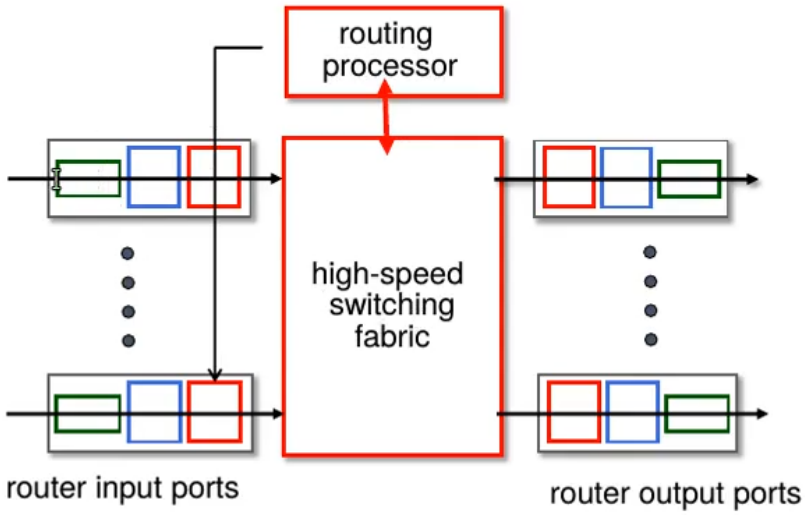
What's inside a router?

- در این درس با ساختار داخلی روتر ها آشنا میشیم.
- روتر ها مهم ترین دستگاه در شبکه های packet switch هستن و وظیفه ی مسیریابی و ارسال بسته ها رو به عهده دارن. در نتیجه آشنایی با ساختار روتر ها می تونه به درک عملکرد شبکه کمک کنه.
- توی شکل روبرو ساختار کلی یه روتر نشون داده شده :



ساختار داخلی یه روتر از ۴ قسمت تشکیل شده : ۱- پورت(های) ورودی ۲- پورت (های) خروجی ۳- routing processor ۴- switching fabric

- پورت های ورودی مسئولیت این رو دارن که کارهای مربوط به لایه ی physical و لایه ی Link رو انجام بدن. یعنی امواج الکترومغناطیس رو تبدیل به بیت کنن ،بیت ها رو فریم بندی کنن،

در سطح لایه ی لینک کارهای مربوط به error detection و error correction رو انجام بدن.

نهایتا مهم ترین کاری که انجام میدن اینه که با استفاده از مقادیر جدول forwarding ، مشخص می کنن که یه بسته باید به کدوم یک از پورت های خروجی منتقل بشه.به این کار میگیم lookup کردن و جداول forwarding ای که برای این کار احتیاج دارن ، توسط routing processor در اختیارشون قرار می گیره.

خطی که توی شکل بالا از routing processor به سمت router input ports کشیده شده ، می تونه به عنوان یک bus مجزایی باشه که جداول forwarding از طریق این bus در پورت های ورودی کپی بشه تا هر پورت ورودی یه کپی از اون رو داشته باشه و بتونه به صورت local (و با استفاده از اطلاعات داخل بسته مثل آدرس مقصد) تعیین کنه که بسته به کدوم یک از پورت های خروجی باید منتقل بشه.

حالا اگه به جای قرار دادن کپی، این جداول forwarding به صورت centralized داخل routing processor باشه ، bottle neck ایجاد میشه ؛ بنابراین برای افزایش سرعت میایم یه کپی (که بهش shadow copy هم میگن) در داخل پورت های ورودی ایجاد می کنیم.

- انتقال بسته ها از پورت های ورودی به پورت های خروجی توسط switching fabric انجام میشه. می تونیم بگیم یک شبکه ای داخل روتر هاست. چون باید بین پورت های ورودی و پورت های خروجی کانکشن برقرار کنه.

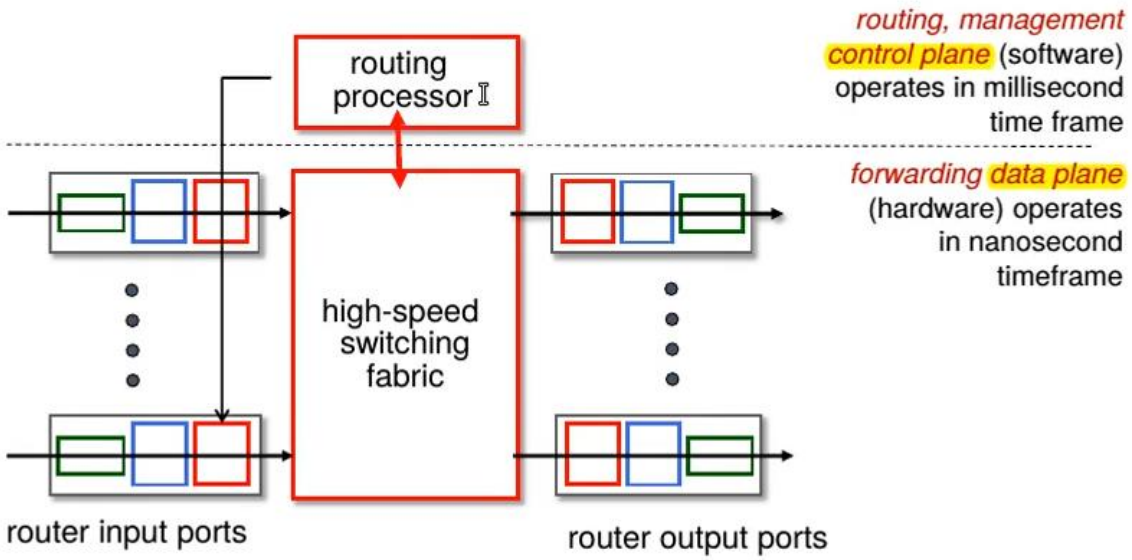
  سرعتش باید n برابر سرعت لینک باشه که n تعداد پورت های ورودی روتر هست.
- پورت های خروجی : قسمت اول پورت های خروجی که switching fabric در اون بسته ها رو قرار میده اهمیت ویژه ای داره.در واقع یه بافری وجود داره که بسته ها در اون قرار داده میشه . فلسفه ی وجود بافر اینه که ما ممکنه از چندتا پورت ورودی به یک پورت خروجی بسته بفرستیم؛ با توجه به این که سرعت لینک محدود هست نیاز به یه بافر داریم که بسته ها رو ذخیره کنیم. عمده ی مباحثی که درباره تاخیر صف در روتر ها داشتیم ، در بافر های پورت های خروجی رخ میده. ما در داخل پورت های ورودی هم بافر داریم ولی میزان استرسی که روی بافر های خروجیه ، بیشتر از پورت های ورودیه چون معمولا از یه switching fabric پر سرعت استفاده می کنیم که تا اون جایی که میشه داخل پورت های ورودی صف ایجاد نکنیم. اما در سمت پورت های خروجی ناگزیریم که بافر داشته باشیم.

در نهایت این بسته هایی که در بافر های پورت های خروجی قرار دارن وقتی نوبتشون شد ، کارهای مربوط به لایه ی physical و Link انجام میشه و روی لینک متصل به پورت خروجی ارسال میشه.

- Routing processor

گفتیم که اون پروتکل های routing ای که ما احتیاج داریم تا مسیریابی انجام بشه و جداول forwarding تهیه بشن،
داخل routing processor ها به صورت سنتی اجرا می شدن و نهایتا forwarding table ها داخل پورت های ورودی کپی می شدن.
در روش SDN ، الگوریتم های routing توی remote controller انجام میشن و routing processor وظیفه ی اصلیش برقراری ارتباط با remote controller و دریافت جداول forwarding و کپی کردنشون توی پورت های ورودی هست.
یه سری کارهای دیگه هم داریم که مربوط به مدیریت شبکه هستن که جایگاه شون توی routing processor عه.

- اگه بخوایم داخل روتر مرزبندی کنیم که کدوم ماژول ها مربوط به data plane هستن و کدوم مربوط به control plane ، به شکل زیر میشه :



از لحاظ سرعت operation ، این دوتا قسمت با هم متفاوتن . مثلا فرض می کنیم line speed = 100 Gbps باشه ، (یعنی هرکدوم از پورت های ورودی ما، لینکی که بهشون متصله 100 Gbps عه) ، و بسته هایی هم که به پورت ها می رسن اندازه شون 64 بایت هست. حالا اگه بخوایم ببینیم طول بسته در حوزه ی زمان چقدره ، باید ببینیم هر بیت در چند ثانیه منتقل میشه که میشه 0.01 نانو ثانیه. چون طول هر بسته هم 64 بایته (که میشه 64*8 بیت) پس طول بسته در واحد زمان میشه 0.01 * 64 * 8 = 5.12 ns

بنابراین در پورت های ورودی، فاصله ی بسته های ورودی از اُردر 5.12 ns هست. اون کار های مربوط به لایه های physical و link هم ، همه باید در زمان 5.12 ns انجام بشن وگرنه یه بسته ی دیگه میاد و اگه نتونیم کارای مربوط به بسته ی قبل رو در زمان 5.12 ns انجام بدیم ، دچار overflow توی صف میشیم.

در ضمن سرعت switching fabric هم باید متناسب با line speed ضرب در پورت های ورودی باشه.

پس اگه بخوایم یه time scale از سرعت کارایی که داره توی قسمت data plane روتر انجام میشه در نظر بگیریم، از اُردر نانو ثانیه هست. در سمت دیگه ی قضیه که routing processor وجود داره ، برای اجرای الگوریتم ها و کارای دیگه ای که نیاز داریم، (مثل پایش وضعیت لینک هایی که به روتر متصل هستن،) از اُردر میلی ثانیه یا ثانیه هست. به همین دلیل توی روتر هایی که high performance هستن باید قسمت data plane توی سخت افزار پیاده سازی بشه ، اما قسمت control plane می تونه به صورت نرم افزاری (یا به عبارتی در قالب سخت افزار CPU) پیاده سازی بشه.

- جزئیات بیشتر در مورد قسمت های مختلف روتر ها

۱-Input ports : پورت های ورودی شامل سه قسمت هستن . دو قسمت اول مربوط به دریافت بیت ها و فریم بندی اون ها هستن(متناظر با لایه های physical و link) . به اون ماژولی که وظیفه ی لایه ی physical رو به عهده داره ، line termination هم میگن.

دلیل این نام گذاری اینه که این ماژول به منزله ی یک ترمینال انتهایی برای بیت هایی هست که توسط line یا همون link در حال جابجایی هستن.

تمرکز ما روی قسمت سومه ، که ماژولی به اسم lookup, queueing , forwarding هست. توی این ماژول با استفاده از اطلاعاتی که در داخل هدر بسته ها هست ، و با توجه به رکورد هایی که توی forwarding table وجود داره ، عمل forwarding انجام میشه. به کل این کار ، match plus action گفته میشه. (تطبیق + عمل)

دلیل این نام گذاری اینه که ما یک سری از اطلاعاتی که داخل هدر هستن رو با یک رکوردی که داخل جدول داریم match می کنیم، و بعد با استفاده از سایر المان هایی که داخل رکورد جدول هستن مشخص میشه که چه action ای باید انجام بشه.

از لحاظ سرعت این process هایی که در داخل پورت ورودی به ازای هر بسته انجام میشه، باید حداقل با سرعت line یا link برابر باشه؛ اگه کمتر باشه ، به این معنیه که در شرایطی که لینک ما داره با سرعت نامیِ خودش کار می کنه ، قادر نیستیم که بسته ها رو بدون این که بسته ای از دست بدیم، پردازش کنیم و این باعث عملکرد نادرست روتر میشه.

اما در هر صورت به یه بافری هم احتیاج داریم. دلیل وجود این بافر نیست که ممکنه سرعت پردازش کمتر از line یا link باشه . بلکه دلیلش اینه که اگه سرعت switching fabric نسبت به سرعت دریافت بسته هایی که از تمام پورت های ورودی دریافت می کنه ، کمتر باشه ، ممکنه موقع تحویل دادن یه بسته به switching fabric، مشغول انتقال دادن یه بسته به پورت خروجی باشه ؛ پس به این دلیل احتیاج به بافر داریم تا در این شرایط بسته رو در بافر ذخیره کنیم.

در مورد عمل forwarding هم دوتا نکته وجود داره :

1 - ما دو نوع عمل forwarding داریم ، یکی بر اساس آدرس مقصد که بهش میگن destination-based forwarding که ما احتیاج داریم توی هدر فقط به IP Address مقصد توجه کنیم و براساس اون پورت خروجی مشخص میشه.(به این روش، روش traditional هم میگن).

یک روش دیگه ی forwarding هم بهش generalized forwarding گفته میشه، که توی این روش ما می تونیم فقط یه فیلد خاص رو در نظر نگیریم. یعنی مثلا در کنار آدرس مقصد، به این که فرستنده ی بسته کی بوده هم توجه کنیم.

2 - به صورت اولیه اگر ما برای آدرس های IP ساختاری نداشته باشیم، جداول forwarding خیلی بزرگ میشن. مثلا اگه بخوایم

از روش destination-based forwarding استفاده کنیم که فقط به فیلد آدرس IP گیرنده توجه کنیم، این فیلد در IPv4 ، ۳۲ بیته و این به این معنیه که جدول forwarding باید 2^{32} تا رکورد داشته باشه که خیلی عدد بزرگیه و پروسه ی forwarding رو دچار مشکل می کنه.

برای همین آدرس های IP رو جوری تعیین می کنیم که یه سلسله مراتبی وجود داشته باشه و هر رنج از آدرس های IP مخصوص به یه منطقه ی جغرافیایی خاص باشه. به این ترتیب جداول forwarding به این شکل در میان :

forwarding table

Destination Address Range	Link Interface
11001000 00010111 00010000 00000000 through 11001000 00010111 00010111 11111111	0
11001000 00010111 00011000 00000000 through 11001000 00010111 00011000 11111111	1
11001000 00010111 00011001 00000000 through 11001000 00010111 00011111 11111111	2
otherwise	3

البته همیشه نمی تونیم انقدر ساده آدرس های IP رو تقسیم بندی کنیم و موردی که توی شکل زیر گفته شده ممکنه رخ بده :

forwarding table

Destination Address Range	Link Interface
11001000 00010111 00010000 00000000 through 11001000 00010111 00010111 11111111	0
11001000 00010111 00010000 00000100 through 11001000 00010111 00010000 00000111	3
11001000 00010111 00011000 00000000 through 11001000 00010111 00011000 11111111	1
11001000 00010111 00011001 00000000 through 11001000 00010111 00011111 11111111	2
otherwise	3

Q: but what happens if ranges don't divide up so nicely?

Network Layer: 4-

ما ممکنه یه تبصره ای داشته باشیم که توی یه sub range از range قبلی که مثلا به پورت صفر فرستاده می شدن، حالا به پورت ۳ ارسال بشن. ما اگه یه آدرس IP مقصدی دریافت کردیم که توی رنج اولمون باشه ، باید صفر رو ادامه بدیم و ببینیم آیا sub range دیگه ای پیدا می کنیم که دامنه اش کوچکتر باشه ، و اگه کوچکتره اون رو ملاک قرار میدیم که آدرس IP رو به چه پورتی ارسال کنیم.

اما خوشبختانه ی روش هوشمندانه تری هم وجود داره به اسم longest prefix matching که ازین قضیه استفاده می کنه که رنج های آدرس IP متناظر با این هستن که یه تعداد از بیت های اول IP Address مث هم هستن . بنابراین متناظر با هر رکوردی لازم نیس که دوتا IP Address رو به عنوان شروع و انتهای اون رنج مشخص کنیم، کافیه IP های اولیه که توی اون

رنج همه ی بیت هاشون مثل هم هست رو به عنوان فیلد اول جدول مون یادداشت کنیم ، و توی فیلد دوم جدول بیایم پورت خروجی متناظر با اون prefix رو یادداشت کنیم.

اون sub range و range که راجع بهش صحبت کردیم هم به این شکل می تونه پیاده سازی بشه که اگه یه آدرس آی پی متناظر با یک رنجی بود، که اون رنج ، sub range ِ یه رنجِ اولیه ی دیگه ای در نظر گرفته میشه، این sub range به عنوان ملاک عمل قرار گرفته میشه. Sub range بودن یک چیزی نسبت به یه رنجی ، خودش رو توی طول prefix بزرگتر نشون میده. یعنی اگه ما توی رکورد های جدولمون ، وقتی بیت های destination IP address مون رو میخوایم match کنیم و ببینیم این بیت های اولیه متعلق به کدوم یکی از این prefix ها هست ،اگه دوتا از رکورد های جدول با تعدادی از بیت های اولیه ی آدرس آی پی مون match شد، نهایتا اون رکوردی برنده میشه که طول prefix بزرگتری داره ، چون طول prefix بزرگتر ، نشون دهنده ی sub range بودن اون محدوده ی IP address نسبت به حالت دیگه هست.

مثال : دوتا IP address به ما داده شده و می خوایم پورت خروجی شون رو با استفاده از جدول forwarding تعیین کنیم :

11001000 00010111 00010110 10100001

11001000 00010111 00011000 10101010

Destination Address Range				Link interface
11001000	00010111	00010***	********	0
11001000	00010111	00011000	********	1
11001000	00010111	00011***	********	2
otherwise				3

- در مورد اول ، اگه بریم بررسی کنیم ، می بینیم بزرگترین prefix ای که در رابطه با این IP address ، match میشه ، همون رکورد اول جدولمون هست ، بنابراین interface مون هم همون interface صفر میشه.
- تو مورد دوم اگه بریم بررسی کنیم می بینیم که رکورد سومی که داخل جدولمون هست تا یه حدودی از بیت ها رو match می کنه اما اگه این قضیه رو ادامه بدیم می بینیم یه رکورد دیگه ای داخل جدول هست که رکورد دومه، و تعداد بیتی که از این آدرس IP ، match می شن ، توی این حالت بزرگتر از حالت قبله ، و طبق قانون longest prefix match در این حالت باید رکورد دوم رو ملاک قرار بدیم و این IP address رو برای interface شماره ی یک ارسال کنیم.

اما این کوچکتر شدن جدول های forwarding تا یه حدی توسط این مکانیم ها قابل انجامه ، و ما نهایتا با جدول های forwarding ای روبرو هستیم که اگه بخوایم به روش سنتی برای سرچ کردن در اون ها استفاده کنیم، از لحاظ زمانی به مشکل بر می خوریم.

راهکاری که برای این قضیه پیشنهاد شده ، یه راهکار سخت افزاریه ، و میایم از تکنولوژی ای به اسم TCAMs(Ternary Content Addressable Memories) در داخل input port ها برای حافظه استفاده می کنیم.

ویژگی ای که این حافظه ها دارن اینه که اگه به عنوان ورودی ، Destination IP address رو بهشون بدیم ، ظرف یه کلاک ساعت اون پورت خروجی ای که متناظر با اون IP address هست در خروجی حافظه ایجاد میشه.یعنی به عبارت دیگه پیچیدگی ما از O(n) تبدیل میشه به O(1). روتر های پر سرعت از این تکنولوژی برای lookup کردن استفاده می کنن.

- در مورد پورت های ورودی ، مهم ترین کارها lookup ، forwarding و queueing بودن و همه ی این کارها به اضافه ی کارهای دیگه ای باید در زمان کمی انجام بشه.(کارهای لایه های physical و link و قسمتی از لایه ی شبکه به جز forwarding . مثلا یه بسته ای که دریافت میشه بعضی از فیلد های لایه ی شبکه اش ، مثل checksum و ttl و version باید بررسی بشه و بعضی ازین فیلد ها هم مقادیرشون باید بازنویسی بشه مثل checksum و ttl) . علاوه بر این به منظور کارهای مدیریتی هم باید داخل پورت های ورودی کارهای آماری انجام بشه . به عنوان مثال ما یک counter ای داریم که قراره تعداد بسته هایی که از پورت های ورودی به روتر می رسن رو شمارش کنه. بنابراین

وقتی یه بسته ی جدیدی به پورت ورودی می رسه ، باید مقدار counter افزایش پیدا کنه.

۲-Switching Fabrics :

این بخش قلب تپنده ی یک روتره و وظیفه ی اصلی این قسمت انتقال بسته ها از پورت های ورودی به پورت های خروجی هست.

توی این بخش یه پارامتری هست به اسم switching rate که به صورت تعداد بسته های منتقل شده از پورت های ورودی به پورت های خروجی در واحد زمان تعریف میشه . معمولا بر حسب ضریبی از line speed یا line rate بیان میشه. در شرایطی که تعداد ورودی های ما برابر با N تا هست ، مقدار مطلوبی که برای switching rate انتظار داریم برابره با :

N * line rate

- از لحاظ تکنولوژی سه حالت عمده برای ساخت switching fabric وجود داره : ۱- shared memory ۲- shared bus ۳- interconnection network
- ساده ترین روشی که برای ساخت switching fabric وجود داره و در واقع نسل های اولیه ی روتر ها از این روش برای ساخت switching fabric استفاده می کردن ، استفاده از memory هست. توی این روش پورت های ورودی و خروجی به عنوان وسایل جانبی متصل به یک

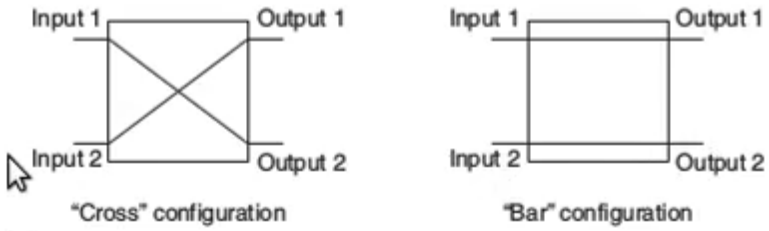
CPU در نظر گرفته می شدن ، و با ورود یک بسته به یک پورت ورودی یک interrupt به CPU داده می شد و بسته توسط system bus از پورت ورودی داخلِ حافظه ی سیستم ذخیره می شد و وقتی بسته توی حافظه قرار می گرفت ، CPU می تونست هدر های مربوط به لایه ی IP رو بررسی کنه و مبتنی بر اون عملیات های lookup و forwarding رو انجام بده ، و نهایتا مجددا از طریق system bus بسته رو از حافظه داخل پورت خروجی مناسب کپی کنه و توسط اون پورت بسته ارسال بشه.

مشکلی که این روش داشته اینه که اولا یک CPU برای همه ی بسته هایی که به روتر می رسن استفاده می کنیم و باعث میشه که مشکل سرعت پیدا کنیم.(سرعت CPU ، bottle neck میشه) . دوم این که از یک حافظه استفاده می کردن و سرعت خوندن و نوشتن توی حافظه به ازای هر عمل forwarding ای که توسط switch ها باید انجام بشه ، محدودیت زا هست و توی روتر های پر سرعت نمیشه ازش استفاده کرد. اما مبتنی بر این ایده ، ما روتر های پر سرعتی داریم ، که مشکلات طرح ابتدایی رو برطرف کرده . مثلا به ازای هر پورت ورودی یه CPU استفاده می کنن که عملیات های lookup و forwarding رو به صورت local انجام میده. و همچنین به جای یک مموری ، چندین مموری به صورت موازی استفاده میشه و مشکلات سرعت مربوط به یک مموری و یک CPU حل میشه.

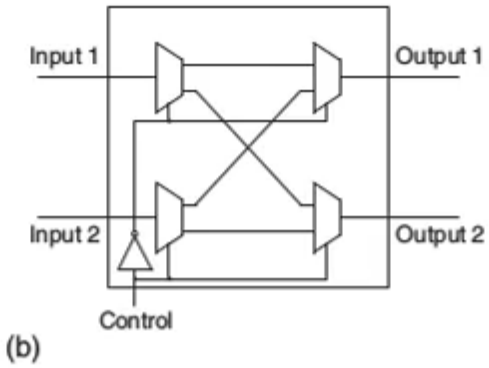
- روش دیگه برای ساخت switching fabric ، استفاده از یک bus هست . bus یک لینک پر سرعت سراسری هست که سایر وسایل جانبی یک سیستم کامپیوتری می تونن به اون لینک متصل بشن و توسط اتصال به این لینک مشترک ، بتونن برای همدیگه بسته بفرستن.
مشکلی که استفاده از این روش داره اینه که switching rate ما محدود میشه به پهنای باندِ bus و چون ما تا یه حدی بیشتر نمیتونیم سرعت bus رو افزایش بدیم ، به همین دلیل خیلی نمیتونیم به switching rate های بالایی توی این معماری برسیم، هر چند که با سرعت های bus در حد 32Gbps روتر های داریم که می تونن برای مصارف با سرعت متوسط ازشون استفاده کنیم.(مثل access router ها)
یک راه حل برای حل مشکل پهنای باند bus مشترک، اینه که بیایم از یه شبکه به جای یه bus مشترک استفاده کنیم که به این شبکه میگیم interconnection network که می تونه شکل ها یا انواع مختلف داشته باشه ، ۱- شبکه های crossbar : متشکل از 2N باس هستن که N پورت ورودی رو به N پورت خروجی متصل می کنن. هر باس عمودی که متناظر با یه پورت خروجی هست ، هر باس افقی که متناظر با یه پورت ورودی هست رو در یک cross point قطع می کنه. این cross point ها داخل هر کدومشون یه cross bar switch وجود

داره که این سوئیچ ها ها می تونن توسط switching fabric controller مدیریت بشن و configuration شون عوض بشه.

Crossbar به این دلیل نام گذاری شده که این سوئیچ ها دو حالت دارن ، یا به صورت cross هستن یا به صورت bar .



این شکلی :

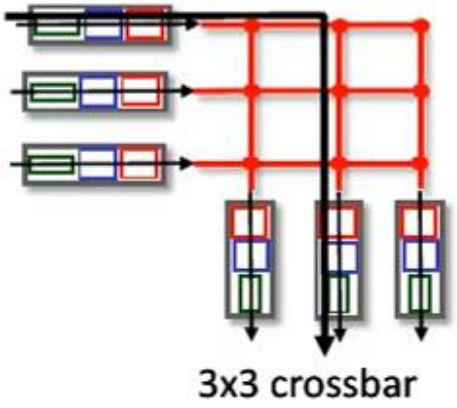


بنابراین اگه یه پورت ورودی برای یه پورت خروجی بخواد بسته ارسال کنه، cross point هایی که روی باس های این دو پورت هستن به نحوی کانفیگ میشن که این پورت ورودی بتونه با قرار دادن بسته روی باس خودش ، بسته رو برای پورت خروجی ارسال کنه. توی این ساختار همزمان بیش از یک بسته می تونه بین پورت های ورودی و خروجی مختلف انتقال پیدا کنه یا به عبارت دیگه پورت های ورودی به شرطی که بخوان برای پورت های خروجی متفاوت از هم دیگه بسته ارسال کنن، می تونن به صورت همزمان این کار رو انجام بدن. امکان ارسال

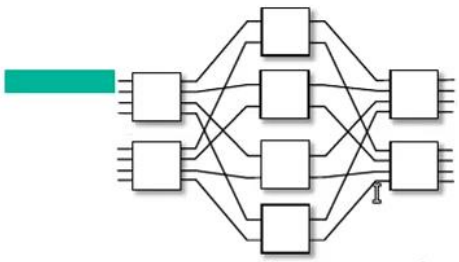
همزمان بسته ها باعث یه مزیت سرعتی نسبی توی این روش (نسبت به روش های قبلی) میشه.

این ساختاری که داخل روتر می بینیم(یه شبکه داخل روتر) خیلی مفهوم شبکه های packet switch رو تداعی می کنه. چرا؟

چون اگه هرکدوم از این پورت های ورودی و خروجی رو شبیه فرستنده و گیرنده در نظر بگیریم، همون طور که در شبکه های packet switch میومدیم یه circuit رو توسط سوئیچ هایی که داشتیم ایجاد می کردیم ، و مادامی که فرستنده و گیرنده باهم ارتباط داشتن این circuit در اختیارشون قرار می گرفت ، این جا هم دقیقا یه همچین اتفاقی میفته و توسط مجموعه سوئیچ هایی که داریم میتونیم یه circuit برای کانکشن بین پورت های ورودی و خروجی مختلف ایجاد کنیم و مادامی که پورت های ورودی و خروجی در ارتباطن،این circuit به طور انحصاری در اختیارشونه.

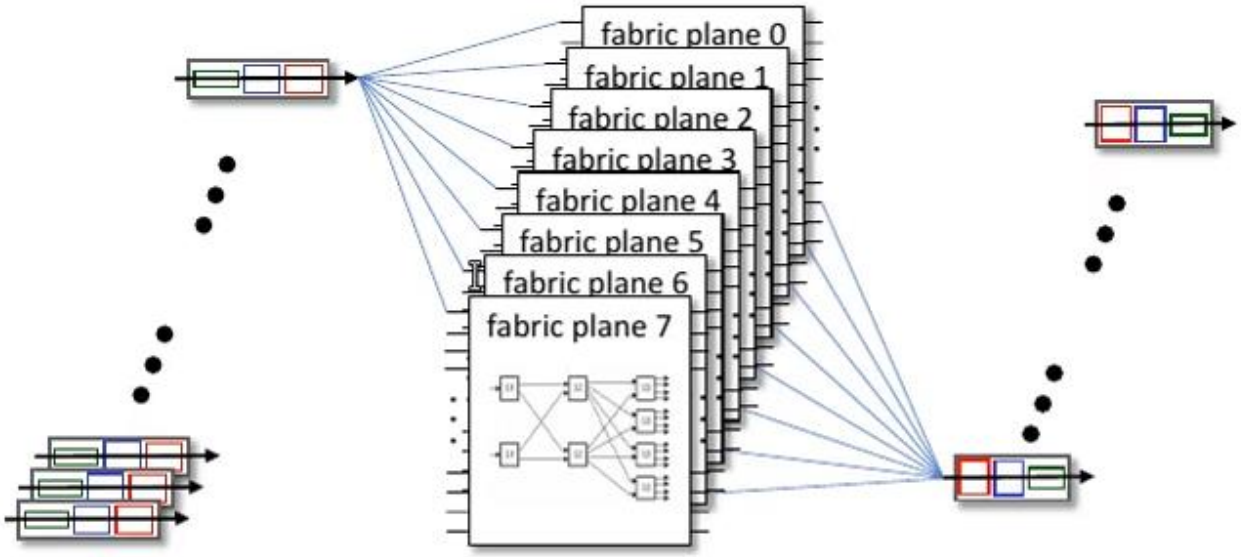


۲ – multistage switch : یه روش دیگه برای ساخت شبکه های interconnection عه . به این شکله :



توی این معماری ما سوئیچ های بزرگ رو توسط سوئیچ های کوچک تر تحقق میدیم و خود این قضیه از لحاظ پیاده سازی مهمه چون اگه تعداد ورودی ها و خروجی ها زیاد باشه ،پیاده سازی به صورت cross bar مقرون به صرفه نیست ، و میتونیم از این معماریِ multistage استفاده کنیم که سوئیچ های کوچکتری داره . علاوه بر این ، باعث افزایش سرعت هم میشه ، به این ترتیب که وقتی ازین شبکه استفاده می کنیم ، بین هر پورت ورودی و هر پورت خروجی چندین مسیر وجود داره و فقط یه مسیر وجود نداره ؛ بنابراین میایم datagram رو fragment می کنیم(به قسمت های کوچکتر میگیم cell) و بعد cell های مختلف رو از مسیرای مختلف ارسال می کنیم تا بتونیم از این ظرفیت وجود چند مسیر بهتر استفاده کنیم؛ این باعث افزایش switching rate میشه. در نهایت این قطعه های مختلف وقتی میخوان به پورت خروجی برسن ، reassemble میشن.

- روش دیگه ای که می تونیم برای افزایش switching rate شبکه های interconnection استفاده کنیم بحث موازی سازیه. توی شکل زیر، هم از روش multistage و هم موازی سازی استفاده شده. با استفاده ازین معماری ما روتر هایی داریم که switching rate اون ها به چند صد Tbps هم می رسه.

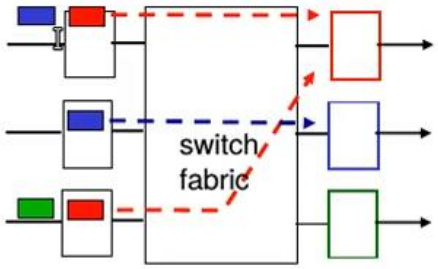


- اگر switching rate مون N * R باشه که N تعداد پورت های ورودی و R سرعت لینک های متصل به پورت های ورودی، در این صورت صفی داخل پورت های ورودی شکل نمی گیره. اما اگه به هر دلیلی switching rate کمتر از سرعت input port ها باشه ، ناگزیر صف تشکیل میشه و ما نیاز داریم بافر هایی داخل پورت های ورودی داشته باشیم و بسته ها داخل اون بافر ها قرار بگیرن ، تا نوبتشون بشه که از switching fabric استفاده کنن.

این بافر ها و تاخیری که بسته ها توی بافر ها پیدا می کنن، یکی از مولفه های تاخیر صفه، یعنی در واقع قسمتی از کل تاخیر صف ، مربوط به تاخیریه که بسته ها داخل پورت های ورودی باهاش مواجه میشن.

همچنین اگه این بافر ها overflow کنن ،می تونه باعث گم شدن بسته ها بشه!

- یه نوع تاخیری که ممکنه در پورت های ورودی رخ بده، HOL(Head of the line) blocking نام داره. مثال :



output port contention: only one red datagram can be transferred. lower red packet is *blocked*

فرض می کنیم در یه زمان به خصوصی ، وضعیت بافر های پورت های ورودی به این شکله :

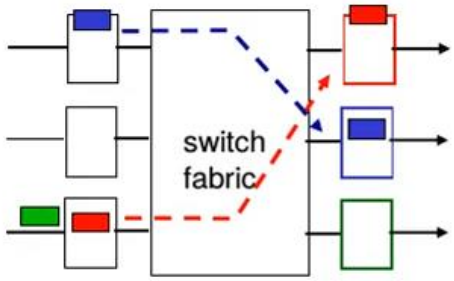
رنگ آمیزی بسته ها و پورت های خروجی بر این اساسه که چه بسته ای به چه پورت خروجی ای ارسال میشه.

بسته ی آبی رنگ بدون مشکل در زمان بعدی ، به پورت خروجی متناظر خودش انتقال داده میشه.

اما از بین دو بسته ی قرمز رنگ که یکی در پورت ورودی اوله و یکی در پورت ورودی سوم، فقط یکیشون می تونه در زمان بعد به پورت خروجی قرمز رنگ منتقل بشه، چون ما فرض کردیم switching fabric ، crossbar network عه و اگه پورت های مقصد یکسان باشن نمیتونیم همزمان بسته ها رو ارسال کنیم.

توی این مثال فرض شده که switching fabric controller ، بسته ی اول رو انتخاب می کنه که به پورت خروجی قرمزرنگ ارسال کنه،

برای همین وقتی در زمان بعد می ریم شرایط بسته ها و بافر ها رو نگاه می کنیم، به این شکله:



one packet time later: green packet experiences HOL blocking

می بینیم که بسته ی آبی رنگ در بافر پورت خروجیِ دومه، بسته ی قرمز رنگ هم در بافر پورت خروجیِ اوله، ولی در بافر ورودی سوم هیچ تغییری ایجاد نشده.

این تاخیری که بسته ی قرمز رنگ باهاش مواجه میشه بهش میگن contention delay یا تاخیر تنازعی. به خاطر این که یه تنازعی بین این دوتا بسته ی قرمز رنگ بوده و بسته ی قرمز اول به پورت خروجی رفته و بسته ی قرمز دوم دچار تاخیر شده.

به جز تاخیری که بسته ی قرمز متحمل میشه، بسته ی سبز رنگ در بافر پورت ورودی سوم هم متحمل یه نوع دیگه از تاخیر میشه. چون اگه به جای بسته ی قرمز ، قرار بود بسته ی سبز رو بفرستیم، بدون این که هیچ نزاعی رخ بده ، می تونستیم بسته ی سبز رنگ رو برای پورت سبز رنگ خروجی ارسال کنیم، و از تمام ظرفیت switching fabric استفاده کنیم. بنابراین بسته هایی که در ابتدای صف هستن و باعث block شدن سرویس دهی به بسته های قبلی داخل صف میشن، یه

تاخیری برای اون بسته ها ایجاد میشه که بهش میگن HOL(head of the line) blocking .

البته با مکانیزم هایی میشه جلوی این تاخیر رو گرفت ؛ مثلا اگه در بافر های پورت های ورودی مون بتونیم ترتیب ارسال بسته ها رو عوض کنیم و ببینیم اگه بسته ای دچار تاخیر contention هست ، به بسته های قبلیش که دچار این تاخیر نیستن، سرویس بدیم .

توی برخی از مطالعات دیدن که اگه برای تاخیر HOL کاری نکنیم می تونه باعث overflow در بافر های ورودی بشه ، حتی وقتی که ظرفیت لینکمون نصف ظرفیت نامی ش هست.