Chapter 2
Application Layer

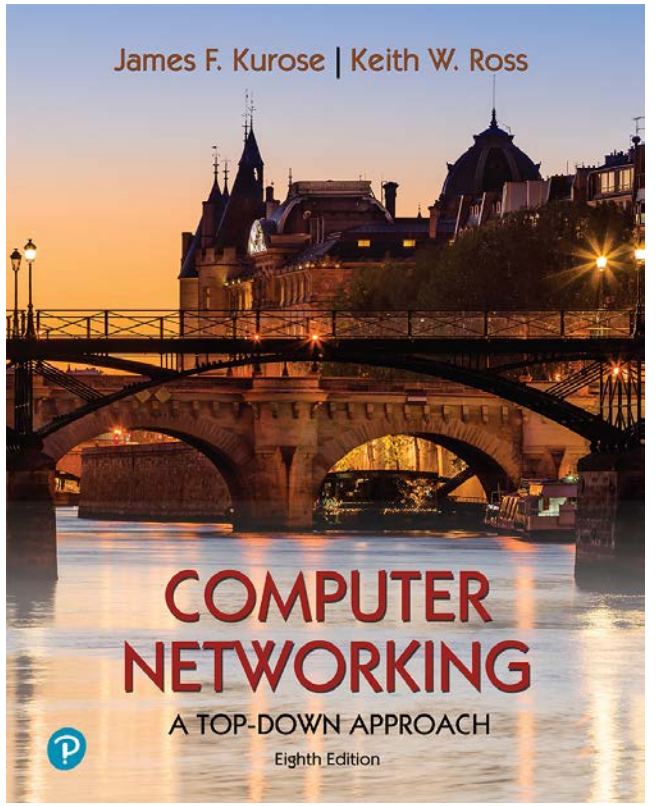
A note on the use of these PowerPoint slides:

We're making these slides freely available to all (faculty, students, readers). They're in PowerPoint form so you see the animations; and can add, modify, and delete slides (including this one) and slide content to suit your needs. They obviously represent a *lot* of work on our part. In return for use, we only ask the following:

- If you use these slides (e.g., in a class) that you mention their source (after all, we'd like people to use our book!)
- If you post any slides on a www site, that you note that they are adapted from (or perhaps identical to) our slides, and note our copyright of this material.

For a revision history, see the slide note for this page.

Thanks and enjoy! JFK/KWR

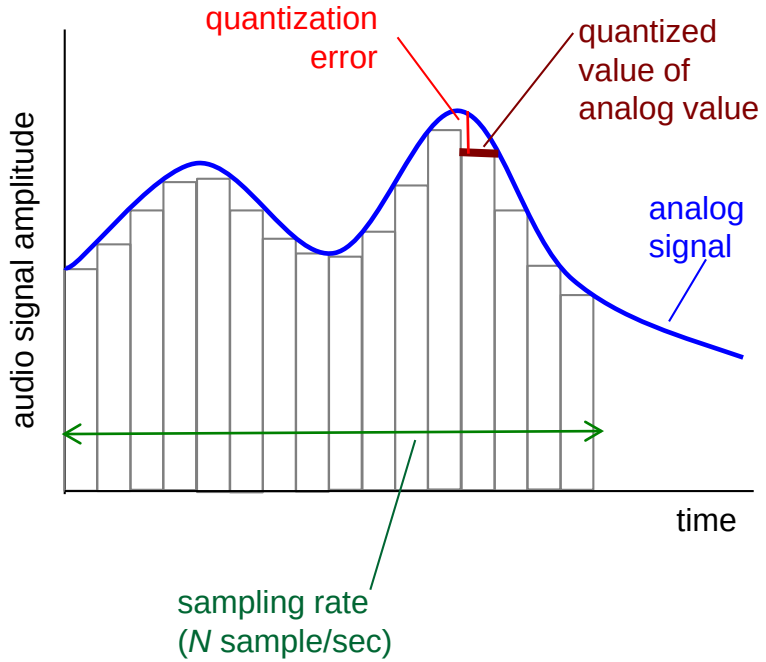


Computer Networking: A Top-Down Approach

8th edition n
Jim Kurose, Keith Ross
Pearson, 2020

Multimedia: audio

- analog audio signal sampled at constant rate
  - telephone: 8,000 samples/sec
  - CD music: 44,100 samples/sec
- each sample quantized, i.e., rounded
  - e.g., 2^8=256 possible quantized values
  - each quantized value represented by bits, e.g., 8 bits for 256 values



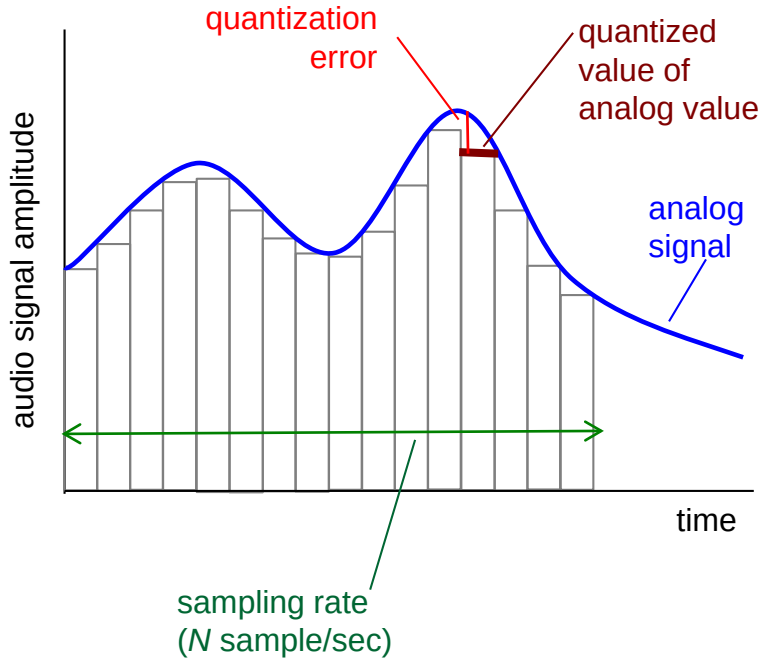
یکی از مدیاهایی که در اکثر اپلیکیشن های Multimedia لازم است از طریق شبکه منتقل بشه صدا است
این صدا وقتی که تولید میشه از طریق میکروفن حس میشه وتبدیل میشه به یک ولتاژ و این ولتاژ
به صورت یک سیگنال الکتریکی که دامنه اون بین دوتا مقدار حداقل و حداکثری داره تغییر می کنه متناسب با شدت و خصوصیات صدا
و این صدا یک سیگنال انالوگ است
برای انتقال صدا از طریق اینترنت ما سیگنال انالوگ رو باید تبدیل بکنیم به دیتا و دیتا رو باید منتقل بکنیم این
چطوری انجام میشه؟ روش تبدیل سیگنال انالوگ به سیگنال دیجیتال به این صورت است که ما سیگنال انالوگ رو در فواصل زمانی مشخصی مقدارشو اندازه گیری میکنیم و به هر مقدار میگیم یک نمونه یا یک سمپل و ما مقادیر ای سمپل هارو می تونیم استفاده بکنیم برای بازسازی سیگنال انالوگ
طبق قانون نایکوست اگر چنان چه تعداد نمونه ها به اندازه کافی باشه در واحد زمان یا عبارت دیگر فاصله های نمونه ها به اندازه کافی کم باشه ما می تونیم عینا سیگنال انالوگ رو بازسازی بکنیم
پس ما صدا داریم ابتدا با میکروفن صدا رو میگیریم و این سیگنال انالوگ رو به دست میاریم و بعد این سیگنال انالوگ رو نمونه برداری میکنیم و سیگنال گسسته به دست میاریم
اگر فرکانس نمونه ها به اندازه کافی باشه می تونیم سیگنال انالوگ رو بعدا بازسازی کنیم
طبق قانون نایکوست این نمونه برداری باید با فرکانس بزرگتر یا مساوی دو برابر عرض باند سیگنال (عرض باند سیگنال ما به بالاترین فرکانسی که سیگنال داره میگیم)
مقادیر این نمونه ها هنوز انالوگ است چون هر سمپل هر مقداری رو بین مینیمم و ماکزیمم سطوح ولتاژ سیگنال انالوگ می تونه داشته باشه و ما باید اونو تبدیل بکنیم به صفر و یک و برای این که اینو تبدیل بکنیم به صفر و یک اول باید مشخص بکنیم هر سمپل رو با چند بیت بیان کنیم؟ برای تلفن این 8 بیت است و ما 8 بیت رو فرض میکنیم استفاده میکنیم و این 8 بیت چند حالت مختلف می تونه داشته باشه؟ 2 به توان 8 ینی 256 حالت مختلف برای هر سمپل مقدارش می تونه تفکیک بشه
بین مینیمم و ماکزیمم چند مقدار مختلف داره؟ بی نهایت
و چند مقدار رو می تونیم با 8 بیت represented بکنیم؟ 256 حالت
پس مجبورم هر نمونه رو با یکی از اون حالت هایی که 256 سطح به ما میده quantized بکنیم پس فاصله بین کمترین ولتاژ وبیشترین ولتاژ رو به 256 سطح مختلف تقسیم می کنیم و هر سطح رو با یک حالتی از اون 8 بیت بیان می کنیم مثلا 0000 تا 1111 و به این ترتیب ما هر نمونه رو می بینیم مقدارش توی کدوم سطح قرار گرفته و اینو با مقدار اون 8 بیت سطح جایگزین میکنیم
پس ما یک مقدار انالوگ که توی اون زمان نمونه برداری شده با یک مقدار quantized جایگزین میکنیم به این ترتیب یک خطایی اینجا اتفاق می افته که یک مقداری دقت سیگنال کم میشه و البته با تعداد بیت بیشتر می تونیم تعداد سطوح رو بیشتر بکنیم و این خطا رو کمتر بکنیم و کیفیت صدا رو افزایش بدیم

Multimedia: audio

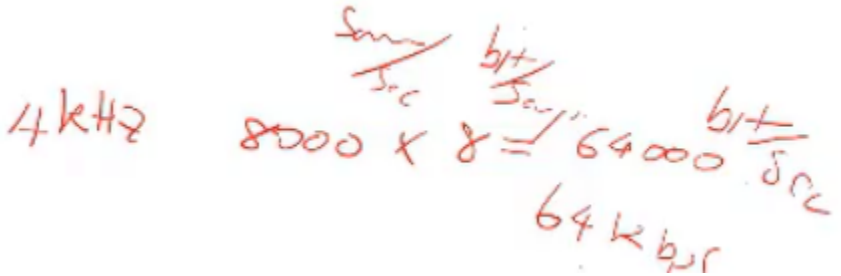
- example: 8,000 samples/sec, 256 quantized values: 64,000 bps
- receiver converts bits back to analog signal:
  - some quality reduction

example rates

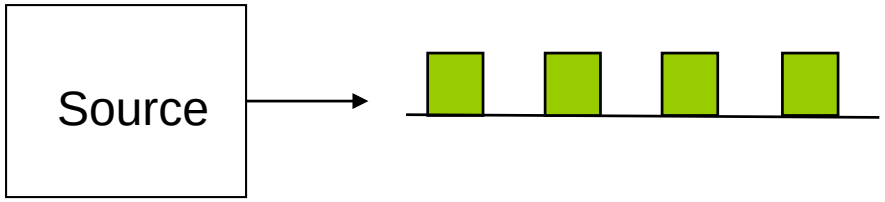
- CD: 1.411 Mbps
- MP3: 96, 128, 160 kbps
- Internet telephony: 5.3 kbps and up



به این ترتیب ما سیگنال تلفنی رو که عرض باند رو فرض میکنیم 4 کیلوهرتز است با 8000 تا نمونه در ثانیه نمونه برداری میکنیم و بعد هر نمونه رو با 8 بیت quantized میکنیم و به این ترتیب 64 هزار بیت به دست میاد که این ریت دیتایی است که از تبدیل انالوگ به دیجیتال این سیگنال تلفنی به دست میاد --> عکس
این مقادیر نمونه های متوالی رو ما می تونیم ذخیره کنیم یا منتقل بکنیم یا ... در نهایت می تونیم این هارو 8 بیت 8 بیت جدا کنیم و این سمپل هارو بازسازی بکنیم با یک مدار الکتریکی و بعد اگر این ها رو در طول زمان ایجاد بکنیم و از یک فیلتر پایین گذر عبور بدیم این سیگنال انالوگ بازسازی میشه
این ریتی که به دست میاریم برای موزیکی با کیفیت CD چیزی نزدیک حدود 1.41Mbps میشه و البته ما میتونیم همین رو کد بکنیم برای برای MP3 و چیزی نزدیک به 96, 128, 160kbps این رو کاهش بدیم --> علتش چی هست؟ علتش این است که ما در همین دیتای دیجیتال شده سیگنال انالوگ هنوز ریداندنسی داریم و می تونیم با فشرده سازی و روش های مختلف کدینگ این ریداندنسی رو کمترش بکنیم و به همین شکل برای سیگنال تلفنی ما حتی تا حدود 5Kbps این 64Kbps رو کاهش بدیم



Voice Packets



- Sampling, Encoding, Packetization
- Typical packetization time of 10-20ms per audio frame.
- Synchronous information stream
- Various encoding standards

Multimedia: video

- video: sequence of images displayed at constant rate
  - e.g., 24 images/sec
- digital image: array of pixels
  - each pixel represented by bits
- coding: use redundancy *within* and *between* images to decrease # bits used to encode image
  - spatial (within image)
  - temporal (from one image to next)

spatial coding example: instead of sending *N* values of same color (all purple), send only two values: color value (*purple*) *and number of repeated values* (N)

frame *i*

frame *i+1*

temporal coding example: instead of sending complete frame at i+1, send only differences from frame i

مدیای دیگری که داریم ویدیو است: ویدیو هم بتونه از طریق شبکه منتقل بشه باید تبدیل به دیتا بشه
چجوری تبدیل به دیتا میشه؟ ویدیو از فریم های متوالی از صحنه ای که فیلم برداری میشه پس ما یک دوربین فیلم برداری داریم صحنه ای رو فیلم برداری میکنه و این به صورت یکسری تصویر در فریم های متوالی از صحنه برداشته میشه --> پس ما یکسری تصویر داریم و عکس هایی که برداشته میشه توسط دوربین در فواصل زمانی یکسانی و به این ترتیب با یک ریتی عکس برداری میکنیم و هر عکس یک تصویر دوبعدی است و ما تک تک این هارو می تونیم تبدیل به دیجیتال بکنیم به این صورت که هر تصویر به صورت یکسری سطر و ستون اسکن میشه و محل تلاقی هر سطر و ستون رو میگیم پیکسل پس به این ترتیب متناسب با انداز فریم و تعداد سطرها و ستون ها که دقت و رزولویشن تصویر رو مشخص میکنه ما یک تعدادی پیکسل اینجا خواهیم داشت و هر پیکسل رو میتونیم در حالت خاکستری شدت تاریکی و روشن بودنش با یک تعداد بین مشخص بکنیم و در حالت رنگی رنگ این رو با ترکیب سه رنگ اصلی می تونیم بازسازی کنیم و کافیه که اندازه هر یک از سه رنگ رو با یک عدد باینری مشخص بکنیم ما اگر سه رنگ اصلی رو استفاده بکنیم و هر رنگ رو با 8 بیت کد بکنیم هر پیکسلمون 3*8=24 که میشه 24 بیت دیتا ایجاد خواهد کرد وبعد این 24 بیت رو ضرب میکنیم به تعداد سطر و ستون یا تعداد پیکسل های تصویر و به این ترتیب تعداد بیت هر فریم به دست میاد و بعد اینو ضربدر 24 میکنیم برای این مثالی که داشتیم و به این ترتیب بیت ریتمون به دست میاد
در مورد ویدیو هم ما این بیت ریتی که به دست میاریم به این ترتیب دوباره توش ریداندنسی توش زیاد است و می شه اینارو با فشرده سازی کد کرد و کدینگ میشه اینارو کاهش داد
این ریداندنسی هم در هر یک از تصویرها وجود داره و هم در تصویرهای فریم ها متوالی--> در هر فریم ما به صورت منطقه ای اطلاعاتی اضافی داریم که میتونیم اینارو حذف بکنیم مثلا برای این عکس قسمت اسمونش تقریبا همه نقاطش یک شکل و یک جور است پس به جای اینکه برای تک تک این پیکسل ها اون 24 بیت رو ذخیره بکنیم ما ما میایم در یکی از این ها اینو ذخیره میکنیم و بعد میایم تعداد اینارو مشخص می کنیم ---> به این ترتیب یک فشرده سازی در هر فریم انجام میگیره و حتی گفتیم بین این فریم و فریم بعدی هم می تونه ریداندنسی وجود داشته باشه معمولا اگر صحنه عوض نشه در یک صحنه بین دوتا فریم متوالی بسیار از نقاط وضعیتشون حفظ میشه و ممکنه تغیرات خیلی کمی در قسمت های متحرک به وجود بیاد با توجه به این مسئله ما به جای اینکه وضعیت تک تک پیکسل های فریم جدید رو بفرستیم کافیه که تفاوت اون رو با فریم قبلی بفرستیم مثلا با یک بیت مشخص بکنیم این پیکسل با فریم قبلی یکسان است یا نه و اگر یکسان نیست دیگه لازم نیست اطلاعاتشو اضافه بکنیم بلکه از همون اطلاعاتی که توی فریم قبلی بود این بازسازی خواهد شد --> به این ترتیب حجم اطلاعاتی که ایجاد میشه در فریم های بعدی خیلی می تونه کاهش پیدا بکنه و البته با توجه به اینکه در کدینگ ویدیو مشخص نیست که دو تا فریم متوالی ایا صحنشون یکی است یا عوض میشه و اگر همون صحنه است چقدر جابه جایی توش وجود داره یا ... به هرحال بیت ریتی که ایجاد میشه با این روش کدینگ و فشرده سازی متغییر با زمان خواهد بود

Multimedia: video

- CBR: (constant bit rate): video encoding rate fixed
- VBR: (variable bit rate): video encoding rate changes as amount of spatial, temporal coding changes
- examples:
  - MPEG 1 (CD-ROM) 1.5 Mbps
  - MPEG2 (DVD) 3-6 Mbps
  - MPEG4 (often used in Internet, < 1 Mbps)

spatial coding example: instead of sending *N* values of same color (all purple), send only two values: color value (*purple*) *and number of repeated values* (N)

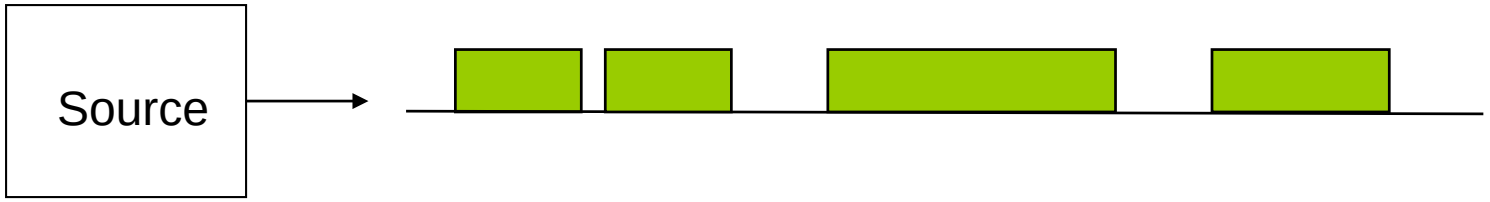
frame *i*

temporal coding example: instead of sending complete frame at i+1, send only differences from frame i

frame *i+1*

کدینگ های مختلفی برای ویدیو استفاده میشه:
CBR: بیت ریت ثابتی رو تولید میکنند --> جاهایی است که ریداندنسی و فشرده سازی کمتر استفاده میشه
اگر از ریداندنسی و فشرده سازی استفاده بکنیم کد حاصل VBR میشه --> ریت دیتای ایجاد شده کم و زیاد میشه

Video Packets



- Sampling, Encoding, Packetization
- Variable bit rate information stream
- Various encoding standards

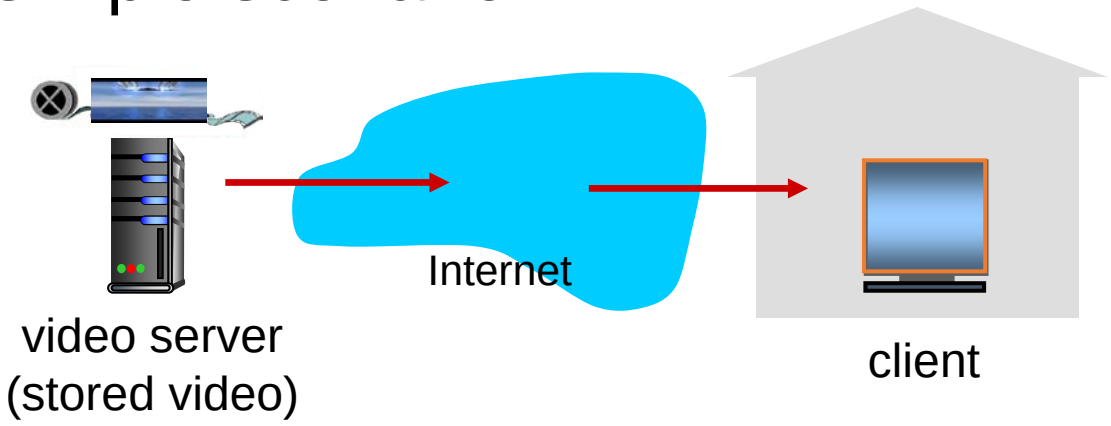
Multimedia networking: 3 application types

- *streaming, stored* audio, video
  - *streaming:* can begin playout before downloading entire file
  - *stored (at server):* can transmit faster than audio/video will be rendered (implies storing/buffering at client)
  - e.g., YouTube, Netflix, Hulu
- *conversational* voice/video over IP
  - interactive nature of human-to-human conversation limits delay tolerance
  - e.g., Skype
- *streaming live* audio, video
  - e.g., live sporting event (futbol)

اپلیکیشن های Multimedia روی اینترنت:
عمدتا مدیاشون ویدیو و audio یا ترکیبی از هر دو است
این اپلیکیشن ها در سه حالت مختلف استفاده میشن:
streaming live: یک دوربینی است که صدا رو میگیره و دیجیتال میکنه و صفر و یک هارو تولید میکنه --> تصویرمون مستقیما تبدیل میشه به دیتا و پکت و منتقل میشه روی شبکه
streaming, stored audio, video:در این حالت فیلممون قبلا برداشته شده و ذخیره شده مثلا توی هاردی است و از روی اون داره پخش میشه و این پخش میشه میتونه حالا همون streaming باشه ینی این دیتا پکت میشه و ارسال میشه به شبکه و از طریق شبکه منتقل میشه ب کامپیوتر یوزر و یوزر به جای اینکه اینو دانلود بکنه و پخش بکنه می تونه به همین صورت که درمیاد می تونه پخش بکنه و ببینه
در این حالت چون فایلمون ذخیره شده از قبل میشه اینو با ریت بالاتری فرستاد ینی با هر ریتی که لینکمون اجازه میده فایل رو می فرستیم و این مستلزم این است که در سمت گیرنده این ذخیره بشه به هرحال و بعد به صورت محلی (ینی لازم نیست تماما دانلود بشه و بعد پخش بشه هر بخشی که دریافت شد این میتونه شروع بکنه به پخش کردنش)
conversationa: در این حالت انتقال صدا یا تصویر یا هردوی اینا به صورت ریل تایم و متقابل است ینی اینجا حالت مکالمه ای وجود داره مثلا کنفرانس دوربین اینو پخش میکنه روی شبکه و اون طرف هم اینو می بینه و ...--> ماهیت مکالمه ریل تایم این است که تاخیر نباید وجود داشته باشه و تاخیر کیفیت مکالمه رو خراب میکنه --> پس در کل هم روی بحث ذخیره سازی ما نمی تونیم اینجا خیلی حساب بکنیم که این ذخیره بشه و بعد پخش بشه و هم روی بحث انتقال از طریق شبکه نمی تونیم تاخیرهای زیاد رو تحمل بکنیم

Streaming stored video

simple scenario:

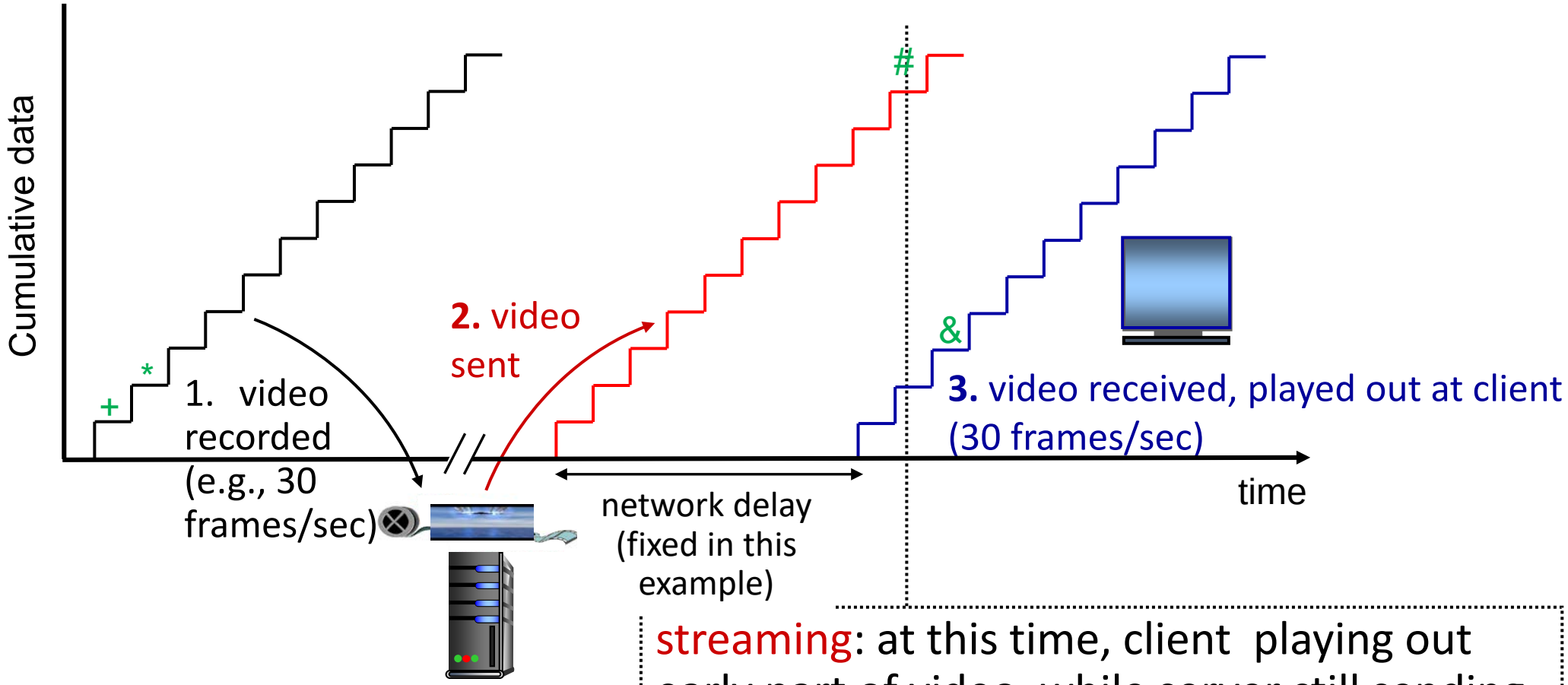


Main challenges:

- server-to-client bandwidth will *vary* over time, with changing network congestion levels (in house, access network, network core, video server)
- packet loss, delay due to congestion will delay playout, or result in poor video quality

---

Streaming stored video

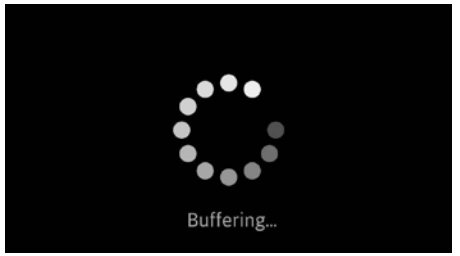


streaming: at this time, client playing out early part of video, while server still sending later part of video

اینجا فرض میکنیم ویدیو ذخیره شده روی یک سروری
و فرض میکنیم یک دوربین داره تصویر رو تصویر برداری می کنه و بعد اینو کد میکنه و در قالب بسته هایی ذخیره می کنه توی سرور مثلا 30 فریم در ثانیه تصویر برداری میشه و ذخیره میشه
در محورعموی --> دیتایی که از لحظه شروع تا حالا ایجاد شده
پس ما اینجا به اندازه یک پکت دیتا به وجود اوردیم و اینو ذخیره کردیم و چون محور عمودی جمع کل دیتایی که تا حالا ایجاد شده نشون میده و پس اگر در مرحله + دیتا ایجاد شده و در مرحله دوم
هم باز به اندازه + بعد توی منحنی جمع دوتارو خواهیم داشت * و بعد جمع این دوتا با فریم بعدی تا اخر..
--> پس این یک منحنی جمع شونده است که جمع کل دیتایی که تا حالا ایجاد شده رو نشون میده
توی مرحله بعدی میایم و اینو ارسال میکنیم توی شبکه --> پکت پکت این دیتارو می فرستیم توی اینترنت
3: این دیتایی رو که داریم سمت یوزر دریافت میکنیم و بعد پخش میکنیم
این دیتایی که اینجا سرور (2) داده به شبکه با یک تاخیری توسط کلاینت دریافت میشه و به این ترتیب پکت بعدی هم همینطورپشت سر اون میاد پس توی (3) ما داریم به صورت جمع شونده جمع کل پکت هایی رو که تا حالا دریافت کردیم رو نشون میدیم توی منحنی مثلا توی لحظه & ما سه تا پکت دریافت کردیم
ما اینجا این تاخیر رو که تاخیر شبکه هست رو توی این مثال اینو ثابت در نظر گرفتیم --> network delay fixed و این شرایط ایده الی است و ما به محض اینکه اولین پکت رو دریافت کردیم می تونیم شروع بکنیم به پخش کردن و اگر با همون ریتی که دوربین فیلم برداری کرده و با همون ریتی که داره ارسال میشه پخش بکنیم ما داریم تصویر اصلی رو سمت یوزر بازسازی
میکنیم
در streaming این فریم هایی که اینجا فرستادیم توی این لحظه شروع می کنن به پخش شدن منتها فرستنده به ارسال فریم های بعدی ادامه داده و توی همون لحظه هم هنوز داره فریم های بعدی رو ارسال میکنه مثلا توی زمان # کلاینت داره فریم هایی که قبلا ارسال شده اند و این داره دریافت می کنه رو پخش میکنه و همزمان سرور داره فریم های بعدی رو ارسال میکنه
نکته: اینا به هر حال می رسن و پخش میشن

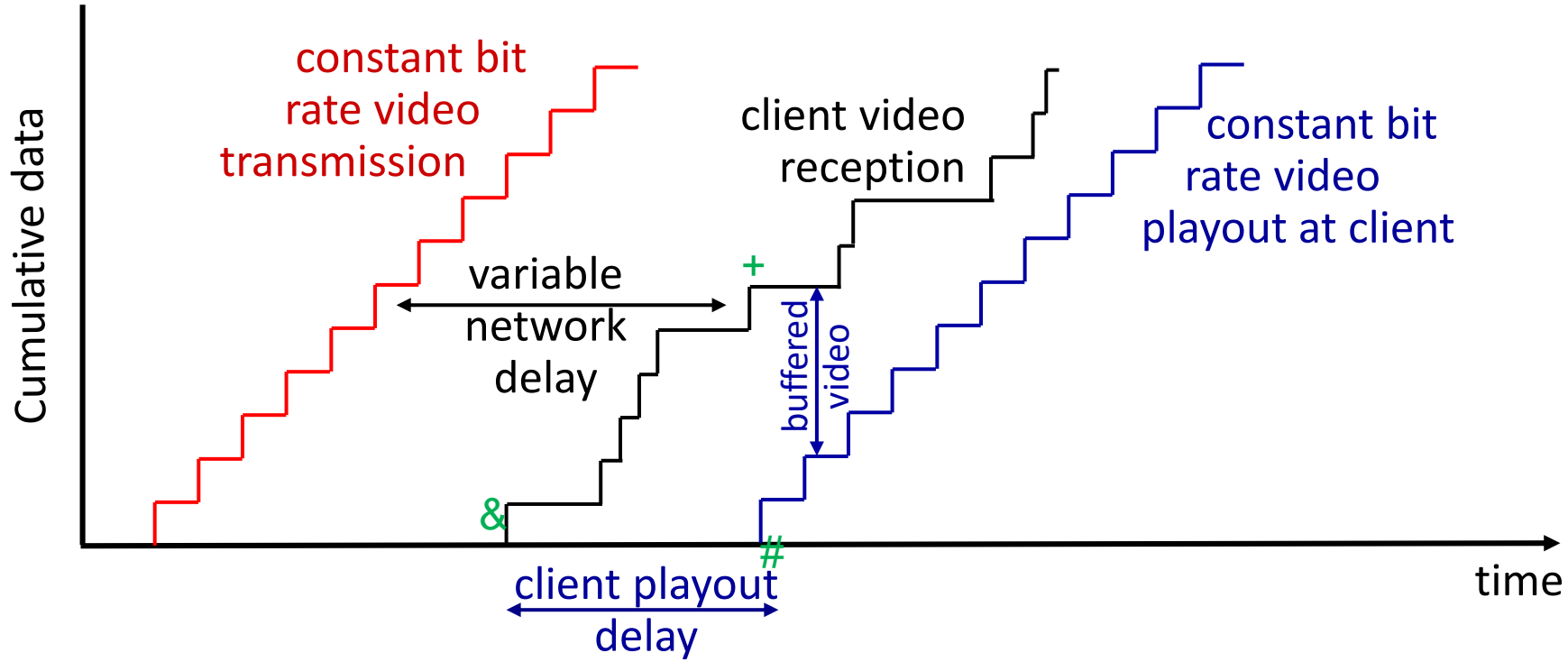
Streaming stored video: challenges

- continuous playout constraint: during client video playout, playout timing must match original timing
  - … but network delays are variable (jitter), so will need client-side buffer to match continuous playout constraint
- other challenges:
  - client interactivity: pause, fast-forward, rewind, jump through video
  - video packets may be lost, retransmitted



مشکل کار کجاست؟
اگر کلاینت ما شروع کرد به پخش کردن ویدیویی که از شبکه میگیره و این دیگه باید پخشش ادامه پیدا کنه پس وقتی شروع به پخش کرد باید ادامه پیدا کنه و این میتونه توی شبکه نقض بشه چون تاخیرهای شبکه تاخیرهای ثابت نیست --> ما jitter داریم ینی تاخیرها برای پکت های متوالی متغییر است و می تونه متفاوت باشه پس اگر ما براساس اون تاخیری که فرض کردیم در مثال قبلی و شروع بکنیم به پخش کردن و انتظار داشته باشیم بقیه بسته ها هم با همان تاخیر بلافاصله خواهند رسید ممکنه این اتفاق نیوفته و ممکنه بسته بعدی تاخیرش بیشتر بشه
در یک حالت حاد ممکنه این تاخیرش دوبرابر بشه
این یک چالشی است برای پخش زنده ویدیو --> برای اینکه یک مشکل پیش نیاد لازمش این است که ما اینو بافر بکنیم سمت گیرنده و پخشمون از بافر باشه نه از دیتایی که از شبکه مستقیم داریم میگیریم
پس این بافر باشه که یک ذخیره ای باشه که اگر یک بسته ای هم دیر اومد پخش قطع نشه
چالش های دیگه هم هست:
مثلا اینکه ما کلاینتمون بخواد pause کنه یا فیلم رو برگردونه عقب یا ... ایا این قابلیت ها رو می تونه داشته باشه یا نه
بحث لاس در شبکه --> اگر لاس اتفاق بیوفته چی کار بکنیم

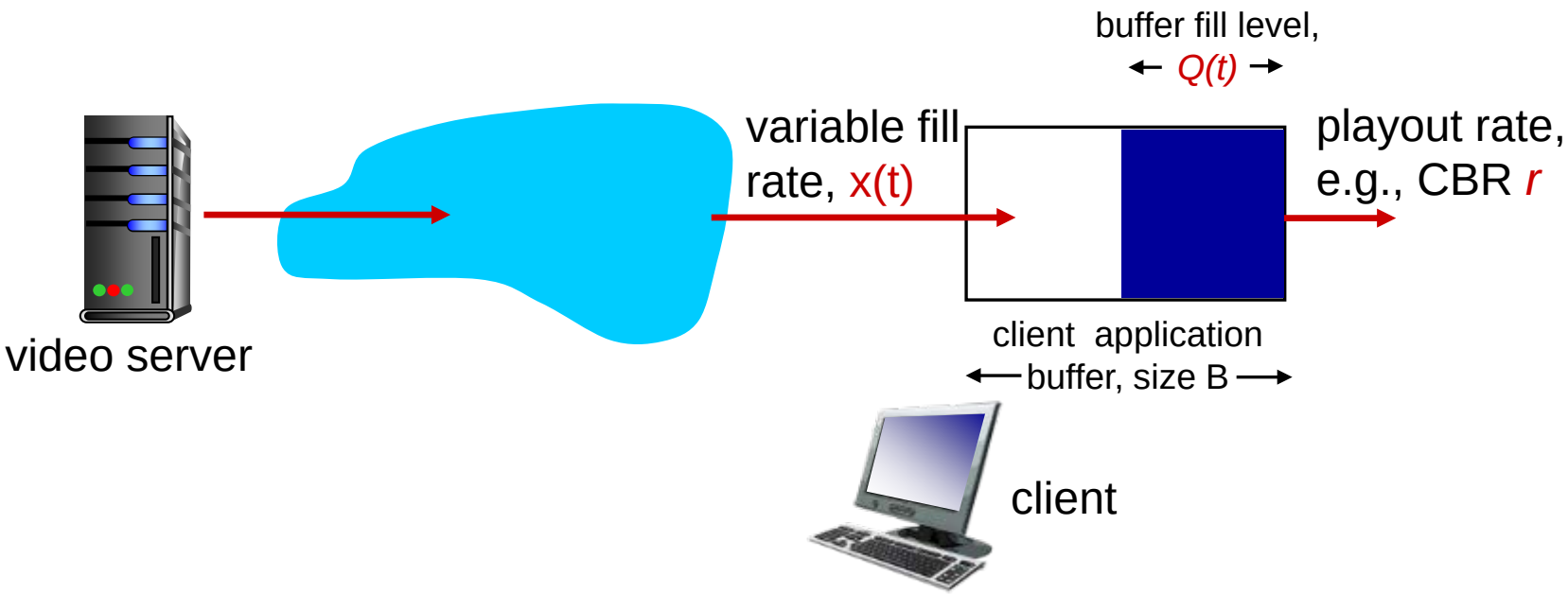
Streaming stored video: playout buffering



- *client-side buffering and playout delay:* compensate for network-added delay, delay jitter

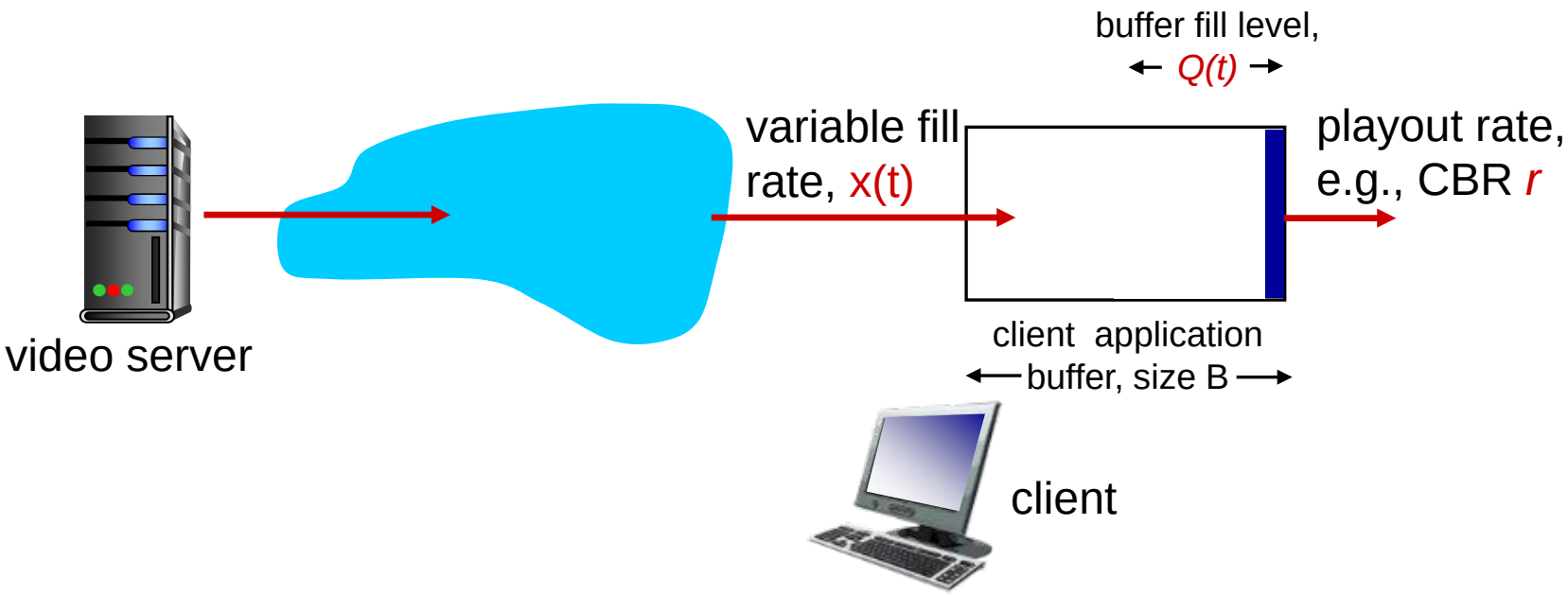
در اینجا ما یک حالت واقعی تر از مثالی که قبلا گفتیم رو دوباره میگیم:
قرمزه: ویدیویی که سرور داره روی شبکه می فرسته --> فرض می کنیم ویدیو کدینگش CBR است در نتیجه در فواصل زمانی یکسانی ما به اندازه مشخصی داریم دیتا می فرستیم روی شبکه و با یک تاخیری این میاد و در کلاینت دریافت میشه منتها این تاخیر واقعیتش این است که ثابت نیست و برای بسته های متوالی ممکنه متفاوت باشه در نتیجه ما اون بسته های متوالی که در سمت کلاینت دریافت میکنیم فاصله زمانیشون متفاوت است
پس اگر اینجا توی همین اولین فریم & اگر بلافاصله اینو پخش بکنیم انتظار داشته باشیم که به اندازه یک فریم تایم بعد بسته بعدی برسه این نمی رسه و دیرتر می رسه و پخشمون همین جا متوقف میشه و در تمام مواردی که تاخیر بیشتر شده ما این ریسک رو داریم که پخشمون مختل بشه
ابی: برای جبران این کاری که میشه کرد این است که درگیرنده بافر بکنیم و بلافاصله پخش نکنیم و این بافر معادل یک تاخیر اضافه تری است که ما داریم به هر پکت اضافه میکنیم به این ترتیب این پکت هایی که دریافت می شه رو بافر میکنیم تا لحظه # و لحظه # شروع میکنیم به پخش کردن و این لحظه ای است که تا + دریافت شده و از این جا # که شروع می کنیم به پخش کردن زمان بندی رو براساس اون ریتی که برای پخش ویدیو لازم داریم رو خودمون تنظیم می کنیم و شبکه نیست که برامون بخواد تعیین بکنه و با همین زمان بندی اینارو از بافر برمیداریم و پخش میکنیم و اگر یک پکت دیر برسه (قسمت مشکی) ما توی بافرمون باز بسته داریم که اینو بفرستیم و ما اینو حس نمی کنیم که دیر رسیده
جمع تاخیر شبکه و تاخیر انتظار توی بافر تا پخش بشه به نوعی برای همه بسته ها یکسان میشه با این ترتیب (قسمت ابی) و اون چیزی که از بافر پخش میشه همون حالت ایده الی است که قبلا گفتیم حالا ایا واقعا میشه همچین کاری کردی؟ در هر لحظه اگر فاصله عمودی بین منحنی مشکی و ابی رو نگا بکنیم این به ما نشون میده تا این لحظه چقدر دیتا اومده توی بافر و توی بافر است
و این buffered videoمقداری است که برای اینکه این حالت ایده ال حفظ بشه نباید صفر بشه
پس ایا ما می تونیم اینو تضمین بدیم ک این بافر هیچ وقت صفر نشه؟ کجا این بافر خالی میشه؟
وقتی که بسته بعدی خیلی تاخیرش زیاد بشه --> پس از کجا بفهمیم اون لحظه ای که داریم شروع می کنیم لحظه مناسبی است؟

Client-side buffering, playout



سناریو:
سرور پخش میکنه و از طریق اینترنت این ها رو دریافت میکنیم و می ریزیم توی بافر و تا یه حدی می ذاریم بافر پر بشه و وقتی که به اون حد رسید شروع می کنیم به پخش کردن و پخش با ریت لازم ویدیو است و کاری به ریتی که از شبکه دریافت میکنیم نداریم
این نوسانات تاخیری که توی شبکه اتفاق میافته این بافر می گیره --> اندازه بافر در طول زمان ممکنه کم و زیاد بشه

Client-side buffering, playout

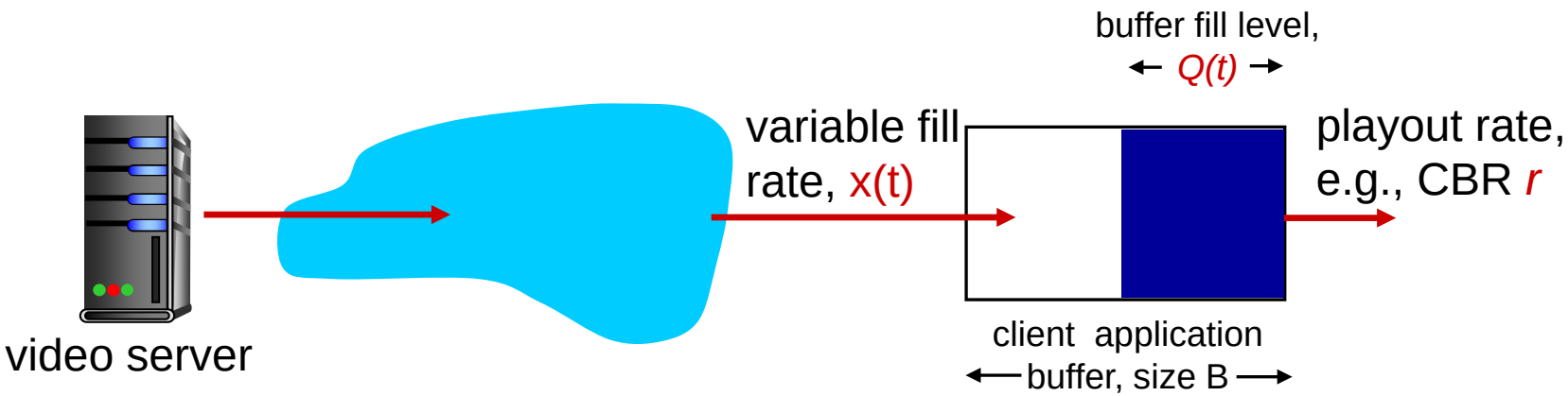


1. Initial fill of buffer until playout begins at t_p
2. playout begins at t_p,
3. buffer fill level varies over time as fill rate x(t) varies and playout rate r is constant

این حالتی است که تاخیر برای بسته های بعدی بیشتر باشه و خیلی از بافر خالی شده و اگر اون تاخیر اونقدر زیاد بشه که این بافر خالی بشه پخش قطع میشه

اون مقداری از بافر که اجازه میدیم پر بشه و بعد پخش رو شروع میکنیم با tp نشون میدیم به این ترتیب بافر تا tp پر میشه و بعد پخش شروع میشه و بعد اندازه بافر در زمان تغییر می کنه که با x(t) اینو نشون میدیم ضمن اینکه پخشمون با ریت ثابت r در جریان است

Client-side buffering, playout



playout buffering: average fill rate (x), playout rate (r):

- *initial playout delay tradeoff:* buffer starvation less likely with larger delay, but larger delay until user begins watching

دوتا نکته:
مقداری که صبر میکنیم تا بافر بشه و بعد شروع میکنیم به پخش کردن --> اینجا هر چی بیشتر صبر بکنیم و هرچی اون مقداری که در ابتدا جمع میشه بیشتر باشه و از مقدار بیشتری شروع بکنیم پخش کردن احتمال اینکه بخاطر تغییرات تاخیر یا تغییرات ریت متوسط لحظه ای ترافیکی که وارد بافر میشه این بافرمون خالی باشه کمترمیشه و این مستلزم این است که در ابتدا بیشتر صبر بکنیم پس تاخیر بیشتر در ابتدا و کمتر شدن ریسک تاخیر در وسط کار اینا با هم مترادف هستن
و اگر بخوایم زودتر پخش شروع بشه در این صورت بافر کمتر پر شده و ممکنه به حالی برسیم که بافر خالی بشه
به غیر از اینا باید ریت پخش بافرهم باید خیلی دقیق باشه نسبت به اون ریتی که فیلم اصلیمون برداشته شده و دیتا شده و اگر چنان چه این ریت از متوسط ریتی که دیتا داره وارد بافر میشه بیشتر باشه به گرسنگی می رسیم ینی بافر خالی میشه و اگر این کمتر باشه از اون ریتی که داره دیتا میاد بافرمون ممکنه سرریز بکنه

playout buffering: average fill rate (x), playout rate (r):

- $\overline{x} < r$: buffer eventually empties (causing freezing of video playout until buffer again fills)
- $\overline{x} > r$: buffer will not empty, provided initial playout delay is large enough to absorb variability in x(t)

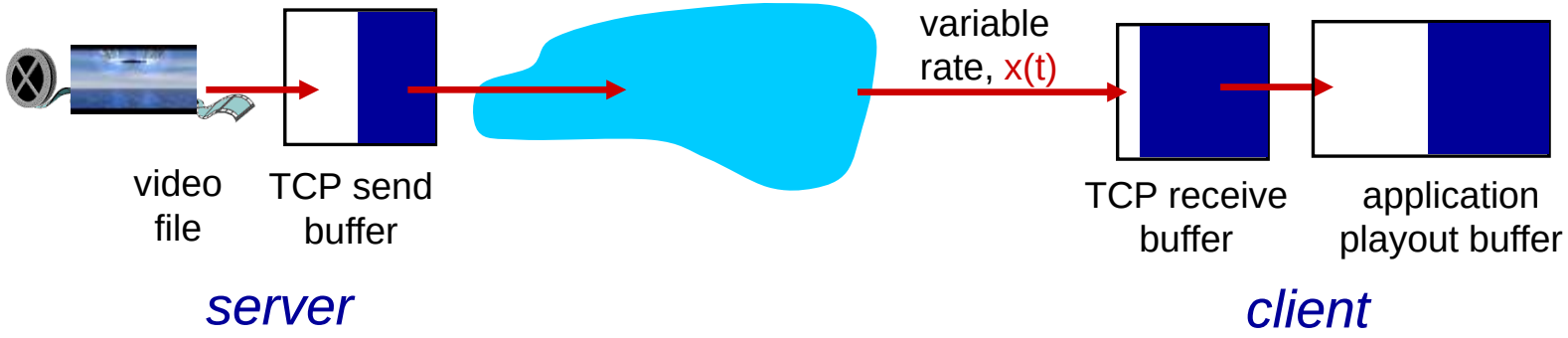
Streaming multimedia: UDP

- server sends at rate appropriate for client
  - often: send rate = encoding rate = constant rate
  - transmission rate can be oblivious to congestion levels
- short playout delay (2-5 seconds) to remove network jitter
- error recovery: application-level, time permitting
- RTP [RFC 2326]: multimedia payload types
- UDP may *not* go through firewalls

Streaming multimedia روی اینترنت به چه شکلی می تونه انجام بگیره؟
یه روش این است که از پکت های udp استفاده بکنیم
استفاده از udp برای مالتی مدیا این حسن رو داره که به صورت ریل تایم پخش می تونه انجام بشه
چون که udp بحث های مثل ارور کنترل و retransmission رو و congestion کنترل و اینهارو نداره و بسته ها به صورت دیتاگرام داده میشن به شبکه و از این طرف دریافت میشن ودر نتیجه میشه با هر ریت دلخواهی که لازم است سرور اطلاعات رو برای کلاینت بفرسته
معمولا این send rate با ریتی که از encoding ناشی میشه برابر است
این رو میشه با وضعیت congestion تنظیم کرد و کدینگ مورد استفاده و ریت حاصل رو کمتر یا بیشترش کرد
برای لاس و ارور توی شبکه چون اینجا udp استفاده میکنیم روی retransmission نمی تونیم حساب بکنیم در نتیجه اپلیکیشن خودش باید یه فکری برای بحث هایی مثل ارور و پکت لاس بکنه
از پروتکل RTP برای انتقال مالتی مدیا روی udp استفاده میشه
udp معمولا بخاطر بحث های دی داس اتک ممکنه فیلتر بکنن یا ریتش رو محدود بکنن در نتیجه ممکنه از فایروال ها عبور نکنه

Streaming multimedia: HTTP

- multimedia file retrieved via HTTP GET
- send at maximum possible rate under TCP



- fill rate fluctuates due to TCP congestion control, retransmissions (in-order delivery)
- larger playout delay: smooth TCP delivery rate
- HTTP/TCP passes more easily through firewalls

برای همین روش دیگه ای که برای مالتی مدیا توی اینترنت استفاده میشه استفاده از http و خود http روی tcp قرار داره پس از tcp استفاده میشه
در این روش از http get استفاده میشه برای گرفتن فایل مالتی مدیا
بسته به اینکه Tcp چه ریتی رو اجازه می ده با ماکزیمم ریتی که tcp اجازه میده این بسته ها منتقل خواهند شد
اینجا چون داریم از tcp استفاده میکنیم بافر Tcp رو داریم که اطلاعات رو جمع اوری میکنه و مرتب میکنه و لاس رو جبران میکنه و در نهایت از اینجا دیتا داده میشه به همون playout بافری که خود اپلیکیشن داره
اینجا میزان دیتای وارد شده به بافر که تحت تاثیر congestion شبکه خواهد بود و retransmissions هایی که لازم خواهد شد --> اینجا هرچی playout delay بیشتر باشه ینی هرچی بیشتر بافر بکنیم و پخش بکنیم پخش smooth تری خواهیم داشت چون احتمال اینکه بافر خالی بشه کمتر خواهد بود
http روی tcp این مزیت رو داره که مشکل زیادی با فایروال ها نداره

Transport layer: roadmap

- Transport-layer services
- Multiplexing and demultiplexing
- Connectionless transport: UDP
- Principles of reliable data transfer
- Connection-oriented transport: TCP
- Principles of congestion control
- TCP congestion control
- Evolution of transport-layer functionality

Evolving transport-layer functionality

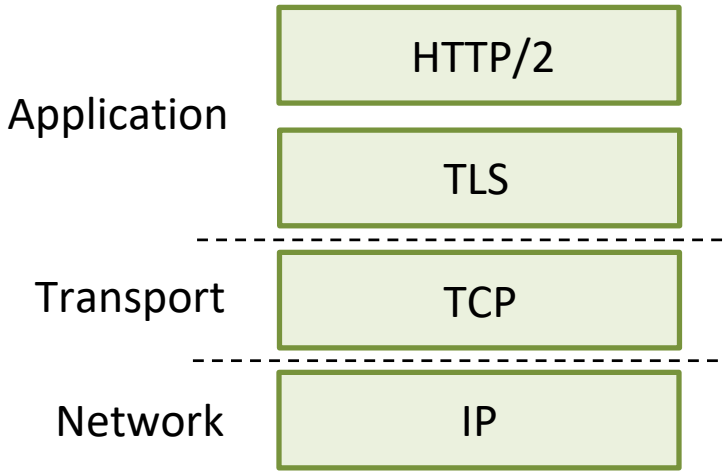
- TCP, UDP: principal transport protocols for 40 years
- different “flavors” of TCP developed, for specific scenarios:

Scenario	Challenges
Long, fat pipes (large data transfers)	Many packets “in flight”; loss shuts down pipeline
Wireless networks	Loss due to noisy wireless links, mobility; TCP treat this as congestion loss
Long-delay links	Extremely long RTTs
Data center networks	Latency sensitive
Background traffic flows	Low priority, “background” TCP flows

- moving transport–layer functions to application layer, on top of UDP
  - HTTP/3: QUIC

QUIC: Quick UDP Internet Connections

- application-layer protocol, on top of UDP
  - increase performance of HTTP
  - deployed on many Google servers, apps (Chrome, mobile YouTube app)



HTTP/2 over TCP

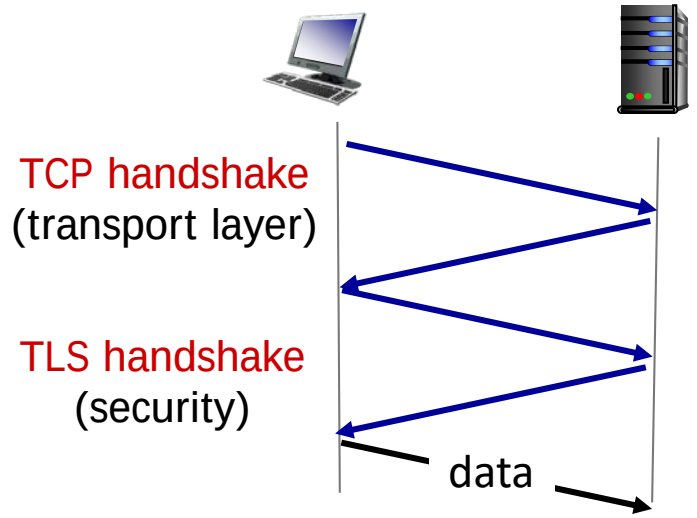
QUIC: Quick UDP Internet Connections

adopts approaches we've studied in this chapter for connection establishment, error control, congestion control

- **error and congestion control:** "Readers familiar with TCP's loss detection and congestion control will find algorithms here that parallel well-known TCP ones." [from QUIC specification]
- **connection establishment:** reliability, congestion control, authentication, encryption, state established in one RTT

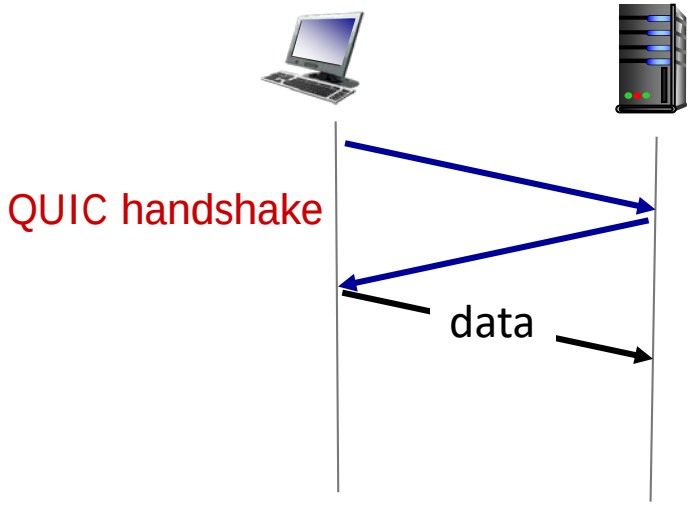
- multiple application-level "streams" multiplexed over single QUIC connection
  - separate reliable data transfer, security
  - common congestion control

QUIC: Connection establishment



TCP (reliability, congestion control state) + TLS (authentication, crypto state)

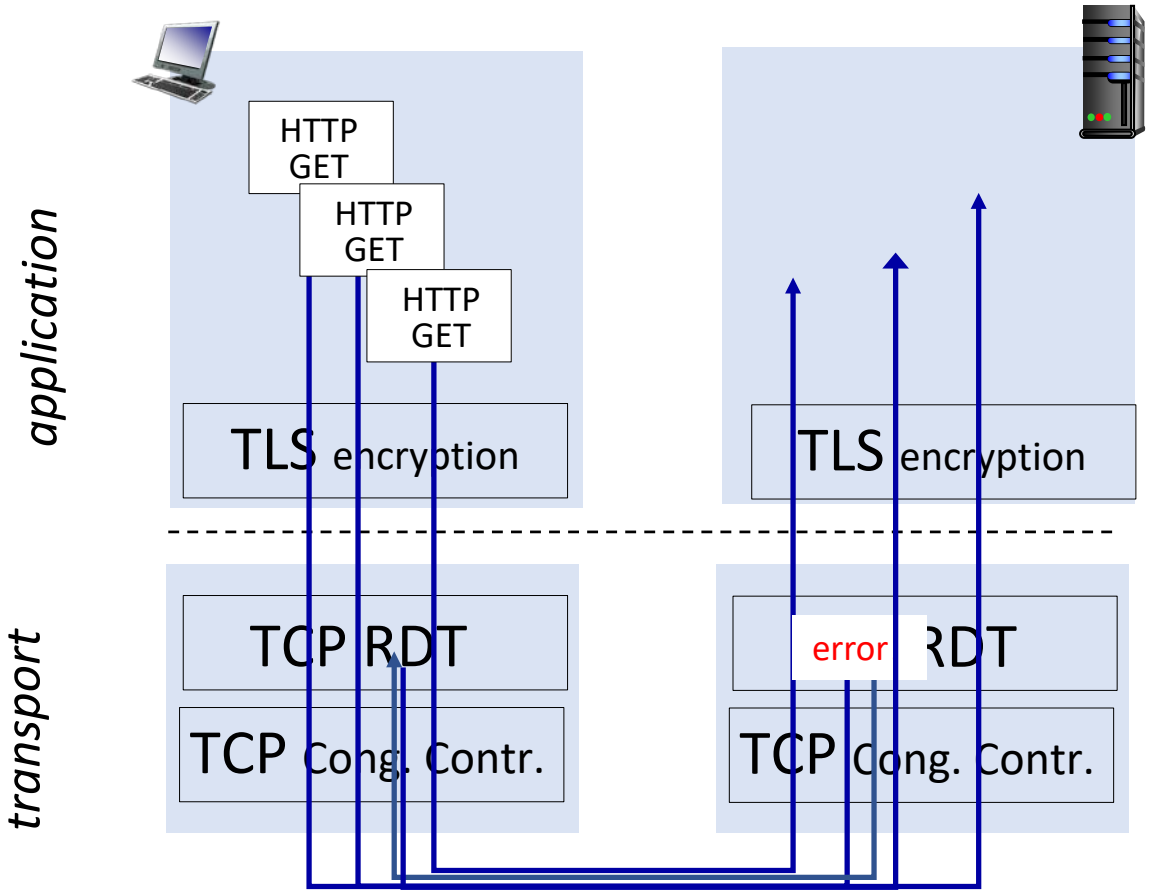
- 2 serial handshakes



QUIC: reliability, congestion control, authentication, crypto state

- 1 handshake

QUIC: streams: parallelism, no HOL blocking



(a) HTTP 1.1