# Multiple Access Protocols(MAC)

- بعضی از لینک ها point-to-point هستن، یعنی از این لینک ها تنها دو نود استفاده می کنن ، که یکی فرستنده و دیگری گیرنده هست. مثل لینک هایی که بین host ها و سوئیچ های Ethernet وجود داره. یا لینک هایی که بین روتر ها در داخل شبکه وجود داره.
- لینک های دیگه ای وجود دارن به اسم لینک های broadcast (یا shared ) که چند نود(که تعداد اون ها می تونه زیاد هم باشه) همزمان از این لینک ها می تونن استفاده کنن و frame های خودشون رو از طریق این لینک ها ارسال کنن. مثل لینک های Ethernet کابلی(Cabled Ethernet ) ، مخابرات سلولار(4G/5G ) ، WiFi ، و مخابرات ماهواره ای (satellite) .
- Cabled Ethernet نسخه ی قدیمی Ethernet هست و یک کابل کواکس وجود داره که همه ی host ها به این یک کابل متصل می شدن و برای ارسال و دریافت frame هاشون از این کابل استفاده می کردن:

استفاده از این تکنولوژی در حال حاضر منسوخ شده و Ethernet ای که ما در حال حاضر ازش استفاده می کنیم، لینک هایی که داخل شبکه های Ethernet هست از نوع point-to-point ان و بین هر host و یک سوئیچ در شبکه ی LAN ، لینک مجزا داریم و یک ساختار ستاره ای شکل داره.

- در لینک های broadcast دیگه ای که نام بردیم (غیر از نسخه ی قدیمی Ethernet ) جنس لینک ها، همه رادیویی هست و این لینک ها معمولا بین چندتا نود به صورت مشترک مورد استفاده قرار می گیرن و مسئله ی MAC هم که قراره راجع بهش صحبت کنیم به طور خاص در مورد لینک های رادیویی مطرحه.

- Multiple Access Protocols

- در این پروتکل ها ، یک لینک shared (یا broadcast ) داریم و این امکان وجود داره که همزمان بیشتر از یک نود از این لینک برای ارسال و دریافت frame هاشون استفاده کنن. اتفاقی که ممکنه بیفته در این شرایط، اینه که frame ها بر روی هم تداخل ایجاد کنن و تصادف (collision)بین frame ها رخ بده.
- تعریف collision : یک گیرنده به جای این که یک frame از کانال مشترک دریافت کنه ، مخلوطی از چندتا (بیش از یک) frame دریافت کنه، چون بیش از یک نود در آن واحد اقدام به ارسال frame کردن.

پیش فرض هم این هست که اگه یه نود ، مخلوطی از چند تا frame رو دریافت کنه، نمی تونه هیچکدوم رو به طرز صحیحی دریافت کنه و به عبارتی وقتی collision رخ میده، ظرفیت لینک از بین میره ؛ به خاطر همین دوست داریم تا جایی که میشه collision رخ نده.

- صورت مسئله ی MAC :

ما دوست داریم یک الگوریتم distributed داشته باشیم که این الگوریتم توسط نود هایی که میخوان به صورت مشترک از لینک broadcast استفاده کنن، اجرا بشه.به نحوی که تا جایی که میشه تصادف رخ نده و در واقع توسط این الگوریتم نود ها بفهمن که چه موقع از کانال استفاده کنن که بهره وری کانال بیشینه بشه و احتمال collision به حداقل برسه.

چالش این مسئله هم از این نشأت می گیره که ما یک کانال دیگه برای هماهنگی نود ها نداریم، و این الگوریتم ها که در داخل نود ها اجرا میشن، کاملا باید به صورت distributed باشن.

- ورودی الگوریتم های MAC اینه که یه لینک مشترک داریم که ظرفیت اون لینک R بیت بر ثانیه هست. حالا ما میخوایم الگوریتم هامون که توزیع شده ست و در حالت ایده آل هیچ ارتباطی بینشون وجود نداره باید این کار ها رو انجام بدن :

1 - اگر تنها یک نود می خواد از اون کانال مشترک استفاده کنه ، توی حالت ایده ال باید اجازه بدیم تمام ظرفیت کانال در اختیار اون نود

قرار بگیره و اون نود بتونه با R rate بیت بر ثانیه frame های خودش رو ارسال کنه.

2 - وقتی M نود میخوان از لینک مشترک استفاده کنن، حالت ایده آل اینه که کانال به صورت منصفانه بین اون ها تقسیم بشه و هرکدوم از اون ها بتونن با R/M rate بیت بر ثانیه frame های خودشون رو ارسال کنن.

3 - دوست داریم الگوریتم ها کاملا decentralized باشن ، یا به عبارتی یک نودی با وظایف خاص در سیستم ما وجود نداشته باشه که هماهنگی بین نود ها انجام بده و دوست نداریم که سیستمون تنها متکی به یک نود باشه ، و اگر اون نود براش مشکلی پیدا شد سیستم از کار بیفته.

همچنین نباید یک کلاک مشترک و یا روش های دیگه ای برای synchronization در حالت ایده آل داشته باشیم.چون مجددا همون مشکلی که گفتیم ممکنه برای synchronization پیش بیاد و اگه دچار مشکل بشه کل سیستم کاراییش رو از دست میده.

4 - پروتکل Multiple Access ایده آل باید ساده باشه و پیچیدگی های محاسباتی زیاد نداشته باشه و ما بتونیم به راحتی در نود ها اجراش کنیم.

- MAC protocols : taxonomy

- به صورت کلی پروتکل هایی که تا حالا ارائه شدن رو می تونیم به سه دسته تقسیم کنیم:

1 - Channel partitioning : مثل متد های TDM و FDM که قبلا باهاشون آشنا شدیم.

2 - Random access : توی این روش به هر نودی که میخواد frame ارسال کنه ، تمام ظرفیت لینک رو اختصاص میدیم. البته توی این روش احتمال collision بالاست چون ممکنه چندتا نود از کانال بخوان استفاده کنن . پس توی این پروتکل ها ، باید روش هایی برای نجات از این وضعیت ها داشته باشیم.

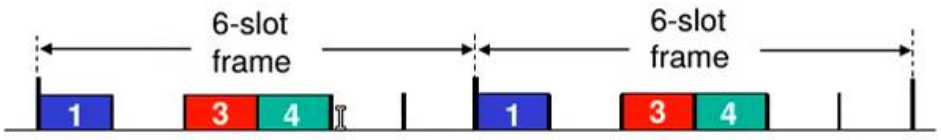
3 - Taking turns : این روش بین دو روش اول قرار می گیره و از مزایای هر دوی این روش ها بهره می بره. به این شکل که نود ها به نوبت اگه دیتایی برای ارسال داشته باشن ، ارسال می کنن. تفاوتش با روش اول اینه که اگه یه نودی نوبتش شد و دیتایی برای ارسال نداشت، کانال بیکار باقی نمی مونه و بلافاصله سعی می کنه نوبت رو در اختیار نود دیگه ای قرار بده.

- Channel partitioning MAC protocols : TDMA

- ایده ی اصلی روش های channel partitioning اینه که ظرفیت کانال در حوزه ی زمان یا فرکانس تقسیم میشه و هر قسمت یا

partition به یک نود تخصیص داده میشه. به این ترتیب دسترسی به کانال کاملا ضابطه مند میشه .

مثلا در روش TDMA که میایم کانال رو در حوزه ی زمان تقسیم می کنیم، دسترسی به کانال در قالب تعدادی round انجام میشه. در هر round ، یک slot زمانی مشخص(fixed) رو به یک نود (station) اختصاص میدیم و هر بار در هر round اون نود می تونه در slot زمانی ای که بهش اختصاص داده شده (و در صورتی که frame ای برای ارسال داشته باشه) از کانال استفاده کنه. مثلا توی شکل زیر ، هر round ما تقسیم شده به ۶ تا time slot (چون ۶ تا نود داریم که میخوان به صورت مشترک از کانال استفاده کنن) . slot اول رو اختصاص میدیم به station 1 و slot دوم رو به station 2 و ... .



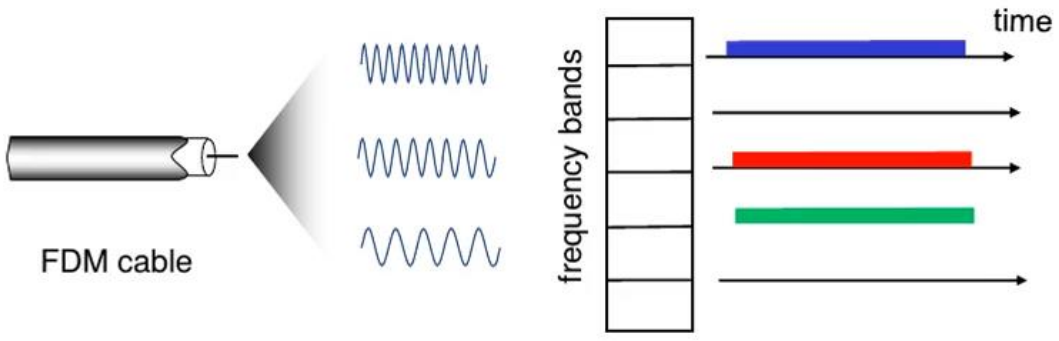

- توی همه ی روش های channel partitioning احتمال collision صفره و این ، یکی از مزیت های این روش ها هست. اما از طرف دیگه ، مشکلی که این روش داره اینه که وقتی ما میایم در هر round ای یک time slot رو به یه station اختصاص می دیم، در صورتی که اون station داده ای برای ارسال نداشته باشه ، از time slot خودش

استفاده نمی کنه و چون به اون نود اختصاص داده شده ، سایر نود ها حتی اگه داده ای برای ارسال داشته باشن ، نمی تونن ازش استفاده کنن. در نتیجه ظرفیت کانال هدر میره. مثلا توی همین شکل بالا ، فرض شده که نود های 2 و 5 و 6 داده ای برای ارسال ندارن ، و در مجموع داریم از نصف ظرفیت کانال استفاده می کنیم.

- Channel partitioning MAC protocols : FDMA

- توی این روش ، partitioning کانال در حوزه ی فرکانس انجام میشه. ما پهنای باند کانال رو در قالب تعدادی باند فرکانسی تقسیم می کنیم ، و هر باند فرکانسی رو در اختیار یه station قرار می دیم.
- مجددا احتمال collision صفر میشه اما باز اگه یه station داده ای برای ارسال نداشته باشه ، از اون باندی که بهش اختصاص دادیم استفاده نمی کنه و بقیه ی نود ها هم نمی تونن استفاده کنن، پس پهنای باند کانال هدر میره.

مثال :



- Random Access Protocols

- نقطه ی مقابل روش های channel partitioning هستن.
- در این روش ها، ظرفیت کانال ها تقسیم بندی نمیشه ، و وقتی یک نود می خواد frame خودش رو ارسال کنه ، در صورتی که بخواد از کانال بخواد استفاده کنه ، کل ظرفیت کانال رو در اختیار می گیره و تمام frame خودش رو ارسال می کنه .این کار بدون هماهنگ قبلی انجام میشه ، بنابراین احتمال collision وجود داره. پس این پروتکل های random access باید مشخص کنن :
  1 - که چطور وقوع collision رو متوجه بشیم.
  2 - در صورت وقوع collision باید چه راهکاری داشته باشیم تا از وقوع مجددش جلوگیری بشه.

- مثال هایی از پروتکل هایی که در لایه ی لینک استفاده میشن و بر مبنای random access کار می کنن :

ALOHA , slotted ALOHA
CSMA, CSMA/CD , CSMA/CA

1 - Slotted ALOHA : از لحاظ تاریخی بعد از پروتکل ALOHA معرفی شد ، اما چون توصیف و بررسی عملکرد این پروتکل نسبت به ALOHA ساده تره، اول این پروتکل رو بررسی می کنیم.

در این پروتکل ، همه ی frame ها از لحاظ زمانی اندازه یکسان دارن. از طرف دیگه محور زمانی به قطعاتی با طول مشخص به نام slot تقسیم شدن. زمان هر slot ، همون زمانی هست که ما احتیاج داریم که یک frame رو ارسال کنیم.(به عبارتی طول زمان یک slot با طول زمان یک frame یکسانه)

در این پروتکل فرض میشه که وقتی نود ها میخوان frame هاشون رو ارسال کنن ،زمان شروع ارسال frame با زمان شروع یک slot مطابقت کنه ؛ یعنی ابتدا و انتهای frame کاملا با ابتدا و انتهای یه slot ، match میشه.

طبق این توضیحات، نود ها باید synchronized باشن ، برای این که محور زمانی یکسانی داشته باشن و درک و فهم یکسانی از زمان های ابتدا و انتهای slot ها داشته باشن تا زمان ارسال frame ها شون رو با زمان شروع یک slot منطبق کنن.

به خاطر همین اگه ۲ یا چند تا نود بخوان به طور همزمان از یه slot استفاده کنن، collision رخ میده و فرض میشه که همه ی نود ها در صورت بروز collision متوجهش میشن .

حالا اگه یک نود ، یه frame جدید از لایه های بالا به دستش برسه ، میاد در اولین time slot ای که می تونه ، frame رو ارسال می کنه . دو حالت به وجود میاد : ۱- collision رخ نمیده و فرض رو بر این می ذاریم که frame مون به طرز صحیح به دست گیرنده می رسه. ۲- collision رخ میده و فرض کردیم که نود ها میهفمن رخ داده؛ پس مجددا frame شون رو ارسال می کنن.

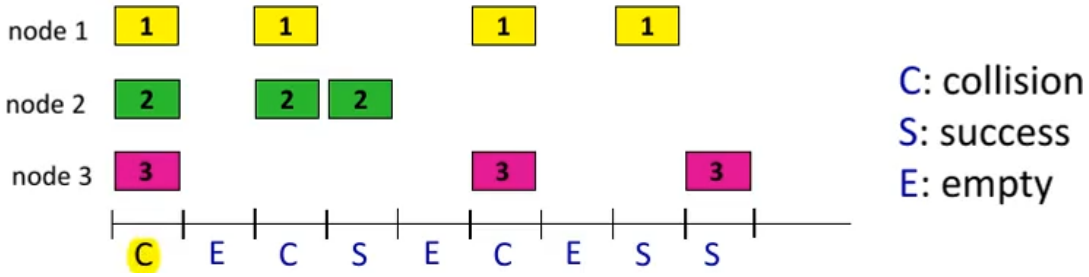
نکته ی کلیدی توی پروتکل slotted ALOHA اینه که اگه یه نودی اقدام به ارسال frame ش کنه و متوجه بشه collision رخ داده، در هر یک از time slot های بعدی، با احتمال p ممکنه که مبادرت به ارسال مجدد اون frame بکنه و این کارو ادامه میده تا بالاخره توی یکی از ارسال هایی که انجام میده، collision رخ نده.

دلیل این که ما میایم از یه روش تصادفی استفاده می کنیم، اینه که نود هایی که همزمان توی یک slot ، frame خودشون رو ارسال کردن، وقتی به صورت تصادفی تصمیم می گیرن که توی time slot های بعدی ارسال مجددشون رو انجام بدن، احتمال این که دوباره همزمان از یک slot استفاده کنن کم میشه و در نتیجه احتمال collision کم میشه.

مثال :

فرض شده که سه تا نود داریم ، و هر کدوم یک frame برای ارسال دارن ، و همزمان این frame ها رو از لایه های بالایی دریافت کردن

تا ارسال کنن ، به خاطر همین طبق پروتکل slotted ALOHA همه، در slot بعدیشون که C هست مبادرت به ارسال frame ها کردن.




طبیعتا توی اسلات C هیچ کدوم از این frame ها به طرز صحیحی به گیرنده هاشون نمی رسن و collision رخ میده.

بعد هر کدوم از نودها متوجه میشن که collision رخ داده، و با یک احتمال مشخصی توی time slot های بعدی و به صورت مجزا ، مبادرت به ارسال frame خودشون بکنن تا نهایتا frame ها بدون collision ارسال بشن.

مثلا توی این مثال فرض شده که نود ها به طور تصادفی هیچ کدوم در اسلات E ، frame ارسال نمی کنن و این time slot از بین میره و ما از ظرفیت کانال استفاده نکردیم.

توی اسلات C ، نود های 1 و 2 تصمیم می گیرن که frame های خودشون رو ارسال کنن و باز collision رخ میده و هیچ کدوم از این frame ها به طرز صحیح به گیرنده ها نمی رسن.

توی اسلات S، فقط نود 2 مبادرت به ارسال frame ش می کنه و collision رخ نمیده و frame موفقیت آمیز ارسال میشه. کار نود 2 اینجا به پایان می رسه.

توی اسلات بعدی که E هست، هیچ کدوم از نود های 1 و 3 ، frame هاشون رو ارسال نمی کنن.

توی اسلات بعدی که C هست، هردو نود 1 و 3 ، frame هاشون رو ارسال می کنن و باز collision رخ میده و ظرفیت کانال از بین میره.

اسلات بعدی هم E هست و به دلیل بیکاری کانال ، ظرفیت کانال از بین رفته.

در دو اسلات بعدی (که S هستن) به ترتیب نود های 1 و 3 موفق میشن frame هاشون رو ارسال کنن.

مزایای slotted ALOHA :

1 - اگه فقط یک نود active داشته باشیم، از تمام ظرفیت کانال می تونه استفاده کنه .

2 - در خوبی این پروتکل decentralized هست به این مفهوم که ما یک نود با وظایف مشخص و ممتازی نسبت به سایر نود ها نداریم. اما این قضیه کامل نیست ( fully decentralized نیست) و ما احتیاج داریم نود هامون از لحاظ زمانی sync باشن.

3 - ساده ست، کافیه یه مکانیزمی برای تولید اعداد تصادفی داشته باشیم تا اگه collision ای رخ داد، مبتنی بر اون random number generator تصمیم بگیریم آیا باید از اسلات بعدی استفاده کنیم یا نه.

معایب slotted ALOHA :

1 - به دو دلیل ظرفیت کانال هدر میره : یکی collision و یکی استفاده نکردن از برخی از اسلات ها.

2 - ممکنه نود ها قبل از این که مدت زمان یه اسلات به پایان برسه، متوجه collision بشن و می دونن که نباید از اون اسلات استفاده کنن، اما طبق این پروتکل ، استفاده از کانال تا زمان شروع تایم اسلات بعدی به تعویق میفته . بنابراین نود ها نمی تونن از همون موقع که متوجه collision میشن نمی تونن از کانال استفاده کنن و در زمان صرفه جویی کنن.

3 - باید یه مکانیزمی وجود داشته باشه که نود ها در محور زمان sync بشن . ایجاد این clock synchronization از معایب این پروتکله.

- Slotted ALOHA : efficiency :

میزان کارایی این پروتکل برای استفاده از کانال به راحتی قابل محاسبه هست.

Efficiency : به طور متوسط چه درصدی از اسلات ها به طور موفقیت آمیزی ازشون استفاده میشه.

نحوه ی محاسبه : ( فرض می کنیم تعداد نود هایی که میخوان از کانال استفاده کنن زیاده و همه ی اون ها frame برای ارسال دارن)

تعداد نود ها N هست و پارامتر تصادفی رو هم با p نمایش می دیم. در این صورت اگه یه نود به خصوص در یه اسلات رو در نظر بگیریم، احتمال این که اون نود به خصوص در اون تایم اسلات موفق بشه که frame خودش رو ارسال کنه از این رابطه به دست میاد:

$p(1-p)^{N-1}$

حالا اگه در نظر بگیریم هر نودی که توی اون اسلات موفق به ارسال frame هاش میشه ، باید احتمال قبلی رو در N ضرب کنیم :

$Np(1-p)^{N-1}$

در گام بعدی ، پارامتر p بهینه ای رو به دست میاریم که عبارتی که تابعی از p هست رو ماکزیمم کنه.(چون می خوایم ببینیم به ازای چه مقدار از p ، حداکثر efficiency رو به دست میاریم). پس به جای p توی رابطه ی بالا ، مشتق رابطه رو برابر با صفر قرار میدیم تا ببینم چه مقداری از p مشتق رابطه رو صفر می کنه و اون مقدار از p رو برابر با

p* قرار می دیم و توی رابطه می ذاریم(p* خودش تابعی از N میشه) و کل این رابطه هم تابعی از N میشه. حالا چون گفتیم می خوایم efficiency رو به ازای تعداد نود های زیادی به دست بیاریم، میایم N رو به سمت بی نهایت میل می دیم و در نهایت می بینیم که حد اون رابطه میل می کنه به سمت 1/e :

Max efficiency = 1/e = 0.37

این قضیه نشون میده که این پروتکل زیاد بهینه نیست و مثلا اگه ظرفیت یه لینکی ، 1 Mbps باشه ، به طور میانگین 370 Kbps از اون نود داریم استفاده می کنیم.

همچنین وقتی داریم یک لینکی رو طراحی می کنیم باید ظرفیت لینک مون رو جوری قرار بدیم که با توجه به این efficiency ای که با استفاده از پروتکل بهش می رسیم، به مشکل بر نخوریم. مثلا اگه فرض کنیم تعداد نود ها ۱۰ تا باشه ، و efficiency پروتکل هم ۳۷٪ در نظر بگیریم، به عنوان ادمین همچین سیستمی باید ظرفیت این لینک رو حدود 2.7 Mbps قرار بدیم، در شرایطی که هر کدوم از نود ها ظرفیت 100 kbps نیاز داشته باشن.(یعنی با وجود این که مجموع ظرفیتی که نود ها احتیاج دارن 1 Mbps هست، اما چون efficiency ، ۳۷ درصده ، باید ظرفیت لینک رو 2.7 Mbps قرار بدیم تا برای نود هایی که قراره از این لینک استفاده کنن مشکلی پیش نیاد)

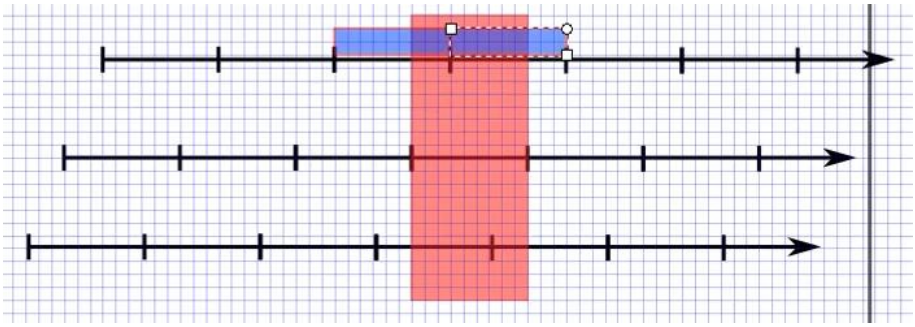
2 - Pure ALOHA :

از لحاظ عملکرد خیلی شبیه به پروتکل slotted ALOHA هست. با این تفاوت که synchronization بین نود ها وجود نداره. به همین دلیل به این پروتکل unslotted ALOHA هم میگن.

از لحاظ پیاده سازی ساده هست و نیازی به یک کلاک مشترک و مکانیزم synchronization سرتاسری بین نود ها نداریم و هر موقع یک نود یک frame رو دریافت کنه طبق محور زمانی خودش می تونه اون رو ارسال کنه.

اما این عدم synchronization توی نود ها باعث میشه که احتمال collision افزایش پیدا کنه و efficiency برابر با 18% (یعنی نصف چیزی که توی slotted ALOHA داشتیم) میشه. برای محاسبه این مقدار دقیقا همون گام هایی که توی slotted ALOHA گفتیم رو طی کنیم ولی یه تفاوت اصلی وجود داره؛ اونم اینه که وقتی میخوایم احتمال استفاده ی موفق از یک تایم اسلات رو محاسبه کنیم، شرایط در این حالت که synchronization بین نود ها نداریم متفاوته.

مثال : فرض کنیم محور زمانی در سه تا نود به شکل زیر باشه. چون synchronization بین نود ها نداریم این نود ها نسبت به هم مقداری شیفت پیدا کردن.



حالا توی این شرایط ، اگه تایم اسلات چهارم توی نود دوم رو در نظر بگیریم و ببینیم احتمال استفاده ی موفق از این نود توی این تایم اسلات چقدره، باید ببینیم نود های دیگه چه مواقعی frame ارسال نمی کنن. این اسلات توی نود دوم ، با دوتا تایم اسلات توی نود اول و همچنین دوتا تایم اسلات توی نود سوم هم پوشانی داره. حالا مثلا اگه نود اول رو در نظر بگیریم، برای این که بتونیم استفاده ی موفق از نود وسط توی این تایم اسلات چهارم داشته باشیم، نود اول نه باید توی اسلات سومش بسته ارسال کرده باشه نه توی اسلات چهارمش. چون اگه توی این اسلات ها frame ارسال کنه با اسلات چهارم نود دوم هم پوشانی داره و collision رخ میده.

بنابراین رابطه ی اولی که برای slotted ALOHA محاسبه کردیم این جا باید تغییر کنه به این رابطه :

$$p(1-p)^{N-1}(1-p)^{N-1} = p(1-p)^{2N-2}$$

اگه سایر گام ها رو هم به طریق مشابه بریم جلو ، یه ضریب 2 توی رابطه ی نهایی به دست میاد :

$$\text{Max efficiency} = 2/e = 0.18$$

همین efficiency پایین در تاریخچه ی پروتکل های multiple access باعث شد به این فکر بیفتن که از synchronization استفاده کنن و نسل بعدی این پروتکل ها ، پروتکل هایی مثل slotted ALOHA باشه که efficiency رو تا دو برابر افزایش بدن.

3 - CSMA (Carrier Sense Multiple Access)

این پروتکل خودش نسخه های متفاوتی داره ؛

1 - Simple CSMA : توی نسخه ی اولیه اش به این شکل کار می کنه که یک نود قبل از این که مبادرت به ارسال یه frame بکنه ، گوش می کنه ، و اون لینکی که قراره ازش استفاده کنه رو اندازه گیری می کنه و می بینه آیا نود دیگه ای از اون لینک استفاده می کنه یا نه ؛ اگه تشخیص داد کانال idle هست و کسی از کانال استفاده نمی کنه، تمام frame خودش رو از طریق کانال ارسال می کنه، و اگه تشخیص داد که کانال busy هست و نود دیگه ای داره از کانال استفاده می کنه ، ارسال frame خودش رو به تعویق میندازه تا زمانی که کانال آزاد بشه.

2 - CSMA/CD ( CSMA with collision detection ) :

توی این پروتکل وقتی یه نودی در حال ارسال frame هست، به طور همزمان لینک رو اندازه گیری می کنه تا ببینه collision رخ

میده یا نه.اگه در مدت زمانی که داره frame خودش رو ارسال می کنه (و همزمان داره لینک رو رصد می کنه) ، به این نتیجه برسه که collision رخ نداده، frame خودش رو ارسال می کنه و فرض رو بر این میذاره که اتفاق بدی نیفتاده . اما اگه احساس کرد که collision رخ داده ، در این صورت در اسرع وقت ارسال frame اش رو متوقف می کنه تا درصد هدر رفت کانال کاهش پیدا کنه.

نکته مهم : استفاده از collision detection در لینک هایی که سیمی هستن به مراتب ساده تر از لینک های wireless هست. دلیل این قضیه اینه که وقتی نود ها قراره از طریق یک سیم با هم ارتباط برقرار کنن، اون سیگنالی که ما ارسال می کنیم تقریبا از لحاظ اندازه برابره با اندازه ی سیگنالی که اگه یه نود دیگه بخواد ارسال کنه ، ما دریافت می کنیم. به همین خاطر وقوع collision رو ما می تونیم متوجه بشیم. اما توی محیط wireless به این شکله که وقتی یک نودی سیگنالی رو ارسال می کنه ، این سیگنال در فضا پخش میشه و می تونیم بگیم درصد توانی که نهایتا توسط آنتن یک نود دیگه دریافت میشه ، به مراتب از توانی که اون نود استفاده کرده تا سیگنال خودش رو ارسال کنه پایین تره.

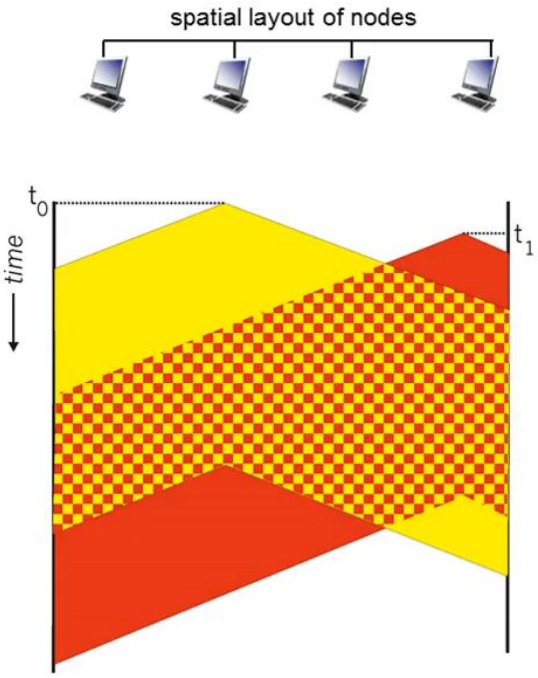
حالا اگه یه نودی بخواد از این مکانیزم CSMA/CD استفاده کنه دچار مشکل میشه . چون با توان خیلی زیادی داره سیگنال ارسال می کنه ، از اون طرف می خواد توان های کوچیکی که توسط نود های دیگه دریافت می کنه رو بشنوه ؛ مثل اینه که یه نفر بخواد داد بزنه و صداهای آهسته ی بقیه رو بشنوه و همچین چیزی ممکن نیست. بنابراین توی سیستم های wireless ما نمی تونیم از پروتکل CSMA/CD استفاده کنیم. توی پروتکل هایی مثل WiFi که در شبکه های wireless LAN استفاده میشه ، از مکانیزم CSMA/CA استفاده می کنیم که CA مخفف collision avoidance هست. یعنی تا جایی که میشه از collision دوری کنیم.

مثال : (در مورد simple CSMA )

توی این مثال می خوایم تاکید کنیم که با وجود این که توی این پروتکل قبل از اینکه نود ها مبادرت به ارسال frame کنن کانال رو بررسی می کنن که نود دیگه ای در حال استفاده از کانال نباشه، اما همچنان ممکنه collision رخ بده. این که ما میایم ابتدا کانال رو sense می کنیم و مطمئن میشیم که نود دیگه در حال استفاده ازش نیست، نسبیه و به طور ۱۰۰ درصد مطمئن نمیشیم که نود دیگه ای داره از کانال استفاده می کنه یا نه. دلیلش هم propagation

delay هست. ممکنه یه نودی از کانال استفاده کنه ولی به دلیل تاخیر انتشار هنوز سیگنالی که روی کانال ارسال کرده به نود دیگه ای که میخواد از کانال استفاده کنه نرسیده.

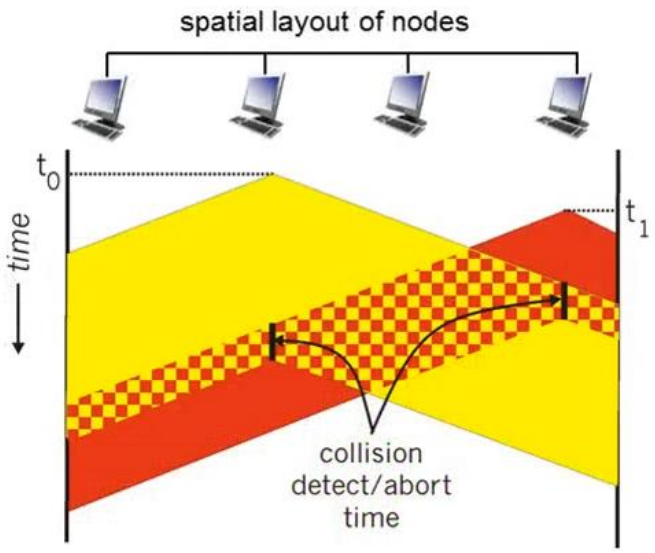
توی این مثال یه شبکه ی Ethernet کلاسیک داریم که با یه کابل کواکس ۴ تا نود رو به هم متصل کردیم . در زمان $t_0$ نود دوم frame ای برای ارسال داشته ، کانال رو sense می کنه و می بینه روی کابل کواکس سیگنال دیگه ای نیست و در زمان $t_0$ مبادرت به ارسال frame اش می کنه. برای این که مشخص بشه propagation delay چه تاثیری می ذاره ، محور زمان رو عمودی در نظر می گیریم و برای این که نشون بدیم طول می کشه تا سیگنال ارسال شده از نود دوم داخل لینک انتشار پیدا کنه، انتشار سیگنال رو با خط های زاویه دار نشون می دیم.




اگه شکل رو کامل کنیم می بینیم collision رخ میده و از یه زمانی به بعد در تمام نقاط لینک مخروط دوتا سیگنال رو داریم.

کل مدت زمان از $t_0$ تا زمانی که ارسال frame به اتمام می رسه، از کانال هدر میره و این قضیه ادامه داره؛ چون نود آخر هم در زمان $t_1$ شروع به ارسال frame اش کرده بود، و تا زمانی که ارسالش به پایان برسه ، طول می کشه تا کانال به حالت clear بر گرده و ما بتونیم از کانال استفاده کنیم.

اما در CSMA/CD ، نود ها همون موقعی که میان سیگنال خودشون رو ارسال می کنن، همزمان وضعیت لینک رو رصد می کنن. در این صورت کانال در زمان کمتری clear میشه و می تونیم با بهره وری بیشتری از کانال استفاده کنیم . به این شکل :




- Ethernet CSMA/CD algorithm :

به عنوان یک نمونه ی عملی استفاده از الگوریتم CSMA/CD می تونیم پروتکل Ethernet کلاسیک رو در نظر بگیریم.

توی این پروتکل وقتی کارت شبکه یه دیتاگرام رو دریافت می کنه ، یه frame می سازه و بعد میاد کانال رو sense می کنه و اگه کانال آزاد

بود، frame رو ارسال می کنه و اگه کانال اشغال باشه، ارسال frame رو تا زمانی که کانال آزاد شه به تعویق میندازه.

اگه در زمان ارسال frame ، (که نود داره همزمان لینک رو رصد می کنه) نود collision ای احساس نکنه ، کارت شبکه کارش با frame تموم میشه. اما اگه کارت شبکه هنگام ارسال frame خودش متوجه بشه که collision رخ داده، دیگه ارسال frame رو ادامه نمیده و یه سیگنال jam ارسال می کنه تا بقیه ی نود ها هم متوجه collision بشن.

در صورت بروز collision نود هایی که frame هاشون دچار مشکل شده ، بعدا باید در زمان دیگه ای اقدام به ارسال مجدد کنن. برای این کار یه بازه ی زمانی رو نود ها در نظر می گیرن و هرکدوم به صورت تصادفی از اون بازه ی زمانی یک زمان رو انتخاب می کنن. هرچی تعداد نود ها بیشتر بشه باید طول بازه ی زمانی هم بزرگتر در نظر گرفته بشه تا احتمال collision کمتر بشه. اما چالشی که داریم اینه که نود ها از تعداد هم خبر ندارن. برای این چالش یه راهکاری به اسم binary (exponential) backoff در پروتکل Ethernet استفاده میشه به این شکل که یه نود بعد از یک بار collision میاد یه عدد تصادفی بین صفر و یک انتخاب می کنه و اگه اسم این عدد رو k بگذاریم ، اون نود بعد از k* 512 bit times منتظر می مونه ، و بعد مجددا frame اش رو ارسال می کنه. اگر در تلاش دومش باز هم collision رخ داد ، این

بار میاد k رو به طور تصادفی از بازه ی {0, 1, 2, 3} انتخاب می کنه و بعد از k* 512 bit times برای بار سوم frame اش رو ارسال می کنه. در حالت کلی بعد از m تا collision ، k رو از بازه ی $\{0, 1, 2, \ldots, 2^m-1\}$ انتخاب می کنه.

هوشمندی این مکانیزم اینه که وقتی collision ها تعدادشون افزایش پیدا می کنه ، شاخص این هست که انگار تعداد نود ها زیاده و میایم به صورت داینامیک بازه ی زمانی (که ازش عدد انتخاب می کنیم) رو بزرگ در نظر می گیریم تا به این ترتیب وقتی نود ها زیاد هستن طول بازه ی زمانی که نود ها موقع retransmission توش پخش میشن بزرگ باشه.

نهایتا بعد از این که نودی که دچار collision شده بود، موفق میشه frame خودش رو ارسال کنه ، یا اگه بعد از تلاش های زیاد مدام collision رخ بده، اعلام می کنه کانال دچار مشکله و از ارسال frame منصرف میشه.

نکته : bit time مدت زمانیه که برای ارسال یه بیت صرف میشه و این زمان بستگی به سرعت لینک داره .

- CSMA/CD efficiency

محاسبه ی این مقدار به سادگی همین مقدار در پروتکل ALOHA نیست. توی فرمول محاسبه ، هرچی مقدار $t_{prop}$ کاهش پیدا کنه

efficiency افزایش پیدا می کنه. یه پارامتر دیگه ای هم توی این فرمول هست به اسم $t_{trans}$ که اگه مدت زمان مورد نیاز برای ارسال frame ها افزایش پیدا کنه مجددا efficiency افزایش پیدا می کنه. (مثلا اگه طول frame ها خیلی بزرگ باشه، وقتی یه نود موفق میشه کانال رو در اختیار بگیره ، برای مدت زمان زیادی بدون این که مشکلی پیش بیاد از کانال می تونه استفاده کنه؛ چون سایر نود ها نمیان frame هاشون رو ارسال کنن و باعث collision بشن؛ بنابراین بهره وری کانال زیاده ، اما چون ما نمیخوایم یک نود همش از کانال استفاده کنه، در پروتکل های CSMA/CD محدودیتی برای حداکثر طول frame وجود داره). فرمول :

$$\text{efficiency} = \frac{1}{1+5\frac{t_{prop}}{t_{trans}}}$$

به این ترتیب efficiency به عدد یک هم می تونه نزدیک بشه و از این بابت الگوریتم CSMA/CD از الگوریتم ALOHA بهتر هست. ساده ، ارزون، و کاملا decentralized هست و synchronization ای برای نود ها نیاز نداریم.

- "Taking turns" MAC protocols

- کلاس سوم از روش های multiple access هستن.
- انگیزه ی لازم برای معرفی این روش ها ، اینه که روش های قبلی که تا حالا گفتیم ، ( channel partitioning و random access ) مزایا و معایبی داشتن . توی این روش ، سعی میشه که از مزایای دو روش قبلی استفاده بشه و معایب رو تا اون جایی که میشه نداشته باشه.
- یک پروتکلی در زمینه ی روش های taking turns هست به اسم پروتکل polling ، که در این پروتکل یک نود به عنوان master عمل می کنه و سایر نود ها slave هستن. Master میاد پیام های دعوتی برای slave ها ارسال می کنه و هر کدوم از slave ها که اون پیام رو دریافت کرد اگه داده ای برای ارسال داشته باشه مجازه که تا حداکثر یه تعداد frame ارسال کنه و اگه داده ای نداشته باشه master رو مطلع می کنه که اون پیام دعوتش رو برای یه نود دیگه ارسال کنه.

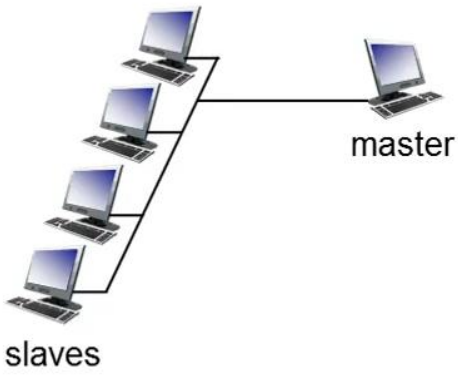
چالش هایی که این روش داره :

1 - می تونه overhead اش اذیت کننده باشه چون از کانالی که به صورت اولیه باید برای ارسال داده ازش استفاده کنیم، برای پیام های کنترلی استفاده می کنیم.

2 - باعث latency میشه؛ مثلا اگه فقط یکی از این نود ها frame برای ارسال داشته باشه، اما وقتی به ترتیب polling انجام می دیم،

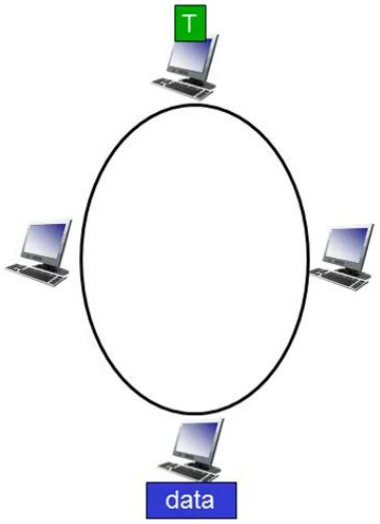
اون نود باید هر از چند گاهی منتظر بمونه تا سایر نود ها پیام دعوت رو دریافت کنن و به master اعلام کنن داده ای برای ارسال ندارن، تا دوباره نوبت به این نود برسه تا بقیه ی frame هاش رو ارسال کنه.

3 - Single point of failure : اگه master از کار بیفته سایر نود های شبکه مکانیزمی برای استفاده مشترک از کانال ندارن و همه چیز بهم می ریزه.




- پروتکل دیگه ای هست تحت عنوان token passing ، که می خواد یه سری از معایب روش قبل رو برطرف کنه، یعنی نود master رو از بین ببره. به این شکل که میاد یه پیامی تحت عنوان token بین نود های شبکه دست به دست میشه، هر نودی که این پیام در اختیارش قرار بگیره، اگه داده ای برای ارسال داشته باشه تا یه حداکثر تعداد frame ای ارسال می کنه و token رو در اختیار نود همسایه اش قرار میده.اگر

هم داده ای برای ارسال نداشته باشه بلافاصله token رو برای نود همسایه اش ارسال می کنه.




توپولوژی ای که نود ها باید داشته باشن تا از همچین پروتکلی استفاده کنن، باید حالت حلقوی داشته باشه تا هر نودی نود همسایه اش رو بشناسه و بعد از مدتی token دوباره به نود اولیه برگرده.

چالش هایی که این پروتکل داره :

1 - Overhead (مثل پروتکل قبلی)

2 - Latency (مثل پروتکل قبلی)

3 - Single point of failure : درسته که یه نود مرکزی نداریم ، اما اگه هرکدوم از این نود ها دچار مشکل شدن، دوباره این پروتکل از کار میفته.