دانشکده‌ی مهندسی کامپیوتر

شبکه‌های کامپیوتری

نیم‌سال دوم ۰۴-۱۴۰۳

استاد: امیرمهدی صادق‌زاده

تمرین اول

پاسخ‌دهنده: معین آعلی - ۴۰۱۱۰۵۵۶۱

فهرست مسائل

مسئله ۱ . . . ۲
الف . . . ۲
ب . . . ۲
ج . . . ۳
د . . . ۴
مسئله ۲ . . . ۵
الف . . . ۵
ب . . . ۵
ج . . . ۵
د . . . ۶
ه . . . ۶
و . . . ۷
مسئله ۳ . . . ۸
الف . . . ۸
ب . . . ۸
ج . . . ۸
مسئله ۴ . . . ۹
الف . . . ۹
ب . . . ۹
مسئله ۵ . . . ۱۰
الف . . . ۱۰
ب . . . ۱۰
ج . . . ۱۰
د . . . ۱۰
مسئله ۶ . . . ۱۱
الف . . . ۱۱
ب . . . ۱۱
ج . . . ۱۲
مسئله ۷ . . . ۱۳
الف . . . ۱۳
ب . . . ۱۳
ج . . . ۱۳
مسئله ۸ . . . ۱۴
توضیح کد . . . ۱۴
نتایج اجرای کد . . . ۱۷

پاسخ مسئله‌ی ۱.

الف

معیار / فناوری	DSL	کابل	FTTH	Wi-Fi	۵G / ۴G
پهنای باند	تا ۱۰۰ Mbps	تا ۱ Gbps	تا ۱۰ +Gbps	تا ۱ Gbps	۴G: ۱ Gbps ۵G: ۱۰ Gbps
تأخیر	۵۰–۲۰ ms	۳۰–۱۰ ms	۱۰–۱ ms	۳۰–۵ ms	۴G : ۵۰–۳۰ ms ۵G : ۱۰–۱ ms
مقیاس‌پذیری	محدود	متوسط	بسیار بالا	محلی	بالا
هزینه	کم	متوسط	بالا	کم	بالا
پوشش	گسترده	شهری خوب	محدود به مناطق مجهز	محیط داخلی	بسیار گسترده

جدول ۱: مقایسه‌ی فناوری‌های شبکه دسترسی از جنبه‌های مختلف

ب

برای هر یک از این مناطق جغرایی یک پیشنهاد دارم که در ادامه شرح می‌دهم.

۱. مناطق شهری با تراکم بالا

پیشنهاد: FTTH

دلایل انتخاب:

- کاربران پرمصرف (استریم، بازی، دورکاری) به سرعت بالا و تأخیر کم نیاز دارند.
- تراکم بالا، هزینه سرانه‌ی نصب فیبر را کاهش می‌دهد و توجیه‌پذیر می‌کند.
- مقیاس‌پذیری بالای FTTH امکان رشد آینده را فراهم می‌کند.

۲. مناطق روستایی

پیشنهاد: کابل ترکیبی

دلایل انتخاب:

- هزینه‌ی پایین‌تر نسبت به FTTH در مناطقی با تراکم کمتر.
- امکان استفاده از زیرساخت‌های موجود (در صورت وجود) مانند کابل تلویزیون و تلفن.
- عملکرد قابل‌قبول برای کاربردهایی مثل ویدئو کنفرانس و استریم با کیفیت متوسط تا بالا.

۳. مناطق دورافتاده

پیشنهاد: اتصال بی‌سیم ۵G و ۴G

دلایل پیشنهاد:

- نصب فیبر یا کابل در این مناطق هزینه‌بر و غیرعملی است.

- دکل‌های سلولی می‌توانند مساحت وسیعی را پوشش دهند.
- با نصب تجهیزات قوی‌تر کاربران می‌توانند به خدمات با کیفیت مناسب دست یابند.

ج

اجزای کلیدی معماری

DSLAM دستگاهی است که سیگنال‌های دیجیتال دریافتی از کاربران DSL را جمع‌آوری و به یک اتصال پرسرعت به سمت شبکه مرکزی (core-network) تبدیل می‌کند. معمولاً در مرکز مخابراتی یا در کابین‌های خیابانی نزدیک به کاربران نصب می‌شود. وظیفه آن تبدیل خطوط تلفن آنالوگ به سیگنال‌های دیجیتال و مدیریت ترافیک چندین کاربر DSL است.

CMTS در سمت ارائه‌دهنده خدمات قرار دارد و داده‌ها را بین اینترنت و مودم‌های کابلی در منازل کاربران رد و بدل می‌کند. در مرکز عملیات شبکه (Headend) نصب میشود. وظیفه آن مدیریت اتصالات کاربران، اختصاص پهنای باند، کنترل ترافیک اینترنت کابلی است.

OLT نقطه مرکزی شبکه فیبر نوری است که ارتباط بین اینترنت و چندین ONU/ONT (دستگاه‌های کاربران) را مدیریت می‌کند. در مرکز سرویس‌دهی اپراتور نصب میشود. وظیفه آن تبدیل سیگنال‌های نوری به الکتریکی و بالعکس، و ارسال/دریافت داده‌ها از کاربران از طریق شبکه GPON یا EPON است.

اثرات خرابی

اگر DSLAM خراب شود:

- همه کاربران DSL متصل به آن دستگاه اتصال اینترنت خود را از دست می‌دهند.
- ممکن است خطوط تلفن ثابت نیز دچار اختلال شوند (در صورت استفاده‌ی مشترک از تجهیزات).
- اختلال فقط روی محدوده‌ای مشخص از کاربران تأثیر می‌گذارد (وابسته به پوشش آن DSLAM).

اگر CMTS یا OLT خراب شود:

- CMTS : قطع ارتباط برای همه کاربران کابل متصل به آن بخش.
- OLT : از کار افتادن کل زیرشبکه فیبر که به آن OLT وصل است (صدها کاربر ممکن است تحت تأثیر قرار بگیرند).

جلوگیری از خرابی

۱. طراحی با Redundancy :

- استفاده از DSLAM، CMTS یا OLT اضافی (Backup) برای پشتیبانی در صورت خرابی دستگاه اصلی.
- ایجاد مسیرهای متنوع برای اتصال کاربران به بیش از یک تجهیز اصلی.

۲. معماری توزیع‌شده

- استفاده از چند DSLAM یا OLT با پوشش‌های کوچک‌تر به‌جای یک دستگاه مرکزی بزرگ.
- کاهش تعداد کاربرانی که در صورت خرابی یک تجهیز دچار اختلال می‌شوند.

۳. دسترسی چندگانه:

- در برخی مناطق، ارائه دسترسی ترکیبی (مثلاً هم Wi-Fi و هم DSL) برای سوئیچ خودکار به مسیر پشتیبان در صورت قطعی.

د

مناطق پرتراکم – شبکه سیمی (FTTH/LAN) + Wi-Fi

برای ساختمان‌ها از FFTB یا Ethernet استفاده شود و در فضاهای داخلی پرتردد از Wi-Fi .

توجیه:

- فیبر/سیمی: ارائه‌ی سرعت بسیار بالا برای استریم، آپلود فایل‌های حجیم، و کلاس‌های آنلاین.
- Wi-Fi : اتصال موبایل و لپ‌تاپ دانشجویان با سهولت و پوشش بالا در فضاهای عمومی مثل کتابخانه یا سلف.

مناطق کم‌تراکم – اتصال بی‌سیم (Outdoor-Wi-Fi / FWA)

- سیم‌کشی به این مناطق (مثل زمین‌های ورزشی یا مراکز تحقیقاتی دورافتاده) پرهزینه و غیرعملی است.
- با نصب آنتن‌های بی‌سیم نقطه به نقطه یا سلولی، دسترسی به اینترنت با تأخیر کم و هزینه پایین حاصل می‌شود.

ارتباط بین تجهیزات اصلی

- فیبر نوری Backbone برای اتصال همه بخش‌ها به دیتاسنتر یا شبکه اصلی دانشگاه.
- سوئیچ‌های توزیع و هاب‌های محلی برای مدیریت ترافیک.

پاسخ مسئله‌ی ۲.

الف

.۱

میزان تاخیر از مبدا تا اولین سوییچ:

$delay = \frac{L}{R} = \frac{۱۰^۶}{۵\times۱۰^۶} = ۰/۲s$

.۲

هر سوییچ بسته را به طور کامل دریافت و سپس ارسال میکند، پس تا مقصد نهایی باید ۳ مرتبه تاخیر مرتبه قبل را حساب کنیم:

$delay = ۳ \times \frac{L}{R} = \frac{۱۰^۶}{۵\times۱۰^۶} = ۰/۶s$

.۳

اگر N ایستگاه در مسیر داشته باشیم، پس N + ۱ مسیر داریم، پس در نهایت هم N + ۱ تاخیر داریم:

$delay = (N+۱) \times \frac{L}{R} = \frac{۱۰^۶}{۵\times۱۰^۶} = (N+۱) \times ۰/۲s$

ب

.۱

میزان تاخیر اولین بسته به این صورت است:

$delay_{packet_۱} = \frac{L}{R} = \frac{۱۰^۴}{۵\times۱۰^۶} = ۲ms$

.۲

همزمان با دریافت کامل بسته اول، بسته دوم ارسال میشود که آن هم ۲ میلی ثانیه بعد به اولین سوییچ میرسد. پس:

$delay_{packet_۲} = ۲ \times delay_{packet_۱} = ۴ms$

.۳

به تعمیم بخش قبل، زمان رسیدن بسته K ام به اولین سوییچ برابر است با:

$delay_{packet_K} = K \times delay_{packet_۱} = ۲K\ ms$

ج

.۱

میدانیم پکت آخر در $t = ۲۰۰ms$ در اولین سوییچ است، پس $۴ms$ بعد در زمان $t = ۲۰۴ms$ به مقصد میرسد. همچنین بدون شکستن پیام، نیاز به $۶۰۰ms$ داشتیم.

این اتفاق به این دلیل است که بخشی از انتقال پیام ها به صورت موازی جلو میرود. هنگامی که پکت اول در حال انتقال از سوییچ دوم به سوم است، پکت دوم از سوییچ اول به دوم منتقل میشود و ...
واضحا هرچه تعداد سوییچ های در میان راه بیشتر باشد، این اختلاف زمانی بیشتر خواهد شد.

.۲

در صورتی که سایز بسته ها متفاوت باشد، زمان ارسال آن ها هم متفاوت خواهد بود و بر روی زمان کل تاخیر اورهد دارد. چون بسته های بزرگتر زمان بیشتری صرف ارسال میکنند و باعث کم شدن موازی سازی بسته های کوچک تر هم میشوند. نیاز به صف و تاخیر صف هم خواهیم داشت در این صورت. اما باز هم این حالت بهتر از حالت اول است که کل پیغام را یکجا ارسال کردیم!

د

.۱

در این حالت یکی از مسیرها زمان بیشتری از s۰/۲ نیاز دارد، پس:
$delay = \frac{L}{R_۱} + ۲ \times \frac{L}{R_۲} = ۰/۵ + ۲ \times ۰/۲ = ۰/۹s = ۹۰۰ms$
در حالت تقسیم بسته هم نیاز به ms۵۱۰ زمان داریم.
$delay = \frac{۱۰۰۰۰۰}{۲\times۱۰^۶} = ۰/۰۰۵ = ۵ms \rightarrow ۱۰۰ \times ۵ + ۲ \times ۵ = ۵۱۰ms$

.۲

اگر بافر ما محدود باشد آنوقت تعدادی از بسته ها دراپ میشوند و نیاز است دوباره ارسال شوند که باعث افزایش تاخیر میشود.

۵

.۱

در حالت تقسیم پیام، به هر ۱۰۰ بسته یک هدر ۱۰۰ بیتی اضافه میشود پس کل زمان ارسال برابر است با:
$delay = (۱۰۰ + ۲) \times \frac{۱۰۰+۱۰۰۰۰}{۵\times۱۰^۶} = ۲۰۶/۰۴ms$
در حالت ارسال بدون تقسیم بندی، این سربار زمانی به مراتب کمتر و ناچیز خواهد بود:
$delay = ۳ \times \frac{۱۰۰+۱۰۰۰۰۰۰}{۵\times۱۰^۶} = ۳ \times ۲۰۰/۰۲ms = ۶۰۰/۰۶ms$

.۲

مکانیزم های زیادی وجود دارد، به عنوان مثال اگر یکی از پکت ها دچار مشکل شود و خراب ارسال شود، نیاز نیست کل پکت ها مجدد ارسال شوند و ما میتوانیم پکت خراب را شناسایی کنیم و فقط آن را دریافت کنیم. میتوان در هر پکت اطلاعاتی اضافی قرار بدیم تا در صورت نیاز خطای آن را اصلاح کرد! البته هردوی این روش ها مقداری سربار دارند و ما داریم اطلاعات اضافی در هر پکت قرار میدهیم، اما به طور کلی باعث بهبود اوضاع میشود و مفید است.

و

.۱

اگر یکی از لینک‌های شبکه دارای پهنای باندی متغیر بین ۲ تا ۵ مگابیت بر ثانیه باشد، در حالتی که پیام به صورت یکپارچه و بدون تقسیم ارسال شود، ممکن است کل پیام در زمانی منتقل شود که لینک در کمترین پهنای باند خود قرار دارد. این موضوع باعث افزایش چشم‌گیر زمان انتقال می‌شود.

اما اگر پیام به بخش‌های کوچکتر تقسیم شود، این امکان به وجود می‌آید که بسته‌ها در زمان‌هایی که پهنای باند بالاتر است، منتقل شوند. در نتیجه، بخشی از داده‌ها سریع‌تر جابجا شده و زمان کلی انتقال کاهش می‌یابد.

.۲

در چنین شرایطی، استفاده از Adaptive-Segmentation و تنظیم پویا اندازه بسته‌ها بر اساس شرایط لحظه‌ای شبکه، می‌تواند بهره‌وری انتقال را به شکل قابل توجهی افزایش دهد و عملکرد کلی شبکه را بهبود بخشد. به این صورت که در زمان ازدحام، بسته های کوچک تری ارسال میشود.

پاسخ مسئله‌ی ۳.

زمان مورد نیاز برای جست‌وجوی DNS به این صورت است:

$$RTT_۱ + RTT_۲ + RTT_۳ + RTT_۴ = ۲۵۰ + ۱۵۰ + ۱۰۰ + ۵۰ = ۵۰۰ms$$

الف

در این حالت به ازای هر فایل رو کانکشن TCP تشکیل داده و فایل را دریافت میکنیم. به ازای هر فایل هم به ۲ RTT نیاز داریم. یکی برای تشکیل کانکشن TCP و یکی هم برای دریافت فایل. پس در مجموع:

$$DNS + ۳ \times (۲ \times RTT_۰) = ۵۰۰ + ۶ \times ۳۰۰ = ۲۳۰۰ms$$

ب

در این حالت هر سه فایل موازی و با یک کانکشن مخصوص به خود دریافت میشوند، پس برای هرسه فایل نیاز به ۲ RTT داریم. زمان DNS هم که همچنان نیاز است، پس:

$$DNS + (RTT_۰ + RTT_۰) = ۳۰۰ + ۳۰۰ + ۵۰۰ = ۱۱۰۰ms$$

ج

در این حالت فقط ۱ کانکشن TCP داریم که همه فایل ها یکی یکی از همین دانلود میشوند. پس یک DNS داریم و یک RTT برای اتصال TCP و به ازای هر فایل هم یک RTT . پس:

$$DNS + (RTT_۰ + RTT_۰ + RTT_۰ + RTT_۰) = ۵۰۰ + ۱۲۰۰ = ۱۷۰۰ms$$

پاسخ مسئله‌ی ۴.

پهنای باند اختصاص داده شده به هر اتصال:

$r = \frac{۵۰۰}{N} bps$

زمان ارسال یک بسته کنترلی:

$t_{ctrl} = \frac{۵۰۰ bit}{\frac{۵۰۰}{N} bps} = Ns$

زمان ارسال یک بسته داده کوچک:

$t_{۱۰۰k} = \frac{۱۰۰۰۰۰}{\frac{۵۰۰}{N}} = ۲۰۰Ns$

زمان ارسال یک بسته داده بزرگ:

$t_{۳۰۰k} = \frac{۳۰۰۰۰۰}{\frac{۵۰۰}{N}} = ۶۰۰Ns$

الف

زمان‌هایی که در این حالت باید بررسی کنیم عبارتند از:

- سه بسته کنترل برای Handshaking TCP
- یک بسته کنترل برای GET + تاخیر پردازش
- ارسال داده
- بسته کنترلی برای ACK نهایی

به هر یک از تاخیرهای بالا یک تاخیر صف هم اضافه خواهد شد.

$T = ۳\times(t_{ctrl}+d_{queue})+(t_{ctrl}+d_{queue}+d_{proc})+(t_{data}+d_{queue})+(t_{ctrl}+d_{queue}) = ۵N+۰/۱+۶\times۰/۰۵+t_{data}$

با توجه به اینکه نسبت تعداد بسته های کوچک و بزرگ برابر است، پس بین طول آن ها میانگین گرفته و فرض میکنیم که بسته های ما ۲۰۰۰۰۰ بیت هستند. پس زمان ارسال آن ها برابر است با: $t_{۲۰۰k} = \frac{۲۰۰۰۰۰}{\frac{۵۰۰}{N}} = ۴۰۰Ns$

پس در نهایت زمان ما برای یک شی برابر است با:

$۵N + ۰/۱ + ۶ \times ۰/۰۵ + t_{data} = (۴۰۵N + ۰/۴)s$

ما در مجموع ۲۱ شی داریم، یکی فایل html و ۲۰ فایل ارجاع شده. و چون N اتصال موازی هستند، پس زمان نهایی برابر است با:

$T_{total} = \frac{۲۱\times(۴۰۵N+۰/۴)}{N} = ۸۵۰۵ + \frac{۸/۴}{N}$

با توجه به سربار اتصال های موازی و ثابت بودن زمان ۸۵۰۵ ثانیه ای، افزایش N عملا کمکی به کاهش زمان کل دانلود نمی‌کند. پس این کار خیلی منطقی نیست.

ب

به صورت شهودی واضح است که بهبود قابل توجهی حاصل نخواهد شد. چون بخش بیشتر زمان دانلود به دلیل حجم بالای داده ها و لینک بسیار کم سرعت است. با HTTP پایا مقدار کمی بهبود سرعت داریم اما در برابر زمان انتقال داده ها بسیار ناچیز است. پس با این کار زمان دانلود مقدار کمی بهبود می یابد، اما آنقدر قابل توجه نیست. برای بهبود سرعت یا باید ظرفیت لینک را افزایش داد یا حجم داده ها را کم کرد.

پاسخ مسئله‌ی ۵.

فرض میکنیم که ریکوئست‌ها به صورت نرمال در ۲ دقیقه زده شده‌اند. پس بین هر ۲ ریکوئست ۱۲ ثانیه فاصله است.

الف

پرسش به ریشه و دریافت ارجاع به TLD : $۵۰ms$

پرسش به سرور TLD و دریافت ارجاع به Authoritative : $۴۰ms$

پرسش نهایی برای آدرس www.example.com و دریافت پاسخ نهایی: $۳۰ms$

بازگشت پاسخ نهایی به کلاینت: $۱۰ms$

پس در مجموع $۱۳۰ms$ برای دریافت نتیجه زمان لازم است.

ب

در این صورت درخواست اول و ششم کش ندارند و باید $۱۳۰ms$ منتظر بمانیم. اما باقی دستورات در DNS محلی کش شده‌اند و هر کدام $۱۰ms$ زمان نیاز دارند. پس:

$$۲ \times ۱۳۰ + ۸ \times ۱۰ = ۳۴۰ms$$

ج

در این صورت درخواست‌های اول و چهارم و هفتم و دهم کش ندارند و هر کدام $۱۳۰ms$ زمان نیاز دارند. باقی درخواست‌ها هر کدام $۱۰ms$ نیاز دارند. پس:

$$۴ \times ۱۳۰ + ۶ \times ۱۰ = ۵۸۰ms$$

د

افزایش TTL در Authoritative-DNS

در این صورت مدت زمان بیشتر آدرس کش میشود و نیاز نیست به DNS Root ریکوئست زده شود. البته از معایب این کار هم دیرتر اپدیت شدن تغییر ip است. البته میتوان با یک عدد معقول حداکثر بهره را از کش کردن برد.

استفاده از کش سمت مرورگر کاربر

در حال حاضر فقط سرور DNS محلی کش دارد (به مدت ۶۰ ثانیه). اگر کلاینت یا مرورگر نیز بتواند نتایج resolve شده را در سطح خودش کش کند، دیگر نیازی به تماس مجدد با سرور DNS محلی در هر درخواست نخواهد بود.

استفاده از DNS-Prefetching در مرورگر

مرورگر یا برنامه، دامنه‌هایی که احتمال میده کاربر به اون‌ها مراجعه کنه رو از قبل resolve می‌کنه.

پاسخ مسئله‌ی ۶.

میدانیم که:

$D_{CS} \geqslant max\{\frac{NF}{u_s}, \frac{F}{d_{min}}\}$
$D_{P۲P} \geqslant max\{\frac{F}{u_s}, \frac{F}{d_{min}}, \frac{NF}{u_s+\sum u_i}\}$

همچنین:

$F = ۴۰Gbit = ۴۰۰۰۰Mbit$
$d = ۲Mbps$
$u_s = ۲۰Mbps$

الف

در حالت Client-Server داریم:

$D_{CS} \geqslant max\{\frac{N\times۴۰\times۱۰۲۴}{۲۰}, \frac{۴۰\times۱۰۲۴}{۲}\} = max\{۲۰۴۸N, ۲۰۴۸۰\}$

Delay	N
ms۵۰ + s۲۰۴۸۰	۱۰
ms۵۰ + s۲۰۴۸۰۰	۱۰۰
ms۵۰ + s۲۰۴۸۰۰۰	۱۰۰۰

در حالت P۲P فرض میکنیم اپلود همگن است، پس: $\sum u_i = N \times u$

پس برای این حالت از این رابطه استفاده میکنیم:

$D_{P۲P} \geqslant max\{\frac{۴۰۹۶۰}{۲۰}, \frac{۴۰۹۶۰}{۲}, \frac{۴۰۹۶۰N}{۲۰+Nu}\} = max\{۲۰۴۸۰, ۲۰۴۸, \frac{۴۰۹۶۰N}{۲۰+Nu}\}$

Delay(s)	u(Mbps)	N
۲۰	۰.۳	۱۰
۲۰	۰.۷	۱۰
۲۰	۲	۱۰
۸۲	۰.۳	۱۰۰
۴۶	۰.۷	۱۰۰
۲۰	۲	۱۰۰
۱۲۸	۰.۳	۱۰۰۰
۵۷	۰.۷	۱۰۰۰
۲۰	۲	۱۰۰۰

پس در $N = ۱۰$ تاخیر به زمان دانلود محدود میشود، اما در N های بیشتر، حالت P۲P نتیجه بهتری دارد.

ب

وقتی ۲۰٪ از همتاها هر ۵ دقیقه از شبکه جدا می‌شوند، بلوک‌هایی که دریافت کرده‌اند در شبکه باقی نمی‌مانند و مجبوریم آن‌ها را مجدداً از سرور یا دیگر همتاها درخواست کنیم. این امر باعث بار اضافه روی سرور و طولانی‌تر شدن توزیع می‌شود.

اگر نرخ آپلود هر همتا از توزیع نرمال با $\sigma = ۰/۲u$ پیروی کند، برخی پییرها کندتر و برخی سریع‌تر عمل می‌کنند. نبود تعادل، جریان گردش بلوک‌ها را مختل و زمان کل را به‌طور قابل توجهی افزایش می‌دهد.

پهنای باند سرور با دوره‌ی ۱۰ دقیقه و دامنه‌ی ±۱۰٪ نوسان دارد. در فازهای افت، ظرفیت ارسال کمتر شده و تأمین

بلوک‌های اولیه کند می‌شود که زمان توزیع را بیشتر می‌کند.

RTT حدود $۵۰ms$ بین سرور و همتا و $۱۰۰ms$ بین همتاها باعث تأخیر در ارسال ACK و درخواست بلوک بعدی می‌شود. این انتظار اضافی نرخ گردش داده را کاهش و توزیع را کندتر می‌کند.

با ترکیب این چهار عامل، توزیع فایل از چند دقیقه‌ی ایده‌آل به ده‌ها دقیقه در سناریوی واقعی کشیده می‌شود.

ج

- ایجاد انگیزه برای ماندگاری: با مکانیزم اعتبار و پاداش‌دهی ، کاربران را تشویق کنید پس از دریافت کامل فایل همچنان فعال بمانند.
- یک یا چند همتای قدرتمند با پهنای باند بالا را همیشه به‌عنوان seed نگه دارید تا هر زمان churn بالا رفت بتوانند جایگزین بلوک‌های از دست‌رفته شوند.
- هر همتا پیش از ترک شبکه چند بلوک اضافی برای همتاهای دیگر دانلود و نگهداری کند تا در دوره‌های خروج انبوه، قطع سرویس نداشته باشیم.
- همتاها را بر اساس تاخیر شبکه گروه‌بندی کنید و در اولویت با همتاهای نزدیک‌تر یا با RTT کمتر دانلود/آپلود کنید تا مدت زمان انتقال هر بلوک کوتاه‌تر شود.

پاسخ مسئله‌ی ۷.

الف

پایین آوردن سریع بیت‌ریت در سطح بافر بحرانی به موقع است. با پایین آوردن بیت‌ریت به ۱ Mbps ، احتمال پر شدن بافر و جلوگیری از توقف پخش افزایش می‌یابد. استفاده از اطلاعات بیت‌ریت قبلی و پهنای باند جاری کمک می‌کند که بیت‌ریت انتخابی با شرایط واقعی شبکه تطبیق داشته باشد. الگوریتم فقط به وضعیت فعلی نگاه می‌کند (نه تغییرات آتی یا روند پهنای باند). این باعث می‌شود در مواجهه با نوسانات سریع، پیش‌بینی‌پذیری پایین بیاید. همچنین افزایش یا کاهش‌های بیش از حد کیفیت در این الگوریتم به همراه داریم. الگوریتم در بیشتر شرایط پایدار یا نیمه‌پایدار پهنای باند عملکرد خوبی دارد و از Rebuffering جلوگیری می‌کند. اما اگر پهنای باند به طور ناگهانی و سریع کاهش یابد و سطح بافر هنوز بالای ۵ ثانیه باشد، ممکن است الگوریتم به اندازه کافی سریع واکنش نشان ندهد، و خطر Rebuffering به‌وجود بیاید.

ب

به عنوان مثال:

پهنای باند در دسترس: $B_t = ۶Mbps$

سطح بافر: $L_t = ۶s$

بیت‌ریت آخرین قطعه: $R_t = ۴Mbps$

چون $L_t > ۵s$ است، شرط اول برقرار می‌شود و بیت‌ریت به پایین‌ترین مقدار کاهش می‌یابد. از آنجا که ما اکنون با بیت‌ریت ۱ شروع کرده‌ایم، حتی اگر بافر کم شود یا پهنای باند بالا باشد، شرط ارتقا هیچ‌گاه فعال نمی‌شود (چون $R_t/۲ = ۰/۵ < B_t$) و در نتیجه سیستم در ۱Mbps قفل می‌ماند.

معمولا وقتی بافر کم است، باید بیت‌ریت را کاهش دهیم و وقتی بافر بسیار بالاست، می‌توانیم به‌تدریج افزایش دهیم. اما اینجا برعکس پیاده شده، به محض رسیدن بافر بالاتر از ۵ ثانیه، بیت‌ریت ناگهان به حداقل افت می‌کند، حتی وقتی شبکه خیلی خوب است.

پیشنهاد بهبود: استفاده از میانگین‌گیری از وضعیت پهنای‌باند و بافر در چند لحظه به جای بررسی فقط یک لحظه‌ی خاص.

ج

به عنوان مثال:

پهنای باند ثابت: $B = ۳/۵Mbps$

طول هر قطعه ویدئو: ۲ ثانیه

شروع با بیت‌ریت $R = ۲Mbps$ و بافر اولیه $L = ۷s$

سطح بافر دائم بین خالی شدن تا زیر $۵s$ و پر شدن تا بالای $۱۰s$ نوسان می‌کند و کیفیت ویدئو پیوسته تغییر می‌کند.

پیشنهادات برای رفع این مشکل:

- استفاده از Smoothing-Filter به این صورت که به جای استفاده از B_t لحظه ای، از میانگین متحرک استفاده کند.
- یک شرط اضافه کنیم که یک حداکثر تغییراتی را بررسی کند تا ویدیو یک دفعه از بالاترین کیفیت به پایین ترین کیفیت نپرد.

پاسخ مسئله‌ی ۸.

توضیح کد

در ابتدا یک سرور FTP با استفاده از کتابخانه pyftpdlib بالا میاوریم:

```python
from pyftpdlib.authorizers import DummyAuthorizer
from pyftpdlib.handlers import FTPHandler
from pyftpdlib.servers import FTPServer
import os

FTP_USERNAME = "admin"
FTP_PASSWORD = "admin"

FTP_DIRECTORY = os.path.join(os.getcwd(), "ftp_files")

if not os.path.exists(FTP_DIRECTORY):
    os.makedirs(FTP_DIRECTORY)

authorizer = DummyAuthorizer()

authorizer.add_user(FTP_USERNAME, FTP_PASSWORD, FTP_DIRECTORY, perm="elradfmwMT")

authorizer.add_anonymous(FTP_DIRECTORY)

handler = FTPHandler
handler.authorizer = authorizer

address = ("127.0.0.1", 21)
server = FTPServer(address, handler)

print(f"FTP Server is running at ftp://{address[0]}:{address[1]}")
server.serve_forever()

```

با توجه به اینکه این سرور برای تست است، یوزرنیم و پسورد آن را هاردکد کرده و در کد سرور و کلاینت استفاده میکنیم. سرور FTP ما روی فولدر ftp_files است و تمامی فایل ها در آن پوشه سرو خواهند شد.

حال به سراغ کد سرور میرویم و مرحله به مرحله آن را شرح میدهیم. ما از کتابخانه های اصلی و مهم Socket و Threading و ftplib استفاده میکنیم. همچنین یک آبکجت برای هایClient متصل و یک Counter برای ID آن ها در نظر گرفته ایم.

```python
import socket
import threading
import json
import os
from ftplib import FTP
import base64

HOST = '127.0.0.1'
PORT = 12345

FTP_SERVER = "127.0.0.1"
FTP_USERNAME = "admin"
FTP_PASSWORD = "admin"
FTP_DIRECTORY = 'ftp_files'

clients = {}
client_id_counter = 1
```

این تابع برای ارسال یک پیغام به هایClient متصل استفاده میشود. البته میتوان یک کلاینت خاص را استثنا قرار داد و به آن پیغام ارسال نکرد.

```python
def broadcast(message, exclude_client=None):
    for client in list(clients.keys()):
        if client == exclude_client:
            continue
        try:
            client.send(message)
        except Exception as e:
            print(f"Error broadcasting to a client: {e}")
```

یک تابع هم برای نمایش لیست فایل های موجود در سرور FTP استفاده میشود. ابتدا به سرور با استفاده از یوزرنیم و پسورد داده شده وصل میشود و سپس لیست فایل های موجود در فولدر سرو شده توسط FTP را برمیگرداند. اگر به ارور بخوریم یا فایلی وجود نداشته باشد، یک لیست خالی برگردانده میشوند.

```python
def list_files():
    try:
        ftp = FTP(FTP_SERVER)
        ftp.login(FTP_USERNAME, FTP_PASSWORD)
        files = ftp.nlst()
        ftp.quit()
        return files
    except Exception as e:
        print(f"FTP list_files error: {e}")
        return []
```

این تابع برای دانلود فایل توسط استاد مورد استفاده قرار میگیرد. به این صورت که نام فایل را ورودی میگیرد و سپس فایل را به صورت باینری و در قالب تعدادی بایت برمیگرداند تا بعدا بر روی فایل جدیدی نوشته شود.

```python
def download_file(filename):
    try:
        ftp = FTP(FTP_SERVER)
        ftp.login(FTP_USERNAME, FTP_PASSWORD)
        file_content = bytearray()

        def handle_binary(more_data):
            file_content.extend(more_data)

        ftp.retrbinary(f"RETR {filename}", handle_binary)
        ftp.quit()
        print(f"Downloaded {filename} successfully from FTP.")
        return bytes(file_content)
    except Exception as e:
        print(f"Failed to download file from FTP server: {e}")
        return None
```

این بخش به نوعی جایگزین رابط گرافیکی برای کاربر سمت سرور است. او میتواند کامندهای زیر را استفاده کند و از طریق این تابع، دستور به تابع هندلر اختصاصی آن کامند ارسال میشود.

```python
def server_command_interface():
    while True:
        command = input("\nEnter command (/chat, /listdir, /download): ")
        if command.startswith("/chat"):
            msg = command[len("/chat "):].strip()
            payload = json.dumps({"type": "chat", "from": "Server", "message": msg}).encode('utf-8')
            broadcast(payload)
            print(f"Sent message: {msg}")

        elif command.startswith("/listdir"):
            files = list_files()
            print("Files on FTP:", ", ".join(files))

        elif command.startswith("/download"):
            parts = command.split(" ", 1)
            if len(parts) < 2:
                print("Usage: /download <filename>")
                continue
            filename = parts[1].strip()
            files = list_files()
            if filename in files:
                file_bytes = download_file(filename)
                if file_bytes:
                    with open(filename, "wb") as f:
                        f.write(file_bytes)
                    print(f"Saved '{filename}' locally.")
                else:
                    print("Error downloading file from FTP.")
            else:
                print(f"File '{filename}' does not exist on FTP server.")
        else:
            print("Invalid command. Use /chat, /listdir or /download <filename>")
```

در ادامه یک تابع handle_client هم داریم که به ازای هر Client که به WebSocket ما متصل میشود در یک ترد یک True While ایجاد میکند و دائم منتظر شنیدن دستورات Client است و با توجه به دستور آن، تابع مورد نظر را فراخوانی میکند و پاسخ را در قالب مناسب به آن برمیگرداند. این بخش از کد منطق خاصی ندارد و صرفا زیاده نویسی است.

بخش اصلی سرور ما این تابع است که سرور را بالا میاورد و به ازای هر کانکشن یک ترد با هندلر مشخص ایجاد میکند.

```python
def start_server():
    server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server.bind((HOST, PORT))
    server.listen(5)
    print(f"Server started on {HOST}:{PORT}")

    threading.Thread(target=server_command_interface, daemon=True).start()

    while True:
        try:
            client, address = server.accept()
            threading.Thread(target=handle_client, args=(client, address), daemon=True).start()
        except Exception as e:
            print(f"Error accepting connections: {e}")
            break

    server.close()


start_server()
```

حال به سراغ توضیح کدهای Client میرویم. همانند کد سرور، در ابتدا تعدادی کانفیگ اولیه داریم، یک تابع هم برای upload که مشابه کد سرور است و بر روی FTP فایل را اپلود میکند. یک تابع receive_messages هم داریم که بعد از تشکیل کانکشن، لاجیک خاصی ندارد و صرفا پیام ها را دریافت میکند و در ترمینال برای کلاینت چاپ میکند.

برای بخش ارسال ایمیل ما از کتابخانه smtplib استفاده کردیم که به server smtp گوگل با استفاده از یوزرنیم و

پسورد جیمیل متصل شده و ایمیل را به مقصد مشخص ارسال میکند. بدنه تابع ارسال ایمیل به این صورت است:

```python
def send_email(subject, body):
    try:
        msg = MIMEMultipart()
        msg['From'] = SMTP_USERNAME
        msg['To'] = 'moeeeinaali@gmail.com'
        msg['Subject'] = subject
        msg.attach(MIMEText(body, 'plain'))

        with smtplib.SMTP(SMTP_SERVER, SMTP_PORT) as server:
            server.starttls()
            server.login(SMTP_USERNAME, SMTP_PASSWORD)
            server.sendmail(SMTP_USERNAME, 'moeeeinaali@gmail.com', msg.as_string())
        print(f"Email sent to moeeeinaali@gmail.com")
    except Exception as e:
        print(f"Failed to send email: {e}")
```

بخشی از تابع start_client به این صورت است، وقتی کد client را ران میکنیم این تابع ران میشود. ابتدا با websocket یک کانکشن با سرور برقرار میکند و سپس در یک ترد جداگانه به پیام های ارسالی از طرف سرور گوش میدهد. (تابع آن را در بخش قبل توضیح دادم) سپس در یک while true نقش واسط کاربر client را اجرا میکند و دستورات کلاینت را دریافت میکند:

```python
def start_client():
    client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    client.connect((HOST, PORT))
    print("Connected to the classroom.")

    threading.Thread(target=receive_messages, args=(client,), daemon=True).start()

    try:
        while True:
            user_input = input("\nEnter a command (/chat, /email, /upload, /download): ")
```

نتایج اجرای کد

ابتدا سرور FTP را بالا میاوریم:

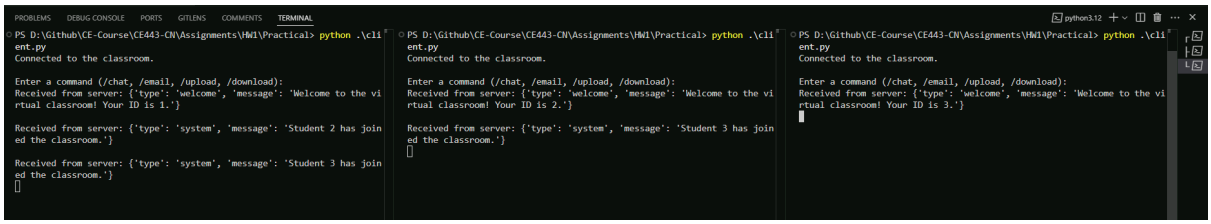
```
D:\Github\CE-Course\CE443-CN\Assignments\HW1\Practical>python ftp.py
FTP Server is running at ftp://127.0.0.1:21
[I 2025-04-23 23:21:30] concurrency model: async
[I 2025-04-23 23:21:30] masquerade (NAT) address: None
[I 2025-04-23 23:21:30] passive ports: None
[I 2025-04-23 23:21:30] >>> starting FTP server on 127.0.0.1:21, pid=9084 <<<
```

سپس سرور استاد را بالا میاوریم:

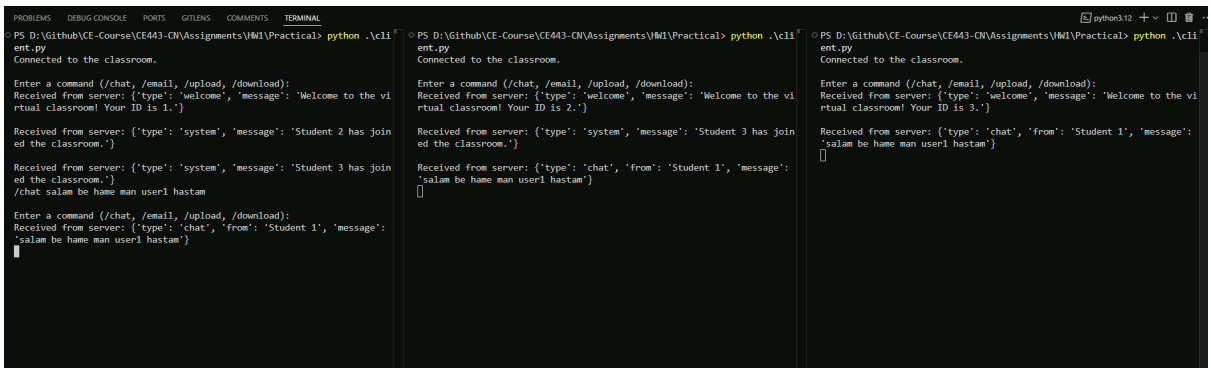
```
PS D:\Github\CE-Course\CE443-CN\Assignments\HW1\Practical> python .\server.py
Server started on 127.0.0.1:12345

Enter command (/chat, /listdir, /download):
```

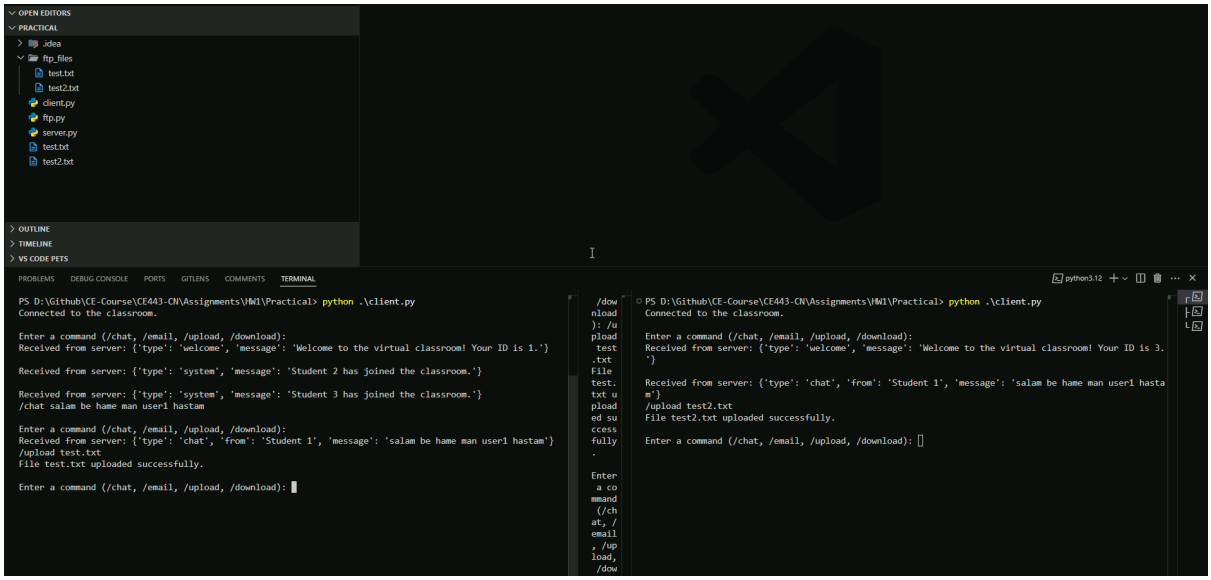
حال با ۳ کلاینت مختلف به سرور وصل میشویم. همانطور که میبینید در ابتدای ورود به هر کاربر Welcome Message ارسال شده و آیدی آن را اعلام میکنیم. همچنین به کاربرانی که از قبل وصل هستند اطلاع میدهیم که یک کاربر جدید اضافه شده است.



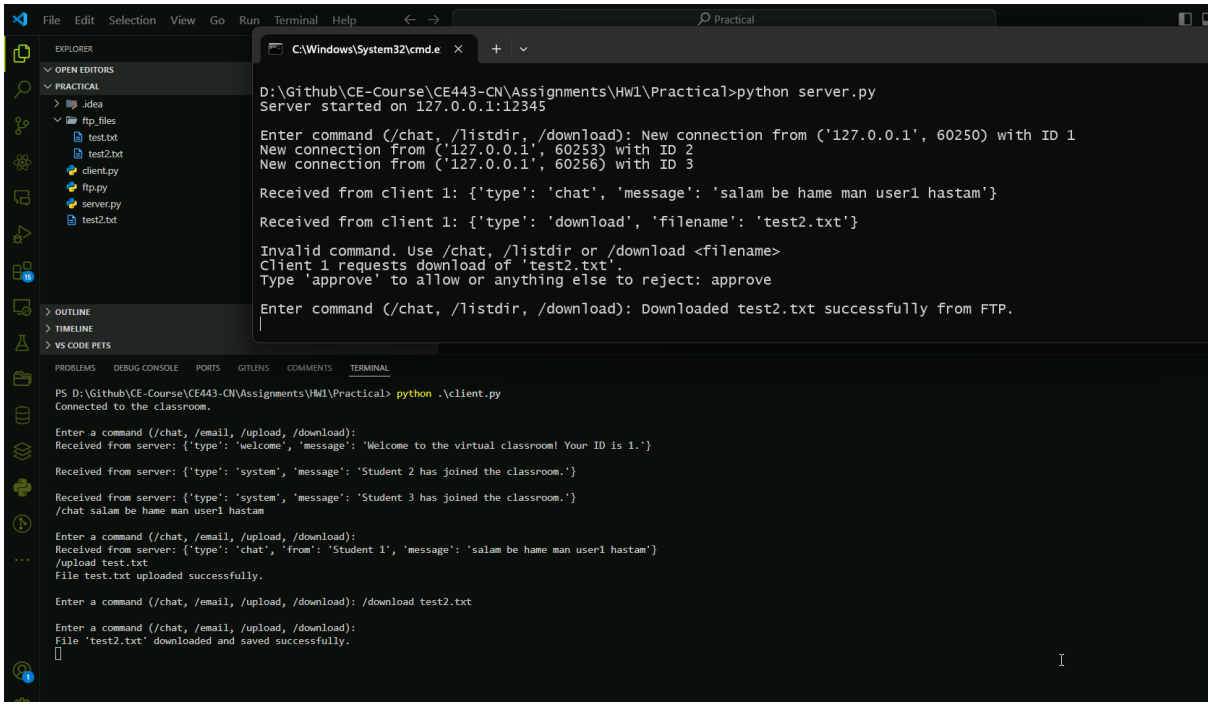
با کاربر ۱ برای بقیه پیام ارسال میکنیم، میبینید که به دست استاد و باقی دانشجوها میرسد:



حال توسط ۲ یوزر، ۲ فایل مختلف را در FTP اپلود میکنیم:

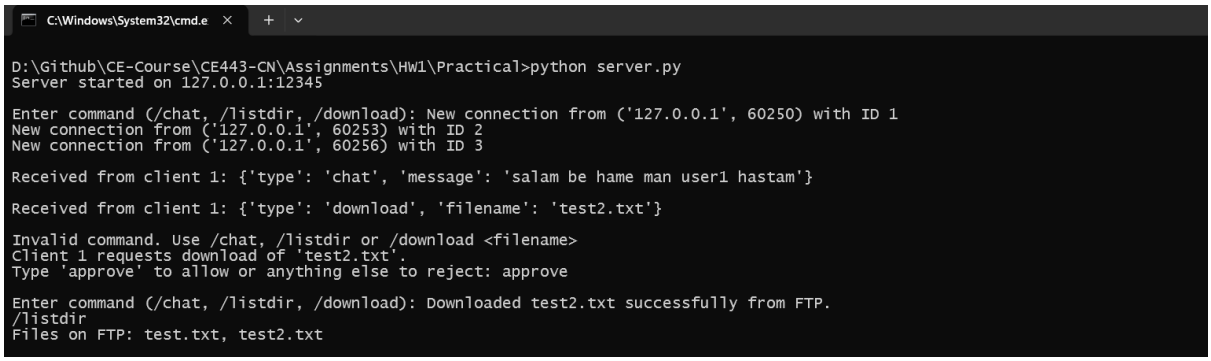


حال اگر درخواست دانلود بدهیم نیاز به تایید استاد است، اگر استاد تایید کند فایل دانلود میشود:

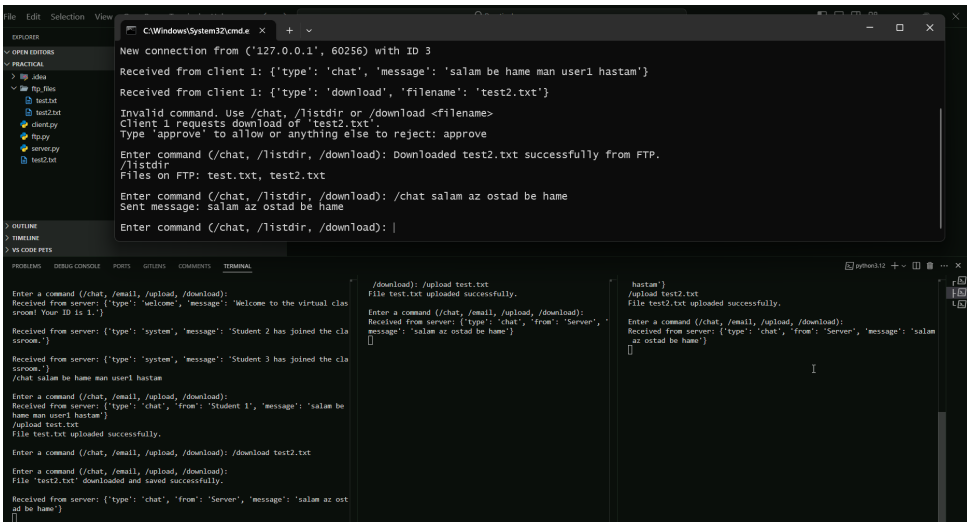


میبنید که کاربر درخواست دانلود میدهد و به سرور پیغام داده میشود، اگر استاد تایید کند فایل برای کاربر دانلود شده و کنار client.py ذخیره میشود. البته لاگ های اپلود فایل ها هم در کد سرور موجود است.

داخل کنسول سرور اگر بخواهیم لیست تمامی فایل ها را ببینیم:

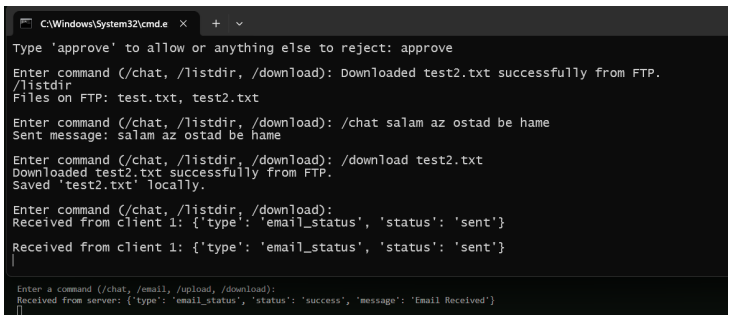


همچنین اگر بخواهیم از طرف استاد به بقیه پیام broadcast کنیم، به این صورت انجام میشود:



همچنین استاد بدون نیاز به تایید میتواند فایل های موجود در FTP را به راحتی دانلود کند:

همچنین با وصل شدن به SMTP گوگل میتوان ایمیل ارسال کرد:



البته باید مراقب باشیم که ایمیل های ما حالت Spam نباشند. در این صورت Google اکانت ما را بن خواهد کرد. (تجربه :))