

شبکه های کامپیوتری لایه انتقال - قسمت 1

سیامک سرمدی، وحید سلوک

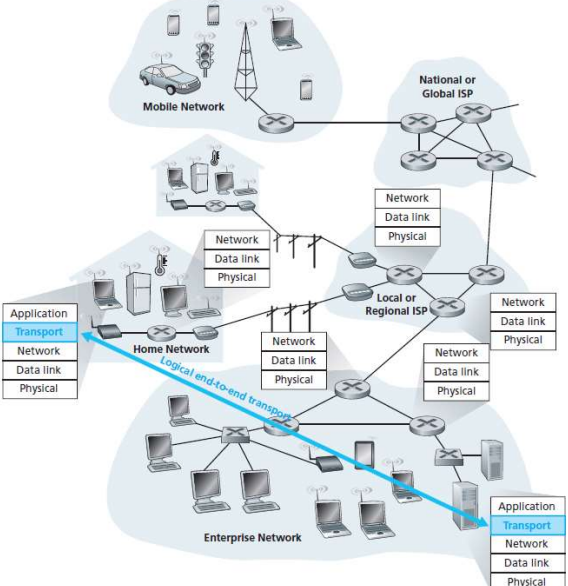
1

فهرست مطالب

- خدمات لایه انتقال
- مالتی پلکسینگ
- خدمات انتقال بدون اتصال
- معرفی پروتکل UDP
- اصول انتقال قابل اطمینان
- خدمات انتقال اتصال گرا
- معرفی پروتکل TCP
- فرآیند کنترل جریان
- مدیریت اتصال
- کنترل ازدحام

2

پروتکل ها و خدمات انتقال



- **ارتباط منطقی:** بین پردازش ها (برنامه های کاربردی) در حال اجرا در میزبان ها برقرار می شود
- به نظر می رسد که دو برنامه کاربردی بطور مستقیم به هم وصل هستند. درحالیکه ممکن است در دو سمت کره زمین بوده و داده ها از طریق تعداد زیادی روترها، لینک های متفاوت و لایه های مختلف شبکه منتقل شوند.
- انتقال اطلاعات بدون نیاز به دانستن جزئیات فوق ممکن میشود.
- روترها به هدر و محتویات بسته های لایه 4 کاری ندارند.
- **محل اجرا:** معمولاً تنها در سیستم های انتهایی اجرا میشود.
- **پروتکل انتقال در سمت فرستنده:** شکستن پیام برنامه کاربردی به قطعات (Segment) و انتقال به لایه شبکه
- **پروتکل انتقال در سمت گیرنده:** هم بندی قطعات و تحویل به لایه کاربرد
- پروتکل های لایه انتقال
- اینترنت: TCP, UDP, SCTP

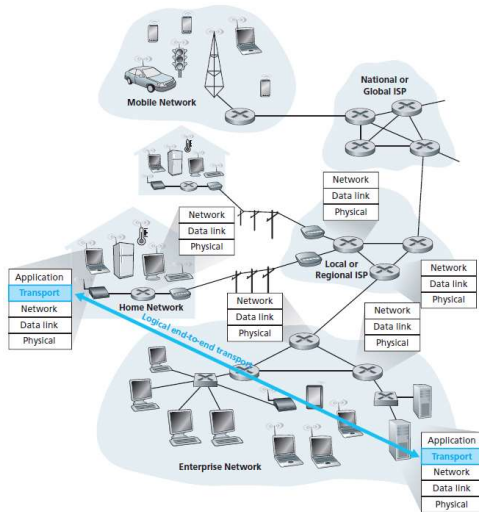
3

لایه های انتقال و شبکه

- **لایه شبکه:** اتصال منطقی میان میزبان ها را ایجاد میکند، یعنی رساندن بسته ها (packet) از یک میزبان اینترنتی به میزبان دیگر (با مخفی کردن جزئیات زیرین).
- **پروتکل IP:** ارسال بسته ها در حد امکان (بهترین تلاش)، غیر قابل اطمینان (از نظر ارسال و انتقال بی عیب)، غیر مرتب
- **لایه انتقال:** اتصال منطقی میان پردازش ها را برقرار میکند. یعنی رساندن پیام ها (با کمک سگمنت های TCP یا دیتاگرام های UDP) از یک برنامه به برنامه دیگر، شامل عملیات زیر:
 - استفاده از خدمات لایه شبکه
 - بهبود خدمات شبکه
- **مثال:** ارسال نامه از طرف 12 کارمند مستقر در سازمان A، به 12 کارمند در سازمان B
 - میزبان ها = سازمان ها
 - پردازش ها = کارمندان
 - پیام های لایه کاربرد = نامه های داخل پاکت
 - پروتکل لایه انتقال = دبیرخانه سازمان ها (فراهم کننده ارتباط منطقی بین کارمندان، فقط داخل میزبان های دو انتها کار میکنند و به نحوه انتقال نامه ها بین دو سازمان کاری ندارند)
 - پروتکل لایه شبکه = نامه رسان (فراهم کننده ارتباط منطقی بین دو سازمان)

4

پروتکل های لایه انتقال



مهمترین وظیفه، امکان پذیر کردن ارتباط بین پردازش ها، با کمک سرویس ارتباط بین میزبان ها (لایه شبکه) است.

TCP: تحویل داده قابل اطمینان – مرتب

کنترل ازدحام

کنترل جریان

برقراری اتصال

UDP: تحویل داده غیر قابل اطمینان – نا مرتب

بر مبنای بهترین-تلاش

بدون ضمانت (تحویل سالم یا مرتب)

البته در هدر یک فیلد عیب یابی (CRC) در سطح بسته دارد

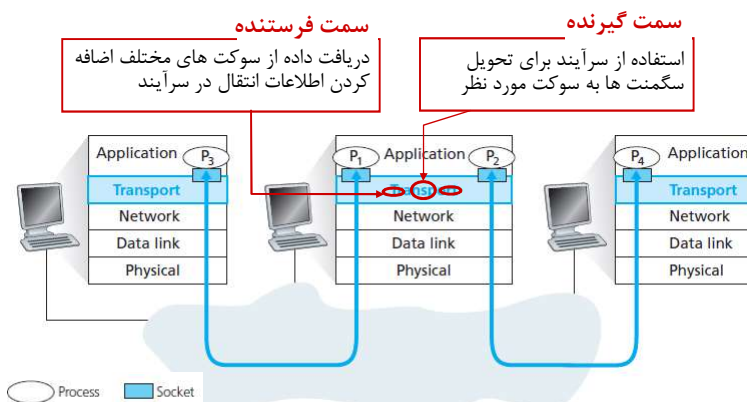
خدمات غیر قابل دسترس در پروتکل های لایه انتقال

ضمانت تاخیر

ضمانت پهنای باند

5

مالتی پلکسینگ (Multiplexing) – ارسال پیام های چند برنامه بر روی لایه انتقال



مالتی پلکسینگ: عبور دادن همزمان

اطلاعات مربوط به چند تبادل داده

(ارتباط) از روی یک لینک

جمع آوری پیام های سوکت های مختلف، ایجاد سگمنت ها و هدرهای آنها، و ارسال سگمنت های مربوط به چند برنامه مختلف، به لایه شبکه

تحویل از سریق سوکت: تحویل پیام ها از

لایه انتقال بطور مستقیم به برنامه

کاربردی انجام نمیگیرد. بلکه تحویل

سوکت رابط میگردند.

هر برنامه کاربردی ممکن است بیش از

یک سوکت فعال داشته باشد و تعداد زیادی

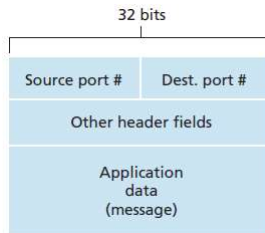
از برنامه های کاربردی ممکن است بر

روی یک میزبان فعال باشند.

6

دی مالتی پلکسینگ – دریافت پیام های چند برنامه بر روی لایه انتقال و ارسال به برنامه های کاربردی

در لایه 4



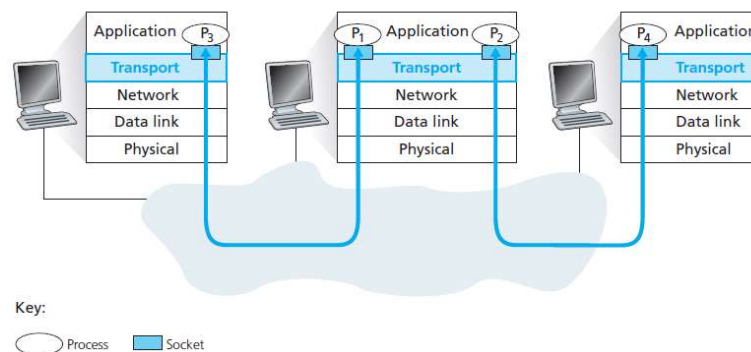
دی مالتی پلکسینگ: تفکیک داده های چند ارتباط و ارسال آنها به گیرنده های متفاوت

- تحویل سگمنت های داده دریافتی به سوکت مقصد توسط لایه انتقال
- **لایه شبکه (IP):** در سمت گیرنده هنگامی که یک بسته دریافت میشود، آدرس گیرنده و فرستنده با کمک هدر های بسته IP (packet) قابل تشخیص است. هر بسته شامل 1 بسته لایه انتقال (سگمنت)
- **لایه انتقال (TCP/UDP):** در لایه انتقال میزبان دریافت کننده، معمولاً سگمنت های مربوط به برنامه های کاربردی مختلف دریافت میگردند. برای تشخیص اینکه سگمنت (یا دیتاگرام) باید به کدام سوکت (مربوط به یکی از برنامه های کاربردی) تحویل گردد، لایه انتقال هدر های بسته لایه 4 را بررسی میکند. آدرس های (پورت) مبدا و مقصد در هدر سگمنت موجود است.
- در UDP با کمک دو مقدار شماره پورت و آدرس مقصد میتوان سوکت مقصد (متعلق به یک برنامه کاربردی خاص) را تشخیص داد.
- در TCP باید با کمک چهار مقدار آدرس مبدا و مقصد و پورت مبدا و مقصد میتوان سوکت مقصد را تشخیص داد (چون تعدادی سوکت مجزا و درست سرور برای تبادل اطلاعات با هر اتصال کلاینت ایجاد میشود و احتمالاً متصل می ماند)

7

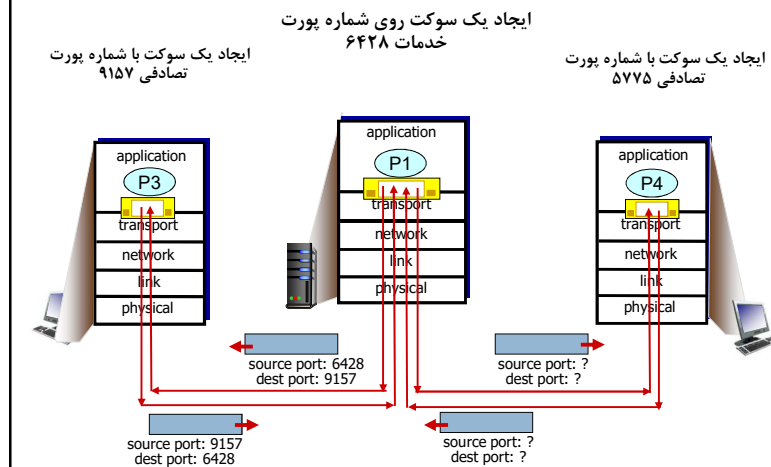
دی مالتی پلکسینگ – دریافت پیام های چند برنامه بر روی لایه انتقال و ارسال به برنامه های کاربردی

- در این تصویر، داده های ارسالی دو پروسه 3 و 4 به میزبان وسط میرسند.
- لایه انتقال میزبان وسط باید تشخیص دهد که هر سگمنت را به کدام سوکت (و در نتیجه پردازش) تحویل دهد.
- در اسلاید قبلی توضیح داده شد که تشخیص سوکت مقصد (و در نتیجه پردازش مقصد) در UDP با کمک دو مقدار "آدرس IP مقصد" و "شماره پورت مقصد" و در TCP با کمک چهار مقدار "آدرس IP مبدا و مقصد" و "شماره پورت مبدا و مقصد" در بسته های سگمنت انجام میشود.



8

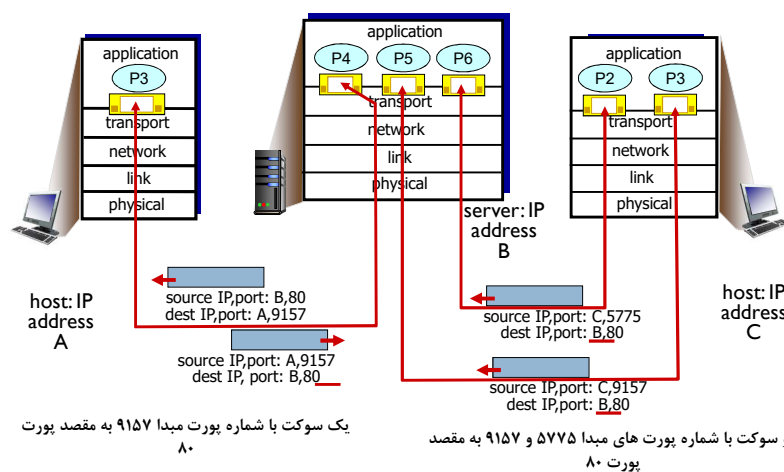
دی مالتی پلکسینگ بدون اتصال (UDP)



- در میزبان وسط سگمنت های رسیده از دو ارتباط مربوط به P3 و P4 به لایه انتقال میرسد و هر دو باید تحویل یک پروسه شوند (که بصورت Single Thread اجرا میشود)
- بنابراین تنها از شماره پورت مقصد و آدرس IP مقصد میتوان برای تشخیص سوکت مقصد استفاده کرد.
- تحویل به این پروسه به نوبت انجام میشود.

9

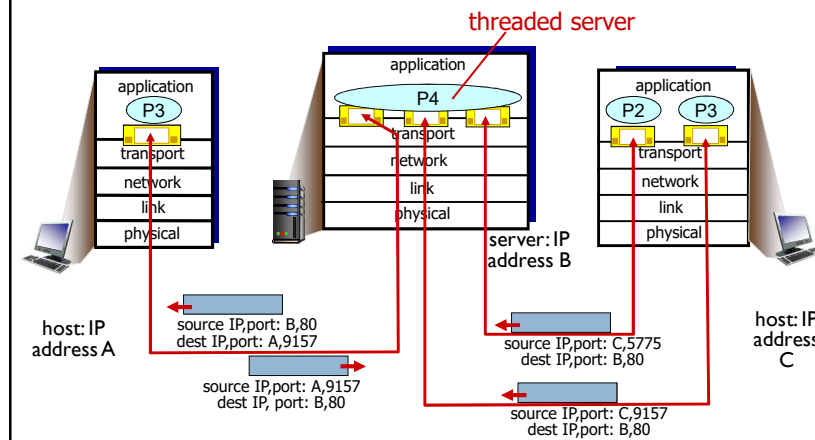
دی مالتی پلکسینگ اتصال گرا (TCP)



- در میزبان وسط بسته های رسیده، همگی برای پورت 80 و در واقع یک برنامه کاربردی (وب سرور) هستند.
- وب سرور برای پاسخگویی به هر کدام از تماس های TCP، یک پردازش جدید را اجرا می کند...
- وب سرور Apache نسخه 1.X به این شکل کار میکند.
- بنابراین فقط بررسی آدرس پورت و IP مقصد برای تشخیص سوکت مقصد کافی نیست و باید آدرس مبدا و پورت مبدا نیز برای این منظور مورد بررسی قرار گیرد.

10

دی مالتی پلکسینگ اتصال گرا: زیر-پردازش



- درمیزبان وسط بسته های رسیده، همگی برای پورت 80 و در واقع یک برنامه کاربردی (وب سرور) هستند.
- وب سرور برای پاسخگویی به هر کدام از تماس های TCP، یک thread جدید را اجرا می کند...
- هر thread یک سوکت مجزا (و در نتیجه یک ارتباط مجزا) را اجرا می کند
- وب سرور Apache نسخه 2.X به این شکل کار می کند.

11

پروتکل دیتاگرام کاربر (UDP)

- **پروتکل UDP:** یک پروتکل انتقال خلاصه، ساده و خالص است که در RFC768 تشریح شده است.
 - فرض کنید که میخواستید پروتکل لایه انتقالی بسازید که تا حد ممکن ساده بوده و سرویس خاصی را ارائه ندهد. در اینصورت احتمالاً به این فکر میکردید که پیام لایه کاربرد را گرفته و بطور مستقیم به لایه شبکه بدهید و در سمت گیرنده نیز پیام را مستقیماً از لایه شبکه به برنامه کاربردی مورد نظر تحویل دهید.
 - ولی یادگرفته ایم که برای تحویل درست بسته ها به سوکت مقصد و ارسال پاسخ به مبداء، نیاز به شماره پورت داریم.
 - پروتکل UDP پروتکلی شبیه به آنچه گفتیم است و اطلاعات محدودی در هدر آن برای تشخیص پروسه های مبدا و مقصد قرار داده شده است و تقریباً سرویس خاصی را (اضافه بر خدمات لایه های شبکه و پایین تر) ارائه نمیکند.
- **مشخصات:**
 - بدون فرآیند کنترل خطا (میتوان در لایه کاربرد اضافه کرد، مثل پروتکل مطمئن QUIC-HTTP گوگل کروم)
 - تحویل بدون ترتیب (میتوان در لایه کاربرد اضافه کرد)
 - بدون برقراری اتصال
 - بدون فرآیند دست تکانی
 - انتقال هر قطعه مستقل از قطعات دیگر
- **کاربردها:** پخش چندرسانه ای (تحمیل تلفات، حساس به تاخیر و پهنای باند)، DNS، SNMP (پروتکل اطلاعات مدیریت شبکه)، RIP

12

مزایا و معایب UDP

مزایای UDP:

کنترل بیشتر روی داده ارسالی و زمان ارسال آن: داده به محض تحویل به UDP و بسته بندی تحویل لایه شبکه میشود. در حالیکه در TCP بخاطر کنترل ازدحام و "تکرار ارسال در صورت خطا"، داده ارسالی و زمان ارسال کاملاً قابل کنترل نیست.

بدون نیاز به اتصال: تاخیر پایین

سادگی:

بدون نگهداری وضعیت اتصال (پارامترهای کنترل ازدحام، شمارنده بسته دریافتی و تاییدیه)

بدون بافر دریافت و ارسال (برای مرتب سازی و انتقال مطمئن)

اندازه سرآیند کوچک: 8 بایت، در مقایسه با 20 بایت در TCP

بدون کنترل ازدحام: سرعت انتقال بالاتر برای داده ها (مثلاً در QUIC-HTTP که کنترل ازدحام TCP را ندارد و در لایه کاربرد مطمئن شده است)

معایب امنیتی:

اگر کاربردهای زیادی به ارسال بسته های UDP بدون کنترل ازدحام بپردازند، شبکه با مشکل ازدحام شدید و اختلال مواجه میشود

ارسال بیش از حد UDP باعث ازدحام و loss بالا شده و باعث میشود که مکانیزم کنترل ازدحام ارتباط های TCP، نرخ ارسال آنها را به شدت پایین بیاورند.

13

ساختار سگمنت UDP

ساختار UDP:

ساختار قطعه/سگمنت (دیتاگرام) UDP در RFC768 تشریح شده است.

فقط چهار فیلد هدر دارد.

Source port و Destination Port: آدرس های پورت (سوکت) مبدا و مقصد

Length: طول کل قطعه با احتساب سرآیند یا هدر

Checksum: جمع مقابله ای قطعه (شامل 3 فیلد اول هدر) برای تشخیص خرابی قطعه در حین انتقال

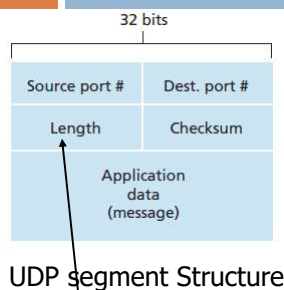
Application data: داده ارسالی لایه کاربردی (مثلاً DNS، Audio، ...)

علت استفاده از UDP از Checksum:

یکی از لینک های میانی در ارتباط ممکن است از پروتکل لایه 2 ای استفاده کند که دارای error-checking نمی باشد (چک خطا در لایه 2 اجباری نیست).

علاوه بر این ممکن است خرابی داده نه در هنگام انتقال، بلکه در حافظه روترها و تجهیزات مخابراتی اتفاق بیفتد.

اگرچه لایه UDP دارای error-checking است ولی در صورت مشاهده خرابی کاری برای دریافت دوباره قطعه نمیکند. در صورت خطا UDP یا بسته را دور می اندازد و یا با ذکر مشکل به لایه کاربرد می دهد. بنابراین با وجود داشتن این امکان، پروتکلی مطمئن به حساب نمی آید.



طول کل قطعه با
حتساب سرآیند

14

جمع کنترلی در UDP

□ سمت فرستنده:

- تجزیه هر قطعه به رشته (بلوک) های 16 بیتی
- شامل سرآیند
- جمع دو به دو بلوک های 16 بیتی به روش مکمل 1
- حفظ طول 16 بیتی حاصل جمع
- جمع حاصل با بلوک بعدی
- مکمل گیری نهایی و بدست آوردن جمع کنترلی
- قرار دادن مقدار جمع کنترلی در فیلد مربوطه

□ سمت گیرنده:

- دریافت قطعه و تجزیه به بلوک های 16 بیتی
- محاسبه جمع کنترلی با جمع دو به دو بلوک ها
- مقایسه حاصل جمع با مقدار جمع کنترلی ارسالی
- مقدار صفر: بدون خطا
- غیر صفر: وجود خطا

15

جمع کنترلی در UDP

□ مثال:

	1	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0
	1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
wraparound	1	1	0	1	1	1	0	1	1	1	0	1	1	1	0	1
sum	1	0	1	1	1	0	1	1	1	0	1	1	1	1	0	0
checksum	0	1	0	0	0	1	0	0	0	1	0	0	0	0	1	1

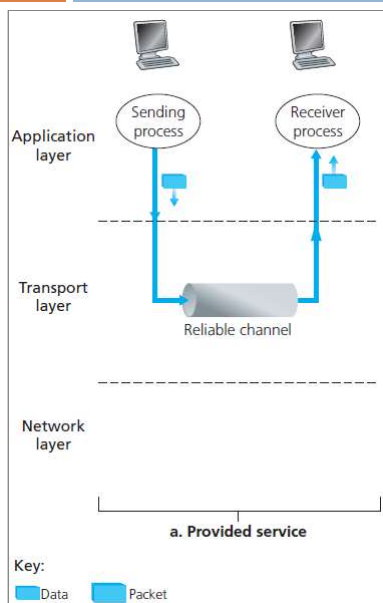
□ نکته: بیت انتقالی (wraparound) بدست آمده به حاصل جمع اضافه می شود

16

اصول انتقال قابل اطمینان

17

اصول انتقال قابل اطمینان (rdt)



□ انتقال مطمئن مسئله ای است که در لایه های کاربرد، انتقال و پیوند داده مطرح می باشد.

□ اگر لیستی از 10 مسئله مهم شبکه تهیه کنیم، احتمالاً مسئله اول خواهد بود.

□ عامل اصلی پیچیدگی در انتقال قابل اطمینان:

□ کانال انتقال غیر قابل اطمینان (مثلاً لایه شبکه IP)

■ بیت ها ممکن است تغییر یابند

■ بیت ها ممکن است از بین بروند

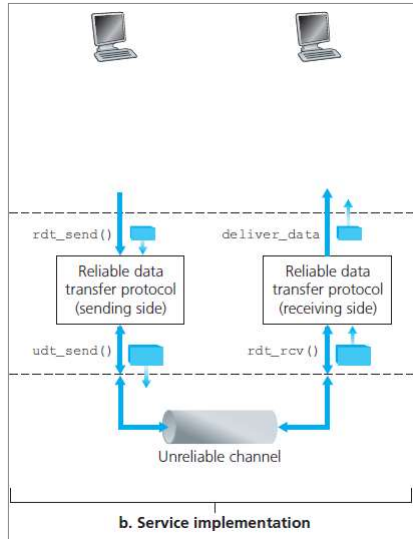
■ ترتیب داده ها ممکن است تغییر کند

□ سوال: منظور از کانال چپست و چرا غیر قابل اطمینان است؟؟

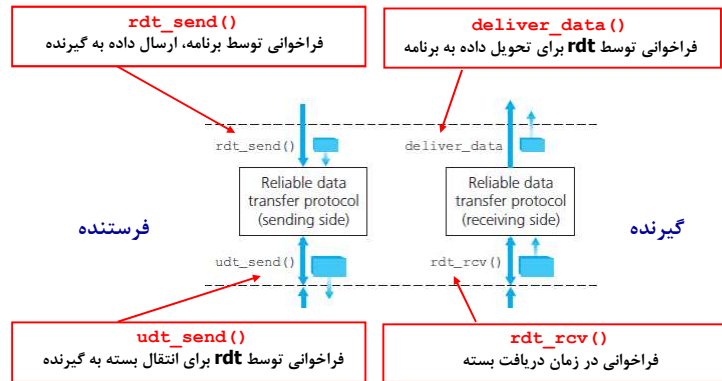
□ رسانه انتقال ممکن است تنها یک لینک و یا مجموعه ای از لینک های روی اینترنت باشد.

18

اصول انتقال قابل اطمینان (rdt)



- برای پیاده سازی یک پروتکل قابل اطمینان میتوان در نظر گرفت که توابع 4 گانه زیر باید پیاده سازی شوند.
- rdt به معنای انتقال مطمئن و udt به معنای انتقال غیر مطمئن است.
- علاوه بر داده، بسته های کنترلی نیز با udt_send توسط دو طرف ارسال میشوند.

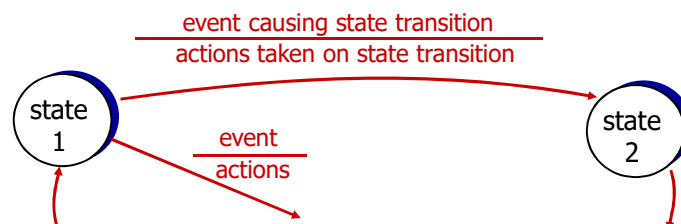


19

طراحی rdt1.0

- در این قسمت یک پروتکل قابل اطمینان بصورت گام به گام و مبتنی بر FSM طراحی میگردد. نحوه نمایش FSM به شکل زیر خواهد بود.
- خط کسری:

- صورت: event یا واقعه ای که باعث تغییر حالت می شود
- مخرج: actions یا عملیاتی که در صورت تغییر حالت اجرا میگردند



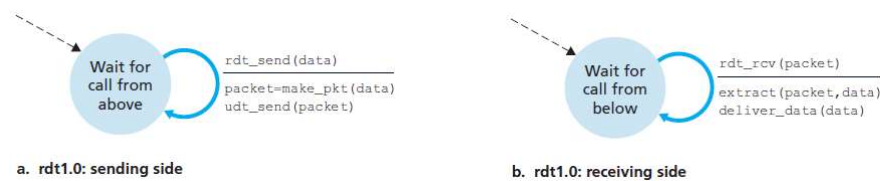
state: when in this "state" next state uniquely determined by next event

20

طراحی rdt1.0 - کانال بدون خطا

نسخه 1.0:

- **فرض اصلی:** لینک قابل اطمینان (عدم وجود خطا و تلف بسته)
- مسیر یک طرفه (یک سمت فرستنده-یک سمت گیرنده)
- فرستنده: ارسال داده روی کانال (یا در حال انتظار برای درخواست برنامه کاربردی)
 - برنامه کاربردی rdt_send() را صدا میزند
 - make_pkt پیام را قطعه قطعه کرده و بسته ها را می سازد و با udt_send() ارسال میکند.
- گیرنده: دریافت داده از کانال (یا در حال انتظار برای ورود داده از کانال)
 - لایه شبکه وقتی داده رسید تابع rdt_rcv() را صدا میزند.
 - محتوای بسته ها به هم چسبانده شده و پیام استخراج میشود.
 - لایه انتقال تابع deliver_data() را صدا میزند تا داده تحویل لایه کاربرد شود.
- با توجه به قابل اطمینان بودن لینک، داده بطور مطمئن به کاربرد مقصد میرسد.



21

طراحی rdt2.0 - کانال با خطا

- **کانال غیر قابل اطمینان:** در این طراحی فرض بر استفاده از کانال غیر مطمئن با فرضیات زیراست.
- **فرضیات:**

- همه بسته ها دریافت می گردند.
- **تغییر بیت ها در بسته** امکان پذیر می باشد (موقع ارسال، در حین پخش روی رسانه و یا در بافر دریافت کننده).
- **نحوه تشخیص خطای بیت:** با استفاده از جمع کنترلی

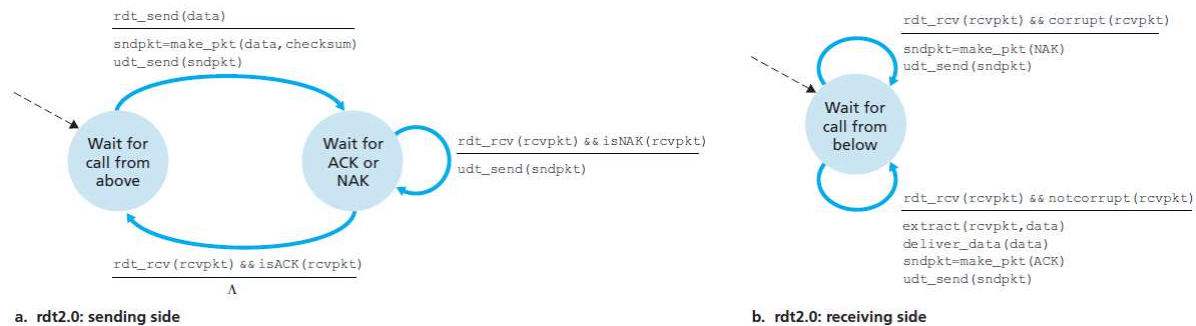
نحوه کنترل خطا:

- استفاده از پیام پاسخ مثبت (ACK): اطلاع به فرستنده از دریافت صحیح بسته توسط گیرنده
- پیام پاسخ منفی (NAK): اطلاع به فرستنده مبنی بر عدم دریافت صحیح بسته
- ارسال مجدد بسته توسط فرستنده با دریافت NAK
- حذف بسته ذخیره شده در فرستنده با دریافت ACK
- **مثال تماس تلفنی:** در تماس تلفنی وقتی پیام طرف مقابل را متوجه می شویم از تاییدیه مثبت (مثلا OK) استفاده میکنیم. در صورت نفهمیدن پیام طرف مقابل از تاییدیه منفی استفاده می کنیم (مثلا Please Repeat That)
- **ARQ:** در شبکه های کامپیوتری این پروتکل به نام Automatic Repeat Request شناخته می شود.

22

FSM - rdt2.0 سمت فرستنده

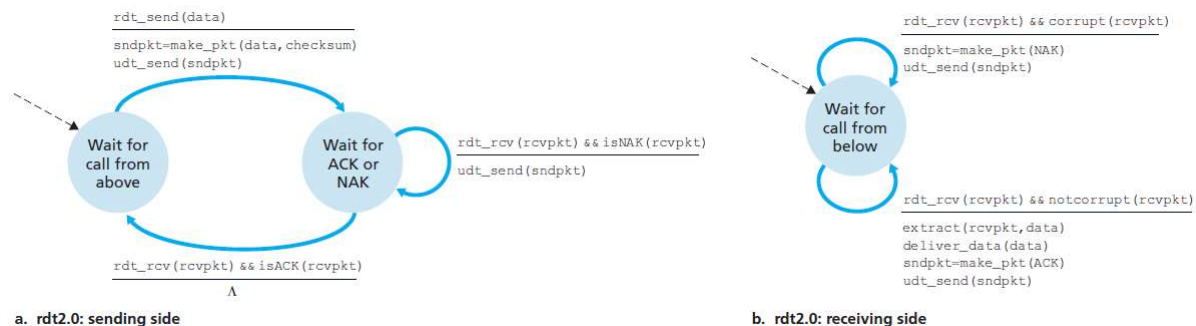
- در تصویر زیر FSM پروتکل مطرح شده نمایش داده شده است.
- **سمت فرستنده:** دو حالت دارد. حالت سمت چپ، حالت انتظار برای تحویل داده یا صدا شدن `rdt_send` توسط لایه کاربرد است. در اینصورت بسته ای شامل داده و `checksum` آن ساخته می شود و با کمک تابع `udt_send` ارسال می شود. سپس به حالت انتظار برای دریافت تاییدیه می رود. در صورتی که تاییدیه برسد و منفی باشد، بسته دوباره ارسال میشود. در صورتیکه تاییدیه مثبت باشد، به وضعیت انتظار برای دریافت و ارسال بعدی می رود.
- هنگام انتظار برای تاییدیه، امکان دریافت داده جدید برای ارسال نیست و بنابراین پروتکل **stop-and-wait** نامیده میشود.



23

FSM - rdt2.0 سمت گیرنده

- **سمت گیرنده:** در سمت گیرنده، فقط یک حالت انتظار برای صدا شدن از لایه پایین (شبکه) وجود دارد.
- در صورت خرابی بسته، یک بسته تایید منفی ساخته شده و با `udt_send` ارسال می شود و دوباره منتظر می شود.
- در صورت سالم بودن بسته، داده از آن استخراج می شود (`extract`)، و با صدا زدن تابع `deliver_data` لایه بالا به آن تحویل می شود. سپس یک بسته تاییدیه مثبت ساخته شده و با `udt_send` به سمت فرستنده ارسال می گردد. سپس دوباره در وضعیت انتظار قرار می گیرد.



24

rdt2.0: اشکالات اساسی

- شاید به نظر برسد که پروتکل rdt2.0 کار خواهد کرد. ولی این پروتکل در صورتی که پیامهای تایید منفی (NAK) و مثبت (ACK) خراب شوند، با مشکل مواجه می شود.
- 1. باید بیت های checksum به پیامهای تایید نیز اضافه شود.
- 2. باید مشخص شود که در صورت خرابی پیامهای تایید چکار باید کرد. سه راه حل پیشنهادی:
 - مشابه مکالمات تلفنی، که اگر پیام تایید شنیده نشود میگوئیم "What did you say?"، درخواست ارسال دوباره پیام تایید را بکنیم. ولی مشکل این است که اگر "پیام درخواست تکرار پیام تایید" نیز خراب شود چه؟ در اینصورت طرف مقابل نیز باید درخواست تکرار پیام کند.... مشخص است که این روش پیچیده خواهد بود...
 - به پیام ها به تعدادی بیت checksum اضافه کنیم که علاوه بر تشخیص خطا، امکان بازیابی پیام نیز وجود داشته باشد. این راه حل برای کانالی که پیام ها را خراب میکنید، ولی از بین نمیببرد مناسب است. ولی اگر کانال پیام ها را از بین ببرد، دیگر این روش جوابگو نیست.
 - راه حل مناسب تر این است که فرستنده داده تا زمانی که یک پیام تایید قابل فهم و بدون خطا دریافت نکرده، ارسال یک بسته داده را تکرار کند. تنها مشکلی که پیش می آید این است که گیرنده متوجه نخواهد شد که آیا این ارسال، یک ارسال تکراری است و یا یک ارسال جدید.
 - برای حل مشکل تشخیص ارسال های تکراری، اکثر پروتکل ها، از یک شماره ترتیب (sequence number) روی بسته های داده استفاده می کنند. در پروتکل های stop-and-wait، شماره ترتیب میتواند یک بیتی باشد (مقادیر 0 و 1 و بعد تکرار همانها برای بسته های بعد).
 - پیامهای تایید نیاز به شماره ترتیب ندارند، زیرا گیرنده پیام می داند که پیام تایید رسیده، مربوط به آخرین داده ارسالی بوده است.

25

rdt2.0: اشکالات اساسی، خلاصه

- | | |
|---|---|
| <ul style="list-style-type: none"> □ نحوه برخورد با تکرار بسته □ ارسال مجدد صرفا در صورت دریافت پیام پاسخ منفی □ حذف بسته در صورت دریافت پیام پاسخ مثبت □ اضافه کردن شماره ترتیب به هر بسته □ کنترل بسته های تکراری در گیرنده با شماره ترتیب | <ul style="list-style-type: none"> □ بروز خطا در پیام های پاسخ (ACK/NAK) □ عدم اطلاع فرستنده از وضعیت بسته □ عدم دریافت وضعیت از گیرنده □ ارسال مجدد ← تکرار بسته |
|---|---|

□ توقف و انتظار (Stop and Wait)

□ در فرستنده:

- ارسال یک بسته
- توقف ارسال
- انتظار برای دریافت پاسخ

26

rdt2.1: (فرستنده) شماره ترتیب برای پیام های پاسخ

□ در نمودار نسخه 2.1، از هرکدام از وضعیت های قبلی، دو عدد وضعیت وجود دارد. یکی برای زمانی که قرار

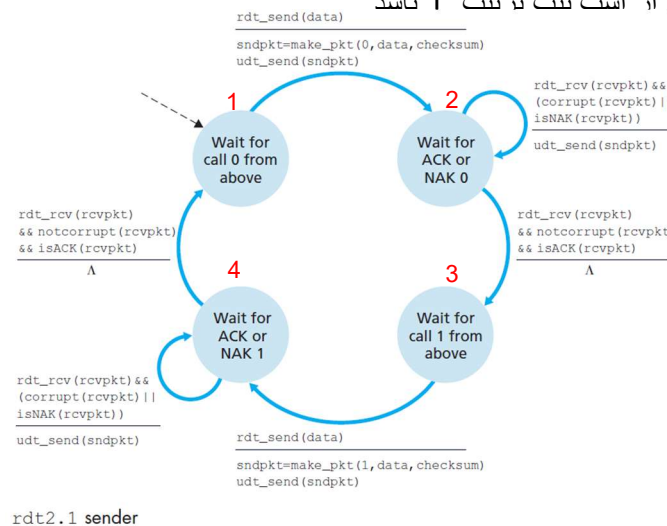
است بیت ترتیب 0 باشد، و دیگری برای زمانی که قرار است بیت ترتیب 1 باشد

□ 1: پس از انتظار برای داده از لایه بالا، داده ارسال

□ 2: سپس جواب دریافت می شود. اگر جواب NAK

□ اگر تاییدیه مثبت بود و بسته خراب نبود، رد شده و با می رود.

□ 3: به محض رسیدن داده از لایه بالا مرحله ارسال می بود...



27

rdt2.1: گیرنده

□ در نمودار سمت دریافت کننده در

نسخه 2.1، دو حالت بجای 1

حالت قبلی برای زمانهایی که

شماره ترتیب 0 یا 1 مورد انتظار

است وجود دارد.

□ 1: وقتی داده ای دریافت شد و

خراب نبود و شماره ترتیب 0

بود، بسته را استخراج کرده،

تحویل داده و بسته تاییدیه مثبت

ساخته و ارسال می شود. و به

وضعیت انتظار برای بسته بعد با

شماره ترتیب 1 می رود

(وضعیت 2)

□ 1: اگر بسته خراب بود، پیام

تاییدیه منفی فرستاده می شود و

دوباره متظر بسته ای با شماره

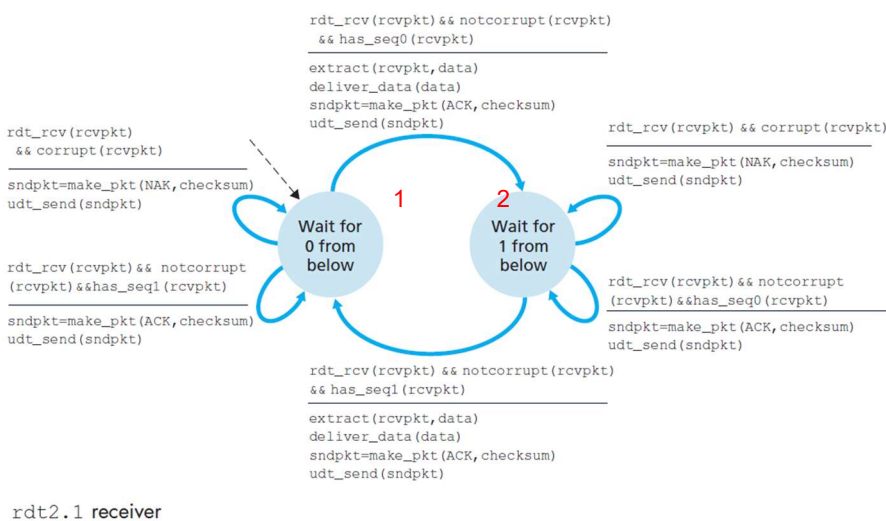
ترتیب 0 می ماند.

□ 1: اگر بسته سالم بود ولی شماره

ترتیب 1 بود، فقط تاییدیه ارسال

می شود و دوباره متظر بسته ای

با شماره ترتیب 0 می ماند.



28

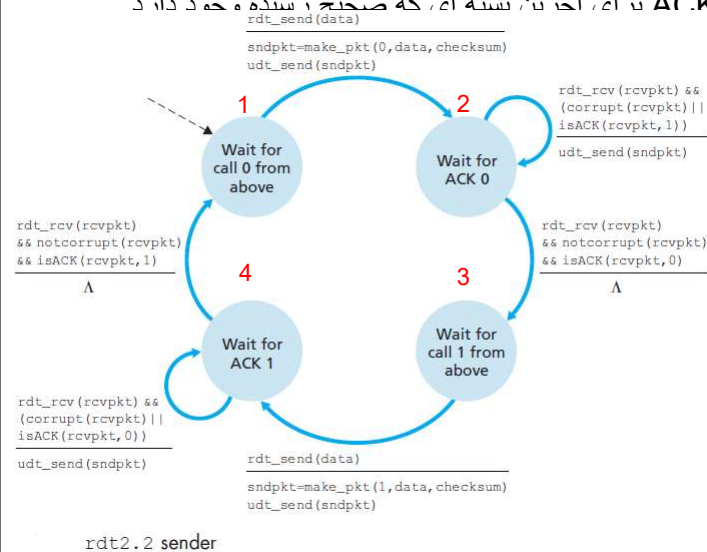
rdt2.1 - خلاصه

- **فرستنده**
 - اضافه کردن شماره ترتیب به بسته
 - شماره های ترتیب: تنها دو مقدار 0 و 1
 - کنترل خطا روی پیام های پاسخ
 - افزودن یک حالت بیشتر
 - برای تشخیص شماره ترتیب
- **گیرنده**
 - بررسی تکراری بودن بسته
 - افزودن یک حالت برای اعلام شماره ترتیب مورد انتظار
 - نکته
 - گیرنده امکان تشخیص اینکه آیا پاسخ تایید ارسالی، در سمت فرستنده به درستی دریافت شده را ندارد

29

rdt2.2 - پروتکل بدون NAK

- امکان اینکه بدون ارسال NAK و تنها با ارسال ACK، به آخر بسته ام، که صحیح و سنده چه د، دار
- **در سمت فرستنده:** بررسی تکراری بودن پاسخ (پرسش)
- ارسال مجدد آخرین بسته در صورت دریافت ACK

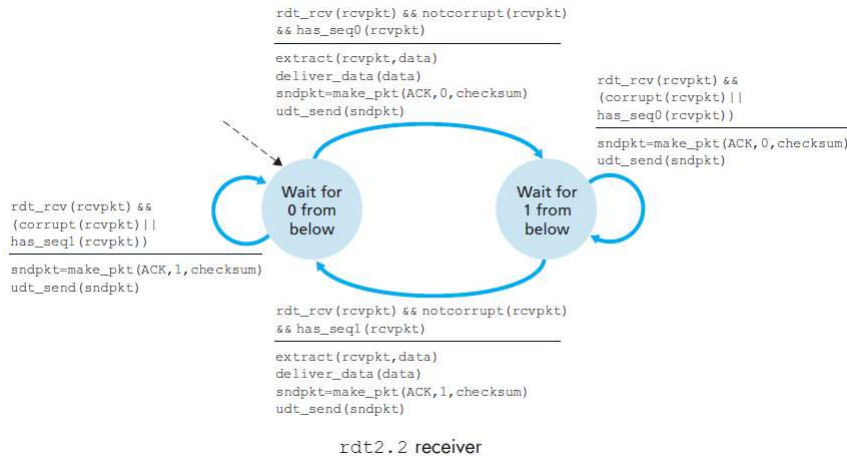


30

rdt2.2 - پروتکل بدون NAK

■ در سمت گیرنده: ارسال ACK مجدد در صورت دریافت بسته با خطا

■ شماره ترتیب پاسخ برابر شماره آخرین بسته دریافت شده سالم خواهد بود



31

rdt3.0 - کانال با خطا و تلفات بسته

□ خلاصه مساله:

■ فرض: علاوه بر خرابی بسته ها، احتمال گم شدن (تلفات) آنها (شامل بسته های داده و پاسخ) حین ارسال در کانال وجود داشته باشد.

■ اکنون باید به دو سوال زیر پاسخ گفت:

■ چطور از وقوع تلفات بسته (packet loss) مطلع شویم؟

■ برای این مسئله نیاز به مکانیزم جدیدی داریم.

■ در صورت وقوع تلفات بسته باید چه کنیم؟

■ Checksum کمک زیادی برای حل این مسئله نمی کند ولی ارسال پیام پاسخ (تایید)، شماره ترتیب و تکرار ارسال میتواند مفید باشند.

□ راه های زیادی برای اطلاع از تلف شدن بسته وجود دارد:

■ در روش پیشنهادی ما، مطلع شدن از تلفات و حل مشکل بر عهده فرستنده گذاشته میشود. در این روش:

■ فرستنده کمی (حداقل به اندازه Roundtrip time) منتظر میشود تا مطمئن شود پاسخ تاییدی برای بسته نیامده (ممکن است پاسخ خراب شده باشد، یا با تاخیر زیاد بیاید)

■ سپس فرستنده بسته را مجددا ارسال میکند.

■ در صورتی که بسته (داده/پاسخ) دارای تاخیر زیادی (بیش از مقدار در نظر گرفته شده توسط فرستنده) باشد:

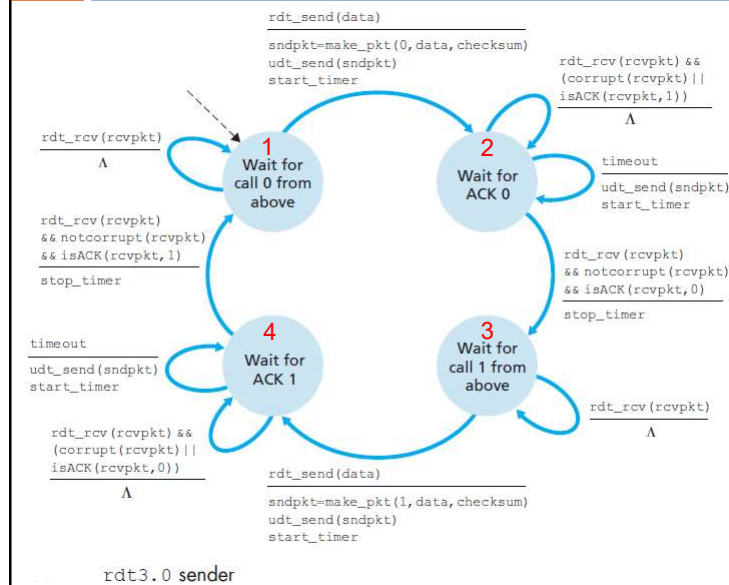
■ ارسال مجدد توسط فرستنده، باعث تکرار می شود ولی این مسئله، با شماره ترتیب مورد استفاده در rdt2.2 قابل تشخیص است

■ گیرنده: باید شماره ترتیب بسته دریافت شده را در پاسخ ذکر کند.

■ فرستنده: نیاز به تایمر شمارش معکوس در این سمت هست.

32

rdt3.0 - فرستنده



□ نمودار FSM ارسال مطمئن 3.0:

□ 1: در انتظار دریافت داده برای ارسال، اگر در این فاصله پاسخی برسد، توجه نمیکند (احتمالا پاسخ تکراری یا با تاخیر است)

داده دریافتی از لایه بالا را با شماره ترتیب 0 ارسال میکند.

□ 2: منتظر تاییدیه با شماره ترتیب 0 است اگر تاییدیه با شماره 1، یا بسته خراب دریافت کند، توجه نمیکند.

در صورت timeout داده را مجددا ارسال میکند

اگر تاییدیه سالم دریافت کرد، تایمر را متوقف کرده و به وضعیت 3 (انتظار برای داده بعدی از لایه بالا میروند)

33

rdt3.0 - گیرنده

□ تمرین:

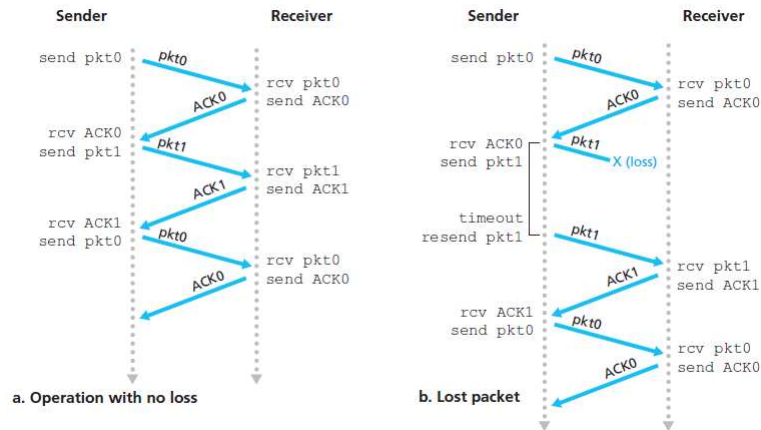
□ FSM سمت گیرنده را بعنوان تمرین رسم کنید.

□ از آنجائیکه شماره ترتیب های بسته ها در rdt3.0 بصورت یک در میان 0 و 1 هستند، به این پروتکل، پروتکل بیت متغیر (alternating-bit protocol) گفته می شود.

34

rdt3.0 - تلف شدن بسته

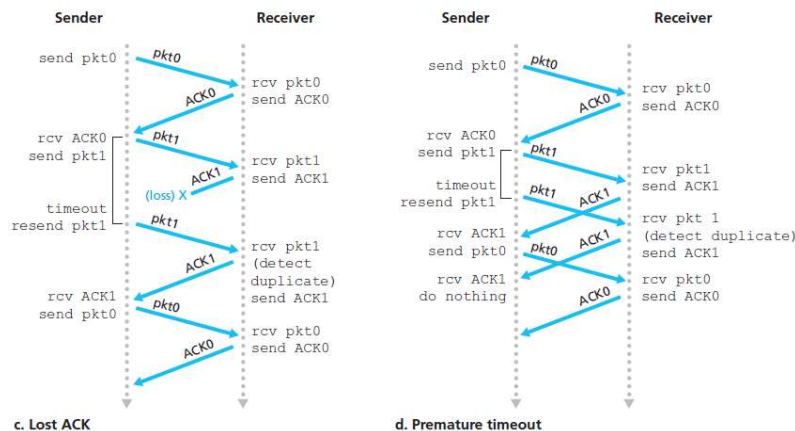
- نمودار زیر تبادل بسته های داده و پاسخ را در وضعیت عدم تلفات و وقوع تلفات نشان میدهد.
- در حالت دوم، چون بسته ارسالی گم شده، پاسخی از سمت گیرنده نمی آید. بعد از اتمام زمان تایمر، بسته دوباره ارسال می شود.



35

rdt3.0 - تلف شدن پاسخ

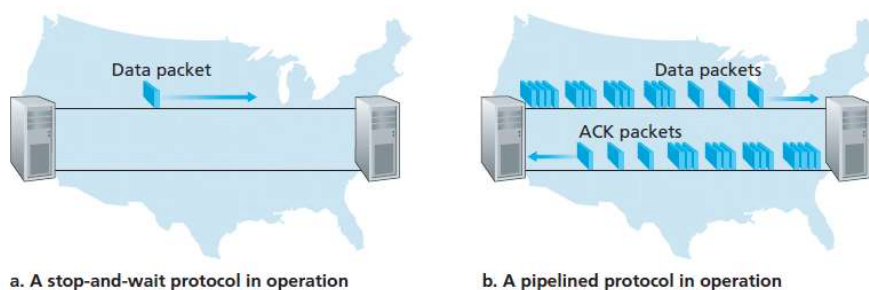
- نمودار زیر تبادل بسته های داده و پاسخ را در وضعیت تلف شدن پاسخ و تایم آوت پاسخ نمایش می دهد.
- در وضعیت دوم، به علت عدم دریافت پاسخ برای pkt1 در زمان مناسب، بسته تکرار میشود. ولی بلافاصله بعد از ارسال مجدد، تاییدیه ارسال اول می رسد و فرستنده بسته بعدی (بعد از ترتیب 1، ترتیب 0 خواهد بود) را ارسال میکند.



36

rdt3.0 - کارایی

- پروتکل rdt3.0 پروتکل درستی است و همانطور که انتظار می رود، اطلاعات را بطور مطمئن ارسال می کند. ولی کارایی آن مناسب نخواهد بود.
- با توجه به اینکه این پروتکل از نوع stop-and-wait است، پس از ارسال یک بسته پروتکل منتظر میشود تا اطمینان یابد که بسته با صحت به مقصد رسیده.
- در این فاصله پهنای باند موجود روی خط در حال اتلاف خواهد بود.
- اگر فاصله مکانی دو نقطه مبدا و مقصد زیاد باشد، این زمان اتلاف ظرفیت خط بسیار طولانی و خسارت بار خواهد بود.
- یکی از راه حل های موجود استفاده از پروتکل های خط لوله ای (pipelined) است.



37

rdt3.0 - کارایی

- مثال: فرض کنید که دو میزبان در دو سمت غربی و شرقی آمریکا باهم پیام تبادل می کنند. پهنای باند 1 Gbps و تاخیر انتشار یک طرفه 15 ms است. اگر اندازه بسته ها 8000 bit باشد:

$$D_{\text{trans}} = \frac{L}{R} = \frac{8000 \text{ bits}}{10^9 \text{ bps}} = 8 \mu\text{s} \quad \square \text{ زمان انتقال هر بسته:}$$

- نسبت زمان انتقال یک بسته به کل زمان لازم برای ارسال و دریافت تاییدیه (*utilization*)

U_{sender} : fraction of time sender busy sending

$$U_{\text{sender}} = \frac{L/R}{RTT + L/R} = \frac{0.008}{30.008} = 0.00027$$

- تاخیر انتشار بسیار پر رنگتر از تاخیر انتقال ← در هر 30 میلی ثانیه تنها 1 کیلوبایت ارسال می شود (276kb/s)
- پس udt3.0 پروتکلی دست و پا گیر و با کارایی کم است!

38

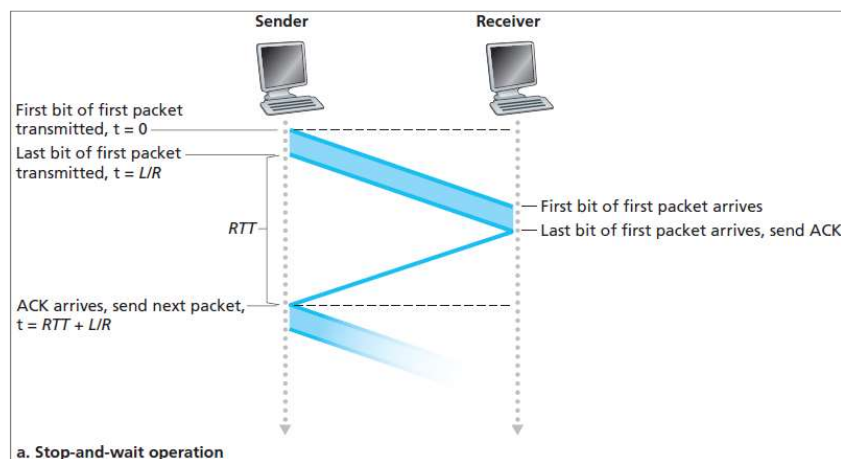
پروتکل های خط لوله ای

- راه حل پیشنهادی: امکان ارسال بسته های بیشتر در مدت زمان انتظار برای پاسخ
 - افزایش تعداد شماره ترتیب ها
 - امکان ذخیره بیش از 1 بسته در فرستنده و گیرنده
- دو پروتکل پایه مبتنی بر خط لوله:
 - *Go-Back-N*
 - *Selective Repeat*

39

پروتکل های خط لوله ای - پروتکل های stop-and-wait

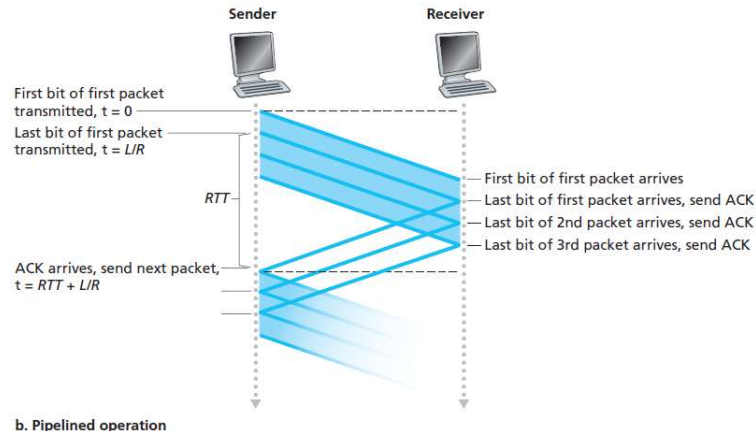
- کارایی خط هنگامی که در هر بار یک بسته ارسال شود و منتظر پاسخ برای آن باشیم، و سپس ارسال بعدی را انجام دهیم کم است.



40

پروتکل های خط لوله ای - افزایش کارایی

□ محاسبه کارایی خط هنگامی که 3 بسته در هر بار ارسال شود و منتظر پاسخ برای هر 3 باشیم.



$$U_{\text{sender}} = \frac{L/R}{RTT + L/R} = \frac{0.024}{30.008} = 0.00081$$

3-packet pipelining increases utilization by a factor of 3!

41

پروتکل های خط لوله ای

Selective Repeat □

- امکان نگهداری حداکثر N بسته در انتظار پاسخ
- گیرنده برای هر بسته دریافتی پاسخ مجزا می فرستد
- فرستنده برای هر بسته منتظر پاسخ یک تایمر دارد
- ارسال مجدد صرفاً بسته های با تایمر منقضی شده

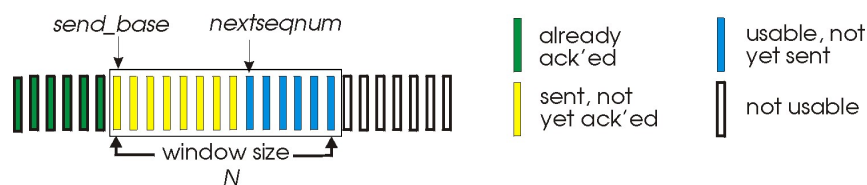
Go-Back-N □

- امکان نگهداری (بافر) حداکثر N بسته در انتظار پاسخ
- گیرنده فقط پاسخ های تجمیع شده ارسال می کند
- ارسال پاسخ فقط در صورت وجود توالی در بسته ها
- فرستنده برای اولین بسته بدون پاسخ تایمر دارد
- در صورت انقضای تایمر ← ارسال مجدد تمامی بسته ها

42

GBN - پنجره ارسال

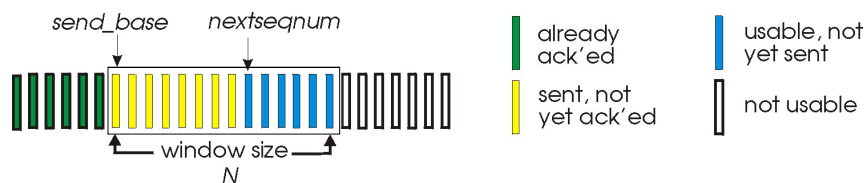
- اگر K بیت برای شماره ترتیب استفاده شود، $2^k - 1$ شماره ترتیب خواهیم داشت.
- اگر اولین بسته ارسال شده ولی تایید نشده را $base$ بنامیم، محدوده شماره ترتیب ها به 4 قسمت تقسیم می شود:
 - $[0-base-1]$: بسته های ارسال شده و تایید شده
 - $[base, nextseqnum-1]$: بسته های ارسال شده ولی تایید نشده
 - $[nextseqnum, base+N-1]$: محدوده بسته های قابل ارسال در همین لحظه (اگر داده ای برای ارسال باشد)
 - $[base+N, ...]$: محدوده اعداد ترتیب غیر قابل استفاده (دستور استفاده، تعداد بسته های تایید نشده بیش از N می شود)



43

GBN - پنجره ارسال

- شماره پاسخ باید در پنجره وجود داشته باشد (شماره ترتیب های قبل پنجره قبلا تایید شده اند، بعد پنجره هنوز قابل استفاده نیستند و ارسال هم نشده اند)
- شماره پاسخی بزرگتر از شماره $base$ ← لغزش پنجره به بعد از شماره تایید شده
- یک تایمر برای کل پنجره روی اولین بسته بدون پاسخ وجود دارد. در صورت $timeout$ شدن، ارسال همه بسته های داخل پنجره تا $nextseqnum$ تکرار میشود.



44

GBN - نمایش عملیات و بسته های ارسالی



47

GBN - مثال

□ در پروتکل GBN با اندازه پنجره 4 و تعداد شماره ترتیب 1024، گیرنده در لحظه t منتظر دریافت بسته شماره k است.

1. شماره ترتیب های موجود در پنجره ارسال (فرستنده) را مشخص کنید.

2. کدام شماره های پاسخ ممکن است در مسیر برگشت باشند؟

□ حل

1. گیرنده منتظر k ← پاسخ تا شماره k ارسال شده

1. اگر همه پاسخ ها سالم به فرستنده رسیده باشد ← پنجره ارسال: $\{k, k+1, k+2, k+3\}$

2. اگر هیچ پاسخی سالم نرسیده باشد ← پنجره ارسال: $\{k-4, k-3, k-2, k-1\}$

2. گیرنده منتظر k ← پاسخ های $\{k-3, k-2, k-1\}$ قبلا ارسال شده است.

1. اگر همه پاسخ ها تا k دریافت شده ← فقط پاسخ k در مسیر برگشت است

2. اگر هیچ پاسخی دریافت نشده ← احتمالا یکی از پاسخ های آخرین بسته های ارسال شده در مسیر برگشت. برای پنجره ارسال 4 با بازه $\{k-3, k-2, k-1, k\}$ ← حداقل پاسخ دریافت شده برابر $k-4$ بوده و یک از 4 پاسخ $\{k-3, k-2, k-1, k\}$ می تواند در مسیر باشد

48

GBN - نکات

- **علت محدود بودن سایز پنجره:** بعداً توضیح داده خواهد شد که برای کنترل جریان و همچنین کنترل ازدحام لازم است که سایز پنجره محدود شود.
- محدودیت باعث می شود که اگر به علت ازدحام، یا عدم توانایی میزبان مقصد در دریافت بسته های ارسالی، پاسخ دریافت نگردد، فرستنده با نرخ کمتری به ارسال بسته ادامه دهد.
- **محدوده اعداد ترتیب:** همانطور که قبلاً گفته شد اگر k بیت برای شماره ترتیب در نظر گرفته شود، محدوده شماره های ترتیب $[0, 2^k - 1]$ خواهد بود. بعد از اینکه شماره ترتیب $2^k - 1$ استفاده شد، شماره ترتیب بعدی 0 خواهد بود.
- در TCP شماره های ترتیب 32 بیتی هستند و بجای شمارش بسته ها، بایت های ارسال شده را می شمارند.

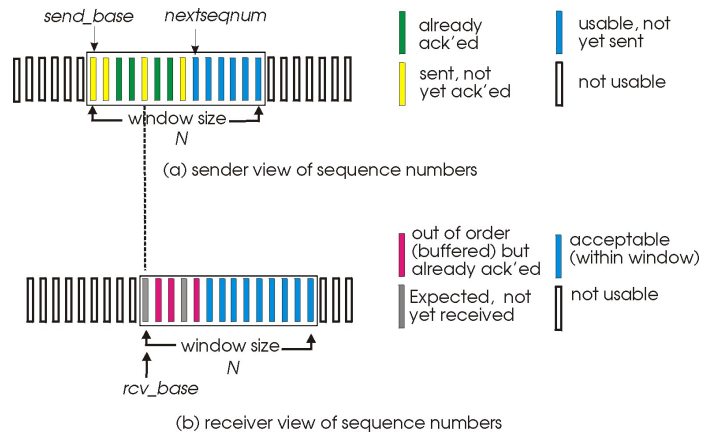
49

Selective Repeat (تکرار انتخابی)

- **پروتکل GBN:** با ارسال بیش از 1 بسته و سپس انتظار برای پاسخ، مقدار استفاده از کانال افزایش یابد ولی در بعضی مواقع خود GBN باعث ایجاد مشکل در کارآیی میشود.
- در مواقعی که سایز پنجره و همچنین حاصل ضرب پهنای باند-تاخیر بالا باشد، تعداد زیادی بسته در خط لوله خواهند بود.
- در این مواقع، تلف شدن حتی یک بسته، میتواند به تکرار ارسال همه بسته های خط لوله منجر شود (که تعداد زیادی از آنها به سلامت به مقصد رسیده اند و واقعا تکرار ارسال آنها اتلاف منابع خواهد بود).
- **روش تکرار انتخابی:** تنها بسته هایی دوباره ارسال میشوند که به سلامت به مقصد نرسیده اند (خراب یا تلف شده اند). برای این منظور:
 - دریافت تک تک بسته ها باید توسط گیرنده تایید شود.
 - برای تک تک بسته ها باید تایمر نگهداری شود.
 - همچنان باید تعداد بسته های ارسال شده ولی تایید نشده با یک پنجره با سایز N ، محدود شود.
 - ولی برخلاف GBN، ممکن است برای بعضی از بسته های داخل پنجره، تاییدیه دریافت شده باشد.

50

Selective Repeat - پنجره های ارسال و دریافت



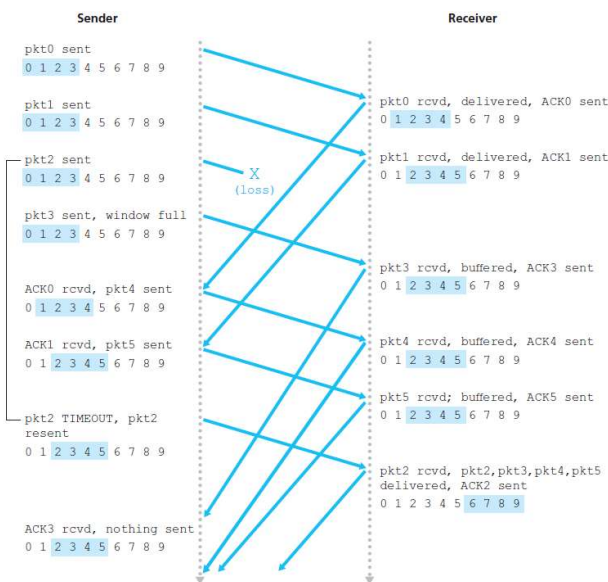
□ در سمت فرستنده، ابتدای پنجره، اولین بسته ارسال شده ولی تایید نشده است. ولی در محدوده $[base, nextseqnum-1]$ ، بعضی بسته ها تایید شده اند.

□ در سمت گیرنده، برای تمام بسته های صحیح دریافت شده ACK ارسال میشود، ولی بسته ها بافر میشوند تا قسمتی از بسته ها به ترتیب و کامل شوند و به لایه بالاتر تحویل گردند.

□ در سمت گیرنده، دید متفاوتی از پنجره وجود دارد ولی سایز پنجره مشابه بوده و محدوده های مشابهی در نظر گرفته میشود (چهار محدوده شبیه سمت فرستنده)

51

Selective Repeat - پیام های رد و بدل شده در وضعیت تلف شدن بسته



□ در تصویر مشاهده میشود که گیرنده با وجود تلف شدن بسته 2، برای بسته های بعدی یعنی بسته های 3، 4 و 5 تاییدیه ارسال میکند.

□ در ضمن فرستنده بعد از ارسال بسته 5، با وجود دریافت تایید برای بسته 3، بسته دیگری را ارسال نمیکند، زیرا سایز پنجره 4 است، و عدم دریافت تاییدیه برای بسته 2، اجازه نمیدهد که پنجره به جلو حرکت کند و بسته های بعد از انتهای پنجره شانس ارسال بیابند.

□ بعد از تایم آوت شدن بسته 2 و ارسال مجدد آن، به محض دریافت تایید برای بسته 2، پنجره میتواند 4 تا جلو برود (زیرا بسته های 3 و 4 و 5 بدرستی تایید شده اند) و ارسال بسته های 6 تا 9 انجام پذیرد.

□ در سمت گیرنده بعد از دریافت بسته 2، مجموعه بسته های 2 تا 5 به لایه بالا تحویل میگردند.

52