Internet Control Message Protocol(ICMP)

- جزئیات این پروتکل در RFC 792 اومده.
- این پروتکل برای ارتباط لایه ی شبکه ای بین روتر ها و host ها استفاده میشه و دوتا کاربرد اصلی داره :

1 - گزارش دادن خطا (error reporting) : به این نحو عمل می کنه که اگه حادثه ای در روتر ها اتفاق بیفته که باعث بشه بسته ای که قراره ارسال کنن رو drop کنن، یه گزارش خطا برای فرستنده ی اون بسته ارسال می کنن.(مثل پیام unreachable destination network)

2 - ارسال و دریافت اکو (echo request/reply)

- معمولا ICMP به عنوان یه پروتکل لایه ی شبکه شناخته میشه ولی در حقیقت بالای لایه ی شبکه هست، و پیام های ICMP در داخل دیتاگرام های IP ارسال میشن.protocol number مربوط به ICMP ، 1 هست و وقتی یه end system یا یه روتر دیتاگرامی رو دریافت می کنه که protocol number در اون یک هست ، محتویات دیتاگرام رو تحویل ICMP در اون روتر یا end system میده.

- پیام های ICMP شامل type و code و 8 بایت اول IP دیتاگرامی هستن که باعث شده اون پیام ICMP تولید بشه. به خاطر این که وقتی پیام به دست فرستنده برسه، فرستنده بتونه بفهمه که این پیام ICMP متناظر با کدوم بسته ای بوده که ارسال کرده.

- لیستی از زوج های type و code های پیام های ICMP :

Type	Code	description
0	0	echo reply (ping)
3	0	dest. network unreachable
3	1	dest host unreachable
3	2	dest protocol unreachable
3	3	dest port unreachable
3	6	dest network unknown
3	7	dest host unknown
4	0	source quench (congestion control - not used)
8	0	echo request (ping)
9	0	route advertisement
10	0	router discovery
11	0	TTL expired
12	0	bad IP header

یه پیام ICMP جالب هست به اسم source quench که به ندرت استفاده میشه ، اما هدف اولیه ی این پیام اینه که مکانیزم کنترل ازدحام رو در اختیار روتر ها قرار بده، اگه یه روتری دچار ترافیک سنگین شده می تونه به یه host ای، یه پیام ICMP با type = 4 و code = 0 ارسال کنه و تقاضا کنه که rate ارسالش رو پایین بیاره.

- برنامه ی Traceroute برای track کردن یه مسیر از مبدأ تا مقصد ارسال میشه ، و مبنای کارش اینه که یه سری پیام های UDP با یک port number تصادفی -که به احتمال زیاد در مقصد وجود نداره- ارسال میشه .

در اولین پیام UDP ، TTL برابر 1 قرار داده میشه ، در پیام دوم برابر 2 و ... . برای همین وقتی n امین پیام به n امین روتر می رسه ، چون TTL توی n امین روتر برابر با صفر میشه ، یک پیام ICMP با type = 11 و code = 0 (که همون TTL expired هست) برای مبدأ ارسال میشه و احتمالا آدرس IP و نام روتر هم در این پیام ICMP قرار داده شده. برنامه ی traceroute بعد از این که این پیام ICMP رو دریافت کرد، متوجه این روتر در مسیر بین فرستنده و گیرنده میشه.

همزمان با ارسال هر بسته هم برنامه ی traceroute یه تایمری رو اجرا می کنه و به همین دلیل وقتی این پیام ICMP ِn ام به دست مبدأ رسید ، RTT تا روتر n ام هم قابل محاسبه هست.

معمولا این طوره که به ازای TTL n سه تا پیام می فرستیم تا بتونیم اون تغییراتی که توی RTT اتفاق میفته رو یه میانگین ازش به دست بیاریم.

- برنامه ی traceroute چطور می فهمه که پیام به جای روتر های میانی، به خود مقصد رسیده و فرایند ارسال پیام های UDP رو خاتمه میده؟(معیار توقف این فرایند چیه؟)

همون طور که گفتیم بسته هایی که برنامه ی traceroute ارسال می کنه ، سگمنت های UDP با شماره پورت تصادفی هستن که به احتمال زیاد در مقصد ، ما UDP socket به ازای اون شماره ی پورت نداریم، بنابراین یک پیام ICMP با code = 3 و type = 3 برای مبدأ ارسال میشه که به معنای destination port unreachable هست.

چون این پیام ، جنسش با جنس پیام های ICMP ای که روتر ها ارسال می کردن متفاوته، برنامه ی traceroute متوجه میشه که بسته به مقصد رسیده و فرایند ارسال بسته های UDP رو پایان میده.

Chapter 6 : The Link Layer and LANs

- Node : به هر دستگاهی داخل شبکه می تونیم بگیم node . اگه بخوایم یه تعریف دقیق تر داشته باشیم ، اون دستگاه هایی که پروتکل لایه ی لینک رو اجرا می کنن ، بهشون node گفته میشه.(اما چون هر دستگاهی که به شبکه متصله این پروتکل های لایه ی لینک رو اجرا می کنه ، می تونیم بهشون node بگیم.)
بنابراین host ها، روتر ها ، سوئیچ ها ، wireless access point ها و مودم ها همگی node هستن.

- Link : به کانال های مخابراتی که نود های همسایه رو به هم وصل می کنن link گفته میشه. Link ها می تونن جنس های مختلف داشته باشن و wired یا wireless باشن .
- Frame : وقتی یه دیتاگرام توسط هدر لایه ی لینک encapsulate میشه ، به اون بسته frame میگیم.
- وقتی یه host می خواد به یه host دیگه در شبکه ارتباط برقرار کنه ، بسته هایی که می خواد ارسال کنه ، باید توی مسیر بین این دوتا host ، لینک به لینک اون مسیر رو طی کنه تا به اون end system برسه. در هر گام، وقتی می خوایم از یه لینک استفاده کنیم، یک نود فرستنده هست و یک نود گیرنده . در هر لینک باید دیتاگرام توسط هدر لایه ی لینک encapsulate بشه و این ، مسئولیت لایه ی لینک هست که بیاد بسته ی دیتاگرام رو نهایتا از یک نود به نود همسایه ش برسونه.

- لینک های مختلفی وجود داره و برای ارسال بسته ها روی لینک های مختلف نیاز داریم که فرستنده و گیرنده ای که در انتهای اون لینک قرار گرفتن، پروتکل های مخصوص اون لینک رو در لایه ی لینکِ خودشون اجرا کنن. مثلا اگه یه لینکی wireless باشه ، باید فرستنده و گیرنده که در ابتدا و انتهای لینکی که مودم wireless و host قرار دارن ، پروتکل WiFi رو اجرا کنن تا بتونن از اون لینک رادیویی استفاده کنن.

در لینک های سیمی هم می تونیم از پروتکل هایی مثل Ethernet استفاده کنیم.

- پروتکل های مختلفی که توی لایه ی لینک داریم، از جهت سرویسی که ارائه می کنن با هم متفاوتن؛ بعضیاشون ممکنه سرویس reliable data transfer رو ارائه کنن و بعضیا این سرویس رو در اختیار نذارن. منظور از reliable data transfer روی یه لینک ، اینه که اون پروتکل لایه ی لینک ، این تضمین رو به ما میده که بسته ها و frame هایی که ارسال میشن ، حتما در سمت گیرنده بدون خطا و مشکل دریافت میشن.

تفاوت این سرویس در لایه ی حمل و نقل با لایه لینک ، اینه که پروتکل های لایه ی حمل و نقل end-to-end رو در نظر می گیرن و این تضمین رو میدن که وقتی بسته توسط فرستنده ی اولیه فرستاده میشه ، بدون مشکل به گیرنده ی نهایی می رسه. اما پروتکل های لایه ی لینک این تضمین رو میدن که وقتی frame توسط پروتکل ارسال میشه، همسایه ای که توسط لینک مستقیم بهش متصل هستیم ، بدون خطا و مشکل اون frame رو دریافت می کنه.

- مثال (برای درک تفاوت لایه ی شبکه و لایه ی لینک) :

یه شخصی می خواد از شهری به شهر دیگه ای مسافرت کنه و طی مسافرتش ، از وسایل حمل و نقل متفاوتی استفاده می کنه .

این فردی که داره مسافرت می کنه شبیه datagram هست، وسایل حمل و نقل متفاوت شبیه لینک های مخابراتی (communication link) هستن که از کنار هم قرار گرفتن این لینک ها ، نهایتا مسیر end-to-end ما شکل می گیره. کارهایی که برای استفاده از وسایل حمل و نقل انجام می دیم، شبیه پروتکل های لایه ی لینک هستن. نقش لایه ی شبکه و مسیریابی ای که انجام میده ، توی این مثال، مشابه نقش آژانس مسافرتی هست که مسیر end-to-end و برنامه ی سفر ما رو مشخص می کنه.

- Link layer : services

- با وجود این که سرویس اصلی لایه ی لینک ، انتقال دیتاگرام بر روی یک لینک از یک نود به نود همسایه ش هست، جزئیات سرویس هایی که پروتکل های لایه لینک فراهم می کنن، می تونه با هم متفاوت باشه. خیلی از این سرویس ها ، مشابه شون در لایه ی بالاتر وجود داره و با کلیت اون ها آشنا هستیم، مثل سرویس های reliable delivery ، flow control ، error detection و half-duplex و full-duplex .

1 - Reliable delivery : یک پروتکل لایه ی لینک ، می تونه ضمانت کنه دیتاگرامی که سمت فرستنده تحویل این پروتکل داده

میشه ، در سمت گیرنده در طرف دیگه ی لینک، بدون خطا و مشکل دریافت بشه.

از همون مکانیزم هایی که توی لایه ی حمل و نقل استفاده می کردیم تا این سرویس رو ارائه کنیم، می تونیم توی لایه ی لینک برای reliable delivery روی یک لینک هم استفاده کنیم.

(مواردی مثل sequence number ، acknowledgment ، retransmission و timer و ...)

در مورد لایه ی لینک ، بستگی به جنس لینک داره که آیا یک پروتکلِ لایه لینک، این سرویس رو ارائه بکنه یا نه.

مثلا توی لینک هایی که احتمال خطای کمی دارن، overhead ای که ما باید داشته باشیم تا سرویس reliable delivery رو ارائه کنیم مقرون به صرفه نیست. برای همین معمولا پروتکل های لینک های سیمی، این سرویس رو ارائه نمیدن، چون لینک های خوبی هستن و احتمال خطا توشون کمه.

اما توی یه سری از لینک ها (عموما لینک های wireless) که احتمال خطا در frame هایی که ارسال می کنیم، زیاده ، پروتکل های این لینک ها سرویس reliable delivery رو دارن. مثل پروتکل WiFi یا LTE .

2 - flow control : فرستنده rate ارسالش رو طوری تنظیم می کنه که در سمت گیرنده دچار overflow نشیم و فرستنده و گیرنده در دو سر لینک باید این قضیه رو مد نظر قرار بدن.

3 - error detection : با قرار دادن یه سری بیت داخل frame ، می تونیم در سمت گیرنده ، وجود خطا رو آشکار کنیم.

در لایه ی لینک ، ما از مکانیزم های قوی تر (نسبت به checksum که توی لایه ی حمل و نقل داشتیم) برای آشکارسازی خطا می تونیم استفاده کنیم . چون پیاده سازی decoder و encoder توی سخت افزار انجام میشه و این کارها با سرعت و کارآمدی بهتری انجام میشه.

البته این طوری نیست که با اضافه کردن بیت، بتونیم فقط وجود خطا رو آشکار کنیم، بعضی از کد ها و روش های افزونگی بیت وجود داره که نه تنها می تونیم وجود خطا رو آشکار کنیم، بلکه محل بیت هایی که دچار خطا شدن هم مشخص بشه و با flip کردن اون ها ، خطا رو اصلاح کنیم. بنابراین ما می تونیم از کد هایی استفاده کنیم که علاوه بر error detection ، error correction هم انجام بدن.

4 - error correction : این کد ها کد های قوی تری هستن و برای محاسبه ی بیت های اضافی که در داخل frame قرار می گیره،

احتیاج به محاسبات بیشتری وجود داره. برای استفاده از این افزونگی (جهت اصلاح خطا) هم محاسبات بیشتری لازمه.

چون پروتکل های لایه ی لینک برخی از مکانیزم هاشون می تونه در سخت افزار پیاده سازی بشه ، error correction به error detection که در لایه های قبلی مثل IP یا حمل و نقل داشتیم، اضافه شده.

5 - half-duplex and full-duplex : یک پروتکل لایه ی لینک ممکنه به ما اجازه بده که همزمان دو طرف لینک، داده ارسال کنیم و در این صورت داریم از لینک به صورت full-duplex استفاده می کنیم. یا این که در آن واحد فقط یکی از نود های همسایه برای دیگری داده ارسال کنه که در این صورت داریم از لینک به صورت half-duplex استفاده می کنیم.

6 - framing , link access : سرویس framing همون encapsulate کردن بسته ای هست که داخل لایه ی لینک از لایه ی بالاتر (لایه ی شبکه) دریافت میشه ، دیتاگرام در داخل payload قرار داده میشه و یه سری هدر هم به frame اضافه میشه . تقریبا همه ی پروتکل های لایه ی لینک framing رو انجام میدن.

سرویس link access اینه که توی کانال هایی که ما می تونیم بیش از یک فرستنده در آن واحد داشته باشیم، باید مکانیزم یا پروتکلی

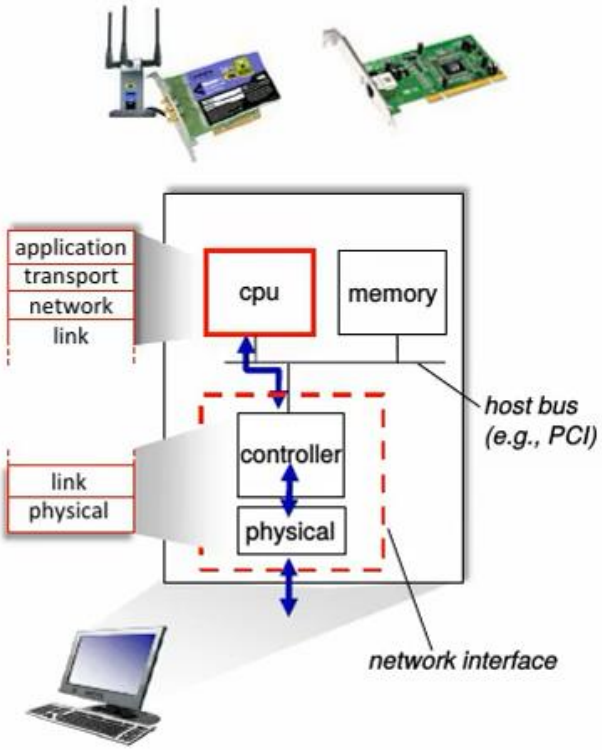
داشته باشیم که احتمال تصادف frame ها با همدیگه، حداقل بشه یا از بین بره . چون اگه دوتا فرستنده همزمان frame ارسال کنن و کانال مشترک باشه، frame ِ جفتشون از بین میره. به خاطر همین ما می خوایم این احتمال تصادف frame ها، تا جایی که میشه حداقل بشه یا کلا صفر بشه.

پس احتیاج به پروتکلی داریم که این نود هایی که به صورت distributed هستن بتونن به طور موثری از لینک مشترک استفاده کنن.به این پروتکل ها یا مکانیزم ها ، MAC(Multiple Access Control) گفته میشه. داخل این پروتکل ، یه سری آدرس استفاده میشه که بهش MAC address میگن .

- این سرویس هایی که گفته شد ، سرویس های اصلی لایه ی لینک هستن. از بین این سرویس ها ، سرویس های link access و reliable delivery از اهمیت ویژه ای برخوردارن.
- یکی از تفاوت هایی که لایه ی لینک با لایه های بالاتر خودش داره ، این تفاوت در محل و نحوه ی پیاده سازی هست. در بسیاری از موارد ، لایه لینک روی یک chip مجزایی نسبت به motherboard پیاده سازی میشه ، به این chip، network interface card(NIC) یا کارت شبکه یا network adapter هم میگن.

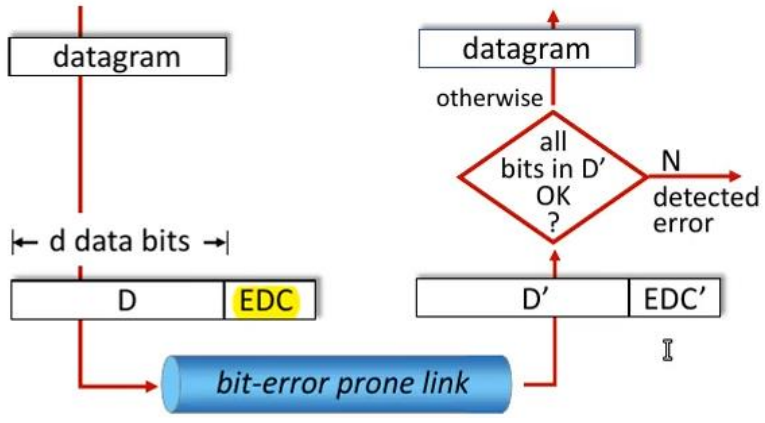
- بسیاری از سرویس های لایه ی لینک ،مثل framing و link access ، آشکار سازی و تصحیح خطا در داخل این chip انجام میشه ، بنابراین می تونیم بگیم بسیاری از کارهای لایه لینک در سخت افزار انجام میشه. ولی همچنان قسمتی از لایه ی لینک در نرم افزار پیاده سازی میشه که کارهای سطح بالاتر لایه ی لینک هستن ، مثلا driver ها که رابط بین سخت افزار و لایه های بالاتر protocol stack هستن ، این درایور ها قسمتی از سیستم عامل هستن که روی CPU اجرا میشن. معمولا با وقوع یک interrupt در سمت گیرنده یا یک system call در سمت فرستنده ، این ماژول های نرم افزاری شروع به کار می کنن.

بنابراین قسمتی از کارهای لایه لینک در سخت افزار، قسمتی در CPU ، و قسمتی به صورت نرم افزاری انجام میشه. درواقع لایه ی لینک جاییه که نرم افزار و سخت افزار در protocol stack به هم می رسن.



- error detection and error correction

- اگر ما به صورت هوشمندانه تعدادی بیت(در شکل زیر EDC نام گذاری شده) ، مبتنی بر بیت های داده، در سمت فرستنده محاسبه کنیم و اون ها رو به همراه بیت های داده ارسال کنیم، در سمت گیرنده در صورت بروز خطا، می تونیم تا حدودی مقاوم باشیم؛ به این معنی که به احتمال زیادی در صورت بروز خطا گیرنده متوجه خطا بشه و حتی بتونه بیت های مخدوش رو تصحیح کنه.(البته این قضیه ۱۰۰ درصد نیست و بازم یه احتمالی وجود داره که متوجه خطا نشه)
- به عمل اضافه کردن بیت برای مقابله با خطا در سمت فرستنده ، عمل کد گذاری یا encoding میگیم. به الگوریتمی که فرستنده برای این کار استفاده می کنه هم الگوریتم کد گذاری گفته میشه.
- در سمت گیرنده ، به عملی که میاد از بیت های اضافه برای مقابله با خطا استفاده می کنه، کد برداری یا decoding میگیم و الگوریتم مربوط به این کار هم ، الگوریتم decoding نام داره.
- قدرت آشکار سازی خطا و تصحیح خطای کد های مختلف، متفاوته. (منظور از کد های مختلف، الگوریتم های encoding و decoding عه)



سه نمونه از کد هایی که استفاده میشن :

1 - Parity checking : از این کد ها به صورت تک بعدی و دوبعدی میشه استفاده کرد.

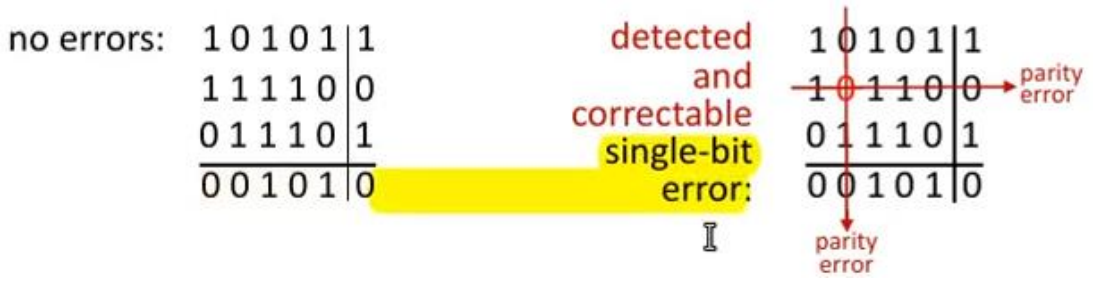
در مدل تک بعدی ، بیت های داده به صورت المان های یک بردار تک بعدی در نظر گرفته میشن و به این بردار، فقط یک بیت اضافه میشه ، به نحوی که باعث ایجاد توازن زوج یا فرد بشه. مثلا در توازن زوج ، مقدار بیت parity به نحوی تنظیم میشه که نهایتا تعداد کل بیت هایی که یک هستن ، زوج بشه. در توازن فرد هم مقدار بیت parity جوری تنظیم میشه که تعداد کل بیت های یک ، یه عدد فرد بشه.

در مدل دو بعدی ، بیت های داده رو در قالب یک ماتریس دو بعدی نمایش می دیم و بعد به ازای هر سطر یا هر ستون ، همون کاری رو که در نسخه ی تک بعدی انجام می دادیم، انجام می دیم ؛ یعنی به ازای هر سطر یک بیت parity و به ازای هر ستون هم یک بیت parity اضافه می کنیم؛ که باز این بیت parity می تونه توازن زوج داشته باشه یا توازن فرد.

دلیل استفاده از نسخه ی دو بعدی اینه که در کنار قابلیت آشکار سازی خطا، (که در نسخه ی تک بعدی داشتیم) در نسخه ی دو بعدی قابلیت تصحیح خطا رو هم پیدا می کنیم.

در نسخه ی تک بعدی اگه یکی از بیت ها flip بشه، به خاطر این که تعداد بیت هایی که یک هستن زوج یا فرد بودنشون تغییر پیدا می کنه، ما متوجه می شیم که در یک بیت خطا رخ داده یا نه. اما اگه توی دوتا بیت خطا رخ بده نمی تونیم آشکارش کنیم.

در نسخه ی دو بعدی ، قابلیت تصحیح خطا هم داریم. توی مثال زیر از توازن زوج استفاده شده و در هر سطر و ستونی تعداد یک ها زوج هست. حالا اگه توی یک بیت ، خطایی رخ بده، در این صورت توازن بیت های یک ، در یک سطر و در یک ستون به هم می ریزه و از تلاقی این سطر و ستون می تونیم محل بیتی که دچار خطا شده رو پیدا کنیم و تصحیحش کنیم.



2 - Internet checksum : از این کد ها قبلا در لایه ی حمل و نقل استفاده کردیم. این کد ها فقط قدرت آشکار سازی خطا رو دارن و نمی تونیم با این کد ها خطا رو تصحیح کنیم. قدرت آشکار سازی خطاشون هم در حد یک بیت هست. اگر تعداد بیشتری بیت دچار خطا شدن گیرنده ممکنه متوجه بشه یا متوجه نشه. (عملکرد این

کد ها مشابه عملکردشون توی لایه ی حمل و نقله؛ دیگه بازگو نمی کنیم.)

3 - Cyclic Redundancy Check(CRC) : این کد ها در آشکار سازی خطا قدرتمند تر از کد های قبلی هستن. ایده ی اصلی این کد گذاری، اینه که به رشته ی صفر و یکی که فرستنده قراره ارسال کنه، به عنوان ارقام یک عدد باینری نگاه می کنیم و اون بیت های اضافی رو جوری تعیین می کنیم که عدد نهایی ، مضرب یک عدد به خصوص بشه که بهش عدد مولد می گیم. در این صورت اگه تعدادی از بیت ها در کانال مخدوش شدن، گیرنده که مجدد میاد به رشته بیت های دریافتی به عنوان یه عدد باینری نگاه می کنه ، به احتمال زیاد این عدد در سمت گیرنده مضرب عدد مولد نیست، و به این ترتیب وقوع خطا در سمت گیرنده آشکار میشه.

مثال : فرض می کنیم قراره یه عدد در مبنای ۱۰ رو برای یه گیرنده ای ارسال کنیم، و این کار رو از طریق یه شخص ثالثی انجام میدیم و متاسفانه این شخص ثالث ، مشکل حافظه داره و ممکنه اون عددی که بهش میدیم تا به گیرنده برسونه ، بعضی از ارقامش اون شکلی که ما میخوایم به گیرنده نرسه. برای این که ما متوجه بشیم همچین اتفاقی افتاده ، یه قراردادی می ذاریم: اون عددی که برای گیرنده می فرستیم رو یک عدد سمت راستش بهش اضافه می کنیم به طوری که کل اون عدد مثلا مضرب ۳ بشه. در این صورت اگه بعضی از این

رقم ها توسط شخص ثالث تغییر پیدا کنه ، گیرنده با چک کردن این که ببینه عدد نهایی مضرب ۳ هست یا نه ، می تونه به احتمال زیادی خطا رو تشخیص بده. (در واقع اگه برخی از بیت ها تغییر پیدا کنن به احتمال زیادی ممکنه اون عدد جدید دیگه مضرب ۳ نباشه)

یک عدد رو چطور به سمت راست عدد اصلی اضافه می کنیم ؟ ابتدا عدد اولیه رو ضرب در ۱۰ می کنیم ، بعد باقی مانده ی عدد جدید رو رو بر ۳ دست میاریم، اگه باقی مانده صفر شد، رقم یکان رو تغییری نمی دیم چون عدد حاصل بر ۳ بخش پذیره. اگه باقی مانده یک بشه ، کافیه رقم یکان رو ۲ بذاریم که در این صورت عدد نهایی مضرب ۳ میشه.اگر هم باقی مانده دو شد، به جای رقم یکان ۱ می ذاریم و برای گیرنده ارسال می کنیم. در نهایت هم گیرنده عدد رو دریافت می کنه و اگه باقی مانده بر ۳ ، صفر بود فرض رو بر این می ذاره که خطایی رخ نداده و رقم یکان رو دور می ریزه و از داده ی اصلی استفاده می کنه؛ اگر هم باقی مانده صفر نبود مطمئن میشه خطایی رخ داده و عدد رو کلا دور می ریزه.

- عین همین کار در کد های CRC و در مبنای باینری انجام میشه. همه ی محاسبات در mod 2 انجام میشه. عملیات در mod 2 به این شکله که ما یه جدول جمع و یه جدول ضرب داریم :

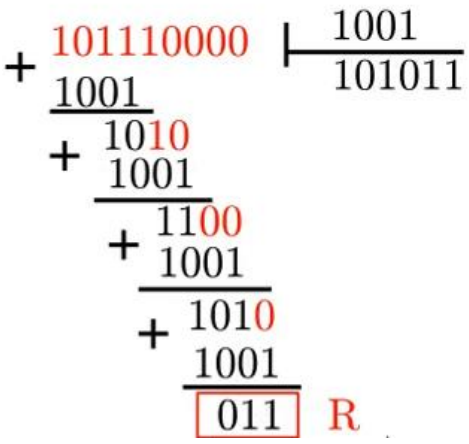
+	0	1
0	0	1
1	1	0

×	0	1
0	0	0
1	0	1

با داشتن این دو عملگر و تعریف ضرب و جمع ، می تونیم دو عملگر تفریق و تقسیم رو هم به دست بیاریم.

مفهوم تفریق وقتی میگیم یه عدد رو از عدد دیگه ای کم می کنیم، در واقع یعنی عدد اول رو با قرینه ی عدد دوم جمع می کنیم، طبق جدول جمع، قرینه ی 0 ، خود 0 عه . قرینه ی 1 هم 1 عه . پس در محاسبات mod 2 ، قرینه ی هر عدد خودش میشه.

نمونه ی تقسیم :



چون مقسوم علیه ۴ رقم داره ، باقی مانده حداکثر می تونه ۳ رقم داشته باشه، پس ۳ تا از رقم ها رو برای باقی مانده نگه می داریم.

رابطه ی هم نهشتی مقسوم و مقسوم علیه و باقی مانده:

$$101110000 \equiv 011 \pmod{1001}$$

محاسبه ی باقی مانده یه عملگر خطیه . یعنی می تونیم بگیم باقی مانده عدد 101110000 + 011 به پیمانه ی 1001 برابره با 011 + 011 که برابر با 0 هست.

حالا توی کد های CRC هم وقتی می خوایم یه رشته ی شامل صفر و یک رو ارسال کنیم(مثلا به اسم D) ، یه عدد مولد یا bit pattern به اسم G در نظر می گیریم که این G ، r + 1 بیت داره. بعد در کنار D ،یه تعداد صفر و یک قرار می دیم و با R نمایشش می دیم و می خوایم مجموعه ی این اعداد در mod 2 مضرب G باشه.

چون G ، r + 1 بیت داره پس باید R تا پوزیشن داشته باشیم و با تغییر بیت هایی که توی این پوزیشن ها هستن کاری کنیم که عدد نهایی مضرب G بشه. پس میایم D رو ضرب در 2^r می کنیم یعنی r تا صفر در کنار D اضافه می کنیم: $<D, R> = D * 2^r$

بعد برای ساختن عدد نهایی ، مقدار باقی مانده ی $D * 2^r$ بر G که برابر R هست رو به عنوان عددی که در سمت راست D باید قرار بدیم ، لحاظ می کنیم. باقی مانده حتما توی r تا بیت جا میشه چون باقی مانده ی یه عدد بر عددی که r+1 بیت داره ، حداکثر r بیتی هست و توی r بیت جا میشه.

- مثال : اگه بخوایم عدد 101110 (D)رو سمت فرستنده به گیرنده ارسال کنیم، و عدد مولد ما (G) 1001 باشه ، D رو ضرب در 2^3 می کنیم که باعث میشه ۳ تا صفر سمت راست D بیاد. بعد باقی مانده ی

این عدد به G رو حساب می کنیم که میشه 011 (R) . و این R رو جایگزین ۳ بیت آخر که عدد جدید می کنیم(یا به اصطلاح باهاش XOR می کنیم) و این عدد نهایی رو برای گیرنده ارسال می کنیم. اگه خطایی رخ نداده باشه این عدد بر G بخش پذیره و گیرنده فرض می کنه که خطایی رخ نداده،بعد هم ۳ بیت آخرش(که اضافی بودن و فقط برای error detection استفاده شدن) رو دور می ریزه و از بقیه ی عدد به عنوان داده ی اصلی استفاده می کنه. ولی اگه به G تقسیم کرد و باقی مانده صفر نشد، یعنی حتما خطایی رخ داده و عدد رو دور می ریزه.