDP استفاده کنند زیرا سریع است.

TCP	UDP
Connection-Oriented	Connectionless
reliable	best effort
flow control	non flow control
congestion control	" congestion control : کنترل ازدحام
doesn't do:	doesn't do:
timing	connection setup
minimum throughput garantees	reliability
security	flow and congestion control
	timing
	throughput
	security

در UDP داده ی Application بلافاصله به شبکه می رود.

← آن کارهایی که انجام نمی دهند را می توانیم در لایه ی Application انجام دهیم. یک copy می گیریم از داده و سپس می فرستیم که اگر داده مطمئن نرسید بتوانیم دوباره بفرستیم. TCP هم همین کار را می کند.

buffering ، برای throughput به کار می رود.

Public Domain Applications:

- e-mail (SMTP)
- web (HTTP)
- file transfer (FTP)

} TCP

- Internet Telephone (RTP, SIP)
- DNS (DNS)
- Network Management (SMNP)

} UDP

سرویس Web و پروتکل HTTP :

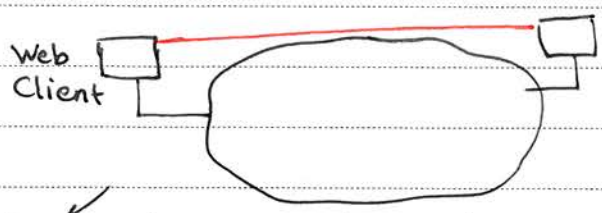
پروتکل HTTP برای انتقال صفحات وب استفاده می شده است.

Web Server
Web browser
} Client-Server مدل

object به وب منتقل می شود. پس صفحات وب object هستند. هرکدام از این object ها را آدرس می دهیم که آن URL می گویند: www.aut.ac.ir/ceit/Logo.jpg
object

Object: Java Applet (Server می فرستد سمت Client تا آن را اجرا شود)، عکس، متن و ... ،
فایل HTML : وقتی آدرس را می زنیم URL سایر object ها هم می آید: اول صفحه می آید، بعد عکس ها و متن هایش ظاهر می شود.

Client باید به Server، request بدهد. HTTP از TCP استفاده می کند، پس باید یک connection ایجاد شود. Server باید آماده ی پذیرفتن درخواست باشد، یعنی connection را ایجاد می کند و یک port(80) باز می کند و روی این port گوش می کند. default port، 80 است.



Client ← port ، Domain و IP Add را بازیابی کند
Server را به TCP می دهد. بین Connection برقرار می شود
و Client از طریق این Connection به Server ، request
می دهد. پاسخ request همان object است که در جواب URL مشخص می شود و به Client برمی گردد
و سپس Connection بسته می شود.

HyperText: یک پیری کلمه که به browser های اولیه فقط text بودند و لینک ها را رفتگی شدند ← www !

HTTP یک پروتکل State less است. یعنی کاری ندارد که Client در دفعه ی قبل هم request داده بود. یعنی به ازای request ها پاسخ می دهد و سپس connection را می بندد. یعنی اگر Client همین request اول یک request دیگر هم بدهد و یا connection قطع شود نمی تواند وضعیت Client را داشته باشد و نگه داری نمی کند. پروتکل هایی که وضعیت را نگه می دارند Stateful هستند. TCP هم Stateful است و کلاً دارای connection ها ، state را نگه می دارند.

Nonpersistent : به ازای هر object ، connection را بازکنیم و سپس ببندیم. خیلی اصراری ندارد که از TCP نهایت بهره را ببرد. ممکن است تاخیر دریافت بسیار زیاد شود.

Persistent : برخلاف بالا ، یک connection بازکنیم و اکثر object ها را بگیریم. یعنی روی یک connection چند درخواست بدهیم. چون زمان هایی توانند با هم overlap شوند ، زمان کاهش پیدا می کند.
(← RTT) ها با هم همپوشانی پیدا می کنند.

WWW (World Wide Web):

Link ها مثل تار عنکبوت هایی شدند که به هم متصل هستند.
از پروتکل HTTP استفاده کردند. این پروتکل object ها را request می دهد توسط URL!
صفحه ی اصلی به فرم HTML می آید و مشخص می شود چند object دارد.

* شبکه باید براساس توافق بین طرفین کار کند (چون تدریجی است) : ISO ، OSI ، TCP/IP

Internet Activity Board (IAB):

فعالیت هایی در زمینه اینترنت صورت می گیرد را با هم هماهنگ کند.

1. Internet Engineering Task Force (IETF):
برعهده است که اگر بخواهیم کاری در زمینه اینترنت انجام دهیم به آن مراجعه می کنیم. مربوط به مهندسی و ساخت است.
چگونگی بهره وری از اینترنت را مطرح می کند.

2. Internet Research Task Force (IRTF):
کارهای تحقیقاتی و مطالعاتی است. پاسخ به مشکلات را آینده نگری می کند و می گوید در آینده باید چه کارهایی انجام شود و ...

Request for Content (RFC): افراد می‌پرسند که در این زمینه چه کارهایی می‌شود انجام داد.

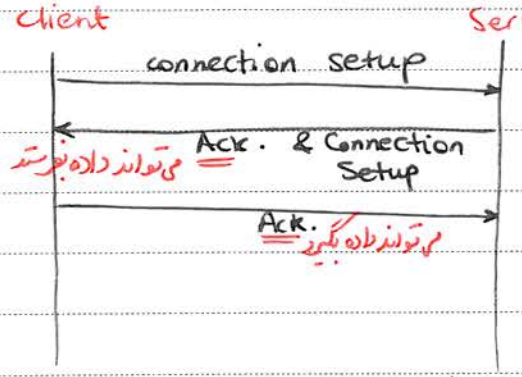
RFC برای HTTP وجود دارد ← RFC 1945
RFC 2616 ↙

* RFC قوانین دارند که این کار را انجام بده، این پروتکل را دریافت کردی ... .

← برای خواندن RFC ها، یک RFC دیگر داریم که قوانین آنها را تعیین می‌کند. مثل کتاب قانون است. →

3-way handshaking:

برای ایجاد connection در TCP: ابتدا Client درخواست به Server می‌دهد. Server، Ack می‌دهد که درخواست برقراری connection را پاسخ داد.



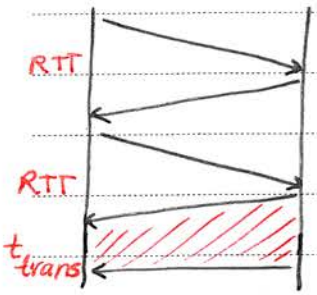
* Client همراه Ack. می‌تواند داده هم بفرستد.

در نتیجه سه پیغام رد و بدل شد. علت خوشامدگویی انجام می‌شود و سپس connection برقرار می‌شود.

Handshaking Signals: سیگنال‌های کنترلی

یک زمان رفت و برگشت طول می‌کشد که به آن Round Trip Time (RTT) می‌گویند.

← برای رفتن هر object، در RTT و یک t_{trans} طول می‌کشد که بستگی به حجم object دارد:



مثال: n تا obj. داریم، چقدر طول می‌کشد؟ $(n+1)\times 2RTT + \sum_{i=1}^{n} t_t(i)$

* وقتی می‌گوییم لایه‌ی n، connection oriented است، یعنی لایه‌ی n+1 سعی دارد به لایه‌ی n+1 متصد وصل می‌کند.

اشکال nonpersistant: Server باید connection را باز نگه دارد برای Client و اگر Client از این منابع استفاده نکند، هزینه‌ی زیادی برای Server بخودی خود بالا می‌رود و Server را پیچیده‌تر می‌کند.

HTTP : دو نوع پیغام دارد : Request (Client) , Response (Server)

Request : 1. Request line : Method: GET (HEAD, POST, PUT, DELETE ← can be) Url: /somedir/page.html HTTP version: HTTP/1.1

2. Header lines {
Host : www.someschool.edu : Server address
user-agent: mozilla/4.0 : Client Application
Connection : close
Accept-language : fr
}

پس از دریافت پاسخ، conn را می‌بندم ؟
صفحه را با چه زبانی دریافت کنم ؟

3. Entity body {

HTTP Request Message : general format

method	space	URL	space	Version	cr	lf
header field name	:	value	cr	lf		
⋮						
"	"	"	"	"		
cr	lf					

* دو تا cr , lf یعنی Header lines تمام شده‌اند.

Entity Body

Head : شبیه get است , برای debugging است . پاسخی دهد ولی Entity body آن خالی است .

Post : می‌خواهیم همراه request یک سری اطلاعات هم به server بدهیم . اطلاعات search را در Entity body می‌فرستیم .

Put : می‌خواهیم object ای را از سمت Client به server upload کنیم .

Delete : object را از سمت server حذف می‌کنیم .

Response : 1. status line : HTTP/1.1 200 OK ... آی URL را داشتیم یا نه و ...

2. Header lines {
connection : close
Date : Thu, 06 Aug 1998 12:00:15 AM : obj. تاریخ ارسال
Server : Apache/1.3.0 (Unix) : Server Application (Web Server?)
last-Modified : Mon, 22 June 1998 آخرین بار کی update شده است
Content-length : 6821 data طول
Content-Type : text/html data type
}

3. Entity Body {data, data, ...

200 OK: Request is succesfully responsed.

301 Moved Permanently: این object ای که request داده‌اید به طور دائمی از این server رفته است.

400 Bad Request: داده شده فرمت request را رعایت نکرده است. request

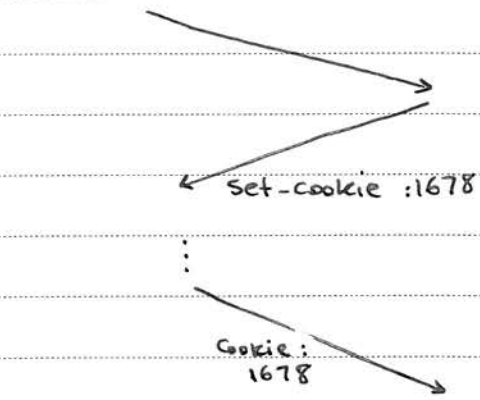
404 Not Found

505 HTTP version not supported

Caching: System / Proxy Server

get conditionally: قبل از این یا بعد از این بود برگشت last modified

Cookies:



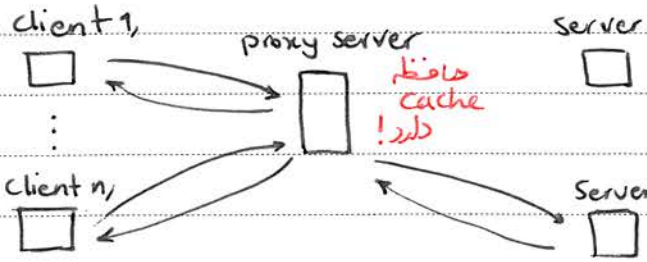
HTTP یک پروتکل stateless است، ولی app. ها نیاز دارند کاربرهای خود را بشناسند و براساس علاقه‌مندی پاسخ بدهند. با استفاده از cookies، app. ها می‌توانند وضعیت را نگه دارند. وقتی client به server درخواست می‌دهد، اگر server او را نشناسد یک cookie برایش برمی‌گرداند. استفاده از cookie اختیاری است

برای حذف از تکنیک‌های aging استفاده می‌شود. مثلاً اگر سن از شش ماه با این cookie کسی به این سایت مراجعه نکرد، پاک می‌شود.

Cache:

عموماً در proxy server، caching انجام می‌شود. proxy server واسط بین client و server است.

از proxy برای filtering و محدود کردن زبان و... هم به کار می‌رود ولی عموماً در caching استفاده می‌شود. cache پنهان است و نباید در عملکرد شما تغییری ایجاد کند. Web cache وظیفه دارد صفحاتی را که client ها به آن خیلی مراجعه می‌کنند، ذخیره می‌کند. به همین دلیل سرعت را بالا می‌برد و از ظرفیت بهتر استفاده می‌کند. نیز الزاماً نسبت درخواست تا سرور برود و برگردد.



proxy از دید client ها یک server است و از دید server ها یک client است.

* صفحات به طور dynamic تغییر می‌کنند و ممکن است اطلاعات cache با server فرق کند.

proxy می‌تواند از t modify و t trans استفاده کند تا این عملکرد را بهتر انجام دهد.

* استفاده از proxy ، optional است ، user می‌تواند انتخاب کند که از cache استفاده شود یا نه. خود browser ها هم گاهی cache دارند. تنظیمات proxy دست user نیست ولی می‌تواند در request اش بگوید که نمی‌خواهد از cache استفاده کند.

Conditional Get :

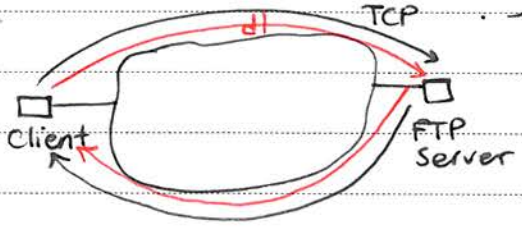
proxy برای server می‌فرستد : در صورتی که تاریخ modify از آن تاریخی که من دارم بزرگتر است برایم بفرست.

```
GET /.../ HTTP 1.1
:
if-modified-since ____
```

اطلاعات cache معتبر است. ← 304 not modified

FTP Servers : روی port 21 یک اتصال برقرار می‌شود. سپس تصدیق هویت انجام می‌شود. از آن جایی که user , pass , text هستند می‌توان آنها را گوش کرد، به همین دلیل باید اتصال TCP را secure کنیم (با استفاده از ssl لایه‌ی ...)

هنگامی که FTP اطلاعات کنترلی (command) می‌فرستد، روی همان اتصال TCP منتقل می‌شود ولی وقتی می‌خواهد download یا upload کند، اتصال دیگری ایجاد می‌کند.



به این مدل که اطلاعات کنترلی و داده از اتصال های مختلف منتقل می‌شوند out-of-band می‌گویند. در صورتی که هر دو از یک اتصال منتقل شوند in-band نامیده می‌شوند.

مزیت های out-of-band :

- ممکن است انتقال یک داده طول بکشد و ما می‌توانیم در حین انتقال command بفرستیم.
- می‌توانیم درخواست دریافت چند فایل را هم‌زمان داشته باشیم (به دلیل ایجاد اتصال های جداگانه)

USER
PASS
LIST — لیست فایل های server را می‌دهد. می‌توانیم dir. عوض کنیم!
RETR — برای download است. بر روی همان dir. که هستیم!
STORE — برای upload است.

{ Stateful, text به صورت

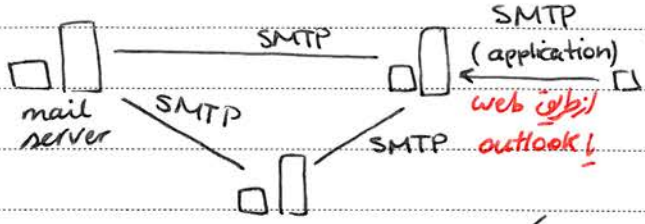
هر command ای که فرستاده می‌شود، برای این که client متوجه بشود یک status code دریافت می‌کند.

مثال:

USER	ALI	→	331	Username OK, Password required
PASS	12345	→	125	dataconnection already open
		↘	425	cannot open data connection

* برای FTP سرورهایی که فایل‌هایی دارند که نیاز به pass ندارد، user anonymus تعریف می‌شود.

E-mail:



هر user یک directory دارد که در آن root می‌گویند. هر mail ای که user دریافت می‌کند داخل inbox قرار می‌گیرد و هر mail ای که ارسال می‌شود در داخل outbox queue می‌شود.

* خود app ها، طی اتوماتیک هر formata دیگری را به این format تبدیل می‌کند. SMTP

email متن داخل 7-bit ASCII

- وقتی یک mail می‌خواهد ارسال بشود یک TCP connection باز می‌کند و شبیه client و server عمل می‌کند. در inbox می‌ماند تا user agent، inbox اش را بخواند.

روش‌های access به mail server:

POP3 RFC 1939: post office protocol ; زمانی که server شما را آنلاین می‌کند، کل inbox شما به agent انتقال می‌یابد.

IMAP RFC 1730: internet mail access protocol: تصویر inbox داخل server را می‌بینید. یعنی باید حتماً internet وجود داشته باشد. برای کسانی خوب است که PC خود را تغییر می‌دهند.

Web: شبیه IMAP است ولی در خود mail server هستید. این بار inbox را درخواست می‌کنیم.

HELO

MAIL FROM

:

* RFC 822 برای SMTP می باشد.

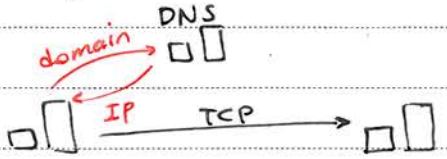
DNS (Domain Name System):

این app. توسط دیگر app. ها استفاده می‌شود. آدرس‌هایی که ما استفاده می‌کنیم domain name هستند:

www.aut.ac.ir

در اینترنت وقتی می‌خواهیم TCP Connection برقرار کنیم باید IP Address استفاده کنیم. پس یک سرویس direct

کرون باید داشته باشیم که domain ها به IP Add. تبدیل شوند.



IP Address هویت است.

Default Gateway } کرنداشته باشیم فقط در شبکه خودمان هستیم. } مهم نیاز است!
DNS هم نداشته باشیم هیچ کس را نمی‌شناسیم.

local DNS: آدرس‌هایی پر کاربرد را در آن قرار می‌دهند. در local networks آدرس‌های داخلی خود را قرار می‌دهند.

1) DNS مکانیزم سلسله مراتبی دارد: www.google.com

2) host aliasing: ممکن است یک web site چند domain name داشته باشد. یک اصلی و دیگری alias هستند:
www.google.com
google.com

دو سرویس لایه transport:

1) UDP ناامن
2) TCP مطمئن

Application
↑ → Application Programming Interface (API)
Transport

اگر Application بخواهد از سرویس TCP یا UDP استفاده کند باید ابتدا یک Socket به خود بیاید تا بتواند داده را ارسال کند.
UDP و TCP هردو port number نیاز دارند.

* در بازکردن Socket باید الزاماً مدل Client-server رعایت شود. پس از باز شدن دیگر برای Transport اهمیتی ندارد که مدل App. چه باشد: UDP or TCP

Server	Client
create socket port=x	

Server باید Socket را قبل از Client باز کند. اگر بعد از Client باز کند ارتباطی برقرار نمی‌شود زیرا Client درخواستی دارد که هنوز آماده نیست!

Client باید آدرس IP سرور و شماره port را بداند.
مثلاً default وب سرور مشخص است. اگر با سروری نیستیم باید Port اش را Client بدهیم.
Server پس از باز کردن port گوش می‌کند و آماده‌ی دریافت است.
Client باید آدرس مقصد و Port سرور را بداند.

UDP Socket Programming:

Client نیز socket را به وجود می‌آورد. اگر port number را مشخص نکنیم، سیستم عامل از App. های خود یک port no. می‌دهد. Server هم لازم نیست شماره port را بداند.

Server

⋮

recieves

⋮

Client

create socket
port = y

Create datagram with Server IP and Port x ⇒ send to (IP, x)

سپس یک datagram (connection-less و like packet) درست می‌کند.
سرور دریافت می‌کند و به App. می‌دهد و ...

برگشت آدرس IP، Client Port در فرستادن data به سرور می‌رسد و سرور از آن استفاده می‌کند. و سپس برای Client می‌فرستد.

اگر کار App. تمام شود و خود port اش را close کند، port آزاد می‌شود و در اختیار بقیه قرار می‌گیرد.
اگر App. از بین برود خود سیستم عامل مانعش (از جمله port) را آزاد می‌کند.

TCP Socket Programming:

سرور گوش می‌کند و آماده است Client ها درخواست بفرستند.
در UDP به محض باز شدن داده می‌فرستد و دریافت می‌کند.

Server

⋮

listen

Client

create Socket
port = y

request for connection setup

اطلاع می‌دهد که request آمده است ← TCP
Ack. →
← Ack.

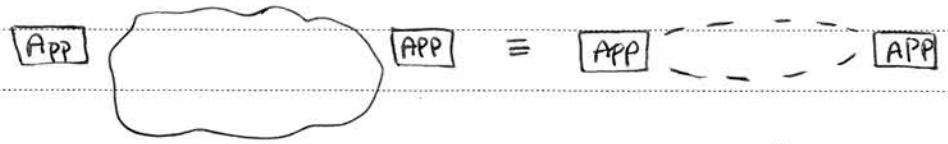
باید پس از دادن request منتظر تایید آن بماند.
request یعنی به TCP گفته است که یک connection با این آدرس و شماره Port به وجود بیاورد.

* سه پیغام کنترلی قبل از تبادل data برقرار می‌شود. از این جا به بعد شبیه UDP است. با این تفاوت که TCP روی این ارتباط کنترل خطا دارد.

لایه‌ی Transport :

وظیفه‌ی حمل سرویس‌های بین دو app را برعهده دارد.
ارتباط منطقی بین دو app ایجاد می‌کند (این کارها از دید لایه‌ی app مخفی می‌ماند).

یک دید منطقی بین دو app به وجود می‌آورد که app‌ها حس می‌کنند مستقیماً در ارتباطند.



Application ها دو دسته‌اند:
- delay sensative : real time , DNS (تاخیر حساس هستند)
- قابلیت اطمینان برایشان مهم‌تر از تاخیر است.

⇐ پس لایه‌ی transport دو سرویس ارائه می‌کند:
- reliable
- best effort

وظایف لایه‌ی Transport :

1. باید بتواند به طریقی ترافیک app‌های مختلف را به سرویس برساند ⇐ multiplexing

ازدحام : Congestion : مجموع جریان‌های ترافیکی عبوری از یک لینک فیزیکی از ظرفیت آن تجاوز کند.
روی یک لینک فیزیکی ممکن است چند جریان ترافیکی عبور کند که اگر از ظرفیت لینک بیشتر شود، حذف می‌شود.
در نتیجه داده‌ها از بین می‌روند. به علت تاخیر صف بندی گره‌ها باعث افزایش تاخیر هم می‌شود.
⇐ منابع را کنترل کنیم که بیش از ظرفیت داده نفرستند.

روش‌های جلوگیری از ازدحام:
1. پیش‌گیرانه (preventive): پیش‌گیری از وقوع ازدحام: ابتدا از منبع بپرسیم حجم ترافیکی‌ات چقدر است و اگر باعث ایجاد ترافیک بشود، اجازه ندهیم.
2. واکنشی (reactive): از کسی جلوگیری نمی‌کنیم اما پس از وقوع ازدحام، به همه‌ی منابع می‌گوییم که از حجم ارسال داده‌شان کم کنند تا ازدحام از بین برود. source ها feed back می‌گیرند از شبکه به طور دائم که اگر جریان ترافیکی رخ داده است، نرخ ارسال را پایین می‌آورد.

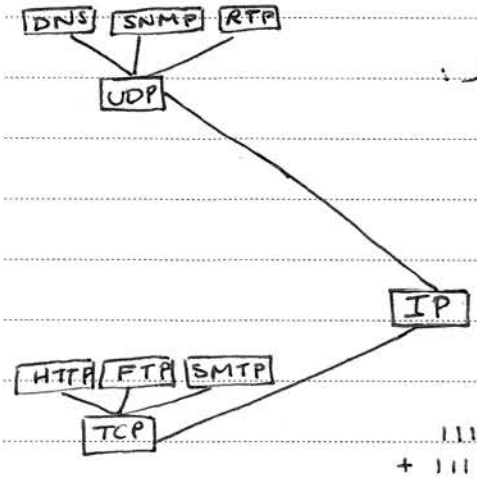
2. کنترل ازدحام به روش reactive

Flow Control اگر فرستنده (منبع) با سرعت زیادی داده‌ها را ارسال کند، گیرنده به علت محدودیت سایز buffer و ... در صورت سرعت کمتر نسبت به فرستنده داده‌ها را از دست می‌دهد.

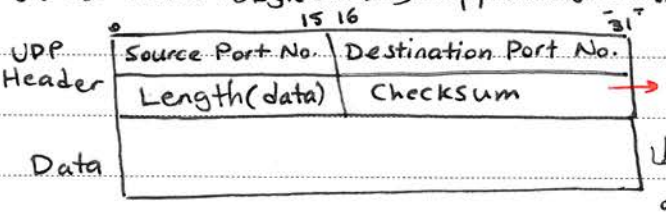
سرویس‌های لایه‌ی Transport به لایه‌ی app.

User Datagram Protocol (UDP):
- best-effort: داده‌هایی که به صورت مستقل منتقل می‌شوند. یعنی UDP داده‌های مستقل از app. می‌گیرد، حتی اگر این پیغام‌ها از دید app. یکی باشند.
- حداکثر اندازه‌ی پیغام به اندازه‌ی حداکثر سایز بسته IP یعنی 64KB است.
- سایز اگر UDP header و IP header را هم حساب کنیم کمتر از این عدد می‌شود. اگر app. پیغامی بزرگ‌تر از 64KB داشته باشد، خود باید عمل segment را انجام دهد.

اگر خطایی در ارسال پیغام رخ دهد، UDP کنترل و تصحیح خطا انجام نمی‌دهد و داده را بدون بررسی و تضمینی برای رساندن داده‌ها ندارد. app اگر داده‌ی مهمی دارد، خود باید مکانیزم‌های کنترل خطا را انجام دهد.
اگر خیلی مهم است نباید از UDP استفاده کند.



UDP روی پیغامی که از app گرفته است 8byte header اضافه می‌کند:



تشخیص خطا detection
در مقصد داده‌ای را که خطا دارد دور می‌ریزد، یعنی داده خطا را به app نمی‌رساند. UDP، optional است.

روش محاسبه UDP Checksum و IP:

داده‌ها را به صورت اعداد 16bit ای در می‌آورند:

```
1111 0011 0011 0011
+ 1010 1010 1010 1010
-------------------
1 1101 1101 1101 1101
  ↑ 1's Complement جمع
-------------------
1101 1101 1101 1110
```

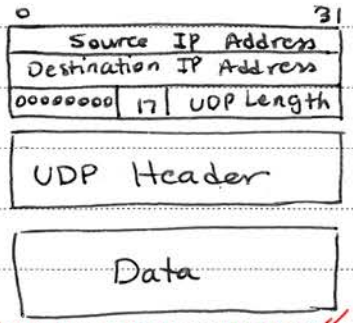
نتیجه 16 بیتی:

```
1's Complement ↓ 0010 0010 0010 0001 → Checksum
1101 1101 1101 1110
```

در Checksum جمع چک می‌شود و در صورت جابجایی ممکن است ترمیم نشود. یعنی تا حدی که نتیجه جمع تغییر کند می‌تواند تشخیص دهد. تمام کدهای تشخیص خطا یک قدرت تشخیص دارند و اگر خطا از حدی بیشتر شود دیگر قدرت ندارند.

* در غیاب 1's Complement دو صفر داریم. همه‌ی بیت‌ها 1 و همه‌ی بیت‌ها 0 (در هر دو حالت همه‌ی بیت‌ها در آخر باید 1 شوند.)

Pseduo header: UDP این header را در فرستنده درست می‌کند برای محاسبه‌ی Checksum و هیچ وقت این header ارسال نمی‌شود.



* برای ارسال این header 8b اضافه شده ارسال می‌شوند و فقط برای checksum بوده‌اند.

خطا در ارسال داده: اگر داده تغییر کند، شماره پورت عمدا یا بقصد تغییر کند.
تشخیص توسط Checksum خطا در مسیریابی، protocol و ... .

← این header را گیرنده از فرستنده (لایه‌ی IP) می‌گیرد و Checksum را مجددا محاسبه می‌کند.

← IP آدس‌ها برای این هستند که اگر مسیریابی اشتباه انجام شد، Checksum بتواند detect کند.

← خطای Protocol یعنی مثلا بسته از UDP آمده است و به TCP رفته است.

اگر data 16b نبود 8b صفر به آن اضافه می‌کنیم.

چرا این app ها و IP دو لایه‌ی UDP و TCP هم قرار می‌گیرند؟ به عنوان مثال UDP هم باید از بستر IP ارسال می‌کند. به علت سرویس‌های زیر:

multiplexing

سرویس های UDP به app ها

checksum

سرویس های لایه transport به لایه ی app.

هم کنترل جریان، هم کنترل ازدحام انجام می دهد. اکثر app. ها از TCP استفاده می کنند. ترافیک Internet به علت TCP است. (TCP) یعنی ترافیک TCP غالب است.

سرویس های TCP :

1. Error Control : خطا در ارسال داده را تشخیص و تصحیح کنیم. داده بدون خطا یعنی حتماً یک نسخه می رسد (duplicate نمی شود)، ترتیب بیت ها به هم نمی خورد و خود بیت ها هم درست هستند.

• Forward Error Control (FEC) فرستنده به داده check bit کنترل خطا اضافه می کند (مثل checksum) که گیرنده به وسیله آنها می توان تشخیص خطا داد و این check bit ها محل وقوع خطا را هم مشخص می کنند ولی سربار آنها خیلی زیاد است. چون تعدادشان نسبت به داده زیاد است.

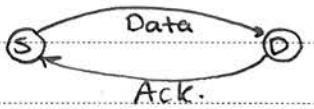
Data 8 | Check bits 4 : میزان استفاده مطلوب $8/12 = 2/3$

کدهای FEC را با جان نرخ مفید مشخص می کنند. ↑ هرچه احتمال خطای کانال بیشتر باشد باید تعداد check bit ها را افزایش دهیم. سپس عمل تصحیح خطا گیرنده انجام می شود.

(S) — (D)

• Backward Error Control
Automatic retransmission (repeat) request (ARQ)

در این روش گیرنده می تواند به وسیله کد تشخیص دهد محل وقوع خطا کجاست پس دور می ریزد. فرستنده کپی نگه می دارد. اگر گیرنده درست دریافت کرد، برای فرستنده Ack. می فرستد. فرستنده در زمان مشخصی انتظار دریافت Ack دارد و اگر دریافت نکند، داده را مجدداً ارسال می کند (retransmit)



پس از k بار retransmission حتماً گیرنده ای وجود ندارد.

negative Acknowledgement : بیت 0 را گرفته است ولی بیت 1 را نگرفته است و سپس بیت 0 را گرفته است. پس می فهمد که بیت 1 را نگرفته ⇐ Ack. منفی می فرستد تا ارسال مجدد صورت گیرد.

ارسال مجدد: timer / negative Ack.

* سربار check bit این سرویس خیلی کم است. سربار اصلی retransmission (و کمتر از آن Ack.) است.

⇐ اگر کانال های ارتباطی شبکه یک طرفه باشد (پخش رادیویی، کانال ماهواره ای) چون گیرنده ارتباط ندارد تنها از FEC می توان استفاده کرد.

هر لایه یک header اضافه می‌کند :

H	Data

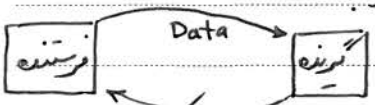
ولی پیغام های کنترلی header ندارند.

* field شماره ترتیب حتماً در header ی ARQ protocol وجود دارد. بدون این field بعضی از خطاها قابل تشخیص نیستند.

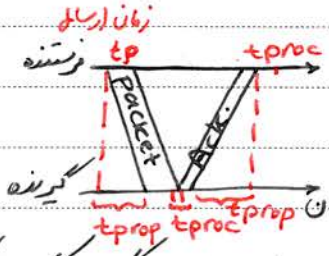
ARQ
- Stop and wait ARQ
- Goback N ARQ
- Selective Repeat ARQ

(پیچیدگی افزایش، کارایی افزایش)

* دقت این پروتکل ها ارتباط بین فرستنده و گیرنده باید دوطرفه باشد!



پیغام های کنترلی
Negative Ack.
Ack.
⋮



Stop & Wait : ساده‌ترین protocol است.
پس از فرستادن packet صبر می‌کند تا تأییدیه برگردد و تا آن زمان ارسالی انجام نمی‌دهد. زمان ارسال بسته و آمدن تأییدیه :

$$t_{prop} + t_{p_{packet}} + t_{proc} + t_{a_{ack.}} + t_{prop} + t_{proc}$$

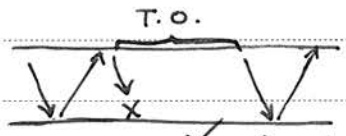
مکانیزم های کنترل خطا

لایه ۲ : Data Link این لایه framing انجام می‌دهد، frame می‌فرستد.
hop-by-hop از هر گره به گره ی دیگر کنترل خطا انجام می‌دهد.

لایه ۴ : Transport end-to-end بین گره های پایانی کنترل خطا انجام می‌دهد.
شبکه از دید لایه ۴ مخفی است و دو گره انتهایی انگار کنار هم هستند.

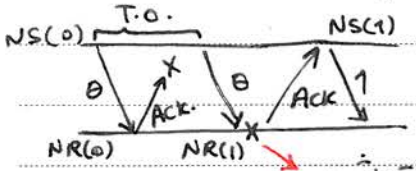
بسته فرستاده شده

اگر خطا نداشته باشد که مانند شکل بالاست.



اگر با خطا برخورد شود هیچ گاه به مقصد نمی‌رسد و دور ریخته می‌شود.

* باید برای بسته ها شماره ی ترتیب بگذاریم، چون ممکن است بسته سالم برسد ولی Ack. را دریافت نکنیم و بس از Time out دوباره بسته قبلی را می‌فرستد و duplication به وجود می‌آید، خود یک خطاست.

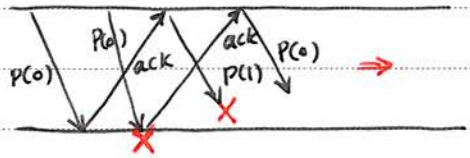


بسته دور ریخته می‌شود
و گیرنده دوباره Ack. می‌دهد.

N(R) : متغیر وضعیت گیرنده، یعنی الان منتظر رسیدن چه بسته ای است.
N(S) : متغیر وضعیت فرستنده، یعنی الان چه بسته ای را ارسال می‌کنیم.
هر بار Ack. می‌گیریم این متغیر را یک واحد اضافه می‌کنیم.

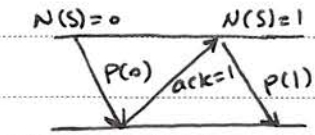
* عدد شماره ترتیب را باید داخل header بگذاریم و ارسال کنیم و چون محدودیت تعداد bit داریم باید کمترین bit را انتخاب کنیم. در stop & wait شماره ترتیب می‌تواند یک بیتی باشد : ... → 0 → 1 → 0 → ...
Alternative Bit ARQ نام دیگر Stop & Wait است.

← فرستنده timer را خوب تنظیم نکرده است و زود time out می‌دهد:



⇒ دچار اشتباه شد زیرا نمی‌داند ack حال کدام بسته است. پس باید در ack هم شماره بسته را لحاظ کنیم.

در تمام protocol های ARQ، N(S) به عنوان شماره ترتیب استفاده می‌شود و N(R) به عنوان شماره ack استفاده می‌شود.



* ack=1 یعنی تا یکی قبل را گرفتم (تأییدیه ۰)
* ack اضافه دوربخته می‌شود. تنها معنای این است که timer درست تنظیم نشده است.

اگر شماره ترتیب n بیتی باشد، با دریافت هر بسته درست:
$N(R) = [N(R)+1] \bmod 2^n$
" " " " " ، با دریافت هر تأییدیه:
$N(S) = [N(S)+1] \bmod 2^n$

در شکل صفحه قبل:
$t_{total} = t_{prop} + t_p + t_{proc} + t_a + t_{prop} + t_{proc}$
$= t_p + t_a + 2(t_{prop} + t_{proc})$
اگر فرض کنیم تاخیر انتشار و تاخیر پردازش دو طرف باهم برابرند:

کارایی:
عددی بین صفر و یک است و به درصدشان داده می‌شود.

$\eta^{\circ}_{S\&W} = \frac{R^{\circ}_{eff}}{R}$ ← بدون خطا

R_{eff}: نرخ استفاده مفید
R: پهنای باند بین مبدا و مقصد bandwidth

$R^{\circ}_{eff} = \frac{n_f - n_0}{t_{total}}$
تعداد بیت ارسالی در واحد زمان
بسته: [H | Data] : در هر t_{total} یک بسته ارسال شده است
n_0 | $n_f - n_0$ → n_f

n_f: تعداد بیت بسته
n_0: تعداد بیت سربار
n_a: تعداد بیت بسته ack

$t_{total} = \frac{n_f}{R} + \frac{n_a}{R} + 2(t_{prop} + t_{proc})$
$R^{\circ}_{eff} = \frac{n_f - n_0}{t_{total}}$

$\Rightarrow R^{\circ}_{eff} = \frac{(n_f - n_0)R}{n_f + n_a + 2(t_{prop}+t_{proc})R}$

$\eta^{\circ}_{S\&W} = \frac{R^{\circ}_{eff}}{R} = \frac{n_f - n_0}{n_f + n_a + 2(t_{prop}+t_{proc})R}$

$\eta^{\circ}_{S\&W} = \frac{1 - \frac{n_0}{n_f}}{1 + \frac{n_a}{n_f} + \underbrace{\frac{2(t_{prop}+t_{proc})R}{n_f}}_{3}}$

عوامل کوچک کردن کسر بالا:
1) $\frac{n_0}{n_f}$ نسبت سربار
2) $\frac{n_a}{n_f}$ سربار ack. (افزایش تعداد بیت‌های ack ← کاهش کارایی)
3) Delay Bandwidth Product

$a = \frac{\text{delay}}{2(t_{prop}+t_{proc})} \times R \times \frac{1}{n_f}$

عامل کاهش کارایی به این شکل که در زمان‌هایی که تاخیر داریم مادر فرستنده idol (بیکار) هستیم و در آن زمان می‌توانستیم delay x R بیت ارسال کنیم.

مثال:

1) فرستنده —R=100Kbps, delay=0.01s→ گیرنده $n_f = 10000$ bits $n_o = n_a =$ ناچیز $\eta_{sw}^{\circ} = \frac{1}{1+2a}$

2) فرستنده —R=10Mbps, delay=0.01s→ گیرنده 1) $\eta_1 = \frac{1}{1.2} \simeq 0.8$ 2) $\eta_2 = \frac{1}{21} \simeq 0.5$

وقتی نرخ ارسال بالاست، تعداد بیت ارسالی در زمان انتظار بیشتر می‌تواند باشد، برای همین کارایی پایین می‌آید.

محاسبه‌ی کارایی با خطا:

اگر نرخ خطا برابر p باشد:

$BER = P$

$P_S = \underbrace{(1-p)(1-p)\cdots(1-p)}_{\text{تعداد کل بیت} = n_f} = (1-p)^{n_f}$ احتمال موفقیت (تمام داده‌ها دست به مقصد برسند)

$P_F = 1 - P_S = 1 - (1-p)^{n_f}$ احتمال عدم موفقیت

* احتمال خطا در هر بیت مستقل از بیت دیگر است.
* به طور متوسط برای آن که یک بسته موفق به مقصد برسد، باید $\frac{1}{P_S}$ عمل ارسال بسته انجام شود.

اگر زمان ارسال t_{total} باشد:

$E[t_{total}] = t_{total} \times \frac{1}{1-P_F}$ به طور متوسط چقدر طول می‌کشد تا یک ارسال موفق داشته باشیم

$R_{eff} = \frac{n_f - n_o}{E(t_{total})}$

$R_{eff} = \frac{n_f - n_o}{\frac{t_{total}}{1-P_F}} = \frac{n_f - n_o}{t_{total}}(1-P_F)$

$\eta_{sw} = \frac{R_{eff}}{R} = \frac{n_f - n_o}{R(t_{total})}(1-P_F)$

$\eta_{sw} = \frac{1 - \frac{n_o}{n_f}}{1 + \frac{n_a}{n_f} + \frac{2(t_{prop} + t_{proc}) \times R}{n_f}}$

اثبات : به طور متوسط برای یک ارسال موفق $\frac{1}{P_s}$ بار تکرار انجام می‌شود :

$P[n=i] = P_f^{i-1}(1-P_f)$ احتمال این که بعد از i بار تکرار، ارسال موفق داشته باشیم!

$$E[t_{total}] = \sum_{i=1}^{\infty} i \cdot t_{total} \cdot P[n=i]$$

$$= t_{total} \cdot \sum_{i=1}^{\infty} i P_F^{i-1}(1-P_f) = t_{total}(1-P_F)\sum_{i=0}^{\infty} i P_F^{i-1}$$

$$= t_{total} \cdot (1-P_f) \sum_{i=1}^{\infty} \frac{d}{df} P_f^i$$

$$= t_{total} \cdot (1-P_f) \frac{d}{df} \underbrace{\sum_{i=1}^{\infty} P_f^i}_{\frac{P_f}{1-P_f}}$$

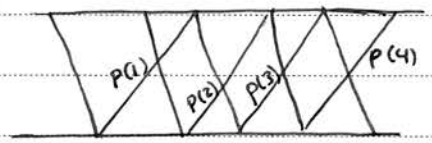
$$= t_{total} \cdot (1-P_f) \cdot \frac{1}{(1-P_f)^2}$$

$$E[total] = t_{total} \cdot \frac{1}{1-P_f}$$

Goback N : هدف آن این است که ارسال فرستنده را قطع نکند تا کارایی افزایش یابد. باید فرستنده اجازه داشته باشد بیش از یک بسته ارسال کند و Ack. نگیرد. تعداد بسته ها به bandwidth product بستگی دارد. باید اجازه دهیم n بسته را ارسال کند به طوری که هنوز هنگامی که Ack. اول را می‌گیرد، ارسال تمام نشده است. پس n-1 بسته هنوز Ack. نگرفته اند و می توانند یک بسته ارسال کنند. و بدین ترتیب اگر خطا نداشته باشیم ارسال قطع نمی‌شود.

$$N \geq \left\lceil \frac{t_{total}}{n_f} \right\rceil$$

$$w_s \cdot t_f \geq t_{total} \Rightarrow ws \geq \left\lceil \frac{t_{total}}{t_f} \right\rceil$$



هر بسته در زمان t_f ارسال یا دریافت می‌شود :

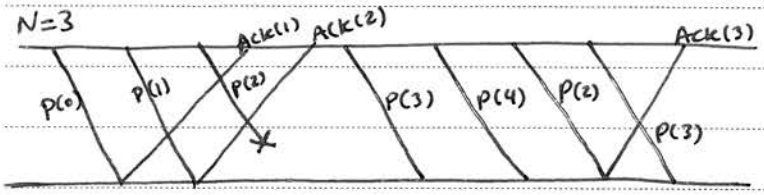
در این جا اثر bandwidth product و سربار Ack. نداریم.

$$R^{\circ}_{eff\,GBN} = \frac{n_f - n_0}{t_f}$$

$$\eta^{\circ}_{GBN} = \frac{R^{\circ}_{eff\,GBN}}{R} = \frac{(n_f - n_0)}{R\,t_f}$$

$$\eta^{\circ}_{GBN} = \frac{n_f - n_0}{n_f} = 1 - \frac{n_0}{n_f}$$

کارایی در حالت خطا :



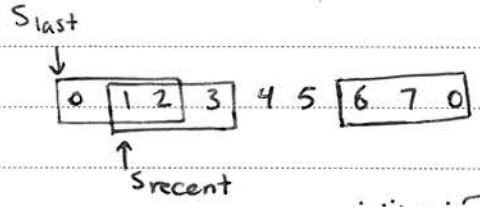
به دلیل این که سه بسته ارسال کرده است و Ack. دریافت نکرده است، حق ندارد بسته‌ی دیگری ارسال کند. برای packet 2 ، time out می‌شود.

یعنی گیرنده از 2 به بعد را نگرفته است و باید دوباره 1,4,3,2 بفرستیم. چون فرستنده به ازای هر خطا n مورد به عقب برگشته و دوباره ارسال مجدد می‌کند. به این پروتکل Goback N می‌گویند.

به آن چه که فرستنده می‌تواند بفرستد، پنجره ارسال ($W_S = N$) و به آن چه که گیرنده می‌گیرد، پنجره دریافت ($W_R = 1$) می‌گویند.

در این جا شماره ترتیب n بیتی است زیرا $W_S = N$. در پروتکل قبلی چون $W_R = W_S = 1$ بود، شماره ترتیب می‌توانست یک بیتی باشد. شماره ترتیب n بیتی شامل اعداد $0, 1, \ldots, 2^n - 1$ است.

پنجره ارسال :



3 بیت :

به این پروتکل ARQ ، پروتکل های sliding window می‌گویند، چون پنجره ثابت است ولی شماره ترتیب های داخل آن می‌لغزند.

با برگشتن Ack. ، اشاره‌گر S_{last} یک واحد افزایش می‌یابد.
با ارسال یک جواب ، " S_{recent} " " " " .

* تعداد بسته‌هایی که فرستادیم ولی ack برنگشته : $S_{recent} - S_{last} + 1$

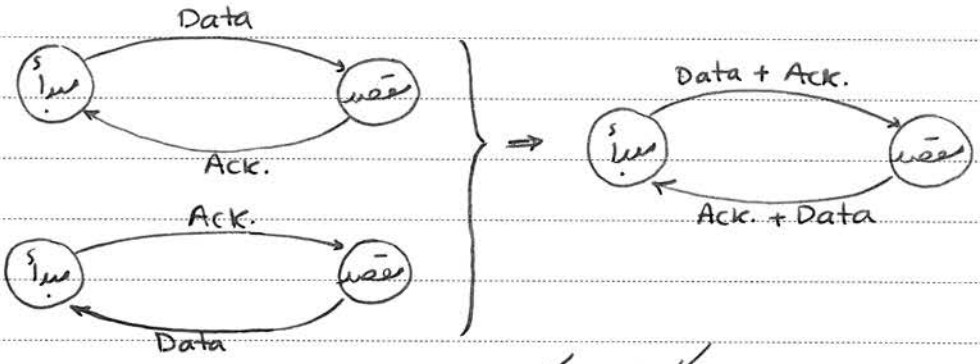
$S_{recent} - S_{last} + 1 \leq N$ یا W_S

$S_{recent} - S_{last} \leq W_S - 1$

پنجره ارسال با پنجره دریافت نباید overlap داشته باشد . اگر شماره ترتیب n بیتی باشد ، پنجره ارسال باید از $2^n - 1$ کوچکتر یا مساوی باشد و پنجره دریافت هم یک است .

$w_s + w_R \leq 2^n$ stop & wait پروتکل

باید کانال ارتباطی دوطرفه باشد و هم از مبدأ به مقصد و هم از مقصد به مبدأ باید connection داشته باشیم .
برای کم کردن سربار می‌توان Data + Ack. را فرستاد .



به این روش piggy backing (سواری مجانی گرفتن) می‌گویند .
اینجا برای فرستادن Ack. ، فقط یک field را پر می‌کنیم (درصورتی که اگر می‌خواستیم مستقل بفرستیم ، سربار زیادی داشت . در روش های ARQ سعی می‌کنند Ack. را به روش piggy backing بفرستند ، البته به این شرط که داده‌ای از مبدأ به مقصد وجود داشته باشد . اما اگر داده وجود نداشته باشد ، از دو راه می‌توان Ack. را فرستاد : 1. Ack. را تنها بفرستیم 2. صبر کنیم شاید داده‌ای بعداً وجود داشته باشد تا Ack. را با آن بفرستیم . البته تا زمانی صبر می‌کنیم که فرستنده time out نکند . یعنی برای گیرنده هم باید timer گذاریم تا متوجه شویم زمان time out فرستنده چقدر است .

بدون خطا t_f

یک خطا $t_f + w_s . t_f$

$$E[t_{GBN}] = t_f + \sum_{i=1}^{\infty} (i-1)\, w_s\, t_f\, P[n_t = i]$$

$$= t_f \left(\frac{1 + (w_s - 1) P_f}{1 - P_f} \right)$$

$$R_{eff\ GBN} = \frac{n_f - n_0}{E[t_{GBN}]} = \frac{n_f - n_0}{\frac{n_f}{R}\left(\frac{1+(w_s-1)P_f}{1-P_f}\right)}$$

$$= \frac{1-\frac{n_0}{n_f}}{1+(w_s-1)P_f} \cdot R \cdot (1-P_f)$$

$$\eta_{GBN} = \frac{R_{eff\ GBN}}{R} = \frac{1-\frac{n_0}{n_f}}{1+(w_s-1)P_f}$$

delay bandwidth اثر

چون w_s خود تابعی از delay است.

$$\eta^{\circ}_{GBN} = 1 - \frac{n_0}{n_f}$$

چون داریم $\left\lceil \frac{t_{total}}{t_f} \right\rceil \geq w_s$ داز طرفی ارسال نباید قطع شود ، w_s باید یک میزان معمولی و میانگین داشته باشد.

در واقع w_s تابعی از a است. برای افزایش کارایی باید این عامل را که اگر خطا رخ داد باید دوباره همه فرستاده بشود ، از بین ببریم. پس از الگوریتم Selective Repeat استفاده می‌کنیم.

: Selective Repeat

به گیرنده اجازه می‌دهیم بسته‌ها را خارج از ترتیب نیز دریافت کند. مثلاً اگر بسته 3 از بین رفت ، می‌تواند 5, 4 را در بافر خود نگه دارد ولی به لایه بالا نمی‌دهد. وقتی 3 را دریافت کرد به لایه‌ی بالا می‌دهد.

فرض کنید گیرنده تا 3 تا خارج از نوبت بگیرد ، این بستگی به حجم بافر دریافتی دارد ، چند تا خارج از ترتیب بگیرد.

$N(R) = 3$ } می‌توانند با هم مساوی نباشند
$W(R) = 3$ } ولی در حالت مساوی مناسب است

ادامه را نفهمیدم!

TCP خودش IP سرویس می‌گیرد که یک سرویس best effort است. ولی سرویسی که خود ارائه می‌دهد مطمئن است:

Connection Oriented

Reliable (error control)

Byte Stream Transfer

Flow Control

Congestion Control

دو app. جریانی از داده را تبادل می‌کنند (از طریق پروتکل TCP)

خروجی شبکه از دید app. ها لخخی است. در این جریان جابجایی byte ها وجود ندارد.

برای این که ترافیک داده به علت تفاوت نرخ ارسال و دریافت به وجود نیاید از مکانیزم Flow Control استفاده می‌کنیم:

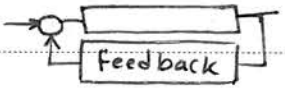
- کنترل نرخ ارسال فرستنده از سمت گیرنده.

$r_1 \longrightarrow \square \longrightarrow r_2$ if $r_1 > r_2 \Rightarrow$ buffer overflow

Congestion Control: مجموع جریان هایی که از روی لینک فیزیکی می‌گذرد از ظرفیت لینک بیشتر می‌شود: $\sum r_i > R$

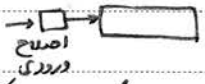
برای از بین بردن ازدحام از دو روش زیر استفاده می‌کنیم:

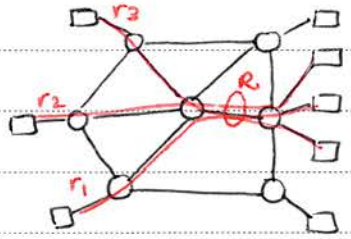
همه جریان هایی تو انتظار بمانند. مبنا از شبکه feedback می‌گیرد.



reactive (closed loop): TCP از این روش استفاده می‌کند

preventive (open loop): جریانهایی که ازدحام ایجاد می‌کنند اجازه ورود نمی‌دهد.





از بین رفتن داده / نگرفتن Ack.

۱- خطا: اگر از خطای بیت صرف نظر کنیم

۲- ازدحام: دلیل اصلی از بین رفتن داده ها ازدحام است

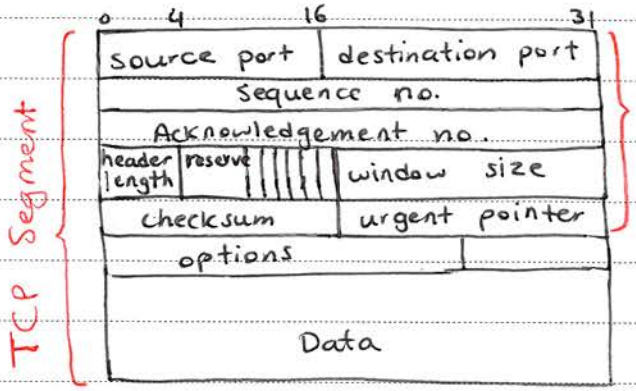
TCP این گونه برداشت می‌کند و نرخ ارسال را پایین می‌آورد. وقتی Ack. گرفت دوباره نرخ را بالا می‌برد، تا جایی که دوباره Ack. دریافت نکند و ...

* در reactive تمام ظرفیت شبکه پر می‌شود ولی در preventive ممکن است شبکه ظرفیت خالی هم داشته باشد که معمولاً یک سیستم best effort هم ارائه می‌دهد برای ظرفیت خالی.

* کیفیت سرویس با throughput رابطه عکس دارد.

IP: بسته تشکیل می‌کند. TCP به تعداد بایت هایی که برای هر بسته به IP می‌دهد segment می‌گوید. اندازه هر segment حداکثر 64Kb بایت. اندازه segment قابل تنظیم است و تا 64Kb قابل افزایش است. TCP داده هایی که از app. می‌گیرد در buffer اش جمع می‌کند. یک header اضافه می‌کند سپس از این که به مقدار مورد نظر رسید به IP می‌دهد. پس از ارسال توسط IP، TCP تعهد بابیان header تشخیص می‌دهد که داده مال کدام app. است.

TCP header از 20b تا 60b می‌تواند تغییر کند. basic header همیشه 20b است:



- شماره ترتیب در TCP، 32b است. زیرا روی هر بایت از یک شماره ترتیب گذاشت. شماره ترتیب یک bytestream 100 تایی اگر x باشد، شماره ترتیب بعدی x+100 است.

- اندازه header واحد 32bit دارد. یعنی اگر تمام بیت های header length 1 باشد، داریم:

$15 \times 4 = 60$

اگر هیچ optionای نداشته باشیم، اندازه header 5 است (5x4=20 basic)

URG	ACK	PSH	RST	SYN	FIN

- Connection Setup:

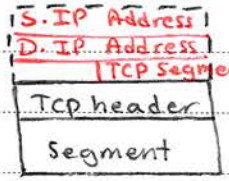
اگر segmentای دریافت شود که SYN 1 باشد به معنی Connection setup است و field داده آن خالی است. درجواب ACK 1 می‌شود. یعنی پذیرفتیم که ارتباط برقرار شود.

Connection Release:

اگر بیت FIN 1 شود، حالت connection است.

- checksum: مثل UDP، TCP در آن یک sudo header قرار می‌دهد:

TCP sudo header:



- URG: app مبدأ یک داده اضطراری دارد که می‌خواهد مقصد سریع آن را پردازش کند، پس این بیت را 1 می‌کند و urgent pointer تا کجا داده اضطراری است. ابتدا همان جایی است که مقصد تا آن جا خوانده است و انتهای آن را urgent pointer می‌گوییم. حالت اول نشان دادن اهمیت داده!

- PSH: segmentای با سایزی کوچکتر از segment size فرستاده می‌شود. حالت دوم نشان دادن اهمیت داده!

- SYN: فرستنده درخواست connection setup است و می‌خواهد یک ارتباط ایجاد کند. با set کردن این بیت، درخواست می‌دهد.

- FIN : درخواست بسته شدن connection در فاز رهاسازی ارتباط.
- RST : connection به صورت غیرعادی بسته شود. با این بیت به فرستنده اطلاع داده می‌شود. یعنی اگر app. درخواست بسته شدن بدهد با FIN نشان می‌دهد.

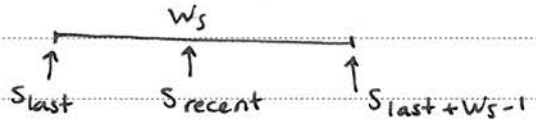
Options:

1. اگر بخواهیم حداکثر اندازه segment را تغییر دهیم در فاز connection setup به فرستنده اطلاع دهیم.
2. option ها باید مضربی از 4byte باشند (به دلیل TCP header). اگر کمتر بود با اضافه کردن padding این کار را انجام می‌دهیم.

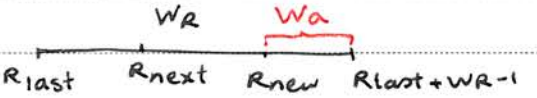
3. Window:

کنترل جریان

هرگاه داده ای ارسال می‌کنیم S_{recent} جلو می‌رود:



$S_{recent} - S_{last} \leq W_S - 1$



application R_{next} را می‌خواند. فضای خالی buffer در پایان پس از دریافت R_{new} از R_{new} تا R_{last+W_R-1} است.

اگر app. سریع باشد R_{last} سریع به R_{next} می‌رسد فضای خالی buffer کم می‌شود.

فضای خالی buffer: $W_a = W_R - (R_{new} - R_{last} + 1)$

به جای این که سرعت پایین بیاید، گیرنده W_a را محاسبه می‌کند و هر بار قصد دارد داده ای برای فرستنده بفرستد، عدد W_a را در window size قرار می‌دهد (این قدر فضای خالی دارم). فرستنده در field را نگاه می‌کند داریش را طوری انجام می‌دهد که حجم اطلاعات ارسالی از W_a بیشتر نشود:

→ $S_{recent} - S_{last} \leq \min(W_S, W_a) - 1$

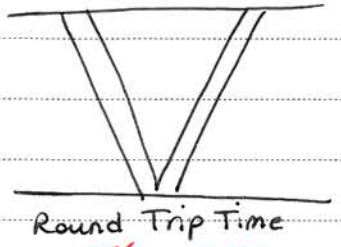
4. Window Scaling:

واحد پنجره W_a تعریف می‌شود. می‌توانیم این جا واحد را تغییر دهیم.

timer: بهترین زمان انتخاب time out زمانی بین رفت و برگشت است. این timer باید برابر این زمان رفت و برگشت لحظه‌ای انتخاب شود نه این که از قبل انتخاب شده باشد.

اگر متوسط تاخیر رفت و برگشت را بدانیم تصادفی است، زیرا ثابت نیست. ولی باید آن را حساب می‌کنیم تا بفهمیم این داده تصادفی نسبت به میانگین چه وضعیتی دارد.

$T_{time\text{-}out} = \bar{\tau}_{RTT} + K\sigma_{RTT}$ (میانگین، انحراف معیار)



پس مطمئن می‌شویم که تعداد زیادی از time out ها در این بازه زمانی است. برای اولین تخمین می‌زنند که زمان time out کجا است و هر بار با ارسال داده این تخمین را تصحیح می‌کنند.

$$t_{RTT(new)} = \alpha\, t_{RTT(old)} + (1-\alpha)\tau_n$$

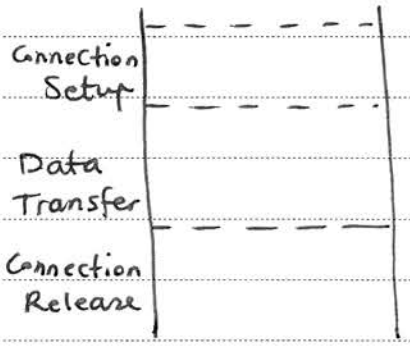
$0 < \alpha < 1$

معمولاً در پروتکل TCP، α را $\frac{7}{8}$ می‌گیرند یعنی به پدیده قدیمی 7 از 8 و به پدیده جدید 1 از 8 می‌دهد.

$$d_{RTT(new)} = \beta\, d_{RTT(old)} + (1-\beta)\,|\tau_n - t_{RTT}|$$ تخمین انحراف معیار

$$T_{t.o} = t_{RTT} + 4d_{RTT}$$ time out

TCP:

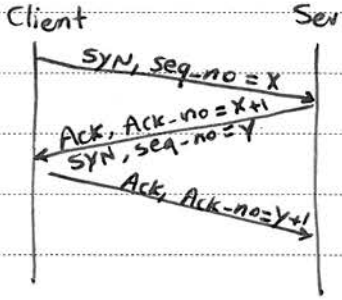


TCB (Transmissio Control Block): منابعی که connection برای انجام عملیات خود نیاز دارد و وقتی با یک connection release انجام می‌دهیم این منابع آزاد می‌شوند.

1. Connection Establishment

در TCP حتماً باید یک conn. برقرار شود از مبدأ به مقصد و برعکس. گره درخواست کننده client و درخواست شونده server است. client آدرس server و شماره پورت app. را باید داشته باشد. server از قبل باید یک پورت را باز کند و گوش کند: connection passive (زمانی که client درخواستی ندارد است.) passive open

connection active (درخواست آمده است.) active open



تا زمانی که connection باز نشده است هیچ data ای رد و بدل نمی‌شود.

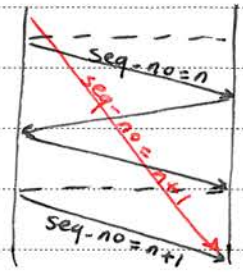
شماره ترتیب Client (از دید Server) یک عدد تصادفی است. از دید Client بامعنی است. Client در هر connection، seq-no را تعیین می‌دهد.

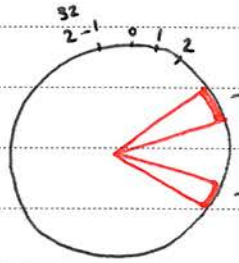
Server با Ack-no می‌گوید که این SYN را گرفته است. و یک SYN به Client می‌دهد که یعنی می‌خواهم یک conn. از Server به Client هم باز شود. X+1 یعنی منتظر داده‌ی X+1 ام است. Client می‌تواند با Ack اش داده هم بفرستد. سه پیام باید رد و بدل شود تا TCP Connection شکل بگیرد.

به این مکانیزم three way handshaking می‌گویند.

2. Data Transfer:

- seq-no را از یک عدد رندم (Initial Sequence Number) شروع می‌کنیم، زیرا ممکن است connection ای را از قبل باز داشته باشیم و این conn. به دلیل غیرفعالی بسته شده است. پس ممکن است هنوز seg هایی از قبل در شبکه مانده باشند که هنوز به مقصد نرسیده‌اند. اگر همیشه seq-no را از عدد ۰ شروع کنیم، ممکن است داده قبلی برسد با همان seq-no و داده جدید به حساب می‌آید و داده ما که داده جدید است تکراری است و دور ریخته می‌شود:





پنجره ارتباط قبلی

پنجره ارتباط بعدی

پس داده قبلی در پنجره دریافت جدید جای می‌گیرد و دور ریخته می‌شود.

- وقتی داده‌ای دریافت یک timer روشن می‌کند.

داده فرستاد: 1kb

فرستاد: 100byte

گیرنده منتظر بایت 2024 است.

خطا:
1. روی داده
2. روی Ack.

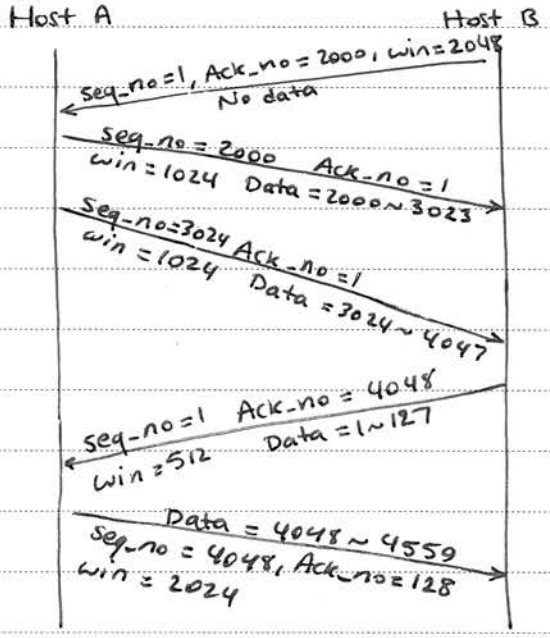
TCP، negative Ack. ندارد، به جای آن:

1. time out: اگر Ack را در زمان مشخصی نگیریم، همان segment دوباره ارسال شود.
2. fast retransmission: فرستنده مطلع شود داده‌ای از بین رفته است.

Ack. تکراری: { بخشی از داده از بین رفته است. / درست نبودن تنظیم timer ها

- به همین دلیل اگر گیرنده سه Ack. تکراری بگیرد، می‌فهمد که داده‌اش از بین رفته است و retransmissions می‌کند.
- از بین رفتن Ack. ها پیچیده‌ای نیست چون فرستنده می‌فهمد کدام داده دست دریافت نشده است و باید دوباره بفرستد، مگر این که time out کند.

TCP Window Flow Control:

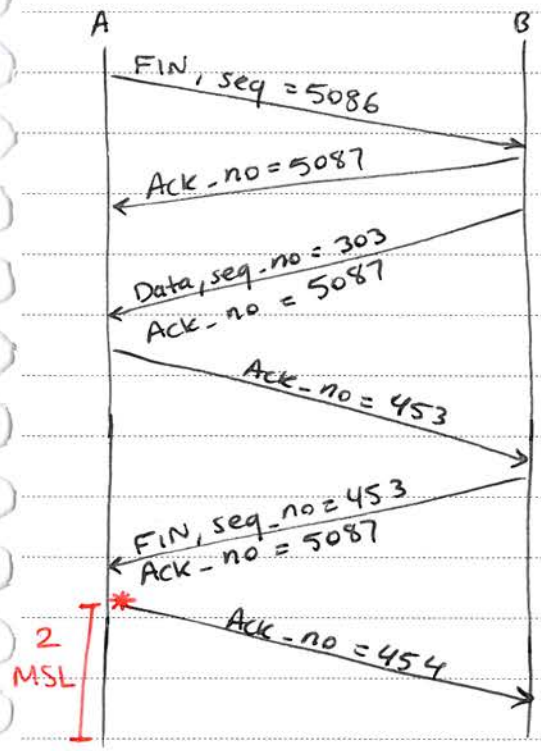


باید برای اولین segment ای که می‌فرستد Ack. بگیرد و window size را بگیرد و دیگر نمی‌تواند بیشتر از window size را بفرستد.

Connection Release

TCP، connection را به آهستگی می‌بندد تا داده‌ای از قبل در شبکه نماند: graceful close

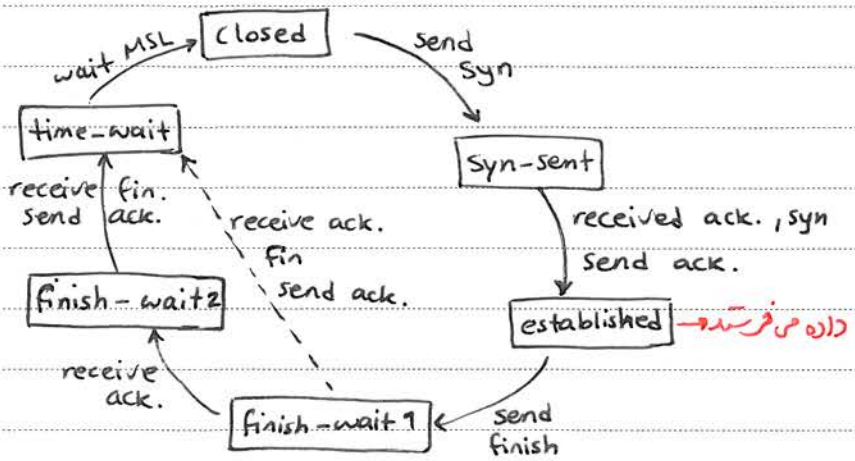
Client که ارتباط را شروع کرده، تمام‌کننده ارتباط نیز هست.



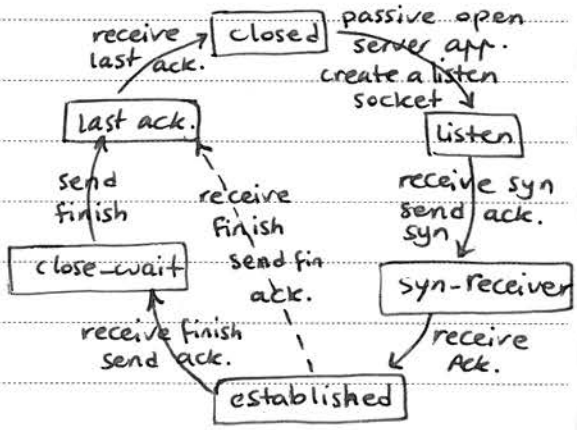
B ممکن است داده‌ای داشته باشد یا نداشته باشد. اگر داده‌ای داشته باشد، ارتباط سمت خودش را می‌بندد، ولی یک Ack. به A می‌فرستد که پیامت را دریافت کردم، A ارتباط داده‌ای اش را می‌بندد.

* اگر B به A داده‌ای داد، A باید جوابش را بدهد. وقتی دقیقاً Ack. A می‌دهد، سریع ارتباط را می‌بندد، و timer به اندازه 2 برابر Maximum Segment Lifetime (MSL) روشن می‌کند تا اگر داده‌ای مانده است برسد.

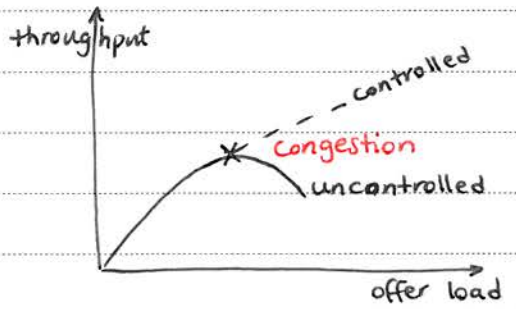
TCP Client Life Cycle



TCP Server Life Cycle

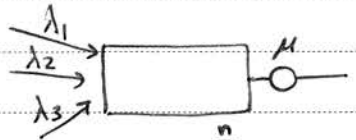


TCP Congestion Control : در TCP به روش reactive (closed loop) است. ازدحام باعث کاهش نرخ گذردهی می‌شود. اگر مجموع جریان های ترافیکی از ظرفیت فیزیکی لینک بیشتر شود، congestion رخ می دهد. اگر ازدحام کنترل نشود، ازدحام خود را تشدید می کند، زیرا اگر یک packet نرسد، با time out شدن، packet دوباره فرستاده می‌شود و ترافیک افزایش یافته و ازدحام تشدید می یابد.



throughput، offer load آرمانی خود رو به رو با افزایش کاهش یابد، packet loss داریم یعنی ازدحام رخ می دهد. ازدحام اگر کنترل نشود می‌تواند باعث ایجاد dead lock شود.

در reactive closed-loop با offer load، کاهش دهیم. feedback به دو صورت صریح و ضمنی صریح صورت می‌گیرد. هر پروتکلی با packet loss می فهمد ازدحام رخ داده است.



$$\lambda = \sum_{i=1}^{n} \lambda_i$$

Network Layer :

شبکه هایی که به صورت ارسال بسته کار می کنند، دو نوع لایه ی Network داریم :

(Virtual Circuit) packet switching connection oriented.

(Datagram) " " connection-less.

آیا لایه ی Network می خواهد کیفیت سرویس (Quality of service) را تضمین کند یا نه؟ سرویس ها دو نوع هستند :

- Q.S را تضمین می کنند : باید یکسری منابع را برای ارتباط رزرو کنند و برای آن سرویس آزانگذ دارند.
- " " " " می کنند ≡ best effort

Q.S تضمین می شود ← Connection Oriented *

قبل از این که ارتباط برقرار شود باید منابع لازم کاربر مشخص شود تا لایه ی شبکه منابع لازم را آماده کند.

* زیرا عموماً قبل از پاسخ به سرویس یک ارتباط برقرار می شود تا منابع مشخص شود و آماده در اختیار قرار دادن سرویس شود.

X.25 , Frame_Relay , ATM: Virtual Circuit

Internet : Datagram

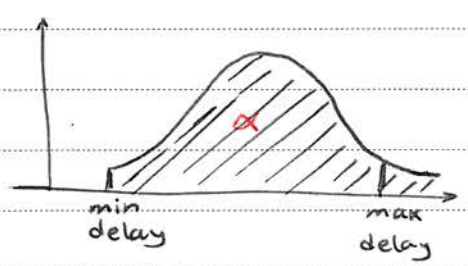
* هرگاه در شبکه تاخیر ایجاد می کند که ناشی از چهار عامل است و تاخیر end to end مجموع ۴ تاخیر فوق هاست :

1. Propagation Delay — تاخیر انتشار که با افزایش فاصله نسبت مستقیم دارد. (جنس رسانه)
2. Transmission Delay — به اندازه بسته و نرخ ارسال بستگی دارد. (سرعت انتشار امواج)
3. Process Delay — به حجم پردازش و پردازنده بستگی دارد. پردازنده قوی زمان کمتری نیاز دارد.
4. Queue Delay — به میزان ترافیک شبکه و همچنین باند لینک خروجی بستگی دارد.

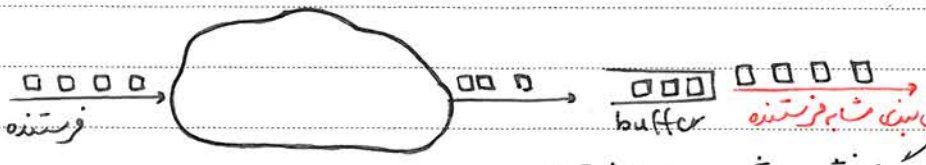
برای application هایی باند، delay، packet loss و ترتیب ارسال مهم است. delay را معمولاً چند نوع بیان می کنند :

delay از دید متصد قابل پیش بینی نیست و یک متغیر تصادفی است : random variable

هر متغیر تصادفی یک تابع توزیع احتمال دارد.



1. α ای تعریف می شود برای احتمال وقوع delay max
2. t_{ad} (average delay) : نسبت به میانگین انحراف معیار بسته ها را داریم که اگر این مقدار زیاد باشد باید نگران کننده بزرگ باشد. به این تغییرات delay می گویند timing jitter.



واریانس هرچه با میانگین نزدیکتر باشد، عموماً بهتر است. واریانس اگر کمتر باشد یعنی اختلافات زیاد است ولی اگر واریانس کم باشد، اختلافات کم است.

حساسیت های application ها:

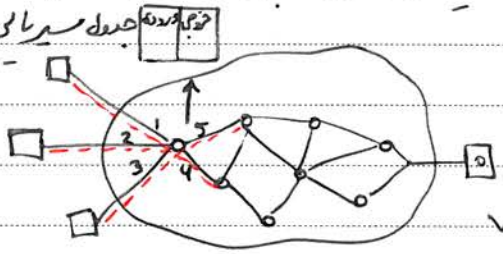
1. احتمال packet loss
2. Max Delay
3. Delay Variation

اگر به تاخیر حساس نباشند مقدار max را خیلی زیاد می گذارند که don't care تبدیل می شود.

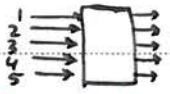
شبکه یک واحد کنترل پذیرش دارد که مسیری را پیدا می کند که نیازمندی های بالا را برآورده کند. در روش مدار مجازی هم بسته ها از یک مسیر عبور می کنند (مسیری که کیفیت سرویس داشته است.)

حسن ایجاد مدار مجازی علاوه بر همپای ماند، حجم سربایی کم می شود زیرا هر بسته نیاز به سربایی ندارد و از این مسیر عبور می کند. یعنی حجم پردازش سربایی در گره های میانی کم می شود. این جا ما از ظرفیت شبکه 100٪ مفید نمی توانیم استفاده کنیم ولی در عوض کیفیت سرویس داریم.

هر بار درخواست برقراری ارتباط می آید یک مسیر مجازی به وجود می آید و یک ورودی وارد جدول می شود و هنگامی که این ارتباط release شد، این ورودی از جدول پاک می شود.



هر virtual circuit یک عدد اختصاص می دهیم به نام Virtual circuit identifier: روی هر لینک ممکن است چندین VC به وجود بیاید که از روی VCI آنها را تشخیص می دهیم: روی header بسته ها VCI را می نویسیم.



line card

جدول سربایی: به ازای هر پورت یک جدول داریم.

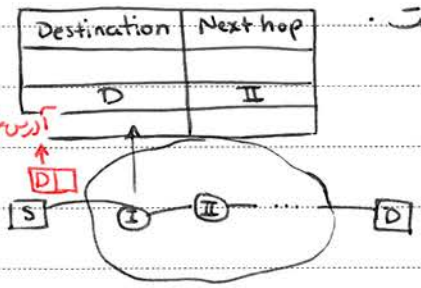
ورودی		خروجی	
Link No.	VCI	Link No.	VCI
1	1	4	2

ممکن است قبلاً روی لینک 4، عدد 1 را به یک VCI دیگر نسبت داده باشیم، پس ID آن را عوض می کنیم و 2 می گذاریم. یعنی ممکن است یک VCI روی هر لینک یک ID داشته باشد. حسن آن این است که VCI اول را کد نمی داریم چون همان index جدول است. بنابراین تاخیر پردازش کم و سرعت زیاد می شود.

Connection Oriented Packet Switching :

* چون از قبل هماهنگی باند را می داند ← منابع را می داند ← ازدحام به وجود نمی آید ← packet loss نداریم ← queuing کم است ← تاخیر کم است ← کیفیت سرویس بهتر است.

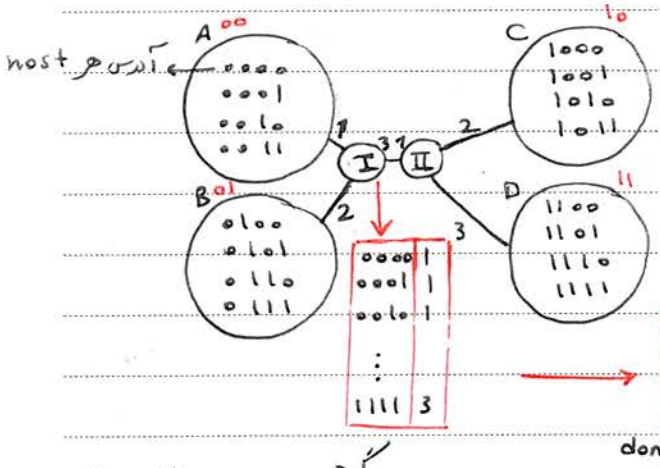
* عمده ترین تاخیر router ها در IP ، search کردن در جدول است.



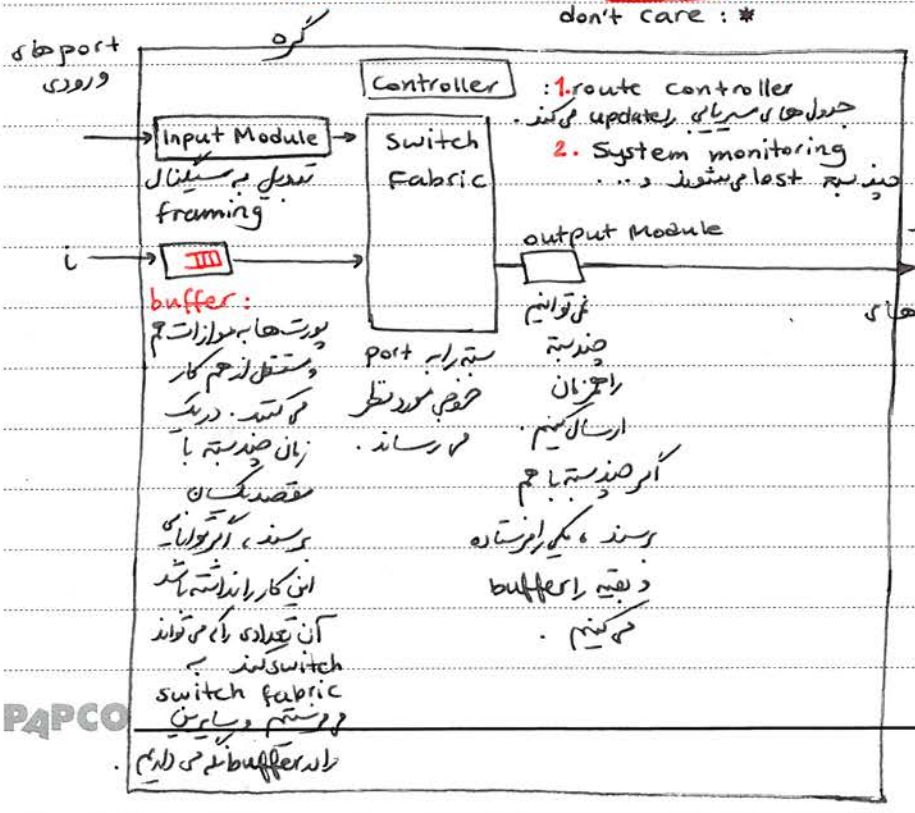
Connection- less Packet Switching :

پیشوند آدرس : تمام کامپیوترهای یک شبکه پیشوند آدرس یکسانی دارند. به همین دلیل اندازه ی جدول کوچک می شود. وقتی آدرس ها سلسله مراتبی باشد، آدرس یابی راحت تر است.

مثال : 4 شبکه داریم، هرکدام 4 host دارند.



* جدول از 16 سطر به 3 سطر کاهش پیدا کرد!



معماری داخلی Router :

2. Management Controller ← چک می کند که آیا router کارش را درست انجام می دهد یا نه!

Input/Output M., switch Fabric

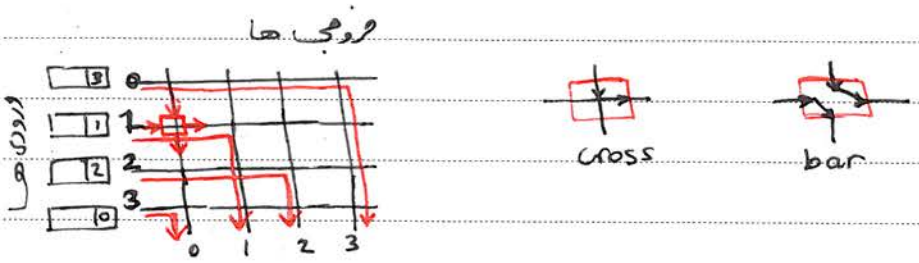
انتقال بسته ها

Controller

کنترل مدیریت switch

Space Division:

cross-bar switch



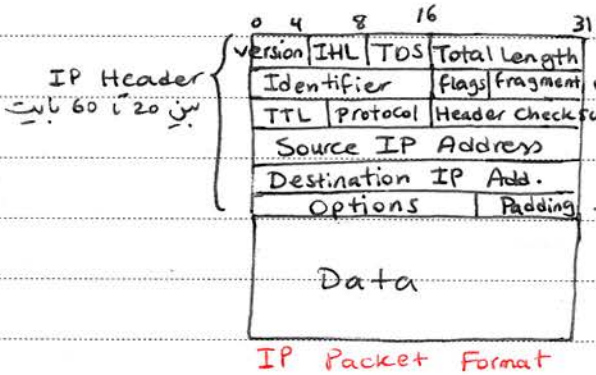
وظایف گره سریاب:

(Input, Switch Fabric, output.) 1. Data Path Plan packet forwarding

(route control, management control) 2. Control Plan جداول سریاب updating

Packet Forwarding با استفاده از بررسی header بیت های IP صورت می گیرد:

Internet Protocol



IP Packet Format

IP Header Length : IHL

طول header مضربی از 32 بیت است.

Type of Service : ToS

مشخص می کند Data چه پارامترهایی از کیفیت سرویس برایش مهم است:

throughput، reliability، cost، delay و 3 بیت اولویت!

optional حداکثر 40 بایت

Time to live : TTL

این بیت وقتی وارد شبکه می شود باید تا یک زمان معقول به مقصد برسد. در غیراین صورت از بین می رود. زمان زنده بودن بسته را بیان می کند. مفهوم آن hop limit است، یعنی هر بسته چند گام حداکثر باید طی کند تا به مقصد برسد. اگر گره ای بسته ای را دریافت و از TTL یک واحد کم کرد، بسته را حذف می کند و برای گره مبدأ پیغام خطا می فرستد.

Protocol: data ای که بسته IP حمل می کند توسط چه پروتکلی تولید شده است: TCP، UDP و...

به هر پروتکل یک عدد اختصاص می دهیم. مثلاً TCP 6 است پس اگر Protocol field عدد 6 باشد، data یک TCP Segment است:

1 ICMP

6 TCP

17 UDP

4 IPsec

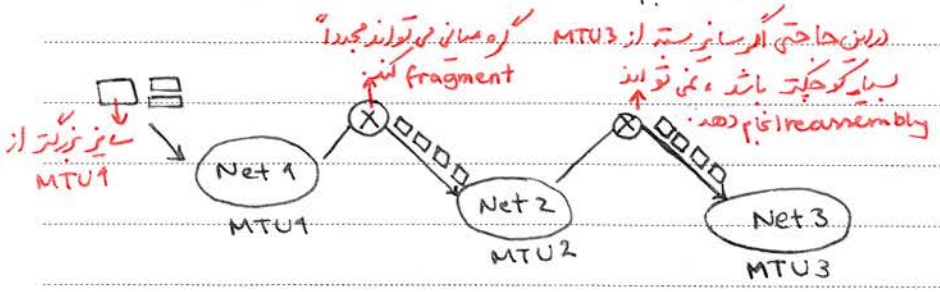
مرحله در مقصد داده را به چه پروتکلی باید بدهد.

Maximum Transfer Unit: MTU

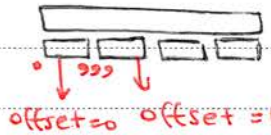
ماکسیمم سایز بسته‌ای که از لایه‌های بالاتر می‌گیرد.

اگر اندازه‌ی بسته‌ی IP بزرگتر از MTU باشد، باید آن را fragment کند.

* در IP، گره‌های میانی هم می‌توانند fragmentation را انجام دهند ولی assemble کردن فقط در گره‌های پایانی رخ می‌دهد.



Fragment offset :



هربیت نسبت به شروع چند offset دارد :

offset یک عدد 16 بیتی است ولی در IP header 13 بیت در نظر گرفته اند. بسته IP 64KB است ⇐ 16 بیت برای offset لازم!

قرار داد: از جای fragment کنیم که سه بیت اول همیشه صفر باشد. پس بدین ترتیب 3 بیت اول را نمی‌نداریم.

Flag ← MF : more fragment
1 : این fragment ای دریافت کردی آخرین نیست.
0 : این آخرین است و می‌توان عمل reassembly را انجام دهی.

اگر آخرین fragment را دریافت کردی هنوز قسمتی مبانی نامده است ← timer روشن می‌کنیم و اگر time out شد، کل بسته را دور می‌ریزیم.

Identification : مبدأ برای هربسته‌ای که می‌فرستد یک ID می‌گذارد. تا مشخص شود هر fragment مال کدام بسته است (تمام fragment های یک بسته بدون توجه به تعداد بار fragment شدن، ID یکسان دارند.)

پس از 2^{16} بار فرستادن بسته دوباره می‌توانند ID ها را از ابتدا شروع کنند.

MTU = 1200

fragment = 1200 − 20 = 1180 | 8
8 | 147
38
32
60
56
4

⇒ 1180 − 4 = 1176 byte

مثال ۱ :

	T.L	ID	MF	F.O	
original packet	3220	X	0	0	
fragment 1	1196	X	1	0	
fragment 2	1196	X	1	147	×8
〃 3	968	X	0	244	×8

! شده fragment

Flag ← reserve : برای استفاده بعدی قرار داده شده است.

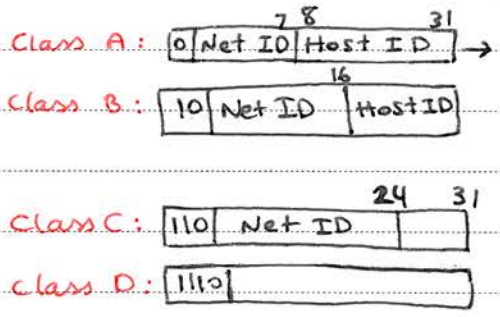
Don't Fragment : DF ←

اگر گره ای بسته ای را دریافت کرد که بیت DF آن 1 است ولی مجبور است به دلیل MTU ، fragment کند ، اجازه ندارد پس بسته را lost می کند . پیغام خطا می فرستد به مبدأ .

شبکه های کلاس A : شبکه های بسیار بزرگ با تعداد host بسیار بالا

" " B : " " بزرگ " " " زیاد

" " C " " کوچک " " " کم



حداکثر 2^{24} host می توانیم داشته باشیم : اگر اولش صفر نبود با 8 بیت ها چک می کرد

آدرس گروهی برای سرویس های multicasing

یک مبدأ برای K مقصد سیگنال را می فرستد (ایستگاه رادیویی)

مشکل : تعداد زیادی از شبکه ها در کلاس B افتادند ، به همین دلیل آدرس ها سریع پر شدند .

راه حل / اندازه field آدرس از 32 بیت بیشتر کنیم تا به 128 بیت تبدیل شود . این کار مستلزم تغییر IP در تمام شبکه ها (در میزبان ها و گره های انتهایی) بود . تا زمانی که این راه حل عملی شده است ↺

راه حل موقت :

⋮

C کلاس 8 : 1024 - 2048

⋮

* آدرس 32 بیتی dot decimal است . یک شبکه را با آدرس IP ، Network mask ، Net ID تعریف می کنیم .

ICMP: Internet Control Message Protocol

در IP، اگر خطا یا استثنایی رخ دهد IP نتواند کارش را به درستی انجام دهد، یک گره ICMP گره ای که بسته از بین برده است ایجاد می‌شود و برای source فرستاده می‌شود. پس IP از این پروتکل برای کنترل پیام ها استفاده می‌کند. اگر source بتواند handle کند این کار را انجام می‌دهد و در غیر این صورت به admin وآگاه می‌کند.
• مثلاً بسته هیچ entry پیدا نمی‌کند که با مقصد آش جور باشد. یا شاید پروتکل مورد نظر در گره ای که بسته از آن عبور می‌کند وجود نداشته باشد.

این پروتکل پیغام های خطایی را که در لایه‌ی IP اتفاق می‌افتد، انتقال می‌دهد. خود این پیغام داخل data field یک بسته IP قرار می‌گیرد و protocol ID آن را 1 می‌گذاریم.

Type	Code
3	0: destination network unreachable شبکه مقصد وجود ندارد
	⇒ بسته حذف می‌شود ولی یک پیغام خطا با type 3 و code 0 برای source فرستاده می‌شود.
3	1: destination host unreachable کامپیوتر مقصدی وجود ندارد.
3	2: " protocol " به مقصدی رسیده است که این پروتکل در آن وجود ندارد.

* ICMP 16 بایت اول بسته را نیز برای source می‌فرستد زیرا این 16 بایت اطلاعات لایه‌ی بالاتر است تا source بفهمد این بسته از کجا آمده است.

echo request: type: 8 code: 0 این پیغام فقط connectivity را چک می‌کند. مقصد با گرفتن این پیغام باید reply کند. (مثل ping)

echo response: type: 0 code: 0 ← زمان رفت و برگشت

Trace Route: IP آدرس های گره های میان تا مقصد را به ما می‌دهد.
یک packet می‌فرستد، و TTL آن را 1 می‌گذارد گره اول از بین می‌رود و پیغام ICMP ای که می‌فرستد، آدرس خود را هم فرستد. بعد TTL را 2 می‌گذارد و دومین گره را هم پیدا می‌کند. اگر echo response باشد، گره آخر را پیدا کرده است. default 30 گام است. این در صورتی است که firewall وجود نداشته باشد که ICMP را فیلتر کند. این firewall به دلیل این است که قدرت پردازشی سرور با کارهای بیهوده کاهش نیابد.

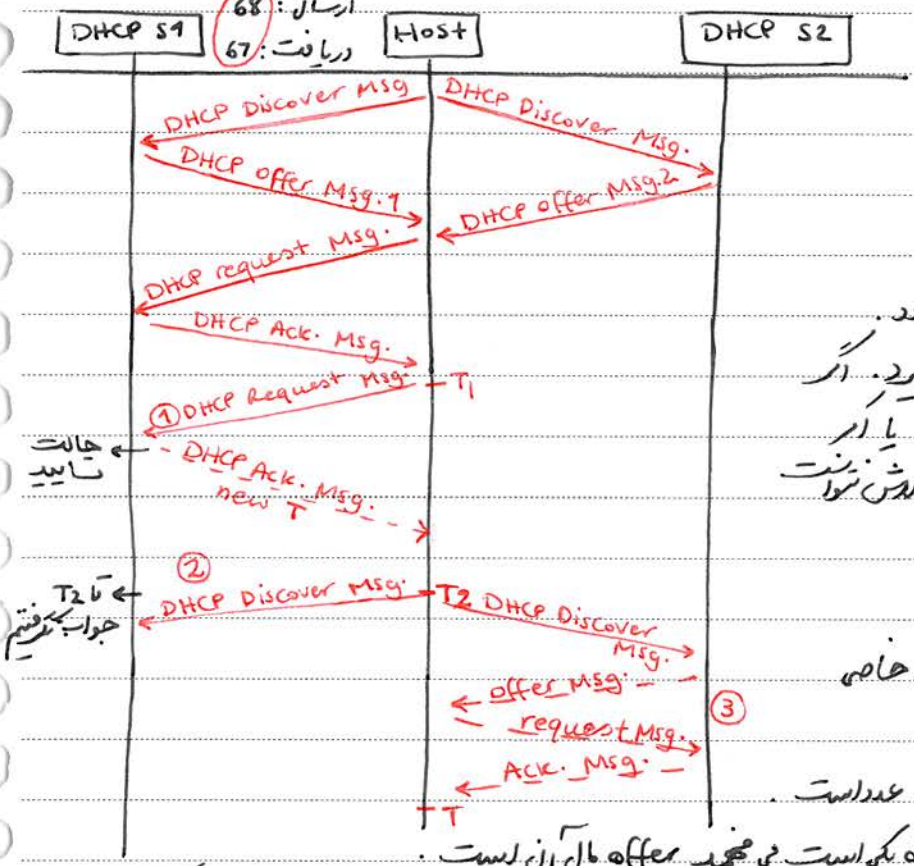
TTL expired: type: 11 code: 0

DHCP: Dynamic Host Configuration Protocol به هر سرور آدرس unique می‌دهد.

پروتکلی است که هم از آن آدرس می‌گیرند و وقتی هر سرور خاموش می‌شود، آدرس را به دیگری اختصاص می‌دهد.

هر کره‌ای که می‌خواهد به شبکه وصل شود باید یکسری تنظیمات اولیه روی آن انجام شود. یا می‌توانیم دستی تنظیم کنیم. این کار یکسری مشکلات دارد. گاهی وقت‌ها این توانایی را ندارد.

برخی DHCP server ها این امکان را دارند که اگر خودشان نمی‌توانند آدرس assign کنند از یک سرور دیگر بگیرند، assign کنند. آدرس‌های range هر سرور DHCP باید با سرور DHCP دیگر متفاوت باشد.



سرویس‌گیرنده‌ای که هنوز IP آدرس نگرفته است ولی می‌خواهد ارسال شود، قسمت 32 بیت را صفر می‌گذارد. پس سرویس‌گیرنده‌ای که IP Add. آن هم اش صفر است، IP آدرس موقتی گرفته است تا زمانی که بهش assign شود.

offer Msg. پیشنهادی می‌دهد که چه آدرسی را بگیرد. اگر offer نگیرد یا DHCP سرور وجود ندارد یا اگر وجود دارد، IP آزاد ندارد. پس از چند بار تلاش شروع به وصل شود، دیگر نمی‌تواند.

اگر یک offer باید، همان را می‌پذیریم.

اگر بیش از یک offer باید. با مکانیزم‌های خاصی تصمیم‌گیری می‌کنیم.

Reference No.: برای هر Host یک عدد است.

Host ای که reference No. ای که می‌گیرد با discover یکی است می‌فهمد offer مال آن است.

T: زمانی که سرور به Host گفته است آدرس به او تعلق دارد. Host باید سه timer set کند:

① وقتی T_1، time out شد به همان server درخواست می‌دهیم که آدرس را renew کند $T_1 = 0.5T$

② از سرور اول ناامید می‌شویم چون تا الان جواب نداده است. پس discover می‌کنیم. $T_2 = 0.875T$

③ اگر تا زمان T نتوانستیم هیچ آدرسی بگیریم باید آدرس را رها کنیم تا به range آدرس‌های آزاد سرور برود. T

برای این که بفهمیم Host ای در شبکه ماست یا نه، باید آدرس را با network mask و and کنیم.

192.168.31.50
255.255.255.0
⇒ 192.168.31.0 : آدرس شبکه

ARP: Address Reservation Protocal

یک جدول با اسم ARP table داخل خود دارد:

IP Address	Physical Address
192.168.31.50	...

هر موقع که IP آدرس، mac add. مقصد را سوال می‌کند، یا جواب می‌دهد (اگر mac add. را داشته باشد) و اگر نداشته باشد، یک ARP Request در شبکه broadcast می‌کند، IP Add. را request می‌گذارد و کسی که mac آن کی است ARP Response می‌فرستد، به کسی که request را فرستاده است. از آن لحظه به بعد آدرس فیزیکی آن ثبت می‌شود. از آن جایی که IP Address تغییر می‌کند نمی‌توان این جدول را برای همیشه داشت. این جدول تا زمانی که مطمئن هستیم تغییری رخ نمی‌دهد معتبر است که معمولاً زمانی بین 3 تا 15 دقیقه است. پس از آن جدول را پاک می‌کنیم و روال را از ابتدا شروع می‌کنیم.

حالتی که بسته به مقصدی داخل همان شبکه می‌رود:

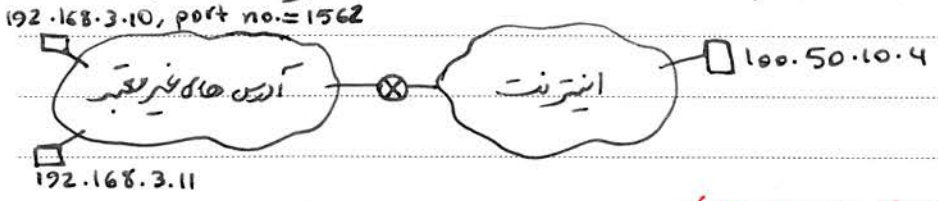
بسته خارج از شبکه: باید به gateway بدهیم. از طریق ARP، gateway IP Add. را داریم، mac آن را درست می‌آوریم و بسته را گام به گام جلو می‌بریم تا به مقصد برسد. زیرا gateway داخل آن network هم هست.

local host — آدرس خودم — 127.0.0.1 — بسته‌ای که به این آدرس می‌دهیم به خودمان برمی‌گردد. برد کارهای تستی می‌خورد.

آدرس‌های invalid اینترنت — این آدرس‌ها فقط محلی است (VPN,...)
- 192.168.x.x
- 10.x.x.x

آدرس‌ها را بین providerها (سرویس‌دهنده) تقسیم می‌کند. شرکت‌های ISP کوچک آدرس‌شان را از ISPهای بزرگ‌تر می‌گیرند. تعداد آدرس‌ها با ظرفیت بینک متناسب است زیرا متناسب با ظرفیت بینک، hostهای بیشتری داریم.

← زمانی که آدرس‌های در اختیار ما کم است، یکسری از آدرس‌های invalid اینترنت را تولید می‌کنیم که زمانی که در اینترنت نسبت از این‌ها استفاده کنند. Nat تبدیل آدرس را برای ورود به اینترنت انجام می‌دهد و در مسیر router خروجی قرار دارد.



NAT (Network Address Translation):

علاوه بر تبدیل آدرس از غیرمعتبر به معتبر، برای TCP باید بتواند آدرس معتبر را نیز به غیرمعتبر تبدیل کند و به همین دلیل آدرس را باید نگه دارد ← NAT table

در TCP در NAT خیلی راحت‌تر است. TCP ابتدا SYN می‌فرستد (درخواست connection setup).
NAT به آدرس مبدأ نگاه می‌کند. ابتدا header را نگاه می‌کند تا ببیند TCP است یا UDP. port no. مبدأ را نیز نگاه می‌کند:

NAT table

192.168.3.10	1562	100.50.10.4	2430	TCP
192.168.3.11	1560	100.50.10.4	2431	TCP
192.168.3.12	80	"	"	"

با استفاده از این جدول به راحتی می‌توانیم این کار را انجام دهیم. هنگامی که connection close می‌شود entry را پاک می‌کنیم.

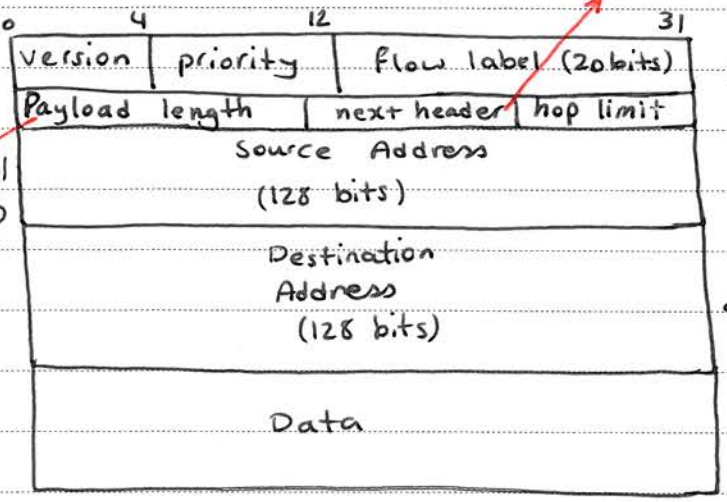
در UDP نمی‌دانیم کی باید entry را پاک کنیم. راه حل ← بازمان (تنظیم توسط admin)

Static NAT: ↑ یک سرور invalid و یک app. ای که روی یک پورت invalid هست را NAT کنیم. admin دستی این entry را به جدول اضافه می‌کند که دیگر پاک نشود (dynamic نباشد)

اشکالات IP، Version 4:

1. در header، field های بلا استفاده وجود دارد: identification، fragment، ...
راه حل در Version 6، عنوان option به آنها نگاه می‌شود.
2. سایز header حداکثر 60 بایت است و اگر ما بخواهیم option های زیادی استفاده کنیم به مشکل برمی‌خوریم.
3. security اصلاً وجود ندارد و یک router میانی می‌تواند header و ... را دستکاری کند.
راه حل در Version 6 یک پروتکل طراحی کرده‌اند که security از option های آن است.
4. بسته‌ها حداکثر 64 کیلو بایت هستند.
راه حل در Version 6 بیشتر از 64KB هم می‌توانیم بفرستیم.
5. TTL عملاً hop limit انجام می‌داد.
در Version 6 اسم واقعی آن برداشته شد ← hop limit
6. کیفیت سرویس end-to-end مشخص نیست زیرا بسته‌ها جدا از هم فرستاده می‌شوند.
راه حل در Version 6 یک field به نام flow table اضافه کردند که بسته‌های متعلق به یک جریان ترافیکی کیفیت سرویس یکسان داشته باشند.

Header IP version 6 :



همان field پروتکل است. الزاماً به TCP یا UDP اشاره نمی‌کند.

اگر option وجود داشته باشد، به آن اشاره می‌کند.

field اندازه داده (16 bits)

نکته: fragmentation، اگر عددی را برای پروتکل استفاده نکرده‌ایم برای next header آن option می‌گذاریم و اگر داخل آن option دیگری بود باز به همین ترتیب و... :

اگر option نداشته باشیم، next header همان پروتکل است.

- IP Checksum field نیز حذف شده است. اگر بسته‌ای خطا داشته باشد به لایه‌ی IP نمی‌رسد. زیرا همه‌ی Network Interface ها خودشان خطا چک می‌کنند. (Ethernet، wireless، ...)

- در این version، گره میانی حق fragment ندارد. یعنی fragmentation فقط در source می‌تواند انجام شود و assembly فقط در گره مقصد. این کار سرعت پردازش بالاتر می‌رود. زیرا این کار سرعت بالایی دارد و استفاده از لینک‌های با ظرفیت بالا بهینه‌تر است. پس source باید روی مینیمم MTU، عمل fragment را انجام دهد. باید از مکانیزمی خارج از ... آن را بدست بیاورد. (از طریق admin و...) و اگر اشتباه تخمین زده شود، Packet ازبین می‌رود و پیغام خطا می‌آید و اصلاح می‌شود.

پیاده سازی IP Version 6 :

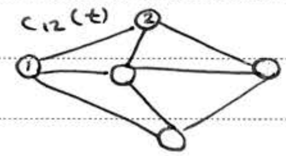
راه حل پیشنهادی : admin می‌تواند در ناحیه شبکه محلی هر را به version6 تبدیل کند و شبکه می‌تواند برای وصل شدن به شبکه بیرونی می‌تواند Convert کند. قانون تبدیل آدرس‌ها تعریف کردند.

tunneling protocol شبکه Version 6 را به هم وصل کنیم که بین آنها یک شبکه Version 4 است. به این صورت، gateway مبدأ به مقصد tunnel می‌زند.

مسیریابی:

شبکه دوست دارد لینک هایی که خلوت ترند استفاده شوند: با maximum througput محدودیت های کاربر را در نظر می‌گیرد و مسیریابی می‌کند. وضعیت لینک ها dynamic است. به صورت dynamic یک cost را به لینک ها assign می‌کنیم:

$c_{12}(t) = f(\underline{wf}, \underline{d}, \underline{r}, \dots)$
پهنای باند آزاد، delay، reliability



⇐ بهترین مسیر، مسیری است که کمترین هزینه را داشته باشد.

الگوریتم های مسیریابی:

1. Static ایستا
2. Dynamic (Adaptive) پویا (تطبیقی)

نباید loop داشته باشد چون نمی‌تواند مسیریابی کند. برای هر مبدأ تا مقصد حداقل یک مسیر را بتواند پیدا کند. خودش را با تغییرات شبکه تطبیق دهد. پیچیدگی محاسباتی نداشته باشد.

Static: برای شبکه هایی که نرخ تغییرات کم است. یکبار مسیریابی را انجام داده و جداول مسیریابی را درست کرده و در گره ها قرار داد.
مثال: مراکز تلفن (تغییرات در شبکه های تلفن کم است)

Dynamic:
- Centeralized (متمرکز): تغییرات شبکه به گره ای مرکزی داده می‌شود. آن گره الگوریتم را اجرا می‌کند و جداول را درست می‌کند و به گره ها می‌دهد. در شبکه بزرگ اگر این گره اتفاقی برایش بیفتد، کار شبکه مختل می‌شود.
- Distributed (توزیع شده): پس توزیعی انجام می‌دهیم. هر گره ای برای خودش جدول مسیریابی ایجاد می‌کند. به عنوان مثال دو گره ای که متصل به یک لینک هستند، وضعیت لینک را متوجه می‌شوند (ترافیک، تاخیر، ظرفیت و ...). تغییراتی را که در یک نقطه تشخیص دادیم به گره های مجاور اطلاع می‌دهیم و به همین ترتیب پس از مدتی همه گره ها متوجه می‌شوند و هر گره ای جداول مسیریابی اش را update می‌کند.

شبکه های با نرخ تغییر بالا: Dynamic(distributed)

الگوریتم های Shortest path با کمترین هزینه:
- centeralized: اطلاعات کل شبکه را داشته باشیم تا بتوانیم کوتاهترین مسیر را پیدا کنیم: dyjestra
- : به محلی داشته باشیم کافیست: bermanford

الگوریتم های Link State: شبیه dijestra هستند، یعنی باید topology کل شبکه را ابتدا بدانند تا بتوانند کوتاهترین مسیر را پیدا کنند.

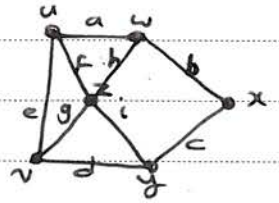
توپولوژی: $G(V,E)$

V: مجموعه گره ها (رأس ها) گراف

E: مجموعه لینک های شبکه

اگر مجموعه لینک ها را داشته باشیم (چون هر لینک دو گره را به هم وصل می کند) آن وقت توپولوژی شبکه را داریم.

مثال:



$V=\{u,w,x,y,z,v\}$

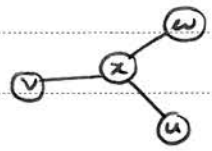
$E=\{a,b,c,d,e,f,g,h,i\}$

الگوریتم های Distance Vector: شبیه Bellman-ford. یعنی نیازی به توپولوژی کل شبکه نداریم برای مسیریابی.

تعداد گره های شبکه $n=|V|$

$D.V. = [\underbrace{\quad\quad}_{n \text{ عضو}}]$

همان جدول مسیریابی است. (می توانیم از گره همسایه ۰ به گره مقصد برویم با هزینه □ !) در نتیجه جداول مسیریابی گره های همسایه را می گیرد و خود را update می کند.



⇒ همین طور ادامه می یابد!

x از همسایه هایش سوال می کند که فاصله هر کدام از z چقدر است. سپس فاصله خودش را تا آن گره حساب می کند و هر کدام جمع فاصله هایش کمتر شد انتخاب می کند.

$$D_x = \min\{c_{xv}+D_v, c_{xu}+D_u, c_{xw}+D_w\}$$

Dijestra Algorithm:

1. Initiation

c_{ij}: هزینه لینک مستقیم گره i به j ⇒ اگر گره ای لینک مستقیم نداشته باشد، این مقدار ∞ است.

D_j: فاصله گره مبدأ s به گره مقصد j

N: مجموعه ای که هر گره ای را در مسیر پیدا کردیم به آن اضافه می کنیم. اگر برابر مجموعه همه گره ها شود، الگوریتم متوقف می شود.

Step 1: $N=\{s\}$

$D_s = 0$

$D_j = c_{sj}$

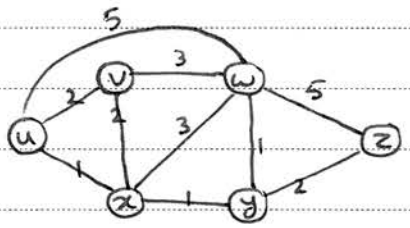
Step 2: $D_i = \min_{j \neq N} \{D_j\}$. به شرطی که قبلاً i را انتخاب نکرده باشیم.

Add i to N, if N contains all Node, stop

Step 3: updating $D_j = \min_{j \neq N} \{D_j, C_{ij} + D_i\}$

go to Step 2.

مثال :

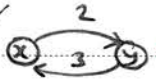


مبدأ u :

	u	v	w	x	y	z
u	0	2	5	1	∞	∞
v	2	0	3	2	∞	∞
w	5	3	0	3	1	5
x	1	2	3	0	1	∞
y	∞	∞	1	1	0	2
z	∞	∞	5	∞	2	0

= C هزینه لینک مستقیم

* اگر مسیر رفت و برگشت در نظر گیریم : $C_{ij} = C_{ji}$



Iteration	N	D_v	D_w	D_x	D_y	D_z
Initial 0	u	2,u	5,u	1,u	∞	∞
1	u,x	*,u	**,x		2,x	∞
2	u,x,v		4,x		2,x	∞
3	u,x,v,y		***,y		****,y	
4	u,x,v,y,w				4,y	
final 5	u,x,v,y,w,z					

u: S $\Rightarrow D_u = 0$

* $D_v = \min\{D_v, D_x + C_{xv}\}$
$= \min\{2, 1+2\} = 2$

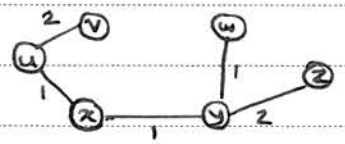
** $D_w = \min\{D_w, D_x + C_{xw}\}$
$= \min\{5, 1+3\} = 4$

*** $D_w = \min\{D_w, D_y + C_{yw}\}$
$= \min\{4, 2+1\} = 3$

**** $D_z = \min\{D_z, D_z + C_{yz}\}$
$= \min\{\infty, 2+2\} = 4$

می توانستیم y را هم انتخاب کنیم.

Topology :



جدول مسیریابی :

آدرس مقصد	N.H.	Cost
x	x	1
v	v	2
w	x	3
y	x	2
z	x	4

Bellman-ford: روش مقصد d

Step 1) Initialization: $D_i = \infty$, $i \neq d$ — برای همه رئوس تا یک مقصد مشخص را به دست می‌آوریم.
برای همه رئوس
$D_d = 0$

Step 2) for all $i \neq d$

$$D_i = \min_{j \neq i} \{ C_{ij} + D_j \}$$

stop when no more changes occur

otherwise go to step 2,

$d = z$, $D_z = 0$

Iteration	D_u	D_v	D_w	D_x	D_y	
0	∞	∞	∞	∞	∞	
1	*	**	5,z	∞	2,z	→ ✓ تغییر
2	10,w	8,w	3,y	3,y	2,z	→ "
3	4,x	5,x	3,y	3,y	2,z	→ "
⋮						

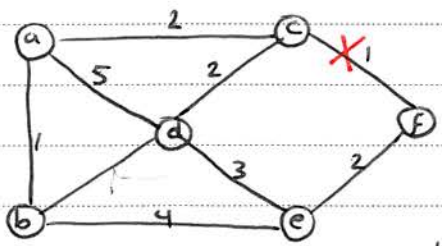
$$* \begin{cases} (C_{ux} + D_x, x) \\ (C_{uv} + D_v, v) \\ (C_{uw} + D_w, w) \end{cases} \Rightarrow \min = \infty$$

$$** \begin{cases} (C_{vx} + D_x, x) \\ (C_{vw} + D_w, w) \\ (C_{vu} + D_u, u) \end{cases} \Rightarrow \min = \infty$$

* حتماً بهترین جواب همگرا می‌شود. (تا بی‌نهایت الگوریتم پیدا می‌کند)
به یک شرط: درصورتی که گراف متصل باشد connected (مسیر وجود دارد.)

← بهترین مسیر با dijestra یکی می‌شود.

مثال:



destination = f

	a	b	c	d	e
before break	3,c	4,d	1,f	3,c	2,f

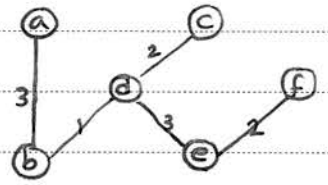
درصورتی که یکی از لینک‌ها قطع شود، الگوریتم Bellman-ford چگونه جدول را بازسازی می‌کند.

وقتی یک لینک قطع شود، روتر مجاور قطع شدن این لینک را تشخیص می‌دهند. روتر به همسایه‌ها اطلاع می‌دهند (در الگوریتم dijestra) و مجدداً بهترین مسیر را پیدا می‌کنند. در الگوریتم Bellman-ford باید از طریق همسایه‌هایش مسیری پیدا کند. c و a و d نگاه می‌کند:

:	a	b	c	d	e
1	3,c	4,d	5,a	3,c	2,f
2	7,b	4,d	5,a	5,b	2,f
3	7,b	6,d	7,d	5,b	2,f
4	9,b	6,d	7,d	5,e	2,f
5	9,b	6,d	7,d	5,e	2,f

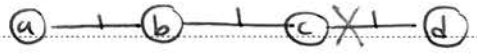
* چندبار در loop می‌افتد و سپس با update از loop در می‌آید. الگوریتم distance vector دیرهمگرا می‌شوند و سرعت همگرایی‌شان کند است در صورتی که الگوریتم‌های link state همه مطلع می‌شوند و همین دلیل سریع update می‌شوند.

آگر سادی شدند، آن‌ها ID کوچکتری دارد:



Link State	Distance Vector
x متمرکز	✓ توزیع شده
✓ سرعت همگرایی بالا	✓ هر گره یک دید کلی از اطلاعاتی که لازم دارد (در هر گره)
می‌توانیم معیار را بر اساس نیازمان ⇒ ✓ هزینه‌ای که لینک‌ها می‌دهیم می‌تواند متفاوت باشد.	محدود است.
تغییر دهیم.	x سرعت همگرایی کم
	x هزینه‌ی همه‌ی لینک‌ها ثابت است.

مثال: counting to infinity (زمانی که مسیری وجود ندارد و شبکه (گراف) چند قسمت می‌شود):



	a	b	c
before break	3,b	2,c	1,d
	3,b	2,c	3,b
	3,b	4,a	4,b
	5,b	4,a	5,b
	5,b	6,a	5,b
	7,b	6,a	7,b
	⋮		

بهبود اول : split horizon

اگر گره y از x سوال کرد که هزینه‌ات تا z چقدر است و z هم جدولش را نگاه کرد و دید که y گره بعدی مسیر است، جواب نمی‌دهد. چندبار که تکرار کرد پاسخی نگرفت می‌فهمد که مسیری وجود ندارد.

بهبود دوم : split horizon with poisened reverse

در این جا گره x می‌گوید هزینه ∞ است. یعنی هزینه واقعی را نمی‌گوید. در این حالت جدول سریع همگرا می‌شود.

مثال : مثال قبل با بهبود دوم :

a	b	c
3,b	2,c	1,d
3,b	2,c	∞,-
3,b	∞,-	∞,-
∞,-	∞,-	∞,-

Link State : این یک گره یک داده را به اطلاع همه‌ی گره‌ها برساند نیاز به یک الگوریتم دارد که معمولاً flooding routing algorithm است : هر گره‌ای که یک packet ای را گرفت به همه گره‌ها جز گره‌ای که آن packet را فرستاده است. در این الگوریتم هر گره هیچ اطلاعاتی از توپولوژی شبکه و وضعیت لینک‌ها ندارد به همین دلیل می‌تواند از این الگوریتم استفاده کند. مثلاً گره‌ای تازه به شبکه وصل شده است با استفاده از این الگوریتم می‌تواند جدول مسیریابی‌اش را تکمیل کند.

شبکه‌هایی که تحرک زیاد است (adhoc) به همین دلیل توپولوژی تغییر می‌کند و از الگوریتم‌های flooding استفاده می‌کنند. تضمین می‌کند که یک پیغام حتماً به همه می‌رسد و یک گره کمترین زمان را صرف می‌کند.

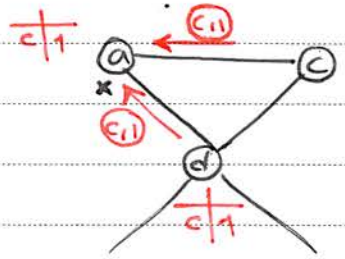
اشکال عمده : سربار خیلی بالاست زیرا به صورت نمایی پخش منتشر می‌شود. پس کارایی شبکه پایین می‌آید و منابع کم می‌شود. ممکن است بسته در حلقه هم بیفتد.

اصلاح : به بسته‌ها hop limit نام field ای اضافه می‌کنند به نام hop limit : بعد از چند گام همه باید بسته را گرفته باشند. به هر گره که رسید یک واحد کم می‌کند و وقتی صفر شد بسته را از بین می‌برد.

حلقه‌ها loop را می‌گیرند : هر گره آدرس خودش را به header اضافه می‌کند و اگر بسته را گرفت آدرس خودش را دید یعنی توی loop افتاده است پس آن را حذف می‌کند.

هم‌چنین a هم می‌تواند به c نزدیک (قبل از این‌که c ببیند و حذف کند) ولی مشکل آن این است که سربار header زیاد می‌شود.

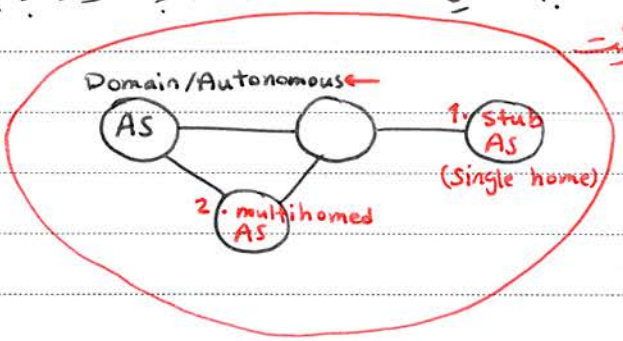
روی هر بسته یک شماره قرار داده می‌شود و عنوان معرفی کننده بسته است. به همین دلیل اگر کسی بسته‌ای را که قبلاً گرفته باشد دوباره دریافت کند، می‌فهمد و بسته را دور می‌ریزد. (با استفاده از شماره بسته و آدرس مبدأ):



این جدول‌ها وقتی مطمئن شدیم بسته از بین رفته است آن‌ها را پاک می‌کنیم.
(بازگشت زمان)
لازمه‌اش آن است که یک حافظه در هر گره داشته باشیم.

روش اول هزینه communication بالاست.
روش دوم هزینه حافظه بالاست.

در شبکه‌های اینترنت باید جداول را به روز کنیم. یک سری پروتکل مسیریابی تعریف می‌کنیم که توسط آن‌ها جدول مسیریابی ایجاد می‌شود. خیلی بزرگ است و از میلیون‌ها شبکه تشکیل شده است و اگر برای هر گره بخواهیم مسیریابی کنیم خیلی پیچیده می‌شود.



← کامپیوترهایی که تحت مدیریت یک مرکز هستند، Domain نامیده می‌شوند.

1. AS ای که برای اتصال به سایر AS های شبکه‌ها تنها یک لینک دارد. داخلی (خود AS)، خارجی بر نسبت به همان یک لینک (مسیریابی ساده)

2. بیش از یک لینک برای اتصال به سایر شبکه دارد. از لینک‌های چندگانه‌اش فقط برای ارسال و دریافت ترافیک خودش استفاده می‌کند. یعنی مبدأ و مقصد ترافیک خودش است و ترافیک را transit نمی‌کند.

3. Transit AS : ترافیک را از خودش عبور می‌دهد.

* هر AS را با یک شماره 16 بیتی به نام AS No. معرفی می‌کنند. AS های stub اکثراً این شماره را ندارند. AS No. آن‌ها مال provider شان است.

مسیریابی در اینترنت { داخل AS ها / بین AS ها

Interior Gateway Protocol: IGP داخل AS ها

Enterior " " : EGP بین AS ها

مثلاً یک بسته ای را از دانشگاه امیرکبیر به دانشگاه تهران می فرستیم. بهترین مسیر برای رساندن بسته به gateway با استفاده از IGP ها پیدا می شود و سپس رساندن آن تا AS دانشگاه تهران با استفاده از EGP صورت می گیرد.

Routing Information Protocol RIP
Open Shortest Path First OSPF
} IGP

Border Gateway Protocol BGP (EGP) → بین borderها (گره های خارجی)

* تحت تاثیر سیاستهای ترافیکی و... AS هاست.

gateway ها در جدول دارند { داخل AS — بغیر از Stub ها!
خارج AS

RIP: از روش distance Vector استفاده می کند. هزینه هر لینک را 1 در نظر می گیرد. هر گره هر چه تعداد hop count کمتری داشته باشد هزینه اش کمتر است. فرض می کند که شبکه ای پیاده سازی می شود که طولانی ترین مسیرش بیشتر از 15 گام نمی شود. 16 را به عنوان ∞ فرض می کند، یعنی اگر مسیری پیدا نکند، cost را 16 می گذارد.

← RIP یک پیغام به عنوان RIP message درست می کند، از طریق UDP Port 520 ارسال می شود. این پیغام ها آن اطلاعات جدول مسیریابی است. اگر یک گره نرسیده بداند RIP message ای از گره همسایه نگرفت، یعنی ارتباطش قطع شده است.

Format RIP Message

Command	Version	zero
Address Family Identifier		zero
IP Address		
zero		
zero		
metric		
⋮		

برای هر گره تکرار می شود

Command: مشخص می کند چه نوع پیغامی است:

1: Request وقتی یک گره به همسایه اش می دهد، برای آدرسی که می خواهد هزینه اش چقدر است.

2: Response: جواب می دهد به request یا وقتی که جدولش update می شود و می خواهد به گره همسایه اطلاع دهد.

Address Family Identifier: در شبکه ای استفاده می کنیم از RIP. در شبکه اینترنت عدد 2 می گذاریم.

IP Address: آدرس گره ها

Metric: هزینه رسیدن به مقصد از طریق گره ای که اطلاع می دهد. همان hop count است که حداکثر 15 است.

OSPF:

از روش link state استفاده می‌کند برای انجام عملیات مسیریابی. هر روتر داخل خود یک link state DB دارد که وضعیت همه لینک‌های شبکه در آن قرار دارد. وقتی می‌گوییم لینک‌های شبکه منظورمان لینک‌های داخل یک AS است چون OSPF یک پروتکل درون AS ای است.

مزیت های OSPF نسبت به RIP :

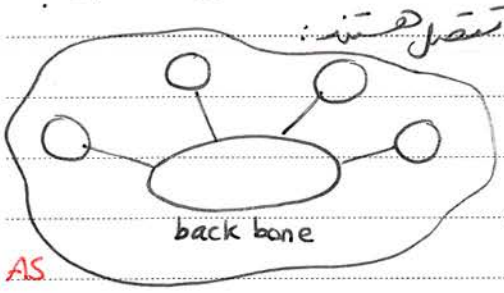
1. در RIP برای هر مقصد یک مسیر پیدا می‌شد ، OSPF چند مسیر را همزمان در جدولش نگاه می‌دارد و ترافیک را بین چند مسیر تقسیم کند (load balancing) تا بهترین مسیر پیدا شود.

وضعیت لینک : با یک عدد 16 بیتی نگه داری می‌شود : link cost: 1 – 65535
این که این cost برای چه معیاری تعیین می‌شود در اختیار مدیر شبکه است که چگونه به لینک‌هایش هزینه می‌دهد مثلاً می‌تواند hop count را به عنوان هزینه انتخاب کند یا معیارهای دیگر.

2. flexability نسبت به RIP بیشتر است زیرا در RIP ، hop count به عنوان معیار مسیریابی بود.

function های OSPF برای کمتر کردن حجم داده ها :

- اگر روی یک شبکه‌ی multi access چند router وجود داشته باشد ، یکی از router ها را به عنوان نماینده multis access انتخاب می‌کند و سنکرون سازی DB روی آن انجام می‌شود.
- اگر AS بزرگی داشته باشیم ، OSPF این AS را ناحیه بندی می‌کند. این امکان را می‌دهد که مدیر شبکه ناحیه بندی کند و یک ناحیه backbone به وجود بیاید و بقیه‌ی نواحی به آن متصل هستند.



به این شکل دیگر لازم نیست وضعیت لینک های هر ناحیه در ناحیه های دیگر گزارش شود. بدین ترتیب forwarding محدود می‌شود. هر router ای که در هر ناحیه است فقط وضعیت لینک های آن ناحیه را دریافت می‌کند.

* هر ناحیه با یک ID معرفی می‌شود در OSPF که یک عدد 32 بیتی است. backbone ID همیشه صفر است : 0.0.0.0 (چهار عدد 8 بیتی)
این ناحیه بندی optional است و در صورتی که شبکه بزرگ نباشد و ما نخواهیم حجم پیغام های مسیریابی را کنترل کنیم می‌توانیم این کار را انجام دهیم.

backbone : یعنی ستون فقرات. وقتی شبکه را طراحی می‌کنید یک core دارد ، از آن طریق همه شبکه به هم متصل می‌شود. به آن هسته ، backbone می‌گوییم.
مجموعه ای از router ها هستند که بقیه شبکه را به هم متصل می‌کند. اگر بتوانیم در شبکه backbone پیدا کنیم ، آن گاه می‌توانیم از ویژگی ناحیه بندی OSPF استفاده کنیم.

* یکی از field های header پیغام های OSPF ، Area ID است . هر router ای که این پیغام را می گیرد ، درصورتی که ID ناحیه اش با پیغام یکسان بود سنکرون سازی را انجام می دهد .

* در شبکه اینترنت یک topology کاملاً نامنظم داریم که هر AS ای می تواند به هر AS ای وصل باشد و نمی توانیم یک backbone پیدا کنیم .

از طریق TCP یا UDP

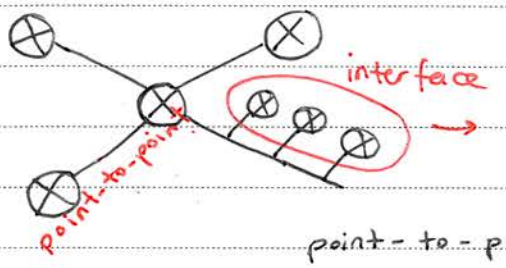
OSPF پیغام هایش را مستقیماً از طریق بسته های IP ارسال می کند . پروتکل ID ای که برای بسته های OSPF در نظر گرفته شده است ۸۹ است (در header بسته های IP ، protocol ID)

مراحل کاری OSPF :

1. OSPF router ابتدا می گردد و همایه هایش را با استفاده از پیغام های خوش آمدگویی پیدا می کند . یعنی به طور دوره ای hello packet می فرستد . درتمام لینک هایش هر router این پیغام را می فرستد . router های که به آن وصل هستند این hello packet را می گیرند و پاسخ می دهند (با یک پیغام hello packet) بدین ترتیب همایه ها شناسایی می شوند .

2. از بین همایه هایی که شناسایی کرده است ارتباط برقرار کند و وضعیت لینک های DB هرکدام را که لازم است با DB خود سنک کند .

- اگر لینک point-to-point باشد ، باید با همه ی routerها تنظیم شود .
- درصورتی که لینک multi access باشد ، یکی از آن ها را به عنوان نماینده شبکه انتخاب می کنیم . دقتی hello packet ها فرستاده شد ، در پاسخ در بسته های hello یک field اولویت وجود دارد که مدیر شبکه به router ها اختصاص داده است . پس router ای به عنوان نماینده انتخاب می شود که اولویت بالاتری دارد . یک backup router را هم انتخاب می کند که درصورتی که نماینده اتفاقی برایش افتاد ، از آن استفاده کند .



چون interface ها هم یکسان است یکی را انتخاب می کنیم .

درواقع هر لینک باید با یکی تنظیم شود :

یک router بیشتر وجود ندارد پس با آن تنظیم می شود : point-to-point

نماینده : multi-access

3. تغییرات وضعیت لینک ها advertise می‌شود. هر گره‌ای یک تغییر را sense کند برای گره‌های همسایه می‌فرستد، Flooding صورت می‌گیرد (هر گره‌ای که update شد برای تمام گره‌های فرستد.)

پیام‌های OSPF :

1. hello packet پیام خوشامدگویی
2. link state description هاDB سنکرون سازی
3. link state request
4. link state update update کردن
5. link state Ack. update تایید

common header

این پنج پیام یک قسمت header‌شان مشترک و بقیه‌اش تفاوت است. شروع پیام‌ها شبیه هم است: یک field به اسم type وجود دارد که نوع پیام OSPF را مشخص می‌کند.

* یکی از ویژگی‌های مثبت OSPF این است که از security پشتیبانی می‌کند. یعنی هر router‌ای که هر پیغامی می‌فرستد معتبر نیست بلکه باید authenticate بشود (تصدیق هویت). یک مکانیزم authentication در header مشترک وجود دارد که تشخیص می‌دهد این router‌ای که این پیغام را فرستاده است router معتبری است یا نه. در شبکه تصدیق هویت با یک ID انجام می‌شود. برای این که بتوان ID را جعل کرد یک رمزنگاری انجام می‌دهیم. فقط گره معتبر می‌تواند رمز را شناسایی کند و ID را تشخیص دهد. در OSPF header یک field برای authentication وجود دارد که داخل آن یک ID رمز شده قرار می‌گیرد. روش رمزنگاری در آن مشخص می‌شود. کلید رمزنگاری هم به routerها داده می‌شود:

رمزنگاری متقارن: یک کلید برای رمز کردن داریم و همان برای تشخیص رمز استفاده می‌کنیم. اشکال این روش این است که اگر کلید لو برود، می‌توان رمز را باز کرد.

رمزنگاری نامتقارن: در این روش یک کلید برای رمز کردن داریم که در اختیار همه گره است. یک کلید هم برای باز کردن رمز داریم که در اختیار کمتر گره‌هاست. اگر هر کدام از این کلیدها لو برود اشکالی ندارد چون کلید دیگر دست فرد معتبر است.

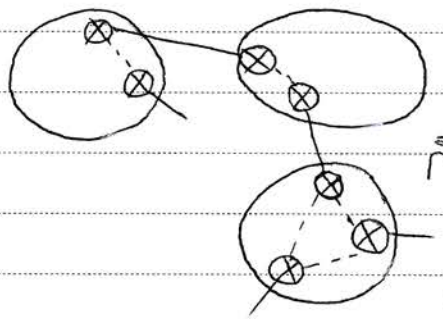
← مشکل RIP این است که تصدیق هویت ندارد. مثلاً یک گره مخرب با ارسال پیام‌های مسیریابی اشتباه باعث ایجاد مشکل در شبکه شود.

مسیریابی داخلی هر AS از پروتکل‌های IGP استفاده می‌کند. از یک دید بالاتر اگر هر AS را یک node ببینیم مسئولیت مسیریابی داخلی به عهده خود AS است. مدیر شبکه انتخاب می‌کند: OSPF، RIP و ...

پس اینترنت یک شبکه است که از AS ها تشکیل شده است. برای مسیریابی بین domain ها از پروتکل‌های EGP استفاده می‌کنیم که معروف‌ترین آن BGP است:

BGP (Border Gateway Protocol):

پروتکل BGP شبیه پروتکل‌های link state است و وضعیت link ها را به AS های مجاور خودش گزارش می‌کند و به همین ترتیب. هر دو AS از طریق یک BG به هم وصل می‌شوند. هر router نیز برای یک شبکه به router نیز برای شبکه‌ی دیگری وصل می‌شود. یک شبکه ممکن است چند BG (Border Gateway) داشته باشد. وضعیت AS های اطراف و اتصال آن‌ها در BG ها باید ذخیره شود و router های میانی نقشی ندارند. این پروتکل مسیریابی بین domain ها فقط روی Border Gateway ها اجرا می‌شود. BG ها با هم یک connection برقرار می‌کنند و وضعیت DB خود را با هم تنظیم می‌کنند.



دو BG که داخل یک AS هستند یا داخل دو AS متفاوت هستند باید وضعیت DB شان یکسان باشد.

BGP Speaker: اصطلاحاً به BG ها می‌گویند.

دو ارتباط بین BG ها وجود دارد:

eBGP (external BGP): دو BG ای که داخل دو AS مختلف هستند.

iBGP (internal BGP): " " " " یک AS هستند.

BGP پیغام‌هایش را از طریق پروتکل TCP ارسال می‌کند. پس قبل از این که پیغام‌های BGP رد و بدل شود بین BG ها باید ارتباط TCP برقرار شود و سپس پیام‌های BG ها رد و بدل می‌شوند. وقتی یک ارتباط TCP بین دو gateway غیرمجاور برقرار می‌شود دیگر از نظر ما مجاور هستند. تفاوت ارتباط بین دو گره مجاور با دو گره غیرمجاور این است که تغییراتی که مجاز است از یک گره‌ی غیرمجاور به گره‌ی مجاور دیگر اطلاع داده می‌شود. طبق یک سیاستی تغییراتی را که مجاز به گزارش آن است اطلاع می‌دهد. مثلاً یک AS نمی‌خواهد که ترافیک داخلی‌اش را به یک AS دیگر گزارش دهد. مدیر شبکه این سیاست‌ها را پیکربندی می‌کند.

* داخل AS همه چیز گزارش می‌شود.

سیاست‌ها:

1. never use ASx to get to a destination in ASy.
2. never use ASx and ASy in the same path.

این سیاست ها به صورت یک سری rule برای BGP درمی آید و به BGP گفته می‌شود که باید این rule ها را رعایت کند و BGP این rule ها را در الگوریتم‌های مسیریابی اش درنظر می‌گیرد.

* BGP ها ارتباطشان را روی TCP روی port number 179 برقرار می‌کنند.
ابتدا TCP Connection برقرار می‌کنند و سپس با پیغام open ارتباط BGP ها با هم برقرار می‌شود. سپس با پیغام update تغییراتی که وجود دارد به هم گزارش می‌شود. سپس پیغام notification خطاهایی که وجود دارد در BGP connection را اطلاع می‌دهد. یک پیغام keep alive هم داریم که شبیه hello packet است به صورت دوره‌ای هر ۳۰ ثانیه یکبار برای این که زنده بودن خود را اعلام کند این پیغام را می‌فرستد اگر پس از مدتی یک BG این پیغام را دریافت نکرد می‌فهمد که BG مقابل از کار افتاده است. ممکن است ارتباط TCP برقرار باشد ولی در BGP مشکلی به وجود آمده باشد.

ارسال open ، ارسال update و ...

- ارتباط IP یا ارتباط TCP قطع شده
- " " BGP "
- BG از کار افتاده است

پیغام OPEN: از اطلاعاتی که در این پیغام می‌آید AS number است. می‌گوید من BGP Speaker ای هستم که از این AS No. ام صحبت می‌کنم. زمان ارسال پیغام keep alive را می‌دهد. (مثلاً 30 ثانیه یکبار) IP Address و ID خودش را هم می‌دهد. Authentication هم انجام می‌شود. شناسایی فردی که با آن صحبت می‌کنیم با پیغام OPEN صورت می‌گیرد.

* گره‌های میانی تا لایه‌ی اینترنت برای انتقال داده‌ی user را دارند. این اطلاعاتی که درباره‌شان صحبت کردیم داده‌های مسیریابی است و برای همین از TCP استفاده می‌شود. برای انتقال داده‌های کنترلی router ها می‌توانند از لایه‌های بالاتر از اینترنت هم استفاده کنند.
یعنی router، application مسیریابی دارد که وظیفه آن انتقال داده‌های مسیریابی شبکه است.

Network Interface:

- Multi Access: چندگره به طور مشترک از آن استفاده می‌کنند. برحاهایی که لینک مشترک داریم، اگر بیش از یک ایستگاه همزمان ارسال کند تداخل ایجاد می‌شود. در کانال‌هایی که بین چند ایستگاه مشترکند، در هر لحظه یک کانال می‌تواند ارسال کند. یک مکانیزم برای کنترل باید وجود داشته باشد که به آن medium access control (MAC) می‌گویند.
- Point-to-Point: احتیاجی به مکانیزم کنترل دسترسی ندارد زیرا در یک لحظه فقط یک ارسال اتفاق می‌افتد.

کارایی پروتکل MAC : وابسته به میزان delay bandwidth product است. هرچه این پارامتر بیشتر باشد کارایی MAC کاهش پیدا می‌کند. این پارامتر وقتی زیاد می‌شود که یا تأخیر انتشار بالاست (فاصله گره‌ها از هم زیاد است) یا سرعت لینک پایین است. وقتی کارایی MAC پایین می‌آید دیگر استفاده از multi access ها به صرفه نیست. به همین دلیل لینک‌های multi access در شبکه‌های کوچک یا محلی (LAN) ها استفاده می‌شود.