به نام خداوند بخشنده‌ی مهربان

تمرین برنامه‌نویسی سوم[1]
شبکه‌های کامپیوتری

مدرس: مهدی خرازی **پاییز ۱۳۸۹**

در این تمرین شما پروتکل‌هایی برای یک مکانیزم متعادل کننده‌ی بار[2] در یک شبکه‌ی توزیع شده‌ی محتوا[3]ی فرضی **طراحی** و **پیاده‌سازی** خواهید کرد.

مقدمه

همانطور که می‌دانید یک CDN مجموعه‌ای از کامپیوترها (مثلاً سرورهای وب یا پایگاه‌های داده) با اطلاعات *دقیقاً یکسان* است که در نقاط مختلف یک شبکه قرار گرفته‌اند تا سرعت دسترسی کاربران آن شبکه به داده‌ها بیشتر شود. به این ترتیب هر کاربر از داده‌ی کامپیوتری در شبکه استفاده می‌کند که سرعت بیشتری را برای او فراهم می‌آورد. بنابراین دیگر تمام درخواست‌ها به یک کامپیوتر وابسته نیست و اصطلاحاً کل بار بر روی آن کامپیوتر نخواهد بود. عموماً کامپیوترها در یک CDN در نقاط مختلف جغرافیایی واقعند. اما مرکز داده[4]ای را در نظر بگیرید که باید پاسخگوی حجم زیادی از درخواست‌ها باشد. هر درخواست می‌تواند یک query از پایگاه داده‌ی در آن مرکز داده بوده و بسته به نوع درخواست زمان زیادی بگیرد. در چنین حالتی نیز استفاده از یک سرور، پاسخگوی حجم زیاد درخواست‌ها نخواهد بود و آن مرکز داده برای رفع این مشکل از چندین سرور با اطلاعات دقیقاً یکسان استفاده می‌کند. در چنین حالتی حتی ممکن است سرورها در یک ناحیه‌ی جغرافیایی و یا اصلاً در همان شرکت ارائه کننده‌ی خدمات مرتبط با مرکز داده‌ای باشند.

ممکن است یک ناحیه به دلیل تقاضای زیاد کاربران آنجا، خود از چندین کامپیوتر در CDN برخوردار باشد (و در واقع مزرعه‌ای از سرورها[5] در یک ناحیه باشد) که در این صورت یک متعادل کننده‌ی بار (که از دید کاربران مخفی است) درخواست‌ها را به کامپیوترهایی که بار کمتری در مزرعه دارند هدایت می‌کنند. البته ممکن است کامپوترهای یک مزرعه به دلیل تقاضاهای زیاد

[1] با تشکر از شایان پویا، بهنام مؤمنی، حسن اسلامی، امیر شیخها، علی فتاح‌المنان، فرزانه مقدم و اشکان نیک‌روش

[2] load balancer
[3] Content Distribution Network (CDN)
[4] Date Center
[5] Server Farm

کاربران آن ناحیه بار زیادی را متحمل شده و در نتیجه اتصال کاربران آن ناحیه به مزرعه‌های دیگر ارتباط سریع‌تری را فراهم کند. برای این منظور لازم است تا متعادل کنندگان بار در مزارع یک CDN با یکدیگر در ارتباط باشند تا وضعیت کامپیوترهای باقی مزارع را بدانند تا درخواست یک کاربر را به بهترین کامپیوتر هدایت کنند.

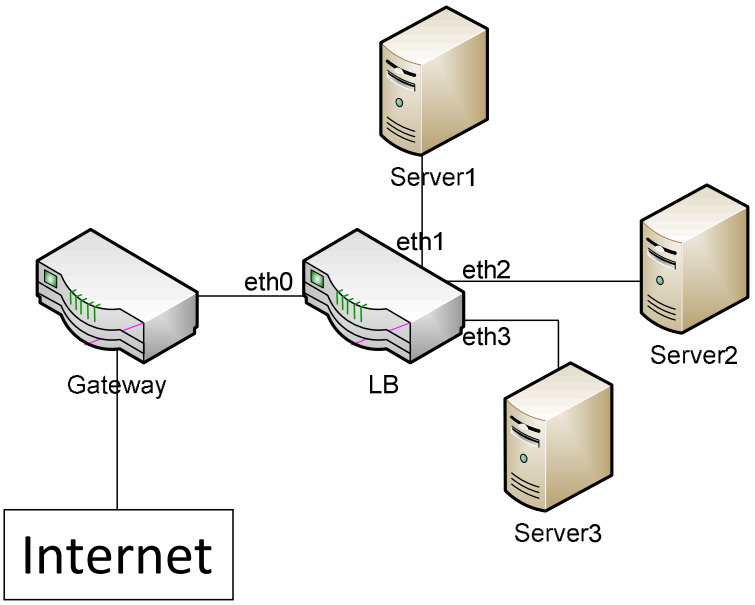
شما باید در این تمرین پروتکلی برای ارتباط متعادل کنندگان بار برای یک شبکه‌ی فرضی از سرورهای پایگاه داده‌ی یک مرکز داده‌ای را طراحی کنید. علاوه بر پیاده‌سازی پروتکل خود، باید سایر وظایف یک متعادل کننده‌ی بار برای هدایت کاربران به سرورهای مربوطه را نیز پیاده‌سازی کنید.

محیط

اساس کار پروتکل شما در این تمرین در لایه ۴ (لایه‌ی بالای IP) است. به همین خاطر در این تمرین نیز، مانند تمرین قبل، از محیط پرتو[6] برای اجرای برنامه‌ی شما استفاده خواهد شد. برای این منظور لازم است برنامه‌های خود را بر روی «چارچوب کاربر»[7] پرتو پیاده‌سازی کنید. برای اطلاعات بیشتر در مورد این چارچوب به مستند «راهنمای چارچوب کاربر» مراجعه کنید.

توپولوژی

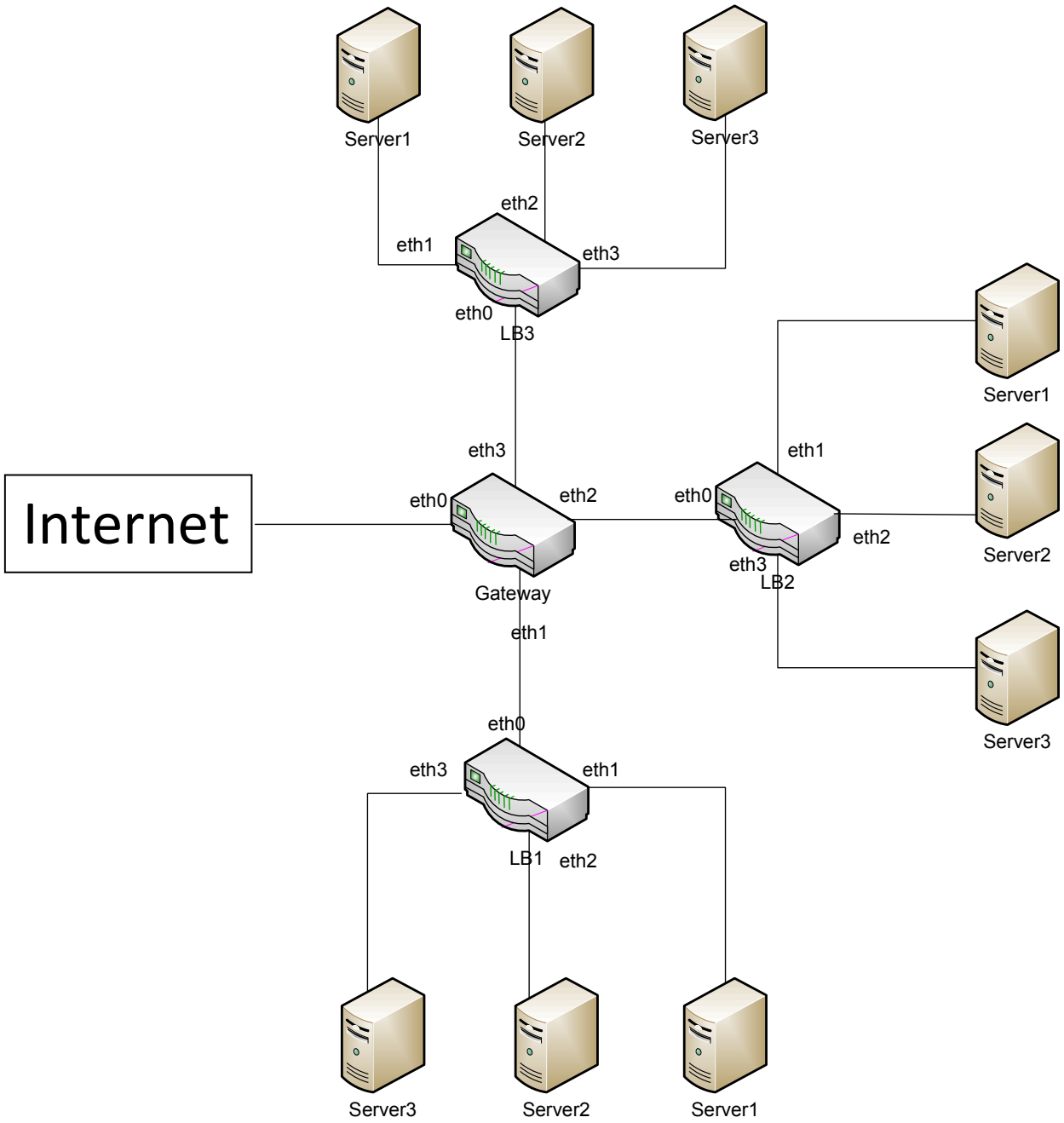
توپولوژی اولیه‌ی هر مزرعه‌ی سرور که در اختیار هر دانشجو قرار می‌گیرد مانند شکل ۱ است. البته برنامه‌ی شما به هیچ وجه نباید وابسته به توپولوژی باشد و یا فرض خاصی در مورد توپولوژی انجام دهد.



شکل ۱. توپولوژی یک مزرعه‌ی سرور

[6] Portable And Reliable Tool fOr Virtualization (PARTOV)
[7] Client Framework (CF)

البته در این تمرین شما باید برنامه خود را همزمان بر روی سه نمونه از این توپولوژی اجرا کنید. به عبارتی توپولوژی‌ای که شما برنامه‌تان را روی آن اجرا می‌کنید مطابق شکل ۲ است.



شکل ۲. توپولوژی با سه مزرعه

هر مزرعه شامل تعدادی سرور و یک متعادل کننده‌ی بار (به اختصار LB) است. تمام سرورها مشابه یکدیگرند و اطلاعات دقیقاً یکسان از یک پایگاه داده را در خود دارند. همچنین تمام سرورها یک IP یکسان دارند و لذا از دید کاربر فقط یک سرور موجود است. هر سرور یک پایگاه داده است که به query هایی که کاربران ارسال می‌کنند پاسخ می‌دهد. بسته به نوع query زمان

پاسخگویی سرور متفاوت خواهد بود. هر درخواست از طرف کاربر در یک بسته‌ی UDP به طرف سرور ارسال می‌شود. این درخواست از طریق اینترنت به Gateway وارد شده و Gateway آن را به interface شماره صفر یکی از LB ها ارسال می‌کند. برای متعادل‌سازی بار مزارع، Gateway ارسال بسته به LB ها را به صورت Round Robin انجام می‌دهد. به عبارتی Gateway بسته‌ها را به صورت چرخشی به LB ها می‌فرستد. سپس LB ای که بسته را دریافت کرده، براساس بار سرورهای مزرعه‌ی خود و همچنین بار سرورهای مزارع دیگر، تصمیم می‌گیرد که این درخواست را به مزرعه‌ی دیگری هدایت کند و یا به یکی از سرورهای مزرعه‌ی خود منتقل کند. در هر حال درخواست باید به یک و فقط یک سرور در یکی از مزارع برسد. دقت کنید که عمل انتقال یا هدایت درخواست از دید کاربر مخفی است و از نظر کاربر فقط یک سرور پاسخگوی او خواهد بود.

در این تمرین برنامه‌ی مربوط به Gateway و Server ها به شما داده شده است و شما باید صرفاً برنامه‌های مربوط به LB ها و client را پیاده‌سازی کنید. جزئیات بیشتر در ادامه خواهند آمد.

انتظارات

شما باید برنامه‌ی مربوط به LB ها و client را بنویسید. یعنی دو برنامه به ترتیب با نام‌های lb.cpp و client.cpp. با اجرای فرمان make این دو برنامه کامپایل می‌شوند و دو فایل اجرایی client.out و lb.out ایجاد می‌شود. نحوه‌ی اجرا کردن این برنامه‌ها با پارامترهای مناسب در فایل‌هایی به همراه «چارچوب کاربر» به شما داده شده است. در ادامه توضیحات دقیق‌تری در مورد نحوه‌ی کار این برنامه‌ها داده خواهد شد.

برنامه‌ها

برنامه‌ی مربوط به client

این برنامه تنها یک IP به عنوان آرگومان ورودی دریافت می‌کند که همان IP سرور پایگاه داده است. این IP در اطلاعات custom information مربوط به LB ها خواهد آمد (در قسمت مربوط به LB گفته خواهد شد). دقت کنید که این IP یک آدرس واقعی در شبکه اینترنت خواهد بود و لذا شما می‌توانید از آن در برنامه‌نویسی سوکت استفاده کنید. به عنوان مثال اگر این IP برابر 1.2.3.4 باشد، برنامه‌ی شما باید به صورت زیر اجرا شود:

```
./client.out 1.2.3.4
```

برای اینکه client بتواند به سرور پایگاه داده query ارسال کند، باید این query را در قالب یک بسته‌ی UDP به پورت 3306 سرور ارسال کند. payload بسته‌های UDP ارسالی از دو بخش تشکیل شده است:

32-bit queryID
Query Information

که queryID یک عدد ۳۲ بیتی است که به ازای هر query تغییر می‌کند و client از آن برای رهگیری پاسخ هر query ارسالی استفاده می‌کند. در حقیقت سرور در پاسخ به query، queryID را نیز در ۳۲ بیت اول payload بسته‌های پاسخ ارسال

می‌کند. همچنین در صورتی که بسته‌ی حاوی query به دلیل بار زیاد سرورها گم شود، بسته‌ای که این موضوع را به client اطلاع می‌دهد در ۳۲ بیت اول خود queryID مربوط به query گم شده را خواهد داشت (بسته‌های اطلاع دهنده‌ی loss و همینطور بسته‌های پاسخ از طرف سرورها در ادامه توضیح داده خواهد شد).

از آنجا که هدف اصلی این تمرین برنامه‌ی مربوط به LB هاست، query information در payload هر query را به طور دلخواه قرار دهید و البته مطمئن شوید که بسته‌ی حاوی query بیش از یک بسته‌ی IP نخواهد شد (مثلاً اگر query information حداکثر ۵۰۰ بایت باشد). لازم نیست برنامه‌ی client شما بسته‌های پاسخ از سرور را پردازش یا حتی نگه‌داری کند. البته در سیستم داوری خودکار، برنامه‌ی client ما در کنار برنامه‌ی LB شما قرار می‌گیرد تا کارایی برنامه‌ی LB شما ارزیابی شود.

در هر حال برنامه‌ی client در محیط پرتو نیست و شما باید صرفاً از برنامه‌نویسی سوکت در آن استفاده کنید.

برنامه‌ی مربوط به LB ها

همانطور که پیش‌تر گفتیم یک LB یک متعادل کننده‌ی بار بین سرورهاست. پس با رسیدن یک درخواست از طرف یک client باید تصمیم بگیرد که درخواست را به چه سروری هدایت کند. بنابراین باید از میزان بار **سرورهای خود** و همچنین **بهترین سرور** در هر مزرعه‌ی دیگر باخبر باشد. لذا باید با LB های دیگر در تماس باشد. به این منظور هر LB یک IP مخصوص به خود دارد تا این ارتباط امکان‌پذیر گردد. البته این IP ها از دید کاربر مخفی است. همچنین هر LB آدرس IP تمام LB های دیگر در CDN را می‌داند (این اطلاعات در custom information آمده است که در انتهای این بخش گفته خواهد شد). شما باید یک پروتکل ارتباطی بین LB ها طراحی و پیاده‌سازی کنید تا هر LB از بار بهترین سرورهای مزارع دیگر نیز باخبر باشد. برای این منظور توجه به مواد زیر ضروری است:

- هر سروری با دریافت query عملیات پردازش آن query را شروع می‌کند. به دلایلی امکان پردازش همزمان چند query در پایگاه داده وجود ندارد. به همین خاطر اگر بسته‌ی دیگری در زمان پردازش به سرور برسد، آن بسته در بافر سرور ذخیره می‌شود. پس از آنکه سرور عمل پردازش را پایان داد، یک یا چند بسته‌ی response را برای client ارسال می‌کند. تمام این بسته‌ها UDP بوده و IP و port مقصدشان، همان IP و port مبدأ در بسته‌ی حاوی query متناظر است. به علاوه ۳۲ بیت اول payload هر کدام از بسته‌های response همان queryID در بسته‌ی حاوی query می‌باشد. دقت کنید که سرور تمام بسته‌های response را به یکباره ارسال می‌کند و این کار همزمان با شروع پردازش query بعدی موجود در بافر انجام می‌شود.
- بار یک سرور در یک مزرعه از دید LB – آن مزرعه، متناسب با response time آن سرور است. به عبارتی LB باید response time تمام سرورهای مزرعه‌ی خود را بداند. response time یک سرور یعنی مدت زمان بین ارسال درخواست به سرور از طرف LB و دریافت اولین بسته‌ی پاسخ از سرور توسط LB. از آنجا که بسته‌های درخواست و پاسخ همگی UDP هستند، و برای محاسبه‌ی response time باید تناظر بین بسته‌های درخواست و پاسخ را بدانیم، لازم است LB اطلاعاتی همچون source IP، source port و همچنین queryID را برای همه بسته‌های درخواست نگه

دارد. در این صورت LB قادر خواهد بود تناظر بین بسته‌های درخواست و پاسخ را دریابد. همچنین فرض کنید در ابتدا response time همه‌ی سرورها صفر است.

- در صورتی که response time چند سرور در مزرعه، برابر با response time کمینه باشد، انتخاب این سرورها برای ارسال درخواست‌ها باید به صورت Round Robin باشد. برای درک بهتر این موضوع به مثال زیر توجه کنید. در این مثال اعداد در هر خط مرحله به مرحله response time های سرورها را نشان می‌دهند و عدد قرمز نشان‌گر سروری است که درخواست را به آن می‌فرستیم. دقت کنید که response time ها با دریاقت بسته‌های پاسخ به روز می‌شوند.

0, 0, 0
0, 0, 0
2, 0, 0
2, 1, 0
2, 1, 1
2, 1, 1

- همانطور که گفته شد، هر بسته‌ی درخواست که به یک سرور می‌رسد، تا زمانی که سرور بتواند به درخواست رسیدگی کند، در بافر سرور ذخیره می‌شود. ممکن است تعداد بسته‌های ذخیره شده در بافر سرور آنقدر زیاد شود که سرور نتواند هیچ بسته‌ی دیگری را بپذیرد. در این صورت بسته‌های درخواست جدید همگی drop می‌شوند. LB باید این اتفاق را تشخیص دهد. با فرض اینکه response time سرور از SERVER_TIMEOUT[8] ثانیه تجاوز نمی‌کند، برای تشخیص این موضوع کافی است LB چک کند که آیا بسته‌ی درخواستی وجود دارد که در فاصه‌ی SERVER_TIMEOUT ثانیه-ی اخیر پاسخی برای آن دریافت شده باشد یا نه. اگر چنین بسته‌ی درخواستی وجود داشت، LB باید آن را به عنوان بسته‌ی drop شده در نظر بگیرد و باید این اتفاق را به client اطلاع دهد. برای این منظور باید یک بسته‌ی UDP بسازد که source IP آن همان IP سرور، source port همان 3306 و destination IP و destination port به ترتیب همان IP و port ـِ client باشد. همچنین باید payload بسته تنها یک عدد ۳۲ بیتی باشد که همان queryID در بسته‌ی درخواست است. در حقیقت تنها فرق بین بسته‌های response از طرف سرور و بسته‌های اطلاع دهنده‌ی loss در این است که بسته‌های اطلاع دهنده‌ی loss به جز queryID چیز دیگری در payload خود ندارند؛ حال آنکه payload بسته‌های response علاوه بر queryID حتماً اطلاعات دیگری را نیز به همراه دارند.
- اگر LB بسته‌ای را به عنوان بسته‌ی drop شده تشخیص داد، پس از ارسال بسته‌ی اطلاع دهنده‌ی loss، باید رکورد مربوط به آن بسته را از حافظه‌ی خود پاک کند.
- اگر LB متوجه شد که یک سرور بسته‌ای را drop کرده، response time را برای آن سرور بی‌نهایت در نظر می‌گیرد. و این response time را به مدت SERVER_REFRESH[9] ثانیه حفظ می‌کند و پس از این زمان وضعیت سرور همانند زمانی است که تازه راه اندازی شده باشد (به عبارتی SERVER_REFRESH + SERVER_TIMEOUT ثانیه

[8] این عدد، ***اولین*** عدد بعد از args-- در اجرا کردن lb.out خواهد بود.

[9] این عدد، ***دومین*** عدد بعد از args-- در اجرا کردن lb.out خواهد بود.

پس از آنکه بسته‌ی drop شده در سرور، توسط LB به سرور داده شده است، وضعیت سرور به همان حالت زمان راه اندازی تغییر می‌کند).

- اگر سرور بسته‌های response را برای بسته‌ای ارسال کند که پیش از این LB آن را به عنوان بسته‌ی drop شده در نظر گرفته بود، LB باید این بسته‌های response را drop کند.
- همانطور که گفته شد هر LB باید علاوه بر میزان بار تمام سرورهای مزرعه‌ی خود، از میزان بار بهترین سرور مزارع دیگر نیز باخبر شود. به این معنی که هر LB باید اطلاعات بهترین سرور در مزرعه‌ی خود را به LB های دیگر اعلام کند. این اطلاعات که درواقع همان response time بهترین سرور در مزرعه‌هاست بین LB ها رد و بدل می‌شود. شما باید این پروتکل ارتباطی را در لایه ۴ (روی IP) طراحی کنید. هر LB باید اطلاعات بهترین سرور خود را هر LB_REFESH[10] ثانیه یک بار به LB های دیگر اعلام کند.
- با فرض اینکه تأخیر انتقال بسته‌ها از یک LB به LB دیگر ناچیز است[11]، اگر یک LB متوجه شود که response time سروری در مزرعه‌ی دیگر کمتر است، باید بسته‌ی درخواست را به آن مزرعه هدایت کند. برای این منظور باید LB بسته-ی درخواست را از طریق یک tunnel به LB دیگر ارسال کند. نحوه‌ی تشکیل این tunnel جزو پروتکلی است که شما باید طراحی کنید. ممکن است به دلیل به روز نبودن اطلاعات LB ها، پس از اینکه بسته‌ای به LB دیگری رسید، آن LB نیز بسته را به LB دیگری هدایت کند. بنابراین ممکن است بسته در یک دور بین LB ها بیفتد و هرگز به سرورها نرسد. برای جلوگیری از این دور، باید در بسته‌های انتقالی در tunnel، از فیلدی همچون TTL در IP استفاده کنید و درصورت تشخیص دور (صفر شدن TTL)، بسته drop شده و این اتفاق (به همان شکل که پیش‌تر گفته شد) به client اطلاع داده شود.
- همانطور که در شکل پیداست، هر LB از طرف interface شماره صفر خود به Gateway متصل است و باقی interface ها به سرورها. بنابراین تعداد سرورها را می‌توان با استفاده از تابع Machine::getCountofInterfaces به دست آورد (همانطور که گفته شد، برنامه‌ی شما نباید وابسته به توپولوژی باشد. بنابراین فرض **نکنید** هر مزرعه فقط ۳ سرور دارد). البته برای سادگی فرض کنید **همواره ۳ عدد LB خواهیم داشت**.
- LB ها نباید بسته‌هایی را خودشان تولید کنند و به سرورها ارسال کنند، چرا که باعث افزایش بار آنها می‌شود که مطلوب نیست. بسته‌های تولید شده توسط LB ها فقط باید بین خودشان رد و بدل شود (البته بسته‌های اعلام کننده‌ی loss به client از این قاعده مستثنی است) و جز این، LB ها باید سایر بسته‌ها را گذر دهند. بنابراین LB برای اینکه مشخصات (response time) سرورهای مزرعه‌ی خود را بفهمد، باید عبور بسته‌ها و اطلاعات به همراه آنها را بررسی کنید.

در نهایت لازم به ذکر است که در custom information برنامه‌های LB اطلاعات زیر به ترتیب و در خطوط متوالی خواهد آمد:

- IP سرور حاوی فایل که برای همه‌ی LB ها ثابت بوده و صرفاً برای اطلاع شماست تا از آن برای اجرای برنامه‌ی client استفاده کنید.

[10] این عدد، ***سومین*** عدد بعد از args-- در اجرا کردن lb.out خواهد بود.

[11] دقت کنید که LB ها همگی در یک سازمان هستند و از طریق یک Gateway با هم در تماسند.

- IP های LB های دیگر، در هر خط یک IP.
- آدرس MAC سرورهای مزرعه‌ی LB (در هر خط یک آدرس).
- آدرس MAC مربوط به Gateway ای که به interface شماره صفر LB متصل است.

نکات ضروری

۱) IP واقعی، IP ای است که با 213.233 شروع می‌شود. البته پیاده‌سازی شما به هیچ وجه نباید مبتنی بر شماره IP باشد و شما نمی‌توانید برنامه‌ی خود را به گونه‌ای بنویسید که مثلاً در آن گفته باشید «اگر فلان IP با 213.233 شروع شده بود فلان کار را بکن».

۲) شما می‌توانید MAC برای هر interface را از طریق فیلد public کلاس Interface با نام mac داشته باشید. به علاوه می‌توانید IP هر interface را با تابع getIP کلاس Interface بدست آورید. همچنین دقت کنید که IP هایی که LB ها از طریق آن با یکدیگر ارتباط برقرار می‌کنند، IP های interface شماره‌ی صفر آنهاست.

۳) برنامه‌ی LB ها نیازی به در نظر گرفتن بسته‌های ARP ندارند. این بسته‌ها، که بین LB ها و Gateway ها و همچنین LB ها و سرورها رد و بدل می‌شوند، توسط محیط پرتو گرفته و بررسی آنها پیاده‌سازی شده است و اساساً برنامه‌ی LB شما چنین بسته‌هایی را دریافت نخواهد کرد.

۴) توپولوژی‌ای که شما برنامه‌ی خود را با آن تست می‌کنید همانند شکل ۲ است که در آن نام LB ها lb1 و lb2 و lb3 است.

۵) برای تست برنامه‌ی خود ابتدا باید run_servers.sh را اجرا کنید تا سرورها شروع به اجرا کنند. سپس LB ها را اجرا کنید و در نهایت به هر تعداد که لازم است برنامه‌ی client را اجرا کنید.

۶) اطلاعات مربوط به delay هر سرور در فایل‌هایی با فرمت txt به شما داده شده است. مثلاً اطلاعات سرور شماره ۲ در مزرعه‌ی lb3 درفایل delay_area3_server2.txt آمده است. این فایل‌ها شامل تعدادی خط است که در هر خط یک زوج عدد آمده است که با space از هم جدا شده‌اند. زوج عدد (x y) به معنی آن است که سرور هر یک از x بسته-ی پاسخ بعدی را با تأخیر y میلی ثانیه ارسال می‌کند. در خط آخر هر فایل، زوج مرتب به صورت (y 0) است، به این معنی که سرور تمام بسته‌های بعدی را با تأخیر y میلی ثانیه ارسال می‌کند. شما می‌توانید برای تست برنامه‌ی خود این فایل‌ها را تغییر دهید. البته برنامه‌ی LB به هیچ وجه نباید از این اطلاعات استفاده کند[۱۲].

[۱۲] مکانیزم ایجاد تأخیر در judge به صورت دیگری انجام می‌شود.

و در نهایت

شما باید یک مستند[۱۳] مختصر (حداکثر ۲ صفحه) بنویسید و در آن تمام پروتکل‌هایی را که طراحی کرده‌اید توضیح دهید. شما صرفاً باید سرآیند پروتکل‌ها و فیلدهایی را که در آن‌ها گرفته‌اید ذکر کنید و بگویید هر فیلد چه معنی‌ای دارد. این فایل را به فرمت PDF و با نام protocol-80123456 (که در آن به جای ۸۰۱۲۳۴۵۶ شماره‌ی دانشجویی خود را می‌آورید) تبدیل کنید.

پس از اتمام تمرین، ابتدا make clean کنید. در پوشه‌ی کاری خود علاوه بر کدهای خود، فایل PDF گفته شده را نیز بگذارید. این پوشه را فشرده کنید و بر روی سیستم خودکار داوری[۱۴] بارگزاری کنید.

[13] documentation

[14] http://partov.sharif.edu/judge