حال اجازه بدهید به کد برنامهٔ مشتری نگاهی بیندازیم. برای آشنایی با عملکرد این برنامه، لازم است با روش فراخوانی و اجرای آن در خط فرمان، آشنا شوید. با فرض آن که نام این برنامه clientانتخاب شده باشد، فراخوانی آن همانند زیر است:

```
client flits.cs.vu.nl  /usr/tom/filename  >f
```

فراخوانی برنامهٔ فوق زمانی موفق است که برنامهٔ سرویس‌دهنده از قبل بر روی ماشین flits.cs.vu.nl اجرا شده و همچنین فایلی با مشخصات /usr/tom/filename موجود و دسترسی برنامهٔ سرویس‌دهنده به آن مجاز باشد. اگر فراخوانی برنامه با موفقیت انجام شود، فایل فوق‌الذکر از طریق اینترنت منتقل و درون **f** نوشته می‌شود؛ پس از آن برنامهٔ client به پایان می‌رسد. از آنجایی که اجرای برنامهٔ سرویس‌دهنده پس از انتقال یک فایل ادامه خواهد یافت فلذا برنامهٔ مشتری را می‌توان بارها اجرا و فایلهای دیگری را دریافت کرد.

در ابتدای برنامهٔ مشتری نیز برخی از تعاریف اولیه آمده است. اجرای واقعی برنامه با بررسی تعداد آرگومانها آغاز می‌شود. (*argc*=3 به معنای سه آرگومان، شامل نام برنامه بهمراه دو آرگومان دیگر است.) دقت کنید که *argv*[1] حاوی نام ماشین سرویس‌دهنده (مثل flits.cs.vu.nl) است و بعداً به کمک تابع سیستمی *gethostbyname* به آدرس IP ترجمه و تبدیل خواهد شد. این تابع سیستمی برای تبدیل نام از سیستم DNS بهره می‌گیرد. DNS را در فصل هفتم تشریح خواهیم کرد.

در ادامه یک سوکت ایجاد و مقداردهی اولیه می‌شود. سپس برنامهٔ مشتری تلاش می‌کند با استفاده از تابع *connect* یک اتصال TCP با سرویس‌دهنده برقرار نماید. اگر سرویس‌دهنده بر روی ماشین با نام مورد نظر، اجرا شده باشد و همچنین به SERVER_PORT گوش بدهد و فضای کافی در صف *listen* وجود داشته باشد، یک اتصال برقرار می‌شود. از طریق این اتصال، برنامهٔ مشتری نام فایل مورد نظر خود را برای سرویس‌دهنده ارسال می‌کند. این کار با تابع *write* انجام می‌شود. پارامتر آخر در تابع *write*، تعداد بایتهایی را که باید ارسال شوند، مشخص می‌کند؛ از آنجایی که نام فایل به همراه کاراکتر '\0' ارسال می‌شود لذا تعداد بایتها یکی بیشتر از طول واقعی رشتهٔ حاوی نام در نظر گرفته شده است.

در اینجا برنامهٔ مشتری به یک حلقه وارد شده و فایل را بلوک به بلوک از سوکت خوانده و در «خروجی استاندارد» کپی می‌نماید. پس از این کار، برنامه به اتمام می‌رسد.

عملکرد زیربرنامهٔ fatal (در صورت فراخوانی) آنست که، یک پیام خطا بر روی خروجی چاپ کرده و از برنامه خارج شود. دقت کنید که برنامهٔ سرویس‌دهنده نیز به همین زیربرنامه احتیاج دارد ولی به دلیل کمبود جا در صفحه، از تعریف آن چشمپوشی شده است. از آنجایی که برنامهٔ سرویس‌دهنده و برنامه مشتری بطور جداگانه کامپایل و بر روی کامپیوترهای متفاوتی اجرا می‌شوند لذا نمی‌توانند از کُد زیربرنامهٔ fatal به صورت اشتراکی بهره بگیرند. این دو برنامه را (بهمراه برخی دیگر از مفاد مرتبط با کتاب) می‌توانید از وب‌سایت *http://www.prenhall.com/tanenbaum* بدست بیاورید. در این سایت بر روی تصویر روی جلد کتاب کلیک کنید؛ در صفحه‌ای که ظاهر می‌شود می‌توانید برنامه‌های فوق را دریافت و آن را بر روی هر سیستم سازگار با یونیکس (مثل سولاریس، BSD و لینوکس) به نحو زیر کامپایل کنید:

```
cc -o client client.c -lsocket -lnst
cc -o server server.c -lsocket -lnst
```

برنامهٔ سرویس‌دهنده با درج نام برنامه بر روی خط فرمان اجرا می‌شود:

```
server
```

به نحوی که اشاره شد برنامه مشتری به دو آرگومان نیاز دارد. نسخهٔ Windows این دو برنامه نیز در وب‌سایت

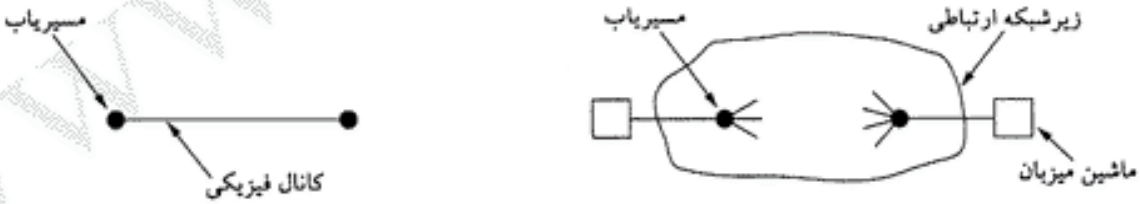
فوق در دسترس می‌باشد.

یادآوری می‌کنیم که این سرویس‌دهنده، از بسیاری جهات تکامل یافته نیست. پردازش و مدیریت خطای آن بسیار ضعیف و گزارشهای خطا ناچیز است. هیچ تمهیدی در خصوص امنیت در آن اندیشیده نشده و فراخوانیهای سیستمی یونیکس در آن، «ویژگی استقلال از محیط اجرا»[1] را مخدوش کرده است. در ضمن این برنامه براساس فرضیاتی نوشته شده است که اصولاً اشتباه هستند؛ به عنوان مثال فرض شده که نام فایل در بافر جا می‌گیرد و به صورت یکجا ارسال می‌شود. از آنجایی که این برنامه تمام تقاضاها را صرفاً به صورت ترتیبی و پشت سرهم پاسخ می‌دهد (چرا که فقط یک «ریسمان» ـThreadـ دارد)، فلذا کارآیی آن بشدت پایین است. با تمام این کاستی‌ها، برنامهٔ فوق کامل بوده و می‌تواند به صورت یک سرویس‌دهندهٔ فایل ایفای نقش کند. خوانندگان گرامی را به توسعهٔ این برنامه دعوت می‌نماییم. برای کسب آگاهی بیشتر در خصوص برنامه‌نویسی سوکت به مرجع (Stevens,1997) مراجعه نمائید.

۲-۶ مؤلفه‌های هر پروتکل انتقال

خدمات انتقال توسط «پروتکلهای انتقال» پیاده‌سازی شده و از آنها بین دو «واحد انتقال» (Transport Entity) استفاده می‌شود. از بسیاری جهات، پروتکلهای انتقال با پروتکلهای لایهٔ پیوند داده که در فصل سوم تشریح شد، شبیه هستند. هر دو با مسئلهٔ نظارت بر خطا، حفظ ترتیب داده‌ها و کنترل جریان و مواردی از این قبیل، سر و کار دارند.

در عین حال، تفاوتهای چشمگیری بین این دو وجود دارد. این تفاوتها عمدتاً ناشی از محیط و بستری است که این دو پروتکل در آن کار می‌کنند. به شکل ۶-۷ دقت کنید: در لایهٔ پیوند داده دو مسیریاب از طریق یک کانال فیزیکی مستقیماً با یکدیگر به تبادل داده می‌پردازند در حالی که در لایه انتقال این کانال فیزیکی جای خود را به یک زیر شبکهٔ کامل [با تعدادی مسیریاب و کانال مخابراتی] می‌دهد. این تفاوت مستلزم پیش‌بینی تمهیدات بسیار مهمی است که در این فصل آنها را بررسی خواهیم کرد.



شکل ۶-۷. (الف) محیط لایهٔ پیوند داده (ب) محیط لایهٔ انتقال.

اولین تفاوت آن است که در لایهٔ پیوند داده، مسیریاب نیاز ندارد که بداند با کدام مسیریاب در حال محاوره است، چراکه هر یک از خطوط خروجی به صورت یکتا، صرفاً یک مسیریاب خاص را مشخص می‌کنند.[2] برعکس در لایهٔ انتقال، نیاز است که آدرس دقیق مقصد مشخص باشد.

---

۱. Platform Independency

۲. به عبارتی دیگر، چون خطوط بین دو مسیریاب مستقیم و نقطه‌به‌نقطه هستند، می‌توان با آن مستقیماً و بدون آن که آدرس طرف مقابل مشخص باشد، مبادله داده کرد. -م

مورد اختلاف دیگر آن است که فرآیند ایجاد یک «اتصال» بر روی سیم (نشان داده شده در شکل ۶-۷-الف) ساده است: طرف مقابل همیشه هست (مگر آنکه از کار بیفتد یا رخدادی جدی حادث شود که در این صورت هیچ کاری نمی‌توان انجام داد!) در لایهٔ انتقال به نحوی که خواهیم دید، ایجاد اتصال فرآیندی پیچیده است.

یکی دیگر از تفاوتهای بسیار آزاردهنده بین لایهٔ پیوند داده و لایهٔ انتقال آن است که زیر شبکه مستعد نگهداری و معطل کردن بسته‌ها در حافظه است. وقتی یک مسیریاب فریمی را بر روی لینک خود ارسال می‌کند بالاخره «» می‌رسد یا به دلیل خطا از دست می‌رود ولی این فریم قادر نیست مدتی سرگردان باشد، در گوشه‌ای از دنیا کور و گم شود و سی ثانیه بعد به ناگاه در مقصد ظاهر گردد. اگر زیرشبکه‌ای از روش دیتاگرام و مسیریابی پویا بهره گرفته باشد، احتمال آن که بسته‌ای در آن ذخیره و چندین ثانیه بعد تحویل مقصد شود قابل چشمپوشی نیست. تبعات ناشی از استعداد زیر شبکه در ذخیره و تعلیق بسته‌ها، بسیار مشکل‌آفرین و برای رفع آنها به پروتکلهای خاص نیازمند است.

تفاوت نهایی بین لایهٔ پیوند داده و لایهٔ انتقال بیشتر در **میزان انتظارات** است تا آن که در **نوع انتظارات** باشد: بافرسازی و کنترل جریان در هر دو لایه الزامی است ولیکن وجود تعداد بسیار زیاد و متغیر «اتصال» (Connection) در لایهٔ انتقال به راهکارها و مکانیزمهای متفاوتی (در مقایسه با راهکارهای اتخاذ شده در لایهٔ پیوند داده) احتیاج دارد. در فصل سوم دیدیم که برخی از پروتکلهای لایهٔ پیوند داده تعداد ثابت و مشخصی بافر برای هر خط اختصاص می‌دهند و هر وقت که فریمی از راه برسد، بافر مورد نیاز موجود خواهد بود. ولیکن در لایهٔ انتقال، ایدهٔ اختصاص تعداد ثابتی بافر برای هر اتصال (به دلیل آن که تعداد اتصالات می‌تواند بسیار زیاد و متغیر باشد) چندان عملی نیست. در بخشهای آتی کلیهٔ این موارد و مواردی از این قبیل بحث و بررسی خواهند شد.

۶-۲-۱ آدرس‌دهی

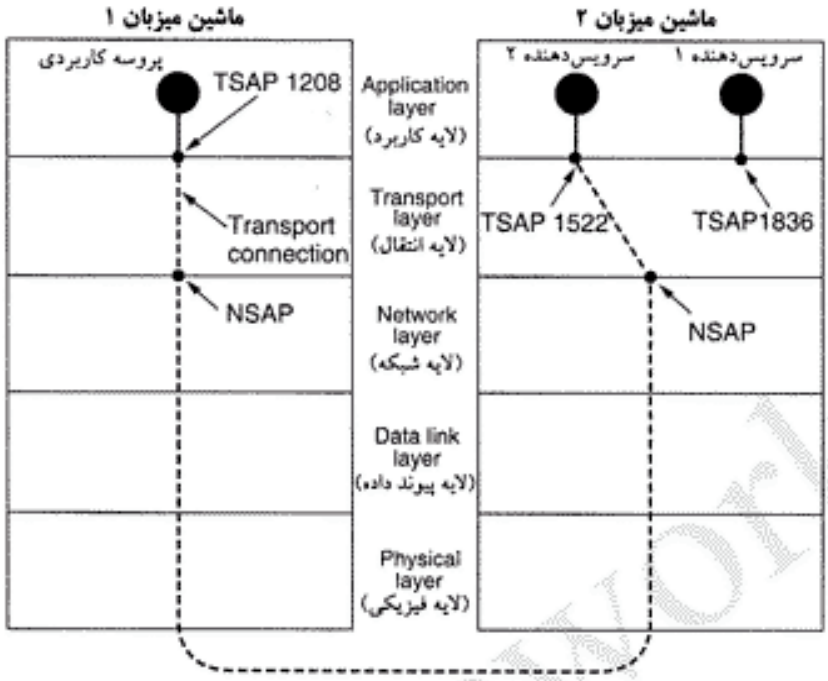
وقتی یک پروسهٔ کاربردی (برنامهٔ کاربر) می‌خواهد با یک پروسهٔ کاربردی راه دور، «اتصال» ایجاد کند باید پروسهٔ مورد نظرش را دقیقاً مشخص نماید. (انتقال بدون‌اتصال نیز با همین مسئله مواجه است: هر پیام برای چه پروسه‌ای ارسال می‌شود؟) روشی که عموماً مورد استفاده قرار می‌گیرد تعریف «آدرس انتقال» (Transport Address) برای پروسه‌هایی است که برای دریافت تقاضای اتصال در حال انتظار (شنود Listening) هستند. [یعنی پروسه‌های در حال شنود توسط یک آدرس یکتا، مشخص می‌شوند.] در اینترنت، به این نقاط پایانی «**پورت**» گفته می‌شود. در شبکهٔ ATM این نقاط، AAL_SAP نام گرفته‌اند. برای آن که عمومیت مطالب تحت تأثیر این واژه‌ها قرار نگیرد ما از واژهٔ کلّی **TSAP**[1] استفاده خواهیم کرد. (مشابه با نقاط پایانی در لایهٔ شبکه -یعنی آدرسهای لایهٔ شبکه- که اصطلاحاً NSAP نامیده می‌شد.)

شکل ۶-۸، ارتباط بین NSAP ، TSAP و اتصالات لایهٔ انتقال را به تصویر کشیده است. پروسه‌های کاربردی اعم از سرویس‌دهنده یا مشتری می‌توانند خود را به یک TSAP متصل نموده تا بتوانند با یک «TSAP راه دور»[2] تماس (اتصال) برقرار کنند. به گونه‌ای که در این شکل نشان داده شده هر اتصال از طریق NSAP برقرار می‌شود.[3] هدف از تعریف TSAP آن است که در شبکه‌ای عظیم که هر کامپیوتر متصل به آن عموماً با یک NSAP واحد شناسایی می‌شود، بتوان نقاط پایانی متعددی را که همگی یک NSAP مشترک دارند، از هم تمیز داد.

---

۱. Transport Service Access Point ۲. Remote TSAP

۳. NSAP یک ماشین واحد را در کلّ شبکه و TSAP یک پروسه را بر روی آن ماشین مشخص می‌کند. ترکیب «TSAP : NSAP» به پروسه‌ها هویت جهانی و منحصر به فرد می‌دهد. -م



شکل ۶-۸ مفهوم TSAP، NSAP و اتصال.

سناریویی برای ایجاد یک اتصال در لایهٔ انتقال، می‌تواند به ترتیب زیر باشد:

۱. یک پروسهٔ سرویس‌دهنده (مثلاً پروسهٔ سرویس‌دهندهٔ Time of Day که تاریخ و ساعت را اعلام می‌کند) بر روی ماشین ۲ خودش را به TSAP 1522 متصل کرده و منتظر دریافت تقاضای تماس دیگران می‌ماند. چگونگی وصل شدن یک پروسه به یک TSAP صرفاً به سیستم عامل محلی هر ماشین بستگی دارد و در محدودهٔ مدل استاندارد شبکه قرار نمی‌گیرد. به عنوان مثال می‌توان تابع سیستمی LISTEN را فراخوانی کرد.

۲. پروسهٔ کاربردی بر روی ماشین ۱ می‌خواهد زمان و تاریخ روز را بداند لذا تقاضای CONNECT را صادر کرده و در آن هویت مبداء (یعنی خودش) را با TSAP 1028 و هویت مقصد (یعنی سرویس‌دهنده) را با TSAP 1522 مشخص می‌کند. این کار موجب می‌شود که یک اتصال بین پروسهٔ کاربردی روی ماشین ۱ با پروسهٔ سرویس‌دهندهٔ روی ماشین ۲ ایجاد شود.

۳. در اینجا پروسهٔ کاربردی تقاضای «زمان و تاریخ» را ارسال می‌کند.

۴. در پاسخ پروسهٔ سرویس‌دهنده، زمان فعلی را اعلام می‌کند.

۵. اتصال (تماس) قطع می‌شود.

دقت کنید که ممکن است بر روی ماشین میزبان ۲، سرویس‌دهنده‌های دیگری نیز اجرا و به TSAP متفاوتی متصل شده باشند؛ همهٔ این سرویس‌دهنده‌ها NSAP مشابهی دارند.[1]

تصویری که ارائه شد مفید و گویاست ولیکن نکتهٔ کوچکی را از قلم انداخته‌ایم: پروسهٔ کاربری ۱ از کجا بداند که سرویس‌دهندهٔ زمان (Time-of-Day Server) به TSAP 1522 متصل شده است؟ یک راه آن است که این سرویس‌دهنده، سالها از TSAP 1522 استفاده کرده باشد و کاربران شبکه به تدریج از این شماره آگاه شده باشند. در این رویکرد، سرویس‌دهنده‌ها TSAP ثابت و پایداری خواهند داشت و می‌توان آنها را درون فایلی در یک

---

۱. در اینترنت NSAP همان آدرس IP است. -م

محل شناخته شده مثل فایل */etc/services* ذخیره کرد. (در این فایل، فهرست تمام سرویس‌دهنده‌های استاندارد و مشهور بهمراه شمارهٔ پورتهایی که بدان متصل هستند، درج شده است.)

آدرسهای ثابت و پایدار TSAP فقط برای سرویس‌دهنده‌های مهم و کلیدی (مثل سرویس‌دهندهٔ وب) مفید خواهد بود، در حالی که پروسه‌های کاربری عموماً می‌خواهند با پروسه‌هائی محاوره کنند که به صورت موقتی اجرا شده‌اند و هیچ آدرس TSAP که از قبل مشخص باشد، ندارند. مضاف بر این، سرویس‌دهنده‌های متنوعی وجود دارند که اغلب آنها کاربرد چندانی ندارند و به صورت موردی اجرا می‌شوند، لذا اختصاص آدرس TSAP ثابت به آنها و فعال نگه داشتنشان بی‌فایده است. کوتاه سخن آن که به روش بهتری نیاز است!

یک روش مناسب در شکل ۶-۹ (به صورت ساده و خلاصه) به تصویر کشیده شده است. این روش به نام «**پروتکل اتصال اولیه**» (Initial Connection Protocol) مشهور است. به جای آن که هر سرویس‌دهنده به یک TSAP شناخته شده گوش بدهد، ماشینی که علاقمند به سرویس‌دهی به کاربران راه دور است پروسهٔ خاصی به نام Process Server را اجرا می‌کند. Process Server ، وکیلِ تمام سرویس‌دهنده‌هایی است که کمتر مورد استفاده قرار می‌گیرند. این پروسه بطور همزمان به مجموعه‌ای از پورتها گوش داده و منتظر تقاضای برقراری ارتباط می‌ماند. کاربران احتمالی، با ارسال تقاضای CONNECT، شمارهٔ TSAP سرویس‌دهندهٔ مورد نظرشان را تعیین می‌کنند. اگر هیچ پروسه‌ای به TSAP مورد نظر آنها متصل نشده و در حال شنود نباشد، لاجرم اتصال آنها مطابق با شکل ۶-۹-الف با Process Server برقرار می‌شود.

وقتی تقاضای برقراری اتصال دریافت شد، Process Server ضمن تشخیص سرویس‌دهندهٔ مورد تقاضا آن را راه‌اندازی و اجرا می‌کند و اتصال ایجاد شده بین خودش و کاربر را به پروسهٔ سرویس‌دهنده تحویل می‌دهد. در این لحظه سرویس‌دهندهٔ تازه اجرا شده، مشغول انجام عملیات درخواستی می‌شود در حالی که Process Server به سر کار اصلیش یعنی گوش دادن به تقاضاهای جدید برمی‌گردد.[1] (شکل ۶-۹-ب)

هر چند «پروتکل اتصال اولیه» برای سرویس‌دهنده‌هایی که فقط برحسب تقاضا و بصورت موردی اجرا می‌شوند بسیار مفید و کارآمد است ولی گاهی اوقات سرویس‌دهنده‌هایی هستند که مستقل از Process Server کار می‌کنند. به عنوان مثال یک سرویس‌دهندهٔ فایل باید به صورت دائمی بر روی یک ماشین خاص اجرا شده باشد و نمی‌توان آن را برحسب تقاضا و موردی اجرا کرد.

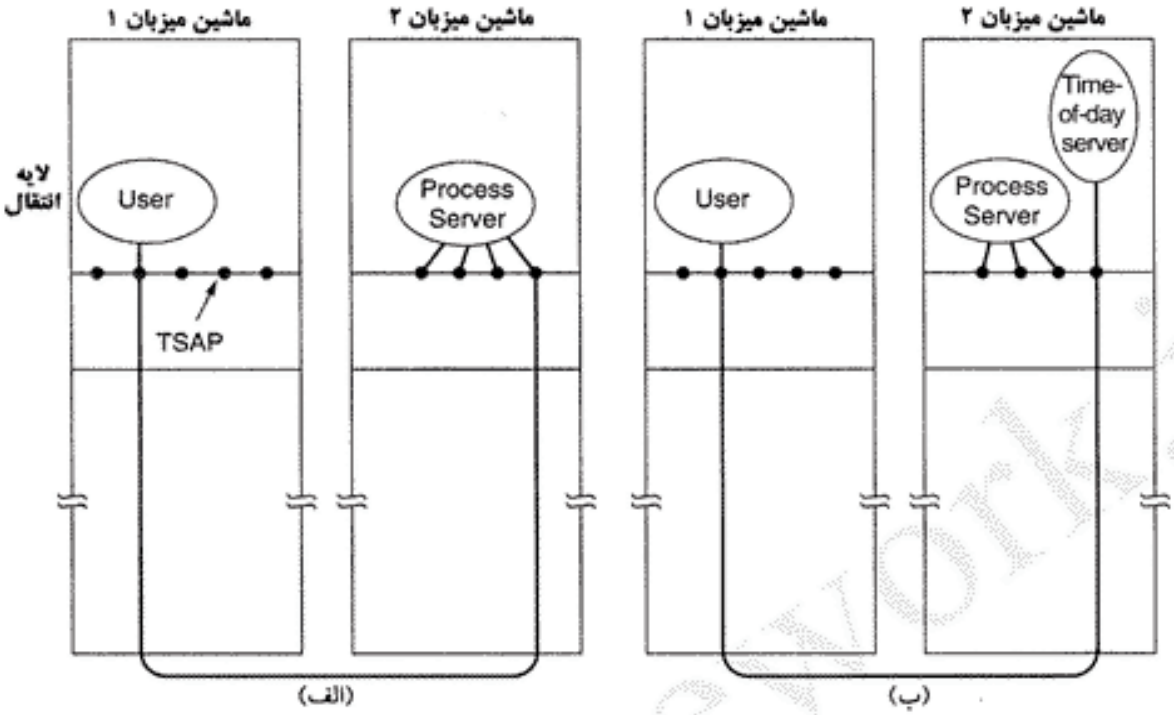
برای حل این مسئله، از الگوی دیگری استفاده می‌شود. در این الگو یک پروسهٔ خاص دیگر به نام Name Server (**سرویس‌دهندهٔ نام**) یا Directory Server (**سرویس‌دهندهٔ دایرکتوری**) بر روی ماشین اجرا می‌شود. برای یافتن آدرس TSAP معادل با نام یک سرویس‌دهنده (مثل سرویس‌دهندهٔ Time-of-Day) ، کاربر ابتدا اتصالی با «سرویس‌دهندهٔ نام» که همیشه به یک شمارهٔ پورت مشهور و شناخته شده گوش می‌دهد، برقرار می‌کند. سپس با ارسال پیامی، نام سرویس‌دهندهٔ مورد نظر خود را سوال کرده و سرویس‌دهندهٔ نام در پاسخ، آدرس TSAP آن را بر می‌گرداند. در اینجا کاربر اتصال خود را با سرویس‌دهندهٔ نام خاتمه داده و اتصال جدیدی با سرویس‌دهندهٔ مورد نظر خود برقرار می‌نماید.[2]

در این مدل هر گاه سرویس‌دهندهٔ جدیدی خلق شود بایستی خودش را در سرویس‌دهندهٔ نام، ثبت کند و نام سرویس (عموماً به صورت یک رشته ASCII) و همچنین TSAP خود را مشخص نماید. سرویس‌دهندهٔ نام این

---

۱. در حقیقت به جای آن که مثلاً n پروسهٔ سرویس‌دهنده به طور همزمان اجرا شود و به پورتهای مورد نظر خود گوش بدهند یک پروسه به نیابت از همهٔ آنها و به طور همزمان به n پورت گوش می‌دهد و در صورت برقراری هر گونه اتصال، پروسهٔ متناظر را اجرا کرده و تماس برقرار شده را به سوی او بر می‌گرداند. ـم

۲. این سرویس‌دهندهٔ نام را با سرویس‌دهندهٔ DNS اشتباه نگیرید.



شکل ۶-۹. چگونگی ایجاد اتصال توسط یک پروسهٔ کاربری بر روی ماشین ۱ با سرویس‌دهندهٔ Time of Day (سرویس‌دهندهٔ تاریخ و ساعت) بر روی ماشین ۲.

اطلاعات را در پایگاه اطلاعات داخلی خود درج می‌کند تا در پرس‌وجوهای بعدی پاسخها را بداند.

عملکرد سرویس‌دهندهٔ نام شبیه به عملکرد اپراتورهای راهنما در سیستم تلفن (۱۱۸) است: اسامی را به شماره، ترجمه می‌نماید. اینجا نیز مثل سیستم تلفن دانستن آدرس TSAP سرویس‌دهندهٔ نام الزامی است. این شماره حداقل اطلاعاتی است که هر کسی باید بداند. اگر شما شمارهٔ اپراتور راهنمای تلفن را ندانید نمی‌توانید برای جستجوی شمارهٔ دیگران با این اپراتور تماس بگیرید. اگر فکر می‌کنید شمارهٔ اپراتور راهنما بدیهی است (مثلاً ۱۱۸)، سعی کنید آن را برای یک کشور خارجی نیز امتحان نمائید!

۶-۲-۲ برقراری اتصال (Connection Establishment)

برقراری یک اتصال ساده به نظر می‌رسد ولیکن حقیقتاً بسیار پیچیده است. در نگاه اول شاید اینگونه به نظر برسد که کافی است «واحد انتقال» (Transport Entity) در یک طرف، بستهٔ تقاضای CONNECTION REQUEST TPDU را به سوی مقصد بفرستد و منتظر پاسخ CONNECTION ACCEPTED بماند. اگر شبکه مستعد از بین بردن بسته‌ها، ذخیره و تعلیق آنها یا تولید بسته‌های تکراری (Duplicate) باشد، در برقراری یک اتصال مطمئن، مشکلات جدی رخ می‌دهد و موجب پیچیدگیهای فراوان در پروتکلهای لایهٔ انتقال می‌شود.

زیرشبکه‌ای را مجسم کنید که با ازدحام مواجه شده و پیامهای تصدیق دریافت بسته‌ها (یعنی پیامهای Ack) سر موعد مقرّر بر نمی‌گردند و هر بسته به دلیل اتمام مهلت، دو یا سه بار ارسال می‌شود. در ضمن فرض کنید که زیر شبکه در درون از روش دیتاگرام بهره گرفته و بسته‌ها از مسیرهای متفاوتی به سوی مقصد می‌روند. ممکن است برخی از بسته‌ها در شلوغی ترافیک زیرشبکه گیر افتاده و تحویل آن به مقصد، مدتی طول بکشد. (یعنی در

زیرشبکه ذخیره شده و با تأخیر از آن بیرون بیاید.)

بدترین کابوس ممکن بدین نحو اتفاق می‌افتد: کاربری یک اتصال با سرویس‌دهندهٔ بانک خود برقرار کرده و با ارسال پیامی از آن می‌خواهد که مقدار قابل توجهی پول به حساب یک شخص نه چندان امین! واریز نماید و سپس اتصال را قطع می‌کند. متأسفانه تمام بسته‌ها در این سناریو به صورت تکراری تولید و در زیرشبکه ذخیره می‌شوند.[1] پس از آنکه اتصال قطع شد بسته‌های معلق از زیرشبکه خارج شده و به ترتیب تحویل مقصد می‌شود یعنی به همان ترتیب قبل با بانک یک اتصال برقرار می‌شود، مجدداً تقاضای انتقال پول انجام می‌گیرد و اتصال قطع می‌گردد. بانک ، راهی برای تشخیص تکراری بودن این بسته‌ها ندارد و طبعاً باید فرض را بر آن بگذارد که تقاضاها جدید هستند و انتقال پول را انجام بدهد!! تا پایان این بخش به بررسی مسئله بروز بسته‌های تکراری در اثر تأخیر خواهیم پرداخت و بر روی الگوریتمهای برقراری مطمئن اتصال (به نحوی که کابوس فوق هرگز اتفاق نیفتد!) تأکید ویژه خواهیم داشت.

معمای این مسئله در وجود بسته‌هایی تکراری نهفته است که با تأخیر دریافت می‌شوند. به روشهای مختلفی می‌توان به این بسته‌ها حمله کرد و آنها را نابود کرد ولی عملکرد هیچکدام صد در صد مطلوب نیست. یکی از این روشها استفاده از آدرسهای انتقال متغیر با زمان است.[2] در این رویکرد هر گاه به آدرس انتقال [یعنی آدرس TSAP] نیاز شد، یک شمارهٔ جدید تولید می‌شود. وقتی اتصالی قطع شد، آن آدرس کنار گذاشته شده و دیگر از آن استفاده نمی‌شود. این استراتژی موجب می‌شود که مدل مبتنی بر Process Server در شکل ۶-۹، ناممکن و غیرعملی باشد.

رویکرد دیگر آن است که به هر اتصال یک «**شناسهٔ اتصال**» (Connection Identifier) (یا به عبارتی یک شمارهٔ ترتیب که به ازای ایجاد اتصال جدید یک واحد افزایش می‌یابد) به انتخاب تماس گیرنده، منسوب شود. این شناسه در هر TPDU درج و ارسال می‌شود. پس از آن که یک اتصال قطع شد، «واحدهای انتقال» (Transport Entities) در دو طرف، شناسهٔ این اتصال را در جدول شناسه‌های قدیمی درج می‌نمایند. در آینده وقتی تقاضای جدیدی مبنی بر برقراری یک اتصال جدید از راه می‌رسد، ابتدا باید به کمک این جدول بررسی شود که مبادا متعلق به اتصالی باشد که قبلاً قطع شده است.

متأسفانه این روش نیز یک اشکال اساسی دارد: هر یک از «واحدهای انتقال» موظفند پیشینهٔ این شناسه‌ها را برای همیشه در جدول خود نگاه دارند. اگر ماشینی از کار بیفتد و حافظهٔ خود را از دست بدهد دیگر نمی‌تواند تشخیص بدهد که کدامیک شناسه‌های اتصال تکراری هستند.

به جای اینها، باید از راه دیگری وارد شویم: به جای آن که اجازه بدهیم بسته‌ها در زیرشبکه عمر جاودان داشته باشند بایستی مکانیزمی اتخاذ کنیم که بسته‌های با عمر زیاد و سرگردان نابود شوند. اگر مطمئن شویم که بسته‌ها هیچگاه بیش از یک مدت معین عمر نمی‌کنند، مشکل فوق‌الذکر قابل کنترل خواهد شد.

طول عمر بسته‌ها را می‌توان با تکنیکهای زیر در حد مشخصی محدود کرد:

۱. طراحی محدود شدهٔ زیرشبکه (Restricted Subnet Design)
۲. درج شمارندهٔ گام در هر بسته (Hop Counter)
۳. درج مهر زمان در هر بسته (Timestamp)

---

۱. چراکه مثلاً هر بسته پس از ارسال در زیرشبکه بیش از حدّ معطل شده و مهلت فرستنده به سرآمده و یک نسخهٔ دیگر از آنرا فرستاده است؛ غافل از آنکه هر دو بستهٔ تکراری نهایتاً با تاخیر به مقصد خواهند رسید. -م

۲. بعبارت دیگر آدرس TSAP برنامهٔ مشتری در هر بار ایجاد اتصال عوض شود. -م

اولین روش، متضمن استفاده از هر راهکاری است که از چرخیدن بسته‌ها در یک حلقه جلوگیری کرده و همچنین تأخیر ناشی از ازدحام را بر روی طولانی‌ترین مسیر ، در حد معینی محدود نگه دارد. دومین روش، مستلزم آن است که در هر بسته یک شمارندهٔ گام قرار داده شده و در مبداء یک مقدار اولیه مناسب به آن منتسب گردد و به ازای عبور از هر مسیریاب یک واحد از آن کم شود. پروتکل لایهٔ شبکه بسته‌هایی را که مقدار این شمارنده در آنها به صفر رسیده حذف می‌نماید. سومین روش نیازمند آن است که هر بسته ، زمان ایجاد خود را با خود حمل نماید و بر این اساس مسیریابها طبق توافق، بسته‌هایی که بیش از یک مدت معین قدیمی شده‌اند را حذف کنند. استفاده از این روش مستلزم آن است که ساعت تمام مسیریابها به دقت با هم تنظیم شده باشد ولیکن این خودش کار ساده‌ای نیست مگر آن که عمل تنظیم ساعتها از بیرون شبکه انجام شود. (مثلاً به کمک سیستم ماهواره‌ای GPS یا ایستگاههای رادیویی خاص تا ساعت دقیق را به همه اعلام کند.)

در عمل نه تنها لازم است که تضمین کنیم بسته‌های قدیمی نابود شده‌اند بلکه باید تمام بسته‌های اعلام وصول آنها (Ack's) نیز از بین رفته باشد. با این استدلال، زمانی بنام T را معرفی می‌کنیم که مقدار آن چند برابر حداکثر طول عمر واقعی بسته‌هاست. (چند برابر بودن، به پروتکل مورد استفاده بستگی دارد.) اگر پس از ارسال یک بسته به اندازهٔ زمان T صبر کنیم، می‌توانیم مطمئن شویم که نه تنها تمام آثار آن بسته از زیرشبکه پاک شده بلکه حتی پیامهای Ack آن نیز از بین رفته است.

با محدود شدن طول عمر بسته‌ها، می‌توان روشی مطمئن و بی‌خطا برای برقراری اتصال ابداع کرد. روشی را که در ادامه معرفی می‌نمائیم، پیشنهاد تاملینسون (Tomlinson/۱۹۷۵) است. این روش مشکل مذکور را حل می‌کند ولی پیچیدگیهای خاص خود را دارد. این روش بعداً توسط دو نفر به نامهای Sunshine و Dalal (۱۹۷۸) بهینه شد. گونه‌های متفاوتی از این روش در عمل مورد استفاده قرار گرفته است (مثلاً در TCP).

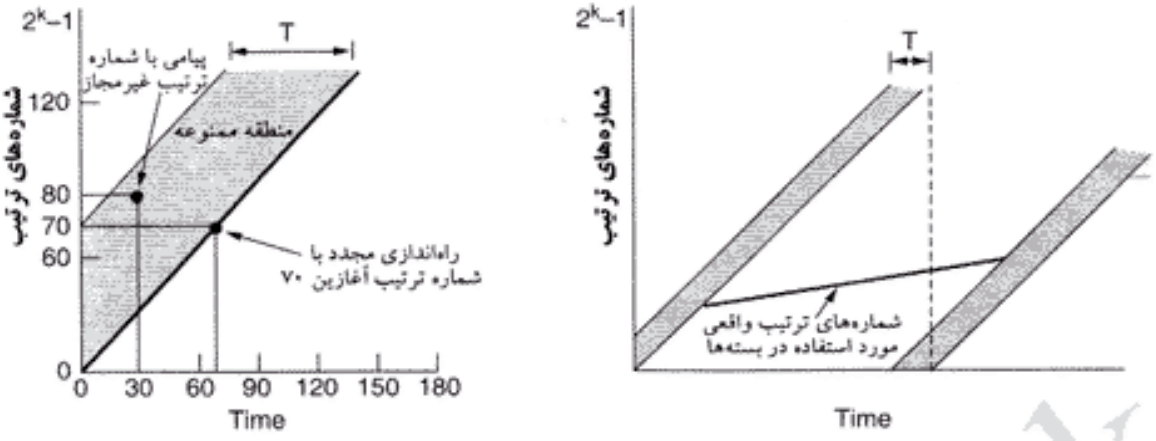
برای آن که مشکل از کار افتادن ماشین و از دست رفتن محتوای حافظه رفع شود، تاملینسون پیشنهاد کرد که در هر ماشین یک ساعت تعبیه شود. لازم نیست که ساعت ماشینهای مختلف با یکدیگر تنظیم شده باشند. فرض بر آن است که این ساعت همانند یک شمارندهٔ باینری عمل می‌کند که در فواصل زمانی مشخص یک واحد افزایش می‌یابد. تعداد بیتهای شمارندهٔ ساعت باید بزرگتر یا مساوی با تعداد بیتهای شمارهٔ ترتیب بسته‌ها باشد. مورد آخر و مهمتر از همه آنکه ساعت مذکور باید حتی وقتی ماشین از کار می‌افتد یا خاموش می‌شود، کار کند.

ایدهٔ اصلی این روش آن است که مطمئن شویم در یک زمان، دو TPDU با شمارهٔ مشابه تولید و ارسال نمی‌شود. برای ایجاد یک اتصال، از k بیت کم ارزش ساعت به عنوان شمارهٔ ترتیب اولیه استفاده می‌شود. (فیلد شمارهٔ ترتیب بسته‌ها نیز k بیتی است.) بنابراین برخلاف پروتکلهای معرفی شده در فصل ۳، در هر اتصال، TPDUها از شمارهٔ ترتیب متفاوتی استفاده می‌کنند. فضای در نظر گرفته شده برای شمارهٔ ترتیب باید آنقدر بزرگ باشد که در صورت برگشت مقدار این شماره به صفر[1] ، TPDUهای قدیمی با همین شماره‌ها، مدتها پیش از بین رفته باشند. رابطهٔ خطی بین «زمان» و «شمارهٔ ترتیب اولیه» در شکل ۶-۱۰ نشان داده شده است.

به محض آن که «واحدهای انتقال» (Transport Entity) در دو طرف، بر روی شمارهٔ ترتیب اولیه به توافق رسیدند، می‌توان از هر پروتکلی نظیر «پروتکل پنجرهٔ لغزان» (Sliding Window Protocol) برای کنترل جریان استفاده کرد. در دنیای واقعی، منحنی شمارهٔ ترتیب (که در شکل با خطوط پر رنگ نشان داده شده) خطی نیست بلکه حالت پلکانی دارد زیرا شمارش ساعت به صورت گسسته (مثلاً در هر هزارم ثانیه) صورت می‌گیرد. برای سادگی از این جزئیات صرفنظر خواهیم کرد.

---

۱. به برگشت یک شمارنده به صفر پدیده Wrapping Around گفته می‌شود. -م



شکل ۶-۱۰. (الف) شمارهٔ ترتیب TPDUها نباید به منطقهٔ ممنوعه وارد شود. (ب) مشکل سنکرون‌سازی مجدّد شماره‌ها

مشکل زمانی بروز می‌کند که ماشین میزبان از کار بیفتد: وقتی از نو شروع به کار می‌کند، «واحد انتقال» از شمارهٔ ترتیب قبلی خود مطلع نیست. یک راه حل آن است که «واحد انتقال» پس از راه‌اندازی مجدد مدت T ثانیه بیکار بماند تا تمام TPDUهای قدیمی از بین بروند. ولی در شبکه‌های بزرگ و پیچیده، مقدار T می‌تواند بسیار بزرگ باشد و بدین ترتیب استراتژی فوق چندان جالب و مفید نیست.

برای آن که نیازی به این زمان مرده نباشد باید محدودیت جدیدی بر روی شماره‌های ترتیب گذاشته شود. با یک مثال به توضیح محدودیت جدید می‌پردازیم: فرض کنید که T (یعنی حداکثر طول عمر بسته‌ها) ۶۰ ثانیه باشد و ساعت سیستم در هر ثانیه یک تیک بزند. به نحوی که در شکل ۶-۱۰-الف با خط تیره رنگ نشان داده شده، شماره ترتیب هر اتصال که در زمان x آغاز می‌شود، همان x است. فرض کنید که در لحظهٔ t=30 یک بستهٔ TPDU معمولی با شمارهٔ ترتیب ۸۰ بر روی اتصال ۵ (که قبلاً ایجاد شده) ارسال گردد. این بسته را TPDU X بنامید. بلافاصله پس از ارسال TPDU X، ماشین میزبان از کار افتاده و سریعاً راه‌اندازی می‌شود. در لحظهٔ t=60 این ماشین اتصالات ۰ تا ۴ را از نو ایجاد می‌کند. در لحظهٔ t=70، اتصال ۵ را نیز از نو برقرار می‌سازد و طبعاً شمارهٔ ترتیب بسته‌ها از ۷۰ شروع می‌شود. در ۱۵ ثانیهٔ بعدی، این ماشین TPDUهای ۷۰ تا ۸۰ را ارسال می‌نماید. در لحظهٔ t=85 یک TPDU جدید با شمارهٔ ۸۰ بر روی اتصال ۵ ارسال و درون زیرشبکه تزریق می‌شود.[1] متأسفانه TPDU X هنوز در زیرشبکه حضور دارد و اگر قبل از TPDU شمارهٔ ۸۰ برسد، پذیرفته شده و بستهٔ جدید (یعنی TPDU 80) به عنوان بستهٔ تکراری حذف خواهد شد.

برای پیشگیری از چنین مشکلاتی، باید راهکاری اتخاذ شود تا شماره‌های ترتیبی که در TPDUهای جدید درج می‌شود تا قبل از انقضای زمان T بهیچوجه مشابه با شماره‌های قبلی نباشد. در شکل ۶-۱۰-الف شماره‌های ترتیبی که در هر لحظه از زمان استفاده از آنها مجاز نیست، تحت عنوان «**ناحیهٔ ممنوعه**» (Forbidden Region) نشان داده شده است.[2] قبل از ارسال هر گونه TPDU بر روی یک اتصال، «واحد انتقال» باید مقدار باینری ساعت

---

۱. فقط شمارهٔ آغازین از ساعت سیستم اخذ می‌شود و برای بسته‌های بعدی این شماره فقط افزایش می‌یابد. -م

۲. به خاطر دشته باشید که ملاک انتخاب شمارهٔ ترتیب برای بسته‌های TPDU در ماشینی که از کار افتاده و سریعاً راه‌اندازی شده ساعت سیستم است. به همین دلیل نمودار «ناحیهٔ ممنوعه» در دو محور زمان و شمارهٔ ترتیب ترسیم شده است. تعبیر ساده‌تر ناحیهٔ ممنوعه آن است که هر ماشین موظف است برای شماره‌گذاری بسته‌ها، از عددی شروع کند که نسبت به مقدار

فعلی را خوانده و بررسی کند که مبادا در ناحیهٔ ممنوعه قرار گرفته باشد.

پروتکل فوق از دو جهت به دردسر می‌افتد: اگر ماشین میزبان تعداد بسیار زیادی بسته را با سرعت فوق‌العاده بالا بر روی اتصال ایجاد شده، بفرستد، آنگاه منحنی «شمارهٔ ترتیب برحسب زمان» شیب بیشتری نسبت به منحنی «شمارهٔ ترتیب اولیه» در شکل ۶-۱۰ پیدا خواهد کرد.[1] این مسئله بدین معناست که حداکثر نرخ ارسال بسته بر روی یک اتصال، نباید از یک TPDU در هر تیک ساعت تجاوز کند. همچنین «واحد انتقال» در ماشینی که از کار افتاده و مجدداً راه‌اندازی شده بایستی قبل از باز کردن هر اتصال جدید آنقدر صبر کند تا ساعت سیستم، تیک بزند مبادا شمارهٔ انتخابی، تکراری شود. هر دوی این موارد فوق، ایجاب می‌کند که تیکهای ساعت بسیار کوتاه باشد. (مثلاً هر چند میکروثانیه یکبار یا حتی کمتر تیک بزند.)

متأسفانه، ورود به ناحیهٔ ممنوعه (در اثر سرعت بسیار زیاد فرستنده) تنها دلیل بروز مشکل نیست. در شکل ۶-۱۰-ب می‌بینیم که اگر نرخ ارسال کمتر از نرخ تیک زدن ساعت باشد، ورود به ناحیهٔ ممنوعه از سمت چپ اتفاق می‌افتد.[2] هر چه شیب منحنی شمارهٔ ترتیب برحسب زمان، بیشتر باشد بروز این مشکل به تعویق خواهد افتاد. همانگونه که قبلاً اشاره کردیم، قبل از ارسال هر TPDU، «واحد انتقال» باید بررسی کند که مبادا به ناحیهٔ ممنوعه وارد شود و اگر اینچنین است بایستی ارسال TPDU را به اندازهٔ T ثانیه به تعویق بیندازد یا آن که شماره‌های ترتیب را از نو سنکرون نماید.

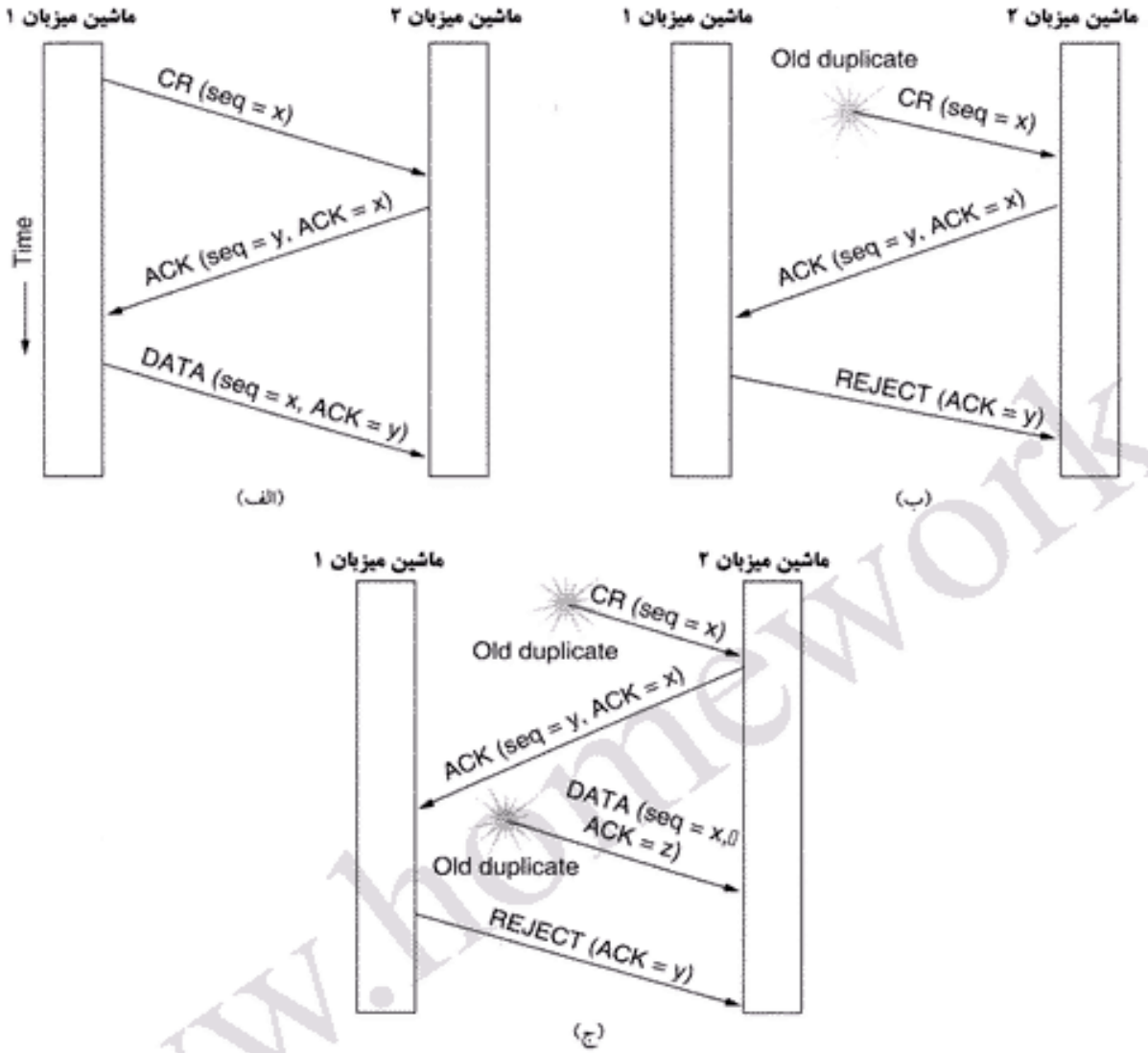
روش مبتنی بر ساعت اگرچه مشکل بسته‌های تکراری (ناشی از تأخیر) را حل می‌کند ولیکن برای آن که این راهکار مفید فایده باشد، ابتدا باید «اتصال» مورد نظر ایجاد شود. از آنجایی که بسته‌های کنترلی TPDU نیز ممکن است با تأخیر مواجه شوند، این استعداد وجود دارد که طرفین، در حین توافق بر روی شمارهٔ ترتیب با مشکل مواجه شوند. به عنوان مثال فرض کنید که با ارسال یک بستهٔ CONNECTION REQUEST TPDU از طرف ماشین ۱ به ماشین ۲ (به همراه شمارهٔ پورت مقصد و شمارهٔ ترتیب اولیه) اقدام به برقراری یک اتصال شود. گیرنده یعنی ماشین میزبان ۲، نیز با ارسال بستهٔ کنترلی CONNECTION ACCEPTED TPDU ، دریافت بستهٔ قبلی را تصدیق (Ack) می‌نماید. اگر بستهٔ CONNECTION REQUEST TPDU از دست رفته و نسخهٔ تکراری آن با تأخیر به ماشین ۲ برسد این اتصال به نحو نادرستی برقرار خواهد شد.

برای حل این مشکل، تاملینسون (۱۹۷۵) روشی به نام **«دست تکانی سه مرحله‌ای»** (Three Way Handshake) پیشنهاد کرد. برای برقراری اتصال، در این پروتکل نیازی نیست که طرفین بر روی شمارهٔ ترتیب مشترک و مشابهی توافق نمایند، فلذا در این پروتکل می‌توان از روشی به غیر از روش مبتنی بر ساعت سراسری سیستم، استفاده کرد. روال معمولی برقراری یک اتصال (که در آن ماشین ۱ شروع کنندهٔ آن است)، در شکل ۶-۱۱-الف نشان داده شده است: ماشین ۱ ، یک شمارهٔ ترتیب دلخواه مثل x انتخاب کرده و با ارسال بستهٔ CONNECTION REQUEST TPDU آن را به طرف مقابل اعلام می‌کند. ماشین ۲ نیز با ارسال بسته‌ای به نام ACK TPDU ، ضمن تصدیق شمارهٔ ترتیب x شمارهٔ ترتیب بسته‌های خودش (یعنی y) را به همتای خود اعلام می‌دارد. نهایتاً ماشین ۱، شمارهٔ ترتیب انتخاب شده توسط ماشین ۲ را (در اولین بستهٔ داده‌ای که ارسال می‌کند)، تصدیق خواهد نمود.

عددی ساعت فعلی، T واحد جلوتر باشد. -م

۱. به عبارت بهتر به دلیل آن که شماره‌های ترتیب در هر تیک ساعت یکبار افزایش می‌یابند لذا اگر بسته‌های TPDU با سرعت بیشتری نسبت به تیکهای ساعت تولید شوند اجباراً برخی از شماره‌های ترتیب تکراری می‌شوند. -م

۲. به عبارت بهتر اگر نرخ ارسال کمتر از نرخ تیک زدن باشد، چون شماره‌های ترتیب باینری و با طول محدود است، به دلیل بالا بودن نرخ تیک زدن، این شماره به صفر برگشته و از نو وارد ناحیهٔ ممنوعه می‌شود. -م



شکل ۶-۱۱. سه سناریوی مختلف برای برقراری اتصال طبق روش «دست تکانی سه‌مرحله‌ای». CR مخفف CONNECTION REQUEST است. (الف) عملکرد طبیعی. (ب) نسخهٔ تکراری و قدیمی بستهٔ CONNECTION REQUEST به ناگاه ظاهر می‌شود. (ج) نسخهٔ تکراری بسته‌های CONNECTION REQUEST و همچنین ACK دریافت می‌شوند.

حال اجازه بدهید بررسی کنیم که این پروتکل در برخورد با نسخه‌های تکراری بسته‌های کنترلی TPDU (که در اثر تأخیر زیرشبکه تولید می‌شوند) چگونه کار می‌کند. در شکل ۶-۱۱-ب، فرض شده که بستهٔ CONNECTION REQUEST که نسخه‌ای تکراری و بجا مانده از یک اتصال قدیمی است، با تأخیر دریافت می‌شود. این TPDU (بدون آن که ماشین ۱ از آن اطلاع داشته باشد) به ماشین ۲ می‌رسد و طبعاً ماشین ۲ با ارسال بستهٔ کنترلی ACK TPDU به آن پاسخ می‌دهد. در حقیقت ماشین ۲ با ارسال این بسته خواسته که بررسی کند آیا ماشین ۱ واقعاً تقاضای برقراری اتصالی جدید داده است. وقتی ماشین ۱، تلاش ماشین ۲ برای برقراری این اتصال را رد می‌کند، ماشین ۲ می‌فهمد که با دریافت یک نسخهٔ تکراری از بستهٔ تقاضا، فریب خورده و از پذیرش اتصال امتناع می‌کند. بدین ترتیب، بسته‌های تکراری (ناشی از تأخیر) خطری ندارند.

بدترین حالت زمانی است که هر دو نسخهٔ تأخیر یافته و تکراری CONNECTION REQUEST و ACK

در زیرشبکه شناور باشند. این حالت در شکل ۶-۱۱-ج دیده می‌شود. همانند مثال قبلی، ماشین ۲ یک نسخهٔ تکراری از بستهٔ کنترلی CONNECTION REQUEST دریافت کرده و بدان پاسخ می‌دهد. در اینجا درک این موضوع اهمیت حیاتی دارد که ماشین ۲ با ارسال y پیشنهاد کرده شمارهٔ ترتیب ترافیک خودش (ماشین ۲) به ماشین ۱ از شماره y شروع شود و y را به گونه‌ای انتخاب کرده که هیچ بستهٔ TPDU یا بستهٔ Ack با شمارهٔ y در شبکه سرگردان نمانده باشد.[1] وقتی بستهٔ تأخیر یافتهٔ دوم (یعنی بسته Ack تکراری و باقیمانده از قبل) دریافت می‌شود، با توجه بدان که به جای y، شمارهٔ ترتیب z مورد تأئید قرار گرفته، ماشین ۲ می‌فهمد که این بسته قدیمی است. نکتهٔ مهم در اینجاست که **هیچ ترکیبی از بسته‌های قدیمی نمی‌تواند باعث شکست پروتکل و ایجاد اتصال ناخواسته گردد.**[2]

۶-۲-۳ خاتمهٔ اتصال

خاتمه دادن به یک اتصال ساده‌تر از ایجاد آن است ولیکن کار به آن سادگی هم که انتظار می‌رود، نیست. قبلاً اشاره کردیم که برای خاتمه دادن به یک اتصال دو سبک وجود دارد: متقارن (Symmetric) و نامتقارن (Asymmetric). روش نامتقارن همان روشی است که در سیستم تلفن هم وجود دارد: وقتی یکی از طرفین گوشی را قطع می‌کند، تماس (اتصال) قطع می‌شود. در خاتمهٔ متقارن، با هر اتصال به صورت دو «ارتباط یکطرفه» رفتار می‌شود که هر یک از آنها به صورت مستقل و مجزا خاتمه می‌یابند.

روش خاتمهٔ نامتقارن، فرآیندی ناگهانی است و می‌تواند منجر به از دست رفتن بخشی از داده‌ها شود. به سناریوی شکل ۶-۱۲ دقت کنید: پس از برقراری اتصال، ماشین ۱ یک بستهٔ TPDU برای ماشین ۲ می‌فرستد و به درستی تحویل می‌شود. سپس ماشین ۱ بسته‌ای دیگر ارسال می‌کند. متأسفانه قبل از تحویل بستهٔ دوم، ماشین ۲ تقاضای قطع ارتباط (DISCONNECT) را صادر می‌نماید. این کار موجب می‌شود که اتصال قطع شود و بستهٔ دوم از دست برود.

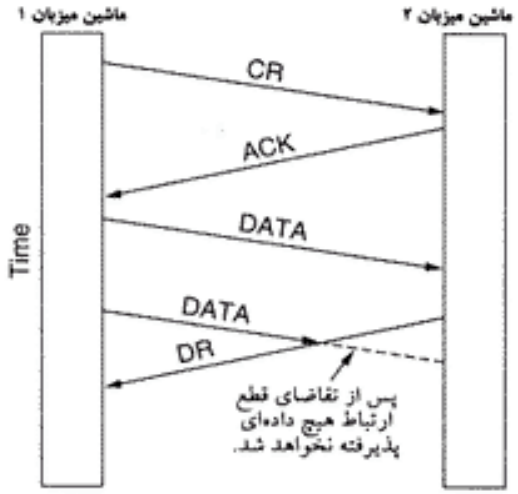
بدیهی است برای آنکه هیچ داده‌ای از دست نرود به پروتکل پیچیده‌تری برای خاتمه دادن به اتصال نیاز است. یک راهکار آن است که از روش متقارن برای قطع اتصال استفاده شود تا اتصال در هر یک از دو طرف به صورت مستقل از دیگری خاتمه یابد. در این روش ماشینی که با ارسال بستهٔ کنترلی DISCONNECT TPDU تقاضای ختم ارتباط می‌کند کماکان به دریافت داده ادامه خواهد داد.

روش خاتمهٔ متقارن (Symmetric Release) زمانی خوب کار می‌کند که هر پروسه حجم ثابت و مشخصی داده برای ارسال داشته باشد و دقیقاً بداند چه زمانی آنها را ارسال کرده است. در شرایطی بغیر از این، تعیین آن که آیا کار به اتمام رسیده و اتصال بایستی قطع شود، ساده نخواهد بود. شاید یک پروتکل پیشنهادی آن باشد که ماشین ۱ بسادگی بگوید: «کار من تمام است. کار شما چطور؟» هر گاه ماشین ۲ پاسخ بدهد که «کار من نیز تمام شد؛ خداحافظ!»، اتصال بین آنها بطور مطمئن خاتمه یابد.

---

۱. یعنی شمارهٔ ترتیب بسته‌های خودش را با اطمینان از عدم وجود بسته‌هایی تکراری و سرگردان با همین شماره‌ها، انتخاب می‌کند. -م

۲. فرآیند فوق را می‌توان در یک عبارت ساده‌تر، بدینگونه خلاصه نمود: ماشین ۱ با ارسال بستهٔ REQUEST اعلام می‌دارد که تمایل به برقراری یک اتصال دارد و در ضمن مایل است بسته‌های خود را از شمارهٔ x شروع کند. x شماره‌ای تصادفی است و از تکراری نبودن آن اطمینان دارد چراکه مثلاً در دو دقیقهٔ قبل تاکنون از آن استفاده نکرده است. ماشین ۲ در پاسخ، ضمن تأئید شمارهٔ x، اعلام می‌دارد که با برقراری اتصال موافق است و او نیز بسته‌هایش را از شمارهٔ y آغاز می‌کند. y نیز شماره‌ای تصادفی و غیرتکراری انتخاب می‌شود. در مرحلهٔ سوم x و y به تأئید نهایی می‌رسد. بدین ترتیب، از آنجایی که بسته‌های تکراری، شماره‌های x یا y آنها فرق می‌کند ـ فارغ از آن که از نوع REQUEST باشند یا Ack ـ پذیرفته نخواهند شد. -م

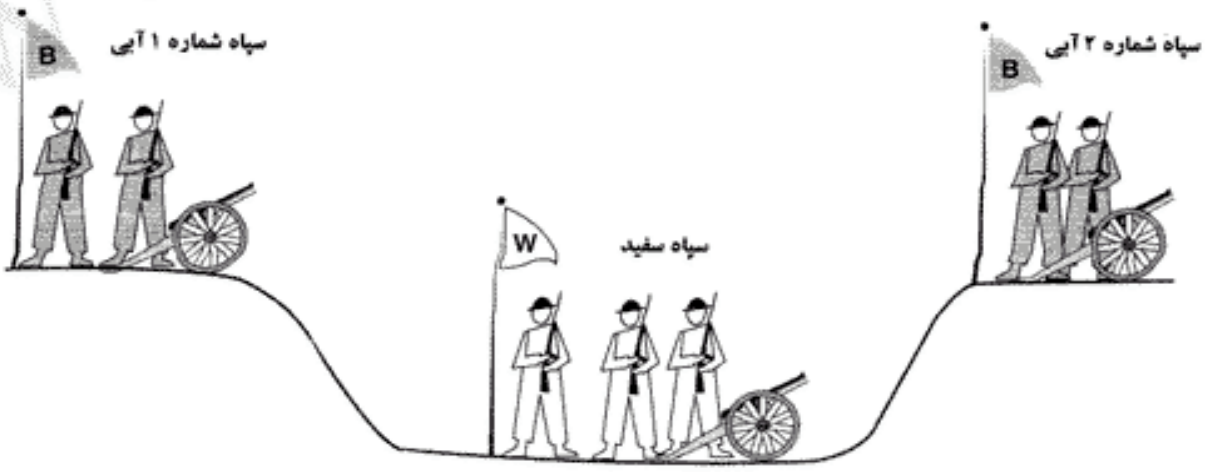


شکل ۶-۱۲. قطع ناگهانی اتصال و از دست رفتن بخشی از داده‌ها.

متأسفانه این پروتکل همیشه کار نخواهد کرد؛ در این خصوص مسئله مشهوری وجود دارد که مشکل پروتکل را مشخص می‌نماید و به نام «**مسئله دو سپاه**» (Two Army Problem) معروف است. تجسم کنید که سپاه سفید در پایین یک درّه اردو زده است. به گونه‌ای که در شکل ۶-۱۳ نشان داده شده، دو سپاه آبی، دره را از دو طرف محاصره کرده‌اند. نفرات سپاه سفید از یک سپاه آبی بیشتر است ولی مجموع نفرات دو سپاه آبی از سپاه سفید افزونتر می‌شود. اگر هر یک از سپاهیان آبی به تنهایی حمله را آغاز کنند شکست خواهند خورد مگر آن که هر دو سپاه آبی در یک زمان یورش ببرند.

سپاهیان آبی می‌خواهند که حملهٔ خود را با یکدیگر هماهنگ نمایند ولیکن تنها راه ارتباط آنها، فرستادن یک پیک به پایین و عبور از درّه است. کاری که ممکن است منجر به دستگیری پیک و از دست رفتن پیغام شود. (به عبارتی آنها با یک کانال ارتباطی نامطمئن مواجه هستند.) سؤال این است که آیا پروتکلی وجود دارد که براساس آن سپاهیان آبی پیروز شوند؟

فرض کنید که فرمانده سپاه یکم آبی پیامی بدین مضمون ارسال می‌کند: «من پیشنهاد می‌کنم که حمله را در سپیده دم روز ۲۹ مارچ شروع کنیم. نظر شما چیست؟» فرضاً این پیام به سلامت می‌رسد و فرمانده سپاه دوم آبی



شکل ۶-۱۳. مسئلهٔ دو سپاه.

موافقت کرده و پاسخ او نیز به سپاه یکم آبی بر می‌گردد. آیا حمله در موعد مقرر انجام می‌شود؟ شاید نه! چرا که فرماندهٔ سپاه دوم نمی‌داند که آیا پاسخ او به سلامت رسیده است یا خیر! به گمان او اگر در راه بازگشت بلایی بر سر پیک آمده باشد، سپاه یکم آبی حمله نمی‌کند، بنابراین شروع چنین نبردی احمقانه خواهد بود!!

حال بیائید پروتکل فوق را از دو مرحله به سه مرحلهٔ دست تکانی (Three Way Handshake) بهبود ببخشیم. فرماندهی که پیشنهاد اول را می‌دهد، باید در نهایت پاسخ طرف مقابل خود را تصدیق کند. حال فرض کنید که هیچ پیامی از دست نرود و سپاه دوم آبی، پیام تصدیق را دریافت نماید ولیکن در اینجا فرماندهٔ سپاه یکم آبی به تردید می‌افتد! چرا که مطمئن نیست که آیا پیام تصدیق او به سپاه دوم رسیده است یا خیر. اگر نرسیده باشد سپاه دوم آبی حمله نخواهد کرد! شاید بتوان پروتکل را چهار مرحله‌ای کرد ولیکن باز هم کمک چندانی نمی‌کند.[۱] در حقیقت می‌توان ثابت کرد که هیچ پروتکلی که بتواند بدون تردید عمل کند وجود ندارد. فرض کنید که چنین پروتکلی وجود داشته باشد. در اینجا یا آخرین پیام پروتکل حیاتی هست یا نیست: اگر حیاتی نیست باید آن را حذف کرد. در حقیقت تمام پیامهای غیرضروری باید حذف شوند تا در پروتکل فقط پیامهای ضروری باقی بمانند. حالا چه اتفاقی می‌افتد اگر آخرین پیام، به مقصد نرسد؟ باید گفت که این پیام ضروری بوده و اگر از دست برود حمله شروع نخواهد شد. از آنجایی که فرستندهٔ آخرین پیام از سرنوشت پیامش مطمئن نیست، خود را با شروع حمله به خطر نخواهد انداخت. بدتر آن که، سپاه طرف مقابل نیز در همین گمان که ممکن است طرف مقابل او حمله نکند، عملیات را آغاز نخواهد کرد!!

برای آن که بتوانید مسئلهٔ دو سپاه را به مسئلهٔ خاتمهٔ اتصال ربط بدهید و با هم قیاس کنید به جای واژهٔ «حمله»، واژهٔ «قطع اتصال» بگذارید. اگر هیچیک از طرفین متقاعد نشوند که طرف مقابل واقعاً آمادهٔ قطع اتصال است، اتصال هیچگاه خاتمه نمی‌یابد!

در عمل، خطر خاتمهٔ یک اتصال از خطر حمله به سپاه سفید کمتر است!! و در ضمن یکی از طرفین آمادگی بیشتری جهت پذیرش چنین خطری را دارد، لذا وضعیت فوق چندان هم ناامیدکننده نیست! در شکل ۶-۱۴ چهار سناریوی مختلفی که ممکن است در پروتکل قطع سه مرحله‌ای اتصال، اتفاق بیفتد به تصویر کشیده شده است. اگرچه این پروتکل چندان مصون از خطا نیست ولیکن معمولاً کفایت می‌کند.

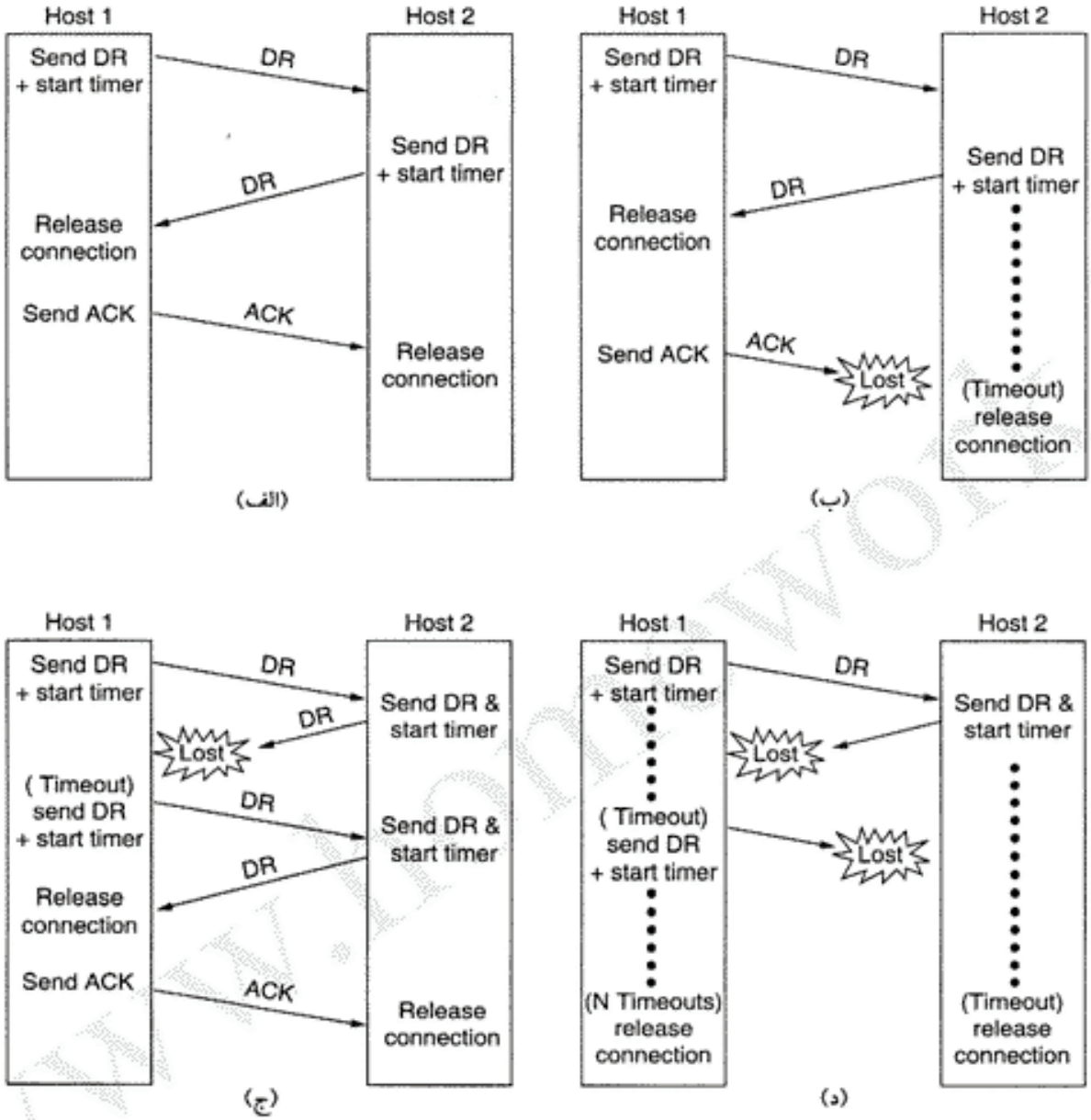
در شکل ۶-۱۴-الف، حالت طبیعی قطع اتصال را مشاهده می‌کنیم که در آن ماشین ۱ با ارسال بستهٔ کنترلی DR TPDU (DISCONNECT REQUEST) تقاضای خاتمهٔ اتصال را به طرف مقابل خود ارسال کرده است. وقتی در طرف مقابل این بسته دریافت شود، گیرندهٔ آن با ارسال متقابل بستهٔ DR TPDU، پاسخ داده و یک تایمر را فعال می‌کند تا در صورت از بین رفتن این بسته از تایمر کمک بگیرد. وقتی این بستهٔ DR دریافت شود، فرستندهٔ اصلی با بازگرداندن بستهٔ ACK TPDU، به این فرآیند پایان داده و اتصال از طرف او قطع می‌شود. در نهایت وقتی بستهٔ ACK TPDU به طرف مقابل می‌رسد او نیز به این اتصال خاتمه می‌دهد. «**خاتمهٔ یک اتصال**» بدین معناست که «واحد انتقال» (Transport Entity) تمام اطلاعاتی را که در خصوص آن اتصال در «جدول اتصالات باز» ذخیره نموده پاک کند و به نحوی به صاحب آن اتصال (کاربر) اطلاع بدهد. این عمل با صدور تابع DISCONNECT که توسط کاربر لایهٔ انتقال انجام می‌گیرد متفاوت است.

شکل ۶-۱۴-ب بیانگر حالتی است که بستهٔ ACK TPDU از دست رفته است (این موضوع توسط تایمر آشکار می‌شود). وقتی مهلت تایمر مربوطه منقضی گردد، خواه ناخواه اتصال مربوطه قطع می‌شود.

اکنون حالتی را در نظر بگیرید که در آن دومین بستهٔ کنترلی DR از بین می‌رود. کاربری که در ابتدا اقدام به تقاضای قطع اتصال کرده پاسخ مورد نظر خود را دریافت نخواهد کرد و این کار را از نو تکرار می‌نماید. شکل

---

۱. زیرا همیشه در آخرین پیام شک و تردید وجود دارد و یک دور باطل ایجاد می‌شود. -م



شکل ۶-۱۴. چهار سناریوی مختلف خاتمه دادن به اتصال. (الف) حالت طبیعی دست‌تکانی سه‌مرحله‌ای. (ب) آخرین ACK از دست رفته است. (ج) پاسخ از دست رفته است. (د) اولین پاسخ و تمام تقاضاهای بعدی قطع ارتباط (یعنی تمام DRها) از دست رفته‌اند.

۶-۱۴-ج چنین حالتی را به تصویر کشیده است (با این فرض که بعد از بستهٔ DR دوم، هیچ بستهٔ دیگری از دست نرود و تمام بسته‌های TPDU به سلامت و سر وقت تحویل شود.)

در آخرین سناریو که در شکل ۶-۱۴-د نشان داده شده، همهٔ شرایط مثل شکل ۶-۱۴-ج است با این تفاوت که فرض کرده‌ایم به غیر از بستهٔ کنترلی DR اول، تمام بسته‌های بعدی نیز از بین رفته‌اند و هر گونه تلاش جهت ارسال مجدد با شکست مواجه شده است. پس از N بار تلاش پیاپی، فرستنده ناامید شده و به ناچار اتصال را قطع می‌نماید. عاقبت، تایمر گیرندهٔ طرف مقابل نیز منقضی شده و او نیز اتصال را خاتمه می‌دهد.

اگرچه برای دنیای عمل، این پروتکل کفایت می‌کند ولی از دیدگاه تئوری، اگر بستهٔ DR اول و تمام N تکرار

بعدی آن از بین بروند، پروتکل با شکست مواجه می‌شود، زیرا اگرچه فرستندهٔ اولین بسته ناامید شده و پس از انقضای مهلت مقرر، به اتصال خاتمه می‌دهد ولیکن طرف مقابل او از تمام این تلاشها و ارسال پیاپی بسته‌ها بی‌خبر است و طبعاً فعال باقی می‌ماند. در چنین شرایطی، اتصال به صورت «**نیمه باز**» (Half-Open) باقی می‌ماند.

برای اجتناب از چنین وضعیتی می‌توان فرستنده را وادار کرد که حتی پس از N تلاش مجدد باز هم ناامید نشود و آنقدر کار را ادامه بدهد تا بالاخره پاسخی دریافت نماید. ولیکن اگر مهلت طرف مقابل منقضی شده و به اتصال خاتمه بدهد، فرستنده هرگز پاسخی دریافت نخواهد کرد و طبعاً در حلقهٔ بی‌نهایت می‌افتد. اگر هم به طرف مقابل اجازه ندهیم که در اثر انقضای مهلت اتصال را قطع کند آنگاه در شرایطی که در شکل ۶-۱۴-د نشان داده شده، پروتکل قفل خواهد شد.

یکی از روشهای حذف اتصالات نیمه باز وضع این قانون است که اگر پس از گذشت زمان معینی (برحسب ثانیه) هیچ بستهٔ TPDU دریافت نشد، اتصال باید به صورت خودکار قطع شود. بدین ترتیب اگر یکی از طرفین به صورت یکجانبه اتصال را قطع نماید، طرف مقابل متوجه عدم فعالیت او شده و او نیز به اتصال خاتمه می‌دهد. البته اگر بخواهیم چنین قانونی اعمال شود لازم است که «واحد انتقال» دارای تایمر خاصی باشد که با ارسال یک TPDU شروع به اندازه‌گیری زمان می‌نماید. اگر پس از ارسال آن TPDU، مهلت تایمر منقضی شد و بستهٔ دیگری جهت ارسال وجود نداشت فرستنده موظف به ارسال یک بستهٔ پوچ (بدون داده) برای طرف مقابل است تا از قطع شدن اتصال (در اثر انقضای مهلت مقرر) جلوگیری نماید. در سمت مقابل نیز اگر قانون قطع خودکار اتصال اعمال شده باشد و برای مدت مشخصی هیچ بستهٔ TPDU دریافت نشود (یا به هر دلیلی بسته‌های پوچ در میانهٔ راه از بین رفته باشند) اتصال به صورت خودکار خاتمه می‌یابد.[۱]

بیش از این به جزئیات نخواهیم پرداخت ولیکن تا اینجا باید مشخص شده باشد که خاتمه دادن به یک اتصال بدون از دست دادن داده، به آن سادگی هم که به نظر می‌رسد نیست!

۶-۲-۴ کنترل جریان و بافرسازی (Flow Control and Buffering)

پس از بررسی جزئیات روشهای برقراری و خاتمهٔ یک اتصال، اجازه بدهید چگونگی مدیریت یک اتصال را در حین کار، مطالعه نماییم. یکی از موارد بسیار مهم «کنترل جریان» (Flow Control) است. اگرچه از بسیاری جهات، مسائل مربوط به کنترل جریان در لایهٔ انتقال با مکانیزمهای کنترل جریان در لایهٔ پیوند داده مشابه هستند ولیکن از جهات دیگر دارای تفاوت هستند. تشابه عمده در هر دو لایه آن است که برای جلوگیری از پیشی گرفتن فرستندهٔ سریع از گیرندهٔ کُند و از دست رفتن داده‌ها در اثر عدم هماهنگی سرعت ارسال و دریافت، باید از پروتکل «پنجرهٔ لغزان» (Sliding Window) یا روشی مشابه استفاده شود. تفاوت اساسی این دو لایه آن است که یک مسیریاب معمولاً دارای تعداد کمی خط ارتباطی است در حالی که یک ماشین میزبان (Host) می‌تواند بطور همزمان تعداد بی‌شماری اتصال مختلف برقرار کند. این تفاوت موجب می‌شود که نتوان استراتژیهای بافرسازی پیاده شده در لایهٔ پیوند داده را به لایهٔ انتقال نیز تعمیم داده و اعمال نمود.

در پروتکلهای پیوند داده که در فصل سوم بررسی شدند، فریمها هم در مسیریاب فرستنده و هم در مسیریاب گیرنده، بافر می‌شدند. به عنوان مثال در پروتکل ششم، به ازای هر خط ارتباطی هم فرستنده و هم گیرنده به تعداد

---

۱. به عبارت ساده‌تر اگر هر یک از طرفین ارتباط برای مدت معینی از طرف مقابل خود بسته‌ای دریافت نکند، اتصال را قطع خواهد کرد لذا طرفین موظفند حتی اگر برای لحظاتی داده برای ارسال نداشته باشند، از خود علائم حیاتی نشان داده و بسته‌های پوچ و بدون داده (Dummy) ارسال نمایند تا قانون قطع خودکار اتصالات نیمه باز به درستی کار کند. -م

MAX_SEQ+1 بافر اختصاصی نیاز داشتند که از این تعداد نصفی برای فریمهای ورودی و نصف دیگر برای فریمهای خروجی در نظر گرفته می‌شد. برای ماشینی که حداکثر تعداد اتصالات آن مثلاً ۶۴ تاست و شماره‌های ترتیب بسته‌ها ۴ بیتی هستند، این پروتکل به ۱۰۲۴ بافر نیازمند است.

در لایهٔ پیوند داده، فرستنده فریمهای ارسالی خود را بافر می‌کند تا در صورت نیاز به ارسال مجدد، آنها را در اختیار داشته باشد. اگر زیرشبکه فقط سرویس دیتاگرام عرضه کرده باشد، «واحد انتقال» (Transport Entity) نیز به دلیل مشابه باید بسته‌های ارسالی خود را در بافر نگاهداری کند. در لایهٔ انتقال اگر گیرنده بداند که فرستنده تمام بسته‌های TPDU ارسالی خود را تا زمانی که دریافت آنها تصدیق (Ack) نشود در بافر نگهداری می‌کند، می‌تواند بر حسب شرایط برای اتصالات خاصی بافر ورودی اختصاصی در نظر نگیرد. (یا برعکس به هر اتصال، بافر ورودی و خروجی مجزا اختصاص بدهد.) به عنوان مثال در لایهٔ انتقال، گیرنده می‌تواند از یک فضای بزرگ و مرکزی به عنوان بافر ورودی برای تمام اتصالات استفاده کند. بدین نحو وقتی یک TPDU وارد می‌شود، گیرنده سعی می‌کند به صورت پویا از این بافر فضایی را به آن اختصاص بدهد. اگر چنین فضایی موجود بود بستهٔ TPDU را می‌پذیرد و در غیر این صورت آن را حذف می‌نماید. از آنجایی که فرستندهٔ بسته، برای ارسال مجدد بسته‌هایی که در زیرشبکه از دست می‌روند، آمادگی دارد فلذا حذف TPDU توسط گیرنده مشکلی ایجاد نخواهد کرد، اگرچه این کار منجر به از دست رفتن منابع [مثل پهنای باند] شبکه خواهد شد. فرستنده، ارسال مجدد بسته‌ها را آنقدر تکرار می‌کند تا بالاخره دریافت آن تصدیق شود.

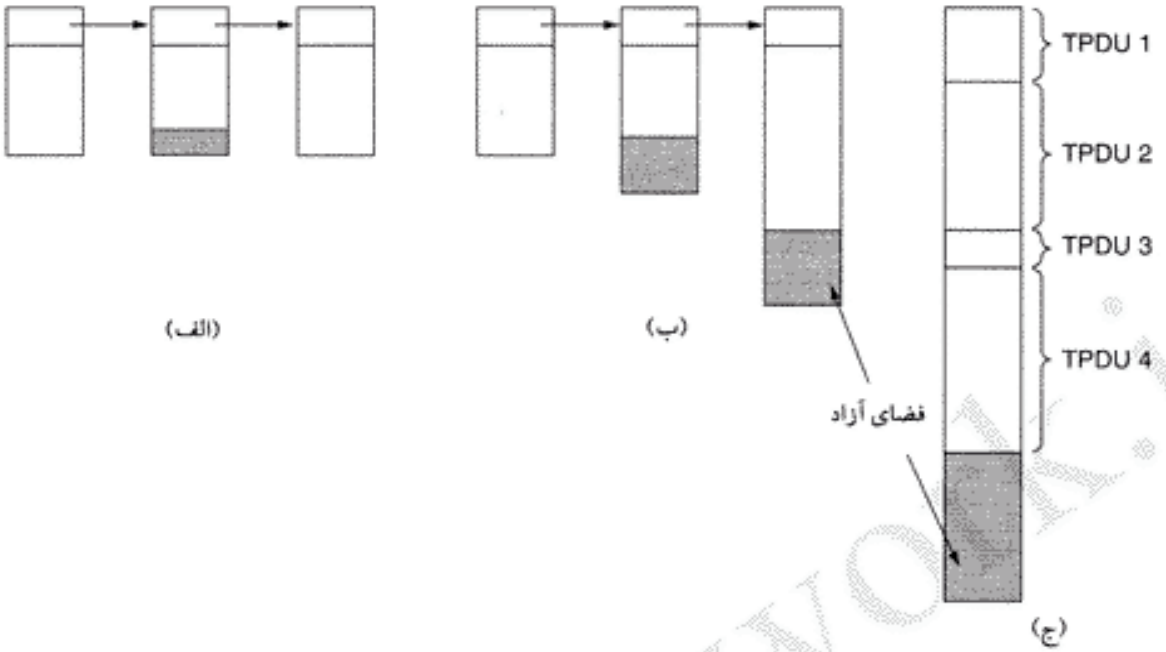
کوتاه سخن آن که، اگر سرویس شبکه غیرقابل اعتماد باشد، فرستنده باید تمام بسته‌های TPDU ارسالی خود را (همانند لایهٔ پیوند داده) بافر کند، ولیکن اگر سرویس شبکه قابل اعتماد باشد می‌توان از راهکارهای بینابینی بهره گرفت. بالاخص اگر گیرنده اطمینان داشته باشد که گیرنده همیشه فضای بافر کافی در اختیار دارد مجبور نیست که نسخه‌ای از TPDUهای ارسالی خود را نگه دارد ولیکن اگر گیرنده نتواند تضمین کند که بسته‌های TPDU ورودی را قطعاً خواهد پذیرفت، فرستنده در هر حال مجبور به بافر کردن آنهاست. در حالت دوم، فرستنده نمی‌تواند به پیغامهای اعلام وصول بسته‌ها (Ack) اعتماد کند چرا که پیغامهای اعلام وصول فقط بدین معناست که بسته‌ها رسیده‌اند نه آن که پذیرفته شده‌اند.[1] در ادامه باز هم به این نکتهٔ مهم باز خواهیم گشت.

حتی اگر گیرنده موافق بافر کردن بسته‌ها باشد باز هم سؤالی باقی می‌ماند و آن هم حجم بافر مورد نیاز آن است. اگر اغلب TPDUها اندازه‌ای تقریباً مساوی داشته باشند، طبیعی آنست که یک فضای حافظهٔ بزرگ با تعدادی بافر هم اندازه ایجاد نماییم (یک بسته در هر بافر). شکل ۶-۱۵-الف این مفهوم را به تصویر کشیده است. ولیکن اگر اندازهٔ بسته‌های TPDU تفاوت فاحش داشته باشند (مثلاً از چند بایت تایپ شده در ترمینال ـ مثلاً Telnet ـ گرفته تا هزاران کاراکتر در حین انتقال فایل)، استفاده از بافرهایی با طول ثابت مشکل‌ساز خواهد شد: از یک طرف اگر اندازهٔ بافرها را معادل با اندازهٔ بزرگترین بستهٔ TPDU در نظر بگیریم، برای بسته‌های کوچک فضای حافظه هدر خواهد رفت. از طرف دیگر، اگر اندازهٔ بافر کمتر از حداکثر طول بسته‌های TPDU انتخاب شود برای بسته‌های بزرگ به چندین بافر نیاز خواهد بود و به پیچیدگی روش می‌انجامد.

راهکار دیگر برای حل مسئلهٔ طول بافر آنست که از بافرهایی با طول متغیر بهره بگیریم. این مفهوم در شکل ۶-۱۵-ب نشان داده شده است. مزیت این روش، استفادهٔ مفید از فضای حافظه است ولیکن به بهای پیچیدگی مکانیزمهای مدیریت بافر تمام می‌شود. راه حل دیگر آن است که به ازای هر اتصال، یک بافر چرخه‌ای بزرگ (Circular Buffer) اختصاص بدهیم. (به شکل ۶-۱۵-ج دقت نمایید.) در این سیستم از حافظه، بخوبی استفاده می‌شود و در اتصالی که حجم مبادلهٔ بار سنگین است کارآیی خوبی دارد ولی برای اتصالی که حجم مبادلهٔ بار کمی

---

۱. ممکن است بسته‌ای به سلامت در ماشین گیرنده دریافت شود ولی پروسه‌ای که صاحب آن است به هر دلیل آن را نپذیرد. ـم



شکل ۶-۱۵. (الف) بافرهای زنجیره‌ای با طول ثابت. (ب) بافرهای زنجیره‌ای با طول متغیر. (ج) یک بافر زنجیره‌ای بزرگ به ازای هر اتصال.

دارد ضعیف عمل می‌کند.

حالت بهینه و متعادل در بافرسازی بسته‌ها در سمت گیرنده و فرستنده به نوع ترافیکی بستگی دارد که از طریق هر اتصال [بین دو پروسه] مبادله می‌شود. برای ترافیکهای انفجاری (Bursty) که به پهنای باند کمی احتیاج دارند (مثل داده‌هایی که از طریق یک ترمینال محاوره‌ای تولید و ارسال می‌شود) بهتر آن است که هیچ بافری از قبل اختصاص داده نشود و در عوض بافر مورد نیاز در هر دو سمت (گیرنده و فرستنده) به صورت پویا و برحسب نیاز تخصیص داده شود. (Dynamic Allocation) از آنجایی که در تخصیص پویا، فرستنده اطمینان ندارد که گیرنده قادر به تخصیص حافظه لازم خواهد بود لذا فرستنده باید نسخه‌ای از هر بستهٔ TPDU ارسالی خود را (مادامی که دریافت آنها تصدیق نشده)، در بافر نگاه دارد. از طرف دیگر برای انتقال فایل یا هر کاربرد دیگری که به پهنای باند بالا نیاز دارد بهتر آن است که گیرنده ، یک پنجرهٔ کامل از بافرها را از قبل رزرو کند تا داده‌ها بتوانند با حداکثر نرخ ممکن جریان داشته باشند.[1] بنابراین برای ترافیک انفجاری ولی با پهنای باند کم بهتر است که فرآیند بافرسازی به نحو جدی در سمت فرستنده انجام شود ولی برای ترافیک یکنواخت با پهنای باند بالا بافرسازی در سمت گیرنده مفیدتر خواهد بود.

وقتی اتصالی باز یا بسته می‌شود یا الگوی ترافیک تغییر می‌کند، گیرنده و فرستنده ملزم به تنظیم خودکار و پویای فضای بافرهای خود هستند. در نتیجه پروتکل لایهٔ انتقال باید این امکان را فراهم کند که ماشین فرستنده، فضای بافر مورد نیاز را در ماشین سمت مقابل خود، سفارش داده و رزرو نماید. بافرها را می‌توان برای هر اتصال بطور مجزا اختصاص داد یا آن که بافرها به صورت یکجا و برای کل اتصالاتی که بین دو ماشین برقرار می‌شود، رزرو گردد. همچنین در سمت مقابل، گیرنده‌ای که از وضعیت بافر خود آگاه است (ولی حجم ترافیک عرضه شده توسط ماشین مقابل را نمی‌داند) می‌تواند به فرستنده اعلام کند که مثلاً: «من به تعداد x بافر برای شما کنار گذاشته‌ام».

---

۱. تا بدلیل مشکلاتی که در حین تخصیص پویای حافظه رخ می‌دهد جریان داده‌ها ناگزیر به قطع موقت نشود. -م

هرگاه تعداد اتصالات باز افزایش یابد ممکن است هر یک از طرفین بدلیل محدودیت حافظه مجبور به کاهش حجم بافر تخصیص یافته شوند، لذا پروتکل لایهٔ انتقال باید از چنین قابلیتی برخوردار باشد.

روش عمومی و عقلانی مدیریت پویای بافرها آن است که مسئله بافرسازی را از مسئلهٔ اعلام وصول بسته‌ها (Acknowledgements) تفکیک نماییم.[1] (برخلاف پروتکل پنجرهٔ لغزان که در فصل سوم تشریح شد.) در نتیجه مدیریت پویای بافرها مستلزم داشتن پنجره‌ای با طول متغیر است. در ابتدا فرستنده براساس پیش‌بینی‌های اولیه میزان بافر مورد نیاز خود را اعلام می‌کند. گیرنده تا حدی که برایش مقدور است حجم خواسته شده را اختصاص می‌دهد. هر وقت فرستنده یک TPDU ارسال کرد، حجم آن را از میزان فضای اختصاص داده شده کم می‌کند و هر گاه میزان این فضا به صفر رسید، ارسال را متوقف می‌نماید. گیرنده نیز اعلام وصول بسته‌ها (Ackها) و همچنین فضای بافر موجود خود را در ترافیک برگشتی (Reverse Traffic) اعلام می‌کند.

شکل ۶-۱۶ مثالی از مدیریت پویای پنجره را در زیرشبکه‌ای دیتاگرام نشان می‌دهد که در آن شماره‌های ترتیب ۴ بیتی هستند. فرض کنید که اطلاعات لازم برای تخصیص بافر در بسته‌های جداگانهٔ TPDU ارسال گردد و در ترافیک برگشتی، جاسازی (Piggyback) نشود. در ابتدا، A هشت بافر تقاضا می‌دهد ولی فقط با چهار بافر موافقت می‌شود. A سه بستهٔ TPDU ارسال می‌کند که سومین آنها از بین می‌رود. [به سطرهای ۱ تا ۵ از شکل ۶-۱۶ نگاه کنید.] در بستهٔ TPDU ششم دریافت بسته‌های ۰ و ۱ تأیید می‌شود و A اجازه می‌یابد تا بافرهای متعلق به این بسته‌ها را آزاد کند [سطر ششم از شکل]. عبارت <ack=1,buf=3> بدین معناست که بسته‌ها تا شمارهٔ ۱ به درستی دریافت شده‌اند و فرستنده باید بسته‌های از ۲ به بعد را ارسال کند؛ در ضمن buf=3 تعداد بافرها را مشخص می‌کند و طبعاً فرستنده مجاز به ارسال حداکثر سه بستهٔ TPDU است (یعنی بسته‌های ۲، ۳، ۴). A می‌داند که بستهٔ شمارهٔ ۲ را قبلاً ارسال کرده، فلذا گمان می‌کند که باید بسته‌های ۳ و ۴ را بفرستد و این کار را انجام می‌دهد. در این نقطه A متوقف می‌شود و صبر می‌کند تا فضای بافر طرف مقابل آزاد شده و به او اعلام شود. در سطر نهم از شکل، می‌بینیم که در اثر انقضای مهلت، بستهٔ دوم (که قبلاً از بین رفته) از نو ارسال شده است. (البته ممکن بود انقضای مهلت تایمر زمانی رخ بدهد که به دلیل عدم وجود فضای بافر، فرستنده متوقف شده باشد.) در سطر دهم از شکل، B دریافت تمام بسته‌های TPDU تا شماره ۴ را اعلام می‌کند [یعنی بسته‌های ۲ و ۳ و ۴ به سلامت رسیده‌اند] ولیکن کماکان به A اجازهٔ ادامهٔ ارسال را نمی‌دهد [چون با اعلام buf=0 فضای بافر خود را صفر گزارش کرده است و A اجازه ندارد چیزی بفرستد.] به خاطر داشته باشید که چنین وضعیتی برای «پروتکل پنجرهٔ ثابت» (Fixed Window Protocol) هرگز پیش نمی‌آید [چرا که در آنجا اعلام وصول فریمها به منزلهٔ آزاد شدن فضای بافر نیز تلقی می‌شود.] در سطر یازدهم، یک بستهٔ TPDU از B به A ارسال شده و ضمن اعلام وجود یک بافر خالی، به A اجازهٔ ادامهٔ ارسال می‌دهد.

روشهای تخصیص بافر همانند روش مثال فوق، مستعد بروز مشکلاتی هستند که در اثر از بین رفتن بسته‌های کنترلی (Control TPDU) در زیرشبکه‌های دیتاگرام، رخ می‌دهند. به سطر شانزدهم از شکل ۶-۱۶ دقت نمائید: B چهار بافر آزاد برای A اختصاص داده و آن را در یک بستهٔ کنترلی به سوی A فرستاده ولیکن این بسته از بین رفته است. از آنجایی که بسته‌های کنترلی شماره‌گذاری نشده‌اند و هیچ تایمری جهت ارسال مجدد ندارند فلذا A

---

۱. یعنی دریافت Ack فقط بیانگر آنست که داده‌های ارسالی بسلامت رسیده‌اند ولی نباید بدین معنا تلقی شود که فرستنده حق ارسال بسته‌های بعدی را دارد چراکه ممکن است بسته‌ها هنوز تحویل پروسهٔ کاربردی نشده باشد و هیچ بافر خالی دیگر برای دریافت بستهٔ بعدی موجود نباشد. -م

	A	پیام	B	توضیح
1	→	< request 8 buffers>	→	A wants 8 buffers
2	←	<ack = 15, buf = 4>	←	B grants messages 0-3 only
3	→	<seq = 0, data = m0>	→	A has 3 buffers left now
4	→	<seq = 1, data = m1>	→	A has 2 buffers left now
5	→	<seq = 2, data = m2>	•••	Message lost but A thinks it has 1 left
6	←	<ack = 1, buf = 3>	←	B acknowledges 0 and 1, permits 2-4
7	→	<seq = 3, data = m3>	→	A has 1 buffer left
8	→	<seq = 4, data = m4>	→	A has 0 buffers left, and must stop
9	→	<seq = 2, data = m2>	→	A times out and retransmits
10	←	<ack = 4, buf = 0>	←	Everything acknowledged, but A still blocked
11	←	<ack = 4, buf = 1>	←	A may now send 5
12	←	<ack = 4, buf = 2>	←	B found a new buffer somewhere
13	→	<seq = 5, data = m5>	→	A has 1 buffer left
14	→	<seq = 6, data = m6>	→	A is now blocked again
15	←	<ack = 6, buf = 0>	←	A is still blocked
16	•••	<ack = 6, buf = 4>	←	Potential deadlock

شکل ۶-۱۶. تخصیص بافر بصورت پویا. (فلشها جهت ارسال را مشخص می‌کنند. علامت ... نشانگر یک TPDU از دست رفته، می‌باشد.)

در یک بن‌بست (Deadlock) قرار می‌گیرد.[1] برای پیشگیری از بروز چنین وضعیتی، هر ماشین باید بطور متناوب بسته‌های کنترلی TPDU (شامل وضعیت بافرها و شمارهٔ Ack) را بر روی هر اتصال بیکار ارسال نماید. بدین نحو، دیر یا زود بن‌بست شکسته خواهد شد.

تا اینجا بطور ضمنی فرض کرده‌ایم که تنها محدودیتی که بر روی نرخ ارسال داده‌های فرستنده تحمیل می‌شود، فضای بافر موجود در گیرنده است.[2] در حالی که با روند رو به کاهش قیمت حافظه، امکان آن فراهم شده که هر ماشین میزبان به حجم بسیار انبوه حافظه مجهز گردد و بدین ترتیب کمبود حافظه به ندرت رخ داده و مشکل بافرها حل می‌شود.

وقتی فضای بافر محدودیتی بر روی حداکثر میزان جریان اعمال نکند گلوگاه دیگری بروز می‌کند: «**ظرفیت حمل زیرشبکه**»[3]. اگر مسیریابهای مجاور قادر به مبادلهٔ حداکثر x بسته در هر ثانیه باشند و در مجموع k مسیر مستقل و مجزا بین یک زوج ماشین وجود داشته باشد، این دو ماشین به هیچ روشی نمی‌توانند بیش از $k.x$ بستهٔ TPDU (در هر ثانیه) مبادله نمایند و فضای بافر موجود در هر یک از طرفین در این مقدار تاثیری ندارد. اگر فرستنده بار سنگینی را به شبکه تزریق کند (به عبارتی بسته‌های TPDU را با سرعتی بیش از $k.x$ TPDU/sec ارسال کند)، زیرشبکه با ازدحام مواجه می‌شود، زیرا قادر نیست بسته‌ها را با همان سرعتی که دریافت می‌کند، تحویل بدهد و بدین ترتیب بخشی از آنها از بین می‌روند.

---

۱. زیرا B با خود می‌اندیشد که چون به A گفته که بافر خالی دارد احتمالاً A داده‌ای برای ارسال نداشته و A هم با خود می‌اندیشد که شاید بافرهای B هنوز خالی نشده است و بدین نحو یک دور باطل ایجاد می‌شود. -م

۲. ازین به بعد فرض بر آنست که گیرنده به طور متناوب فضای بافر خود را به فرستنده اعلام می‌کند و فرستنده ملزم به ارسال داده متناسب با این فضا است. -م

۳. Network's Carrying Capacity

در اینجا به مکانیزمی نیاز داریم که مبتنی بر ظرفیت حمل زیرشبکه، جریان بسته‌ها را کنترل کند نه براساس ظرفیت بافر در گیرنده. روشن است که مکانیزم کنترل جریان بایستی در سمت فرستنده اعمال گردد تا بسته‌های بیهوده‌ای که دریافت آنها تصدیق نشده، متوالیاً ارسال و در زیرشبکه سرگردان نشوند.[۱] «بلسنس» (Belsnes) در سال ۱۹۷۵ با استفاده از «پروتکل پنجرهٔ لغزان» (Sliding Window) روشی پیشنهاد کرد که در آن فرستنده برای کنترل جریان، باید اندازهٔ پنجرهٔ خود را (به صورت پویا) به نحوی تنظیم کند که با ظرفیت حمل شبکه متناسب باشد. اگر شبکه بتواند حداکثر c TPDU/sec را حمل نماید و «**زمان گردش**» (شامل زمان انتقال، تأخیر انتشار، تأخیر انتظار در صف، زمان پردازش در گیرنده و زمان برگشت پیام اعلام وصول ـAckـ) نیز r ثانیه فرض شود، ظرفیت پنجرهٔ فرستنده باید معادل $c.r$ باشد. اگر پنجره‌ای با این اندازه انتخاب شود، فرستنده می‌تواند در شرایطی کاملاً عادی و به مثابهٔ یک خط لوله (Pipeline) عمل کند. [بدون آن که بسته‌ای در اثر ازدحام از دست برود.] یک کاهش کوچک در کارآیی شبکه می‌تواند موجب توقف او شود.

برای آن که بتوان اندازهٔ پنجره را به صورت متناوب و خودکار تنظیم کرد، فرستنده می‌تواند بر مقدار این دو پارامتر [یعنی ظرفیت حمل شبکه و زمان گردش] نظارت داشته باشد و اندازهٔ پنجرهٔ خود را براساس آنها تعیین نماید. برای محاسبهٔ ظرفیت حمل شبکه می‌توان به سادگی تعداد بسته‌هایی را که در دورهٔ زمانی مشخص، ارسال و دریافت آنها تصدیق می‌شود، شمرد و حاصل را بر طول زمان تقسیم کرد. در خلال زمان محاسبه، فرستنده باید بسته‌های خود را با حداکثر نرخ ممکن ارسال نماید تا مطمئن شود آنچه که برگشت بسته‌های اعلام وصول (Ack) را محدود کرده ظرفیت حمل شبکه است نه سرعت پایین خودش. زمان ارسال یک بستهٔ TPDU و برگشت پیغام وصول آن (Ack) را می‌توان به دقت اندازه‌گیری کرد. از آنجایی که ظرفیت فعلی شبکه متغیر با زمان است، اندازهٔ پنجره نیز باید به دفعات تنظیم شود تا هر گونه تغییر در ظرفیت شبکه را دنبال کرده و اندازهٔ پنجره را با آن تطبیق بدهد. بعداً خواهیم دید که اینترنت از روشی شبیه به همین الگو استفاده می‌کند.

۵-۲-۶ مالتی‌پلکسینگ (تسهیم)

مالتی‌پلکس کردن چندین محاورهٔ همزمان (Conversation) بر روی اتصالات، مدارات مجازی یا لینکهای فیزیکی، نقش بسیار مهمی در لایه‌های مختلف معماری یک شبکه ایفاء می‌کند.[۲] در لایهٔ انتقال از چندین جهت به عمل مالتی‌پلکس نیاز است. به عنوان مثال اگر یک ماشین میزبان فقط دارای یک آدرس در شبکه باشد[۳]، تمام اتصالات برقرار شده در لایهٔ انتقال مجبور به مالتی‌پلکسینگ هستند. یعنی به راهکاری نیاز است تا هر گاه یک TPDU وارد می‌شود بتوان پروسهٔ تحویل‌گیرندهٔ آنرا، مشخص کرد. به این وضعیت «**مالتی پلکس روبه‌بالا**» (Upward Multiplexing) گفته می‌شود و شمائی از آن در شکل ۶-۱۷-الف نشان داده شده است. در این شکل چهار اتصال ایجاد شده در لایهٔ انتقال، همگی از اتصال مشترکی در لایهٔ شبکه (مثلاً آدرس IP مشترک) بهره گرفته‌اند.[۴]

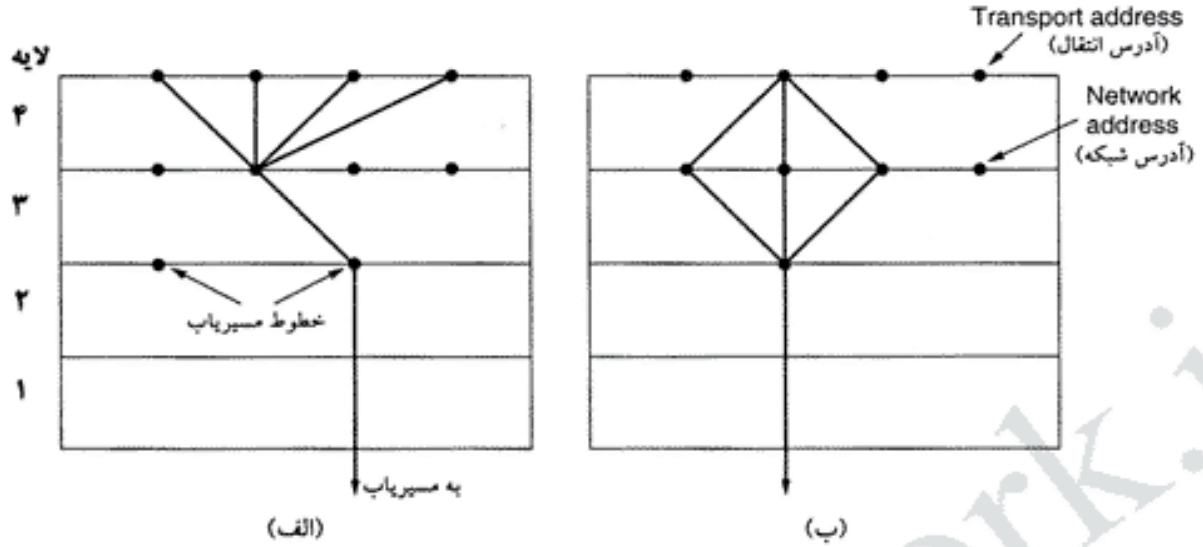
---

۱. بعبارت دیگر اگر ظرفیت حمل شبکه محدود باشد فقط فرستنده می‌تواند جریان ارسال داده‌های خود را متناسب با این ظرفیت تنظیم کند؛ بنابراین هر مکانیزمی بدین منظور، فقط بر روی فرستنده قابل اجراست. ـم

۲. به عبارت دیگر عمل مالتی‌پلکس را می‌توان مکانیزمی جهت استفادهٔ مشترک چندین پروسه از یک لینک واحد، ـ دار مجازی واحد یا آدرس مشترک تعبیر کرد. ـم

۳. عموماً اینگونه است و اغلب ماشینهای اینترنت فقط یک آدرس IP دارند. ـم

۴. به عبارت دیگر تمام داده‌های این چهار پروسه وقتی به لایهٔ شبکه می‌رسند درون بسته‌هایی قرار می‌گیرند که آدرس مبداء همهٔ آنها یکسان است و همگی نهایتاً بر روی یک لینک مشترک ارسال می‌شوند و باید بنحوی در لایهٔ انتقال از یکدیگر تفکیک و به پروسه‌های متناظر خود تحویل داده شوند. فرآیند تفکیک بسته‌ها بین پروسه‌ها در لایهٔ انتقال، «مالتی‌پلکس روبه‌بالا» نام دارد.



شکل ۶-۱۷. (الف) مالتی‌پلکس روبه بالا. (ب) مالتی‌پلکس رو به پایین.

مالتی پلکس در لایهٔ انتقال، از جهت دیگری نیز می‌تواند مفید واقع شود. به عنوان مثال فرض کنید در زیرشبکه‌ای که از درون مبتنی بر مدار مجازی (Virtual Circuit) است، بر روی حداکثر نرخ ارسال هر مدار مجازی، محدودیت گذاشته شده است. اگر کاربر به پهنای باند بیشتری نسبت به نرخ حداکثر هر مدار مجازی نیاز داشته باشد، یک راهکار آن است که چندین اتصال مدار مجازی همزمان در شبکه ایجاد شود و ترافیک داده‌های یک پروسه به نوبت و چرخشی (Round Robin) بر روی این مدارات مجازی توزیع شود. این مفهوم که در شکل ۶-۱۷-ب به تصویر کشیده شده است، **«مالتی‌پلکس روبه‌پایین»** نام دارد. با داشتن k اتصال باز در سطح شبکه (بعبارت دیگر k مدار مجازی همزمان)، پهنای باند مؤثر با ضریب k افزایش می‌یابد. مثالی از مالتی‌پلکس روبه‌پایین را می‌توان کاربران خانگی عنوان کرد که از خط ISDN بهره می‌گیرند. این خط دو اتصال 64kbps دو طرفه را در اختیار می‌گذارد. با استفادهٔ همزمان از این دو اتصال برای وصل به یک شرکت ارائه دهندهٔ خدمات اینترنت و توزیع ترافیک بر روی آنها، پهنای باند مؤثر 128kbps حاصل می‌شود.

۶-۲-۶ جبران از کارافتادگی (Crash Recovery)

اگر ماشینهای میزبان یا مسیریابها مستعد خرابی و از کارافتادگی باشند، موضوع احیاء و برگرداندن آنها به فعالیت طبیعی، مسئلهٔ مهمی است. اگر **«واحدهای انتقال»** (Transport Entities) بطور کلی در درون ماشینهای میزبان مستقر باشند، از کارافتادگی شبکه و مسیریاب به سادگی اصلاح و جبران خواهد شد. اگر چنین شبکه‌ای خدمات دیتاگرام عرضه نماید، «واحدهای انتقال» همیشه آمادگی از بین رفتن بسته‌های TPDU را دارند و روش برخورد با چنین مشکلی را می‌دانند.[1] اگر لایهٔ شبکه خدمات مدار مجازی عرضه کرده باشد، از دست رفتن یک مدار مجازی بدین نحو جبران می‌شود که مدار مجازی جدیدی برقرار شده و از واحد انتقال در ماشین راه دور سؤال می‌گردد که چه TPDUهایی را دریافت کرده و کدام را دریافت ننموده است؛ آنهایی که دریافت نشده‌اند از نو ارسال می‌شوند.

مسئله دردسرآفرین، آنست که ماشینهای میزبان را چگونه پس از خرابی به فعالیت طبیعی بازگردانیم؟ برای آن که دشوار بودن این عمل را نشان بدهیم فرض کنید که یک ماشین میزبان در حال ارسال یک فایل طولانی برای

---

۱. بعبارتی از بین رفتن بسته‌ها در شبکه‌های دیتاگرام امری طبیعی است و نیاز به مکانیزمی اضافی ندارد، لذا خرابی یک کانال یا از کارافتادن موقت مسیریابهای میانی مشکلی برای لایهٔ انتقال ایجاد نمی‌کند. -م

ماشین میزبان دیگر (مثلاً سرویس‌دهندهٔ فایل) به روش «**توقف و انتظار**» (Stop & Wait) است. [یعنی یک قطعه از فایل ارسال شده و منتظر اعلام وصول آن می‌ماند.] لایهٔ انتقال در ماشین سرویس‌دهنده، بسته‌های TPDU را یکی‌یکی به پروسهٔ کاربردی مربوطه تحویل می‌دهد. در اثنای کار، سرویس‌دهنده از کار می‌افتد؛ پس از راه‌اندازی مجدد، وقتی این ماشین فعالیت طبیعی خود را از سر می‌گیرد، جداول او مقداردهی اولیه می‌شوند فلذا او نمی‌داند که دقیقاً در کجای کار این خرابی اتفاق افتاده و طبعاً همه چیز باید از نو تکرار شود.

برای آن که بتوان وضعیت را به حالت قبلی برگرداند، سرویس‌دهنده می‌تواند یک بستهٔ TPDU را به روش پخش فراگیر (Broadcast) برای بقیهٔ ماشینهای میزبان بفرستد و با اعلام آن که از کار افتاده بوده از تمام ماشینهای مشتری (Clients) درخواست کند که وضعیت تمام اتصالات باز خود را [که با او برقرار کرده بوده‌اند] اعلام نمایند. هر مشتری ممکن است یکی از این دو وضعیت را داشته باشد: منتظر یک TPDU باشد (وضعیت S1) منتظر هیچ TPDU نباشد (وضعیت S0). براساس این اطلاعات وضعیت، مشتری می‌تواند در خصوص ارسال مجدد TPDUهای اخیر تصمیم بگیرد.

در نگاه اول این فرآیند ساده به نظر می‌رسد: مشتری باید بسته‌های TPDU اعلام وصول نشده را از نو ارسال نماید (یعنی وضعیت S1). ولیکن بررسی موشکافانهٔ این روش، مشکلات آن را آشکار می‌کند. به عنوان مثال وضعیتی را در نظر بگیرید که واحد انتقال در ماشین سرویس‌دهنده، پیام اعلام وصول (Ack) یک بستهٔ TPDU را ارسال کرده و محتوای آن را به برنامه کاربردی تحویل داده است. نوشتن یک TPDU در یک استریم و سپس ارسال پیغام اعلام وصول دو رخداد کاملاً مجزا است و نمی‌تواند بطور همزمان انجام شود. اگر خرابی ماشین دقیقاً زمانی رخ بدهد که پیغام اعلام وصول بسته، ارسال شده ولی محتوای آن بسته هنوز در استریم متعلق به برنامهٔ کاربردی نوشته نشده ، مشتری پیغام اعلام وصول بسته را دریافت می‌کند و طبعاً در وضعیت S0 قرار می‌گیرد ولیکن محتوای بسته حقیقتاً به برنامهٔ کاربردی نرسیده و ماشین مشتری آن را از نو ارسال نخواهد کرد چرا که به غلط فکر می‌کند که این TPDU دریافت شده است. چنین تصمیمی منجر به از دست رفتن یک TPDU خواهد شد.

در اینجا ممکن است با خود بیندیشید که: «این مشکل را براحتی می‌توان حل کرد. کل کاری که باید انجام شود آنست که واحد انتقال (Transport Entity) بازنویسی و اصلاح شود تا اول بسته را بنویسد و سپس آن را اعلام وصول کند!» ولیکن تجسم کنید که ابتدا عمل نوشتن در استریم برنامهٔ کاربردی انجام شود ولی دقیقاً قبل از ارسال پیغام اعلام وصول، ماشین از کار بیفتد. این رخداد منجر به تولید بسته‌های TPDU تکراری شده و چون تکراری بودن آنها کشف نمی‌شود در استریم خروجی پروسهٔ کاربردی سرویس‌دهنده نوشته خواهد شد.

فارغ از آنکه برنامهٔ سرویس‌دهنده و مشتری چگونه برنامه‌نویسی شده‌اند همیشه وضعیتی وجود دارد که پروتکل در حین برگشت به حالت طبیعی (پس از خرابی ماشین) با شکست جدی مواجه می‌شود. سرویس‌دهنده را می‌توان به یکی از دو روش ذیل برنامه‌نویسی کرد: اول پیغام اعلام وصول بسته را بفرستد یا اول محتوای بسته را در استریم پروسهٔ کاربردی بنویسد. برنامهٔ مشتری را می‌توان به چهار روش نوشت: (۱) همیشه آخرین TPDU ارسالی خود را مجدداً بفرستد. (۲) هیچگاه آخرین بستهٔ TPDU را نفرستد. (۳) فقط وقتی بستهٔ آخر را مجدداً ارسال کند که در وضعیت S0 باشد. (۴) بستهٔ آخر را فقط وقتی از نو ارسال کند که در وضعیت S1 باشد. ترکیب اینها هشت حالت مختلف را پدید می‌آورد ولیکن به گونه‌ای که خواهیم دید برای هر ترکیب، مجموعه‌ای از رخدادها منجر به شکست پروتکل می‌شود.

در سرویس‌دهنده وقوع سه رخداد محتمل است: (۱) ارسال پیام Ack (A) (۲) نوشتن در پروسهٔ خروجی (W) (۳) از کارافتادگی (C). این سه رخداد می‌تواند به شش ترتیب مختلف اتفاق بیفتد: AC(W) ، AWC ،

C(AW) ، C(WA) ، WAC و WC(A). پرانتزها نمایانگر آن هستند که وقتی سیستم از کار می‌افتد طبعاً مراحل بعدی که درون پرانتز نشان داده شده‌اند، نمی‌تواند ادامه پیدا کند. (یعنی وقتی سیستم خراب شد، کار تمام است و رخدادهای درون پرانتز فرصت وقوع پیدا نمی‌کنند.) شکل ۶-۱۸ هشت ترکیب مختلف ناشی از عملکرد سرویس‌دهنده و مشتری و ترکیبات معتبر آنها نشان داده شده‌اند. دقت کنید که در هر استراتژی، دنباله‌ای از رخدادها می‌تواند منجر به شکست پروتکل شود. به عنوان مثال، اگر ماشین مشتری، همیشه آخرین بستهٔ ارسالی خود را ارسال کند، استراتژی AWC[1] منجر می‌شود که بسته‌ای تکراری و غیرقابل تشخیص، دریافت و تحویل پروسه شود، حتی اگر دو رویداد دیگر [یعنی AW] به درستی انجام گردد.

استراتژیهای بکار رفته در ماشین فرستنده	استراتژیهای بکار رفته در ماشین گیرنده					
	First ACK, then write			First write, then ACK		
	AC(W)	AWC	C(AW)	C(WA)	W AC	WC(A)
Always retransmit	OK	DUP	OK	OK	DUP	DUP
Never retransmit	LOST	OK	LOST	LOST	OK	OK
Retransmit in S0	OK	DUP	LOST	LOST	DUP	OK
Retransmit in S1	LOST	OK	OK	OK	OK	DUP

OK = پروتکل بدستی عمل می‌کند.
DUP = پروتکل یک پیام تکراری تولید می‌کند.
LOST = پروتکل پیامی را از دست می‌دهد.

شکل ۶-۱۸. ترکیبات مختلف استراتژیهای اتخاذ شده در سرویس‌دهنده و مشتری.

پیچیده و هوشمندتر کردن پروتکل کمک چندانی به حل مشکل نمی‌کند. حتی اگر مشتری و سرویس‌دهنده، قبل از آنکه سرویس‌دهنده تلاش کند داده‌ها را به پروسهٔ کاربردی تحویل بدهد (یا به عبارت دیگر آنها را بنویسد) چندین بستهٔ TPDU مبادله کنند تا مشتری بداند چه اتفاقی در حال وقوع است، باز هم در صورت از کار افتادن سرویس‌دهنده و برگشت به حالت طبیعی، مشتری نمی‌تواند بفهمد که وقوع خرابی قبل از نوشتن بسته‌ها بوده یا بعد از آن. بنابراین ناگزیر به پذیرش این نتیجه هستیم: در شرایطی که رخدادها بطور همزمان اتفاق نمی‌افتند[2] از کار افتادن ماشین میزبان و بازگشت آن به حالت طبیعی را نمی‌توان بی‌سر و صدا اصلاح کرد و از دید لایه‌های بالاتر مخفی نخواهد ماند.[3]

اگر بخواهیم نتیجه‌گیری فوق را تعمیم بدهیم می‌توان آن را بدین نحو بیان کرد که «**اگر لایهٔ N دچار از کار افتادگی شود، بازیابی و بازگرداندن آن به شرایط طبیعی، تنها در لایهٔ N+1 مقدور است و آن هم مشروط به آنکه لایهٔ بالایی اطلاعات کافی از وضعیت فعلی نگه داشته باشد.**» در بالا اشاره شد که هر گونه خرابی در لایهٔ شبکه می‌تواند توسط لایهٔ انتقال اصلاح و جبران شود به شرط آن که هر یک از طرفین یک اتصال، وضعیت فعلی خود و دیگری را نگه داشته باشند.

---

۱. AWC یعنی ابتدا اعلام وصول بسته، سپس تحویل داده‌ها به پروسه و سپس از کار افتادن ماشین -م

۲. یعنی اول اعلام وصول و بعد تحویل داده‌ها به پروسه انجام می‌شود یا بالعکس ولی نه **همزمان** -م

۳. بدون تمهیدات برنامه‌های کاربری در لایهٔ کاربرد، داده‌هایی که قبل از خرابی ماشین دریافت شده‌اند از درجهٔ اعتبار ساقط است. -م

این مسئله ما را متوجه معنای حقیقی موضوعی می‌کند که اصطلاحاً با عنوان «**تصدیق دریافت داده‌ها به صورت انتهابه‌انتها**» (End-to-End Acknowledgement) معرفی می‌شود. اصولاً پروتکل لایهٔ انتقال «انتهابه‌انتها» است و همانند لایه‌های زیرین به صورت «زنجیره‌ای» عمل نمی‌کند.[1] حال وضعیتی را مدّنظر قرار بدهید که یک کاربر تقاضایی را بر روی یک پایگاه دادهٔ راه دور اعمال می‌کند. فرض کنید واحد انتقال (Transport Entity) به گونه‌ای برنامه‌نویسی شده که ابتدا بسته‌های TPDU را به لایهٔ بالاتر تحویل بدهد و بعد از آن پیغام تصدیق دریافت (Ack) بفرستد. در چنین حالتی حتی بازگشت پیغام Ack به ماشین کاربر، الزاماً به معنای آن نیست که این درخواست واقعاً بر روی بانک اطلاعاتی اعمال و بهنگام‌سازی شده است. یک مکانیزم «تصدیق انتهابه‌انتها» که در آن دریافت Ack به معنای انجام صددرصد کار و عدم دریافت آن به معنای انجام نشدن کار باشد، در عمل دست نیافتنی است. این نکته به تفصیل در مرجع (Saltzer et al. 1984) بحث شده است.

۶-۳ یک پروتکل سادهٔ انتقال

برای تبیین ایده‌هایی که تا اینجا بررسی شدند، در این بخش به مطالعهٔ مبسوط یک مثال عملی از لایهٔ انتقال می‌پردازیم. خدمات اولیه ارائه شده همان توابع اولیهٔ (Primitives) اتصال‌گرا هستند که در شکل ۶-۲ نشان داده شده‌اند. انتخاب این توابع و عملکرد اتصال‌گرای اولیه، مثال ما را بسیار شبیه به پروتکل TCP (ولی ساده‌تر از آن) خواهد نمود.

۶-۳-۱ توابع اولیهٔ ارائه خدمات در مثال فوق

اولین مسئله‌ای که با آن مواجهیم تشریح صحیح و دقیق «عملکردهای اولیهٔ انتقال» است. عمل CONNECT ساده است: ما یک تابع کتابخانه‌ای به نام ***connect*** خواهیم داشت که برای برقراری یک اتصال، می‌توان آنرا با پارامترهای مناسب فراخوانی کرد. این پارامترها عبارتند از شناسهٔ TSAP مبداء (محلی) و شناسهٔ TSAP مقصد (راه دور). پس از فراخوانی، پروسهٔ صدا زننده متوقف می‌شود (به عبارت دیگر معلق می‌گردد) تا واحد انتقال برای برقراری اتصال اقدام کند. اگر برقراری اتصال انجام شود، پروسهٔ صدازننده از حالت توقف خارج شده و می‌تواند شروع به ارسال داده بنماید.

هر گاه پروسه‌ای در سمت سرویس‌دهنده بخواهد قادر به پذیرش تقاضاهای ارتباط باشد، تابع ***listen*** را فراخوانی کرده و TSAP مشخصی را که باید شنود شود، تعیین می‌کند. پروسهٔ صدازنندهٔ این تابع، مادامی که یک پروسهٔ راه دور تقاضای برقراری اتصال با این TSAP را ندهد در حالت توقف باقی خواهد ماند.

دقت کنید که این مدل کاملاً نامتقارن است: یک طرف «غیرفعّال» (Passive) است و با اجرای تابع ***listen*** آنقدر منتظر می‌ماند تا اتفاقی بیفتد [پروسه‌ای ارتباط برقرار کند.]. طرف دیگر فعّال (Active) است و در زمان دلخواه شروع به برقراری اتصال می‌کند. یک سؤال جالب آن است که اگر طرف فعّال زودتر شروع کند چه باید کرد. یک استراتژی آن است که اگر هیچ پروسهٔ در حال شنود به TSAP راه دور وجود نداشته نباشد [یعنی سرویس‌دهنده هنوز اجرا نشده باشد]، تلاش برای برقراری ارتباط ناکام گذاشته شود. استراتژی دیگر آن است که پروسهٔ آغازکننده آنقدر منتظر نگاه داشته شود تا بالاخره پروسه‌ای شروع به گوش دادن به TSAP مورد نظر کند.

---

۱. یعنی مثلاً یک بسته در لایهٔ شبکه ممکن است به صورت زنجیره‌ای دهها بار در مسیریابهای واقع بر مسیر پردازش شوند ولی یک بستهٔ TPDU در مبداء تولید و در مقصد پردازش و مصرف می‌شود لذا تمام رخدادهای اتفاقی در لایه‌های زیرین در مقصد قابل بررسی و اصلاح است. مثلاً اگر یک مسیریاب به ناگاه خراب شود و چند بستهٔ TPDU را با خود از بین ببرد هیچ اتفاقی برای لایهٔ انتقال نمی‌افتد چراکه فرستنده متوجه عدم اعلام وصول آنها شده و پس از انقضای مهلت آنها را از نو ارسال می‌کند. ـم

در مثالمان از یک حالت میانه استفاده کرده‌ایم: تقاضای برقراری اتصال برای یک مدت زمان مشخص منتظر نگه داشته می‌شود تا اگر قبل از انقضای مهلت، پروسه‌ای در سمت مقابل تابع ***listen*** را فراخوانی کرد، اتصال برقرار شود. در غیر این صورت تقاضای برقراری اتصال ردّ شده و پروسه صدازننده از حالت توقف خارج و پیغام خطایی به او برگردانده می‌شود.

برای خاتمه دادن به یک اتصال، از تابع کتابخانه‌ای ***disconnect*** بهره گرفته‌ایم. وقتی هر دو طرف ، ارتباط را قطع کردند، اتصال خاتمه می‌یابد. به عبارت دیگر از مدل «متقارن» برای ختم ارتباط استفاده نموده‌ایم.

مسئلهٔ انتقال داده دقیقاً شبیه به برقراری اتصال است: عمل ارسال ماهیّتی «فعال» و فرآیند دریافت ماهیّتی «غیرفعال» دارد؛ لذا برای انتقال داده از راه حل مشابه با برقراری اتصال استفاده خواهیم کرد: فراخوانی تابع ***send*** داده‌ها را فوراً ارسال می‌کند در حالی که فراخوانی تابع ***receive*** غیرفعال بوده و تا زمانی که یک بستهٔ TPDU دریافت نشود منجر به توقف پروسه می‌شود.

بدین ترتیب مجموعهٔ سرویسهای ارائه شدهٔ ما، شامل پنج عملکرد اولیه (تابع پایه) است: (۱) CONNECT (۲) LISTEN (۳) DISCONNECT (۴) SEND (۵) RECEIVE. به ازای هر یک از این پنج عملکرد اولیه دقیقاً یک تابع کتابخانه‌ای خاصّ وجود دارد که سرویس مربوطه را پیاده کرده است. پارامترهای این توابع کتابخانه‌ای به صورت زیر هستند:

```
connum = LISTEN(local)
connum = CONNECT(local,remote)
status = SEND(connum,buffer,bytes)
status = RECEIVE(connum,buffer,bytes)
status = DISCONNECT(connnum)
```

تابع اولیهٔ LISTEN اعلام می‌کند که پروسهٔ صدازنندهٔ آن علاقمند به پذیرش آن دسته از تقاضاهای برقراری اتصال است که با شناسهٔ TSAP مشخص شده، وارد می‌شوند. پروسه‌ای که این تابع را فراخوانی کرده، مادامیکه که یک پروسهٔ راه دور سعی در برقراری اتصال نکند، متوقف و منتظر خواهد ماند. در اینجا هیچ مهلتی تعیین نشده است.

تابع اولیهٔ CONNECT دو پارامتر را دریافت می‌کند: (۱) شناسهٔ TSAP محلی (Local TSAP) (که در تعریف تابع با نام local ظاهر شده است.) (۲) شناسهٔ TSAP راه دور (با نام remote)؛ سپس سعی می‌کند یک اتصال بین این دو برقرار نماید. اگر این کار با موفقیت انجام شد یک عدد نامنفی در متغیر connum برمی‌گرداند تا برای مشخص کردن هویت اتصال در فراخوانیهای بعدی مورد استفاده قرار بگیرد. اگر ایجاد اتصال با شکست مواجه شود، عددی منفی در connum برمی‌گرداند. در مدل سادهٔ ما، هر TSAP می‌تواند فقط برای ایجاد یک اتصال مورد استفاده قرار بگیرد، لذا یک دلیل موجه برای شکست در برقراری اتصال آن است که آدرس انتقال (یعنی TSAP مورد نظر) در حال استفاده توسط پروسهٔ دیگر باشد. برخی از دلایل دیگر عبارتند از: خاموش بودن ماشین راه دور، آدرس محلی نامعتبر، آدرس راه دور نامعتبر.

تابع اولیهٔ SEND محتویات بافر را به عنوان یک پیام، بر روی اتصال مشخص شده ارسال می‌کند؛ البته در صورت نیاز داده‌ها را در چند قطعه (چند بستهٔ TPDU) می‌فرستد. خطاهای احتمالی در متغیر status برگردانده می‌شود. برخی از علل خطاهای احتمالی عبارتند از: عدم وجود اتصال، نامعتبر بودن آدرس بافر یا منفی بودن پارامتر تعداد بایتی که باید ارسال شود (یعنی منفی بودن پارامتر bytes).

تابع اولیهٔ RECEIVE ، مشخص‌کنندهٔ آن است که پروسهٔ صدازنندهٔ آن تمایل به دریافت داده دارد. اندازهٔ پیام دریافتی در پارامتر bytes قرار داده می‌شود. اگر پروسهٔ راه دور اتصال را قطع کرده باشد یا آدرس بافر نامعتبر (مثلاً

آدرسی خارج از فضای برنامهٔ کاربر) باشد، یک کد خطا در متغیر status برگردانده می‌شود تا ماهیت خطا را مشخص نماید.

تابع اولیه DISCONNECT اتصال مورد نظر را قطع می‌نماید. پارامتر connum ، مشخص می‌کند که کدام اتصال باید بسته شود. علل خطاهای احتمالی عبارتند از: connum شمارهٔ اتصالی متعلق به پروسه‌ای دیگر باشد یا شناسهٔ اتصال نامعتبر باشد. بهر حال در متغیر status ، کدی برگردانده می‌شود که مقدار صفر به معنای موفقیت و مقدار غیرصفر به معنای کد مشخصهٔ خطاست.

۶-۳-۲ واحد انتقال در مثال فوق

قبل از آن که کدهای برنامهٔ «واحد انتقال» (Transport Entity) را بررسی نماییم لطفاً به خاطر بسپارید که این مثال شبیه به مثالهایی که در فصل سوم عرضه شدند جنبهٔ آموزشی دارد و طرحی جدّی محسوب نمی‌شود. برای پرهیز از پیچیدگی، بسیاری از جزئیات فنی (مثل کنترل دقیق و گستردهٔ خطا) که در سیستمهای واقعی مورد نیاز هستند، حذف شده‌اند.

لایهٔ انتقال برای ارسال یا دریافت هر TPDU از سرویسهایی که لایهٔ شبکه در قالب توابع اولیه در اختیار گذاشته، بهره می‌گیرد. در این مثال ابتدا بایستی نوع سرویس لایهٔ شبکه و توابع اولیه مورد استفاده مشخص شود. یک گزینهٔ آن است که از سرویس نامطمئن دیتاگرام بهره گرفته شود. برای آن که مثالمان ساده باشد ما این گزینه را انتخاب نمی‌کنیم. در سرویس دیتاگرام کد برنامهٔ لایهٔ انتقال بسیار طولانی و پیچیده خواهد شد، زیرا بخش بزرگی از این برنامه صرف مدیریت بسته‌های از بین رفته و مسائل ناشی از تأخیر خواهد شد. مضاف بر این، بسیاری از ایده‌های مرتبط با این موضوعات قبلاً به تفصیل در فصل سوم بررسی شده‌اند.

در عوض، ترجیح داده‌ایم از سرویسهای مطمئن یک شبکهٔ اتصال‌گرا بهره بگیریم. بدین ترتیب قادر خواهیم بود که بر روی موضوعاتی از لایهٔ انتقال متمرکز شویم که در لایه‌های زیرین مطرح نمی‌شوند. این موضوعات عبارتند از: برقراری اتصال، خاتمه دادن به اتصال، مدیریت اعتبار (Credit Management). سرویسهای یک لایهٔ انتقال ساده که بر روی شبکهٔ ATM به کار گرفته می‌شود، شبیه به مثال ما خواهد بود.

«واحد انتقال» عموماً بخشی از سیستم عامل ماشین میزبان است؛ گاهی هم در قالب یک مجموعه از توابع کتابخانه‌ای ارائه و در فضای آدرس برنامهٔ کاربر اجرا می‌شود. برای سادگی ، برنامهٔ مثال ما در قالب یک مجموعه از توابع کتابخانه‌ای عرضه شده است ولی با تغییرات اندک می‌توان آن را تبدیل به بخشی از سیستم عامل کرد.

به هر حال اشاره به این نکته خالی از لطف نیست که «واحد انتقال» در این مثال یک بخش کاملاً مستقل نیست و بخشی از پروسهٔ کاربردی محسوب می‌شود. خصوصاً وقتی کاربر یک تابع اولیه نظیر LISTEN (که برنامه را بلوکه می‌کند) اجرا نماید ، کل واحد انتقال نیز متوقف می‌شود. اینگونه طراحی برای ماشینهایی مناسب است که تک کاربره و تک پروسه‌ای هستند، در حالی که در ماشینی با چندین کاربر و پروسه، طبعاً نیاز به یک واحد انتقال مستقل داریم تا از پروسه‌های کاربران مجزا باشد.

تعامل با لایهٔ شبکه از طریق پروسیجرهای to_net و from_net انجام می‌شود. هر یک از این پروسیجرها، شش پارامتر دارند: اولین پارامتر شناسهٔ اتصال مدار مجازی است. پارامترهای بعدی، بیتهای Q و M هستند که هر کدام به ۱ تنظیم شده باشند، به ترتیب مشخص می‌کنند که (۱) پیام از نوع کنترلی است (۲) بخشی از داده‌های این پیام در بسته‌های بعدی می‌آید. پس از آن نوع بسته مشخص می‌شود که می‌تواند یکی از شش نوع معرفی شده در جدول ۶-۱۹ باشد. سپس اشاره‌گری که آدرس محل شروع داده‌ها را مشخص می‌کند و نهایتاً یک عدد صحیح می‌آید که تعداد بایتهای داده را تعیین می‌نماید.

وقتی پروسیجر to_net فراخوانی می‌شود، «واحد انتقال» تمام این پارامترها را پس از مقداردهی در اختیار لایهٔ

نام بسته لایه شبکه	توصیف
CALL REQUEST	به منظور ایجاد اتصال ارسال می‌شود.
CALL ACCEPTED	پاسخ بسته CALL REQUEST (بمعنای موافقت با برقراری اتصال).
CLEAR REQUEST	به منظور قطع اتصال ارسال می‌شود.
CLEAR CONFIRMATION	پاسخ بسته CLEAR REQUEST (بمعنای موافقت با ختم اتصال).
DATA	برای انتقال داده بکار می‌رود.
CREDIT	یک بسته کنترلی برای مدیریت پنجره

شکل ۶-۱۹. بسته‌هائی از لایهٔ شبکه که در مثالمان از آنها بهره گرفته‌ایم.

شبکه قرار می‌دهد؛ برعکس، با فراخوانی from_net ، لایهٔ شبکه این پارامترها را از بطن بسته‌های ورودی بیرون کشیده و تحویل واحد انتقال می‌دهد. وقتی اطلاعات لازم به صورت پارامترهای پروسیجر به لایهٔ شبکه تحویل می‌شود (به جای آن که این اطلاعات با تحویل بسته‌های واقعی رد و بدل گردد)، لایهٔ انتقال از درگیری با جزئیات پروتکل لایهٔ شبکه دور نگاه داشته می‌شود.[1] هرگاه «**پنجرهٔ لغزان مدار مجازی**»[2] پر شده باشد و واحد انتقال سعی در ارسال بسته‌ای کند ، اجرای to_net آنرا آنقدر معطل نگاه خواهد داشت تا فضای کافی در پنجره آزاد گردد. این مکانیزم با استفاده از دستوراتی شبیه به enanble_transport_layer یا disable_transprot_layer (که در فصل سوم مشابهشان را بررسی کردیم) در لایهٔ شبکه انجام می‌گیرد و بطور کامل از دید واحد انتقال پنهان خواهد ماند. مدیریت پنجره‌ها نیز در لایهٔ شبکه انجام می‌شود.

اضافه بر مکانیزم «**تعلیق**» (suspension) فوق‌الذکر [که توسط لایهٔ شبکه به لایهٔ انتقال تحمیل می‌شود و لایهٔ انتقال نقشی در آن ندارد]، در لایهٔ انتقال دو پروسیجر ***sleep*** و ***wakeup*** (که در فهرست نیامده‌اند) تعریف شده که به منظور ایجاد وقفه در واحد انتقال، مورد استفاده قرار می‌گیرند. پروسیجر ***sleep*** زمانی فراخوانی می‌شود که واحد انتقال منطقاً در انتظار وقوع یک رخداد (مثلاً دریافت یک بسته) باشد. پس از فراخوانی ***sleep***، اجرای واحد انتقال (و به تَبَع آن پروسهٔ کاربردی) متوقف می‌شود.

کُد برنامهٔ «واحد انتقال» در شکل ۶-۲۰ نشان داده شده است. هر اتصال همیشه در یکی از هفت وضعیت زیر قرار دارد:

۱. **وضعیت** IDLE : هنوز هیچ اتصالی برقرار نشده است.
۲. **وضعیت** WAITING : تابع CONNECT اجرا و تقاضای برقراری اتصال (CALL REQUEST) ارسال شده است.
۳. **وضعیت** QUEUED : تقاضای CALL REQUEST دریافت شده ولیکن هنوز LISTEN نشده است.
۴. **وضعیت** ESTABLISHED : اتصال برقرار شده است.
۵. **وضعیت** SENDING : کاربر منتظر دریافت مجوز ارسال یک بسته است.
۶. **وضعیت** RECEIVING : تابع RECEIVE اجرا و در حال دریافت بسته است.
۷. **وضعیت** DISCONNCTING : تابع DISCONNECT جهت قطع اتصال بصورت محلّی اجرا شده است.

---

۱. یعنی بجای آنکه واحد انتقال مستقیماً خودش بسته‌هایی را که قرار است بر روی شبکه ارسال شوند، تولید کند پارامترهای لازم را به لایهٔ شبکه تحویل می‌دهد تا اینکار در لایهٔ شبکه انجام شود. -م

۲. Virtual Circuit's Sliding Window

```
#define MAX_CONN 32                 /* max number of simultaneous connections */
#define MAX_MSG_SIZE 8192           /* largest message in bytes */
#define MAX_PKT_SIZE 512            /* largest packet in bytes */
#define TIMEOUT 20
#define CRED 1
#define OK 0

#define ERR_FULL -1
#define ERR_REJECT -2
#define ERR_CLOSED -3
#define LOW_ERR -3

typedef int transport_address;
typedef enum {CALL_REQ,CALL_ACC,CLEAR_REQ,CLEAR_CONF,DATA_PKT,CREDIT} pkt_type;
typedef enum {IDLE,WAITING,QUEUED,ESTABLISHED,SENDING,RECEIVING,DISCONN}  cstate;

/* Global variables. */
transport_address listen_address;   /* local address being listened to */
int listen_conn;                    /* connection identifier for listen */
unsigned char data[MAX_PKT_SIZE];   /* scratch area for packet data */

struct conn {
  transport_address local_address, remote_address;
  cstate state;                     /* state of this connection */
  unsigned char *user_buf_addr;     /* pointer to receive buffer */
  int byte_count;                   /* send/receive count */
  int clr_req_received;             /* set when CLEAR_REQ packet received */
  int timer;                        /* used to time out CALL_REQ packets */
  int credits;                      /* number of messages that may be sent */
} conn[MAX_CONN + 1];               /* slot 0 is not used */

void sleep(void);                   /* prototypes */
void wakeup(void);
void to_net(int cid, int q, int m, pkt_type pt, unsigned char *p, int bytes);
void from_net(int *cid, int *q, int *m, pkt_type *pt, unsigned char *p, int *bytes);

int listen(transport_address t)
{ /* User wants to listen for a connection. See if CALL_REQ has already arrived. */
  int i, found = 0;

  for (i = 1; i <= MAX_CONN; i++)   /* search the table for CALL_REQ */
      if (conn[i].state == QUEUED && conn[i].local_address == t) {
            found = i;
            break;
      }

  if (found == 0) {
      /* No CALL_REQ is waiting.  Go to sleep until arrival or timeout. */
      listen_address = t;  sleep();  i = listen_conn ;
  }
  conn[i].state = ESTABLISHED;      /* connection is ESTABLISHED */
  conn[i].timer = 0;                /* timer is not used */
```

```
 listen_conn = 0;                          /* 0 is assumed to be an invalid address */
 to_net(i, 0, 0, CALL_ACC, data, 0);       /* tell net to accept connection */
 return(i);                                /* return connection identifier */
}

int connect(transport_address l, transport_address r)
{ /* User wants to connect to a remote process;  send CALL_REQ packet. */
 int i;
 struct conn *cptr;

 data[0] = r;  data[1] = l;                /* CALL_REQ packet needs these */
 i = MAX_CONN;                             /* search table backward */
 while (conn[i].state != IDLE && i > 1) i = i - 1;
 if (conn[i].state == IDLE) {
     /* Make a table entry that CALL_REQ has been sent. */
     cptr = &conn[i];
     cptr->local_address = l; cptr->remote_address = r;
     cptr->state = WAITING; cptr->clr_req_received = 0;
     cptr->credits = 0; cptr->timer = 0;
     to_net(i, 0, 0, CALL_REQ, data, 2);
     sleep();                              /* wait for CALL_ACC or CLEAR_REQ */
     if (cptr->state == ESTABLISHED) return(i);
     if (cptr->clr_req_received) {
          /* Other side refused call. */
          cptr->state = IDLE;              /* back to IDLE state */
          to_net(i, 0, 0, CLEAR_CONF, data, 0);
          return(ERR_REJECT);
     }
 } else return(ERR_FULL);                  /* reject CONNECT: no table space */
}

int send(int cid, unsigned char bufptr[], int bytes)
{ /* User wants to send a message. */
 int i, count, m;
 struct conn *cptr = &conn[cid];

 /* Enter SENDING state. */
 cptr->state = SENDING;
 cptr->byte_count = 0;                     /* # bytes sent so far this message */
 if (cptr->clr_req_received == 0 && cptr->credits == 0) sleep();
 if (cptr->clr_req_received == 0) {
     /* Credit available; split message into packets if need be. */
     do {
          if (bytes - cptr->byte_count > MAX_PKT_SIZE) {/* multipacket message */
               count = MAX_PKT_SIZE;  m = 1;/* more packets later */
          } else {                         /* single packet message */
               count = bytes - cptr->byte_count;  m = 0;/* last pkt of this message */
          }
          for (i = 0; i < count; i++) data[i] = bufptr[cptr->byte_count + i];
          to_net(cid, 0, m, DATA_PKT, data, count);/* send 1 packet */
          cptr->byte_count = cptr->byte_count + count;/* increment bytes sent so far */
     } while (cptr->byte_count < bytes);  /* loop until whole message sent */
```

```
 cptr->credits--;                        /* each message uses up one credit */
      cptr->state = ESTABLISHED;
      return(OK);
 } else {
      cptr->state = ESTABLISHED;
      return(ERR_CLOSED);                /* send failed: peer wants to disconnect */
 }
}

int receive(int cid, unsigned char bufptr[], int *bytes)
{ /* User is prepared to receive a message. */
 struct conn *cptr = &conn[cid];

 if (cptr->clr_req_received == 0) {
      /* Connection still established; try to receive. */
      cptr->state = RECEIVING;
      cptr->user_buf_addr = bufptr;
      cptr->byte_count = 0;
      data[0] = CRED;
      data[1] = 1;
      to_net(cid, 1, 0, CREDIT, data, 2); /* send credit */
      sleep();                            /* block awaiting data */
      *bytes = cptr->byte_count;
 }
 cptr->state = ESTABLISHED;
 return(cptr->clr_req_received ? ERR_CLOSED : OK);
}

int disconnect(int cid)
{ /* User wants to release a connection. */
 struct conn *cptr = &conn[cid];

 if (cptr->clr_req_received) {           /* other side initiated termination */
      cptr->state = IDLE;                /* connection is now released */
      to_net(cid, 0, 0, CLEAR_CONF, data, 0);
 } else {                                /* we initiated termination */
      cptr->state = DISCONN;             /* not released until other side agrees */
      to_net(cid, 0, 0, CLEAR_REQ, data, 0);
 }
 return(OK);
}

void packet_arrival(void)
{ /* A packet has arrived, get and process it. */
 int cid;                                /* connection on which packet arrived */
 int count, i, q, m;
 pkt_type ptype;    /* CALL_REQ, CALL_ACC, CLEAR_REQ, CLEAR_CONF, DATA_PKT, CREDIT */
 unsigned char data[MAX_PKT_SIZE]; /* data portion of the incoming packet */
 struct conn *cptr;

 from_net(&cid, &q, &m, &ptype, data, &count); /* go get it */
 cptr = &conn[cid];
```

```
switch (ptype) {
  case CALL_REQ:                    /* remote user wants to establish connection */
    cptr->local_address = data[0];  cptr->remote_address = data[1];
    if (cptr->local_address == listen_address) {
        listen_conn = cid;  cptr->state = ESTABLISHED;  wakeup();
    } else {
        cptr->state = QUEUED;  cptr->timer = TIMEOUT;
    }
    cptr->clr_req_received = 0;   cptr->credits = 0;
    break;
  case CALL_ACC:                    /* remote user has accepted our CALL_REQ */
    cptr->state = ESTABLISHED;
    wakeup();
    break;
  case CLEAR_REQ:                   /* remote user wants to disconnect or reject call */
    cptr->clr_req_received = 1;
    if (cptr->state == DISCONN) cptr->state = IDLE;/* clear collision */
    if (cptr->state ==  WAITING || cptr->state == RECEIVING || cptr->state == SENDING) wakeup();
    break;
  case CLEAR_CONF:                  /* remote user agrees to disconnect */
    cptr->state = IDLE;
    break;
  case CREDIT:                      /* remote user is waiting for data */
    cptr->credits += data[1];
    if (cptr->state == SENDING) wakeup();
    break;
  case DATA_PKT:                    /* remote user has sent data */
    for (i = 0; i < count; i++) cptr->user_buf_addr[cptr->byte_count + i] = data[i];
    cptr->byte_count += count;
    if (m == 0 ) wakeup();
 }
}
void clock(void)
{ /* The clock has ticked, check for timeouts of queued connect requests. */
 int i;
 struct conn *cptr;

 for (i = 1; i <= MAX_CONN; i++) {
     cptr = &conn[i];
     if (cptr->timer > 0) {                 /* timer was running */
         cptr->timer--;
         if (cptr->timer == 0) {            /* timer has now expired */
             cptr->state = IDLE;
             to_net(i, 0, 0, CLEAR_REQ, data, 0);
         }
     }
 }
}
```

شكل ۶-۲۰. مثالي از كُد برنامة واحد انتقال (Transport Entity).

گذار از یک وضعیت به وضعیتی دیگر زمانی اتفاق می‌افتد که یکی از وقایع ذیل اتفاق بیفتد: یک تابع اولیه اجرا شود، بسته‌ای دریافت شود یا مهلت تایمر منقضی شود.

پروسیجرهای نشان داده شده در شکل ۶-۲۰ بر دو نوعند: اغلب آنها مستقیماً قابل فراخوانی در برنامهٔ کاربر هستند در حالیکه ***Packet-arrival*** و ***clock*** متفاوتند. این دو تابع در اثر رخدادهای خارجی به صورت خودکار شروع به کار می‌کنند: اولی در اثر دریافت یک بسته و دومی با هر تیک ساعت به کار می‌افتد. در حقیقت این دو، روتینهای وقفه (Interrupt Routines) هستند. فرض خواهیم کرد که این دو تابع در حین اجرای هیچیک از پروسیجرهای واحد انتقال فراخوانی نمی‌شوند بلکه فقط زمانی که پروسهٔ کاربر در حالت توقّف (sleeping) یا در حال اجرای کُدی غیر از پروسیجرهای لایهٔ انتقال باشد، فراخوانی می‌شوند. این ویژگی برای عملکرد صحیح کُد برنامه، الزامی و حیاتی است.

وجود بیت Q (Qualifer) در سرآیند بسته اجازه می‌دهد که از سربار پروتکل لایهٔ انتقال جلوگیری شود: پیامهای حاوی داده‌های معمولی، به صورت بسته‌هایی با Q=0 ارسال می‌شوند؛ پیامهای کنترلی لایهٔ انتقال (که در مثال ما فقط یک پیام و آن هم CREDIT است) در قالب بسته‌ای ارسال می‌شوند که در آنها Q=1 است. این پیامهای کنترلی در سمت گیرنده توسط «واحد انتقال» کشف و پردازش می‌شوند.

ساختمان دادهٔ اصلی که توسط واحد انتقال مورد استفاده قرار می‌گیرد، آرایهٔ conn است که به ازای هر اتصال یک رکورد در آن ذخیره می‌شود. این رکورد وضعیت فعلی یک اتصال را نگه می‌دارد و شامل اطلاعات ذیل است: (۱) آدرس انتقال طرف مقابل (یعنی آدرس TSAP طرف مقابل) (۲) تعداد پیامهای ارسالی یا دریافتی از آن اتصال (۳) وضعیت جاری [یکی از هفت وضعیت] (۴) اشاره‌گر آدرس بافر کاربر (۵) تعداد بایتهایی که تاکنون از پیام فعلی ارسال یا دریافت شده‌اند. (۶) یک بیت که مشخص می‌کند کاربر راه دور، دستور DISCONNECT را صادر کرده است. (۷) یک «شمارندهٔ مجوز» (Permission Counter) که ارسال پیامها را فعال و ممکن می‌سازد. البته در مثال سادهٔ ما از تمام این فیلدها استفاده نمی‌شود بلکه یک «واحد انتقال» کامل و دقیق به این فیلدها و حتی فیلدهای بیشتر، نیاز خواهد داشت. فرض بر آنست که هر درایه از آرایهٔ conn در وضعیت اولیهٔ IDLE قرار می‌گیرد. [یعنی با IDLE مقداردهی اولیه می‌شود.]

وقتی کاربری تابع CONNECT را فراخوانی می‌کند، به لایهٔ شبکه دستور داده می‌شود که یک بسته از نوع CALL REQUEST برای ماشین راه دور بفرستد؛ سپس کاربر به حالت توقف (استراحت) می‌رود. وقتی بستهٔ CALL REQUEST در طرف مقابل دریافت شود، به «واحد انتقال» وقفه (اینتراپت) داده می‌شود تا با اجرای روتین packet_arrival بررسی کند که آیا یک کاربر محلی به آدرس مشخص شده گوش می‌دهد یا خیر؛ اگر اینگونه باشد، بستهٔ CALL ACCEPTED بازگردانده شده و کاربر راه دور مجدداً فعال می‌شود. اگر اینگونه نباشد، CALL REQUEST در صف انتظار قرار می‌گیرد تا ساعت، به تعداد TIMEOUT تیک بزند. اگر در خلال این مدت عمل LISTEN انجام شود، اتصال برقرار خواهد شد؛ در غیر این صورت مهلت زمان، منقضی و با ارسال بستهٔ CLEAR REQUEST این اتصال ردّ خواهد شد تا طرف مقابل تا ابد در حالت توقف نماند.

اگرچه در مثالمان سرآیند پروتکل انتقال را حذف کرده‌ایم ولی بهرحال به راهی نیاز داریم تا بتوان متوجه شد که هر بسته متعلق به کدام اتصال است، چراکه ممکن است بطور همزمان چندین اتصال فعّال وجود داشته باشد. ساده‌ترین راه حل آن است که از شمارهٔ مدار مجازی در لایهٔ شبکه به عنوان شمارهٔ اتصال در لایه انتقال استفاده شود. مضاف بر این، از شمارهٔ مدار مجازی می‌توان به عنوان اندیس آرایهٔ conn بهره گرفت. یعنی وقتی بسته‌ای به مدار مجازی شمارهٔ k در لایهٔ شبکه وارد شود متعلق به اتصال شمارهٔ k در لایهٔ انتقال می‌باشد، و وضعیت این اتصال در رکورد conn[k] قرار گرفته است. برای اتصالاتی که از یک ماشین میزبان شروع می‌شود، شمارهٔ اتصال

توسط «واحد انتقال مبداء»[1] انتخاب می‌شود. برای تقاضاهای اتصال، لایهٔ شبکه این انتخاب را انجام می‌دهد و طبعاً یک شمارهٔ بلااستفاده از بین شماره‌های مدار مجازی انتخاب می‌کند.

برای آن که مجبور به مدیریت بافرها در درون واحد انتقال نباشیم از یک مکانیزم کنترل جریان خاصّ و متفاوت از روش معمولی پنجرهٔ لغزان استفاده کرده‌ایم. وقتی یک کاربر، تابع RECEIVE را فراخوانی می‌کند یک پیام خاص به نام CREDIT (پیام اعتبار) برای واحد انتقال در ماشین فرستنده ارسال می‌شود و در آرایهٔ conn ثبت می‌شود. وقتی تابع SEND فراخوانی شود، واحد انتقال ابتدا بررسی می‌کند که آیا برای اتصال مربوطه «اعتبار» لازم وجود دارد یا خیر: اگر اعتبار لازم وجود داشته باشد، پیام ارسال می‌شود (در صورت لزوم در چند بسته)؛ سپس به اندازهٔ طول پیام از اعتبار موجود کاسته می‌شود؛ در غیر این صورت، واحد انتقال خودش را به حالت توقف (Sleep) می‌برد تا اعتبار لازم برسد. این مکانیزم تضمین می‌کند که هیچ پیامی ارسال نشود مگر آن که طرف مقابل قبلاً فرمان RECEIVE را اجرا کرده باشد. لذا وقتی پیامی از راه برسد مطمئناً بافر لازم برای آن موجود و آماده خواهد بود. این روش را می‌توان براحتی تعمیم داد تا گیرندگان بتوانند چندین بافر تهیه کرده و تقاضای چند پیام بدهند.

سادگی کُد برنامهٔ نشان داده شده در شکل ۶-۲۰ را به خاطر بسپارید. یک واحد انتقال واقعی باید اعتبار تمام پارامترهای عرضه شده توسط کاربر را بررسی نماید، جبران از کار افتادگی در لایهٔ شبکه را بر عهده بگیرد، برخورد دو تماس (Call Collisions) را مدیریت کند و از سرویسهای انتقال عمومی‌تری شامل تسهیلات وقفه، سرویس دیتاگرام و «نسخه‌های غیرقابل تعلیق SEND و RECEIVE»[2] پشتیبانی نماید.

۶-۳-۳ بررسی مثال فوق از دید «ماشین حالت محدود»

نوشتن یک برنامه برای «واحد انتقال» خصوصاً برای پروتکلهای واقعی و عملی‌تر، بسیار دشوار است. برای آن که احتمال خطا کاهش یابد، توصیف وضعیت پروتکل در قالب یک «**ماشین حالت محدود**» (Finite State Machine) اغلب سودمند است. قبلاً دیدیم که در پروتکل مثال ما، برای هر اتصال هفت وضعیت وجود دارد. در مجموع می‌توان ۱۲ رخداد مختلف تعریف کرد که می‌توانند وضعیت یک اتصال را تغییر بدهند. پنج تا از این رخدادها، در اثر فراخوانی توابع اولیه حادث می‌شوند. شش‌تای بعدی در اثر ورود ۶ بستهٔ معتبر رخ می‌دهند. آخرین رخداد ناشی از انقضای مهلت تایمر است. در شکل ۶-۲۱ فعالیتهای اصلی پروتکل به شکل ماتریسی نشان داده شده است. ستونها بیانگر «وضعیت» و سطرها بیانگر ۱۲ «رخداد» مختلف است.

هر درایه (Entry) در ماتریس شکل ۶-۲۱ (یا به عبارتی در ماشین حالت محدود) حداکثر سه فیلد دارد: (۱) **گزاره (Predicate) (۲) کُنش (Action) (۳) وضعیت جدید (new state)**

«گزاره» مشخص می‌کند که تحت چه شرایطی «کُنش» تعیین شده انجام خواهد گرفت. به عنوان نمونه، درایهٔ گوشهٔ بالا و سمت چپ ماتریس نشان می‌دهد که اگر LISTEN اجرا شود ولی در جدول، فضای حافظه باقی نمانده باشد (گزارهٔ P1)، اجرای LISTEN با شکست مواجه شده و وضعیت پروتکل تغییر نخواهد کرد. از طرف دیگر، اگر یک بستهٔ درخواست CALL REQUEST برای آدرسی که یک پروسه در حال گوش دادن به آن است، بیاید (گزارهٔ P2)، اتصال مربوطه بلافاصله برقرار می‌شود. حالت ممکن بعدی آن است که گزارهٔ P2 نادرست باشد یعنی هیچ بستهٔ CALL REQUEST دریافت نشده باشد که در این حالت، اتصال در وضعیت بیکار (IDLE) و در انتظار بستهٔ CALL REQUEST باقی خواهد ماند.

اشاره به این نکته مهم است که انتخاب وضعیتهایی که در ماتریس فوق بکار رفته است، در برگیرندهٔ تمام

---

۱. Originating Transport Entity ۲. nonblocking SEND/RECEIVE

		State						
		Idle	Waiting	Queued	Established	Sending	Receiving	Dis-connecting
Primitives (فراخوانی عملکردهای اولیه)	LISTEN	P1: ~/Idle P2: A1/Estab $\overline{P2}$: A2/Idle		~/Estab				
	CONNECT	P1: ~/Idle $\overline{P1}$: A3/Wait						
	DISCONNECT				P4: A5/Idle $\overline{P4}$: A6/Disc			
	SEND				P5: A7/Estab $\overline{P5}$: A8/Send			
	RECEIVE				A9/Receiving			
Incoming packets (بسته‌های ورودی)	Call_req	P3: A1/Estab $\overline{P3}$: A4/Queu'd						
	Call_acc		~/Estab					
	Clear_req		~/Idle		A10/Estab	A10/Estab	A10/Estab	~/Idle
	Clear_conf							~/Idle
	DataPkt						A12/Estab	
	Credit				A11/Estab	A7/Estab		
Clock	Timeout			~/Idle				

Predicates

P1: Connection table full
P2: Call_req pending
P3: LISTEN pending
P4: Clear_req pending
P5: Credit available

Actions

A1: Send Call_acc
A2: Wait for Call_req
A3: Send Call_req
A4: Start timer
A5: Send Clear_conf
A6: Send Clear_req
A7: Send message
A8: Wait for credit
A9: Send credit
A10: Set Clr_req_received flag
A11: Record credit
A12: Accept message

شکل ۶-۲۱. مدل پروتکل مثال قبل در قالب «ماشین حالت محدود». هر درایه (Entry) یک «گزاره» اختیاری ، یک «کُنش» اختیاری و یک «وضعیت» جدید دارد. علامت ~ بیانگر آنست که هیچ «کنشی» صورت نمی‌گیرد. علامت - بر روی یک گزاره، منفی آنرا نشان می‌دهد.درایه‌های خالی مربوط به رخدادهای ناممکن یا نامعتبر هستند.

وضعیتهای ممکن نیست: در این مثال، وضعیت LISTENING (یعنی وضعیت انتظار برای برقراری اتصال) وجود ندارد در حالی که این وضعیت، پس از صدور فرمان LISTEN اتفاق می‌افتد و وضعیتی معقول به شمار می‌رود. چرا این وضعیت لحاظ نشده است؟ زیرا تصمیم گرفته‌ایم که از شمارهٔ مدار مجازی بکار رفته در لایهٔ شبکه به عنوان شناسهٔ اتصال (Connection Identifier) استفاده کنیم و به واسطهٔ صدور فرمان LISTEN هیچ رکورد اتصال خاصّی که مبیّن وضعیت LISTENING باشد در جدول اتصالات ایجاد نخواهد شد. یعنی برای LISTEN شمارهٔ مدار مجازی، نهایتاً توسط لایهٔ شبکه و در هنگام ورود بستهٔ CALL REQUEST انتخاب می‌شود.

کُنشهای A1 تا A12 ، عملیات اصلی تلقی می‌شوند؛ عملیاتی نظیر ارسال بسته و راه‌اندازی تایمرها. عملیات

کوچک و جانبی مثل مقداردهی اولیه به فیلدهای یک رکورد اتصال، فهرست نشده‌اند. اگر یک «کنش» منوط به فعال کردن یک پروسهٔ معلق باشد، عملیات لازم پس از فعال‌سازی پروسه نیز جزو آن کنش محسوب می‌شود: به عنوان مثال هر گاه بستهٔ CALL REQUEST وارد شود و پروسه‌ای در انتظار آن معلق و منتظر مانده باشد، ارسال بستهٔ CALL ACCEPT که بلافاصله پس از فعال شدن پروسه انجام می‌گیرد به عنوان بخشی از عمل CALL REQUEST محسوب می‌شود. پس از انجام هر عمل، اتصال در وضعیت جدیدی قرار می‌گیرد. (به شکل ۶-۲۱ دقت کنید.)

توصیف عملکرد پروتکل در قالب ماتریس سه مزیت دارد: اول آن که بدین شکل برنامه‌نویس ساده‌تر می‌تواند ترکیبات مختلف «وضعیت» و «رخدادها» را بررسی کرده و نیاز به یک «کنش» را تشخیص بدهد. در پیاده‌سازی عملی و واقعی پروتکل، برخی از این ترکیبات (که اکنون به حال خود رها شده‌اند) برای مدیریت خطا کاربرد دارند. در شکل ۶-۲۱ هیچ تمایزی بین «وضعیتهای غیرممکن» و «وضعیتهای نامعتبر» قائل نشده‌ایم.[1] به عنوان مثال هرگاه یک اتصال در وضعیت «انتظار» (waiting) قرار داشته باشد، رخداد DISCONNECT غیرممکن است چراکه در وضعیت انتظار، پروسهٔ کاربر متوقف شده و هرگز نمی‌تواند هیچ تابعی را اجرا نماید. برعکس، در وضعیت sending (ارسال)، انتظار دریافت بسته نمی‌رود زیرا هیچ اعتباری جهت ارسال برای طرف مقابل صادر نشده است و دریافت یک بستهٔ داده، خطای پروتکل محسوب می‌شود.

دومین مزیت نمایش ماتریسی پروتکل، پیاده‌سازی آنست. برنامه‌نویس می‌تواند آرایه‌ای دو بعدی در نظر بگیرد که در هر عنصر آن یعنی $a[i][j]$، اشاره‌گر شروع یا اندیس پروسیجری قرار گرفته که هنگام بروز رخداد i در وضعیت j، باید اجرا شود و ادارهٔ امور را بر عهده بگیرد. یکی از راهکارهای پیاده‌سازی «واحد انتقال» آنست که برنامه را در قالب یک حلقهٔ کوتاه (حلقهٔ تکرار در برنامه‌نویسی) بنویسیم که در ابتدای حلقه منتظر وقوع یک رخداد باقی می‌ماند. وقتی یک رخداد اتفاق می‌افتد اتصال مربوطه شناسایی و تعیین شده و وضعیت آن استخراج می‌گردد. براساس وضعیت و رخدادهایی که در این ماتریس تبیین شده، «واحد انتقال» اندیس مناسب را استخراج و با رجوع به آرایهٔ a، پروسیجر مناسب را فراخوانی و اجرا می‌نماید.

حُسن سوم در روش «ماشین حالت محدود»، سادگی توصیف پروتکل است. در مستندات برخی از استانداردها، هر پروتکل در قالب یک «ماشین حالت محدود» (همانند آنچه که در شکل ۶-۲۲ نشان داده شده) ، توصیف و تشریح شده است. هر گاه واحد انتقال در قالب یک ماشین حالت محدود توصیف شده باشد راحتتر می‌توان از طریق آن به سوی پیاده‌سازی عملی آن حرکت کرد.

اشکال روش «ماشین حالت محدود» آن است که فهم و تحلیل آن دشوارتر از برنامه‌نویسی معمولی است. البته این اشکال را می‌توان با ترسیم ماشین حالت محدود به صورت گراف، تا حدودی حل کرد. این نوع نمایش در شکل ۶-۲۲ نشان داده شده است.

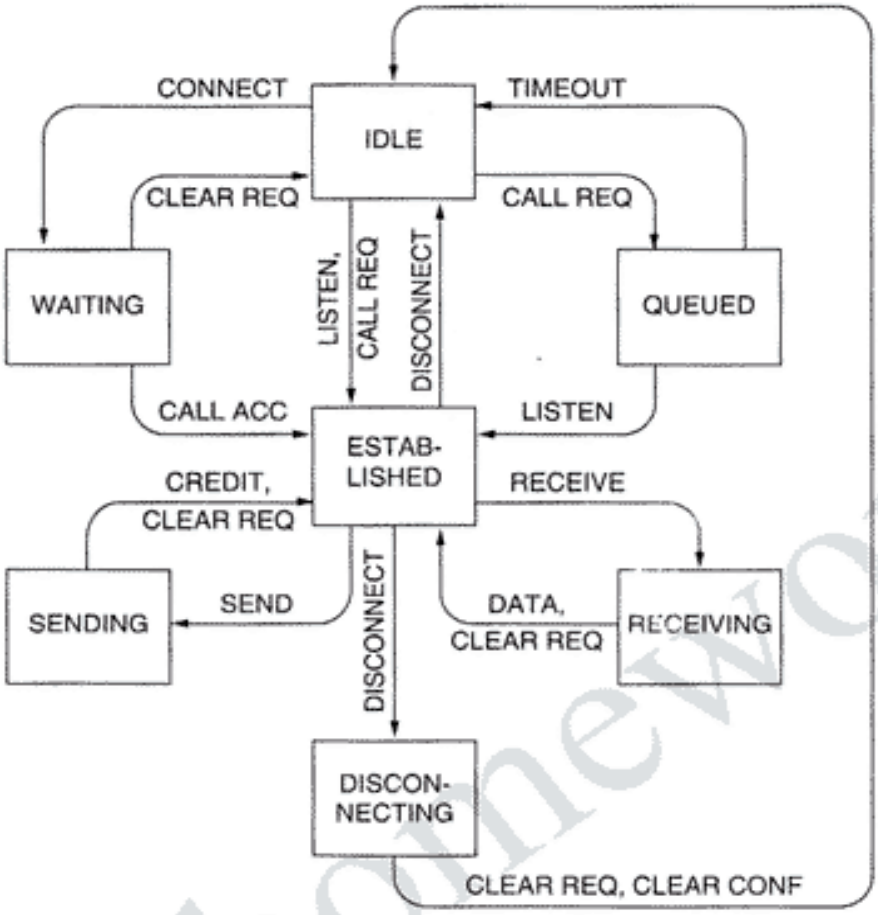
۶-۴ پروتکلهای لایهٔ انتقال در اینترنت: UDP[2]

اینترنت در لایهٔ انتقال دو پروتکل عمده دارد: یکی بدون‌اتصال و دیگری اتصال‌گرا. در بخشهای آتی به مطالعهٔ این دو پروتکل خواهیم پرداخت. پروتکل بدون‌اتصال در اینترنت، UDP و پروتکل اتصال‌گرای آن، TCP نام دارد. از آنجایی که UDP اساساً همان IP به همراه یک سرآیند کوتاه است لذا با آن شروع خواهیم کرد. در ضمن به دو کاربرد مختلف UDP خواهیم پرداخت.

---

۱. «وضعیتهای غیرممکن» یعنی وضعیتهایی که هرگز اتفاق نخواهد افتاد و «وضعیتهای نامعتبر» وضعیتهایی است که نباید اتفاق بیفتد! -م

۲. User Datagram Protocol



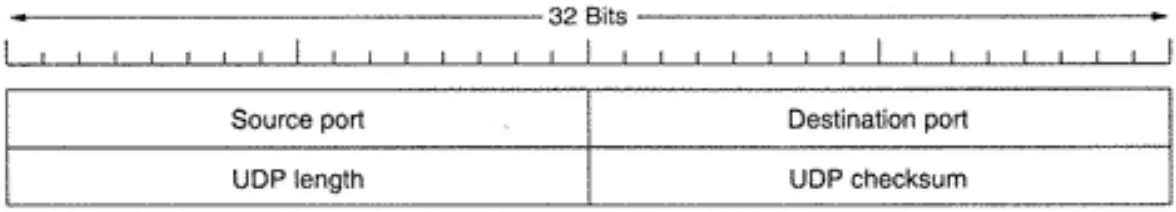
شکل ۶-۲۲. مدل پروتکل مثال قبلی در قالب گراف. برای سادگی «گذارهایی» (Transitions) که وضعیت اتصال را تغییر نمی‌دهند از گراف حذف شده‌اند.

۶-۴-۱ مقدمه‌ای بر UDP

پشتهٔ پروتکلی اینترنت در لایهٔ انتقال از یک پروتکل بدون‌اتصال به نام UDP پشتیبانی می‌کند. UDP امکان آن را فراهم آورده که برنامه‌های کاربردی بتوانند بدون ایجاد هر گونه اتصال و هماهنگی قبلی، داده‌ای را درون یک دیتاگرام IP جاسازی کرده و آن را بفرستند. پروتکل UDP در RFC 768 تشریح شده است.

UDP داده‌ها را در قالب قطعاتی[1] ارسال می‌کند که در ابتدای آنها ۸ بایت سرآیند و سپس داده‌های لایهٔ کاربرد قرار می‌گیرد. این سرآیند در شکل ۶-۲۳ نشان داده شده است. دو فیلد شمارهٔ پورت[2] به منظور شناسائی نقاط پایانی (پروسه‌های نهایی) در ماشینهای مبداء و مقصد به کار می‌آیند. وقتی یک بستهٔ UDP از راه می‌رسد، محتوای آن به پروسهٔ متصل[3] به شمارهٔ پورت مقصد، تحویل داده می‌شود. عمل اتصال پروسه به یک پورت از طریق تابع اولیهٔ BIND (که تعریف آنرا برای TCP در شکل ۶-۶ دیدیم) انجام می‌شود. (فرآیند مقیدسازی پروسه به یک پورت در TCP یا UDP تفاوتی ندارد.) در حقیقت، آنچه که UDP در مقایسه با IP معمولی اضافه‌تر دارد پورتهای مبداء و مقصد هستند. بدون فیلدهای مربوط به پورت، لایهٔ انتقال نمی‌داند که با یک بسته چه کار کند. با این فیلدها، قطعه داده به درستی تحویل پروسهٔ مربوطه خواهد شد.

۱. Segments ۲. Source & Destination Port ۳. Attach



شکل ۶-۲۳. سرآیند UDP (UDP Header).

برای آن که بتوان برای پروسهٔ مبداء پاسخی برگرداند، به شمارهٔ پورت مبداء نیاز است. بدین منظور محتوای فیلد پورت مبداء (Source Port) از بستهٔ ورودی، در فیلد پورت مقصد از بستهٔ خروجی، کپی و ارسال می‌شود. بدین ترتیب فرستندهٔ پاسخ ، پروسهٔ تحویل گیرندهٔ بسته را مشخص می‌نماید.

فیلد UDP Length اندازهٔ کل قطعهٔ داده (شامل ۸ بایت سرآیند) را مشخص می‌نماید. فیلد UDP checksum به منظور کشف خطاهای احتمالی کاربرد دارد و اختیاری است؛ در صورت عدم محاسبه، در این فیلد عدد صفر درج می‌شود. (البته اگر مقدار واقعی این کُد صفر باشد به جای آن تمام فیلد با بیتهای ۱ پر می‌شوند.) عدم استفاده از این فیلد نوعی سهل‌انگاری است مگر آن که کیفیت داده‌ها چندان مهم نباشد. (مثلاً در مورد صدای دیجیتال)

شاید اشارهٔ صریح به کارهایی که UDP انجام نمی‌دهد، ارزشمند باشد. این پروتکل کنترل جریان (Flow Control) و کنترل خطا (Error Control) ندارد و در صورت دریافت یک قطعه دادهٔ خراب، آن را «ارسال مجدّد» نخواهد کرد. تمام این عملیات بر عهدهٔ پروسهٔ کاربر گذاشته شده است. آنچه که این پروتکل انجام می‌دهد عبارت است از ایجاد یک واسط (Interface) بین پروسه‌های کاربردی و پروتکل IP و انجام عمل دی‌مالتی‌پلکس قطعات داده بین پروسه‌های کاربردی.[1] این تمام کاری است که UDP انجام می‌دهد. برای برنامه‌های کاربردی که نیاز به کنترل دقیق و جدی جریان، کنترل خطا و زمان‌بندی دارند، UDP امکانات چندانی را در اختیار نمی‌گذارد و آن را به خود برنامه محول کرده است.

یکی از زمینه‌هایی که استفاده از UDP مفید است، برنامه‌های خاص سرویس‌د، نده/مشتری هستند که در آنها مشتری تقاضای کوتاهی را برای سرویس‌دهنده می‌فرستد و انتظار پاسخی کوتاه دارد. اگر پیام تقاضا یا پاسخ از بین برود، مهلت مشتری به سر می‌آید و از نو شروع می‌کند. در این حالت، نه تنها کد برنامه ساده‌تر می‌شود بلکه به مبادله پیامهای کمتری در دو جهت نیاز است.

یکی از برنامه‌های کاربردی که از UDP بهره گرفته، DNS[2] است که آن را در فصل هفتم مطالعه خواهیم کرد. خلاصهٔ عملکرد DNS بدین نحو است: یک برنامه که در جستجوی آدرس IP یک ماشین میزبان با نامی مثل www.cs.berkeley.edu است یک بستهٔ UDP حاوی نام ماشین میزبان به سوی DNS می‌فرستد. سرویس‌دهندهٔ DNS، با ارسال یک بستهٔ UDP، آدرس IP ماشین میزبان مورد نظر را به اطلاع ماشین سوال کننده می‌رساند. در این حالت نیازی به هماهنگی قبلی و ایجاد یا ختم اتصال نیست و در مجموع فقط دو پیام ردّ و بدل می‌شود.

۶-۴-۲ فراخوانی پروسیجرهای راه دور (RPC : Remote Procedure Call)

از برخی جهات، ارسال یک پیام برای یک ماشین راه دور و دریافت پاسخ، با فراخوانی یک تابع در زبانهای

---

۱. یعنی با بررسی شمارهٔ پورت مقصد از هر بستهٔ UDP آنرا بین پروسه‌های مربوطه توزیع می‌نماید. م-

۲. Domain Name System

برنامه‌نویسی مشابهت دارد. در هر دو مورد یک یا چند پارامتر را تحویل می‌دهید و نتیجه‌ای به شما بازگردانده می‌شود. این شباهت، افراد را بدین نکته رهنمون ساخت که عملیات مبتنی بر «پرسش و پاسخ[1] در شبکه» را می‌توان در قالب فراخوانی پروسیجر سازماندهی و تنظیم کرد. چنین ساختاری، برنامه‌نویسی برنامه‌های کاربردی شبکه را ساده‌تر و از لحاظ مفهومی آشناتر و ملموس‌تر می‌کند. به عنوان مثال، پروسیجری به نام get_IP_address(host_name) را مدّ نظر قرار بدهید که با ارسال یک بستهٔ UDP برای سرویس‌دهندهٔ DNS کار خود را آغاز کرده و منتظر پاسخ آن باقی می‌ماند و اگر پاسخ آن در مهلت مقرر دریافت نشود، تلاش خود را تکرار می‌کند. بدین ترتیب تمام جزئیات شبکه از دید برنامه‌نویس مخفی می‌ماند.

در این زمینه کارهایی اساسی توسط دو نفر به نامهای Birrell و Nelson (۱۹۸۴) انجام شده است. چکیدهٔ آنچه که Birrell و Nelson پیشنهاد کردند آنست که به برنامه‌ها اجازه داده شود پروسیجرهایی را بر روی ماشینهای راه دور فراخوانی نمایند. وقتی پروسه‌ای بر روی ماشین ۱، پروسیجری را بر روی ماشین ۲ فراخوانی می‌کند، پروسهٔ صدازننده بر روی ماشین ۱ متوقف و معلق شده و اجرای پروسیجر بر روی ماشین ۲ آغاز می‌شود. اطلاعات لازم از پروسهٔ صدازننده، در قالب چند پارامتر تحویل پروسیجر شده و نتیجهٔ اجرای پروسیجر در پارامترهایی بازگردانده می‌شود. بدین نحو ردّ و بدل هر گونه پیام، از دید برنامه‌نویس مخفی می‌ماند. این تکنیک اصطلاحاً RPC نامیده می‌شود (Remote Procedure Call) و به زیربنای بسیاری از برنامه‌های کاربردی شبکه تبدیل شده است. بطور مرسوم پروسیجر صدازننده با عنوان «مشتری» و پروسیجر فراخوانی شده، با عنوان «سرویس‌دهنده» معرفی می‌شوند و ما نیز از همین اسامی بهره می‌گیریم.

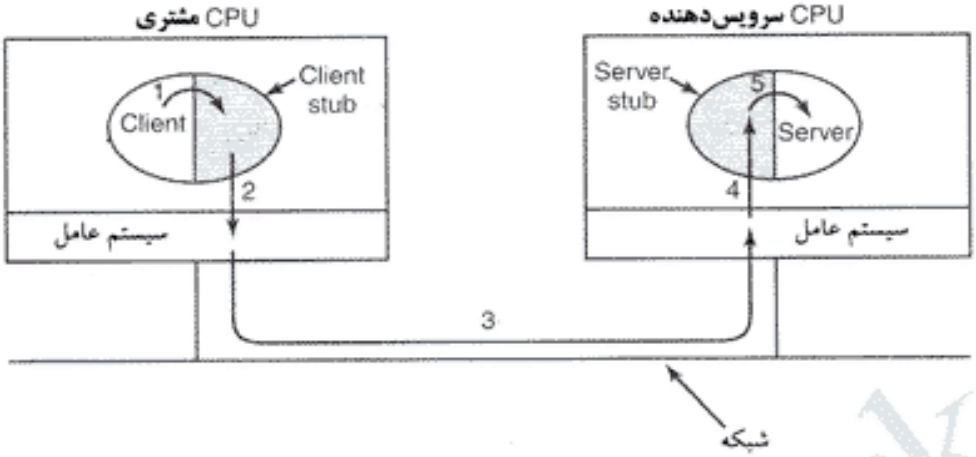
ایده‌ای که در ورای RPC قرار دارد آن است که فراخوانی پروسیجرهای راه دور حتی‌الامکان شبیه به فراخوانی پروسیجرهای محلی باشد. در ساده‌ترین شکل ممکن، برای فراخوانی پروسیجرهای راه دور، باید به برنامهٔ مشتری یک پروسیجر کتابخانه‌ای کوچک به نام client stub مقیّد (bind) شود که پروسیجر سرویس‌دهنده را در فضای آدرس برنامهٔ مشتری تصویر می‌کند. به روش مشابه، سرویس‌دهنده نیز به پروسیجر کوچکی به نام server stub مقید می‌شود که وظیفهٔ آن، پنهان‌سازی این واقعیت است که فراخوانی پروسیجر توسط یک برنامهٔ مشتری بر روی سرویس‌دهندهٔ محلی انجام نشده است.

در شکل ۶-۲۴، مراحل واقعی یک فراخوانی RPC نشان داده شده است. در مرحلهٔ ۱، برنامهٔ مشتری پروسیجر client stub را فراخوانی می‌کند. این مرحله، یک فراخوانی پروسیجر محلی است و طبق معمول پارامترهای لازم به درون پشته هدایت می‌شوند. در مرحلهٔ ۲، پروسیجر client stub این پارامترها را در یک پیام جاسازی کرده و برای ارسال آن یک فراخوانی سیستمی انجام می‌دهد. جاسازی پارامترها در یک پیام اصطلاحاً عمل «**مارشالینگ**» (تشریفات) نام دارد. در مرحلهٔ سوم هستهٔ سیستم عامل، این پیام را از ماشین مشتری به سوی ماشین سرویس‌دهنده ارسال می‌کند. در مرحلهٔ چهارم هستهٔ سیستم عامل سرویس‌دهنده، بستهٔ ورودی را به server stub تحویل می‌دهد. نهایتاً در مرحلهٔ پنجم، server stub پروسیجر سرویس‌دهنده را با پارامترهای دریافتی اجرا می‌نماید. فرآیند ارسال پاسخ در جهت مقابل نیز طبق همین روال انجام خواهد شد.

اشاره به این نکتهٔ کلیدی مهم است که در برنامهٔ مشتری که کاربر آن را می‌نویسد، فراخوانی پروسیجر client stub به روش معمولی انجام می‌شود و نام آن با نام پروسیجر سرویس‌دهنده یکی است. از آنجایی که برنامهٔ مشتری و پروسیجر client stub در فضای آدرس مشابهی قرار دارند لذا تحویل پارامترهای لازم برای فراخوانی پروسیجر به روش معمول (و از طریق پشته) انجام می‌گیرد. به روش مشابه، پروسیجر سرویس‌دهنده

---

۱. request & reply



شکل ۶-۲۴. مراحل لازم برای عمل فراخوانی پروسیجر راه دور (RPC). Client Stub و Server Stub نشان داده نشده است.

توسط پروسیجر دیگر یعنی server stub (در فضای آدرس خودش و با پارامترهای لازم) به روش معمولی فراخوانی می‌شود. برای پروسیجرهای سرویس‌دهنده هیچ چیزی غیرمعمول و نامتعارف نیست. بدین ترتیب به جای آن که عملیات I/O از طریق سوکت انجام شود، مبادلهٔ اطلاعات در شبکه با شبیه‌سازی و تقلید فراخوانی پروسیجر انجام می‌گیرد.

فارغ از زیبایی مفهوم RPC، مار در آستین آن نهان است!! مهمترین مشکل آن پارامترهای نوع اشاره‌گر هستند. بطور طبیعی، ارسال یک اشاره‌گر به پروسیجرهای محلی، اشکالی ندارد. پروسیجر محلی می‌تواند از اشاره‌گرها به همان نحوی که پروسهٔ صدازننده از آنها بهره می‌گیرد، استفاده کند چراکه هر دوی آنها در فضای آدرس مجازی یکسانی به سر می‌برند. در RPC ، ارسال پارامترهایی که از نوع اشاره‌گر هستند غیرممکن است چراکه مشتری و سرویس‌دهنده در فضای آدرس متفاوتی هستند.

البته در برخی موارد می‌توان برای ارسال اشاره‌گر به پروسیجرها از روشهای زیرکانه بهره گرفت. فرض کنید که اولین پارامتر پروسیجر، یک اشاره‌گر به عدد صحیح k باشد. پروسیجر client stub می‌تواند با سازماندهی و تنظیم «مقدار k» در درون یک بسته، آن را برای سرویس‌دهنده بفرستد. پروسیجر server stub پس از ذخیرهٔ مقدار k در حافظه، یک اشاره‌گر برای k ساخته و طبق معمول آن را به پروسیجر سرویس‌دهنده تسلیم می‌کند. وقتی پروسیجر سرویس‌دهنده، کنترل اجرا را به پروسیجر server stub برمی‌گرداند، این پروسیجر مقدار جدید k را برای مشتری باز می‌گرداند تا مقدار قدیم k با مقدار جدیدی که سرویس‌دهنده محاسبه کرده، جایگزین شود. در حقیقت مکانیزم «**فراخوانی به روش ارجاع**»[1] با روش «**فراخوانی از طریق تحویل مقدار و بازیابی نتیجه**» جایگزین شده است. متأسفانه این راهکار همیشه کار نمی‌کند چراکه اگر اشاره‌گر مثلاً به یک گراف یا ساختمان دادهٔ پیچیده اشاره کرده باشد نمی‌توان از این روش بهره گرفت. به همین دلیل باید بر روی پارامترهایی که برای فراخوانی پروسیجرهای راه دور مورد نیاز هستند، محدودیتهایی اعمال کرد.

مسئله دوم آن است که در زبانهایی مثل C که در آن محدودیتهای «نوع داده» (Data Type) چندان قوی نیست نوشتن پروسیجری که مثلاً ضرب داخلی دو بردار (دو آرایه) را حساب می‌کند، بدون مشخص کردن بزرگی اندازهٔ آرایه‌ها، کاملاً صحیح و معتبر است. حال مثلاً قرارداد شده که انتهای این آرایه‌ها با یک مقدار ویژه مشخص گردد و

۱. Call by Reference

پروسیجر صدازننده و صداشونده هر دو آنرا به رسمیت می‌شناسند. در چنین شرایطی client stub اساساً نمی‌تواند چنین پارامترهایی را استخراج، سازماندهی (مارشالینگ) و ارسال نماید چراکه اندازهٔ پارامترها را نمی‌داند.

مسئله سوم آن است که همیشه این امکان وجود ندارد که بتوان نوع داده‌ای پارامترها را از روی توصیف رسمی (Formal Specification) یا کد برنامه استنتاج کرد. مثلاً printf می‌تواند به هر تعداد پارامتر داشته باشد (حداقل یکی) و پارامترها نیز می‌توانند مخلوطی از متغیرهای صحیح (از نوع int، short، long)، کاراکتر، رشته‌ای (string)، اعداد اعشاری با طول دلخواه و انواع دیگر باشند. تلاش در فراخوانی پروسیجر printf از راه دور، عملاً غیرممکن است چراکه زبان C در خصوص انواع داده آسان می‌گیرد. از طرفی اگر قانونی وضع شود که استفاده از RPC را منوط به عدم استفاده از C (یا C++) کند، آنرا از عمومیت و رواج می‌اندازد.

مسئله چهارم در ارتباط با متغیرهای سراسری برنامه است. بطور طبیعی پروسیجرهای صدازننده و صداشونده می‌توانند به غیر از پارامترها از طریق متغیرهای سراسری نیز با یکدیگر تبادل داده داشته باشند. اگر پروسیجر فراخوانی شده به یک ماشین راه دور منتقل شود، این کد با شکست مواجه خواهد شد چراکه متغیرهای سراسری در آنجا معتبر و قابل استفاده نیستند.

این مسائل بدین معنا نیست که باید از RPC سلب امید کرد. در واقع بطور گسترده‌ای از آن استفاده می‌شود ولیکن باید محدودیتهایی را اعمال کرد تا در عمل بدرستی کار کند.

البته RPC لزوماً از UDP استفاده نمی‌کند ولیکن RPC و UDP زوج متناسبی هستند و عموماً برای RPC از UDP بهره گرفته می‌شود. علیرغم این، وقتی پارامترها یا نتیجهٔ اجرای پروسیجر از اندازهٔ حداکثر یک بستهٔ UDP بیشتر باشد یا وقتی که عملیات مورد تقاضا قابل تکرار نباشند یا تکرار آن، نتایج نامشخصی به همراه داشته باشد ممکن است به جای استفاده از UDP مجبور به تنظیم یک اتصال TCP و ارسال تقاضا از طریق آن باشیم.

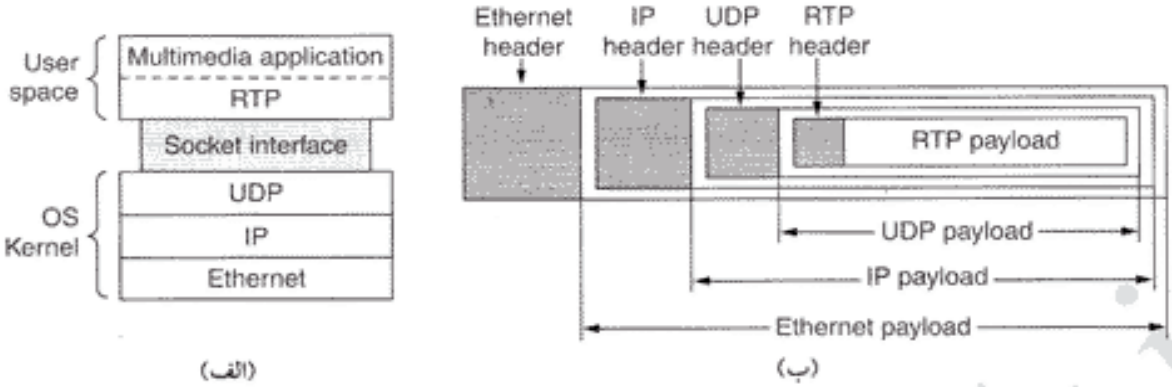
۶-۴-۳ پروتکل انتقال بی‌درنگ

مدل سرویس‌دهنده/مشتری مبتنی RPC، زمینه‌ای است که در آن از UDP استفادهٔ گسترده‌ای می‌شود. زمینهٔ دیگر استفاده از UDP، برنامه‌های کاربردی چند رسانه‌ای بی‌درنگ است. خصوصاً با رواج روزافزون رادیوی اینترنتی، تلفن اینترنتی، موزیک درخواستی از شبکه، ویدیوکنفرانس و دیگر کاربردهای چندرسانه‌ای، عموم افراد دریافتند که همه در حال حرکت به سوی یک پروتکل انتقال بی‌درنگِ کم و بیش مشابه هستند. به تدریج روشن شد که داشتن یک پروتکل عمومی و استاندارد برای انتقال بی‌درنگ داده‌ها ایدهٔ خوبی است. بدین ترتیب پروتکل RTP[1] متولد شد. این پروتکل در سند RFC 1889 تشریح شده و امروزه رواج گسترده‌ای دارد.

جایگاه RTP در پشتهٔ پروتکلی اندکی عجیب به نظر می‌رسد. تصمیم گرفته شد که RTP در فضای پروسهٔ کاربر قرار گرفته و از طریق UDP اجرا شود. عملکرد RTP به نحو زیر است: برنامهٔ چندرسانه‌ای حاوی چندین قطعهٔ صدا، ویدیو، متن و احتمالاً استریمهای دیگر می‌باشد؛ این داده‌ها به «کتابخانهٔ RTP[2]» که در فضای برنامهٔ کاربر قرار دارد و به همراه آن اجرا می‌گردد، خورانده می‌شود. این کتابخانه، استریم داده‌ها را مالتی‌پلکس کرده و آنها را در بسته‌های RPT جاسازی می‌کند. سپس آنها را بر روی یک سوکت قرار می‌دهد. در آنسوی سوکت، (یعنی در هستهٔ سیستم عامل)، بسته‌های UDP تولید شده و در درون بسته‌های IP قرار می‌گیرند. حال اگر کامپیوتر بر روی شبکهٔ اترنت واقع شده باشد، بسته‌های IP جهت انتقال بر روی کانال، در درون یک فریم اترنت جاسازی می‌شود. در چنین شرایطی، پشتهٔ پروتکلی شبیه به شکل ۶-۲۵-الف خواهد بود. تودرتویی بسته‌ها نیز در شکل ۶-۲۵-ب نشان داده شده است.

---

۱. Real-time Transport Protocol ۲. RTP Library



شکل ۶-۲۵. (الف) موقعیت RTP در پشتهٔ پروتکلی. (ب) ترتیب تودرتویی بسته‌ها.

در نتیجهٔ این سبک طراحی، به سادگی نمی‌توان گفت که RTP در کدام لایه قرار می‌گیرد. از آنجایی که RTP در فضای برنامهٔ کاربر اجرا می‌شود و نهایتاً به برنامهٔ کاربردی لینک می‌شود فلذا به یک پروتکل لایهٔ کاربرد شبیه است. از طرف دیگر، RTP یک پروتکل مستقل از لایهٔ کاربرد است و امکانات لایهٔ انتقال را عرضه می‌کند لذا شبیه به یک پروتکل لایهٔ انتقال به نظر می‌رسد. شاید بهترین تعبیر آن باشد که: «RTP یک پروتکل لایهٔ انتقال است که در لایهٔ کاربرد پیاده‌سازی شده است.»

عملکرد اصلی پروتکل RTP آنست که چندین استریم از داده‌های بی‌درنگ را در یک استریم از بسته‌های UDP مالتی‌پلکس کند. این استریم UDP را می‌توان برای یک ماشین مقصد (یعنی تک‌پخشی) یا چندین ماشین مقصد (یعنی چندپخشی) فرستاد. از آنجایی که RTP صرفاً از بسته‌های معمولی UDP بهره می‌گیرد لذا مسیریابها با این بسته‌ها به صورت ویژه رفتار نمی‌کنند مگر آن که ویژگی کیفیت خدمات (QoS) در IP پیاده و فعّال شود. خصوصاً آنکه هیچ تضمینی در خصوص تحویل این بسته‌ها، میزان تأخیر و لرزش (Jitter) وجود ندارد.

به هر بسته در یک استریم RTP، شماره‌ای داده می‌شود که یک واحد از شمارهٔ بستهٔ قبلی بیشتر است. شماره‌گذاری بسته‌ها به مقصد اجازه می‌دهد تا گم شدن احتمالی بسته‌ها را تشخیص بدهد. اگر بسته‌ای از دست برود بهترین کاری که مقصد می‌تواند انجام بدهد آنست که مقادیر بستهٔ از دست رفته را به روش درون‌یابی (Interpolation) تخمین بزند.[1] ارسال مجدد بسته‌های گم‌شده از لحاظ عملی ممکن نیست زیرا بسته‌هایی که مجدداً ارسال می‌شوند، احتمالاً آنقدر دیر به مقصد می‌رسند که دریافت آنها هیچ فایده‌ای نخواهد داشت. در نتیجه پروتکل RTP هیچ مکانیزمی برای کنترل جریان، کنترل خطا، اعلام وصول بسته (Ack) یا تقاضای ارسالِ مجدد ندارد.

محتوای بسته‌های RTP می‌تواند انواع مختلف داده را در بربگیرد و طبعاً به دلخواه برنامهٔ کاربردی کُد می‌شود. برای جلوگیری از تنوع زیاد و ناسازگاری، RTP چندین پروفایل (مثل استریم صدا) تعریف و برای هر پروفایل چندین روش کدینگ پیشنهاد کرده است. به عنوان مثال یک استریم صدا می‌تواند به روش PCM هشت بیتی با نرخ نمونه‌برداری 8KHz یا روش «مدولاسیون دلتا[2]»، «کدینگ پیشگویانه[3]»، «کدینگ GSM»، روش MP3 یا نظائر آن کُد و ارسال شود. پروتکل RTP فیلدی را در سرآیند هر بسته پیش‌بینی کرده تا به کمک آن مبداء بتواند

---

۱. درون‌یابی بسته‌های گمشده که برای صدا و تصاویر ویدیویی کاربرد دارد بدین نحو است که گیرنده از روی فریمهای قبلی و بعدی بخشی از داده‌های از دست رفته را بصورت تقریبی محاسبه و آنرا جایگزین می‌کند تا کیفیت تصویر غیرقابل تحمل نشود.

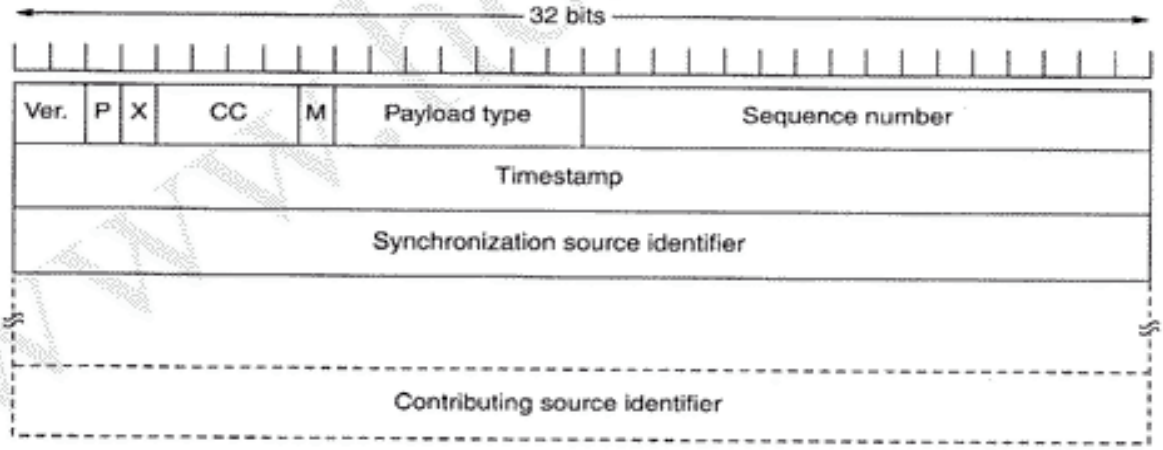
۲. Delta Modulation ۳. Predictive Encoding

نوع کدینگ خود را مشخص کند.

امکان دیگری که بسیاری از کاربردهای بی‌درنگ بدان نیاز دارند، «درج مُهر زمان» (Timestamping) برروی بسته‌هاست. ایدهٔ اصلی آنست که به مبداء اجازه بدهیم تا به اولین نمونه[1] از هر بسته یک مهر زمان نسبت بدهد. مُهر زمان نسبت به زمان شروع استریم محاسبه می‌شود فلذا فقط اختلاف مقادیر مُهر زمان بسته‌ها اهمیت دارد و مقادیر مطلق آنها بی‌ارزش است. در این مکانیزم، مقصد نیاز کمی به بافرسازی بسته‌ها خواهد داشت و فارغ از آن که بسته‌ها چه زمانی دریافت شده‌اند، براساس مهر زمان درج شده بر روی آنها، سر موعد مقرر اجرا (Play) می‌شوند.[2] درج مهر زمان نه تنها تاثیر منفی «لرزش» (Jitter) را کاهش می‌دهد بلکه امکان آنرا فراهم می‌آورد تا استریمها از لحاظ زمانی سنکرون بمانند. به عنوان مثال یک برنامهٔ تلویزیون دیجیتالی می‌تواند یک استریم ویدیو و دو استریم صدا داشته باشد. این دو استریم صدا می‌توانند یکی برای پخش فیلم با صدای اصلی و دیگری با صدای دوبله شده به زبان محلی باشد تا بیننده به میلِ خود یکی از آنها را انتخاب کند. هر یک از این استریمها از ابزارهای فیزیکی متفاوتی منشاء می‌گیرند ولیکن اگر با یک شمارندهٔ واحد شماره‌گذاری شوند، می‌توان آنها را صورت هماهنگ و سنکرون اجرا کرد، حتی اگر به صورت نامنظم و پس و پیش ارسال شده باشند.

سرآیند بستهٔ RTP در شکل ۶-۲۶ نشان داده شده است. این سرآیند از سه کلمهٔ ۳۲ بیتی و چند بخش اختیاری توسعه (Extension) تشکیل شده است. اولین کلمه، در برگیرندهٔ این فیلدهاست: فیلد Version که فعلاً ۲ است. امیدواریم که این نسخه نزدیک به نسخهٔ متکامل و نهایی آن باشد چراکه فقط یک شمارهٔ بیشتر (یعنی نسخهٔ ۳) باقی نمانده است. (البته می‌توان عدد ۳ را بدین نحو تعریف و تعبیر کرد که شمارهٔ واقعی نسخهٔ پروتکل درون سرآیند اختیاری قرار گرفته است!)

بیت **P**، بیانگر آنست که به بسته، داده‌های زائدی اضافه شده تا طول آن ضریبی از ۴ باشد. در حقیقت مقدار آخرین بایت زائد مشخص می‌کند که چند بایت به داده‌ها اضافه شده است.[3] بیت **X** مشخص می‌کند که در این بسته



شکل ۶-۲۶. سرآیند RTP (RTP Header).

---

۱. sample

۲. در ذهن خود، «مُهر زمان» را برای صدا (یا ویدیو) بدینگونه تصور کنید: اولین نمونهٔ صدا در لحظهٔ $t=t_0$ تولید شده است. بنابراین فرضاً در مهر زمان اولین بستهٔ ارسالی ۰ درج می‌شود. زمان تولید بسته‌های بعدی نسبت به اولین بسته، محاسبه و درون فیلد مهر زمان درج می‌شود. بنابراین گیرنده حتی اگر بسته‌ها را بصورت نامنظم دریافت کند براساس این مقدار می‌تواند زمان دقیق اجرای هر بسته را مشخص نماید. -م

۳. بعبارتی بیت P مشخص می‌کند که حداقل یک بایت زائد به داده‌ها اضافه شده است. آخرین بایت، تعداد بایتهای زائد را مشخص می‌کند. -م

«سرآیند توسعه» (Extension Header) وجود دارد. قالب و معنای سرآیند توسعه در پروتکل تعریف نشده است جز آنکه اولین کلمهٔ سرآیند توسعه، طول آنرا مشخص می‌کند. این سرآیند امکان آنرا فراهم آورده تا بتوان نیازهای پیش‌بینی نشده را مرتفع کرد.

فیلد CC بیانگر آنست که چند مبداء در تولید بسته دخالت داشته‌اند (از صفر تا ۱۵). در ادامه به این موضوع خواهیم پرداخت. بیت M یک علامت ویژهٔ برنامه‌های کاربردی است. مثلاً می‌توان از این علامت به معنای شروع اولین فریم ویدیو، شروع اولین کلمه در کانال صوتی (یا هر چه که برنامهٔ کاربردی بخواهد) استفاده کرد.

فیلد Payload Type روش کدینگ مورد استفاده را تعیین می‌کند (به عنوان مثال صدای هشت بیتی فشرده‌نشده، MP3 یا امثال آن). از آنجایی که هر بسته، این فیلد را با خود حمل می‌کند فلذا می‌توان در حین ارسال روش کُدینگ را تغییر داد.

فیلد Sequence Number (شمارهٔ ترتیب) یک شمارنده است که به ازای ارسال هر بستهٔ RTP یک واحد افزایش می‌یابد. از این شماره برای کشف بسته‌های گمشده استفاده می‌شود. در فیلد Timestamp ، «مُهر زمان» درج می‌شود تا مشخص گردد بستهٔ جاری (نسبت به زمان تولید اولین بسته) دقیقاً در چه زمانی تولید شده است.[۱] با درج مهر زمان، «لرزش» (Jitter) کاهش می‌یابد زیرا لحظهٔ اجرای[۲] هر بسته از زمان دریافت بسته جدا می‌شود. فیلد Synchronization Source Identifier مشخص می‌کند که بستهٔ جاری به کدام استریم تعلق دارد.[۳] به کمک این فیلد استریمهای متعدد در یک دنباله از بسته‌های UDP معمولی، مالتی‌پلکس یا دی‌مالتی‌پلکس می‌شوند.

در آخر، فیلد Contributing Source Identifier قرار می‌گیرد که در صورت استفاده، بیانگر آنست که در استودیو، میکسر وجود دارد.[۴] در چنین حالتی، میکسر موظف به سنکرونیزاسیون استریمهاست؛ در ضمن استریمهایی که باید میکس گردند در این فیلد فهرست می‌شوند.

پروتکل RTP یک خواهر کوچک به نام RTCP[۵] دارد. (شاید هم دوقلو و همزاد باشند!) این پروتکل عملیات فیدبک، سنکرون‌سازی و واسط کاربری را مدیریت می‌کند ولیکن هیچ داده‌ای را انتقال نمی‌دهد. اولین کاربرد آن گرفتن فیدبک از میزان تأخیر، لرزش (Jitter)، پهنای باند، ازدحام (congestion) و کسب آگاهی از دیگر ویژگیهای شبکه برای ماشین مبداء است. از این اطلاعات می‌توان در فرآیند کدینگ داده‌ها بهره گرفت تا مثلاً وقتی که شبکه خوب عمل می‌کند، نرخ ارسال را افزایش بدهد (یا کیفیت را بهبود ببخشد) و هنگامی که مشکلی در شبکه حادث می‌شود (مثل ازدحام) نرخ ارسال را پایین بیاورد. با در اختیار داشتن یک فیدبک دائم از شرایط فعلی شبکه، الگوریتم کدینگ می‌تواند همیشه بهترین روش را در تناسب با شرایط جاری، انتخاب نماید. به عنوان مثال وقتی پهنای باند در حین ارسال افزایش یا کاهش می‌یابد، الگوریتم کدینگ می‌تواند از روش MP3 به PCM (یا بالعکس) تغییر روش بدهد. وجود فیلد Payload Type این امکان را فراهم آورده که در هر لحظه از زمان بتوان الگوریتم کدینگ را به روش مناسب تغییر داد.

پروتکل RTCP ، سنکرون سازی بین استریمها را نیز بر عهده دارد. مشکل از آنجا ناشی می‌شود که

---

۱. دقت کنید که این زمان مربوط به تولید اولین کلمهٔ بسته یا به عبارت دقیقتر اولین نمونه (Sample) در بسته است زیرا بسته مملو از کلمات یا نمونه‌های مربوط به صدا، تصویر یا امثالهم است که از لحاظ زمانی بعد از اولین نمونه تولید شده‌اند. -م

۲. Playback time

۳. استریمهای صدا، تصویر و امثالهم می‌توانند بطور همزمان تولید یا دریافت شوند لذا باید راهی برای تفکیک و سنکرونیزاسیون بسته‌های متعلق به هر استریم وجود داشته باشد. -م

۴. بدین معنا که همانند یک استودیو، منابع تولید استریم بسیار متعددند و نهایتاً باید میکس شوند. -م

۵. Real-time Transport Control Protocol

استریمهای متفاوت از سیگنالهای ساعت با مشخصات متفاوتی مثل فرکانس، Drift بهره می‌گیرند. پروتکل RTCP می‌تواند برای سنکرون‌سازی این استریمها بکار گرفته شود.

در آخر، RTCP روشی برای نامگذاری مبداء استریمهای مختلف (مثلاً در قالب اسامی ASCII) ارائه کرده است. این اطلاعات می‌تواند بر روی صفحهٔ نمایش گیرنده، نشان داده شود تا مشخص شود چه کسی در حال صحبت یا ارسال استریم است.

اطلاعات بیشتر در خصوص RTP را می‌توانید در مرجع (Perkins, 2002) بیابید.

۵-۶ پروتکلهای لایهٔ انتقال در اینترنت: TCP[1]

UDP پروتکل بسیار ساده‌ای است و کاربردهای خاص خود را (مثل تعامل سرویس‌دهنده/مشتری یا کاربردهای چند رسانه‌ای) دارد ولیکن اغلب کاربردهای اینترنت به تحویل مطمئن و مرتب شدهٔ داده‌ها نیازمندند. UDP چنین امکانی را فراهم نمی‌کند و طبعاً به پروتکل دیگری احتیاج است. این پروتکل TCP نام دارد و بیشتر بار اینترنت را به دوش می‌کشد. اجازه بدهید جزئیات این پروتکل را بررسی نماییم.

۵-۶-۱ مقدمه‌ای بر TCP

TCP بدین منظور طراحی شد تا یک دنباله (استریم) از بایتها را به صورت مطمئن و عاری از خطا بین دو نقطهٔ پایانی (یعنی دو پروسه) از شبکه‌ای که نامطمئن و مستعد خطاست، منتقل نماید. «شبکه‌ای از شبکه‌ها» (Internetwork) از لحاظ ساختاری با یک شبکهٔ واحد تفاوت دارد زیرا هر بخش از آن دارای توپولوژی مختلف، پهنای باند، تأخیر، طول بسته و پارامترهای متفاوت است. TCP طراحی شد تا بطور پویا خود را با ویژگیهای چنین شبکهٔ ناهمگونی تطبیق بدهد و در مقابل ناکارآمدیها و شکستهای گوناگون، تحمل‌پذیر (Robust) باشد.

پروتکل TCP رسماً در RFC 793 تعریف شده است. به مرور زمان، انواع خطاها و تناقضات آن آشکار شد و در برخی از زمینه‌ها، نیازها نیز تغییر کرد. تبیین این مفاد و برخی از اصلاحیه‌ها در RFC 1122 آمده است. بسط و بهبود آن نیز در RFC 1323 تشریح شده است.

هر ماشین که از TCP پشتیبانی می‌کند باید دارای «**واحد انتقال** TCP»[2] باشد، خواه در قالب پروسیجرهای کتابخانه‌ای یا یک پروسهٔ کاربری و خواه در قالب بخشی از هستهٔ سیستم عامل. در تمامی این حالات، واحد انتقال، استریمهای TCP را مدیریت کرده و به صورت واسطی بین لایهٔ IP و پروسه‌های کاربردی انجام وظیفه می‌کند. «واحد انتقال» دنباله‌ای از داده‌های کاربر را از پروسه‌های محلی گرفته و آن را به قطعاتی با طول کمتر از 64KB می‌شکند (البته در عمل این قطعات اغلب ۱۴۶۰ بایتی هستند تا پس از الحاق سرآیند IP و TCP به آنها، جمعاً ۱۵۰۰ بایت شده و متناسب با ظرفیت یک فریم اترنت باشند.) سپس هر یک از این قطعات را [پس از افزودن سرآیند لازم] در قالب یک دیتاگرام مستقل IP ارسال می‌نماید. وقتی دیتاگرامهای حاوی داده‌های TCP به ماشین مقصد می‌رسند، به واحد انتقال تحویل داده شده و براساس آنها دنبالهٔ بایتهای اصلی ساخته خواهد شد. برای سادگی اغلب بجای «واحد انتقال TCP» (یعنی یک قطعه نرم‌افزار) و هم بجای پروتکل TCP (یعنی مجموعه‌ای از قوانین) به تنهایی از واژهٔ TCP استفاده می‌نمائیم. مضمون کلام مشخص می‌کند که منظورمان از TCP چیست. مثلاً وقتی می‌گوئیم «کاربر داده‌ها را به TCP تحویل می‌دهد» روشن است که منظور از TCP همان «واحد انتقال TCP» است.

لایهٔ IP هیچ تضمینی در تحویل صحیح و مرتب بسته‌ها نمی‌دهد لذا این وظیفه بر عهدهٔ TCP است که پس از

---

۱. Transmission Control Protocol ۲. TCP Transport Entity

انقضای مهلت مقرر آنها را از نو ارسال نماید. از طرفی ممکن است دیتاگرامها به صورت نامرتب و پس‌وپیش به مقصد برسند لذا وظیفهٔ دیگر TCP آن است که آنها را مرتب کرده و پیام اصلی را بازسازی نماید. کوتاه سخن آنکه TCP باید آن قابلیت اطمینانی را که کاربران می‌خواهند ولی IP عرضه نمی‌کند، در اختیار آنان قرار بدهد.

۶-۵-۲ مدل خدمات TCP

برای استفاده از خدمات TCP، پروسه‌های گیرنده و فرستنده یک نقطهٔ پایانی به نام «**سوکت**» ایجاد می‌کنند. (در مورد سوکت در بخش ۶-۱-۳ بحث شد.) هر سوکت دارای یک «شمارهٔ سوکت» (یا به عبارتی آدرس سوکت) است که این شماره متشکل از آدرس IP ماشین میزبان و یک عدد ۱۶ بیتی یکتا و محلی است که اصطلاحاً «**پورت**» (Port) نامیده می‌شود. «پورت» معادل نام TSAP در ادبیات پروتکل TCP است. برای بهره‌گیری از خدمات TCP باید یک اتصال بین سوکت ماشین مبداء و سوکت ماشین مقصد ایجاد شود. توابع اولیهٔ مرتبط با سوکت در شکل ۶-۵ فهرست شده‌اند.

می‌توان از یک سوکت برای برقراری چندین اتصال همزمان بهره گرفت. به عبارت دیگر ممکن است دو یا چند اتصال به یک سوکت واحد ختم شوند. هویت هر «اتصال» توسط شناسه‌های سوکت طرفین در قالب (*socket1* , *socket2*) مشخص می‌گردد. یعنی از شماره‌های مدار مجازی یا هر شناسهٔ دیگر استفاده نمی‌شود.

پورتهائی با شمارهٔ زیر ۱۰۲۴ اصطلاحاً به نام «**پورتهای شناخته‌شده**[1]» مشهورند و برای سرویسهای استاندارد رزرو شده‌اند. مثلاً هر پروسه‌ای که می‌خواهد به منظور انتقال فایل، با ماشینی ایجاد اتصال کند می‌تواند با پورت ۲۱ از ماشین مقصد که «دیمون FTP» (FTP Daemon) بدان گوش می‌کند تماس بگیرد. فهرست پورتهای شناخته‌شده در آدرس www.iana.com در دسترس می‌باشد. تاکنون بیش از ۳۰۰ شماره از آنها منتسب شده‌اند که چند تا از مشهورترین شمارهٔ پورتها را در شکل ۶-۲۷ ملاحظه می‌کنید.

در هنگام راه‌اندازی سیستم، دیمون FTP می‌تواند خود را به پورت ۲۱ متصل کند؛ دیمون TelNet به پورت ۲۳ و دیگر پروسه‌ها نیز به همین ترتیب می‌توانند خود را به یک پورت متصل نمایند. از آنجایی که اکثر این دیمونها در بیشتر زمانها بیکار هستند لذا اجرای آنها در هنگام راه‌اندازی سیستم، فضای حافظه را بیهوده تلف خواهد کرد. به جای همهٔ آنها، عموماً یک دیمون واحد که در یونیکس به «دیمون inetd»[2] مشهور است، اجرا شده و بطور همزمان خود را به تعدادی پورت متصل می‌نماید و منتظر تماسهای ورودی می‌شود. وقتی تقاضای برقراری یک اتصال دریافت شد، پروسهٔ inetd یک پروسهٔ جدید ایجاد کرده و دیمون مناسب را برای سرویس‌دهی به این تقاضا، راه اندازی و اجرا می‌نماید. بدین نحو، تمام دیمونهای دیگر به غیر از دیمون inetd فقط و فقط زمانی فعال می‌شوند که کاری برای انجام داشته باشند. دیمون inetd شمارهٔ پورتهایی که باید به آنها گوش بدهد را از فایل پیکربندی خاصّ خود استخراج می‌نماید. مسئول سیستم (admin) می‌تواند سیستم را به گونه‌ای پیکربندی کند که برخی از دایمونها به طور دائم به یک پورت گوش بدهند (مثل پورت ۸۰) و به مابقی پورتها فقط دایمون inetd گوش بدهد.

تمام اتصالات TCP، «دوطرفهٔ کامل»[3] و «نقطه‌به‌نقطه» هستند. «دو طرفهٔ کامل» یعنی ترافیک می‌تواند بطور همزمان در دو جهت جریان داشته باشد؛ نقطه‌به‌نقطه نیز یعنی یک اتصال صرفاً دو نقطهٔ پایانی (End Point) دارد.[4] TCP از ارسال چند پخشی (Multicasting) یا پخش فراگیر (Broadcast) پشتیبانی نمی‌کند.

یک اتصال TCP استریمی از بایتها را منتقل می‌نماید نه دنباله‌ای از پیامها را؛ یعنی مرز پیامهایی که بین دو نقطه

---

۱. Well-known Ports

۲. Internet Daemon

۳. Full Duplex و Point-to-Point

۴. یا بعبارتی صرفاً دو پروسهٔ مبداء و مقصد با یکدیگر محاوره می‌کنند. -م

پورت	پروتکل	کاربرد
21	FTP	انتقال فایل
23	Telnet	ورود به سیستم از راه دور
25	SMTP	پست الکترونیکی (ایمیل)
69	TFTP	پروتکل ساده انتقال فایل
79	Finger	جستجوی اطلاعات در خصوص یک کاربر
80	HTTP	تور جهان‌گستر یا همان سیستم جهانی وب
110	POP-3	دسترسی به نامه‌های الکترونیکی از راه دور
119	NNTP	اخبار یوزنت (سیستم خبررسان)

شکل ۶-۲۷. برخی از شماره‌های پورت انتساب داده شدهٔ مشهور.

مبادله می‌شوند حفظ نخواهد شد. به عنوان مثال اگر پروسهٔ فرستنده، چهار بستهٔ ۵۱۲ بایتی را بر روی یک استریم بفرستد ممکن است در قالب چهار قطعه دادهٔ ۵۱۲ بایتی یا دو قطعهٔ ۱۰۲۴ بایتی یا یک قطعه ۲۰۴۸ یا بترتیبی مشابه، به پروسهٔ گیرنده تحویل شود. (شکل ۶-۲۸ را ببینید.) هیچ راهی برای آن که گیرنده بتواند اندازهٔ قطعات داده را تشخیص بدهد، وجود ندارد.

شکل ۶-۲۸. (الف) چهار قطعهٔ ۵۱۲ بایتی در قالب دیتاگرامهای مستقل IP ارسال شده‌اند. (ب) تودهٔ ۲۰۴۸ بایتی داده‌ها، بطور یکجا و با یکبار فراخوانی READ، تحویل برنامهٔ کاربردی شده است.

در یونیکس، فایلها نیز همین ویژگی را دارند یعنی پروسه‌ای که از یک فایل می‌خواند نمی‌تواند بفهمد که آیا فایل به صورت بلوکی نوشته شده، بایت به بایت نوشته شده یا به صورت یکجا و در آن واحد نوشته شده است. همانند یک فایل در یونیکس، نرم‌افزار TCP نیز هیچ تصوری از معنای بایتها ندارد. یک بایت فقط یک بایت است!

وقتی یک برنامهٔ کاربردی داده‌های خود را به TCP تحویل می‌دهد، ممکن است TCP آنها را فوراً ارسال نماید یا آنها را موقتاً بافر کند. با این حال برخی از برنامه‌های کاربردی می‌خواهند که داده‌هایشان فوراً ارسال شود. مثلاً فرض کنید کاربری از راه دور به ماشینی وارد شده باشد (عمل login). پس از آنکه خط فرمان تکمیل و کلید Enter زده شد، انتظار می‌رود که این خط فوراً به ماشین راه دور تحویل شود و بهیچوجه تا رسیدن خط فرمان بعدی بافر نگردد. برای آن که TCP مجبور به ارسال سریع و آنی داده‌ها شود، برنامهٔ کاربردی می‌تواند از بیت پرچم PUSH (PUSH Flag) بهره بگیرد تا به TCP تفهیم شود که انتقال داده‌ها نباید به تأخیر بیفتد. (در خصوص این بیت بعداً صحبت خواهیم کرد.)

برخی از برنامه‌های کاربردی ابتدایی، از بیت پرچم PUSH به عنوان نوعی نشانه برای تعیین مرز هر پیام بهره می‌گرفتند.[1] اگرچه این حقّه گاهی اوقات کار می‌کند ولی گاهی نیز با شکست مواجه می‌شود زیرا در تمام

---

۱. به عبارت دیگر وقتی یک بستهٔ TCP که در آن بیت پرچم PUSH فعال شده، می‌رسد اگرچه بدین معناست که نباید بافر شود و سریعاً باید به برنامهٔ کاربردی تحویل گردد ولی برخی برنامه‌های کاربردی ابتدایی از این بیت به عنوان حقه‌ای جهت تفهیم پایان پیام جاری به طرف مقابل، بهره می‌گرفتند. -م

پیاده‌سازیهای TCP، در طرف گیرنده الزاماً بیت پرچم PUSH به برنامهٔ کاربردی تحویل داده نمی‌شود. مضاف براین، اگر قبل از آنکه اولین بسته با پرچم PUSH مَجال ارسال پیدا کند (مثلاً به دلیل شلوغی خط خروجی)، چند بستهٔ دیگر از همین نوع تولید و تحویل TCP شود، TCP مُجاز خواهد بود که تمام بسته‌های با علامت PUSH را پشت سرهم ادغام کرده و آن را در قالب یک دیتاگرام IP به صورت یکجا ارسال نماید که در این صورت مرز قطعات مختلف قابل تشخیص نخواهد بود.

یکی دیگر از خدمات TCP که اشاره به آن خالی از لطف نخواهد بود، موضوع «**داده‌های اضطراری**» (Urgent Data) است. وقتی کاربری که از راه دور با یک برنامه‌ای کاربردی در تعامل است کلید DEL یا CTRL-C را فشار می‌دهد (تا روند اجرای برنامه‌ای که از قبل شروع شده را قطع کند)، برنامهٔ کاربردی فرستنده، پاره‌ای اطلاعات کنترلی در «استریم داده» قرار داده و بیت پرچم URGENT را در آن فعّال و آنرا جهت ارسال به TCP تسلیم می‌کند. این رخداد باعث می‌شود که TCP به جمع‌آوری داده خاتمه بدهد و هر آنچه راکه دارد سریعاً برای طرف مقابل بفرستد.

وقتی «داده‌های اضطراری» به مقصد می‌رسند به برنامهٔ کاربردی گیرنده، «وقفه» داده می‌شود (در اصطلاح یونیکس به آن برنامه سیگنال داده می‌شود)؛ آن برنامه طبعاً کارش را متوقف کرده و استریم داده را می‌خواند تا داده‌های اضطراری را یافته و سریعاً پردازش نماید. پایان داده‌های اضطراری علامت‌گذاری شده است لذا برنامهٔ کاربردی به راحتی می‌تواند انتهای این داده‌ها را تشخیص بدهد. البته ابتدا داده‌های اضطراری علامتگذاری نمی‌شود و تشخیص آن بر عهدهٔ برنامهٔ کاربردی است. این روش فقط مکانیزم سیگنال‌دهی به پروسه‌ها را ارائه می‌کند و مدیریت بقیهٔ امور بر عهدهٔ برنامهٔ کاربردی گذاشته شده است.

۶-۵-۳ پروتکل TCP

در این بخش یک دید کلی از پروتکل TCP ارائه خواهیم کرد و در بخش بعدی سرآیند پروتکل را فیلد به فیلد بررسی می‌نماییم.

ویژگی کلیدی در پروتکل TCP (که طراحی کل پروتکل از آن تاثیر پذیرفته) آنست که هر بایت ارسالی بر روی یک اتصال TCP دارای یک شمارهٔ ترتیب ۳۲بیتی است. زمانی که اینترنت شروع به کار کرد، خطوط ارتباطی بین مسیریابها، اغلب خطوط اجاره‌ای 56kbps بودند و حتی اگر یک ماشین با تمام سرعت، اقدام به ارسال داده می‌کرد بیش از یک هفته طول می‌کشید تا این شمارهٔ ۳۲بیتی به آخر رسیده و به صفر برگردد. در سرعتهای جدید شبکه، شماره‌های ترتیب می‌توانند در مدت کوتاه و خطرناکی به آخر رسیده و تکرار شوند. (بعداً به این مسئله بیشتر می‌پردازیم.) برای اعلام وصول داده‌ها (یعنی Acknowledgement) و «مکانیزم پنجره» نیز از فیلدها و شماره‌های ترتیب متفاوتی استفاده شده است.

واحدهای انتقال TCP در سمت فرستنده و گیرنده، داده‌ها را در قالب یکسری «**قطعه**» (Segment) ردّ و بدل می‌کنند. هر قطعهٔ TCP متشکل از ۲۰ بایت سرآیند ثابت و اجباری (و در صورت نیاز یک بخش اختیاری در سرآیند) است و به دنبال آن به تعداد صفر یا چند بایت داده قرار می‌گیرد. نرم‌افزار TCP راساً در مورد اندازهٔ هر قطعه تصمیم می‌گیرد.[1] TCP ممکن است داده‌هایی را که در چند مرحله جهت ارسال در استریم نوشته شده‌اند، جمع‌آوری کرده و آنها را در قالب یک قطعهٔ TCP بفرستد و یا بالعکس داده‌هایی را که در یک مرحله تحویل می‌شوند ، در چند قطعه ارسال نماید. دو عامل می‌تواند محدودکنندهٔ اندازهٔ قطعات باشد. اول آن که هر قطعه

---

۱. یعنی برنامهٔ کاربردی نمی‌تواند در خصوص آنکه بسته‌ها (قطعه‌ها) در چه اندازه‌ای ارسال شود، اعمال نظر کند. -م

(شامل سرآیند آن) باید بتواند در فضای ۶۵۵۱۵ بایتی فیلد داده از بستهٔ IP جا بگیرد؛ دوم آن که در هر شبکه پارامتری به نام MTU (Maximum Transfer Unit) (یعنی حداکثر طول بستهٔ قابل انتقال) وجود دارد و اندازهٔ هر بسته باید متناسب با این مقدار باشد. در عمل، MTU عموماً ۱۵۰۰ بایت است (اندازهٔ فیلد حمل دادهٔ اترنت) و طبعاً حد بالایی طول هر قطعه باید متناسب با این مقدار باشد.

TCP از پروتکل «پنجرهٔ لغزان» بهره گرفته است: به محض آن که فرستنده، قطعه‌ای را ارسال می‌کند یک تایمر برای آن روشن می‌نماید. وقتی این قطعه به مقصد رسید، واحد TCP (TCP Entity)، قطعه‌ای را بر می‌گرداند که در آن «شمارهٔ تصدیق بستهٔ دریافتی» (Ack.No.) درج شده است. این قطعه می‌تواند حاوی داده‌های متقابل گیرنده باشد و در صورت عدم وجود هر گونه داده، بدون داده ارسال شود. شمارهٔ تصدیق بستهٔ دریافتی در حقیقت حاوی شمارهٔ ترتیب بایتی است که گیرنده از آن شماره به بعد منتظر دریافت آنهاست. اگر مهلت تایمر به پایان برسد و در این زمان دریافت قطعهٔ ارسالی، تصدیق نشود، فرستنده، قطعهٔ قبلی را از نو ارسال می‌کند.

اگرچه این پروتکل ساده به نظر می‌رسد ولی پیچیدگیها و ظرافتهایی چند، در آن نهفته است که در ادامه به آن خواهیم پرداخت. قطعات ارسالی می‌توانند به صورت نامرتب به گیرنده برسند؛ مثلاً اگر بایتهای ۳۰۷۲ تا ۴۰۹۵ در قالب یک قطعه برسد نمی‌توان آن را اعلام وصول کرد چراکه فرضاً بایتهای ۲۰۴۸ تا ۳۰۷۱ نرسیده‌اند. همچنین ممکن است قطعات ارسالی آنقدر در شبکه معطل شوند که مهلت فرستنده منقضی شده و آنها را از نو ارسال نماید و منجر به تولید قطعات تکراری شود. همچنین ارسال مجدد داده‌ها ممکن است در قالب قطعات جدیدی صورت بگیرد.[1] فلذا این موضوع نیز باید به دقت مدیریت و نظارت شود تا گیرنده بتواند بر دنبالهٔ بایتهایی که دریافت کرده و آنهایی که دریافت نکرده به دقت کنترل داشته باشد. خوشبختانه چون هر بایت در یک دنباله (استریم) دارای یک آفست یکتاست فلذا انجام چنین کار مهمی امکان‌پذیر است.

TCP باید بتواند تمام این مسائل را به نحو کارآمد و مؤثری حل و فصل نماید. در ضمن تلاشهای وافری برای بهینه‌سازی کارآیی استریمهای TCP صورت گرفته است تا در مواجهه با مشکلات شبکه خود را تطبیق بدهد. تعدادی از این الگوریتمها که در اغلب پیاده‌سازیهای عملی TCP بکار رفته را در ادامه معرفی کرده‌ایم.

۶-۵-۴ سرآیند قطعهٔ TCP

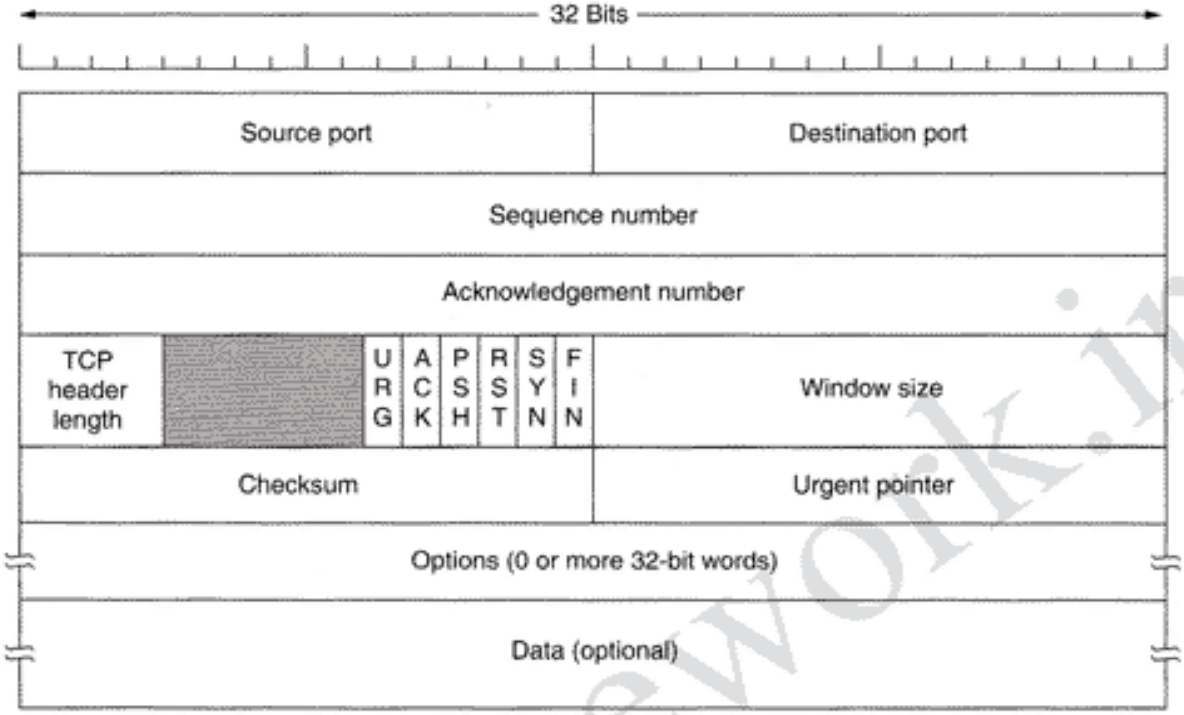
شکل ۶-۲۹، ساختار یک قطعهٔ TCP (TCP Segment) را نشان می‌دهد. هر قطعه با یک سرآیند ۲۰ بایتی و با قالب ثابت شروع می‌شود. پس از این سرآیند ثابت، ممکن است یک سرآیند اختیاری قرار بگیرد. در صورت عدم وجود سرآیند اختیاری، در ادامه می‌تواند حداکثر ۶۵۴۹۵=(۲۰-۲۰-۶۵۵۳۵) بایت داده قرار بگیرد که کسر بیست بایت اول مربوط به سرآیند IP و بیست بایت دوم مربوط به سرآیند TCP است. قطعات بدون داده [یعنی فقط سرآیند]، معتبر و قانونی هستند و عموماً برای اعلام وصول داده‌ها (Ack) و پیامهای کنترلی کاربرد دارند. حال اجازه بدهید که سرآیند TCP را فیلد به فیلد کالبدشکافی کنیم.

فیلدهای «پورت مبداء» (Source Port) و «پورت مقصد» (Destination Port) نقاط انتهایی دو طرف یک اتصال را مشخص می‌نمایند.[2] شمارهٔ پورتهای شناخته‌شده و مشهور در آدرس www.iana.org فهرست شده‌اند ولی هر ماشین میزبان می‌تواند به دلخواه خود، آنها را به پروسه‌ها اختصاص بدهد. یک شمارهٔ پورت ۱۶ بیتی به همراه آدرس IP ماشین میزبان، آدرس ۴۸ بیتی یک نقطهٔ پایانی را تشکیل می‌دهد. آدرس نقاط پایانی مبداء و

---

۱. یعنی مثلاً یک قطعه حاوی ۱۵۰۰ بایت ارسال شده ولی چون وصول آن تصدیق نشده ممکن است ارسال مجدد آن در قالب دو قطعهٔ ۱۰۰۰ و ۵۰۰ بایتی انجام شود. -م

۲. یعنی این دو فیلد هویت پروسه‌های گیرنده و فرستنده را تعیین می‌نمایند. -م



شکل ۶-۲۹. سرآیند TCP.

مقصد، هویت یک اتصال را تبیین می‌کنند.

فیلدهای «شمارهٔ ترتیب» (Sequence Number) و «شمارهٔ تصدیق» (Acknowledgement Number) عملکرد طبیعی خود را دارند یعنی اولی شمارهٔ ترتیب قطعهٔ داده و دومی اعلام وصول داده‌ها است. دقت کنید که فیلد «شمارهٔ تصدیق»، شمارهٔ بایتی را مشخص می‌کند که از آن بایت به بعد منتظر دریافت داده‌هاست نه آخرین بایت دریافتی.[1] هر دوی این فیلدها ۳۲ بیتی هستند زیرا در استریم TCP هر بایت دارای شمارهٔ ترتیب است.

فیلد TCP Header Length مشخص می‌کند که سرآیند قطعهٔ TCP چند کلمهٔ ۳۲ بیتی است. از آنجایی که فیلد Options اندازهٔ متغیری دارد فلذا به وجود این فیلد که طول سرآیند را مشخص می‌کند، نیاز خواهد بود. از دیدگاه فنی این فیلد «نقطهٔ شروع داده‌ها در هر قطعه» را بر مبنای کلمات ۳۲ بیتی مشخص می‌کند ولی از دیدگاه دیگر می‌توان مقدار این فیلد را «طول سرآیند قطعهٔ TCP» بر مبنای کلمه فرض کرد ولیکن تأثیر هر دوی این تعابیر یکی است.

در ادامه یک فیلد شش بیتی بلااستفاده آمده است. این حقیقت که از چنین فیلدی برای حدود ربع قرن استفاده نشده، مؤید آن است که TCP با تفکر و بینش بسیار جامعی طراحی شده است. کمتر پروتکلی برای اصلاح اشکالات طرح اصلی TCP، از این فیلد استفاده کرده است.

در ادامه شش پرچم تک‌بیتی (Flag) آمده است: اگر درون فیلد Urgent Pointer مقدار معتبری قرار گرفته باشد، بیت پرچم URG به ۱ تنظیم می‌شود. مقدار فیلد Urgent Pointer مشخص می‌کند که «داده‌های اضطراری» (نسبت به اولین بایت داده‌های قطعهٔ جاری) از چه موقعیتی شروع می‌شوند. در حقیقت این فیلد نقش «پیام وقفه» (Interrupt Message) را ایفاء می‌کند. همانگونه که قبلاً اشاره کردیم این فیلد روش سرراستی برای سگینال

---

۱. به عبارتی اگر در فیلد شمارهٔ تصدیق مثلاً عدد ۱۲۳۴۵ درج شده باشد بدین معنا تعبیر می‌شود که تا بایت ۱۲۳۴۴ دریافت شده و باید از بایت شمارهٔ ۱۲۳۴۵ به بعد ارسال شود. -م

دادن فرستنده به گیرنده (از راه دور) است.

برای آن که مشخص شود فیلد «شمارهٔ تصدیق» (Acknowledgement Number) دارای مقدار معتبر است، بیت پرچم ACK به ۱ مقداردهی می‌شود. اگر بیت ACK معادل صفر باشد، قطعهٔ TCP جاری دارای هیچ «شمارهٔ تصدیق» معتبری نیست و مقدار آن نادیده گرفته می‌شود.

بیت PSH مشخص کننده آن است که داده‌های قطعهٔ جاری باید سریعاً تحویل داده شود (PUSH شود). با تنظیم این بیت در سرآیند قطعهٔ TCP، از گیرنده خواهش می‌شود که داده‌های موجود در قطعه را بلافاصله تحویل برنامهٔ کاربردی بدهد و تا پر شدن بافر (که معمولاً برای افزایش کارآیی و سرعت کاربرد دارد) منتظر نماند.

از بیت RST زمانی استفاده می‌شود که TCP به هر دلیلی (مثل از کار افتادن ماشین میزبان) بخواهد یک اتصال را بطور ناگهانی و یکطرفه قطع کند. همچنین زمانی که TCP بخواهد یک قطعهٔ نامعتبر یا تلاش برای برقراری یک اتصال را نپذیرد از این بیت استفاده می‌کند. کلاً هر گاه قطعه‌ای دریافت شود که در آن بیت RST به ۱ تنظیم شده، نشان‌دهندهٔ وجود یک مشکل است.

از بیت SYN برای برقراری یک اتصال استفاده می‌شود. در حقیقت تقاضای برقراری اتصال با ارسال قطعه‌ای با مشخصات SYN=1 و ACK=0 انجام می‌گیرد که در آن SYN=1 علامت تقاضای برقراری ارتباط و ACK=0 به معنای عدم وجود مقدار معتبر در فیلد Achnowledgement Number است. پاسخ به این تقاضا به شکل SYN=1 و ACK=1 است و در فیلد Ack.No مقدار معتبری وجود دارد. [بعداً بدین مورد خواهیم پرداخت.] در اصل، بیت SYN برای اعلام هر یک از پیامهای CONNECTION REQUEST و CONNECTION ACCEPTED بکار می‌رود و تمایز بین این دو پیام با بیت ACK مشخص می‌شود.

بیت FIN برای خاتمه دادن به یک اتصال بکار می‌رود. ۱ بودن این بیت بدان معناست که فرستنده، دادهٔ دیگری برای ارسال ندارد. ولیکن پروسه‌ای که با ارسال چنین قطعه‌ای اتصال را از سمت خود می‌بندد می‌تواند تا هر زمانی به دریافت داده‌های طرف مقابل ادامه بدهد. قطعات حاوی بیت SYN=1 یا FIN=1 دارای شمارهٔ ترتیب هستند، فلذا تضمین می‌شود که این بسته‌ها به درستی پردازش خواهند شد.[۱]

کنترل جریان در TCP به کمک یک پنجرهٔ لغزان با اندازه متغیر انجام می‌شود. مقدار درج شده در فیلد Window Size در هر بستهٔ TCP، به طرف مقابل تفهیم می‌کند که از بایتی که شمارهٔ آن در فیلد Acknowledgement Number درج شده، به اندازهٔ چند بایت حق ارسال داده دارد. درج عدد صفر در فیلد Window Size کاملاً قانونی و معتبر است و در حقیقت بیان می‌کند که اگرچه تا بایت شمارهٔ Acknowledgement Number-1، دریافت شده ولیکن به دلیل کمبود شدید فضای بافر، تا اطلاع ثانوی نمی‌تواند پذیرای دادهٔ بیشتری باشد. گیرنده بعداً می‌تواند به فرستنده مجوز ارسال بدهد: این کار با فرستادن یک بسته که در آن فیلد Acknowledgement Number همان مقدار قبلی را دارد و فیلد Window Size آن غیرصفر است، صورت می‌گیرد.

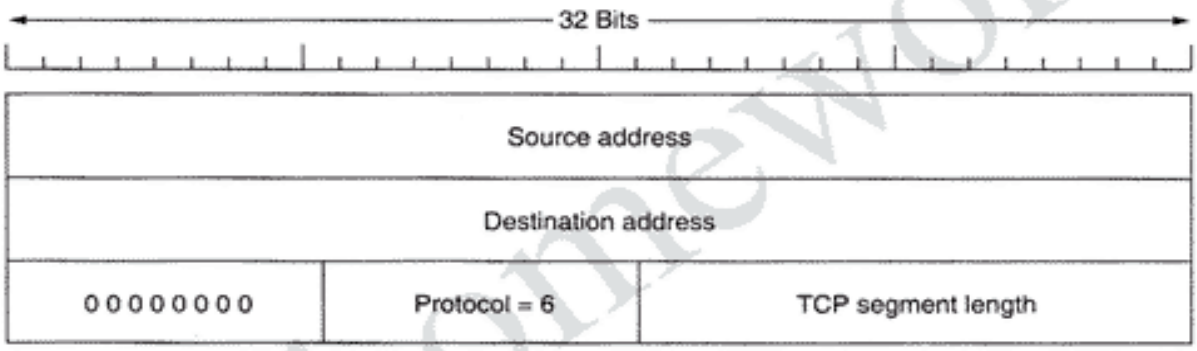
در پروتکلهای فصل سوم، تصدیق وصول فریمها و مجوز ارسال فریمهای جدید یکی هستند. [یعنی دریافت یک پیغام Ack، ضمن تصدیق وصول بسته به فرستنده اجازه می‌دهد که بستهٔ جدیدی بفرستد.] این مسئله از آنجا ناشی می‌شود که طول پنجرهٔ هر یک از پروتکلهای یاد شده ثابت و بدون تغییر است. در پروتکل TCP، اعلام وصول داده‌ها (یعنی Ack) و مجوز ارسال دادهٔ بیشتر از هم تفکیک شده‌اند. در نتیجه یک گیرنده می‌تواند بگوید:

---

۱. به عبارت دیگر اگر به هر دلیلی یک بستهٔ FIN=1 برای پروسه‌ای ارسال شود، پردازش نخواهد شد مگر آن که شمارهٔ ترتیب آن صحیح و دقیق باشد.

«من تا بایت شمارهٔ k را به درستی دریافت کرده‌ام ولیکن فعلاً تمایلی‌به دریافت دادهٔ دیگر ندارم!» تفکیک این دو مفهوم و داشتن پنجره‌ای با طول متغیر، قابلیت انعطاف بیشتری به پروتکل اعطا می‌کند. در ادامه این موضوع را به تفصیل بررسی خواهیم کرد.

فیلد Checksum یک کد کشف خطا است و جهت اطمینان از صحت داده‌ها کاربرد دارد. این کد حاصل جمع سرآیند، داده‌ها و یک «شبه‌سرآیند فرضی» (Pseudoheader) است. قالب شبه‌سرآیند فرضی در شکل ۶-۳۰ مشخص شده است. برای محاسبهٔ این کُد ابتدا فیلد Checksum صفر فرض می‌شود و در صورت فرد بودن تعداد بایتها، تعدادی صفر زائد به انتهای داده‌ها اضافه می‌گردد تا تعداد بایتها زوج شود. الگوریتم محاسبهٔ Checksum بسیار ساده است: مجموعهٔ بایتها به صورت کلمات ۱۶ بیتی (یعنی دو بایت دو بایت) با هم جمع شده و حاصل جمع به صورت «متمم ۱» (One's Complement) منفی می‌شود و درون فیلد Checksum قرار می‌گیرد. نتیجتاً وقتی دریافت‌کننده این محاسبه بر روی کل قطعه (شامل فیلد Checksum) انجام می‌شود نتیجهٔ آن باید صفر باشد. در غیر این صورت داده‌ها قابل اعتماد و سالم نیستند.



شکل ۶-۳۰. شبه‌سرآیندی که در محاسبهٔ کد کشف خطای Checksum TCP دخالت داده می‌شود.

شبه‌سرآیند (Pseudoheader) از فیلدهای زیر تشکیل شده‌اند: آدرسهای IP سی و دو بیتی مبداء و مقصد، شمارهٔ پروتکل TCP (یعنی عدد ۶) و تعداد کل بایتهای قطعهٔ TCP (شامل سرآیند آن). وارد کردن این شبه‌سرآیند در محاسبهٔ مقدار Checksum اگرچه به کشف بسته‌هایی که به اشتباه دریافت شده‌اند کمک خواهد کرد ولیکن اصول و قواعد تفکیک سلسله‌مراتب پروتکلها و مخفی ماندن جزئیات هر لایه را نقض می‌کند چراکه آدرسهای IP متعلق به لایهٔ IP است و ربطی به لایهٔ TCP ندارد. UDP نیز برای محاسبهٔ کد Checksum از همین شبه سرآیند استفاده می‌کند.

فیلد Options برای درج گزینه‌های اختیاری در قطعهٔ TCP تعریف شده است و می‌توان از آن برای اضافه کردن فیلدهایی که در پروتکل TCP پیش‌بینی نشده، استفاده کرد. مهمترین گزینهٔ اختیاری، گزینه‌ای است که به ماشین میزبان اجازه می‌دهد طول حداکثر قطعات TCP که تمایل به پذیرش آنها را دارد، به اطلاع طرف مقابل برساند. استفاده از قطعات بزرگ، کارآمدتر از ارسال قطعات کوچک است زیرا سربار ناشی از سرآیند ۲۰ بایتی، بر روی دادهٔ بیشتری سرشکن می‌شود ولی ممکن است ماشینهای میزبان کوچک از عهدهٔ پذیرش قطعات بزرگ برنیایند. در خلال برقراری یک اتصال، هر یک از طرفین طول حداکثر قطعات مورد پذیرش خود را به اطلاع طرف مقابل خود می‌رساند. اگر ماشینی از این گزینه استفاده نکند، اندازهٔ پیش فرض داده‌ها در هر قطعه ۵۳۶ بایت خواهد بود. تمام ماشینهای میزبان در اینترنت موظف به پذیرش قطعات TCP با اندازهٔ ۵۵۶=۲۰+۵۳۶ بایت هستند. الزامی به یکسان بودن اندازهٔ قطعات ارسالی در دو جهت وجود ندارد.

برای خطوط با پهنای باند بالا یا تأخیر زیاد (یا هر دو)، پنجرهٔ ۶۴کیلوبایتی می‌تواند مشکل‌زا باشد.[۱] در یک خط T3 (با سرعت 44.736Mbps)، تخلیه و ارسال یک بافر ۶۴کیلوبایتی فقط ۱۲ میلی‌ثانیه طول می‌کشد. یا مثلاً اگر تأخیر انتشار رفت‌وبرگشت [یعنی رفت بسته و برگشت Ack آن] ۵۰میلی‌ثانیه طول بکشد (که برای خطوط فیبرنوری بین قاره‌ای کاملاً طبیعی است)، فرستنده سه چهارم از زمان مفید خود را در انتظار بازگشت پیام اعلام وصول (Ack) تلف می‌کند. برای ارتباطات ماهواره‌ای وضع از این هم بدتر است.[۲] استفاده از پنجره‌ای با طول بزرگتر از ۶۴کیلوبایت می‌تواند از توقف فرستنده به دلیل پر شدن فضای پنجره جلوگیری کند ولیکن با فیلد ۱۶ بیتی Window Size نمی‌توان اندازهٔ چنین بافری را اعلام کرد. در RFC 1323 یک گزینهٔ اختیاری به نام Window Scale پیشنهاد شده که به کمک آن فرستنده و گیرنده می‌توانند «**ضریب مقیاس پنجره**» را به اطلاع یکدیگر برسانند. این گزینهٔ اختیاری در فیلد Options قرار خواهد گرفت و اجازه می‌دهد که هر یک از طرفین بتوانند فیلد Window Size خود را تا ۱۴ بیت به سمت چپ شیفت بدهند. بدین ترتیب اندازهٔ پنجره می‌تواند تا 2^{30} بایت (یک گیگابایت) افزایش یابد. امروزه در اغلب پیاده‌سازیهای TCP، از این گزینه حمایت می‌شود.

گزینهٔ دیگری که در RFC 1106 پیشنهاد شده و در اغلب پیاده‌سازیهای TCP از آن پشتیبانی می‌شود آن است که به جای استفاده از پروتکل Go Back n (عقبگرد به اندازهٔ n) از پروتکل «**تکرار انتخابی**» (Selective Repeat) بهره گرفته شود. اگر گیرنده، ابتدا قطعه‌ای خراب و به دنبال آن چندین قطعهٔ بزرگ و سالم دریافت نماید، در صورتی که از پروتکل معمولی TCP استفاده شود، فرستنده پس از انقضای مهلت مقرر تمام قطعات اعلام وصول نشده را از نو ارسال خواهد کرد (حتی بسته‌هایی را که سالم رسیده‌اند) زیرا از پروتکل Go Back n بهره گرفته شده است. در RFC 1106 مفهوم جدیدی به نام NAK (پیغام عدم وصول) معرفی شده که به گیرنده اجازه می‌دهد قطعهٔ خاصی را درخواست بدهد. پس از دریافت قطعهٔ ناقص، وصول تمام داده‌هایی که از قبل بافر شده‌اند به یکباره تصدیق می‌شود و بدین ترتیب حجم داده‌هایی که بیهوده ارسال مجدد می‌شوند، کاهش خواهد یافت.

۶-۵-۵ برقراری اتصال TCP

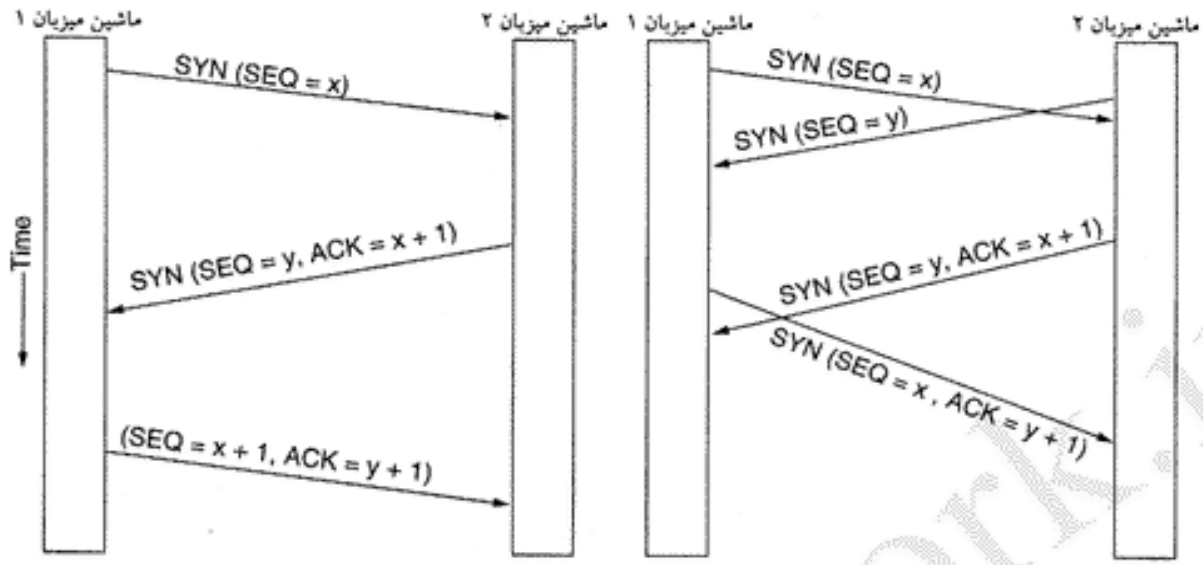
در پروتکل TCP برای برقراری اتصال از روش «**دست‌تکانی سه‌مرحله‌ای**» (Three-Way Handshake) (که در بخش ۶-۲-۲ تشریح شد) بهره گرفته می‌شود. برای آن که اتصالی ایجاد شود باید یکی از طرفین (که آن را سرویس‌دهنده می‌نامیم) از طریق اجرای توابع اولیهٔ LISTEN و ACCEPT منتظر تقاضاهای ورودی باقی بماند. (البته می‌تواند مبداء مورد نظر خود را تعیین کند یا آن که مبداء خاصی را مدّ نظر نداشته باشد.) سمت مقابل (یعنی مشتری)، تابع اولیه CONNECT را اجرا کرده و آدرس IP و پورت کسی را که می‌خواهد با آن اتصال برقرار کند و همچنین طول حداکثر قطعات TCP مورد پذیرش (و در صورت نیاز برخی از اطلاعات کاربری نظیر کلمهٔ عبور) را مشخص می‌نماید. تابع اولیهٔ CONNECT، یک قطعهٔ TCP با مشخصات SYN=1 و ACK=0 برای طرف مقابل فرستاده و منتظر برگشت پاسخ باقی می‌ماند.

وقتی چنین بسته‌ای به مقصد می‌رسد، واحد TCP در آنجا ابتدا بررسی می‌کند که آیا پروسه‌ای توسط تابع LISTEN در حال گوش دادن به شمارهٔ پورت مشخص شده در فیلد شمارهٔ پورت مقصد (Destination Port) هست یا خیر؟ اگر چنین نبود با ارسال یک قطعهٔ TCP حاوی بیت RST=1، تقاضای برقراری اتصال را ردّ می‌کند. اگر پروسه‌ای در حال گوش دادن به پورت مربوطه باشد، قطعهٔ TCP ورودی، بدان پروسه تحویل داده خواهد

---

۱. از آنجایی که فیلد Window Size شانزده بیتی است لذا طول پنجره به 64KB (۶۵۵۳۵ بایت) محدود شده‌است. -م

۲. تأخیر رفت بسته و برگشت Ack آن، در کانالهای ماهواره‌ای ژئوسنکرون حدود ۸۰۰میلی‌ثانیه است. -م



شکل ۶-۳۱. (الف) ایجاد یک اتصال TCP در شرایط معمولی (ب) تلاقی دو تماس.

شد. پروسهٔ مربوطه می‌تواند آن اتصال را بپذیرد یا رد کند. اگر پذیرفته شود یک قطعهٔ تصدیق (Acknowledgement) برگردانده خواهد شد. توالی ارسال قطعات TCP جهت برقراری اتصال، در شکل ۶-۳۱-الف (در حالت طبیعی) نشان داده شده است.

در حالتی که تصادفاً دو ماشین بطور همزمان بخواهند اتصالی بین دو سوکت یکدیگر برقرار نمایند، رخدادهایی با ترتیب شکل ۶-۳۱-ب اتفاق می‌افتد. نتیجهٔ نهایی این رخدادها آن است که فقط و فقط یک اتصال برقرار می‌شود نه دو تا، زیرا هویت هر اتصال برحسب نقاط پایانی آن [یعنی آدرسهای IP و پورت دو پروسهٔ پایانی] شناسایی می‌شود؛ اگر اولین اتصال برقرار شده با شناسهٔ (x,y) مشخص شود، اتصال دوم نیز دارای همین شناسه خواهد بود و در جدول اتصالات فعال فقط یک درایه (Entry) با شناسهٔ (x,y) درج خواهد شد.

شمارهٔ ترتیب اولیهٔ پیشنهادی در هر اتصال به دلیلی که قبلاً بدان اشاره کردیم از صفر شروع نخواهد شد. برای انتخاب این شماره از ساعتی استفاده می‌شود که هر ۴ میکروثانیه تیک می‌زند. برای اطمینان بیشتر هر گاه ماشینی از کار بیفتد می‌تواند به مدت حداکثر طول عمر بسته‌هایی که از اتصال قبلی در جایی از شبکهٔ اینترنت سرگردان مانده‌اند، صبر کند و سپس از نو راه‌اندازی شود.

۶-۵-۶ خاتمه دادن به اتصال TCP

هر چند اتصالات TCP، دو طرفهٔ کامل (Full Duplex) هستند ولی برای فهم چگونگی قطع اتصال ، بهتر است یک اتصال TCP را به صورت یک جفت اتصال یکطرفه (Simplex) در نظر بگیریم. هر اتصال یکطرفه، مستقل از جفت خود قطع می‌شود. برای خاتمه دادن به یک اتصال، هر یک از طرفین می‌توانند یک قطعهٔ TCP که در آن بیت FIN به ۱ تنظیم شده، ارسال نمایند. چنین قطعه‌ای بدین معناست که دادهٔ دیگری جهت ارسال وجود ندارد. وقتی دریافت این قطعه تصدیق شد اتصال در یکی از جهات، خاتمه می‌یابد ولیکن جریان داده‌ها در جهت مخالف می‌تواند بطور نامحدود ادامه داشته باشد. وقتی که اتصال در هر دو جهت قطع شد، اتصال TCP خاتمه می‌یابد. طبیعتاً برای خاتمه دادن به یک اتصال، به ارسال چهار قطعهٔ TCP نیاز است: یکی برای FIN و یکی برای ACK ، در هر یک از جهات. با این حال ممکن است اولین ACK برگشتی با FIN طرف مقابل در یک بسته ادغام شود و تعداد این بسته‌ها به سه تا کاهش یابد.

دقیقاً مشابه با تماسهای تلفنی که در آن هر دو نفر می‌توانند با خداحافظی، گوشی تلفن خود را بگذارند، دو پروسهٔ انتهایی یک اتصال نیز ممکن است بطور همزمان قطعه‌ای حامل FIN=1 بفرستند. هر یک از این تقاضاهای قطع اتصال به روش معمولی تصدیق شده و به اتصال خاتمه داده می‌شود.

برای آن که مشکل «**دو سپاه**» (بخش ۶-۲-۳) پیش نیاید، از تایمر نیز استفاده شده است. اگر پاسخ به FIN به مدت دو برابر طول عمر حداکثر هر بسته دریافت نشود، فرستندهٔ FIN، بطور یک جانبه به اتصال خاتمه خواهد داد. طرف مقابل نیز عاقبت متوجه خواهد شد که هیچکسی در طرف مقابل به او گوش نمی‌دهد و مهلت او نیز منقضی می‌شود و به اتصال خاتمه خواهد داد. اگرچه این راهکار چندان کامل و مطلوب نیست ولیکن از آنجایی که از لحاظ تئوری هیچ راهکار مطمئن و کاملی وجود ندارد مجبور به برگزیدن این راهکار هستیم. البته این مشکلات در عمل به ندرت بروز می‌کند.

۶-۵-۷ مدلسازی فرآیند مدیریت اتصال در TCP

مراحل لازم برای برقراری و ختم اتصال در TCP را می‌توان در قالب یک «**ماشین حالت محدود**» (Finite State Machine) با یازده وضعیت مختلف که در جدول ۶-۳۲ فهرست شده، نشان داد. در هر «**وضعیت**» وقوع برخی از «**رخدادها**» معتبر هستند. وقتی یک رخداد معتبر حادث می‌شود سلسله‌ای از «**کنشها**» (Actions) انجام خواهد شد. برای برخی دیگر از رخدادها نیز، فقط به گزارش خطا بسنده می‌شود.

هر اتصال از وضعیت CLOSED آغاز می‌شود. اگر یکی از رخدادهای «بازکردن غیرفعال» (با اجرای تابع اولیهٔ LISTEN) یا «بازکردن فعال»[1] (با اجرای تابع اولیهٔ CONNECT) رخ بدهد، اتصال از این «وضعیت» خارج خواهد شد. اگر طرف مقابل اتصال دقیقاً در وضعیت معکوس قرار داشته باشد یک اتصال برقرار شده و وضعیت جدید اتصال، ESTABLISHED خواهد شد. قطع یک اتصال می‌تواند توسط هر یک از طرفین شروع شود. وقتی فرآیند قطع اتصال تکمیل شد، اتصال به وضعیت قبلی CLOSED باز خواهد گشت.

وضعیت (حالت)	توصیف
CLOSED	هیچ اتصالی فعال یا معلق (و منتظر) نیست.
LISTEN	سرویس‌دهنده منتظر یک تماس ورودی (تقاضای اتصال) است.
SYN RCVD	یک تقاضای برقراری اتصال دریافت شده است. منتظر ACK آن است.
SYN SENT	برنامه کاربردی (مشتری) شروع به ایجاد یک اتصال کرده است.
ESTABLISHED	وضعیت عادی مبادله داده (وضعیت برقراری اتصال)
FIN WAIT 1	برنامه کاربردی اعلام کرده کارش به اتمام رسیده است.
FIN WAIT 2	طرف مقابل نیز با قطع اتصال و خاتمه تماس موافقت کرده است.
TIMED WAIT	حالت انتظار برای آنکه تمام بسته‌های سرگردان از بین بروند.
CLOSING	طرفین بطور همزمان سعی در بستن اتصال کرده‌اند.
CLOSE WAIT	طرف مقابل مراحل قطع ارتباط را آغاز کرده است.
LAST ACK	حالت انتظار برای آنکه تمام بسته‌های سرگردان ACK از بین بروند.

شکل ۶-۳۲. وضعیتهایی که در مدل «ماشین حالت محدود» از مدیریت اتصال TCP بکار می‌رود.

۱. Active Open / Passive Open

«ماشین حالت محدود» برای TCP، در شکل ۶-۳۳ به تصویر کشیده شده است. حالتی که در آن ماشین مشتری بطور فعّال به یک سرویس‌دهندهٔ غیرفعال متصل می‌شود با خطوط تیره رنگ نشان داده شده است: خطوط توپر برای تقاضای مشتری و نقطه چین برای پاسخ سرویس‌دهنده. خطوط کمرنگ، توالی رخدادهای نامتعارف را نشان می‌دهند. کنار هر خط در شکل ۶-۳۳ یک زوج مشخصه به صورت event/action (رخداد/کنش) نوشته شده است. هر رخداد می‌تواند ناشی از یک فراخوانی سیستمی باشد که توسط کاربر آغاز می‌شود (مثل فراخوانیهای CONNECT ، LISTEN ، SEND یا CLOSE)، یا در اثر دریافت یک قطعهٔ TCP (مثل SYN ، FIN ، ACK یا RST) بروز کند و یا در اثر انقضای مهلت تایمر حادث شود. «کنشی» که در حین بروز یک رخداد انجام می‌شود ، ارسال یک قطعهٔ کنترلی (مثل SYN ، FIN یا RTS) است یا هیچ کاری صورت نمی‌گیرد (که در شکل با علامت - مشخص شده است). توضیحات لازم درون پرانتز نشان داده شده است.

برای فهم دیاگرام، بهتر آنست که ابتدا مسیر منشعب از مشتری (خطوط توپر ضخیم) و بعد از آن مسیر سرویس‌دهنده (خطوط ضخیم نقطه چین) دنبال شود. وقتی برنامهٔ کاربردی بر روی ماشین مشتری تقاضای CONNECT صادر می‌کند، «واحد TCP» در آن ماشین، ابتدا یک رکورد برای آن اتصال ایجاد کرده و پس از علامتگذاری اتصال در وضعیت SYNSENT یک قطعهٔ SYN ارسال می‌نماید. دقت کنید که ممکن است چندین اتصال از طرف چند برنامهٔ کاربردی بطور همزمان باز شده (یا در حال باز شدن) باشد فلذا به ازای هر اتصال یک «رکورد وضعیت» مستقل ایجاد و وضعیت آن اتصال در رکورد مربوطه درج می‌شود. وقتی قطعهٔ کنترلی حاوی SYN+ACK دریافت می‌شود، TCP با ارسال ACK نهایی، فرآیند دست تکانی سه‌مرحله‌ای را تکمیل کرده و به ESTABLISHED تغییر وضعیت می‌دهد. در این وضعیت می‌توان داده فرستاد یا دریافت کرد.

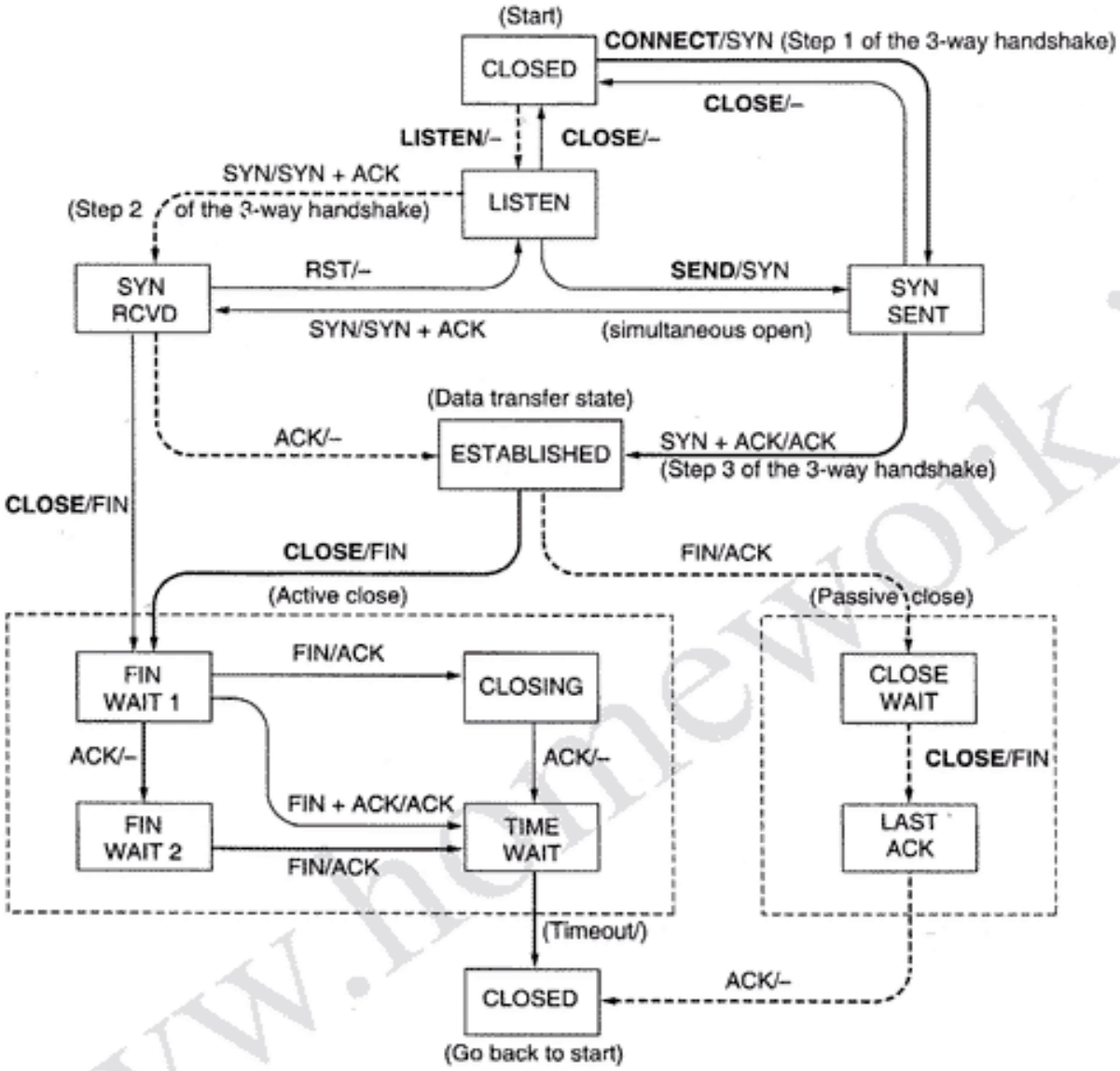
وقتی کار برنامهٔ کاربردی به پایان رسید، تابع اولیهٔ CLOSE را اجرا می‌کند. این کار موجب خواهد شد که «واحد انتقال TCP» یک قطعهٔ کنترلی FIN فرستاده و منتظر ACK مربوطه بماند. (مستطیل نقطه چین با عنوان active close نشان داده شده است.) وقتی ACK مربوطه دریافت شود، وضعیت اتصال به حالت FIN WAIT2 تغییر کرده و یک جهت از اتصال بسته خواهد شد. وقتی طرف مقابل نیز اتصال را ببندد، یک بستهٔ FIN از راه می‌رسد و وصول آن تصدیق می‌شود. حال اگرچه هر دو طرف اتصال را بسته‌اند ولیکن TCP بایستی به اندازهٔ دو برابر طول عمر حداکثر بسته‌ها صبر کند تا تمام بسته‌هایی که احتمالاً از اتصال قبلی در شبکه سرگردان هستند از بین بروند. به محض آن که مهلت چنین تایمری به پایان رسید، TCP رکورد اتصال مربوطه را حذف خواهد کرد.

حال مدیریت اتصال را از دیدگاه سرویس‌دهنده بررسی می‌نماییم: سرویس‌دهنده با انجام عمل LISTEN منتظر می‌ماند تا کسی تماس بگیرد. وقتی یک SYN وارد می‌شود، دریافت آن بلافاصله تصدیق شده و سرویس‌دهنده به وضعیت SYN RCVD وارد می‌شود. وقتی SYN-ACK ارسالی سرویس‌دهنده توسط مشتری اعلام وصول شد، فرآیند دست تکانی سه‌مرحله‌ای تکمیل می‌گردد و سرویس‌دهنده به وضعیت ESTABLISHED می‌رود. انتقال داده‌ها اکنون می‌تواند شروع شود.

وقتی کار مشتری به انجام رسید، اقدام به صدور فرمان CLOSE می‌کند. این کار موجب می‌شود که سرویس‌دهنده، FIN دریافت کند. (مستطیل نقطه چین، خاتمهٔ غیرفعال[1] را نشان می‌دهد.) در این لحظه به سرویس‌دهنده سیگنال داده می‌شود. هر گاه او نیز فرمان CLOSE را صادر کند، یک قطعهٔ حاوی FIN برای مشتری ارسال می‌شود. وقتی مشتری دریافت FIN را تأیید کرد، سرویس‌دهنده به اتصال مربوطه خاتمه داده و رکورد متناظر با آن اتصال را از جدول خود حذف می‌نماید.

---

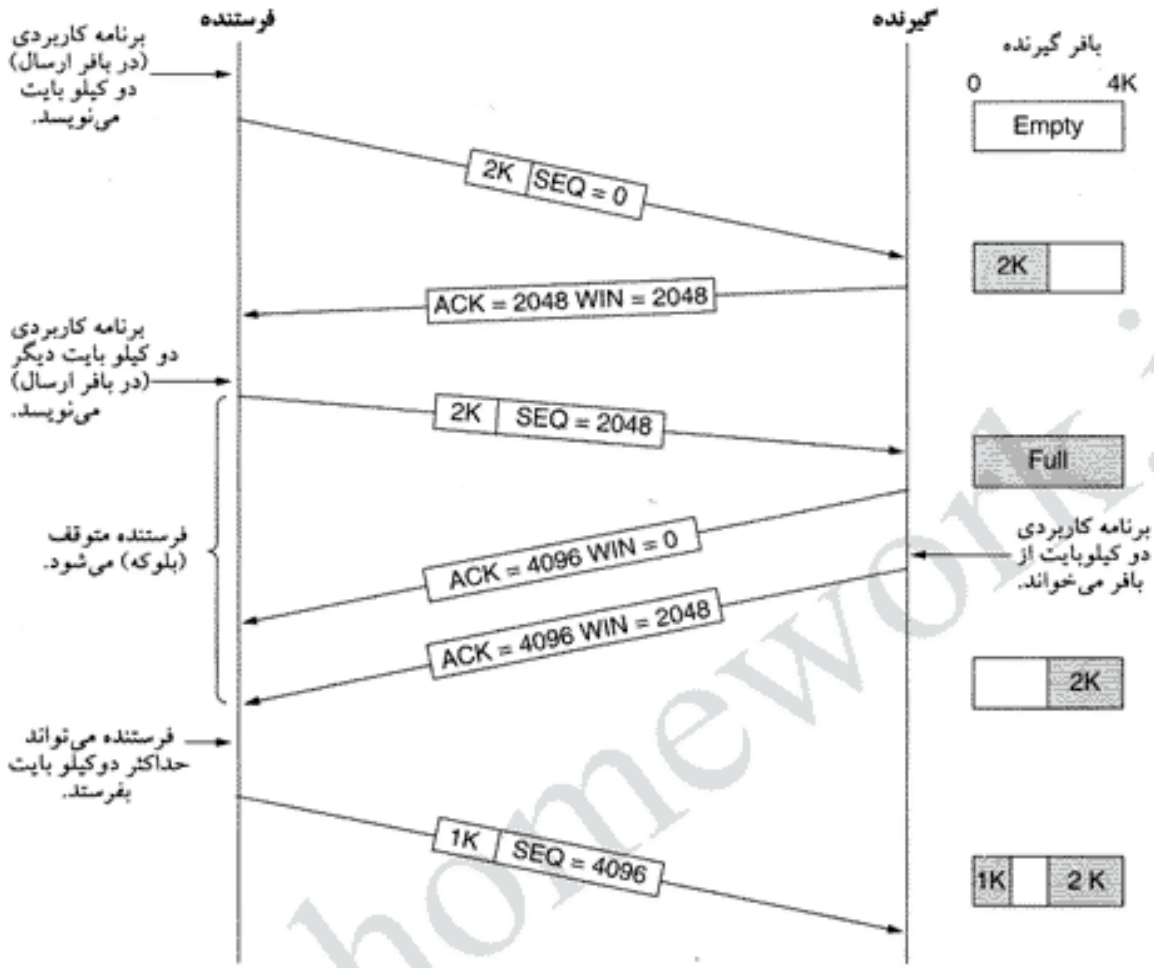
۱. Passive Close



شکل ۶-۳۳. «ماشین حالت محدود» برای مدیریت اتصال TCP. خطوط ضخیم توپر مسیر طبیعی عملکرد مشتری را نشان می‌دهد. خطوط ضخیم نقطه‌چین مسیر طبیعی عملکرد سرویس دهنده را نشان می‌دهد. خطوط نازک توپر بروز رخدادهای نامعمول را نشان می‌دهد. بر روی هر «گذار» (Transition) برچسبی وجود دارد که «رخداد» و «کنش» مربوطه را مشخص می‌کند.

۶-۵-۸ سیاستهای انتقال در TCP

همانگونه که قبلاً اشاره شد، مدیریت پنجره در TCP، وابستگی مستقیمی به تصدیق دریافت داده‌ها (Ack) ندارد (در حالی که در اغلب پروتکلهای لایهٔ پیوند داده اینگونه است). به عنوان مثال در شکل ۶-۳۴ فرض کنید که گیرنده ۴۰۹۶ بایت فضای بافر در اختیار دارد. اگر فرستنده یک قطعهٔ ۲۰۴۸ بایتی ارسال کند و به درستی و بدون خطا دریافت شود، گیرنده بلافاصله وصول آنرا تأیید خواهد کرد. با این وجود، از آنجایی که پس از دریافت این قطعه، فقط ۲۰۴۸ بایت از فضای بافر خالی است (مگر آن که برنامهٔ کاربردی بخشی از داده‌ها را از بافر بردارد) فلذا به طرف مقابل خود اعلام می‌دارد که از بایت بعدی (که شمارهٔ آن در فیلد Ack.No. مشخص شده) فقط حق ارسال ۲۰۴۸ بایت داده دارد. یا بعبارت فنی اعلام می‌کند، پنجرهٔ ۲۰۴۸ بایتی از شمارهٔ بایتی که در فیلد ۳۲ بیتی



شکل ۶-۳۴. مدیریت پنجره در TCP.

Ack. No. مشخص شده، آغاز می‌گردد.

حال فرستنده، ۲۰۴۸ بایت دادهٔ دیگر ارسال می‌نماید و دریافت آنها نیز تأیید می‌شود ولیکن، اندازهٔ پنجره، صفر اعلام می‌شود. فرستنده باید متوقف شود و آنقدر منتظر بماند تا پروسهٔ کاربردی گیرندهٔ داده‌ها، مقداری داده از بافر بردارد و TCP بتواند پنجرهٔ بزرگتری را اعلام کند.

وقتی اندازه پنجره صفر اعلام شده، فرستنده عموماً نمی‌تواند قطعهٔ دیگری ارسال کند مگر در دو مورد استثنا: اول «داده‌های اضطراری» (Urgent Data)؛ برای آنکه به کاربر اجازه بدهیم مثلاً پروسهٔ کاربردی اجرا شده بر روی ماشین راه دور را از بین ببرد.[1] دوم آن که فرستنده ممکن است قطعه‌ای حاوی یک بایت داده ارسال کند تا گیرنده وادار شود اندازهٔ پنجرهٔ خود و شمارهٔ ترتیب بایتی را که از آن به بعد منتظر دریافت است، از نو اعلام نماید. استاندارد TCP، از این گزینه برای اجتناب از بروز بن‌بست بهره گرفته است تا در صورت عدم اعلام طول پنجره، پروسهٔ متوقف شده در سمت گیرنده، تا ابد منتظر نماند.

فرستنده ملزم نیست که به محض تحویل گرفتن داده‌ها از برنامهٔ کاربردی، فوراً آنها را ارسال نماید. گیرندهٔ داده‌ها نیز مجبور نیست به محض دریافت و در اسرع زمان، دریافت آنها را تصدیق نماید. به عنوان مثال در شکل ۶-۳۴ وقتی اولین دو کیلوبایت از برنامهٔ کاربردی دریافت شد، TCP با اطلاع از آن که ۴ کیلوبایت فضای بافر در

---

۱. به اصطلاح یونیکس killکند.

اختیار دارد، می‌تواند ارسال آن را به تأخیر بیندازد تا ۲کیلوبایت بعدی نیز دریافت شود و هر چهار کیلوبایت را یکجا ارسال نماید. این آزادی عمل می‌تواند به افزایش کارآیی TCP کمک کند.

حال یک اتصال Telnet را مدّ نظر قرار بدهید که با فشار هر کلید در «ویرایشگر محاوره‌ای» (Interactive Editor)، از راه دور باید واکنش نشان بدهد. در بدترین حالت، وقتی که یک کاراکتر تکی جهت ارسال به «واحد انتقال TCP» تحویل می‌شود، TCP یک قطعهٔ ۲۱ بایتی برای آن تشکیل داده و آن را به IP می‌دهد. IP نیز آن را به صورت یک دیتاگرام ۴۱ بایتی ارسال می‌نماید. (۲۰ بایت سرآیند TCP + ۲۰ بایت سرایند IP + ۱ بایت کاراکتر ارسالی). در سمت گیرنده، TCP بلافاصله وصول آن را با ارسال یک دیتاگرام ۴۰ بایتی تصدیق می‌کند. بعداً وقتی برنامه ویرایشگر این بایت را می‌خواند، TCP بار دیگر مقدار جدید پنجرهٔ خود را که فقط یک واحد افزایش داشته، اعلام می‌دارد. این بسته نیز ۴۰ بایتی است. در آخر نیز وقتی برنامهٔ ویرایشگر، کاراکتر ارسالی را پردازش کرد، آن را در قالب یک بستهٔ ۴۱، بازگشت می‌دهد. (یا به عبارتی آن را Echo می‌کند.) بدین نحو، به ازای هر کاراکتر تایپ شده، ۱۶۴ بایت از پهنای باند مصرف و جمعاً چهار قطعهٔ TCP مبادله می‌شود. وقتی پهنای باند ارزشمند و محدود باشد این روش ، بهیچوجه کارآمد نیست.

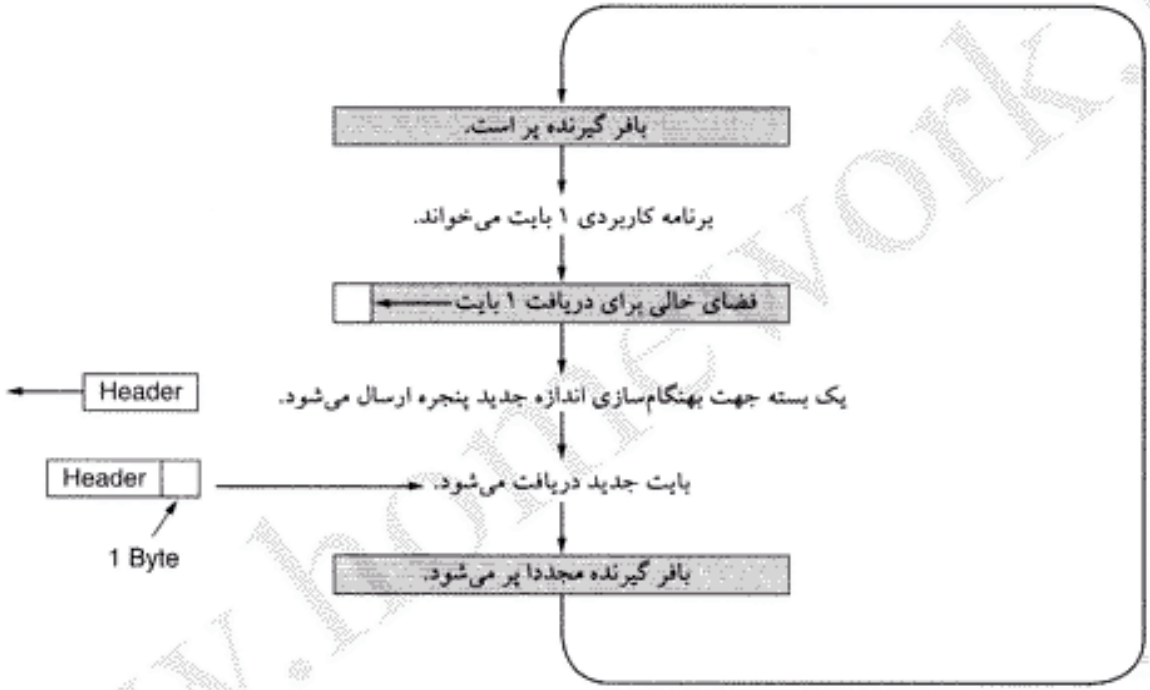
یک راهکار که در بسیاری از پیاده‌سازیهای عملی TCP برای بهینه سازی این وضعیت مورد استفاده قرار گرفته، آنست که اعلام وصول داده‌ها و اعلام اندازهٔ پنجره، به مدت ۵۰۰ میلی‌ثانیه به تعویق بیفتد، بدین امید که داده‌هایی جهت ارسال تولید و در بسته‌ای که باید ارسال می‌شد به رایگان حمل شوند. با فرض آنکه برنامه ویرایشگر در مثال بالا، هر کاراکتر را پس از ۵۰۰میلی‌ثانیه بازگشت بدهد (Echo کند)، فقط به ارسال یک بستهٔ ۴۱ بایتی برای کاربر نیاز است و تعداد بسته‌ها و میزان مصرف پهنای باند به نصف کاهش می‌یابد.

اگرچه این قاعده بار تحمیل شده توسط گیرنده بر روی شبکه را کاهش می‌دهد ولیکن هنوز هم فرستنده بسیار ناکارآمد عمل می‌کند چرا که برای ارسال یک بایت، ۴۱ بایت منتقل می‌شود. یک روش برای کاهش مصرف پهنای باند، «**الگوریتم ناگل**» (Nagle's Algorithm) نام دارد. (Nagle, 1984) آنچه که ناگل پیشنهاد کرد ساده بود: وقتی داده‌ها به صورت تک بایتی جهت ارسال تحویل فرستنده می‌شوند، فرستنده اولین بایت را ارسال می‌نماید بقیهٔ آنها را تا وقتی که دریافت همین یک بایت تأیید شود، بافر می‌کند. پس از اعلام وصول اولین بایت، فرستنده تمام کاراکترهای بافر شده را در قالب یک قطعهٔ TCP به یکباره ارسال می‌کند و مجدداً تا اعلام وصول آن، به بافر کردن کاراکترها ادامه می‌دهد. با این روش، اگر سرعت تایپ کاربر زیاد و شبکه بسیار کُند باشد، تعداد قابل توجهی کاراکتر در قالب یک قطعهٔ TCP ارسال می‌شود و پهنای باند مورد نیاز، کاهشی چشمگیر خواهد داشت. مضاف بر این، «الگوریتم ناگل» اجازه می‌دهد که بستهٔ جدید فقط زمانی ارسال شود که داده‌ها نیمی از فضای پنجره را پر کرده باشند و یا از طول مجاز یک قطعهٔ TCP بیشتر شده باشد.

«الگوریتم ناگل» بطور گسترده‌ای در پیاده‌سازیهای عملی TCP بکار گرفته شده است ولیکن در برخی از مواقع، غیرفعال کردن آن مفیدتر است. خصوصاً وقتی که از برنامهٔ کاربردی X Window در اینترنت استفاده می‌شود، حرکت ماوس باید به کامپیوتر راه دور ارسال شود. (سیستم X Window یک سیستم گرافیکی مبتنی بر پنجره است که در اغلب سیستمهای یونیکس از آن استفاده می‌شود. کاربران راه دور می‌توانند از آن جهت ورود از راه دور به سیستم یونیکس و تعامل با آن بهره بگیرند.) استفاده از الگورتیم ناگل موجب می‌شود که TCP ارسال حرکات ماوس را جمع‌آوری کرده تا آنها را به یکباره ارسال نماید و این کار حرکت ماوس را غیرطبیعی جلوه می‌دهد و اسباب ناخشنودی کاربران را فراهم می‌آورد.

یکی دیگر از مشکلاتی که می‌تواند کارآیی TCP را کاهش بدهد، «**سندرم پنجرهٔ ناموزون**» نام دارد. (Silly Window Syndrome, Clark, 1982) این مسئله زمانی رخ می‌دهد که داده‌ها در قالب بلوکهای بزرگ به

واحد انتقال TCP تحویل شود ولیکن برنامهٔ کاربردی گیرنده در طرف مقابل ، داده‌ها را به صورت بایت به بایت بخواند! برای درک بهتر این مشکل به شکل ۶-۳۵ نگاه کنید. در ابتدا بافر TCP در سمت گیرنده پر است و فرستنده از این موضوع اطلاع دارد. (به عبارت دیگر اندازهٔ پنجره صفر اعلام شده است.) سپس برنامهٔ کاربردی از استریم TCP (یا به عبارتی از بافر)، یک کاراکتر می‌خواند. این کار موجب می‌شود که TCP با ارسال بسته‌ای مقدار جدید پنجرهٔ خود را به اطلاع فرستنده برساند و به او تفهیم کند که اجازهٔ ارسال فقط یک بایت دارد! فرستنده اطاعت کرده و یک بایت ارسال می‌کند. حال بافر مجدداً پر می‌شود و ضمن تصدیق وصول این یک بایت، اندازهٔ پنجره صفر اعلام می‌شود. این رفتار می‌تواند تا بی‌نهایت ادامه داشته باشد.



شکل ۶-۳۵. سندروم پنجرهٔ ناموزون (Silly Window Syndrome).

راه حل پیشنهادی «کلارک» (Clark) آن بود که گیرنده برای یک بایت، مقدار جدید اندازهٔ پنجرهٔ خود را اعلام ننماید؛ در عوض باید آنقدر منتظر شود تا فضای موجود در بافر به حد متناسبی برسد، آنگاه این مقدار اعلام گردد. به ویژه، گیرنده نباید مقدار جدید اندازهٔ پنجرهٔ خود را اعلام کند مگر آن که قادر باشد یک قطعه داده را با طولی که در ابتدای برقراری اتصال به پذیرش آن متعهد شده، در بافر خود بپذیرد یا آن که حداقل نیمی از بافرش خالی شده باشد.

مضاف بر این، فرستنده می‌تواند با نفرستادن قطعات کوچک، به کاهش مشکل کمک کند: فرستنده باید سعی کند که قبل از ارسال یک قطعه آنقدر منتظر بماند تا داده‌های کافی متناسب با اندازهٔ یک قطعهٔ کامل یا معادل با نصف بافر گیرنده جمع‌آوری شود. (اندازهٔ بافر گیرنده را می‌توان براساس اعلامهای متوالی اندازهٔ پنجره، تخمین زد.)

الگوریتم ناگل و راه حل کلارک برای درمان «سندرم پنجرهٔ ناموزون»، مکمل یکدیگر هستند. ناگل سعی می‌کرد مشکلی را حل کند که در اثر تحویل بایت به بایت داده‌ها توسط برنامهٔ کاربردی به TCP پدید می‌آید. کلارک نیز سعی می‌کرد عکس این مشکل را یعنی وقتی که برنامهٔ کاربردی داده‌های خود را از TCP بایت به بایت

تحویل می‌گیرد، حل و فصل نماید. هر دوی این راهکارها معتبرند و در کنار هم کار می‌کنند. هدف آن است که فرستنده، قطعات کوچک نفرستد و گیرنده نیز قطعات را در اندازهٔ کوچک تحویل نگیرد.

در سمت گیرنده، TCP می‌تواند به غیر از بکارگیری روش کلارک، به گونهٔ دیگری نیز برای بهبود کارآیی اقدام نماید. همانند فرستنده، گیرنده نیز می‌تواند داده‌ها را بافر نماید؛ یعنی درخواست برنامهٔ کاربردی جهت خواندن داده‌ها را آنقدر معلق نگاه دارد تا آن که یک تودهٔ بزرگ داده جمع‌آوری شود. انجام این کار تعداد فراخوانیهای TCP را کاهش داده و طبعاً سربار سیستم کمتر می‌شود. البته این کار زمان تایمر و تأخیر اجرای فرمان READ را افزایش خواهد داد ولیکن برای کاربردهایی نظیر انتقال فایل، کارآیی بیشتر ارجح‌تر از زمان پاسخ سریع است.

مورد دیگری که گیرنده با آن روبروست، دریافت قطعات دادهٔ نامرتب است. گیرنده می‌تواند بر حسب شرایط بسته‌هایی را که خارج از ترتیب می‌رسند، نگاه دارد یا آنها را دور بریزد. البته وصول داده‌ها را می‌توان فقط زمانی اعلام کرد که همهٔ داده‌ها تا شماره‌ای که اعلام می‌شود دریافت شده باشد. اگر گیرنده قطعات ۰، ۱، ۲، ۴، ۵، ۶ و ۷ را (به استثنای ۳) دریافت کند تنها می‌تواند دریافت داده‌ها تا آخرین بایت قطعهٔ شمارهٔ ۲ را تصدیق نماید. وقتی مهلت فرستنده منقضی می‌شود، تمام قطعات از شمارهٔ ۳ به بعد از نو ارسال می‌شوند. اگر گیرنده، قطعات ۴ تا ۷ را بافر کرده باشد به محض دریافت قطعهٔ ۳ می‌تواند دریافت تمام بایتها را تا آخر قطعهٔ هفتم، تأیید کند.

۹-۵-۶ کنترل ازدحام در TCP

هر گاه بار تحویل شده به شبکه بیش از ظرفیتی باشد که می‌تواند از عهدهٔ آن برآید، ازدحام پدید خواهد آمد. اینترنت نیز از این مشکل مستثنی نیست. در این بخش به تشریح الگوریتمهایی خواهیم پرداخت که در طول ربع قرن گذشته برای حل و فصل مشکل ازدحام توسعه یافته‌اند. اگرچه لایهٔ شبکه نیز در مدیریت ازدحام می‌کوشد ولی بار سنگین این مسئولیت بیشتر بر عهدهٔ TCP می‌باشد چرا که راه حل واقعیِ رفع ازدحام، کاهش نرخ ارسال داده‌ها در این لایه است.

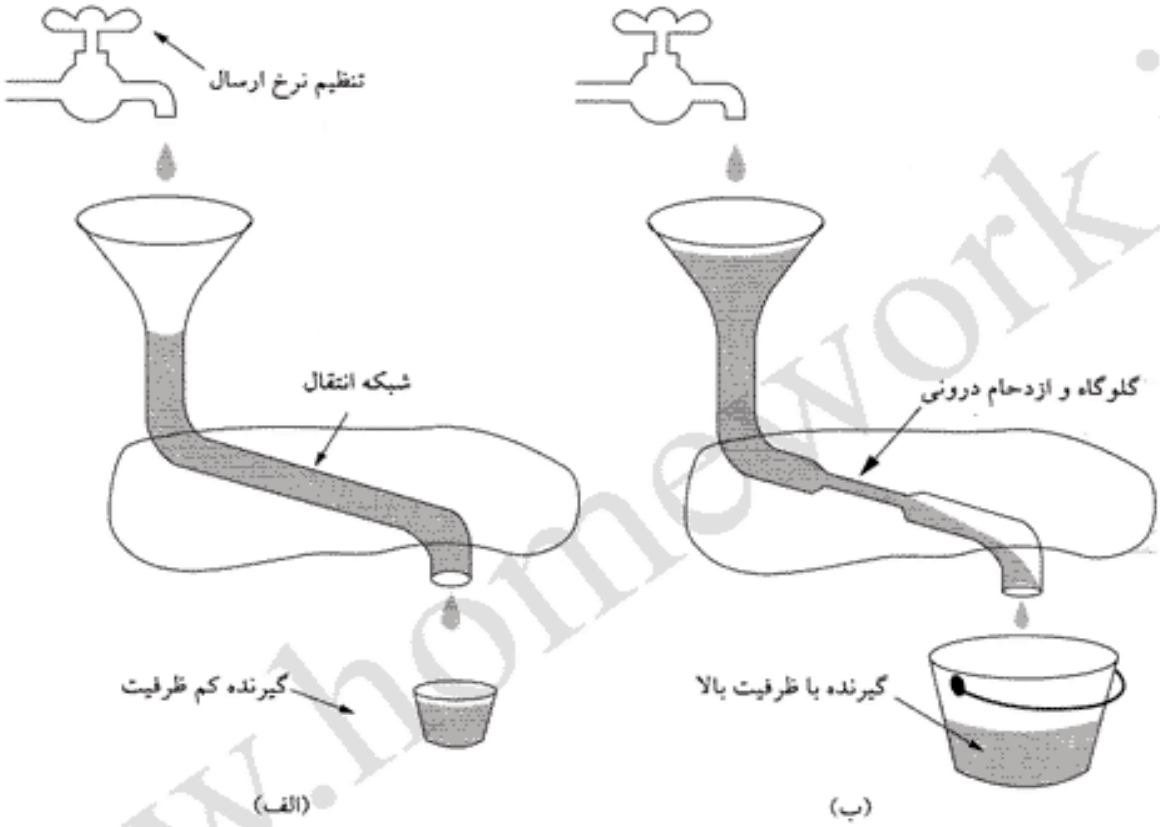
از دیدگاه تئوری، با بکارگیری یک اصل از دانش فیزیک می‌توان به مدیریت ازدحام پرداخت: «**اصل بقای بسته‌ها**»! ایدهٔ اصلی مبنی بر آن است که وقتی بستهٔ قبلی از شبکه خارج شد (یعنی به مقصد تحویل گردید)، بستهٔ جدیدی به شبکه تزریق شود. TCP سعی می‌کند با تنظیم پویا و خودکار اندازهٔ پنجره، بدین هدف نایل آید.

اولین گام در مدیریت ازدحام، تشخیص آن است. در روزگاران گذشته، تشخیص ازدحام بسیار دشوار بود. در آن دوران، به دو دلیل بسته‌ای از دست می‌رفت و مهلت فرستنده منقضی می‌شد: (۱) نویز روی خطوط انتقال (که می‌توانست بسته‌ها را نابود کند) (۲) حذف بسته‌ها توسط مسیریاب دچار ازدحام. تشخیص آنکه کدامیک از عوامل فوق منجر به از دست رفتن یک بسته شده، چندان ساده نبود.

امروزه، از بین رفتن بسته‌ها در اثر خطای خطوط انتقال پدیده‌ای بسیار نادر است زیرا اغلب شاه‌راههای ارتباطی از جنس فیبرنوری هستند. (هر چند شبکه‌های بی‌سیم داستان دیگری دارد.) طبعاً می‌توان نتیجه گرفت که انقضای مهلت ارسال در شبکهٔ اینترنت ناشی از ازدحام است. تمام الگوریتمهای TCP در اینترنت فرض را بر آن گذاشته‌اند که انقضای مهلت (Timeout) و عدم وصول بسته به دلیل بروز ازدحام بوده است و به همین دلیل TCP بر زمانهای Timeout (یعنی زمان انقضای مهلت اعلام وصول هر بسته) به دقت نظارت می‌کند.

قبل از تشریح واکنش TCP در برخورد با ازدحام، ابتدا بررسی می‌کنیم که این پروتکل چه تلاشی در پیشگیری از بروز آن می‌کند. پس از برقراری یک اتصال، بایستی اندازهٔ مناسبی برای پنجره انتخاب شود. گیرنده می‌تواند اندازهٔ پنجره را برحسب بافر در اختیار خود، تعیین نماید. اگر فرستنده به اندازهٔ پنجرهٔ گیرنده پایبند باشد مشکلی از بابت سرریز شدن بافر طرف گیرنده پیش نخواهد آمد ولیکن هنوز هم احتمال بروز مشکلات ناشی از ازدحام درون یک شبکه، وجود دارد.

در شکل ۶-۳۶، این مشکل را با تعبیر هیدرولیکی آن به تصویر کشیده‌ایم. در شکل ۶-۳۶-الف ، یک لولهٔ قطور را مشاهده می‌کنیم که به یک گیرندهٔ کم حجم منتهی شده است. مادامی که فرستنده بیش از ظرفیت سطل، آب تولید و ارسال نکند، سطل هم سرریز نخواهد شد. در شکل ۶-۳۶-ب، عامل محدودکننده، ظرفیت سطل نیست بلکه ظرفیت داخلی حمل زیرشبکه، ایجاد محدودیت کرده است. اگر حجم زیاد آب با سرعت بالا به قیف وارد شود، آن را سریعاً پر کرده و آب به هدر خواهد رفت (در این حالت به دلیل سرریزی قیف).



شکل ۶-۳۶. (الف) یک شبکه سریع که یک گیرنده با ظرفیت کم را تغذیه می‌کند. (ب) یک شبکه کُند که یک گیرنده با ظرفیت بالا را تغذیه می‌کند.

راه حل اینترنت آن است که عوامل بروز این دو مشکل بالقوه را پذیرفته و با هر یک از آنها بطور مجزا و مستقل برخورد کنیم.[1] برای این کار هر فرستنده دو پنجره ایجاد می‌نماید: پنجره اول که براساس اعلام گیرندهٔ طرف مقابل ایجاد می‌شود و پنجرهٔ دوم، «**پنجرهٔ ازدحام**» (Congestion Window). هر یک از این پنجره‌ها تعداد بایتهایی را مشخص می‌کنند که فرستنده می‌تواند ارسال کند. تعداد بایتهایی که فرستنده مجاز به ارسال آنهاست ، مینیمم اندازهٔ این دو پنجره است. بنابراین اندازهٔ مؤثر پنجره، مقدار مینیمم آنچیزی است که فرستنده فکر می‌کند صحیح است و آنچه که گیرنده فکر می‌کند آن باید باشد! مثلاً اگر گیرنده عنوان کند که «هشت کیلوبایت بفرست» ولی فرستنده بداند که ارسال بیش از چهار کیلوبایت شبکه را دچار انسداد می‌کند، چهار کیلوبایت ارسال خواهد کرد. برعکس اگر گیرنده اعلام کند که «هشت کیلوبایت بفرست» و فرستنده نیز بداند که ارسال تا ۳۴کیلوبایت بلامانع است، فقط هشت کیلوبایت درخواستی را ارسال خواهد کرد.

---

۱. تفکیک عامل «ظرفیت شبکه» از «ظرفیت گیرنده»

وقتی اتصال برقرار می‌شود، فرستنده، اندازهٔ «پنجرهٔ ازدحام» را با طول حداکثر هر قطعه که در حین اتصال توافق شده، مقداردهی اولیه می‌کند؛ سپس یک قطعه با طول حداکثر می‌فرستد. اگر اعلام وصول این قطعه قبل از انقضای مهلت مقرر دریافت شد، اندازهٔ پنجرهٔ ازدحام را به اندازهٔ طول حداکثر قطعه اضافه می‌کند و دفعهٔ بعدی معادل دو قطعه ارسال می‌نماید. مادامی که پس از ارسال هر قطعه دریافت آن تصدیق می‌شود، به اندازهٔ «پنجرهٔ ازدحام» معادل با طول حداکثر هر قطعه اضافه خواهد شد. وقتی اندازهٔ پنجرهٔ ازدحام معادل با طول n قطعه باشد و تمام قطعات ارسالی سر موعد اعلام وصول شوند به پنجرهٔ ازدحام معادل با طول کل n قطعه (برحسب بایت) اضافه خواهد شد. کوتاه سخن آن که اگر در هر بار ارسال (معادل با n قطعه) تمام آنها اعلام وصول شوند، طول پنجرهٔ ازدحام دو برابر می‌شود.

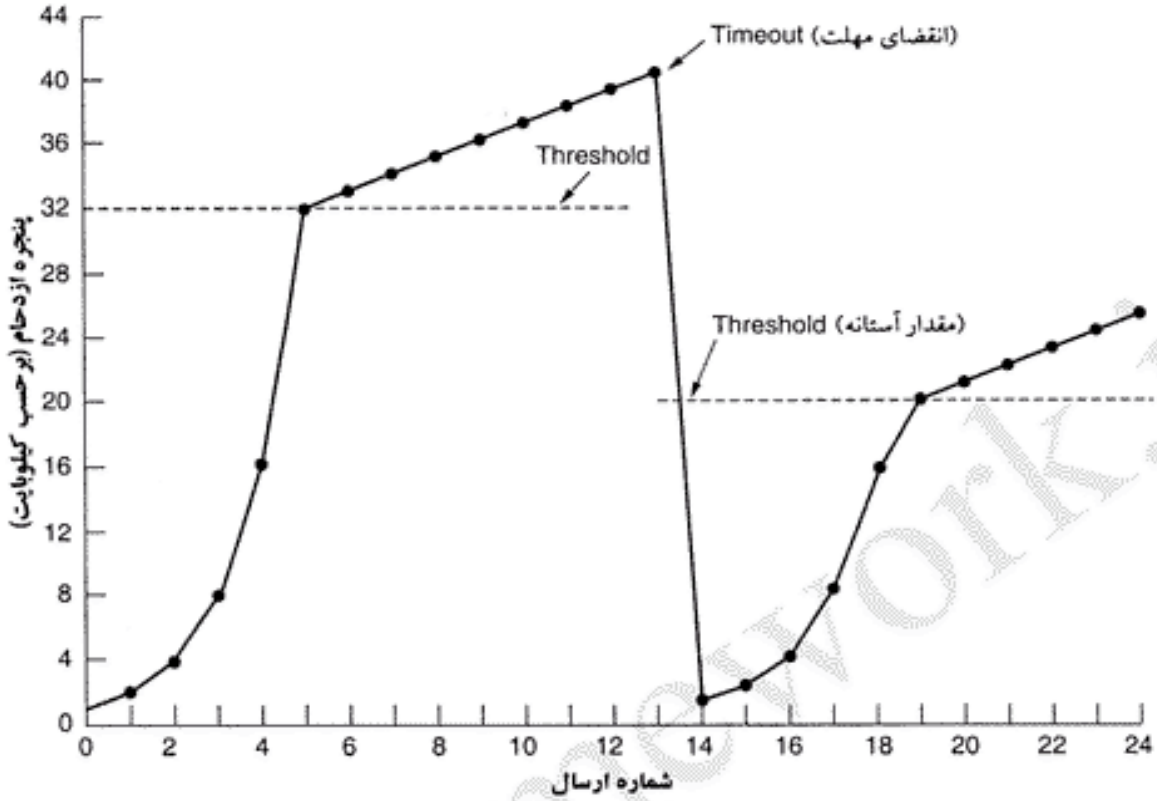
اندازهٔ «پنجرهٔ ازدحام» آنقدر به صورت نمایی رشد می‌کند تا آنکه پس از ارسال قطعات، یا وصول آنها تصدیق نشود و یا آنکه به اندازهٔ پنجرهٔ گیرنده برسد. به عنوان مثال اگر اندازهٔ پنجرهٔ ۱۰۲۴ بایتی، ۲۰۴۸ بایتی، ۴۰۹۶ بایتی به درستی عمل کند ولی ارسال چندین قطعه معادل با ۸۰۹۶ بایت، با مشکل انقضای مهلت اعلام وصول (Timeout) مواجه شود برای اجتناب از ازدحام، اندازهٔ پنجره به ۴۰۹۶ بایت تنظیم می‌شود. مادامی که پنجرهٔ ازدحام بر روی ۴۰۹۶ ثابت می‌ماند حتی اگر طول پنجرهٔ اعلام شده توسط گیرنده از ۴۰۹۶ بیشتر باشد، فرستنده هیچگاه قطعاتی را که مجموع طول آنها از ۴۰۹۶ بیشتر می‌شود نخواهد فرستاد. این الگوریتم اصطلاحاً «**شروع آهسته**» (Slow Start) نام گرفته ولیکن هرگز کُند عمل نمی‌کند زیرا روند آن نمایی است. (Jocobson, 1988) تمام پیاده‌سازیهای TCP، ملزم به حمایت از این الگوریتم هستند.

حال اجازه بدهید به الگوریتم کنترل ازدحام در اینترنت بپردازیم: در اینترنت به غیر از پنجرهٔ گیرنده و پنجرهٔ ازدحام، از پارامتر سومی به نام «**آستانه**» (Threshold) استفاده می‌شود. مقدار اولیهٔ این پارامتر 64KByte است. وقتی پس از ارسال قطعاتی، مهلت اعلام وصول آنها منقضی شود، پارامتر «**آستانه**» به نصف مقدار پنجرهٔ ازدحام تنظیم شده و مقدار پنجرهٔ ازدحام مجدداً به مقدار طول حداکثر یک قطعه بر می‌گردد. حال مجدداً الگوریتم «شروع آهسته» (Slow Start) برای تعیین اندازهٔ مناسب طول پنجرهٔ ازدحام شروع به کار می‌نماید، با این تفاوت که رشد نمایی به محض رسیدن به مقدار آستانه، متوقف می‌شود. پس از رسیدن به نقطهٔ آستانه، ارسال موفق یک قطعه اندازهٔ پنجره را دو برابر نمی‌کند بلکه آن را به صورت خطی افزایش می‌دهد. طبعاً این الگوریتم حدس می‌زند که کاهش طول پنجره به نصف معقول است، فلذا از این نقطه شروع کرده و اندازهٔ آنرا به تدریج افزایش می‌دهد.

برای آن که مشخص شود این الگوریتم کنترل ازدحام چگونه کار می‌کند به شکل ۶-۳۷ نگاه کنید. در اینجا اندازهٔ حداکثر هر قطعه، ۱۰۲۴ بایت است. در ابتدا، اندازهٔ «پنجرهٔ ازدحام»، ۶۴کیلوبایت بوده ولی به دلیل انقضای مهلت اعلام وصول قطعه‌ها (Timeout)، مقدار آستانه به ۳۲کیلوبایت تنظیم شده و اندازهٔ پنجرهٔ ازدحام برای ارسال شمارهٔ صفر به مقدار ۱۰۲۴ کاهش یافته است. از آن به بعد پنجرهٔ ازدحام بطور نمایی رشد کرده تا به مقدار آستانه (یعنی 32KB) رسیده است. پس از آن رشد اندازهٔ پنجرهٔ ازدحام به صورت خطی بوده است.

ارسال سیزدهم موفق نبوده است و مجدداً مشکل عدم اعلام وصول داده‌ها در مهلت مقرر، بروز کرده است. (بروز مشکل در این لحظه چندان دور از ذهن نیست.) لذا مجدداً مقدار آستانه به نصف اندازهٔ پنجرهٔ فعلی تنظیم شده است. (در اینجا اندازهٔ پنجره ۴۰کیلوبایت بوده فلذا نصف آن، ۲۰کیلوبایت می‌شود.) بار دیگر الگوریتم «شروع آهسته» کار خود را آغاز می‌کند. از آنجایی که در ارسال چهاردهم تا بیستم، پیغام ACK برگشته لذا در هر ارسال اندازهٔ پنجرهٔ ازدحام دو برابر و از آن به بعد رشد پنجره خطی شده است.

اگر هیچگاه مهلت اعلام وصول منقضی نشود، پنجرهٔ ازدحام، رشد خود را آنقدر ادامه می‌دهد تا به اندازهٔ پنجرهٔ گیرنده برسد. در این نقطه، رشد پنجره متوقف شده و مادامی که طول پنجرهٔ گیرنده تغییر نکند و مهلت اعلام



شکل ۶-۳۷. مثالی از الگوریتم کنترل ازدحام در اینترنت.

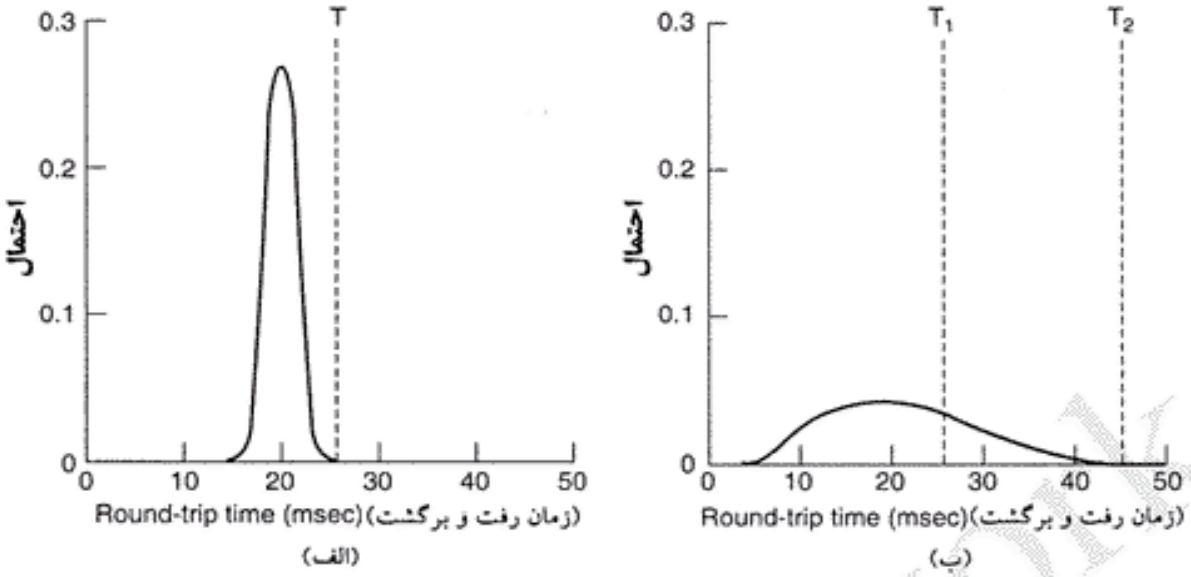
وصول قطعات ارسالی منقضی نشود، اندازهٔ پنجرهٔ ازدحام ثابت خواهد ماند. مضاف بر این اگر پیغام ICMP SOURCE QUENCH دریافت و تحویل TCP شود این رخداد نیز به مثابهٔ انقضای مهلت تلقی می‌شود. راهکار جدیدتر در RFC 3168 تشریح شده است.

۶-۵-۱۰ مدیریت تایمرها در TCP

TCP برای آن که بتواند کارش را بدرستی انجام بدهد از چندین تایمر بهره می‌گیرد. مهمترین آنها، «**تایمر ارسال مجدد**» (Retransmission Timer) است. وقتی قطعه‌ای ارسال می‌شود این تایمر شروع به کار می‌کند. اگر قبل از آن که مهلت این تایمر منقضی گردد، قطعهٔ ارسالی اعلام وصول شود (یعنی Ack آن برگردد)، تایمر متوقف می‌شود ولیکن اگر وصول داده‌ها قبل از صفر شدن مقدار این تایمر تأیید نشود، این قطعه از نو ارسال و تایمر مجدداً راه‌اندازی می‌گردد. حال سؤال مهمی پیش می‌آید: زمان انقضای مهلت چقدر باید باشد؟

این مسئله در لایهٔ انتقال از شبکهٔ اینترنت در مقایسه با همین مسئله در پروتکلهای لایهٔ پیوند داده (فصل سوم)، دشوارتر و پیچیده‌تر است. در لایهٔ پیوند داده، تأخیرها کاملاً قابل تخمین هستند (یا به عبارتی واریانس تأخیر پایین است) فلذا به نحوی که در شکل ۶-۳۸-الف مشاهده می‌شود می‌توان تایمر را به همان مقداری تنظیم کرد که انتظار می‌رود اعلام وصول بسته‌ها (یعنی Ack) قبل از این زمان برگردد. از آنجایی که در لایهٔ پیوند داده‌ها، پیغامهای Ack با تأخیر پیش‌بینی نشده مواجه نمی‌گردد (زیرا مشکل ازدحام وجود ندارد) فلذا عدم دریافت Ack، عموماً به معنای نابودی فریم ارسالی یا نابودی Ack بازگشتی تلقی می‌شود.

TCP با محیطی کاملاً متفاوت مواجه است. «تابع چگالی احتمال زمان بازگشت Ack» به جای آن که شبیه به شکل ۶-۳۸-الف باشد بیشتر شبیه به شکل ۶-۳۸-ب است. تعیین زمان رفت قطعهٔ داده و بازگشت Ack آن



شکل ۶-۳۸. (الف) چگالی احتمال زمان دریافت پیغامهای تصدیق وصول (ACK) در لایهٔ پیوند داده. (ب) چگالی احتمال زمان دریافت پیغامهای تصدیق وصول در TCP (لایهٔ انتقال).

(Round Trip Time) پیچیده است. حتی اگر این زمان مشخص باشد، تصمیم‌گیری در خصوص مهلت دریافت Ack (یعنی مقدار اولیه تایمر) بسیار دشوار است. اگر زمان انقضای مهلت (یا به عبارتی مقدار اولیه تایمر ارسال مجدد) کوتاه در نظر گرفته شود (مثلاً مقدار T_1 در شکل ۶-۳۸-ب) آنگاه برخی از قطعات داده، بیهوده ارسال مجدد شده و اینترنت با بسته‌های زائد شلوغ می‌شود. برعکس اگر این مقدار بسیار طولانی در نظر گرفته شود (مثلاً مقدار T_2) آنگاه وقتی بسته‌ای از بین می‌رود به دلیل تأخیر بسیار زیادی که در ارسال مجدد بسته پیش می‌آید، کارآیی بشدت افت می‌کند. مضاف بر این مقدار میانگین و واریانس زمان اعلام وصول بسته‌ها می‌تواند در عرض چند ثانیه تغییر جدی داشته باشد. (این تغییر می‌تواند در اثر بروز ازدحام یا رفع آن باشد.)

راه حل نهایی، استفاده از یک الگوریتم کاملاً پویا (Dynamic) است به گونه‌ای که مدام مهلت بازگشت Ack بسته‌ها را تخمین زده و تنظیم نماید. این کار براساس اندازه‌گیری دائمی کارآیی شبکه انجام می‌گیرد. الگوریتمی که عموماً در TCP بکار گرفته می‌شود توسط «ژاکوبسن» (۱۹۸۸) پیشنهاد شده و به نحو زیر کار می‌کند: TCP برای هر اتصال یک متغیر به نام RTT نگاه می‌دارد که مقدار آن بهترین تخمین از زمان رفت بسته به مقصد مورد نظر و بازگشت Ack آن می‌باشد. وقتی یک قطعهٔ داده ارسال می‌گردد یک تایمر شروع به کار می‌کند تا اولاً مشخص شود بازگشت Ack چقدر طول می‌کشد، ثانیاً در صورتی که مهلت مقرر منقضی شد آنرا از نو ارسال نماید. اگر قبل از انقضای مهلت مقرر، دریافت قطعه تصدیق شود، TCP مقدار زمان رفت قطعه و برگشت Ack آنرا، (بکمک مقدار فعلی تایمر RTT) اندازه‌گیری می‌کند. این مقدار را M بنامید. سپس براساس M، مقدار جدید RTT را طبق فرمول زیر بهنگام می‌نماید:

$$RTT = \alpha . RTT + (1 - \alpha).M$$

در فرمول فوق، α ضریب هموارسازی (Smoothing Factor) است و تعیین می‌کند که مقدار قبلی RTT با چه وزنی در محاسبهٔ مقدار جدید حضور دارد. α عموماً 7/8 است.

حتی با در اختیار داشتن مقدار RTT، انتخاب مقدار مناسب برای تایمر «ارسال مجدد» چندان ساده نیست. عموماً TCP مقدار تایمر را $\beta.RTT$ در نظر می‌گیرد ولیکن مسئله اصلی انتخاب β است. در پیاده‌سازیهای اولیهٔ

TCP، مقدار β همیشه ۲ در نظر گرفته می‌شد ولیکن تجربه نشان داد که انتخاب مقدار ثابت برای β مقدار مناسب و منعطفی نیست زیرا با افزایش واریانس، به درستی جواب نمی‌دهد.

در سال ۱۹۹۸ «ژاکوبسن» پیشنهاد کرد که β متناسب با «انحراف معیار تابع چگالی احتمال زمان دریافت پیغامهای Ack»[1] در نظر گرفته شود. بدین طریق اگر واریانس بالا باشد، β نیز زیاد خواهد بود (و بالعکس). همچنین او پیشنهاد کرد که برای بدست آوردن مقدار تخمینی انحراف معیار از «میانگین انحراف» استفاده شود. به همین دلیل در الگوریتم او به متغیر جدیدی به نام D نیاز است که مقدار تخمینی و هموارشدهٔ (Smoothed) «انحراف» را نگه می‌دارد: وقتی پیغام Ack یک قطعهٔ داده دریافت می‌شود، اختلاف بین مقدار مورد انتظار (یعنی RTT) و مقدار اندازه‌گیری شده (یعنی M) به صورت |RTT-M| محاسبه می‌شود. سپس مقدار هموار شدهٔ D طبق رابطه زیر محاسبه می‌شود:

$$D = \alpha.D+(1 - \alpha).|RTT-M|$$

مقدار α در فرمول بالا می‌تواند با مقدار α در فرمول RTT یکسان باشد یا فرق کند. اگرچه مقدار D دقیقاً معادل با انحراف معیار نیست ولیکن به آن نزدیک است و برای محاسبات ما کفایت می‌کند. ژاکوبسن نشان داد که می‌توان فرمول بالا را با جمع، تفریق و شیفت چند عدد صحیح، پیاده‌سازی کرد. اغلب پیاده‌سازیهای عملی TCP از همین الگوریتم بهره گرفته‌اند و زمان انقضای مهلت (یعنی مقدار تایمر ارسال مجدّد) را به صورت زیر تنظیم می‌کنند:

$$Timeout = RTT+4\times D$$

انتخاب ضریب ۴ اختیاری است ولیکن استفاده از این عدد دو مزیت دارد: اول آن که ضرب یک عدد صحیح در ۴ را می‌توان با یک عمل شیفت (دو بیت شیفت به چپ) انجام داد. دوم آن که از انقضای مهلت و ارسال مجدد و غیرضروری جلوگیری می‌کند زیرا کمتر از یک درصد از بسته‌ها با تأخیری بیش از چهار برابر انحراف معیار دریافت می‌شوند. [به عبارت دیگر احتمال آن که بسته‌ای بیهوده ارسال مجدد شود کمتر از یک درصد است.] (در واقع، پیشنهاد ژاکوبسن عدد ۲ بود ولی تجربه نشان داد که عدد ۴، کارآمدتر و بهینه‌تر است.)

مشکلی که در محاسبهٔ پویای RTT بروز می‌کند آن است که وقتی پس از ارسال یک قطعه، مهلت اعلام وصول آن منقضی و قطعه از نو ارسال شد چه باید کرد؟ وقتی که پس از ارسال مجدد یک قطعه، Ack آن دریافت شد روشن نیست که آیا این Ack مربوط به قطعه اول است که دیر رسیده، یا آنکه واقعاً مربوط به قطعهٔ دوم است. حدس اشتباه در این مورد می‌تواند، منجر به غلط شدن نتیجهٔ تخمین RTT شود. شخصی به نام Phil Karn به این مسئله پی برد. وی یک آماتور علاقمند به سیستمهای رادیویی است که می‌خواست بسته‌های TCP/IP را به کمک کانالهای رادیویی که بشدت نامطمئن و توام با خطا هستند ارسال نماید. (در شرایط خوب فقط نیمی از بسته‌ها سالم دریافت می‌شوند!) وی یک پیشنهاد ساده ارائه داد: برای بسته‌هایی که از نو ارسال می‌شوند، مقدار RTT بهنگام نگردد. در عوض مهلت دریافت پیغام Ack، به دو برابر افزایش یابد. این اصلاحیه به نام الگوریتم Karn مشهور است و در پیاده‌سازی TCP از آن استفاده می‌شود.

«تایمر ارسال مجدد» (Retransmission Timer)، تنها تایمری نیست که TCP از آن استفاده می‌کند. تایمر دومی نیز به نام Persistent Timer وجود دارد. از این تایمر برای پیشگیری از بن‌بست ذیل استفاده می‌شود: فرض کنید گیرنده با ارسال Ack اندازهٔ پنجرهٔ خود را به فرستنده صفر اعلام کرده و از او می‌خواهد که منتظر بماند. بعداً گیرنده اندازهٔ جدید پنجرهٔ خود را بهنگام‌سازی و اعلام می‌کند ولیکن بستهٔ حاوی مقدار جدید از بین می‌رود. حال گیرنده و فرستنده هر دو در انتظار یکدیگر به سر می‌برند. برای آن که این انتظار تا ابد ادامه نیابد، به محض

۱. Standard Deviation of Acknowledgement Arrival Time Probability Density Function

انقضای مهلت تایمر، فرستنده با ارسال بسته‌ای به گیرنده ، سعی می‌کند او را بیازماید. در پاسخ بدین بسته، اندازهٔ پنجره اعلام خواهد شد. اگر این مقدار کماکان صفر باشد، تایمر مجدداً تنظیم شده و این روند تکرار می‌شود ولی اگر این مقدار غیرصفر باشد فرستنده می‌تواند داده‌های خود را ارسال نماید.

تایمر سومی که در برخی از پیاده‌سازیهای عملی TCP از آن استفاده شده، Keepalive Timer نام دارد. وقتی اتصالی برای مدت زمان طولانی بیکار بماند و مهلت این تایمر منقضی شود، یکی از طرفین سعی می‌کند فعال بودن طرف مقابل خود را بررسی کند. اگر طرف مقابل پاسخی ندهد، اتصال قطع می‌شود. این ویژگی اندکی مناقشه‌برانگیز شده است زیرا اولاً سربار تحمیل می‌کند و ثانیاً ممکن است به دلیل وقفهٔ موقت در شبکه به یک اتصال سالم و صحیح خاتمه داده شود.

آخرین تایمر مورد استفاده در هر اتصال TCP، تایمریست که در وضعیت TIMEED WAIT (شکل ۶-۳۳) در حین بستن یک اتصال، مورد استفاده قرار می‌گیرد. پس از آنکه یک اتصال قطع می‌شود به کمک این تایمر به مدت دو برابر حداکثر طول عمر بسته‌ها، اجازهٔ ایجاد اتصالی با همین شماره پورت را نخواهد داد تا اطمینان حاصل شود که بسته‌های سرگردان (حاصل از اتصال قبل) کاملاً از بین رفته‌اند.

۶-۵-۱۱ TCP و UDP بی‌سیم

از دیدگاه تئوری، پروتکلهای لایهٔ انتقال باید مستقل از تکنولوژی بکار رفته در لایهٔ شبکه باشند. به ویژه، نباید برای TCP اهمیت داشته باشد که IP بر روی فیبرنوری کار می‌کند یا بر روی کانالهای رادیویی، ولیکن در عمل این موضوع اهمیت دارد زیرا پیاده‌سازی عملی TCP اغلب بر مبنای مفروضاتی بهینه‌سازی می‌شود که فقط برای شبکه‌های سیمی صادق هستند و برای شبکه‌های بی‌سیم با شکست مواجه می‌گردد. نادیده انگاشتن ویژگیهای انتقال بی‌سیم می‌تواند به نوعی از پیاده‌سازی TCP منتهی شود که از دیدگاه منطقی درست است ولی عملاً کارآیی بسیار پایینی دارد.

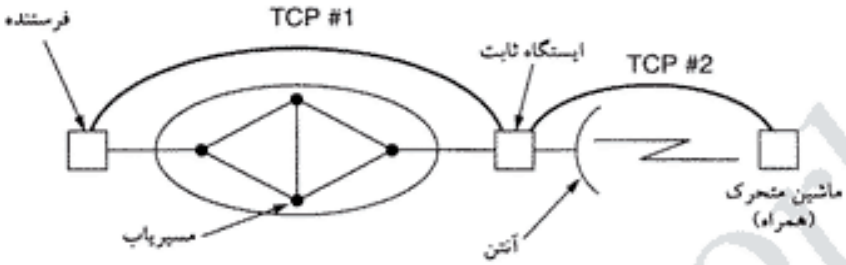
بنیادی‌ترین مشکل در الگوریتم کنترل ازدحام بروز می‌کند. امروزه تقریباً در تمام پیاده‌سازیهای عملی TCP فرض شده که انقضای مهلت (Timeouts) در اثر ازدحام بوده است و طبعاً خطای کانال انتقال را نادیده می‌گیرند. در نتیجه وقتی تایمر به صفر می‌رسد و مهلت دریافت Ack منقضی می‌شود، TCP سرعت خود را پایین آورده و با شدت کمتری ارسال می‌کند. ایده‌ای که در پشت این راهکار نهفته است کاهش بار شبکه و کم کردن ازدحام می‌باشد.

متأسفانه لینکهای ارتباطی بی‌سیم بسیار غیرقابل اعتماد و توأم با خطا هستند و همیشه برخی از بسته‌ها را از بین می‌برند. در اینجا راهکار مناسب در مواجهه با بسته‌های نابود شده آنست که در اسرع وقت از نو ارسال شوند. کاهش سرعت ارسال، وضع را به مراتب بدتر می‌کند. اگر مثلاً بیست درصد از کل بسته‌ها از بین بروند ، وقتی فرستنده ۱۰۰ بسته در ثانیه ارسال می‌کند درصد مفید آن ۸۰ بسته در ثانیه خواهد بود. در چنین حالتی کاهش نرخ ارسال بسته به ۵۰ بسته در ثانیه، ظرفیت مفید خروجی را به ۴۰ بسته در ثانیه کاهش خواهد داد.

در نتیجه وقتی بسته‌ای در یک شبکهٔ سیمی از دست می‌رود باید سرعت ارسال فرستنده کاهش یابد ولیکن وقتی همین اتفاق در یک شبکهٔ بی‌سیم می‌افتد نه تنها فرستنده نباید سرعت خود را پایین بیاورد بلکه باید تلاش بیشتری در ارسال سریع و مجدد آنها داشته باشد. وقتی که فرستنده از ماهیت شبکهٔ خود آگاه نیست تصمیم‌گیری درست دشوار است.

بسیار اتفاق می‌افتد که مسیر بین فرستنده و گیرنده ناهمگون است یعنی ممکن است ۱۰۰۰ کیلومتر از مسیر در شبکه‌ای سیمی و ۱ کیلومتر آخر بی‌سیم باشد. در چنین حالتی تصمیم‌گیری در خصوص مهلت دریافت Ack به مراتب دشوارتر است زیرا محل بروز مشکل اهمیت دارد. [اگر بسته‌ای در شبکهٔ سیمی از بین رفته باشد ناشی از

مشکل ازدحام و اگر در شبکهٔ بی‌سیم خراب شده باشد ناشی از خطای کانال بوده است.] برای حل این مشکل راه حلی توسط Bakne و Badrinath (۱۹۹۵) ارائه شده که «TCP غیرمستقیم» (indirect TCP) نام دارد. در این روش، یک اتصال TCP به نحوی که در شکل ۶-۳۹ دیده می‌شود به دو اتصال مجزا تقسیم می‌شود: اتصال اول بین فرستنده و ایستگاه ثابت (Base Station) برقرار می‌شود. اتصال دوم بین ایستگاه ثابت و گیرنده شکل می‌گیرد. ایستگاه ثابت به سادگی بسته‌ها را در هر دو جهت، بین این دو اتصال کپی می‌کند. [به عبارتی از اتصال اول دریافت و بر روی اتصال دوم ارسال می‌کند و بالعکس]



شکل ۶-۳۹. تقسیم یک اتصال TCP به دو اتصال مجزا.

مزیت این روش آن است که هر دو اتصال در شبکه‌ای همگن شکل می‌گیرند. انقضای مهلت در اتصال اول می‌تواند از سرعت ارسال بکاهد ولیکن بروز همین مشکل در اتصال دوم می‌تواند به افزایش سرعت فرستنده بینجامد. پارامترهای دیگر نیز براساس محیط هر یک از این دو اتصال به صورت مستقل و پویا تنظیم می‌شوند. اشکال این روش نیز آن است که مفهوم TCP را نقض می‌کند. از آنجایی که هر یک از بخشهای یک اتصال خودش یک اتصال کامل است فلذا وقتی دریافت داده‌ها به فرستنده اعلام می‌شود به معنای آن نیست که گیرندهٔ حقیقی آنها را دریافت کرده است بلکه به معنای دریافت آنها توسط ایستگاه ثابت می‌باشد.

راه حل دیگری نیز توسط Balakrishnan و همکاران او پیشنهاد شده که مفهوم TCP را نقض نمی‌کند. (۱۹۹۵) روش آنها مستلزم ایجاد تغییرات کوچکی در کُد اجرایی لایهٔ شبکه در ایستگاه ثابت است. یکی از تغییرات، افزودن یک «**عامل تحقیق و تفحص**» (Snooping Agent) به لایهٔ شبکه از ایستگاه ثابت می‌باشد؛ این عامل تمام قطعات TCP را که به سوی یک ماشین متحرک بی‌سیم می‌روند یا بسته‌های حاوی Ack بازگشتی از آنرا تشخیص داده و موقتاً در حافظهٔ نهان (cache) خود ذخیره می‌نماید. اگر «عامل تحقیق و تفحص» متوجه شود که قطعه‌ای برای ماشین متحرک ارسال شده ولی Ack آن در مدت زمان معقول و کوتاهی برنگشته، آن قطعه را بدون اطلاع دادن به فرستندهٔ آن از نو ارسال می‌کند. همچنین وقتی یک Ack تکراری از ماشین متحرک می‌بیند متوجه می‌شود که این ماشین چیزی را از دست داده است. Ackهای تکراری نیز در «عامل تحقیق و تفحص» حذف می‌شوند تا ماشین مبداء، دریافت آنها را اشتباهاً بمعنای بروز ازدحام تعبیر نکند. در این روش سعی می‌شود ناهمگونی شبکه‌ها به نحو زیرکانه‌ای از چشم طرفین پنهان بماند.

یک اشکال در این روش نامرئی آنست که اگر لینک بی‌سیم بسیار پر خطا باشد و درصد بالایی از بسته‌ها را نابود کند، ممکن است ماشین مبداء به دلیل عدم دریافت Ack در مهلت مقرر، الگوریتم کنترل ازدحام خود را فراخوانی و اعمال نماید. در روش «TCP غیرمستقیم» الگوریتم کنترل ازدحام هرگز شروع نخواهد شد مگر آنکه واقعاً در بخش سیمی شبکه ازدحام رخ داده باشد.

در مقالهٔ Balakrishnan و همکاران او، راه حلی نیز برای مشکل از بین رفتن بسته‌های ماشین متحرک (بی‌سیم) ارائه شده است: وقتی ایستگاه ثابت متوجه می‌شود که در بین داده‌های ارسالی از ماشین متحرک یک

قطعهٔ جا افتاده وجود دارد، با استفاده از یک گزینهٔ جدید در فیلد Option از قطعهٔ TCP، تقاضایی را تولید کرده و جهت دریافت این قطعهٔ جا افتاده برای ماشین متحرک می‌فرستد. به کمک این اصلاحیه، لینک بی‌سیم در دو جهت قابل اعتمادتر می‌شود بدون آن که مبداء چیزی در این خصوص بداند و بی‌آن که مفهوم واقعی TCP نقض شود.

اگرچه UDP از مشکلاتی که TCP با آن مواجه است، رنج نمی‌برد ولی UDP نیز در محیطهای بی‌سیم مشکلات خاص خود را دارد. مشکل عمده آن است که برنامه‌های کاربردی استفاده‌کننده از UDP انتظار قابلیت اعتماد بالا دارند. اگرچه UDP تحویل داده‌ها را تضمین نکرده ولیکن انتظار می‌رود که حتی‌الامکان درست عمل کند. [یعنی درصد بسیار ناچیزی از داده‌ها از بین برود.] در محیط بی‌سیم، UDP بسیار نامطمئن و پرخطا عمل خواهد کرد. اگر آن دسته از برنامه‌های کاربردی که می‌توانند پیامهای UDP گمشده را تشخیص داده و بازیابی کنند، از محیطی که در آن احتمال از بین رفتن بسته‌ها ناچیز است به محیطی که بخشی از داده‌ها بطور دائم از دست می‌روند، منتقل شوند منجر به کاهش بحرانی کارآیی آنها خواهد شد.

ارتباط بی‌سیم، جدای از مسئله کارآیی، در زمینه‌های دیگری نیز تأثیر منفی دارد. به عنوان مثال چگونگی پیدا کردن یک چاپگر محلی و برقراری اتصال با آن، خود یک مسئله است. یا مشکل دیگر آنست که چگونه می‌توان به صفحه وب محلی در سلول جاری دسترسی پیدا کرد در حالی که نام آن را نمی‌دانیم. معمولاً طراحان صفحهٔ وب [در محیطهای شبکه محلی] فرض را بر آن می‌گذارند که پهنای باند موجود فراوان است. حال اگر در هر صفحهٔ وب لوگویی قرار داده شود که مراجعه به آن در یک شبکهٔ بی‌سیم مثلاً ده ثانیه طول می‌کشد، خوشایند کاربر نخواهد بود.

شبکه‌های بی‌سیم در حال رواج هستند و مشکل اجرای TCP بر روی اینگونه شبکه‌ها روز به روز حادتر می‌شود. در این خصوص کارهای دیگری نیز انجام شده که گزارش آنها در مراجع زیر در دسترس است:

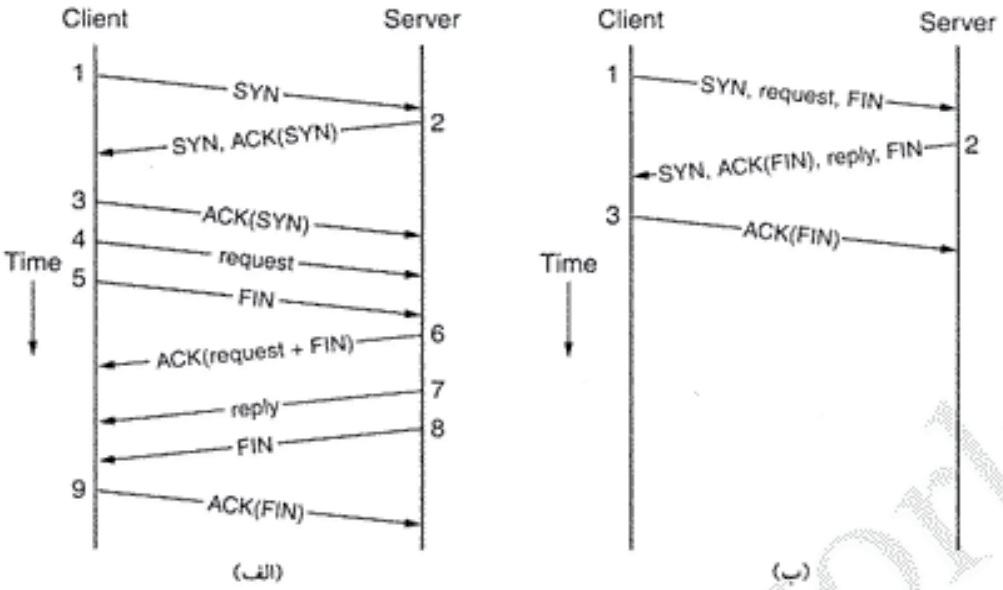
Barakat et al., 2000; Ghani and Dixit, 1999; Huston, 2001; and Xylomenos et al., 2001)

۱۲-۵-۶ TCP تراکنشی (Transactional TCP)

قبلاً در این فصل به موضوع فراخوانی پروسیجرهای راه دور (RPC) به عنوان روشی برای پیاده‌سازی سیستمهای سرویس‌دهنده/مشتری، نگاهی انداختیم. اگر «تقاضا» (request) و «پاسخ» (Reply) آنقدر کوچک باشند که در یک بستهٔ واحد جا بگیرند و تقاضاهای متوالی، مستقل از هم باشند، بسادگی می‌توان از UDP بهره گرفت. ولیکن اگر این شرایط برقرار نباشد، استفاده از UDP چندان مفید نخواهد بود. به عنوان مثال اگر پاسخ به یک تقاضا، داده‌ای بسیار بزرگ باشد، قطعات داده باید شماره‌گذاری شده و مکانیزمی جهت ارسال مجدد قطعات از بین رفته، ابداع و پیاده‌سازی شود. در نتیجه، برنامهٔ کاربردی ملزم به پیاده‌سازی عملیات TCP خواهد بود.

روشن است که این رویکرد جالب نیست ولی استفاده از TCP به جای UDP نیز فایده‌ای ندارد. مشکل استفاده از TCP، کارآیی آن است. اگر از RPC بر روی TCP استفاده شده باشد، توالی بسته‌هایی که بین سرویس‌دهنده و مشتری (برای انجام یک فراخوانی راه دور) مبادله می‌شوند در شکل ۶-۴۰-الف نشان داده شده است. در بهترین حالت باید ۹ بسته مبادله شود. این نه بسته عبارتند از:

۱. برنامهٔ مشتری، یک بستهٔ SYN به منظور برقراری اتصال برای سرویس‌دهنده ارسال می‌کند.
۲. سرویس‌دهنده در پاسخ به دریافت بستهٔ SYN، یک بستهٔ ACK بازپس می فرستد.
۳. برنامهٔ مشتری فرآیند دست تکانی سه مرحله‌ای را تکمیل می‌نماید.
۴. برنامهٔ مشتری، تقاضای اصلی خود را ارسال می‌دارد.
۵. برنامهٔ مشتری، بستهٔ FIN را جهت اعلام خاتمهٔ کار خود می‌فرستد.



شکل ۶-۴۰. (الف) عمل RPC بکمک TCP معمولی. (ب) عمل RPC بکمک T/TCP.

۶. سرویس‌دهنده دریافت بستهٔ حاوی تقاضا و FIN را تصدیق می‌کند. (با ارسال ACK)
۷. سرویس‌دهنده پاسخ مورد نظر مشتری را باز می‌گرداند.
۸. سرویس‌دهنده یک بستهٔ FIN می‌فرستد تا خاتمهٔ کار خود را اعلام نماید.
۹. مشتری، دریافت بستهٔ FIN متعلق به سرویس‌دهنده را تصدیق می‌کند.

دقت کنید که این ۹ مرحله در بهترین حالت ممکن اتفاق می‌افتد. در بدترین حالت، تقاضای مشتری و بستهٔ FIN بطور جداگانه اعلام وصول می‌شود و به همین نحو پاسخ سرویس‌دهنده و بستهٔ FIN آن بطور مجزا ارسال می‌گردد.

سؤالی که مطرح می‌شود آنست که آیا راهی وجود دارد که کارآیی و سرعت RPC مبتنی بر UDP (دقیقاً با دو پیام) و قابلیت اعتماد TCP را با هم تلفیق کرد. پاسخ به این سؤال این است: تقریباً! این کار را می‌توان با یک گونهٔ آزمایشی از TCP که اصطلاحاً T/TCP (Transactional TCP) نامیده می‌شود انجام داد. شرح T/TCP در RFC 1379 و RFC 1644 آمده است.

ایدهٔ اصلی در T/TCP آن است که روند استاندارد برقراری یک اتصال را به گونه‌ای اصلاح کنیم که بتوان در حین برقراری اتصال، داده‌ها را نیز ارسال کرد. پروتکل T/TCP در شکل ۶-۴۰-ب نشان داده شده است. اولین بستهٔ ارسالی توسط مشتری، حاوی بیت SYN، اصل تقاضا و بیت FIN است. در حقیقت می‌گوید: «مایل به برقراری یک اتصال هستم، این هم داده‌های من، کارم نیز به اتمام رسید!»

وقتی سرویس‌دهنده، این تقاضا را دریافت می‌کند، پاسخ آن را پیدا یا محاسبه کرده و سپس چگونگی ارسال آن را انتخاب می‌نماید: اگر پاسخ در یک بستهٔ واحد جا بگیرد، بسته‌ای را که در شکل ۶-۴۰-ب نشان داده شده، ارسال می‌دارد. در حقیقت اعلام می‌کند: «دریافت FIN شما را تأیید می‌کنم، این هم پاسخ شما، کار من هم تمام شد!» در اینجا، مشتری دریافت FIN سرویس‌دهنده را تصدیق کرده و پروتکل با مبادلهٔ سه پیام خاتمه می‌یابد.

ولیکن اگر پاسخ بزرگتر از یک بسته باشد، سرویس‌دهنده می‌تواند بیت FIN را فعال نکند و چندین بسته بفرستد. سپس در آخرین بسته FIN را فعال و ارسال نماید.

اشاره به این نکته ارزشمند است که T/TCP تنها نسخهٔ بهبود یافتهٔ TCP برای عملیات تراکنشی نیست.

پیشنهاد دیگر SCTP (Stream Control Transmission Protocol) است. از ویژگیهای آن می‌توان به موارد ذیل اشاره کرد: (۱) حفظ مرز پیام (۲) پشتیبانی از چندین حالت تحویل (مثل تحویل نامرتب) (۳)حمایت از Multihoming (ماشینهای مقصد پشتیبان) (۴) اعلام وصول بسته‌ها به صورت انتخابی (Selective Repeat). (Stewart and Metz, 2001) با این حال وقتی یک نفر پیشنهاد ایجاد تغییر در پروتکلی را که برای سالها به خوبی کار کرده، می‌دهد یک مناقشهٔ عظیم بین طرفداران این دو نظریه در می‌گیرد: «افرادی که معتقدند باید برای پاسخ به نیاز کاربران ویژگیهای بیشتری را به پروتکل افزود» و «افرادی که معتقدند تا وقتی پروتکل به بن‌بست نخورده نباید آن را اصلاح کرد!»

۶-۶ مسائل مرتبط با کارآیی[1]

مسائل مرتبط با کارآیی در شبکه‌های کامپیوتری از اهمیت ویژه‌ای برخوردارند. وقتی صدها یا هزاران کامپیوتر به هم متصل می‌شوند، تعامل پیچیدهٔ آنها با تبعات غیرمنتظره‌ای همراه است. اکثراً این پیچیدگی منجر به کارآیی بسیار ضعیف شبکه می‌شود و هیچکس هم نمی‌داند علت چیست. در بخشهای آتی به بررسی مواردی خواهیم پرداخت که به کارآیی شبکه مربوط می‌شود تا ببینیم که چه مشکلاتی وجود دارد و با آنها چه باید کرد.

متأسفانه، تشخیص کارآیی شبکه بیشتر یک هنر است تا یک علم! تئوری کمی وجود دارد که بتوان در عمل از آن بهره گرفت. بهترین کاری که می‌توانیم انجام بدهیم آنست که برخی از قواعد تجربی با پشتوانه محکم و مثالهایی از دنیای واقعی را معرفی کنیم. تعمداً این مبحث را تا اینجا به تعویق انداختیم تا پس از مطالعهٔ TCP بتوانیم از آن به عنوان مثال استفاده نماییم.

لایهٔ انتقال تنها نقطهٔ بروز مشکلات مرتبط با کارآیی نیست. در فصل قبلی برخی از این مشکلات را در لایهٔ شبکه بررسی کردیم. علیرغم این، لایهٔ شبکه بیشتر درگیر مسائل مسیریابی و کنترل ازدحام است. موارد مرتبط با سیستمهای نهایی به لایهٔ انتقال بر می‌گردد لذا این فصل جایگاه مناسبی برای بررسی مشکلات مرتبط با کارآیی است. در پنج بخش بعدی به پنج جنبه از کارآیی شبکه نگاهی خواهیم انداخت:

۱. مشکلات کارآیی
۲. اندازه‌گیری کارآیی شبکه
۳. طراحی سیستم برای رسیدن به کارآیی بهتر
۴. پردازش سریع TPDU
۵. پروتکلهایی برای شبکه‌های کارآمد در آینده

برای هر واحد اطلاعات که توسط لایهٔ انتقال مبادله می‌شود، نیاز به یک اسم عمومی داریم. واژهٔ بکار رفته در TCP یعنی «قطعه» (Segment) گمراه‌کننده است و در هیچ جای دیگر به غیر از پروتکل TCP از این واژه استفاده نشده است. واژه‌های بکار رفته در ATM (مثل CS-PDU ، SAR-PDU و CPCS-PDU) نیز خاص شبکهٔ ATM هستند. واژهٔ «بسته» در لایهٔ شبکه و واژهٔ «پیام» در لایهٔ کاربرد بکار می‌روند. به دلیل فقدان یک استاندارد و اجماع عمومی، ما از همان واژهٔ TPDU استفاده خواهیم کرد. وقتی بخواهیم به بسته و TPDU درون آن بطور همزمان اشاره کنیم واژهٔ کلی «بسته» را بکار خواهیم برد مثلاً وقتی می‌گوییم: «CPU باید آنقدر سریع باشد که بتواند بسته‌های ورودی را به صورت بی‌درنگ پردازش کند» منظورمان هم بستهٔ لایهٔ شبکه و هم TPDU جاسازی شده در آن می‌باشد.

---

۱. Performance

۶-۶-۱ مشکلات کارآیی در شبکه‌های کامپیوتری

منشاء برخی از مشکلات کارآیی مثل ازدحام به استفادهٔ بیش از حد از منابع موجود شبکه برمی‌گردد. اگر به ناگاه ترافیکی بیش از توان و ظرفیت مسیریاب، به آن تحویل داده شود ازدحام پدید آمده و موجب کاهش کارآیی می‌شود. در فصل قبلی مفصلاً به موضوع ازدحام پرداختیم.

همچنین وقتی که منابع شبکه از لحاظ ساختاری توازن و تعادل نداشته باشند کارآیی کاهش خواهد یافت. به عنوان مثال اگر یک خط ارتباطی یک گیگابیتی به یک PC معمولی متصل شده باشد پردازندهٔ ضعیف این کامپیوتر قادر نخواهد بود که بسته‌های ورودی را به سرعت پردازش کند و برخی از آنها از دست می‌رود. بسته‌های از دست رفته نهایتاً از نو ارسال می‌شوند و طبعاً تأخیر افزایش می‌یابد، پهنای باند هدر می‌رود و در مجموع کارآیی شبکه افت می‌کند.

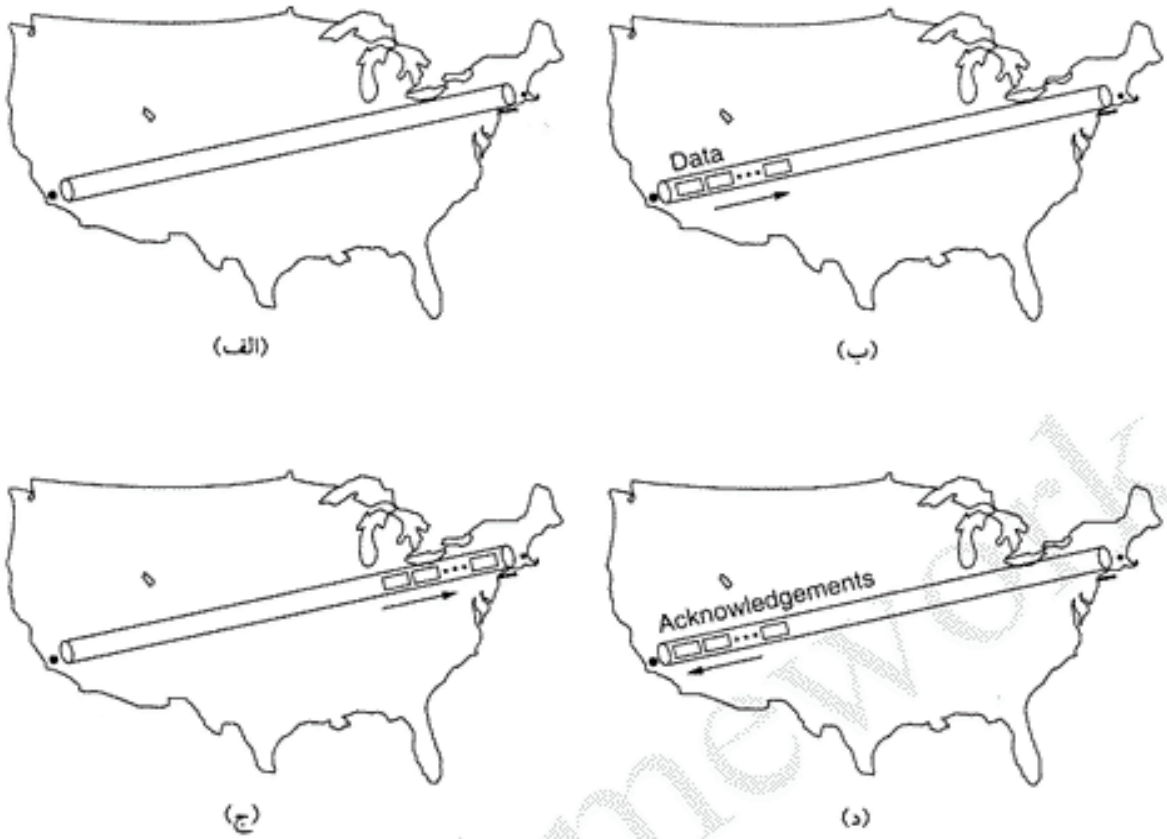
بار بیش از حد (Overload) نیز ممکن است بطور لحظه‌ای اتفاق افتاده و کارآیی شبکه را کاهش بدهد. به عنوان مثال اگر یک TPDU حاوی یک پارامتر اشتباه باشد (مثلاً شمارهٔ پورت مقصد آن صحیح نباشد)، در بسیاری حالات، گیرنده یک گزارش خطا باز می‌گرداند. حال در نظر بگیرید که اگر یک TPDU حاوی پارامتر اشتباه به صورت پخش فراگیر (Broadcast) برای ۱۰۰۰۰ ماشین ارسال شود چه اتفاقی می‌افتد: ممکن است همهٔ آنها یک پیغام خطا برگردانند! این اتفاق منجر به بروز «**طوفان پخش فراگیر**» (Broadcast Storm) شده و می‌تواند شبکه را فلج کند. UDP از این مشکل رنج می‌برد مگر آن که این پروتکل به نحوی تغییر کند که در مواجهه با TPDUهای اشتباه که به صورت فراگیر پخش شده‌اند پیغام خطا برنگرداند.

مثال دوم از تحمیل بار بیش از اندازه به شبکه (بطور همزمان)، پس از قطع برق اتفاق می‌افتد. وقتی برق مجدداً بر می‌گردد همهٔ ماشینها بطور همزمان از طریق ROM خود سعی می‌کنند از نو راه‌اندازی شوند. عموماً در روال راه‌اندازی، هر ماشین شبکه در ابتدا به برخی از سرویس‌دهنده‌ها (مثل DHCP) مراجعه می‌کند تا اول هویت خود را بشناسد؛ پس از آن برای دریافت سیستم عامل به سرویس‌دهندهٔ فایل مراجعه می‌نماید. اگر صدها ماشین به طور همزمان این کار را انجام بدهند، احتمال دارد سرویس‌دهنده در زیر چنین باری از کار بیفتد.

حتی اگر بار، بیش از حد و همزمان اتفاق نیفتد و منابع شبکه نیز بقدر کفایت در اختیار باشد باز هم ممکن است به دلیل عدم تنظیم صحیح سیستم، کارآیی شبکه کاهش یابد. به عنوان مثال اگر ماشینی دارای پردازندهٔ بسیار سریع و حجم انبوه حافظه باشد ولیکن به قدر کافی برای فضای بافر حافظه اختصاص داده نشود ممکن است کمبود بافر به از دست رفتن TPDU بینجامد. همچنین اگر الگوریتم زمان‌بندی پروسه‌ها، اولویت بالایی را به پردازش TPDUها ندهد باز هم ممکن است برخی از بسته‌ها از بین بروند.

مشکل دیگر، تنظیم صحیح مهلت تایمرهاست. وقتی یک TPDU ارسال می‌گردد برای آن که بتوان از گم شدن آن آگاه شد یک تایمر برای آن تنظیم می‌شود. اگر مهلت این تایمر کوتاه در نظر گرفته شود، ارسالهای مجدد بیهوده اتفاق می‌افتد و پهنای باند کانال را هدر می‌دهد. اگر هم این زمان طولانی در نظر گرفته شود در هنگام از دست رفتن یک TPDU تأخیر بی‌مورد تحمیل می‌شود. پارامتر دیگری که باید تنظیم شود مهلت تایمری است که وقتی داده‌ای دریافت می‌شود و نمی‌خواهیم برای آن ACK مجزا بفرستیم باید به آن مدت صبر کرد تا داده‌ای جهت ارسال تولید شود و بتوان ACK را به آنها ضمیمه نمود. اگر این زمان انتظار زیاد در نظر گرفته شود، ارسالهای تکراری و بیهوده اتفاق می‌افتد.

شبکه‌های گیگابیتی مشکلات جدیدی در کارآیی شبکه به همراه آورده‌اند. به عنوان مثال در نظر بگیرید که یک تودهٔ ۶۴کیلوبایتی داده از سن دیه‌گو به بوستون ارسال می‌شود تا در بافر ۶۴کیلوبایتی گیرنده قرار بگیرد. فرض کنید تأخیر انتشار این لینک (با احتساب سرعت نور در فیبر) معادل ۲۰میلی‌ثانیه باشد. در لحظهٔ t=0 خط خالی



شکل ۶-۴۱. وضعیت ارسال یک مگابیت داده از سن‌دیه گو به بوستون. (الف) در لحظهٔ t=0. (ب) پس از گذشت ۵۰۰ مایکروثانیه. (ج) پس از گذشت ۲۰ میلی ثانیه. (ب) پس از گذشت ۴۰ میلی ثانیه.

است و به نحوی که در شکل ۶-۴۱-الف می‌بینید داده‌ای بر روی خط ارسال نشده است. حدود ۵۰۰ میکروثانیه بعد، تمام TPDUهای حاوی ۶۴ کیلوبایت داده بر روی فیبر قرار گرفته‌اند و ارسال آنها خاتمه یافته است. حالا اولین TPDU ارسالی جایی در نزدیکی براولی (Brawley) است! (شکل ۶-۴۱-ب) با این حال فرستنده باید ارسال خود را تا اعلام مجدد اندازهٔ پنجره متوقف کند چرا که متناسب با اندازهٔ پنجرهٔ گیرنده، داده ارسال شده است.

پس از ۲۰ میلی‌ثانیه، اولین TPDU به بوستون می‌رسد و طبق شکل ۶-۴۱-ج اعلام وصول آنها آغاز می‌شود. ۴۰ میلی‌ثانیه پس از شروع، اولین پیغام ACK به فرستنده باز می‌گردد و دومین تودهٔ داده را می‌توان ارسال کرد. از آنجایی که در طی ۴۰ میلی‌ثانیه، فقط ۰/۵ میلی‌ثانیه از خط انتقال استفاده شده، کارآیی آن حدود ۱/۲۵ درصد است. چنین وضعیتی برای پروتکلهای قدیمی که بر روی خطوط گیگابیتی اجرا می‌شوند، کاملاً طبیعی است.

یکی از کمیتهای بسیار مهمی که در هنگام تحلیل کارآیی شبکه باید به خاطر داشته باشید «**حاصل ضرب پهنای باند در تأخیر**» (Bandwidth-Delay Product) است. این کمیت را می‌توان با ضرب مقدار پهنای باند (برحسب بیت بر ثانیه) در زمان تأخیر رفت و برگشت خط (برحسب ثانیه) محاسبه کرد. این حاصلضرب، ظرفیت خط لولهٔ بین فرستنده و گیرنده و بالعکس را برحسب بیت تعیین می‌کند.

به عنوان مثال در شکل ۶-۴۱ حاصلضرب پهنای باند در تأخیر، معادل ۴۰ میلیون بیت است. یعنی فرستنده مجبور است یک تودهٔ ۴۰ میلیون بیتی از داده‌ها را بفرستد تا اولین پیغام ACK باز گردد. این تعداد بیت کل خط را

در دو جهت پر می‌کند.[1] این همان دلیلی است که راندمان خط برای ارسال نیم میلیون بیت (۶۴کیلوبایت) داده به مقدار ۱/۲۵ درصد کاهش می‌یابد زیرا فقط ۱/۲۵ درصد از حجم ۴۰ میلیون بیتی ارسال و متوقف شده است.

نتیجه‌ای که می‌توان گرفت آنست که برای کارآیی خوب ، پنجرهٔ گیرنده حداقل باید به بزرگی «**حاصلضرب پهنای باند در تأخیر**» باشد. (حتی ترجیحاً بیشتر زیرا ممکن است گیرنده نتواند فوراً پاسخ بدهد.) برای خطوط گیگابیتی بین قاره‌ای به حداقل ۵ مگابایت فضای حافظه نیاز است.

اگر کارآیی خط برای ارسال یک مگابیت داده اینقدر پایین باشد تصور کنید که برای یک تقاضای کوتاه چند صد بایتی چقدر است! اگر وقتی اولین مشتری منتظر دریافت پاسخ از طرف مقابل است از خط استفادهٔ دیگری نشود، خط یک گیگابیتی هیچ مزیتی بر خط یک مگابیت بر ثانیه نخواهد داشت؛ فقط گرانتر است.

مشکل دیگر کارآیی، که برای کاربردهای حساس به زمان مثل صدا و تصویر رخ می‌دهد، «لرزش» (Jitter) است. کوتاه بودن زمان متوسط انتقال کافی نیست؛ انحراف معیار این زمان نیز باید پایین باشد. رسیدن به زمان متوسط انتقال کم و انحراف معیار پایین مستلزم تلاشهای مهندسی بسیار جدی است.

۶-۶-۲ اندازه‌گیری کارآیی شبکه

وقتی شبکه ضعیف و ناکارآمد عمل می‌کند کاربران آن شروع به شکوه و گلایه کرده و خواهان بهبود کارآیی می‌شوند. برای بهبود کارآیی، اپراتورها باید اول تعیین کنند که مشکل از کجا منشاء می‌گیرد. بمنظور پیگیری و تشخیص مشکل بوجود آمده، آنها مجبور به انجام پاره‌ای محاسبات و اندازه‌گیری هستند. در این بخش به موضوع اندازه‌گیری کارآیی شبکه نگاهی می‌اندازیم. مباحث ذیل برگرفته از پژوهشهای Mogul (۱۹۹۳) می‌باشد. روال بهبود کارآیی شبکه شامل مراحل زیر است:

۱. پارامترهای شبکهٔ مورد نظر و کارآیی آن را اندازه‌گیری کنید.
۲. سعی کنید وضعیت فعلی شبکه را ارزیابی نمایید.
۳. یکی از پارامترها را تغییر بدهید.

این مراحل آنقدر تکرار می‌شوند تا کارآیی شبکه به حد مطلوب برسد یا روشن شود که آخرین حد کارآیی همین است.

اندازه‌گیریها را می‌توان به روشهای متعدد و در موقعیتهای متفاوت انجام داد (به صورت فیزیکی یا در پشتهٔ پروتکل). یکی از اساسی‌ترین نوع اندازه‌گیری آن است که تایمری را در ابتدای انجام یک عمل روشن کرده و بررسی کنیم که آن عمل چقدر طول می‌کشد. به عنوان مثال دانستن زمانی که طول می‌کشد تا پس از ارسال TPDU، دریافت آن تأیید شود، بسیار اهمیت دارد. اندازه‌گیریهای دیگری را نیز می‌توان به کمک شمارنده‌ها انجام داد (مثلاً تعداد TPDUهایی که از دست رفته‌اند). یکی دیگر از پارامترهایی که دانستن آن اهمیت دارد تعداد بایتهایی است که در یک مدت زمان معین پردازش می‌شوند.

اندازه‌گیری کارآیی و پارامترهای شبکه پیچیدگیهای خاص خود را دارد. در زیر به برخی از آنها اشاره می‌کنیم. در هر گونه تلاش سیستماتیک برای اندازه‌گیری کارآیی شبکه باید موارد ذیل را مد نظر داشته باشید:

مطمئن شوید که تعداد نمونه‌های آماری به قدر کافی زیاد است

فقط به اندازه‌گیری زمان ارسال یک TPDU اکتفا نکنید بلکه اندازه‌گیریهای خود را مثلاً یک میلیون بار تکرار کرده و میانگین آن را بدست بیاورید. در اختیار داشتن تعداد بسیار زیاد نمونه‌های آماری، عدم قطعیت مقدار میانگین و

---

[1]. به عبارتی پس از ارسال ۴۰ میلیون بیت بر روی یک خط 1Gbps تازه اولین پیغام ACK باز خواهد گشت. فرستنده باید قادر به ارسال و باقر کردن چنین حجمی از داده باشد. -م

انحراف معیار اندازه‌گیری شده را کاهش خواهد داد. میزان «عدم قطعیت» به کمک فرمولهای استاندارد آماری قابل محاسبه است.

دقت داشته باشید که نمونه‌های آماری به صورت جامع انتخاب شده‌اند

ایده‌آل آنست که مثلاً یک میلیون بار اندازه‌گیری و نمونه‌برداری، در زمانهای مختلف یک روز و در طی هفته تکرار شود تا تأثیر بار سیستم در زمانهای متفاوت بر روی کمیت اندازه‌گیری شده، مشخص گردد. به عنوان مثال اندازه‌گیری ازدحام در لحظه‌ای که ازدحام وجود ندارد، چندان مفید نیست. گاهی اوقات نتایج بدست آمده در بدو امر غیرطبیعی به نظر می‌رسند (مثل ازدحام سنگین در خلال ساعت ۱۰ تا ۲ و عدم ازدحام در حول و حوش ظهر ـوقت ناهار– که کاربران کار خود را رها کرده‌اند!).

وقتی از ساعت نه چندان دقیق استفاده می‌کنید مراقب نتیجهٔ اندازه‌گیریها باشید

ساعت کامپیوترها بدین نحو کار می‌کنند که در فواصل زمانی معین به یک یا چند شمارنده، یک واحد می‌افزایند. به عنوان مثال تایمر یک میلی‌ثانیه‌ای در هر میلی‌ثانیه، یک واحد به شمارنده اضافه می‌کند. استفاده از چنین تایمری برای اندازه‌گیری رخدادهایی که کمتر از یک میلی‌ثانیه طول می‌کشد امکان‌پذیر است ولی مستلزم دقت زیادیست. (البته بعضی از کامپیوترها ساعت دقیقتری دارند.)

برای محاسبهٔ زمان ارسال یک TPDU، بایستی ساعت سیستم (که مثلاً دقت آن میلی‌ثانیه است) در ابتدای شروع «کُد برنامهٔ واحد انتقال» و همچنین به محض خاتمهٔ کار خوانده شود. اگر زمان واقعی ارسال یک TPDU، ۳۰۰ میکروثانیه باشد، اختلاف بین دو زمان خوانده شده یا صفر است یا یک، که هر دوی آنها اشتباه است. ولیکن اگر اندازه‌گیری زمان برای ارسال یک میلیون TPDU انجام گیرد و زمان کل بر عدد یک میلیون تقسیم شود، میانگین زمان دقتی حدود یک میکروثانیه خواهد داشت.

مطمئن باشید که در خلال آزمایشات هیچ چیز پیش‌بینی نشده‌ای وجود ندارد

انجام اندازه‌گیری بر روی سیستم یک دانشگاه در روزی که مثلاً چندین پروژهٔ آزمایشگاهی عظیم تحت آزمایش است می‌تواند نتیجهٔ کاملاً متفاوتی با اندازه‌گیریهای روز بعد داشته باشد. به دلیل مشابه، اگر چندین پژوهشگر در حین آزمایشهای آماری شما تصمیم به اجرای یک کنفرانس ویدیویی بر روی شبکه بگیرند ممکن است شما را با نتایج گمراه‌کننده‌ای مواجه سازند. بهترین کار آن است که بررسیها را بر روی یک سیستم بیکار انجام بدهید و باری را که می‌خواهید بر روی شبکه یا سیستم بگذارید خودتان ایجاد کنید. البته این راهکار هم ممکن است معضلات خود را داشته باشد. شاید با خود بیندیشید که هیچکس در ساعت ۳ بامداد از شبکه استفاده نمی‌کند در حالی که امکان دارد در همین زمان برنامه‌هایی که بطور خودکار نسخه‌ای پشتیبان از دیسک را بر روی نوار مغناطیسی منتقل می‌کنند در حال کار باشند. مضاف بر این ممکن است ترافیک سنگینی از وب‌سایت شما به نقطه‌ای از جهان (که ساعت محلی آنها متفاوت است) در حال انتقال باشد.

فرآیند ذخیره در حافظهٔ نهان (Caching) می‌تواند صحّت نتایج اندازه‌گیری را به خطر بیندازد

روش بدیهی برای اندازه‌گیری زمان انتقال فایل آن است که یک فایل بزرگ را باز کنید، کلّ آن بخوانید و نهایتاً آنرا ببندید و سپس زمان کلّ را محاسبه نمایید. برای بالا بردن دقت محاسبه، باید این کار چندین بار تکرار شده و میانگین گرفته شود. مشکل اینجاست که امکان دارد، سیستم این فایل را در حافظه نهان ذخیره کرده باشد؛ بنابراین فقط اولین اندازه‌گیری با ترافیک شبکه سر و کار دارد و بقیهٔ اندازه‌گیریها کاملاً غلط هستند چرا که از بافر محلی خوانده می‌شوند. نتیجه‌ای که از چنین آزمایشی بدست می‌آید به کل فاقد اعتبار است. (مگر آن که خواسته باشید

کارآیی حافظهٔ نهان ـ cache ـ را اندازه‌گیری کنید!)

اغلب می‌توانید با سرریز کردن بافر آن را دور بزنید. به عنوان مثال اگر فضای حافظهٔ نهان ۱۰ مگابایت باشد در حلقهٔ آزمایش می‌توانید **دو** فایل ده مگابایتی را باز کنید؛ سپس آنها را پشت سرهم بخوانید و نهایتاً آنها را ببندید تا در هر بار که یکی از فایلها آزمایش می‌شود، میزان استفاده از حافظهٔ نهان (cache) به صفر برسد. با این وجود توصیه می‌شود با احتیاط این کار را انجام دهید مگر آن که کاملاً با الگوریتم caching خود آشنا باشید.

بافرسازی داده‌ها نیز نتیجهٔ مشابهی دارد. مشهور است که یکی از برنامه‌های رایج اندازه‌گیری کارآیی TCP/IP، کارآیی UDP را بیشتر از ظرفیت ممکن خط، گزارش می‌کند!! چرا این اتفاق افتاده است؟ دلیل این است که پس از فراخوانی UDP به محض آن که پیام ارسالی تحویل هستهٔ سیستم عامل می‌شود، آن پیام در انتهای صف ارسال قرار گرفته و کنترل اجرا به برنامهٔ فراخواننده بر می‌گردد. اگر فضای بافر به مقدار کافی وجود داشته باشد حتی هزار بار فراخوانی UDP به معنای ارسال آنها نخواهد بود. بیشتر آنها هنوز در اختیار هستهٔ سیستم عامل هستند ولی ابزار محاسبهٔ کارآیی فکر می‌کند که واقعاً ارسال شده‌اند.

دقیقاً بدانید که چه چیزی را اندازه‌گیری می‌کنید

وقتی زمان خوانده شدن یک فایل راه دور را اندازه می‌گیرید، اندازه‌گیری شما به شبکه ، سیستم عامل دوطرف (سرویس‌دهنده و مشتری) ، سخت‌افزار واسط شبکه ، برنامه‌های راه‌انداز آنها (drivers) و عوامل دیگر بستگی دارد. در صورتی که اندازه‌گیریها بدقت انجام شده باشد، زمان انتقال فایل براساس پیکربندی فعلی بدست می‌آید. اگر هدف نهایی شما تنظیم همین پیکربندی است این اندازه‌گیری مفید خواهد بود ولیکن اگر همین اندازه‌گیری را بر روی سه سیستم انجام می‌دهید تا براساس آن یک کارت شبکهٔ مناسب برای خرید انتخاب کنید، نتایج بدست آمده می‌تواند کاملاً فاقد اعتبار باشد چرا که مثلاً برنامهٔ راه‌انداز کارت شبکه نامناسب بوده و فقط از ده درصد کارآیی کارت شبکه ، بهره‌برداری کرده است!

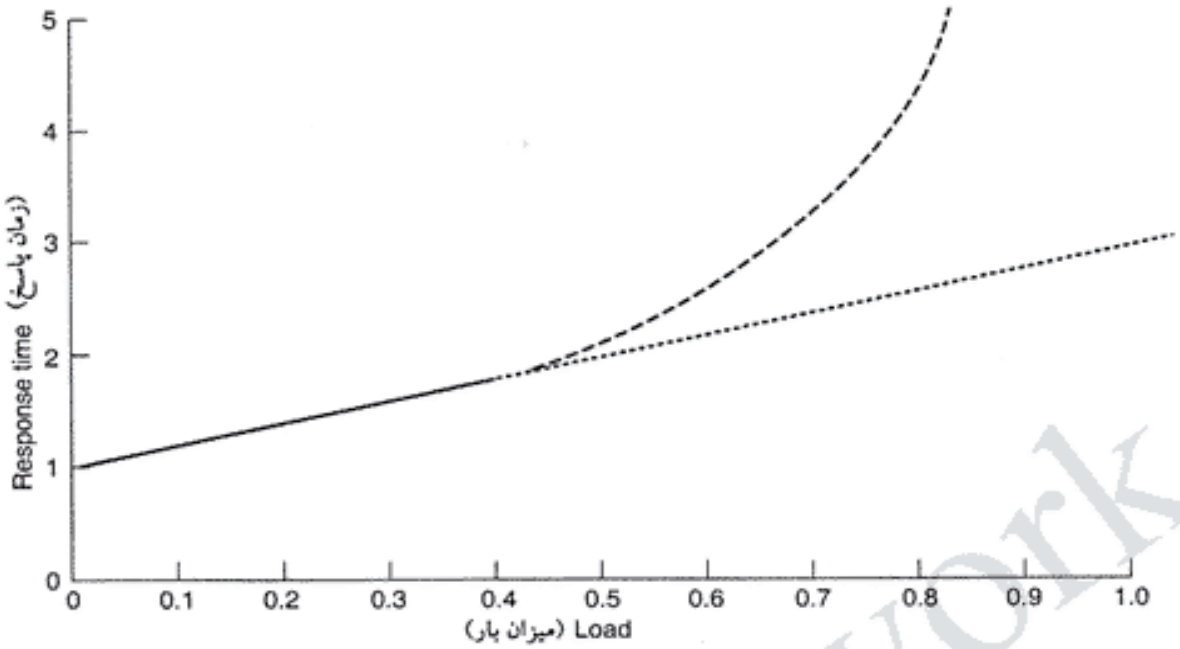
در خصوص نتایج حاصل از برون‌یابی دقت (Extrapolating) داشته باشید

فرض کنید که اندازه‌گیری پارامتری مثل زمان پاسخ را با ایجاد بار شبیه سازی شده از صفر (بیکار) تا ۰/۴ (۴۰ درصد کل ظرفیت) اندازه‌گیری کرده‌اید و یک نمودار شبیه به خط توپر در شکل ۶-۴۲ بدست آورده‌اید. براساس برون‌یابی خطی، نتیجه‌ای اشتباه مثل خط نقطه‌چین بدست می‌آید. در حالی که نتایج آزمایشاتی که با صفّ سر و کار دارند از عاملی به صورت $1/(1-\rho)$ تأثیر می‌پذیرند که در آن ρ، پارامتر بار (Load) است لذا مقادیر واقعی شبیه به نمودار خط‌چین هستند که شیب بیشتری نسبت به نمودار خطی دارد.

۶-۶-۳ طراحی سیستم برای کارآیی بهتر

اندازه‌گیری و بهره‌برداری از نتایج محاسبات اغلب می‌تواند کارآیی را تا حد قابل توجهی افزایش بدهد ولیکن نمی‌تواند از همان ابتدا جایگزین طراحی خوب شود. کارآیی یک شبکه با طراحی ضعیف را فقط تا حدی می‌توان بهبود داد. پس از آن باید شبکه را از نو طراحی کرد.

در این بخش قواعدی را برای طراحی ارائه می‌دهیم که مبتنی بر تجارب گسترده از شبکه‌های متعدد است. این قواعد مرتبط با طراحی سیستم هستند نه طراحی شبکه، زیرا نرم‌افزار و سیستم عامل، اغلب نقش مهمتری در مقایسه با مسیریابها و کارتهای واسط شبکه در کارآیی ایفاء می‌کنند. بیشتر نظریاتی که ارائه می‌شود برای طراحان شبکه یک دانش پایهٔ عادی و حاصل تجربیاتی است که نسل به نسل منتقل شده و تکامل یافته است. این نظریات برای اولین بار توسط Mogul (۱۹۹۳) به صورت مدون ارائه شد که ما نیز از وی پیروی می‌کنیم. در این زمینه مرجع (Metcalfe, 1993) نیز برای مطالعه مناسب است.



شکل ۶-۴۲. «زمان پاسخ» بعنوان تابعی از «بار».

قانون شمارهٔ ۱: سرعت CPU از سرعت شبکه مهمتر است

تجارب طولانی مدت نشان داده که تقریباً در تمام شبکه‌ها، سربار سیستم عامل و پشتهٔ پروتکلی از زمان واقعی استفاده از کانال بیشتر است.[۱] به عنوان مثال از دیدگاه تئوری زمان لازم برای یک عمل RPC (یعنی فراخوانی پروسیجر از راه دور) بر روی شبکهٔ اترنت حداقل ۱۰۲ میکروثانیه است. (شامل انتقال یک تقاضای ۶۴ بایتی و پاسخ ۶۴ بایتی). در عمل غلبه کردن بر سربار تحمیلی توسط نرم‌افزار و رساندن زمان RPC به همین حدود موفقیت بزرگی محسوب می‌شود.

به دلیل مشابه، بزرگترین مشکل کار کردن در سرعت 1Gbps ، استخراج بیتها از بافر کاربر و انتقال آن بر روی فیبرنوری و همچنین در اختیار داشتن پردازنده‌ای است که در سمت مقابل بتواند با سرعت کافی آنها را دریافت کند. کوتاه سخن آن که اگر سرعت CPU را دو برابر کنید اغلب توان خروجی نیز به نزدیکی دو برابر می‌رسد. دو برابر کردن ظرفیت شبکه معمولاً نتیجه‌ای در پی ندارد چرا که گلوگاه اصلی در ماشینهای میزبان است.

قانون شمارهٔ ۲: کاهش تعداد بسته‌ها برای کاهش سربار نرم‌افزار

پردازش یک TPDU به میزان معینی سربار به ازای هر TPDU (مثلاً برای پردازش سرآیند) و همچنین به میزان معینی سربار به ازای هر بایت (مثلاً برای محاسبهٔ کد کشف خطای Checksum) به CPU تحمیل می‌کند.[۲] وقتی یک میلیون بایت ارسال می‌شود سربار تحمیلی آن به ازای این تعداد بایت ثابت است و به اندازهٔ TPDUها بستگی ندارد. ولیکن سرباری که بدلیل استفاده از TPDUهای ۱۲۸ بایتی تحمیل می‌شود ۳۲ برابر نسبت به زمانی که از TPDUهای ۲ کیلوبایتی استفاده می‌گردد، بیشتر خواهد بود.

مضاف بر سربار TPDU در لایهٔ انتقال، در لایه‌های زیرین هم سربار قابل توجهی تحمیل می‌شود. هر بستهٔ دریافتی یک «وقفه» (interrupt) ایجاد می‌کند. در پردازنده‌های مدرن مبتنی بر معماری Pipeline، بروز وقفه (۱)

---

۱. بعبارتی تأخیر تحمیلی توسط نرم‌افزار می‌تواند بیشتر از تأخیر ناشی از انتقال باشد. -م

۲. Overhead per TPDU / Overhead per Byte

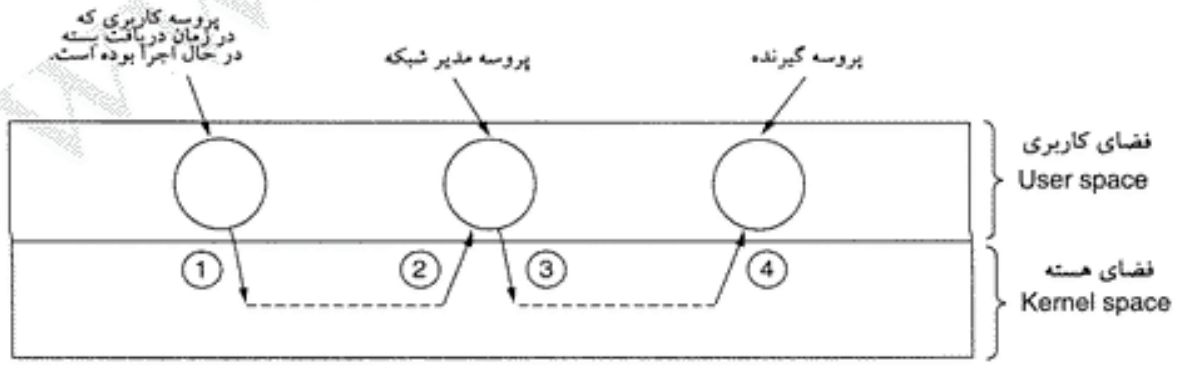
«خط لولهٔ پردازنده» را قطع می‌کند، (۲) حافظهٔ نهان (cache) پردازنده را درهم می‌ریزد، (۳) نیاز به تغییر روال کار مدیریت حافظه (یعنی عمل تعویض فضای کاری حافظه) دارد (۴) CPU را وادار به ذخیره‌سازی تعدادی رجیستر می‌کند. کاهش تعداد TPDUهای ارسالی با ضریب n، تعداد وقفه‌ها و میزان سربار بسته‌ها را با ضریب n کاهش خواهد داد.

این واقعیت بیانگر آنست که برای کاهش وقفه‌ها در طرف مقابل، باید قبل از ارسال، ابتدا مقدار قابل توجهی داده، جمع‌آوری شود. الگوریتم Nagle و راه حل Clark که در بخشهای قبلی تشریح شدند بدین منظور مفید هستند.

قانون شمارهٔ ۳: کاهش تعداد «تعویض متن» (Context Switch)

انجام عملیات «تعویض متن» (مثلاً از حالت هسته به حالت کاربری) از لحاظ سربار تحمیلی مهلک است! این کار تمام ویژگیهای بد وقفه‌ها را دارد و بدترین آنها این است که دنباله‌ای طولانی از محتوای اولیهٔ حافظهٔ نهان CPU از دست می‌رود. داشتن یک پروسیجر کتابخانه‌ای که داده‌های ارسالی را تا رسیدن به حجم قابل توجه بافر کند، به کاهش دفعات «تعویض متن» کمک خواهد نمود. همینطور در سمت گیرنده، TPDUهای کوچک دریافتی باید جمع‌آوری شده و بطور یکجا تحویل کاربر گردد تا دفعات «تعویض متن» کاهش یابد.

در بهترین حالت، یک بستهٔ ورودی موجب می‌شود که سیستم عامل مجبور به «تعویض متن» از حالت کاربر به حالت هسته[1] باشد. سپس باید از هسته به پروسهٔ دریافت‌کنندهٔ این بستهٔ ورودی تغییر حالت بدهد. متأسفانه در بسیاری از سیستمهای عامل به دفعات بیشتری «تعویض متن» نیاز است. به عنوان مثال، اگر مدیر شبکه پروسهٔ خاصی را در فضای کاربری[2] اجرا کرده باشد، دریافت یک بسته موجب می‌شود که سیستم عامل ابتدا از کاربر فعلی به هسته، تغییر وضعیت (و تعویض متن) بدهد، سپس تغییر وضعیتی دیگر از هسته به پروسهٔ مدیر شبکه، سپس تغییری دیگر از پروسهٔ مدیر شبکه به هسته و نهایتاً از هسته به پروسهٔ تحویل گیرندهٔ بسته. این توالی در شکل ۶-۴۳ نشان داده شده است. تمام این عملیات تعویض متن که به ازای ورود هر بسته انجام می‌شود، زمان CPU را تا حد بسیار زیادی هدر خواهد داد و تأثیر مخربی بر روی کارآیی شبکه خواهد گذاشت.



شکل ۶-۴۳. چهار بار تعویض متن برای رسیدن یک بسته به پروسهٔ مدیر شبکه در فضای کاربری.

قانون شمارهٔ ٤: حداقل کردن دفعات کپی برداری

تعدّد کپی برداری حتی از تعدّد دفعات تعویض متن هم بدتر است. این موضوع که بخواهیم یک بستهٔ ورودی را قبل از تحویل محتوای درونی آن به پروسهٔ نهایی، سه یا چهار بار کپی کنیم، امری نامعمول و عجیب نیست. معمولاً

---

۱. User mode to Kernel mode Context Switch

۲. User Space

پس از دریافت یک بسته توسط کارت واسط شبکه و ذخیرهٔ موقت آن در یک بافر سخت‌افزاری ویژه، بسته به درون بافر هستهٔ سیستم عامل کپی می‌شود. از آنجا نیز به بافر لایهٔ شبکه و از آنجا به بافر لایهٔ انتقال و نهایتاً به بافر پروسهٔ کاربردی گیرنده منتقل می‌شود.

یک سیستم عامل هوشمند قادر به کپی یک کلمه در آن واحد است ولیکن ممکن است برای کپی یک کلمه حتی به ۵ دستورالعمل نیاز باشد. (شامل بار کردن کلمه در رجیستر، ذخیرهٔ آن در موقعیت جدید، اضافه کردن به رجیستر شاخص index register- تست برای تشخیص پایان داده‌ها و انشعاب شرطی) سه بار کپی کردن هر بسته در شرایطی که انتقال هر کلمهٔ ۳۲ بیتی با ۵ دستورالعمل انجام می‌شود به ۴÷۱۵ دستورالعمل یعنی حدوداً چهار دستورالعمل به ازای هر بایت، نیاز خواهد داشت. در یک CPU با سرعت 500MIPS ، هر دستورالعمل ۲ نانوثانیه زمان می‌برد لذا سه بار انتقال هر بایت حدوداً ۸ نانوثانیه طول می‌کشد؛ یعنی حدود ۱ نانو ثانیه به ازای هر بیت؛ بدین ترتیب چنین پردازنده‌ای قادر به پردازش 1-Gbps است. وقتی سربار ناشی از پردازش سرآیند، مدیریت وقفه، عملیات تعویض متن (Context Switching) را نیز در نظر بگیریم **شاید** بتوان به سرعت پردازش 500Mbps رسید، در حالی که پردازش اصلی داده‌ها را نیز به حساب نیاورده‌ایم. بدیهی است که پردازش داده‌های شبکهٔ اترنت 10-Gbps که با تمام سرعت ارسال می‌کند، بدین نحو میسر نخواهد بود.

در حقیقت، شاید با CPU فوق، حتی نتوان از عهدهٔ پردازش داده‌های یک خط 500Mbps که با سرعت تمام در جریان است، برآمد. از طرفی در محاسبات فوق، سرعت ماشین را 500Mbps فرض کرده‌ایم یعنی می‌تواند پانصد میلیون دستورالعمل را در هر ثانیه اجرا کند. در دنیای واقعی، ماشینها فقط زمانی می‌توانند با چنین سرعتی کار کنند که به حافظه رجوع نداشته باشند. عملیات حافظه عموماً ده بار کندتر از دستورالعملهای رجیستر به رجیستر اجرا می‌شوند. (به عبارتی ۲۰ نانو ثانیه برای هر دستورالعمل حافظه). اگر بیست درصد از دستورات نیازمند رجوع به حافظه باشند (به عبارتی به بیرون از حافظهٔ نهان رجوع کنند) که در خصوص ورود یک بسته کاملاً محتمل است، متوسط زمان اجرای دستورالعملها ۵/۶ نانو ثانیه خواهد بود. (0.8x2+0.2 ×20) اگر برای پردازش هر بایت چهار دستورالعمل اجرا شود، به ۲۲/۴ نانو ثانیه برای پردازش هر بایت یا ۲/۸ نانو ثانیه برای هر بیت نیاز خواهیم داشت که در این صورفت ظرفیت پردازش به 357Mbps کاهش می‌یابد. اگر پنجاه درصد سربار اضافی را در محاسبات خود وارد کنیم، این مقدار به 178Mbps می‌رسد. در اینجا سخت‌افزار نمی‌تواند کمک بیشتری بکند. مشکل اینجاست که تعداد کپی‌هایی که در سیستم عامل انجام می‌شود زیاد است.

قانون شمارهٔ ۵: می‌توانید پهنای باند بیشتری بخرید ولی تأخیر پایین‌تر خریدنی نیست

سه قانون بعدی به جای آن که به پردازش پروتکل بپردازند در خصوص مسائل ارتباطی شبکه بحث می‌کنند. قانون اول بیان می‌کند که اگر پهنای باند بیشتری خواستید قادر به تهیهٔ آن خواهید بود. قرار دادن یک رشته فیبرنوری دیگر در کنار رشتهٔ قبلی پهنای باند شما را دو برابر می‌کند ولیکن تأخیر را کاهش نخواهد داد. کم کردن تأخیر مستلزم بهبود نرم‌افزار پروتکل، سیستم عامل، یا کارت شبکه است. حتی اگر به تمام این بهبودها نائل شوید ، تأخیر شما هرگز از تأخیر انتقال کمتر نخواهد شد. [بطور ذاتی هر کیلومتر سیم ۵ میکروثانیه و هر کیلومتر فیبرنوری ۳/۳ میکروثانیه تأخیر دارد و این تأخیر اجتناب‌ناپذیر است. -م]

قانون شمارهٔ ۶: پیشگیری از بروز ازدحام بهتر از رفع آن است

قطعاً این مثل قدیمی که: «پیشگیری به مراتب بهتر از درمان است» برای مسئله ازدحام در شبکه نیز صادق است. وقتی شبکه‌ای با ازدحام مواجه می‌شود بسته‌هایی از بین می‌روند، پهنای باند تلف می‌شود، تأخیرهای نامعقول پدید می‌آید و تبعاتی از این قبیل. بیرون آمدن و رفع ازدحام دشوار و زمان‌بر است. بهتر آن است که نگذاریم

ازدحام رخ بدهد. پیشگیری از ازدحام همانند تزریق واکسن DTP است: اندکی دردسر دارد ولی از مشکلات بزرگ پیشگیری می‌کند.

قانون شمارهٔ ۷: اجتناب از انقضای مهلت

وجود تایمرها در هر شبکه‌ای قطعاً لازم است ولی فقط باید در موارد ضروری از تایمر استفاده کرد و مهلت انقضای زمان تایمرها بایستی حداقل (بهینه) باشند. وقتی تایمری به صفر می‌رسد، عموماً عملیاتی از نو تکرار خواهد شد. اگر تکرار این عملیات واقعاً لازم نباشد منابع سیستم بیهوده هدر می‌رود.

برای اجتناب از انجام عملیات زائد باید زمان تایمرها به دقت تنظیم شده و اندکی با مقدار لازم فاصله داشته باشد، زیرا محافظه‌کاری زیاد و تنظیم مقادیر بزرگ در تایمرها باعث بروز تأخیرات نامعقول در هنگام از دست رفتن TPDU می‌شود. از طرفی اگر تایمر در زمانی که واقعاً موعد آن نیست به صفر برسد، زمان CPU صرف عملیات بیهوده می‌شود، پهنای باند به هدر می‌رود و بی‌هیچ دلیل بار زیادی بر روی دهها مسیریاب تحمیل می‌کند.

۶-۶-۴ پردازش سریع TPDU

نتیجه‌گیری منطقی از قضایایی که در بالا بدان پرداختیم آنست که مانع اصلی در ایجاد شبکه‌های سریع، نرم‌افزار پروتکل آنهاست. در این بخش روشهایی را مرور می‌کنیم که به این نرم‌افزار سرعت می‌بخشند. برای کسب آگاهی بیشتر به مراجع (Clark et al., 1989; and Chase et al., 2001) مراجعه نمایید.

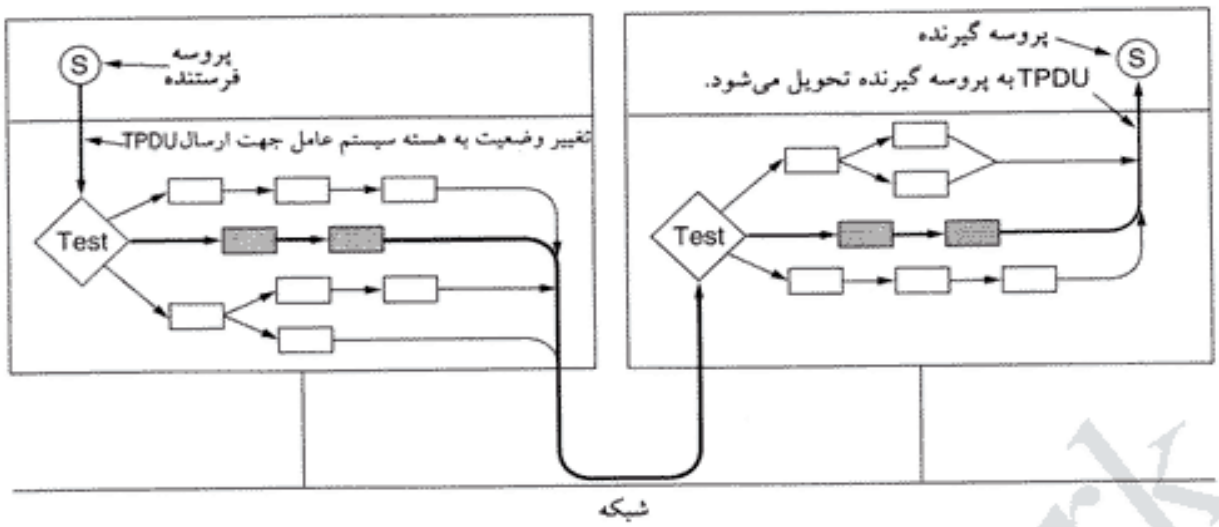
سربار پردازش TPDU دو عامل دارد: سربار تحمیلی به ازای هر TPDU و سربار تحمیلی به ازای هر بایت. هر دوی آنها را باید کاهش داد. عامل کلیدی در پردازش سریع TPDU آن است که TPDUهای معمولی (حاوی داده) را از مابقی جدا کرده و آنها را به صورت ویژه پردازش نماییم. اگرچه به دنباله‌ای از TPDUهای خاص نیاز است تا بتوان به حالت ESTABLISHED وارد شد ولیکن از اینجا به بعد پردازش TPDU سرراست و ساده خواهد بود، تا وقتی که یکی از طرفین اقدام به قطع اتصال نماید.

اجازه بدهید فرض را بر آن بگذاریم که طرف فرستنده اقدام به برقراری اتصال کرده و اکنون در حالت ESTABLISHED قرار دارد و می‌خواهد داده‌هایی را بفرستد. برای سادگی بحث فرض می‌کنیم که «واحد انتقال» (Transport Entity) درون هستهٔ سیستم عامل قرار دارد؛ اگرچه ایده‌هایی که مطرح می‌شوند برای زمانی که «واحد انتقال» یک پروسه در فضای کاربری است یا حتی به صورت توابع کتابخانه‌ای درون پروسهٔ فرستندهٔ جای گرفته، نیز صادق هستند. در شکل ۶-۴۴ ، پروسهٔ فرستندهٔ داده، به هستهٔ سیستم عامل رجوع می‌کند تا عملیات SEND (ارسال) انجام شود. اولین کاری که واحد انتقال انجام می‌دهد بررسی و تشخیص حالت معمولی است یعنی باید: (۱) اتصال در حالت ESTABLISHED قرار داشته باشد، (۲) هیچیک از طرفین سعی در بستن اتصال نکرده باشند، (۳) TPDU کامل و معتبر باشد و (۴) در سمت گیرنده فضای پنجره به اندازهٔ کافی موجود باشد. اگر تمام شرایط فوق برآورده شود به آزمایش بیشتری نیاز نیست و آن TPDU می‌تواند در مسیر **پردازش سریع** قرار بگیرد. طبعاً اکثر بسته‌های TPDU از همین مسیر می‌گذرند.

در حالت عادی سرآیند TPDUهای متوالی و حاوی داده، تقریباً مشابه هم هستند.[1] برای بهره‌برداری مفید از این ویژگی، «نمونهٔ سرآیند» (Header Prototype) درون واحد انتقال کپی می‌شود. در ابتدای مسیر پردازش سریع، سرآیند TPDU با حداکثر سرعت و کلمه به کلمه درون بافر جدید کپی می‌گردد. فیلدهایی که از یک TPDU به TPDU دیگر تغییر می‌کنند درون بافر رونویسی می‌شوند. عموماً مقدار فیلدهایی را که در هر TPDU

---

۱. از لحاظ مقدار «تقریباً» مشابهند وگرنه از لحاظ ساختار سرآیند یقیناً مثل هم هستند. -م



شکل ۶-۴۴. مسیر پردازش سریع بسته‌ها از فرستنده به گیرنده با خطّ تیره‌تر مشخص شده است. مراحل پردازش نیز بصورت خاکستری نشان داده شده است.

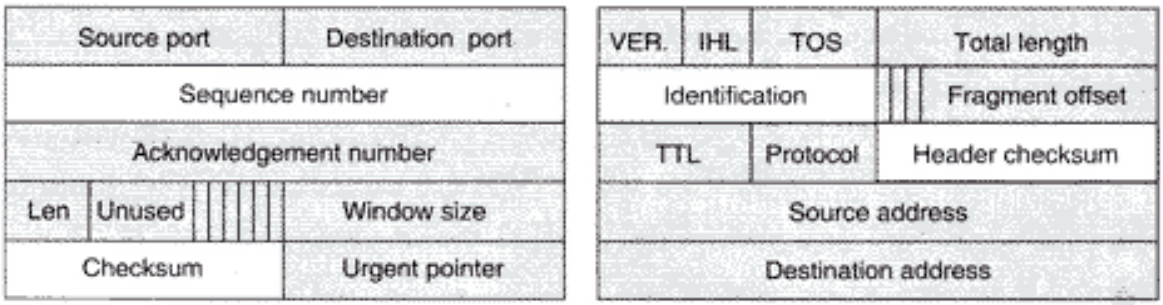
تغییر می‌کنند (همانند فیلد شمارهٔ ترتیب و شمارهٔ Ack)، به سادگی می‌توان از «متغیرهای حالت»، موجود در واحد انتقال بدست آورد. سپس یک اشاره‌گر به سرآیند کامل TPDU، به همراه اشاره‌گر دومی به داده‌های کاربر، تحویل لایهٔ شبکه می‌شود. در لایهٔ شبکه نیز استراتژی مشابهی دنبال می‌شود. (استراتژی لایهٔ شبکه در شکل ۶-۴۴ نشان داده نشده است.) در آخر، لایهٔ شبکه بستهٔ حاصله را جهت انتقال به لایهٔ پیوند داده تحویل می‌دهد.[1]

برای آن که ببینیم این روال در عمل چگونه کار می‌کند، TCP/IP را در نظر می‌گیریم. در شکل ۶-۴۵-الف سرآیند TCP نشان داده شده است. فیلدهایی که در TPDUهای متوالی مشابه هستند (یعنی مقدار آنها تغییر نمی‌کند) به صورت خاکستری نشان داده شده‌اند. تمام کاری که «واحد انتقال» انجام می‌دهد آن است که (۱) این پنج کلمه را از درون «نمونهٔ سرآیند» (Header Prototype) به بافر خروجی منتقل کند؛ (۲) شمارهٔ ترتیب بعدی را در محل مربوطه قرار بدهد (از طریق انتقال یک کلمه در حافظه)، (۳) کد جمع کنترلی (Checksum) را محاسبه کند و (۴) به شمارهٔ ترتیب در حافظه اضافه نماید. سپس سرآیند تنظیم شده و داده‌ها تحویل پروسیجر IP می‌شود تا بطور طبیعی ارسال گردد. نرم‌افزار IP نیز «نمونهٔ سرآیند» پنج کلمه‌ای خود را (شکل ۶-۴۵-ب) به درون بافر کپی می‌کند، فیلد Identification را مقداردهی و کد جمع کنترلی آن را محاسبه می‌نماید. اکنون بسته، آماده ارسال است.

حال ببینیم، مسیر پردازش سریع در طرف گیرنده از شکل ۶-۴۴ چگونه کار می‌کند. در مرحلهٔ ۱ ابتدا «**رکورد اتصال**» مربوط به TPDU ورودی پیدا می‌شود. در نرم‌افزار TCP، رکورد اتصال می‌تواند در یک «جدول درهم‌سازی» (Hash Table) ذخیره شده و کلید آن برحسب تابعی ساده از آدرس IP و دو شمارهٔ پورت محاسبه گردد. به محض پیدا شدن موقعیت رکوردِ متناظر با بستهٔ ورودی، ابتدا آدرسهای IP و شماره‌های پورت آن با این رکورد مقایسه می‌شود تا صحت رکورد پیدا شده تأیید گردد.

---

۱. اگر بخواهیم عملکرد «مسیر پردازش سریع» را به زبان ساده بیان کنیم بدین نحو عمل می‌کند که اگر TPDU از نوع معمولی و حاوی داده باشد، برای آن از قبل یک سرآیند ایجاد می‌شود که خوشبختانه برای هر TPDU جدید فقط تعداد کمی از فیلدهای آن تغییر می‌کند. بلافاصله پس از تنظیم سرآیند هر TPDU، اشاره‌گر محل شروع این سرآیند و اشاره‌گر محل شروع داده‌های کاربر، تحویل لایهٔ شبکه می‌شود. بنابراین بجای انتقال و کپی یک بستهٔ کامل، اشاره‌گرها بین دو لایه ردّ و بدل می‌شوند. ـم



شکل ۶-۴۵. (الف) سرآیند TCP (ب) سرآیند IP. در هر دو سرآیند، فیلدهای خاکستری بدون هیچ تغییری از «نمونهٔ سرآیند» (Header Prototype) موجود گرفته می‌شود.

یک بهینه‌سازی که می‌تواند سرعت جستجوی «رکورد اتصال» را افزایش بدهد آن است که همیشه اشاره‌گر آخرین رکوردی که از آن استفاده شده، نگهداری گردد و جستجو از آن نقطه آغاز شود. کلارک و همکارانش (Clark et al. 1989) این بهینه سازی را آزمایش و مشاهده کردند که احتمال موفقیت سریع در جستجو به بیش از ۹۰ درصد افزایش می‌یابد. روشهای جستجوی دیگر در مرجع (McKenney and Dove; 1992) تشریح شده است.

سپس TPDU بررسی می‌شود تا مشخص گردد که آیا از نوع معمولی است یعنی: اتصال در حالت ESTABLISHED قرار داشته باشد، هیچیک از طرفین تقاضای خاتمهٔ اتصال نداده باشند، TPDU یک بستهٔ کامل و معتبر باشد، هیچیک از بیتهای پرچم (Flag Bits) در TPDU، تنظیم نشده باشد و شمارهٔ ترتیب TPDU دریافتی همان شمارهٔ مورد نظر باشد. این بررسی‌ها مستلزم اجرای چندین دستورالعمل ماشین است. اگر این شرایط برآورده شود. یک «پروسیجر پردازش سریع» برای آن TPDU فراخوانی می‌شود.

«مسیر پردازش سریع» رکورد اتصال را بهنگام‌سازی کرده و داده‌ها را برای پروسهٔ کاربر کپی می‌نماید. در حین کپی، کد جمع کنترلی (checksum) داده‌ها را محاسبه می‌کند تا نیاز به یک گذر اضافی برای این عمل نباشد. اگر کد جمع کنترلی صحیح بود، رکورد اتصال بهنگام سازی شده و پیغام Ack بازگردانده می‌شود. الگوی کلی تست سریع سرآیند برای تشخیص آن که آیا سرآیند TPDU، سرآیند مورد انتظار هست یا خیر توسط پروسیجری به نام «**پیشگویی سرآیند**» (Header Prediction) انجام می‌شود و در بسیاری از پیاده‌سازیهای عملی TCP از آن استفاده شده است. هر گاه این بهینه‌سازی و بهینه‌سازیهای دیگری که در این فصل تشریح کردیم در کنار هم مورد استفاده قرار بگیرد می‌توان به ۹۰ درصد سرعت کپی داده‌ها از حافظه به حافظه رسید. (با فرض آن که شبکه به قدر کافی سریع است.)

دو مورد دیگر که به کمک آنها می‌توان بهبود چشمگیری در کارآیی بوجود آورد مدیریت بافر و مدیریت تایمرهاست. موضوع مرتبط با مدیریت بافر، اجتناب از کپی‌برداریهای بی‌مورد است. به این موضوع در بالا اشاره کردیم. مدیریت تایمر اهمیت بیشتری دارد چرا که تقریباً اغلب آنها (اگر بدرستی تنظیم شده باشند) منقضی نمی‌شوند: آنها بدین منظور تنظیم و راه‌اندازی می‌شوند که اگر یک TPDU از بین رفت از نو ارسال شود در حالی که اغلب TPDUها به درستی تحویل و پیغام اعلام وصول آنها نیز به سلامت بر می‌گردند. بنابراین مدیریت بهینهٔ تایمرها بسیار اهمیت دارد.

یک ساختار رایج برای مدیریت تایمرها آن است که از یک «لیست پیوندی» (Linked List) متشکل از

رخدادهای تایمر که برحسب زمان انقضای آنها مرتب شده، استفاده گردد.[1] اولین عنصر این لیست پیوندی حاوی یک شمارنده است که تعیین می‌کند تا انقضای مهلت آن، چند تیک ساعت باقی مانده است. هریک از عناصر متوالی این لیست پیوندی شمارنده‌ای دارند که مشخص می‌کند در مقایسه با عنصر قبلی در لیست، چند تیک ساعت بعدتر منقضی می‌شوند. بنابراین اگر سه تایمر باید به ترتیب پس از ۳، ۱۰ و ۱۲ تیک منقضی شوند مقادیر شمارنده‌های آنها در لیست پیوندی به ترتیب ۳ و ۷ و ۲ خواهد بود.

در هر تیک ساعت، شمارندهٔ اولین عنصر لیست پیوندی یک واحد کاهش می‌یابد. هر گاه این شمارنده به صفر برسد، رخداد مربوطه پردازش شده و با حذف آن از لیست، عنصر بعدی در جلوی این لیست قرار می‌گیرد. بدون آن که نیازی به تغییر در مقدار شمارندهٔ این عنصر باشد. در این روش، حذف و اضافهٔ یک تایمر به لیست پیوندی، عملیاتی پر هزینه است و زمان انجام این دو عمل متناسب با طول لیست پیوندی می‌باشد.

اگر مقدار حداکثر تایمرها محدود و از قبل معلوم باشد می‌توان از روش کارآمدتری استفاده کرد. در این روش از آرایه‌ای به نام «**چرخ زمان‌سنجی**» بهره گرفته می‌شود. این آرایه در شکل ۶-۴۶ نشان داده شده است. هر «**برش**» (Slot) متناظر با یک تیک ساعت است. در این شکل زمان فعلی، T=4 فرض شده است. تایمرها به گونه‌ای برنامه‌ریزی شده‌اند که نسبت به زمان فعلی، پس از ۳، ۱۰ و ۱۲ تیک ساعت منقضی شوند. اگر در همین لحظه به تایمری احتیاج شد که قرار است پس از ۷ تیک ساعت منقضی شود، یک درایه (Entry) برای آن در برش ۱۱ (Solt 11) ایجاد می‌شود. به روش مشابه اگر نیاز شد که تایمر T+1 حذف شود، لیستی که شروع آن در برش ۱۴ مشخص شده جستجو و درایهٔ مورد نظر حذف می‌شود. دقت کنید که آرایهٔ شکل ۶-۴۶ نمی‌تواند به غیر از تایمرهایی را که زیر T+15 هستند ایجاد کند. [به عبارتی حداکثر زمان قابل تنظیم در تایمرها معادل ۱۵ تیک ثانیه است.]

شکل ۶-۴۶. یک «چرخ زمان‌سنجی» (Timing Wheel).

۱. یعنی هر یک از عناصر این لیست پیوندی مشخص می‌کنند که چقدر بعد چه اتفاقی باید بیفتد. -م

وقتی ساعت تیک می‌زند، اشاره‌گر زمان فعلی به اندازهٔ یک برش در آرایه جلو می‌رود. (به صورت چرخشی) اگر درایه‌ای که اکنون به آن اشاره می‌شود غیرصفر باشد، تمام تایمرهای تعریف شده در آن پردازش می‌شوند.[1] گونه‌های بی‌شماری از روش فوق در مرجع (Varghese, Lauck, 1987) بحث شده است.

۶-۶-۵ پروتکلهایی برای شبکه‌های گیگابیتی

شبکه‌های گیگابیتی در ابتدای دههٔ نود در صحنه ظاهر شدند. عموم افراد سعی کردند که از همان پروتکلهای قدیمی بر روی شبکهٔ جدید استفاده کنند ولی چیزی نگذشت که مشکلات خود را نشان دادند. در این بخش به برخی از این مشکلات و راهکارهایی که پروتکلهای جدید برای حرکت به سمت شبکه‌های سریعتر برگزیده‌اند، خواهیم پرداخت.

اولین مشکل آن است که اکثر پروتکلهای فعلی، از شماره ترتیب ۳۲ بیتی برای بسته‌ها، استفاده کرده‌اند. وقتی اینترنت شروع به کار کرد، خطوط مابین مسیریابها، غالباً خطوط اجاره‌ای 56Kbps بودند و اگر یک ماشین با تمام سرعت اقدام به ارسال می‌کرد بیش از یک هفته طول می‌کشید تا شمارهٔ ترتیب ۳۲ بیتی به مقدار قبلی آن برگردد. برای طراحان TCP، عدد 2^{32} تقریب بسیار خوبی از بی‌نهایت تلقی می‌شد زیرا خطر آن که بسته‌ای برای یک هفته در زیرشبکه سرگردان باشد و سپس به صورت تکراری دریافت شود عملاً وجود نداشت. در شبکهٔ اترنت 10Mbps، زمان تکرار اعداد ۳۲ بیتی به ۵۷ دقیقه کاهش یافت ولی هنوز قابل تحمل و مدیریت‌پذیر بود. در اترنت یک گیگابیت بر ثانیه، زمان تکرار اعداد به ۳۴ ثانیه رسید که متأسفانه کمتر از طول عمر حداکثر هر بسته یعنی ۱۲۰ ثانیه است. در اینجا 2^{32} تقریب خوبی از مقدار بی‌نهایت نیست و ممکن است در حالی که هنوز بسته‌های قدیمی در زیرشبکه موجودند بسته‌هایی با شمارهٔ تکراری تولید شده و گیرنده احتمالاً به اشتباه بیفتد. اگرچه در RFC 1323 راهی موقت برای حل این مشکل پیشنهاد شده است.

مشکل از آنجا ناشی می‌شود که طراحان پروتکل به سادگی فرض کرده‌اند زمانی که طول می‌کشد تا از کل فضای ۳۲ بیتی شماره ترتیب استفاده شود از حداکثر طول عمر بسته بیشتر است. در نتیجه (با این فرض) لازم نبود کسی در خصوص وجود بسته‌هایی با شماره‌های تکراری (که در اثر برگشت مقادیر ۳۲ بیتی به مقدار قبلی پیش می‌آید) نگران باشد. در سرعتهای گیگابیتی این فرض نانوشته با شکست مواجه می‌شود.

مشکل دوم آن است که سرعت مخابرهٔ داده‌ها از سرعت پردازش کامپیوترها پیشی گرفته است. (قابل توجه مهندسین کامپیوتر: به میدان رفته و مهندسین مخابرات را شکست بدهید!! ما روی شما حساب می‌کنیم!) در اوائل دههٔ هفتاد میلادی، شبکهٔ ARPANET از خطوط 56Kbps و کامپیوترهایی با سرعتی حدود 1 MIPS بهره می‌گرفت. بسته‌های اطلاعاتی نیز ۱۰۰۸ بیت بودند و بدین ترتیب ARPANET قادر به تحویل حدود ۵۶ بسته در هر ثانیه بود. هر ماشین میزبان در زمانی حدود ۱۸ میلی‌ثانیه که به ازای هر بسته مهلت پردازش داشت، می‌توانست ۱۸۰۰۰ دستورالعمل ماشین را اجرا نماید. البته اختصاص تمام این دستورالعملها به پردازش بسته، کل زمان CPU را مصرف می‌کند ولیکن اگر فقط ۹۰۰۰ دستورالعمل ماشین به پردازش هر بسته اختصاص داده شود، نیمی از توان CPU برای کارهای اصلی باقی می‌ماند.

این اعداد و ارقام را با کامپیوترهای 1000 MIPS که بسته‌های ۱۵۰۰ بایتی را بر روی خطوط گیگابیتی منتقل

---

۱. به زبان ساده، آرایهٔ فوق را در یک دایره با ۱۶ قطاع تجسم کنید که عقربه‌ای روی آن حرکت می‌کند و در هر لحظه در یکی از این قطاعها قرار می‌گیرد. اگر به فرض نیاز به تایمری داشتید که باید ۵ تیک ثانیه بعد منقضی شود نسبت به مکان فعلی عقربه، ۵ قطاع جلوتر، اشاره‌گر آن را قرار می‌دهید. هر بار که عقربه یک قطاع جلو می‌رود تمام کارهایی که اشاره‌گرشان درون آن قطاع قرار دارد، انجام می‌شود. -م

می‌کنند، مقایسه نمایید. در چنین شبکه‌ای نرخ ارسال بیش از ۸۰۰۰۰ بسته در ثانیه است و طبعاً اگر بخواهیم نیمی از توان پردازشی CPU را به بقیهٔ برنامه‌های کاربردی اختصاص بدهیم، بایستی پردازش هر بسته در ۶/۲۵ میکروثانیه تکمیل شود. یک کامپیوتر 1000MIPS در ۶/۲۵ میکروثانیه قادر به اجرای ۶۲۵۰ دستورالعمل است یعنی حدود یک سوم تعداد دستورالعملهایی که ماشینهای ARPANET قادر به اجرای آنها برای هر بسته بودند. مضاف بر این هر دستورالعمل در ماشینهای مدرن RISC ، نسبت به دستورالعمل ماشینهای قدیمی‌تر CISC کار کمتری انجام می‌دهد لذا مشکل حادّتر از آن چیزی است که به نظر می‌رسد. نتیجه‌گیری آن است که برای پردازشهایی که در نرم‌افزار پروتکلها انجام می‌شود زمان کمتری در اختیار است فلذا پروتکلها باید ساده‌تر و سریعتر شوند.

مشکل سوم آن است که پروتکل Go Back n بر روی خطوطی که در آنها حاصلضرب پهنای باند در تأخیر بسیار زیاد است، ضعیف عمل می‌کند. به عنوان مثال به یک خط چهار هزار کیلومتری که با سرعت 1-Gbps کار می‌کند، دقت نمایید: زمان تأخیر رفت و برگشت ۴۰ میلی‌ثانیه است و در این زمان فرستنده قادر به ارسال ۵ مگابایت داده می‌باشد؛ طبعاً اگر خطایی گزارش شود مربوط به ۴۰ میلی‌ثانیه قبل بوده است. اگر از پروتکل Go Back n (عقبگرد به اندازهٔ n) استفاده شده باشد، فرستنده نه تنها مجبور است بستهٔ خراب را از نو بفرستد بلکه باید ۵ مگابایت بسته‌های بعد از آن را نیز مجدداً ارسال نماید. به وضوح استفاده از این روش، اتلاف بسیار زیادی در منابع شبکه خواهد بود.

مشکل چهارم آن است که خطوط گیگابیتی تفاوت بنیانی با خطوط مگابیتی دارند: این تفاوت از آنجاست که خطوط طولانی گیگابیتی نسبت به «تأخیر» محدودیت دارند در حالی که دیگری محدودیت «پهنای باند» دارد. در شکل ۶-۴۷ زمانی که طول می‌کشد یک فایل یک **مگابیتی** از طریق خطی ۴۰۰۰ کیلومتری با نرخهای متفاوت ارسال شود، نشان داده شده است. تا سرعت 1-Mbps، زمان انتقال متأثر از نرخ ارسال بیتهاست. در سرعت 1-Gbps، تأخیر ۴۰ میلی‌ثانیه‌ای رفت‌وبرگشت بر زمان ۱ میلی‌ثانیه‌ای انتقال بیتها بر روی فیبرنوری غالب می‌شود. از اینجا به بعد افزایش پهنای باند تأثیر چندانی در تأخیر انتقال ندارد.

شکل ۶-۴۷ نکات مأیوس‌کننده‌ای برای پروتکلهای شبکه دارد. این شکل بیانگر آن است که پروتکلهای «توقف و انتظار» (Stop & Wait) همانند RPC از لحاظ کارآیی محدودیت ذاتی دارند. این محدودیت به خاطر سرعت نور تحمیل می‌شود و هیچگونه پیشرفت تکنولوژیک در فیزیک نور نمی‌تواند بر این محدودیت فائق آید.

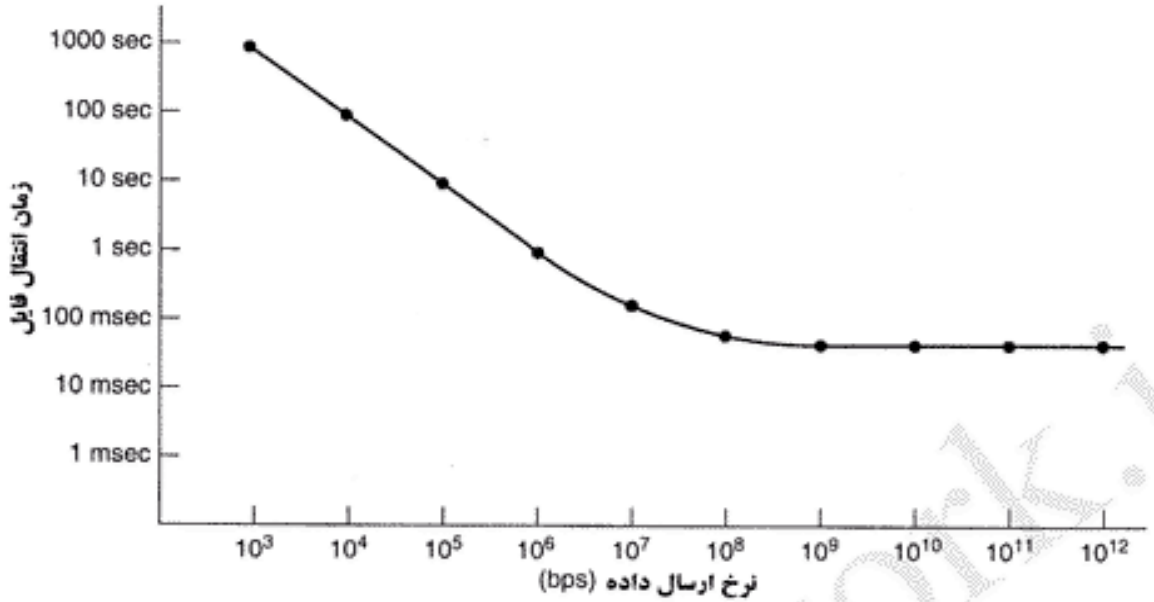
مشکل پنجم که اشاره به آن خالی از لطف نیست ریشه در مسائل تکنولوژیک، پروتکل یا مسائلی بدین شکل ندارد بلکه ناشی از کاربردهای جدید شبکه است. در یک بیان ساده در بسیاری از کاربردهای شبکهٔ گیگابیتی (مثل کاربردهای چند رسانه‌ای) «واریانس زمان دریافت بسته‌ها»، اهمیتی همسنگ با «تأخیر متوسط دریافت» آنها دارد. تحویل آهسته ولی یکنواخت بسته‌ها ارجحتر از تحویل سریع ولی ناگهانی آنهاست.

پس از معرفی مشکلات اجازه بدهید به روشهای حل و فصل آنها بپردازیم. ابتدا تذکراتی کلی ارائه می‌کنیم و سپس به مکانیزمهای پروتکل، ساختار بسته‌ها و نرم‌افزار پروتکل نگاهی خواهیم انداخت.

یک اصل اساسی که تمام طراحان شبکه باید در نظر داشته باشند آن است که:

«طراحی باید مبتنی بر سرعت بالا باشد نه برای بهینه‌سازی پهنای باند»

پروتکلهای قدیمی‌تر اغلب بدان منظور طراحی شده‌اند که تعداد بیتهای ارسالی بر روی کانال به حداقل برسد و بدین منظور از فیلدهای کوچک و ادغام چند فیلد در یک بایت یا کلمه بهره گرفته‌اند. امروزه پهنای باند فراوانی در اختیار است و میزان پردازش در پروتکل به یک مشکل تبدیل شده است، فلذا پروتکلها باید برای به حداقل رساندن حجم پردازش لازم، طراحی شوند. طراحان IPv6 به روشنی این اصل اساسی را درک کرده بودند.



شکل ۶-۴۷. زمان انتقال و تصدیق وصول یک فایل یک مگابیتی از طریق یک خط ۴۰۰۰ کیلومتری.

یک راهکار اغواکننده برای رسیدن به سرعت بالا، ساختن کارتهای واسط شبکه به کمک سخت‌افزار سریع است. [یعنی بخشی از پروتکلهای نرم‌افزاری توسط سخت‌افزار شبکه پیاده‌سازی شود.] مشکل در پیش گرفتن این استراتژی آن است که بدون داشتن یک پروتکل بسیار ساده، کارت سخت‌افزاری به خودی خود تبدیل به یک برد اضافی با یک CPU مستقل و یک برنامهٔ خاص می‌شود. برای آنکه «کمک‌پردازندهٔ شبکه» ارزانتر از پردازندهٔ اصلی تمام شود، اغلب از CPU کُندتر استفاده می‌شود. در نتیجه اگر خود پروتکل ساده نباشد، CPU سریع باید آنقدر منتظر بماند تا CPU دیگر (CPU کُندتر) کار خود را انجام بدهد. این ایده که CPU سریع در حین انتظار به کارهای دیگری بپردازد بیشتر به یک افسانه شبیه است و مشکلات خاص خود را دارد. مضاف بر این وقتی دو CPU همه‌منظوره، با یکدیگر در ارتباط باشند شرایط رقابتی (Race) رخ می‌دهد و به پروتکلهای پیچیده‌ای برای هماهنگ کردن آنها با یکدیگر نیاز خواهد بود. عموماً بهترین راهکار، ساده‌تر کردن پروتکل و محول نمودن کار به CPU اصلی است.

حال ببینیم مسئله فیدبک [یعنی اعلام طول پنجره یا هر مورد مشابه توسط سمت مقابل] در پروتکلهای بسیار سریع چگونه حل و فصل می‌شود. به دلیل وجود حلقه با تأخیر (نسبتاً) بالا، حتی‌الامکان باید از ایجاد فیدبک اجتناب شود زیرا زمان زیادی طول می‌کشد تا سیگنال گیرنده به فرستنده برگردد. مثالی از فیدبک، اعمال نظارت و کنترل نرخ ارسال با استفاده از پروتکل پنجرهٔ لغزان می‌باشد. برای رهایی از تأخیر زیادی که در اعلام طول پنجرهٔ گیرنده به فرستنده همیشه وجود دارد، در شبکه‌های سریع بهتر است از یک پروتکل مبتنی بر نرخ پایه (و بدون فیدبک) بهره گرفته شود. در چنین پروتکلی فرستنده می‌تواند به هر اندازه که تمایل دارد داده بفرستد ولی حق ندارد با نرخی بیشتر از مقداری که قبلاً توافق کرده‌اند، ارسال نماید.

مثال دیگری از فیدبک، الگوریتم «شروع آهسته ژاکوبسن» است. این الگوریتم چندین آزمایش انجام می‌دهد تا مشخص کند که شبکه از عهدهٔ چه مقدار داده بر می‌آید. [بخش ۶-۵-۹] در شبکه‌های بسیار سریع، انجام چند آزمایش (و حتی یکی دو آزمایش) برای ارزیابی چگونگی پاسخ شبکه، بخش بسیار بزرگی از پهنای باند را تلف می‌کند. الگوی کارآمدتر آن است که در همان ابتدای برقراری اتصال، فرستنده، گیرنده و شبکه همگی منابع مورد نیاز را رزرو نمایند. رزرو کردن منابع از قبل، این حسن بزرگ را دارد که ساده‌تر می‌توان مقدار لرزش (Jitter) را

کاهش داد. کوتاه سخن آن که حرکت به سوی شبکه‌های بسیار سریع، طراحی پروتکلها را اجباراً به طرف اتصال‌گرا شدن یا چیزی شبیه به آن سوق می‌دهد. البته اگر در آینده، پهنای باند آنقدر فراوان و کم اهمیت شود که کسی نگران هدر رفتن مقدار قابل توجهی از آن نباشد، اصول طراحی نیز بسیار متفاوت خواهد بود.

ساختار و قالب هر بسته در شبکه‌های گیگابیتی حائز اهمیت است. در سرآیند هر بسته حتی‌الامکان باید تعداد بسیار کمی فیلد تعریف شده باشد تا زمان پردازش آن کاهش یابد. در ضمن این فیلدها باید بقدر کافی بزرگ باشند تا کار خود را بدون محدودیت و نیاز به پردازشهای اضافی انجام بدهند. همچنین هر فیلد باید به صورت کلمه (یعنی ۱ بایتی یا ۲ و ۴ و ۸ و ۱۶ بایتی) تعریف شوند تا پردازش آنها بسیار ساده باشد. منظورمان از فیلد «به قدر کافی بزرگ» آن است که مشکلاتی نظیر برگشت شمارهٔ ترتیب به مقادیر تکراری یا ناتوانی گیرنده از اعلام اندازهٔ بزرگ پنجرهٔ خود و مسائلی از این قبیل پدید نیاید.

بعلاوه بایستی برای بخش سرآیند و بخش داده‌ها در هر بسته ، دو کُد کشف خطای مجزا و مستقل محاسبه و درج شود، به دو دلیل: اول آن که در صورت عدم نیاز به نظارت بر خطای داده‌ها [مثلاً برای داده‌های صدا و تصویر]، بتوان فقط سرآیند بسته را از لحاظ صحت مقادیر بررسی کرد. دوم آن که بتوان قبل از کپی کردن داده‌ها در فضای پروسهٔ کاربر از صحت سرآیند مطمئن شد. مطلوب آن است که کنترل صحت بخش داده در خلال کپی کردن آنها در فضای پروسهٔ کاربر انجام شود ولیکن اگر سرآیند بسته صحیح نباشد ممکن است داده‌ها به اشتباه برای پروسهٔ دیگری کپی شوند. برای اجتناب از کپی اشتباهی داده‌ها، داشتن دو کد کشف خطای مستقل الزامی است. در این صورت می‌توان بررسی صحت بخش داده را به زمان کپی آن موکول کرد.

طول حداکثر داده‌ها در هر بسته باید بزرگ باشد تا حتی در مواجهه با تأخیر زیاد، کارآیی عملیات بالا باشد. هر چه طول داده‌ها بیشتر باشد درصدی از پهنای باند که صرف سرآیند هر بسته می‌شود کاهش خواهد یافت. بسته‌های ۱۵۰۰ بایتی برای شبکه‌های گیگابیتی بسیار کوچکند.

ویژگی ارزشمند دیگر آن است که فرستنده بتواند به همراه تقاضای برقراری اتصال، مقداری داده نیز بفرستد. بدین ترتیب در زمان رفت و برگشت صرفه‌جویی می‌شود.

در آخر، چند کلمه صحبت در خصوص نرم‌افزار پروتکل خالی از لطف نیست. اندیشهٔ مثبت بر موفقیت‌آمیز بودن عمل متمرکز می‌شود در حالی که بسیاری از پروتکلهای قدیمی بیشتر بر این محور تکیه دارند که اگر خطایی رخ داد (مثلاً بسته‌ای در بین راه از بین رفت) چه باید کرد. برای آن که اجرای پروتکلها سریع شود، طراح باید هدف خود را بر آن بگذارد که وقتی همه چیز به خوبی پیش می‌رود به حداقل پردازش نیاز باشد. سپس بدان بپردازد که چگونه می‌توان در هنگام بروز خطا حجم پردازشها را به حداقل رساند. [اولویت با حداقل بودن پردازش در شرایط عادی است چرا که شرایط غیرعادی به ندرت پیش می‌آید. -م]

مسئلهٔ دیگر در طراحی نرم‌افزار پروتکلها به حداقل رساندن زمان کپی‌برداری [انتقال داده‌ها در حافظه] است. همانگونه که قبلاً دیدیم، کپی‌برداری از داده‌ها عامل عمدهٔ تحمیل سربار است. آرمانی آنست که سخت‌افزار، بستهٔ ورودی را در قالب یک بلوک یکپارچه و به صورت یکجا به حافظه منتقل نماید. سپس نرم‌افزار باید این بسته را تنها با یک عمل کپی بلوکی به بافر کاربر منتقل کند. براساس چگونگی عملکرد حافظهٔ نهان CPU ممکن است اجتناب از حلقهٔ تکرار در برنامه‌نویسی به منظور کپی داده‌ها، مطلوبتر باشد. به عبارت دیگر برای کپی کردن ۱۰۲۴ بایت داده، بهتر آن است که به جای حلقه از ۱۰۲۴ دستورالعمل MOVE پشت سرهم (یا ۱۰۲۴ جفت دستورالعمل LOAD و STORE) استفاده شود! به لحاظ اهمیت حیاتی، زیربرنامهٔ کپی بایستی با کدهای اسمبلی نوشته شود مگر آن که بتوان به طریقی کامپایلر را به منظور تولید کد بهینه و سریع تنظیم کرد.

۶-۷ خلاصه

لایهٔ انتقال کلید آشنایی با پروتکلهای لایه‌ای است. این لایه خدمات متنوعی را عرضه می‌کند ولی مهمترین آنها ایجاد یک استریم مطمئن، انتها به انتها و اتصال‌گرا بین گیرنده و فرستنده به منظور انتقال دنباله‌ای از بایتهاست. دسترسی به خدمات این لایه از طریق یکسری توابع اولیه و پایه صورت می‌گیرد که این توابع برقراری اتصال، استفاده از آن (جهت ارسال و دریافت) و خاتمهٔ اتصال را امکان‌پذیر کرده‌اند. یکی از واسطهای رایج برای لایهٔ انتقال با نام «سوکتهای برکلی» (Berkeley Socket) ارائه شده است.

پروتکلهای لایهٔ انتقال باید بتوانند بر اتصالاتی که در یک شبکهٔ نامطمئن ایجاد می‌شوند مدیریت کنند. برقراری اتصال از آن جهت پیچیده است که امکان دارد بسته‌های تکراری و معلق در زیرشبکه، در لحظه‌ای به گیرنده برسند که او را به اشتباه بیندازند. برای حل و فصل این مشکل، باید برای برقراری یک اتصال از روش دست‌تکانی سه‌مرحله‌ای (3-Way Handshake) استفاده شود. ختم یک اتصال ساده‌تر از برقراری آن است ولیکن به دلیل «مشکل دو سپاه» (Two-Army Problem) چندان هم پیش پا افتاده نیست.

حتی وقتی که لایهٔ شبکه کاملاً مطمئن و بدون خطاست باز هم لایهٔ انتقال کار زیادی باید انجام بدهد. این لایه باید وظیفهٔ ارائه خدمات اولیه (توابع اولیه)، مدیریت اتصالات و تایمرها را بر عهده بگیرد و «اعتبار» تخصیص بدهد.

اینترنت در لایهٔ انتقال دو پروتکل دارد: UDP و TCP. UDP پروتکلی بدون‌اتصال است که در حقیقت همان خصوصیات پروتکل IP را دارد، با این ویژگی اضافی که بتوان بسته‌های IP را که همگی دارای آدرس مشترک هستند، بین پروسه‌های یک ماشین مالتی‌پلکس و دی‌مالتی‌پلکس کرد. از UDP می‌توان برای تعامل بین سرویس‌دهنده و مشتری بهره گرفت (مثلاً به کمک RPC). همچنین می‌توان از UDP برای ساختن پروتکلهای بی‌درنگ همانند RTP استفاده کرد.

پروتکل اصلی لایهٔ انتقال در اینترنت، TCP است. این پروتکل یک استریم از بایتها را (به صورت دو طرفه) بین دو پروسه ایجاد کرده و در اختیار می‌گذارد. به یک واحد اطلاعاتی که توسط TCP تولید می‌شود اصطلاحاً «قطعه» (Segment) گفته می‌شود. هر قطعه دارای یک سرآیند ۲۰ بایتی است. قطعات ارسالی ممکن است توسط مسیریابهای اینترنت قطعه قطعه شوند فلذا ماشین میزبان باید آمادگی بازسازی آنها را داشته باشد. کارهای بسیار زیادی نیز برای بهینه سازی کارآیی TCP صورت گرفته و بدین منظور از الگوریتمهایی نظیر «ناگل»، «کلارک»، «ژاکوبسن»، «کارن» استفاده شده است. لینکهای بی‌سیم پیچیدگیهای متعددی را به TCP تحمیل می‌کنند. «TCP تراکنشی» (Transaction TCP) گونهٔ توسعه یافتهٔ TCP است که تعامل سریع بین سرویس‌دهنده و مشتری و کاهش تعداد بسته‌ها را بر عهده دارد.

کارآیی شبکه عموماً متأثر از سربار پروتکل و پردازش TPDU است و در سرعتهای بالا این وضعیت به مراتب بدتر می‌شود. پروتکلها بایستی به نحوی طراحی شوند که تعداد TPDUها، دفعات تعویض متن (Context Switch) و دفعات کپی‌برداری از TPDU را به حداقل برساند. برای شبکه‌های گیگابیتی، پروتکلهای ساده مورد توجه هستند.

مسائل

۱. در مثالی که در شکل ۶-۲ در خصوص عملکردهای پایهٔ انتقال (Primitives) ارائه کردیم، عمل LISTEN یک نوع فراخوانی متوقف‌کننده (Blocking) است. آیا این رفتار واقعاً لازم بوده است؟ اگر نه، شرح بدهید که چگونه می‌توان از یک تابع غیرمتوقف‌کننده (nonblocking) به جای آن استفاده کرد و این روش چه مزیتی

بر الگویی که در متن تشریح شد ، دارد؟

۲. در مدل شکل ۶-۴ فرض بر آنست که احتمالاً برخی از بسته‌ها در لایهٔ شبکه از بین می‌روند و هر بسته باید بطور مستقل اعلام وصول شود. فرض کنید که لایهٔ شبکه صد در صد مطمئن و بدون خطاست و هیچ بسته‌ای از دست نمی‌رود. در این حالت چه تغییری در شکل ۶-۴ لازم است؟

۳. در هر دو بخش از کُد برنامهٔ شکل ۶-۶ بدین نکته اشاره شده که مقدار SERVER_PORT باید در سرویس‌دهنده و مشتری یکسان باشد. چرا این موضوع اینقدر اهمیت دارد؟

٤. فرض کنید که برای تولید مقدار اولیه برای شمارهٔ ترتیب از روش مبتنی بر ساعت استفاده شده و شمارندهٔ ساعت ۱۵ بیتی است. ساعت در هر ۱۰۰ میلی‌ثانیه یکبار تیک می‌زند و حداکثر طول عمر بسته‌ها ۶۰ ثانیه است. در چه مواقعی به هماهنگ‌سازی دوباره (Resynchronization) نیاز است: (پاسخ خود را برای موارد ذیل محاسبه کنید)

الف) در بدترین حالت

ب) وقتی که برای ارسال داده‌ها در هر دقیقه ۲۴۰ شمارهٔ ترتیب مصرف می‌شود.

۵. چه لزومی دارد که حداکثر طول عمر بسته‌ها یعنی T ، باید آنقدر طولانی باشد تا مطمئن شویم که نه تنها بسته بلکه حتی پیغامهای اعلام وصول آنها (ACK) نیز از بین رفته‌اند؟

۶. تصور کنید که برای برقراری اتصال به جای روش «دست‌تکانی سه‌مرحله‌ای» از «دست‌تکانی دومرحله‌ای» استفاده شده باشد. (به عبارت دیگر به پیام سوم نیازی نباشد.) آیا امکان بروز بن‌بست وجود دارد. یا نشان بدهید که بن‌بستی بوجود نمی‌آید یا مثالی از بن‌بست ارائه بدهید.

۷. مسئله «دو سپاه» را به صورت عمومی «n سپاه» (n-Army) در نظر بگیرید که فقط توافق دو سپاه آبی برای حمله، پیروزی آنها را تضمین می‌کند. آیا پروتکلی وجود دارد که براساس آن سپاه آبی به یقین پیروز میدان شود؟

۸. مشکل جبران از کارافتادگی یک ماشین میزبان و بازگرداندن آن به حالت طبیعی را مدّ نظر قرار بدهید؛ (شکل ۶-۱۸). هرگاه بتوان فاصلهٔ زمانی بین نوشتن داده‌ها در پروسه و ارسال پیغام اعلام وصول (ACK) یا برعکس را کم کرد، دو تا از بهترین استراتژیهای فرستنده و گیرنده که احتمال شکست پروتکل را به حداقل می‌رساند، کدامند؟

۹. آیا در واحد انتقال معرفی شده در شکل ۶-۲۰ امکان بروز بن‌بست وجود دارد؟

۱۰. شخصی که واحد انتقال شکل ۶-۲۰ را پیاده‌سازی کرده است تصمیم گرفته که در پروسیجر ***sleep*** یک شمارنده بگذارد تا در خصوص آرایهٔ conn آمارگیری کند. از جملهٔ این آمارها، تعداد اتصالهایی است که در هر یک از هفت حالت ممکن قرار دارد. (n_i, i=1,2,...,7) پس از نوشتن برنامه‌ای مفصل به زبان فرترن برای تحلیل داده‌های جمع‌آوری شده، برنامه‌نویس ما مشاهده می‌کند که رابطهٔ Σn_i=MAX_CONN همیشه صادق است. آیا روابط این چنین دیگری (که حاصل ثابتی داشته باشند) براساس این هفت «متغیر حالت» وجود دارد؟ [حالات هفتگانهٔ n_i را در شکل ۶-۲۱ مرور کنید. -م]

۱۱. اگر یک کاربر با استفاده از واحد انتقال نشان داده شده در شکل ۶-۲۰ یک پیام با طول صفر بفرستد چه اتفاقی می‌افتد؟

۱۲. تمام رخدادهایی را که ممکن است در واحد انتقال شکل ۶-۲۰ اتفاق بیفتد، در نظر بگیرید. حال بگویید که آیا این رخدادها هنگامی که کاربر در حالت *sending* متوقف مانده است باز هم معتبرند؟

۱۳. مزایا و معایب استفاده از روش مبتنی بر اعتبار را در مقایسه با پروتکلهای پنجرهٔ لغزان تشریح کنید.

۱٤. چرا به وجود UDP نیاز است؟ آیا اگر پروسه‌های کاربری داده‌های خود را در درون بسته‌های IP جاسازی و ارسال کند کافی نیست؟

۱۵. یک پروتکل بسیار سادهٔ لایهٔ کاربرد را در نظر بگیرید که بر روی UDP کار می‌کند و امکان آنرا فراهم آورده تا مشتری بتواند یک فایل را از سرویس‌دهندهٔ راه دور دریافت نماید. آدرس سرویس‌دهنده (یعنی آدرس IP و پورت) کاملاً مشخص است. مشتری ابتدا یک تقاضا حاوی نام فایل درخواستی برای سرویس‌دهنده می‌فرستد. سرویس‌دهنده در پاسخ دنباله‌ای از بسته‌های داده را که هر یک بخشی از فایل درخواستی را حمل می‌کنند، برای مشتری ارسال می‌دارد. برای اطمینان از صحّت داده‌ها و همچنین حفظ ترتیب بسته‌های دریافتی، سرویس‌دهنده و مشتری از پروتکل «توقف و انتظار» (Stop & Wait) بهره گرفته‌اند. صرفنظر از مسئله کارآیی، آیا مشکل دیگری در این پروتکل می‌بینید؟ به دقت در مورد امکان از کار افتادن (crashing) پروسه‌ها فکر کنید.

۱۶. یک برنامهٔ مشتری، از طریق یک فیبرنوری به طول ۱۰۰ کیلومتر و سرعت یک گیگابیت بر ثانیه، تقاضایی ۱۲۸ بایتی (RPC) برای سرویس‌دهنده می‌فرستد. کارآیی خط در خلال این فراخوانی از راه دور چقدر است؟

۱۷. وضعیت توصیف شده در مسئله قبلی را مدّ نظر قرار بدهید. حداقل زمان پاسخ سرویس‌دهنده را برای خطوط 1-Gbps و 1-Mbps محاسبه نمائید. چه نتیجه‌ای می‌توانید بگیرید؟

۱۸. UDP و TCP هر دو برای مشخص کردن پروسهٔ تحویل گیرنده پیام از شمارهٔ پورت استفاده می‌کنند. دو دلیل بیاورید که چرا این دو پروتکل برای مشخص کردن پروسه‌ها، شناسه‌های جدیدی تعریف کرده‌اند (یعنی شمارهٔ پورتها) و از شناسهٔ پروسه (Process ID) که از قبل [در هستهٔ سیستم عامل] وجود دارد استفاده نمی‌کنند؟

۱۹. حداقل اندازهٔ کل یک بستهٔ TCP (شامل سربار IP و TCP، بدون در نظر گرفتن سربار لایهٔ پیوند داده) چقدر است؟

۲۰. فرآیند قطعه‌قطعه کردن دیتاگرام و بازسازی آنها بر عهدهٔ IP است و از دید TCP مخفی می‌ماند. آیا این قضیه بدین معنا تلقی می‌شود که TCP نباید نگران تحویل نامرتب داده‌ها باشد؟

۲۱. برای انتقال صدا با کیفیت CD، از پروتکل RTP استفاده می‌شود و باید در هر ثانیه ۴۴۱۰۰ جفت نمونهٔ ۱۶ بیتی ارسال شود. (هر یک از این جفت نمونه‌ها برای یکی از کانالهای استریو است.) RTP در هر ثانیه باید چند بسته ارسال کند؟

۲۲. آیا می‌توان کُد اجرایی RTP را در کنار UDP درون هستهٔ سیستم عامل قرار دارد؟ پاسخ خود را شرح بدهید.

۲۳. به یک پروسه بر روی ماشین میزبان ۱ شمارهٔ پورت p انتساب داده شده است. همچنین به پروسهٔ دیگری بر روی ماشین ۲، شمارهٔ پورت q منتسب شده است. آیا این امکان وجود دارد که در یک زمان دو یا چند اتصال TCP بین این دو پورت برقرار شود؟

۲٤. در شکل ۶-۲۹ می‌بینیم که به غیر از فیلد ۳۲ بیتی Acknowledgement، در چهارمین کلمه یک بیت دیگر با نام ACK وجود دارد. آیا این بیت واقعاً کاری انجام می‌دهد؟ چرا بله و چرا نه؟

۲۵. حداکثر طول فیلد حمل داده (Payload) در یک قطعهٔ TCP، ۶۵۴۹۵ بایت است. این عدد عجیب از کجا آمده است؟

۲۶. در شکل ۶-۳۳، دو راه وارد شدن به حالت SYN RCVD را تشریح نمایید.

۲۷. اشکال بالقوهٔ «الگوریتم ناگل» (Nagle) را در بکارگیری آن بر روی شبکه‌ای که شدیداً با مشکل ازدحام مواجه شده، تشریح کنید.

۲۸. تأثیر استفاده از «الگوریتم شروع آهسته» (Slow start algorithm) را بر روی خطی بدون ازدحام و با تأخیر رفت و برگشت ده‌میلی‌ثانیه، مد نظر قرار بدهید. پنجرهٔ دریافت 24KB و حداکثر طول هر قطعه 2KB است. چقدر طول می‌کشد تا به اندازهٔ یک پنجرهٔ کامل (یعنی ۲۴KB)، داده ارسال شود؟

۲۹. فرض کنید که «پنجرهٔ ازدحام TCP» به 18KB تنظیم شده باشد و انقضای مهلت تایمر رخ بدهد. اگر چهار ارسال بعدی موفق باشد اندازهٔ پنجره چقدر است؟ (فرض کنید که حداکثر طول قطعه 1KB می‌باشد.)

۳۰. فرض نمایید زمان رفت و برگشت (یعنی RTT)، فعلاً ۳۰میلی‌ثانیه است و سه پیغام ACK بعدی به ترتیب پس از ۲۶، ۳۲ و ۲۴ میلی‌ثانیه دریافت شوند. با فرض $\alpha = 0.9$ در الگوریتم ژاکوبسن، مقدار جدید RTT چقدر است؟

۳۱. یک ماشین مبتنی بر TCP، به اندازهٔ یک پنجرهٔ ۶۵۵۳۵ بایتی بر روی یک کانال 1-Gbps، داده می‌فرستد. کانال فقط در یک جهت ده‌میلی‌ثانیه تأخیر دارد. حداکثر توان خروجی (Throughput) قابل حصول چقدر است؟ کارآیی خط چقدر است؟

۳۲. فرض کنید حداکثر طول عمر بسته‌ها ۱۲۰ ثانیه است. یک ماشین میزبان حداکثر با چه سرعتی می‌تواند بسته‌های TCP با ۱۵۰۰ بایت داده را بفرستد در حالی که خطر تکرار شمارهٔ ترتیب تهدیدکننده نباشد؟ سربار ناشی از سرآیند TCP، IP و اترنت را هم در نظر بگیرید. فرض کنید که فریمهای اترنت را می‌توان پیاپی و بی‌وقفه فرستاد.

۳۳. در یک شبکه اندازهٔ حداکثر هر TPDU، ۱۲۸ بایت و حداکثر طول عمر آن ۳۰ثانیه است. اگر شمارهٔ ترتیب هر TPDU هشت بیتی باشد، حداکثر نرخ ارسال در هر اتصال را محاسبه نمایید.

۳۴. فرض کنید زمان دریافت یک TPDU را اندازه‌گیری می‌کنید. وقتی یک وقفه رخ می‌دهد (وقفهٔ دریافت بسته) شما بلافاصله ساعت سیستم را برحسب میلی‌ثانیه می‌خوانید. وقتی TPDU کاملاً پردازش شد مجدداً اقدام به خواندن ساعت می‌کنید. این کار را یک میلیون بار تکرار کرده‌اید و ۲۷۰۰۰۰ بار زمان را صفر و ۷۳۰۰۰۰ بار زمان را ۱ میلی‌ثانیه بدست آورده‌اید. با این اندازه‌گیریها زمان دریافت یک TPDU چقدر است؟

۳۵. یک CPU دستورالعملهای ماشین را با سرعت 1000-MIPS اجرا می‌کند. داده‌ها را می‌توان به صورت کلمات ۶۴ بیتی کپی کرد و کپی هر کلمه معادل زمان اجرای ۱۰ دستورالعمل طول می‌کشد. اگر هر بستهٔ ورودی نیاز به چهار بار کپی‌برداری داشته باشد، آیا این سیستم قادر است از عهدهٔ یک خط 1-Gbps برآید؟ برای سادگی فرض کنید تمام دستورالعملهای ماشین حتی آنهایی که از حافظه می‌خوانند یا در آن می‌نویسند، همگی با سرعت 1000-MIPS اجرا شوند.

۳۶. برای رهایی از مشکل تکراری شدن شمارهٔ ترتیب بسته‌ها (در حالی که ممکن است بسته‌های قدیمی هنوز در زیرشبکه سرگردان باشند) می‌توان از شماره‌های ۶۴ بیتی استفاده کرد. با این وجود از دیدگاه تئوری می‌توان بکمک فیبرنوری به نرخ 75Tbps (75000 Gbps) رسید. حداکثر طول عمر بسته‌ها چقدر باشد که در شبکه‌های آینده با سرعت 75Tbps شماره‌های ترتیب ۶۴ بیتی تکرار نشوند؟ (فرض کنید که همانند TCP، هر بایت دارای یک شمارهٔ ترتیب است.)

۳۷. یکی از مزیتهای استفاده از RPC بر روی UDP را نسبت به استفاده از «TCP تراکنشی» بیان کنید. یک مزیت T/TCP را نسبت به RPC عنوان نمایید.

۳۸. در شکل ۶-۴۰-الف مشاهده می‌کنید که برای تکمیل یک فراخوانی راه دور (RPC) به مبادله حداقل ۹ بسته نیاز است. آیا وضعیت دیگری وجود دارد که دقیقاً به ۱۰ بسته نیاز باشد؟

۳۹. در بخش ۶-۶-۵ محاسبه کردیم که خطی یک گیگابیتی ۸۰۰۰۰ بسته در هر ثانیه به درون ماشین میزبان منتقل می‌کند. همچنین فرض کردیم که برای پردازش هر بسته فقط ۶۲۵۰ دستورالعمل اجرا گردد و نیم دیگر از توان پردازشی CPU برای کاربردهای دیگر کنار گذاشته شود. این محاسبات بر مبنای بسته‌های ۱۵۰۰ بایتی بود. همین محاسبات را برای بسته‌های ۱۲۸ بایتی (هم اندازهٔ بسته‌های ARPANET) انجام بدهید. در هر دو حالت فرض را بر آن بگذارید که سربار کل در اندازهٔ بسته لحاظ شده است. [یعنی نیازی به اضافه کردن طول سرآیندها به مقادیر ۱۲۸ یا ۱۵۰۰ نیست.]

۴۰. برای شبکه‌ای 1-Gbps که طول کانال آن ۴۰۰۰ کیلومتر است، عامل اصلی محدودیت تأخیر خط است نه پهنای باند. حال یک MAN را در نظر بگیرید که میانگین فاصلهٔ مبداء و مقصد، ۲۰ کیلومتر است. در چه نرخ ارسالی تأخیر رفت و برگشت (ناشی از سرعت نور) معادل با تأخیر انتقال یک بستهٔ یک کیلوبایتی است؟ [تاخیر انتشار ناشی از سرعت نور است درحالیکه تاخیر انتقال متاثر از نرخ ارسال]

۴۱. حاصلضرب پهنای باند در تأخیر را برای شبکه‌های زیر محاسبه کنید: (۱) T1 (با نرخ 1.5Mbps) (۲) اترنت (10Mbps) (۳) T3 (45Mbps) (۴) STS-3 (155 Mbps). زمان رفت و برگشت یعنی RTT را ۱۰۰ میلی‌ثانیه فرض کنید. به خاطر داشته باشید که در سرآیند TCP فقط ۱۶ بیت جهت اعلام طول پنجره رزرو شده است. از محاسبات خود به چه نتیجه‌ای می‌رسید؟

۴۲. حاصلضرب پهنای باند در تأخیر یک کانال ماهواره‌ای همگام با زمین (ماهوارهٔ نوع GEO) چقدر است؟ اگر تمام بسته‌ها با احتساب سربار، ۱۵۰۰ بایتی باشند اندازهٔ فیلد پنجره در هر بسته باید چقدر باشد؟

۴۳. سرویس‌دهندهٔ فایل نشان داده شده در شکل ۶-۶ بسیار ضعیف عمل می‌کند و می‌توان بهبودهایی را در آن ایجاد کرد. اصلاحات زیر را در آن اعمال نمایید:

الف) به برنامهٔ مشتری آرگومان سومی اضافه کنید که محدودهٔ بایتهائی که باید ارسال شوند را مشخص کند.

ب) به برنامهٔ مشتری آرگومان پرچم w- را اضافه کنید تا بتوان فایلی را بر روی سرویس‌دهنده نوشت.

۴۴. برنامهٔ شکل ۶-۲۰ را به نحوی اصلاح کنید که عملیات جبران خطا (Error Recovery) را نیز انجام بدهد. یک بستهٔ نوع جدید به نام reset تعریف کنید که فقط زمانی دریافت می‌شود که طرفین اقدام به برقراری اتصال کرده باشند ولی به هر دلیلی هیچیک از طرفین آن را قطع نکرده باشند. این رخداد که بطور همزمان در طرفین یک اتصال بروز می‌کند بدین معناست تمام بسته‌هایی که قبلاً ارسال شده‌اند یا دریافت گردیده‌اند و یا از بین رفته‌اند ولیکن به هر حال در زیرشبکه نیستند.

۴۵. برنامه‌ای بنویسید که به جای سیستم مبتنی بر اعتبار (credit) که در واحد انتقال شکل ۶-۲۰ از آن استفاده شد، مدیریت بافرها را مبتنی بر پروتکل پنجرهٔ لغزان (برای کنترل جریان) شبیه سازی کند. باید پروسه‌های لایهٔ بالاتر بتوانند اتصالاتی را باز کنند، داده بفرستند و نهایتاً اتصال را ببندند. برای ساده شدن برنامه فرض کنید که کل داده‌ها در یک جهت و از ماشین A به ماشین B جریان دارند. در برنامهٔ خود استراتژیهای مختلف تخصیص بافر را در ماشین B بیازمائید، مثلاً روش تخصیص پیشاپیش بافر برای هر اتصال را با روش

معمولی بافرسازی (یعنی استفاده از یک بافر بزرگ) مقایسه کرده و توان خروجی آن دو مقایسه کنید.

۴۶. یک سیستم گپ زنی (chat) طراحی کنید که امکان گفتگو و محاورهٔ چندین گروه از کاربران را فراهم کند. این سیستم از سه بخش تشکیل شده است: (۱) هماهنگ‌کنندهٔ گفتگو (Chat Coordinator) (۲) سرویس‌دهندهٔ گفتگو (Chat Server) (۳) برنامهٔ مشتری (Chat Client). سیستم هماهنگ‌کنندهٔ گفتگو در آدرس شناخته‌شده‌ای از شبکه قرار گرفته است. سیستم هماهنگ کننده با برنامه‌های مشتری به روش UDP تبادل داده می‌کند و برای هر نشستی که ایجاد می‌شود یک سرویس‌دهندهٔ گفتگو (Chat Server) تنظیم می‌کند. یعنی به ازای هر نشست یک سرویس‌دهندهٔ گفتگو وجود دارد. [یک نشست را مجموعه‌ای از کاربران در نظر بگیرید که حول یک موضوع خاص گفتگو می‌کنند.] از دیگر وظائف سیستم هماهنگ‌کننده، نگهداری یک دایرکتوری از نشستهای موجود است. سرویس‌دهندهٔ گفتگو برای ارتباط با هر مشتری آن نشست، از TCP بهره می‌گیرد. برنامهٔ مشتری نیز به کاربران اجازه می‌دهد که یک نشست ایجاد کنند، به یک نشست موجود بپیوندند یا یک نشست را ترک کنند. کُد برنامهٔ سیستم هماهنگ‌کننده، سرویس‌دهنده و مشتری را طراحی و پیاده‌سازی نمائید.

۷ لایه کاربرد

بعد از به سرانجام رساندن همهٔ مقدمات، اکنون به لایه‌ای رسیده‌ایم که تمام کاربردهای شبکه در آن قرار دارد: لایهٔ کاربرد (application layer) . لایه‌های زیرین لایهٔ کاربرد فقط برای سرویس دادن به این لایه هستند، و هیچ کار واقعی برای کاربران انجام نمی‌دهند. در این فصل با کاربردهای واقعی شبکه آشنا خواهید شد.

با این حال در لایهٔ کاربرد هم به پروتکلهای پشتیبانی‌کننده، پروتکلهایی که بار وظایف را بر گردن می‌گیرند، نیاز داریم. بهمین دلیل، برای شروع یکی از این پروتکلها را بررسی خواهیم کرد. این پروتکل که نام آن DNS است، نامگذاری در اینترنت را بر عهده دارد. پس از آن، سه تا از کاربردهای واقعی شبکه را مورد بررسی قرار خواهیم داد: پُست الکترونیک (ایمیل)، تارنمای جهانی (یا باختصار، وب)، و چندرسانه‌ای.

۱ـ۷ سیستم نام ناحیه – DNS

با اینکه از نظر تئوری برنامه‌ها می‌توانند برای تماس با کامپیوترها، صندوق‌های پستی و منابع دیگر از آدرس شبکهٔ (مثلاً، IP) آنها استفاده کنند، حفظ کردن این قبیل آدرسها برای افراد دشوار است. همچنین اگر آدرس ایمیل «حسن» hasan@128.111.24.41 باشد، و ISP یا سازمان متبوع وی تصمیم بگیرد کامپیوتر سرویس‌دهنده پست الکترونیک خود را به آدرس IP دیگری منتقل کند، آدرس ایمیل «حسن» نیز عوض خواهد شد. بهمین دلیل برای تفکیک نام ماشینها از آدرس آنها، استفاده از نامهای معمولی باب شد. به این ترتیب، آدرس ای. بل «حسن» چیزی شبیه hasan@art.ucsb.edu خواهد شد. با این وجود، کامپیوترها فقط آدرسهای عددی را می‌فهمند، پس باید مکانیزمی برای تبدیل اسامی معمولی به آدرسهای شبکه فراهم کنیم. در این قسمت روش تبدیل نامهای معمولی به آدرس عددی را در اینترنت بررسی خواهیم کرد.

سالها قبل در آرپانت فایلی وجود داشت بنام hosts.txt ، که نام کامپیوترها و آدرس IP آنها در این فایل لیست می‌شد. کامپیوترهای شبکه هر شب این فایل را از جایی که قرار داشت، می‌خواندند و خود را به روز می‌کردند. برای شبکه‌ای با دهها (و یا صدها) کامپیوتر این روش بخوبی کار می‌کرد.

ولی وقتی تعداد کامپیوترهای شبکه از مرز هزاران PC و کامپیوتر بزرگ گذشت، همه دریافتند که این روش دیگر جوابگو نیست. اولین دلیل آن بود که اندازهٔ چنین فایلی بشدت بزرگ می‌شد، ولی از آن مهمتر مشکل نامهای تکراری بود که ضرورت یک مدیریت مرکزی را اجتناب‌ناپذیر می‌کرد (چیزی که بزرگی و بار شبکه آنرا ناممکن می‌کرد). برای غلبه بر این مشکلات بود که DNS (سیستم‌نام‌ناحیه – Domain Name System) اختراع شد.

ایدهٔ اصلی DNS یک روش نامگذاری سلسله مراتبی بر اساس ناحیه‌ها بود، که بصورت یک پایگاه اطلاعاتی

توزیع‌یافته پیاده‌سازی می‌شد. هدف اولیهٔ این سیستم تبدیل نام کامپیوترها و آدرسهای ایمیل (پست الکترونیک) به آدرسهای IP بود، ولی می‌توانست کاربردهای دیگری هم داشته باشد. DNS در RFC 1034 و RFC 1035 تعریف شده است.

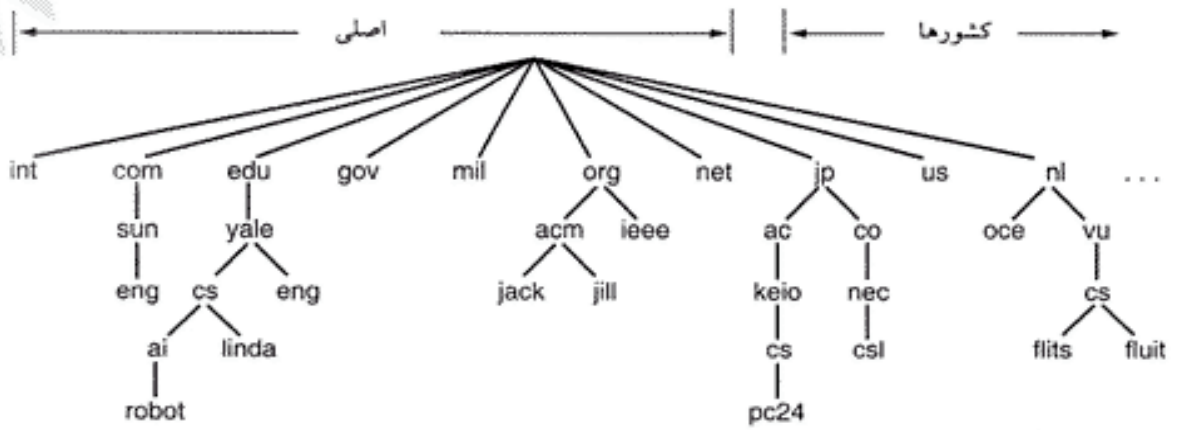
روش کار DNS خیلی خلاصه چنین است: برای تبدیل یک نام به آدرس IP ، برنامه یک تابع کتابخانه‌ای بنام تبدیل‌کننده (resolver) را فراخوانی می‌کند، و نام موردنظر را بصورت پارامتر به آن می‌دهد. تابع *gethostbyname* در شکل ۶-۶ نمونه‌ای از یک تبدیل‌کننده است. تبدیل‌کننده یک بستهٔ UDP به سرویس‌دهنده DNS محلی می‌فرستد، که این DNS آدرس IP معادل نام خواسته شده را یافته و به تبدیل‌کننده برمی‌گرداند، که آن هم بنوبهٔ خود آدرس را به برنامهٔ فراخوانی‌کننده تحویل می‌دهد. برنامه هم پس از بدست آوردن آدرس IP کامپیوتر مقصد، می‌تواند با آن ارتباط TCP برقرار کرده یا بسته‌های UDP به آن بفرستد.

۷-۱-۱ فضای نام DNS

مدیریت مجموعه‌ای بزرگ و دائماً در تغییر از نامها بهیچوجه کار ساده‌ای نیست. در سیستم پُست، مدیریت نامها از طریق اجبار افراد به نوشتن نام کشور، استان (یا ایالت)، شهر، خیابان و شمارهٔ پلاک مقصد انجام می‌شود. با این روش دیگر «پلاک ۷، خیابان آزادی، تهران» هرگز با «پلاک ۷، خیابان آزادی، اصفهان» اشتباه نخواهد شد. DNS هم به همین روش کار می‌کند.

اینترنت به بیش از ۲۰۰ ناحیهٔ سطح‌بالا (top-level domain) (که هر کدام تعداد زیادی کامپیوتر را در برمی‌گیرند) تقسیم شده است. هر ناحیه به چندین زیرناحیه (subdomain) ، و آنها نیز بنوبهٔ خود به زیرناحیه‌های کوچکتر، تقسیم می‌شوند. این سلسله مراتب را می‌توان بصورت یک درخت نمایش داد (شکل ۷-۱ را ببینید). ناحیه‌هایی که زیرناحیه ندارند، برگهای این درخت را تشکیل می‌دهند. هر یک از این برگ‌ها می‌تواند یک کامپیوتر، یک شبکهٔ کوچک (با چند کامپیوتر)، و یا شرکتی بزرگ (با هزاران کامپیوتر) باشد.

ناحیه‌های سطح بالا بر دو گونه‌اند: عمومی و کشورها. ناحیه‌های عمومی اولیه عبارت بودند از: *com* (مخفف commercial ، تجاری)، *edu* (مخفف educational ، مؤسسات آموزشی)، *gov* (مخفف goverment ، ادارات دولتی)، *int* (مخفف international ، مؤسسات بین‌المللی)، *mil* (مخفف military ، نظامی)، *net* (مخفف network provider ، شرکتهای خدمات شبکه و اینترنت)، و *org* (مخفف organization ، مؤسسات غیرانتفاعی). هر کشور نیز دارای یک ناحیهٔ خاص (در ناحیهٔ کشورها) است، که در استاندارد ISO 3166 تعریف شده است.



شکل ۷-۱. بخشی از فضای نام ناحیه در اینترنت.

در نوامبر ۲۰۰۰، ICANN چهار ناحیهٔ سطح بالای جدید را برای مصارف عمومی تصویب کرد: *biz* (مخفف businesses ، مشاغل)، *info* (مخفف information ، شرکتهای اطلاعاتی)، *name* (نام افراد)، و *pro* (مخفف proffession ، صاحبان حِرَف مانند وکلا و پزشکان). علاوه بر آن، سه ناحیهٔ سطح بالای تخصصی نیز معرفی شد، که برخی از صنایع خاص می‌توانند از آنها استفاده کنند؛ این سه ناحیه عبارتند از: *aero* (مخفف aerospace ، صنایع هوا-فضا)، *coop* (مخفف co-operatives ، تعاونی‌ها)، و *museum* (موزه‌ها). به احتمال زیاد در آینده ناحیه‌های سطح بالای دیگری نیز اضافه خواهند شد.

از طرف دیگر هر چه اینترنت بیشتر تجاری می‌شود، بحث و جدل هم بالاتر می‌گیرد. مثلاً همین ناحیهٔ *pro* را در نظر بگیرید. این ناحیه برای صاحبان حِرَف که صلاحیت آنها تأئید شده باشد، در نظر گرفته شده است. اما حرفه‌ای کیست؟ و چه کسی باید صلاحیت‌ها را تأئید کند؟ یک پزشک یا وکیل مسلماً حرفه‌ایست، اما یک عکاس، معلم پیانو، شعبده‌باز، لوله‌کش، آرایشگر، خالکوب، آدمکش حرفه‌ای و یا فاحشه چطور؟ آیا اینها هم حرفه‌اند، و شایستهٔ دریافت ناحیهٔ *pro* ؟ و اگر پاسخ مثبت است، چه کسی صلاحیت داوطلبان را تأئید می‌کند؟

در کل، گرفتن ناحیهٔ سطح-دوم در یک ناحیهٔ سطح بالا ، مانند *name-of-company.com* ، ساده است: تنها کاری که باید کرد مراجعه به پایگاه اطلاعاتی ناحیهٔ سطح بالا (در اینجا *com*) و اطمینان از آزاد بودن نام موردنظر است. اگر مشکلی وجود نداشته باشد، درخواست‌کننده پول کمی بابت هزینهٔ سالیانه پرداخته، و آن نام را بدست می‌آورد. می‌توان به جرأت گفت که، اکنون تمام نامهای بامعنای انگلیسی در ناحیهٔ *com* گرفته شده‌اند (اگر باور ندارید، امتحان کنید!).

نام هر ناحیه بصورت مسیری رو به بالا و به سمت یک ریشه (که نامی ندارد) مشخص می‌شود. اجزای این نام با نقطه(که «دات» تلفظ می‌شود) از هم جدا می‌شوند. برای مثال، نام ناحیهٔ بخش مهندسی شرکت سان میکروسیستمز می‌تواند *eng.sun.com.* باشد (در حالیکه همین نام در سیستم عامل یونیکس بصورت */com/sun/eng* نوشته می‌شود). دقت کنید که با این روش نامگذاری دیگر نام ناحیهٔ بخش مهندسی سان میکروسیستمز با دانشکدهٔ مهندسی دانشگاه ییل (که مثلاً *eng.yale.edu.* است) اشتباه نخواهد شد.

نام ناحیه می‌تواند مطلق (absolute) یا نسبی (relative) باشد. یک نام مطلق همیشه به نقطه ختم می‌شود (مانند *eng.sun.com.*)، در حالیکه نامهای نسبی چنین نیستند. نامهای نسبی بدون توجه به جایی که بکار رفته‌اند، معنی نمی‌دهند. در هر دو حالت، یک نام ناحیه به گرهی خاص در درخت نامها (و تمام گرههای ذیل آن) اشاره می‌کند.

کوچکی یا بزرگی حروف در نام ناحیه بی‌تأثیر است، بعبارت دیگر *edu* ، *Edu* و *EDU* همگی یک معنی می‌دهند. هر جزء از نام ناحیه حداکثر ۶۳ حرف، و کل مسیر نام ناحیه حداکثر ۲۵۵ حرف می‌توانند داشته باشند.

برای قرار دادن یک ناحیه در درخت نامهای ناحیه دو روش وجود دارد. برای مثال، ناحیهٔ *cs.yale.edu* می‌تواند در ذیل شاخهٔ کشور آمریکا (*us*) و بصورت *cs.yale.ct.us* هم ثبت شود. در کل، اغلب سازمانها و شرکتها در آمریکا ترجیح می‌دهند از ناحیه‌های عمومی استفاده کنند، در حالیکه در خارج از آمریکا بیشتر از نام کشور بعنوان ناحیهٔ سطح بالا استفاده می‌شود. هیچ منعی برای ثبت نام ناحیه در شاخه‌های متعدد وجود ندارد، با این حال چنین گرایشی بجز در شرکتهای چندملیتی (مانند شرکت سونی که ناحیه‌های *sony.com* و *sony.nl* را ثبت کرده) وجود ندارد.

هر ناحیه ناحیه‌های ذیل خود را کنترل می‌کند. برای مثال، کشور ژاپن دارای ناحیه‌های *ac.jp* و *co.jp* است، که بترتیب مشابه *edu* و *com* هستند؛ در حالیکه در هلند چنین تقسیم‌بندی وجود ندارد، و تمام ناحیه‌ها ذیل *nl* قرار دارند. برای مثال، ناحیه‌های زیر همگی دانشکده‌های کامپیوتر در کشور مربوطه هستند:

۱. cs.yale.edu (دانشگاه ییل، ایالات متحدهٔ آمریکا)
۲. cs.vu.nl (دانشگاه فریژه، هلند)
۳. cs.keio.ac.jp (دانشگاه کی‌یو، ژاپن)

برای ایجاد یک زیرناحیه، مجوز ناحیهٔ بالاتر مورد نیاز است. برای مثال اگر گروه VLSI در دانشکدهٔ کامپیوتر دانشگاه ییل تأسیس شود، و بخواهد به نام *vlsi.cs.yale.edu* شناخته شود، باید مجوزهای لازم را از مسئول *cs.yale.edu* کسب کند. بهمین ترتیب اگر دانشگاه جدیدی، مثلاً دانشگاه شمالی داکوتای جنوبی، تأسیس شود، و بخواهد ناحیهٔ *unsd.edu* را برای خود ثبت کند، باید هماهنگیهای لازم را با مسئول ناحیهٔ *edu* بعمل آورد. قرار دادن مسئولیت زیرناحیه‌ها بر عهدهٔ ناحیهٔ بالاتر باعث می‌شود تا هیچ دو ناحیه‌ای همنام نشوند. همین که یک ناحیه ایجاد شد (مانند، *unsd.edu*)، دیگر می‌تواند بدون نظارت ناحیه‌های بالاتر به ایجاد زیرناحیه‌های دلخواه خود (مثلاً، *cs.unsd.edu*) بپردازد.

نامگذاری ناحیه‌ها امری سازمانی است نه فیزیکی. برای مثال، دانشکده‌های کامپیوتر و مهندسی برق یک دانشگاه می‌توانند ناحیه‌های کاملاً مستقلی داشته باشند، حتی اگر در یک ساختمان باشند و از شبکهٔ LAN واحدی استفاده کنند. از طرف دیگر، بخشهای مختلف یک دانشکده به یک ناحیه تعلق دارند، حتی اگر از نظر فیزیکی از هم جدا باشند.

۲-۱-۷ رکوردهای منابع

هر ناحیه، خواه ناحیه‌ای سطح بالا یا ناحیه‌ای با یک کامپیوتر، دارای تعدادی رکوردمنابع (resource record) است. برای یک کامپیوتر، متداولترین رکورد منبع آدرس IP آن است، اما انواع دیگری از رکوردهای منابع می‌تواند وجود داشته باشد. وقتی یک تبدیل‌کننده نام ناحیه را به DNS می‌دهد، چیزی که دریافت می‌کند تمام رکوردهای منابع وابسته به آن نام است. بنابراین، اصلی‌ترین وظیفهٔ یک DNS تبدیل نام ناحیه به رکوردهای منابع است.

هر رکورد منبع پنج بخش دارد. با اینکه رکوردهای منابع را می‌توان برای کارایی بهتر بصورت باینری در آورد، ولی در اغلب مواقع این رکوردها بصورت متنی (ASCII) - یک رکورد در هر خط - نگهداری می‌شوند. فرمت یک رکورد منبع مانند زیر است:

Domain_name Time_to_live Class Type Value

که در آن *Domain_name* نام ناحیه‌ایست که این رکورد متعلق به آن است. معمولاً هر ناحیه تعداد زیادی رکورد دارد، و در هر پایگاه داده اطلاعات چندین ناحیه نگهداری می‌شود. در نتیجه این فیلد کلید اصلی جستجو در پایگاه دادهٔ DNS است. (ترتیب قرار گرفتن رکوردها در پایگاه داده اهمیتی ندارد.)

فیلد *Time_to_live* مشخص می‌کند که این رکورد چقدر دوام می‌آورد. این فیلد در رکوردهای با دوام مقدار زیادی دارد، مثلاً ۸۶۴۰۰ (تعداد ثانیه‌های یک شبانه‌روز)؛ و بر عکس، اطلاعات کم دوام عمر کوتاهی دارد، مثلاً ۶۰ (یک دقیقه). در بحث حافظهٔ نهان به این موضوع برخواهیم گشت.

فیلد سوم هر رکورد منبع *Class* است. برای اطلاعات اینترنتی این فیلد همیشه *IN* است؛ برای اطلاعات غیراینترنتی از کُدهای دیگری هم می‌توان استفاده کرد (که البته بندرت دیده می‌شوند).

فیلد *Type* نوع رکورد منبع را مشخص می‌کند. مهمترین انواع رکوردهای منابع را در شکل ۷-۲ ملاحظه می‌کنید.

رکورد *SOA* نام منبع اصلی اطلاعات منطقه (zone) ، آدرس ایمیل سرپرست ناحیه، شماره سریال منحصر به فرد آن، و مشخصات دیگر ناحیه را مشخص می‌کند.

نوع	مفهوم	مقدار
SOA	Start of Authority	پارامترهای منطقه
A	IP address of a host	عدد صحیح ۳۲ بیتی
MX	Mail exchange	تقدم دریافت ایمیل
NS	Name Server	نام سرویس دهنده ناحیه
CNAME	Canonical name	نام ناحیه
PTR	Pointer	نام مستعار برای آدرس IP
HINFO	Host description	مشخصات CPU و سیستم عامل
TXT	Text	متن تسفیر نشده

شکل ۷-۲۰ انواع رکوردهای منابع اصلی DNS برای IPv4 .

مهمترین نوع رکورد منبع، رکورد *A* (آدرس) است. هر رکورد *A* یک آدرس IP ۳۲ بیتی را در خود نگه می‌دارد. هر کامپیوتر اینترنت باید حداقل یک آدرس IP داشته باشد، تا کامپیوترهای دیگر بتوانند با آن تماس بگیرند. برخی از کامپیوترها دو یا چند آدرس IP دارند، که برای هر آدرس IP چنین کامپیوتری باید یک رکورد *A* وجود داشته باشد. DNS را می‌توان بگونه‌ای پیکربندی کرد که در میان این رکوردها بچرخد، یعنی برای اولین درخواست اولین رکورد را برگرداند، برای درخواست دوم دومین رکورد را، و الی‌آخر.

رکورد مهم بعدی رکورد *MX* است. این رکورد آدرس سرویس‌دهنده پُست الکترونیک (ایمیل) ناحیه را مشخص می‌کند. اختصاص یک نوع رکورد خاص به سرویس‌دهنده پست الکترونیک بدین خاطر است که تمام کامپیوترها چنین قابلیتی (دریافت ایمیل) ندارند. برای مثال، اگر کسی بخواهد به *bill@microsoft.com* ایمیل بفرستد، باید آدرس سرویس‌دهنده پست الکترونیک *microsoft.com* را پیدا کند. این اطلاعات را رکورد *MX* عرضه می‌کند.

رکورد *NS* **سرویس‌دهنده‌نام** (name server) را مشخص می‌کند. برای مثال، معمولاً هر پایگاه دادهٔ DNS یک رکورد NS برای ناحیه‌های سطح بالا دارد. (باز هم به این موضوع برمی‌گردیم.)

از رکورد *CNAME* می‌توان برای ایجاد **نامهای‌مستعار** (alias) استفاده کرد. برای مثال، فرض کنید دوستی بنام پاول در دانشکدهٔ مهندسی کامپیوتر دانشگاه MIT دارید، و می‌خواهید برای وی ایمیل بفرستید. فقط می‌دانید که نام وی در شبکهٔ این دانشگاه *paul* است، اما آدرس ایمیل کامل او را ندارید. حدس می‌زنید که ناحیهٔ دانشکدهٔ مزبور *cs.mit.edu* باشد، اما مسئولین این دانشکده (شاید از روی کج‌سلیقگی) نام *lcs.mit.edu* را برای ناحیهٔ خود را انتخاب کرده‌اند. اگر ایمیلی به آدرس *paul@cs.mit.edu* بفرستید، مسلماً برگشت خواهد خورد؛ ولی اگر مسئولین این دانشکده کمی هوشیاری بخرج دهند، با تعریف یک نام مستعار می‌توانند تا حدی اشتباه خود را جبران کنند - برای این منظور می‌توان از یک رکورد *CNAME* مانند زیر استفاده کرد:

```
cs.mit.edu        86400        IN        CNAME    lcs.mit.edu
```

رکورد *PTR* هم، مانند *CNAME* ، به یک نام دیگر اشاره می‌کند. اما بر خلاف *CNAME* (که در واقع یک ماکرو است)، رکورد *PTR* یک نوع دادهٔ معمولی DNS است که تفسیر آن به محتویات این رکورد بستگی دارد. در عمل، تقریباً همیشه از این رکورد برای **جستجوی‌معکوس** (reverse lookup) - تبدیل آدرس IP به نام ماشین - استفاده می‌شود.

رکورد *HINFO* اطلاعات مربوط به نوع ماشین و سیستم عامل آنرا برمی‌گرداند. از رکورد *TXT* هم می‌توان

برای برگرداندن اطلاعات متنی اضافی به کاربران استفاده کرد. رکوردهای *HINFO* و *TXT* فقط برای راحتی کاربران تعبیه شده‌اند، و الزامی نیستند. اغلب اوقات چنین اطلاعاتی وجود ندارد، و اگر هم وجود داشته باشد، نمی‌توان به آن کاملاً اطمینان کرد.

آخرین فیلد رکورد منبع، فیلد *Value* (مقدار) است. این فیلد می‌تواند یک عدد، نام ناحیه، یا یک رشتهٔ متنی باشد. طرز وارد کردن این مقدار به نوع آن بستگی دارد (در شکل ۷-۲ توضیح کوتاهی دربارهٔ نوع هر فیلد آورده شده است).

برای آشنایی بیشتر با اطلاعاتی که در پایگاه دادهٔ DNS یک ناحیه می‌توان یافت، شکل ۷-۳ را ببینید. در این شکل قسمتی از پایگاه دادهٔ نیمه‌فرضی ناحیه‌ای بنام *cs.vu.nl* (شکل ۷-۱) نشان داده شده است. در این پایگاه داده هفت نوع رکورد منبع وجود دارد.

اولین خط غیرتوضیحی شکل ۷-۳ مقداری اطلاعات اولیه دربارهٔ ناحیهٔ *cs.vu.nl* می‌دهد، که فعلاً با آنها کاری نداریم. دو خط بعدی مقداری اطلاعات متنی دربارهٔ این ناحیه (و محل آن) می‌دهند. پس از آن دو کامپیوتری که مسئول دریافت ایمیل‌های ناحیهٔ *cs.vu.nl* هستند، مشخص شده‌اند. اولین کامپیوتری که ایمیل‌ها باید به آن فرستاده شوند، *zephyr* نام دارد؛ و اگر *zaphyr* جواب نداد، نوبت به *top* می‌رسد.

بعد از یک خط خالی (که فقط برای خواناتر کردن فایل است)، رکوردهایی آمده‌اند که می‌گویند *flits* یک کامپیوتر Sun با سیستم عامل UNIX است، و دو آدرس IP دارد (130.37.16.112 و 192.31.231.165). پس از آن سه ماشین برای دریافت ایمیل‌های *flits.cs.vu.nl* مشخص شده است: اولین آنها طبیعتاً خود *flits* است، ولی

```
; Authoritative data for cs.vu.nl
cs.vu.nl.       86400   IN  SOA     star boss (952771,7200,7200,2419200,86400)
cs.vu.nl.       86400   IN  TXT     "Divisie Wiskunde en Informatica."
cs.vu.nl.       86400   IN  TXT     "Vrije Universiteit Amsterdam."
cs.vu.nl.       86400   IN  MX      1 zephyr.cs.vu.nl.
cs.vu.nl.       86400   IN  MX      2 top.cs.vu.nl.

flits.cs.vu.nl. 86400   IN  HINFO   Sun Unix
flits.cs.vu.nl. 86400   IN  A       130.37.16.112
flits.cs.vu.nl. 86400   IN  A       192.31.231.165
flits.cs.vu.nl. 86400   IN  MX      1 flits.cs.vu.nl.
flits.cs.vu.nl. 86400   IN  MX      2 zephyr.cs.vu.nl.
flits.cs.vu.nl. 86400   IN  MX      3 top.cs.vu.nl.
www.cs.vu.nl.   86400   IN  CNAME   star.cs.vu.nl
ftp.cs.vu.nl.   86400   IN  CNAME   zephyr.cs.vu.nl

rowboat                 IN  A       130.37.56.201
                        IN  MX      1 rowboat
                        IN  MX      2 zephyr
                        IN  HINFO   Sun Unix

little-sister           IN  A       130.37.62.23
                        IN  HINFO   Mac MacOS

laserjet                IN  A       192.31.231.216
                        IN  HINFO   "HP Laserjet IIISi" Proprietary
```

شکل ۷-۳. قسمتی از پایگاه دادهٔ DNS در ناحیهٔ *cs.vu.nl* .

اگر این ماشین خاموش بود، *zephyr* و *top* گزینه‌های بعدی خواهند بود. بعد از آن یک نام مستعار برای ماشین *star.cs.vu.nl* تعریف شده است: *www.cs.vu.nl* . با تعریف این نام مستعار، کاربران می‌توانند بدون دغدغهٔ تغییر آدرس صفحهٔ وب *cs.vu.nl* به آن مراجعه کنند. همین کار برای *ftp.cs.vu.nl* نیز انجام شده است.

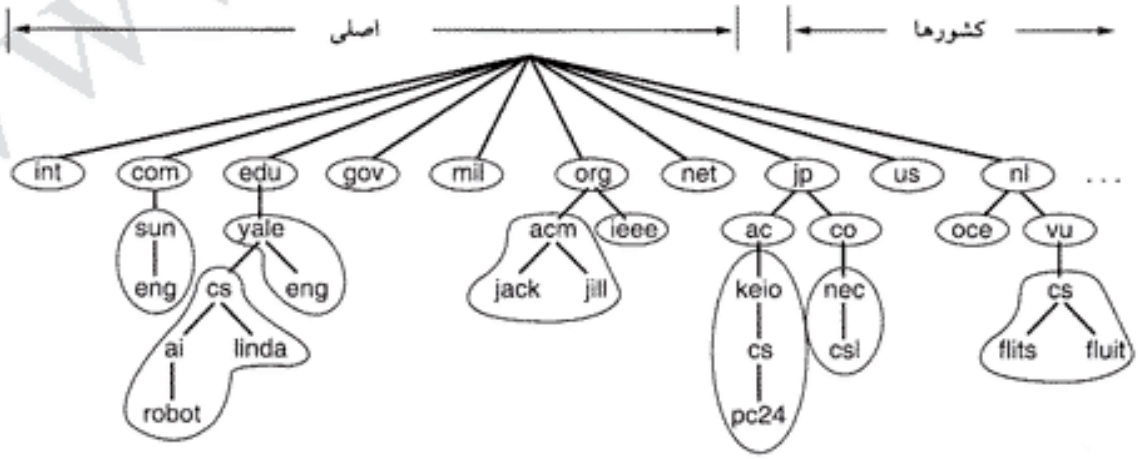
در چهار خط بعدی رکوردهای منبع کامپیوتری بنام *rowboat.cs.vu.nl* تعریف شده‌اند. این اطلاعات شامل آدرس IP ، محل دریافت ایمیل (اولیه و ثانویه)، و اطلاعاتی دربارهٔ خود ماشین است. پس از آن یک کامپیوتر MAC (بدون قابلیت دریافت ایمیل)، و بدنبال آن یک چاپگر لیزری که به اینترنت متصل است، تعریف شده‌اند.

چیزی که در اینجا نشان داده نشده (و در واقع در این فایل هم نیست)، آدرس IP ناحیه‌های سطح بالا است. از آنجائیکه این کار در حوزهٔ مسئولیت ناحیهٔ *cs.vu.nl* نیست، در این فایل هم رکوردی برای آن وجود ندارد. این قبیل اطلاعات را کامپیوترهایی بنام **سرویس‌دهنده‌ریشه** (root server) ارائه می‌کنند، که با هر بار اجرای سرویس‌دهنده DNS آدرس آنها در حافظه حافظهٔ نهان DNS بار می‌شود. تعداد سرویس‌دهنده‌های ریشه در حدود ۱۲ تاست که در سراسر دنیا پراکنده‌اند، و آدرس IP تمام ناحیه‌های سطح بالا را دارند. بنابراین، اگر ماشینی آدرس IP حداقل یکی از این سرویس‌دهنده‌های ریشه را داشته باشد، می‌تواند نام هر ناحیه‌ای را پیدا کند.

۳-۱-۷ سرویس‌دهنده نام

از نظر تئوری، برای نگهداری تمام اطلاعات DNS و پاسخ دادن به درخواست‌ها یک سرویس‌دهنده DNS کافیست. اما در عمل، بار کاری چنین کامپیوتری آنقدر سنگین خواهد شد که عملاً آنرا بلااستفاده می‌کند. علاوه بر آن، اگر این کامپیوتر از کار بیفتد، تمام اینترنت هم با آن به خواب خواهد رفت.

برای اجتناب از چنین وضعیتی، فضای نام DNS به چندین منطقه (zone) با مرزهای مشخص و غیرمشترک تقسیم شده است. در شکل ۷-۴ یکی از راههای تقسیم فضای نام شکل ۷-۱ را مشاهده می‌کنید. هر منطقه شامل بخشی از درخت DNS است، و سرویس‌دهنده‌های نام آنرا در خود نگه می‌دارد. معمولاً هر منطقه دارای یک **سرویس‌دهنده‌نام**(name server) اولیه است که اطلاعاتش را از فایلی روی دیسک خود می‌گیرد، و یک یا چند سرویس‌دهنده نام ثانویه نیز دارد که آنها اطلاعات خود را از سرویس‌دهنده نام اولیه می‌گیرند. برای بالا بردن ضریب اطمینان، می‌توان تعدادی از سرویس‌دهنده‌های نام یک منطقه را خارج از آن منطقه مستقر کرد.



شکل ۷-۴. فضای نام DNS به منطقه‌های مختلف تقسیم شده است.

تعیین مرزهای یک منطقه بر عهدهٔ سرپرست آن است. تصمیم‌گیری در این باره تا حد زیادی به تعداد سرویس‌دهنده‌های نام منطقه و محل استقرار آنها بستگی دارد. برای مثال، در شکل ۷-۴، دانشگاه ییل دارای یک

سرویس‌دهنده نام برای ناحیه‌های *yale.edu* و *eng.yale.edu* است، اما ناحیهٔ *cs.yale.edu* در منطقهٔ دیگری قرار دارد. چنین تصمیماتی بیشتر به تمایل ناحیه‌ها برای کنترل مستقیم منطقهٔ خود بستگی دارد. در مثال فوق، ناحیهٔ *cs.yale.edu* منطقه‌ای مستقل است، در حالیکه *eng.yale.edu* چنین نیست.

وقتی یک تبدیل‌کننده می‌خواهد آدرس ناحیه‌ای را بداند، ابتدا درخواست خود را به سرویس‌دهنده‌های نام محلی خود می‌دهد. اگر این ناحیه در محدودهٔ قانونی سرویس‌دهنده نام مزبور بود (مانند *ai.cs.yale.edu* که در قلمرو *cs.yale.edu* است)، سرویس‌دهنده نام رکوردهای منبع معتبر را به آن برمی‌گرداند. یک **رکورد معتبر** (authoritative record) رکوردی است که مستقیماً از سرپرست ناحیه منشأ می‌گیرد، و بنابراین همیشه صحیح و معتبر است (بر خلاف **رکوردهای ذخیره‌شده** – cached record – که تاریخ اعتبار آنها می‌تواند منقضی شده باشد).

ولی اگر آن ناحیه در قلمرو سرویس‌دهنده‌های محلی نباشد، سرویس‌دهنده نام این درخواست را به سرویس‌دهنده نام سطح بالای ناحیهٔ مزبور می‌فرستد. برای روشنتر شدن مطلب، به مثالی در شکل ۷-۵ توجه کنید. در اینجا یک تبدیل‌کننده در ناحیهٔ *flits.cs.vu.nl* می‌خواهد آدرس IP ماشین سرویس‌دهنده ناحیهٔ *linda.cs.yale.edu* را بداند. در مرحلهٔ ۱، این تبدیل‌کننده درخواست خود را به سرویس‌دهنده نام محلی، یعنی *cs.vu.nl* ، می‌فرستد. این درخواست شامل نام ناحیهٔ موردنظر (رکوردهای نوع *A* و کلاس *IN*) می‌باشد.

شکل ۷-۵. مراحل جستجوی نام ناحیه.

اجازه دهید فرض کنیم سرویس‌دهنده نام محلی تا بحال چیزی از ناحیهٔ *linda.cs.yale.edu* نشنیده و دربارهٔ آن هیچ اطلاعاتی ندارد. این سرویس‌دهنده می‌تواند از همسایه‌های خود در این باره پرس و جو کند، ولی اگر آنها هم بی‌اطلاع بودند، یک بستهٔ UDP به سرویس‌دهنده نام *edu* یعنی *edu-server.net* (که آدرس آنرا در حافظه‌اش دارد) می‌فرستد (شکل ۷-۵ را ببینید). احتمال کمی هست که این سرویس‌دهنده آدرس *linda.cs.yale.edu* (یا حتی *cs.yale.edu*) را بداند، ولی حتماً بچه‌های خودش را می‌شناسد، پس درخواست را به سرویس‌دهنده نام *yale.edu* منتقل می‌کند (مرحلهٔ ۳). سرویس‌دهنده *yale.edu* هم درخواست را به *cs.yale.edu* هدایت می‌کند (مرحلهٔ ۴)، که باید اطلاعات معتبر را در اختیار داشته باشد. از آنجائیکه این مسیر از مشتریهای مختلفی عبور کرده، رکوردهای درخواستی هم باید از همان مسیر به *flits.cs.vu.nl* برگردند (مراحل ۵ تا ۸).

وقتی این رکوردها به *cs.vu.nl* رسید، در حافظهٔ نهان آنجا ذخیره می‌شود تا در دفعات بعد مورد استفاده قرار گیرد. ولی این اطلاعات معتبر نیستند، چون هر تغییری که در *cs.yale.edu* داده شود بطور خودکار در حافظهٔ نهان DNS هایی که این رکوردها در آنجا وجود دارد، پخش نخواهد شد. بهمین دلیل، آیتمهای حافظهٔ نهان نباید عمری طولانی داشته باشند؛ و علت قرار دادن فیلد *Time_to_live* در رکوردهای منبع نیز همین است. این فیلد به سرویس‌دهنده نام می‌گوید که تا چه مدتی می‌تواند رکورد را در حافظهٔ نهان خود نگه دارد. اگر یک ماشین IP خود را سالها حفظ می‌کند، نگه داشتن آن برای ۱ روز در حافظهٔ نهان DNS چندان غیرمنطقی نیست. اما اطلاعات ناپایدارتر را بهتر است بیش از چند ثانیه (یا حداکثر ۱ دقیقه) در حافظهٔ نهان نگه نداریم.

به پرس و جوهایی که به طریق بالا عمل می‌کنند، **جستجوی بازگشتی** (recursive query) می‌گویند، چون هر سرویس‌دهنده‌ای که اطلاعات خواسته شده را نداشته باشد، آنرا به سرویس‌دهنده بالاتر هدایت کرده و جواب را

باز می‌گرداند. روش جستجوی دیگری نیز وجود دارد: سرویس‌دهنده‌ای که اطلاعات درخواست شده را ندارد، خود به سرویس‌دهنده بالاتر مراجعه نمی‌کند، بلکه آدرس آنرا به درخواست کننده برمی‌گرداند. برخی از سرویس‌دهنده‌های DNS قادر به انجام جستجوی بازگشتی نیستند، و همیشه آدرس سرویس‌دهنده بعدی را برمی‌گردانند.

اگر یک مشتری DNS در زمان مقرر پاسخ خود را دریافت نکند، سراغ سرویس‌دهنده DNS بعدی خواهد رفت. این اتفاق بیشتر در مواردی رخ می‌دهد که سرویس‌دهنده DNS بدلایلی از مدار خارج شده باشد.

با اینکه DNS کار ساده‌ای انجام می‌دهد (تبدیل نام به آدرس IP)، اما کارکرد صحیح آن در اینترنت اهمیت حیاتی دارد. DNS هیچ کمکی برای یافتن افراد، منابع، سرویسها، اشیاء و چیزهایی از این قبیل نمی‌تواند بکند؛ برای این کارها سرویس دیگری بنام LDAP (پروتکل سبک دسترسی دایرکتوری – Light-weight Directory Access Protocol) تعریف شده است. این سرویس شکل ساده شدهٔ سرویس دایرکتوری OSI X.500 است، که در استاندارد RFC 2251 تشریح شده است. در LDAP (که می‌توان آنرا «دفتر تلفن اینترنتی» بشمار آورد)، اطلاعات بصورت درختی منظم شده‌اند، و امکانات فراوانی برای جستجو در این درخت تعبیه شده است. در این کتاب بیش از این دربارهٔ LDAP صحبت نخواهیم کرد؛ برای کسب اطلاعات بیشتر می‌توانید به (Weltman and Dahbura, 2000) مراجعه کنید.

۲-۷ پُست الکترونیک

بیش از دو دهه است که پُست الکترونیک، یا آنطور که هوادارانش می‌گویند ایمیل (e-mail)، در صحنه حضور دارد. تا سال ۱۹۹۰، این سرویس بیشتر در دانشگاهها و مراکز علمی وجود داشت، ولی وقتی در این سال بصورت سرویسی عمومی در آمد، با چنان سرعتی رشد کرد که در طی یک دهه تعداد نامه‌های الکترونیکی فرستاده شده از تعداد نامه‌های کاغذی فراتر رفت.

ایمیل، مانند سایر روشهای ارتباطی، دارای قواعد و شیوه‌های خاص خود است. جاذبهٔ ایمیل بسیار بالاست، بطوریکه حتی آنهائیکه بندرت نامه‌های معمولی می‌نویسند، در نوشتن نامه‌های الکترونیکی (حتی به مقامات رسمی و سطح بالا) تردیدی بخود راه نمی‌دهند.

نامه‌های الکترونیکی پُر از کلماتیست که قبلاً در هیچ کجا دیده نشده‌اند: BTW (By The Way – راستی)، ROTFL (Rolling On The Floor Laughing – از خنده غش کردم)، و IMHO (In My Humble Opinion – به نظر من ناقابل) از آن نمونه‌اند. بسیاری افراد نیز در ایمیلهای خود از علائم خاصی موسوم به خندانک (smiley) یا احساس‌نما (emoticon) استفاده می‌کنند. در شکل ۷-۶ تعدادی از معروفترین این خندانک‌ها (و معنای آنها) را می‌بینید. اگر می‌خواهید بهتر متوجه معنای این علائم شوید، کتاب را ۹۰° در جهت

مفهوم	خندانک	مفهوم	خندانک	مفهوم	خندانک
دماغ گنده	(:+)	عمو لینکن	(:-|=)	من خوشحالم	(:-)
غبغب بزرگ	(:-))	عمو سام	(:-)=	من غمگین / ناراحتم	(:-(
سبیلو	(:-{)	بابا نوئل	(:-<*)	من بی تفاوتم	(:-|
ژولیده مو	(:-#)	کودن / احمق	(:-<___)	من چشمک می زنم	(;-)
عینکی	(8-)	استرالیایی	(:-)	من خمیازه می کشم	(O):-)
با هوش	(C:-)	مرد فلکی	(X-:)	حالم به هم خورد	(*)-:

شکل ۷-۶. چند خندانک. اینها جزء امتحان نهایی نیستند (:

عقربه‌های ساعت بچرخانید. برای دیدن تعداد زیادی از این قبیل خندانک‌ها به (Sanderson and Dougherty, 1993) مراجعه کنید.

اولین سیستم ایمیل فقط یک پروتکل سادهٔ انتقال فایل (file transfer) بود، که آدرس گیرنده در خط اول پیام (فایل) نوشته می‌شد. با گذشت زمان محدودیت‌های این روش آشکارتر شد، که برخی از آنها عبارت بودند از:

۱. فرستادن یک پیام به چند نفر مشکل بود. این اشکال بیشتر مدیران را آزار می‌داد، چون آنها میل داشتند پیامهای خود را به تمام افراد زیردست خود بفرستند.
۲. پیامها هیچگونه ساختار داخلی نداشتند، و بهمین دلیل پردازش کامپیوتری آنها مشکل بود. برای مثال، اگر پیامی از طرف یک شخص واسطه هدایت یا فرستاده می‌شد، استخراج قسمت هدایت شده مشکل بود.
۳. فرستندهٔ نامه هرگز نمی‌توانست بداند پیامش به گیرنده رسیده یا نه.
۴. اگر کسی قصد داشت برای مدتی به مرخصی برود و می‌خواست در این مدت نامه‌های وارده به دست منشی‌اش برسد، کار ساده‌ای نبود.
۵. واسط کاربر (جایی که نامه را می‌نوشت) با قسمت ارسال نامه یکپارچه نبود. کاربر باید ابتدا نامه را می‌نوشت، و برای ارسال آن برنامهٔ ادیتور را ترک می‌کرد، و به قسمت انتقال فایل می‌رفت.
۶. نامه‌ها فقط متن بود؛ ارسال تصویر، طرح، صدا و مانند آنها ممکن نبود.

بتدریج سیستمهای ایمیل بهتری عرضه شد. در سال ۱۹۸۲، آرپانت سیستم ایمیل پیشنهادی خود را در RFC 821 (پروتکل انتقال) و RFC 822 (فرمت پیام) ارائه کرد. این پیشنهادها با تغییراتی اندک با عنوان RFC 2821 و RFC 2822 به استاندارد اینترنت تبدیل شد - اما هنوز هم استاندارد ایمیل اینترنت را با نام RFC 822 می‌شناسند.

در ۱۹۸۴، CCITT توصیه‌ای بنام X.400 ارائه کرد. بعد از دو دهه، اغلب سیستمهای ایمیل همچنان به RFC 822 پایبند هستند، و X.400 عملاً کنار گذاشته شده است. این که چگونه سیستمی به عظمت X.400 که تمام مقامات رسمی استاندارد، شرکتهای مخابرات سراسر دنیا، دولتها و بسیاری از صنایع کامپیوتری پشتیبان آن بودند، مغلوب سیستمی که چند دانشجوی کامپیوتر آنرا نوشته‌اند، می‌شود بیشتر به داستان داوود و گولیات شبیه است.

علت موفقیت RFC 822 خوبی آن نبود، بلکه این X.400 بود که چنان پیچیده و بد طراحی شده بود که پیاده‌سازی آن را عملاً غیرممکن می‌کرد. انتخاب بین یک سیستم ساده ولی کاری (مانند RFC 822) و سیستمی فوق‌العاده جالب که در عمل کار نمی‌کرد (مانند X.400) چندان دشوار نبود. این عبرت تاریخ است.

۱-۲-۷ معماری و سرویسها

در این قسمت خواهید دید که یک سیستم ایمیل چه کاری می‌تواند انجام دهد، و سازماندهی آن چگونه است. هر سیستم ایمیل دارای دو زیرسیستم است: **عامل‌کاربر** (user agent) ، که به افراد اجازه می‌دهد پیامهای خود را بفرستند و پیامهای رسیده را بخوانند، و **عامل‌انتقال‌پیام** (message transfer agent) ، که پیامها را به دست گیرنده می‌رساند. عامل کاربر برنامه‌ایست (با ظاهر معمولی) روی کامپیوتر محلی کاربر، که با سیستم ایمیل بر هم کنش دارد. در حالیکه عامل انتقال پیام معمولاً یک **سرویس** (service یا daemon) است که در پس‌زمینه اجرا می‌شود، و وظیفهٔ آن انتقال پیام در سیستم ایمیل است.

یک سیستم ایمیل، معمولاً، پنج کارکرد اصلی دارد، که آنها را در زیر توضیح می‌دهیم.

تصنیف: نوشتن پیام و جواب آن. با اینکه از هر ادیتوری می‌توان برای نوشتن پیامها استفاده کرد، ولی اغلب سیستمهای ایمیل دارای ادیتور خاص خود هستند که بسیاری از کارها (از جمله نوشتن آدرس، و سرآیند ایمیل) را بطور خودکار انجام می‌دهند. برای مثال، وقتی می‌خواهید به یک نامه جواب بدهید، سیستم ایمیل می‌تواند آدرس فرستندهٔ نامه را بطور خودکار استخراج کرده و در فیلد گیرندهٔ جوابیه (reply) قرار دهد.

انتقال: فرستادن پیام از فرستنده به گیرنده. این فرآیند سه مرحله دارد: تماس با ماشین گیرنده (یا یک ماشین واسط)، فرستادن پیام، و قطع ارتباط. سیستم ایمیل این کارها را بطور خودکار و بدون دخالت کاربر انجام می‌دهد.

گزارش‌دهی: مطلع کردن کاربر از سرنوشت پیام فرستاده شده. نامه تحویل شد؟ گیرنده آنرا قبول نکرد؟ در راه گم شد؟ در برخی مواقع اطمینان از رسیدن نامه بدست گیرنده اهمیت حیاتی و تبعات قانونی دارد (مانند احضاریه‌های دادگاه).

نمایش: پیامهای رسیده باید بگونه‌ای مناسب در معرض دید کاربر قرار گیرند، تا وی بتواند براحتی آنها را بخواند. گاهی لازم است برای خواندن محتویات برخی نامه‌ها (مانند نامه‌هایی که پیوست صوتی یا تصویری دارند) از برنامه‌های کمکی استفاده شود. برخی از سیستمهای ایمیل نیز نامه‌ها را بگونه‌ای خاص فرمت کرده و نمایش می‌دهند.

بایگانی: نگهداری نامه‌های رسیده. سرنوشت پیامهای رسیده متفاوت است: برخی پیامها حتی قبل از خوانده شدن دور انداخته می‌شوند، برخی فقط یک بار ارزش خواندن دارند، و برخی دیگر را باید حتماً ذخیره کرد. یک سیستم ایمیل باید بتواند نامه‌ها را به انحاء مختلف پردازش کند.

علاوه بر این سرویسهای اصلی، برخی از سیستمهای ایمیل (مخصوصاً سیستمهای رسمی) دارای ویژگیهای پیشرفتهٔ دیگری نیز هستند، که در زیر به برخی از آنها اشاره می‌کنیم.

وقتی یک فرد از محلی نقل مکان می‌کند (یا برای مدتی به مأموریت می‌رود)، باید بتوان پیامهای وی را به محل جدید **هدایت‌کرد** (forward) . سیستم ایمیل باید بتواند این کار را بصورت خودکار انجام دهد.

در اکثر سیستمها کاربران اجازه دارند برای ذخیره کردن پیامهای خود **صندوق‌پستی** (mailbox) داشته باشند. سیستم ایمیل باید فرمانهایی برای ایجاد، مدیریت و یا از بین بردن این صندوق‌ها داشته باشد.

اغلب مدیران نیاز دارند تا یک پیام را به افراد متعددی (کارمندان، مشتریان، یا شرکتهای طرف قرارداد) بفرستند. از اینجا بود که ایدهٔ **لیست‌پستی** (mailing list) – که در واقع لیستی از آدرسهای ایمیل است – پیدا شد. وقتی پیامی به یک لیست پستی فرستاده می‌شود، تمام افراد لیست کپی‌های کاملاً یکسانی از آن پیام دریافت خواهند کرد.

ویژگیهای دیگر عبارتند از: کپی (CC) ، کپی ناشناس (BCC) ، ایمیل با اولویت زیاد، ایمیل سرّی (رمز شده)، گیرندهٔ جانشین (وقتی گیرندهٔ اصلی در دسترس نبود)، و تحویل نامه‌های رئیس به منشی.

امروزه ایمیل کاربرد گسترده‌ای برای ارتباطات داخلی شرکتها و سازمانها دارد. ایمیل می‌تواند گروه گسترده‌ای از افراد (حتی آنهایی که بسیار از یکدیگر دور هستند) را در یک پروژه گرد آورد. ایمیل با حذف اکثر تمایزها (مانند مقام، سن، و جنسیت) باعث تمرکز روی اهداف می‌شود. با ایمیل، ایدهٔ درخشان یک کارآموز ساده اهمیت فزونتری نسبت به ایده‌های (اکثراً احمقانهٔ) مدیر عامل خواهد یافت.

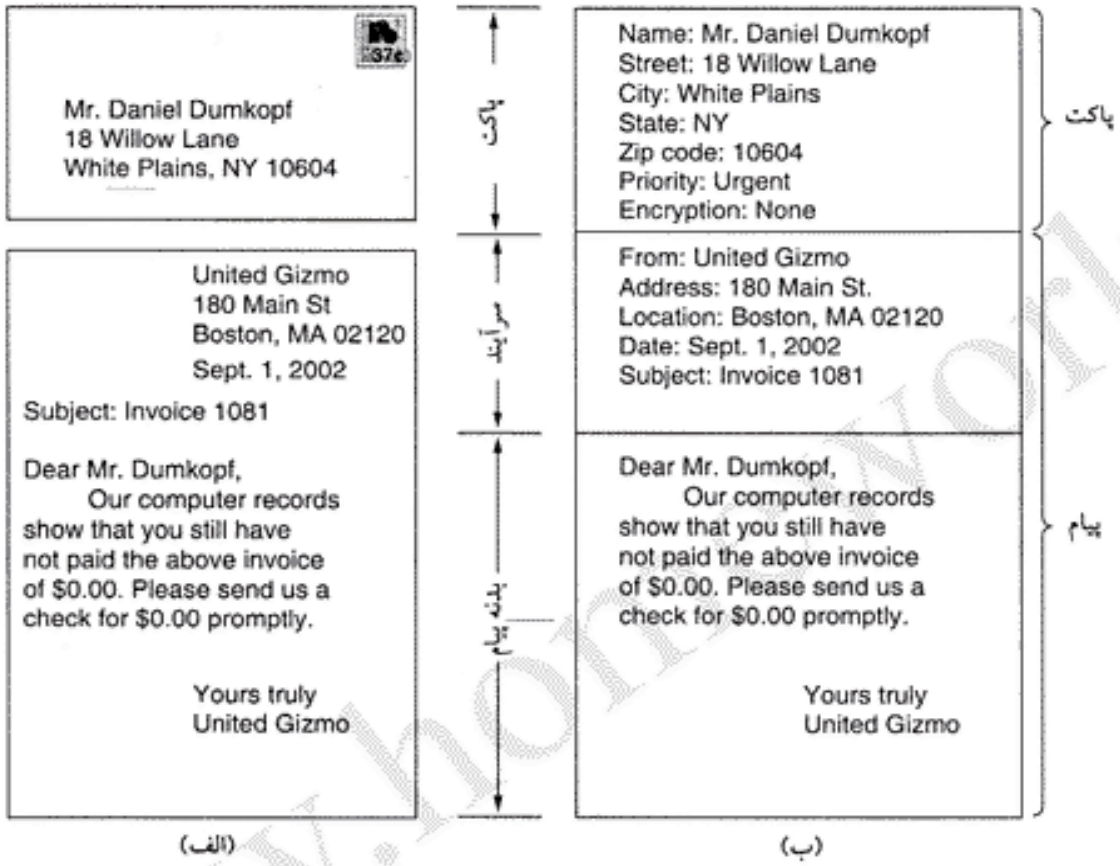
ایدهٔ کلیدی در سیستم ایمیل تمایز بین پاکت و محتویات نامه است. پاکت نامه پیام را در خود جای می‌دهد، و شامل اطلاعاتی از قبیل آدرس گیرنده، اولویت و سطح امنیتی آن می‌شود، که معمولاً ارتباطی با محتویات نامه ندارند. عامل انتقال پیام از اطلاعات این پاکت برای جابجایی صحیح پیام استفاده می‌کند (درست مثل ادارهٔ پُست).

محتویات پاکت دو بخش دارد: سرآیند (header) و بدنه (body) . سرآیند شامل اطلاعات کنترلیست که عامل کاربر از آنها برای کار خود استفاده می‌کند. بدنه بخشی از پیام است که به گیرنده مربوط می‌شود. در شکل ۷-۷ رابطهٔ پاکت و پیام نشان داده شده است.

۷-۲-۲ عامل کاربر

همانطور که گفتیم، سیستمهای ایمیل دو بخش اصلی دارند: عامل کاربر، و عامل انتقال پیام. در این قسمت بخش

اول (یعنی، عامل کاربر) را بررسی خواهیم کرد. عامل کاربر یک برنامهٔ معمولیست (و گاهی به آن نامه‌خوان – mail reader – نیز می‌گویند)، که می‌تواند در نوشتن پیامها، خواندن نامه‌های رسیده، پاسخ به آنها و کارهایی از این قبیل به کاربر کمک کند. طیف وسیع و متنوعی از برنامه‌های عامل کاربر وجود دارد، اما کارکرد اصلی آنها بسیار شبیه یکدیگر است.



شکل ۷-۷. پاکت و پیام در (الف) پُست کاغذی، (ب) پُست الکترونیک.

فرستادن ایمیل

برای فرستادن یک ایمیل، کاربر باید ابتدا متن نامه را نوشته و آدرس گیرنده را هم مشخص کند. برای نوشتن نامه می‌توان از ادیتورها یا واژه‌پردازهای مستقل، و یا ادیتوری که به همراه عامل کاربر می‌آید، استفاده کرد. آدرس گیرنده باید با فرمتی باشد که برای عامل کاربر آشناست؛ بسیاری از آنها آدرسها را با فرمت *user@dns-address* می‌پذیرند. با فرمت نامهای DNS در ابتدای همین فصل آشنا شدید.

غیر از آدرسهای DNS فرمتهای دیگری هم برای آدرسهای ایمیل وجود دارد، که بویژه نوع X.400 آن قابل توجه است. آدرسهای X.400 تفاوت قابل توجهی با آدرسهای DNS دارند. هر آدرس X.400 مجموعه ایست از زوجهای *value* = *attribute* که با / از هم جدا می‌شوند:

/C=US/ST=MASSACHUSETTS/L=CAMBRIDGE/PA=360 MEMORIAL DR./CN=KEN SMITH/

در این آدرس بترتیب از چپ براست کشور (C) ، ایالت (ST) ، محل (L) ، آدرس (PA) ، و نام شخص (CN) نوشته می‌شود. مشخصات دیگری (مانند شرکت و شغل) نیز وجود دارد، که با استفاده از آنها می‌توانید پیام خود را حتی به فردی که آدرس ایمیل او را نمی‌دانید، بفرستید. با اینکه آدرسهای X.400 بسیار غریب‌تر از نامهای

DNS هستند، اما اغلب سیستمهای ایمیل اجازه می‌دهند تا هر آدرس یک نام مستعار ساده (موسوم به nickname) داشته باشد. بدین‌ترتیب دیگر لازم نیست کاربر آدرس ایمیل (حتی X.400) را بطور کامل وارد کند.

اغلب سیستمهای ایمیل از لیست پُستی پشتیبانی می‌کنند، بنابراین می‌توان یک نامه را یکباره برای تعداد زیادی از افراد فرستاد. اگر لیست پستی بصورت محلی نگهداری شود، ایمیل در همان مبدأ بین افراد آن لیست توزیع خواهد شد. ولی اگر لیست پستی در محل دیگری نگهداری شود، پیام در آنجا پخش می‌شود. برای مثال، اگر گروهی از طرفداران حیات وحش یک لیست پستی بنام *birders* در دانشگاه آریزونا داشته باشند *(birders@meadowlark.arizona.edu)* ، نامه‌ای که به این آدرس فرستاده شده باشد، فقط بعد از رسیدن به دانشگاه آریزونا بین افراد آن توزیع خواهد شد. آنهایی که در یک لیست پستی قرار دارند، بهیچوجه متوجه این مطلب نخواهند شد؛ آنها هم مانند سایر افراد پیامهایی از یک فرستندهٔ مشخص دریافت می‌کنند.

خواندن ایمیل

معمولاً، وقتی عامل کاربر کار خود را شروع می‌کند، قبل از هر چیز صندوق پستی کاربر را چک کرده و نامه‌های رسیده را از آنجا برمی‌دارد. پس از آن، تعداد نامه‌های رسیده را به کاربر اعلام کرده، و احتمالاً آنها را بصورت خلاصه (شامل اطلاعاتی از قبیل فرستنده، تاریخ دریافت، و موضوع نامه) به کاربر نمایش می‌دهد، و سپس منتظر اقدام بعدی کاربر می‌ماند.

بعنوان مثال، یک سناریوی ساده را در زیر بررسی می‌کنیم. بعد از شروع برنامهٔ عامل کاربر، اولین اقدام معمولاً نمایش خلاصه‌ای از نامه‌های رسیده است. این خلاصه می‌تواند چیزی شبیه شکل ۷-۸ باشد: هر خط مشخصات یک پیام را نشان می‌دهد. در این مثال، هشت پیام در صندوق پستی وجود دارد.

#	پرچم	بایت	فرستنده	موضوع
1	K	1030	asw	Changes to MINIX
2	KA	6348	trudy	Not all Trudys are nasty
3	K F	4519	Amy N. Wong	Request for information
4		1236	bal	Bioinformatics
5		104110	kaashoek	Material on peer-to-peer
6		1223	Frank	Re: Will you review a grant proposal
7		3110	guido	Our paper has been accepted
8		1204	dmr	Re: My student's visit

شکل ۷-۸ نمایش محتویات صندوق پستی.

در هر خط اطلاعات مختلفی دیده می‌شود. در یک سیستم ایمیل ساده این اطلاعات ثابت است، ولی در سیستمهای پیچیده‌تر کاربر می‌تواند نحوهٔ نمایش اطلاعات را بنا به میل خود تغییر دهد (و این تنظیمات را در فایلی بنام **پروفایل‌کاربر** – user profile – ذخیره کند). در مثال بالا، اولین فیلد شمارهٔ پیام است. فیلد دوم، *Flags* ، مقادیر مختلفی می‌تواند بگیرد: *K* یعنی این پیام آرشیوی است (جدید نیست و قبلاً خوانده شده)؛ *A* یعنی به این پیام پاسخ داده شده است؛ *F* یعنی این پیام به فرد دیگری هدایت شده است (forward) . پرچمهای دیگری نیز در این فیلد می‌تواند وجود داشته باشد، که هر کدام معنای خاص خود را دارند.

فیلد سوم طول پیام، و فیلد چهارم فرستندهٔ آنرا نشان می‌دهند. از آنجائیکه این فیلد از نامهٔ رسیده استخراج

می‌شود، محتویات آن به نامهٔ فرستاده شده بستگی دارد. و بالاخره، در فیلد موضوع خلاصه‌ای از محتویات نامه را می‌بینید. سعی کنید همیشه نامه‌هایتان «موضوع» داشته باشد، چون تجربه نشان داده که نامه‌های بدون موضوع براحتی نادیده گرفته می‌شوند.

بعد از خواندن خلاصهٔ نامه، کاربر می‌توان اقدامات مختلفی روی آن انجام دهد: نامه را باز کند، آنرا حذف کند، به نامه جواب دهد، آنرا برای کس دیگری بفرستد، و مانند آن. برای هر یک از این اعمال، فرمان خاصی در برنامهٔ عامل کاربر وجود دارد.

سیستمهای ایمیل امروزی خیلی بیش از انتقال فایل ساده‌اند. مدیریت حجم زیادی از نامه‌ها با برنامه‌های امروزی کار چندان دشواری نیست؛ و برای افرادی که در سال هزاران ایمیل رد و بدل می‌کنند، این مزیت کوچکی نیست.

۷-۲-۳ فرمت پیامها

اجازه دهید کمی هم دربارهٔ فرمت پیامهای ایمیل صحبت کنیم. ابتدا به ایمیلهای متنی ساده با فرمت RFC 822 می‌پردازیم، و پس از آن دربارهٔ الحاقات چندرسانه‌ای در RFC 822 توضیح خواهیم داد.

RFC 822

هر پیام یک پاکت ساده (که مشخصات آن در RFC 821 آمده) دارد، بعلاوهٔ تعدادی سرآیند، یک خط خالی، و بعد از آن متن یا بدنهٔ پیام. هر فیلد سرآیند عبارتست از یک خط متن ASCII مشتمل بر: نام فیلد، علامت : ، و مقدار فیلد (که البته برخی از فیلدها می‌توانند مقدار نداشته باشند). در استاندارد RFC 822 که بیش از دو دهه از عمر آن می‌گذرد، تمایز آشکاری بین فیلدهای پاکت نامه و فیلدهای سرآیند وجود ندارد. با اینکه در RFC 2822 این مشکل مرتفع شده، ولی بعلت رواج گستردهٔ آن در عمل چنین اتفاقی نیفتاده است. در حالت عادی، عامل انتقال پیام با استخراج اطلاعات از نامه‌ای که از عامل کاربر دریافت می‌کند، پاکت نامه را می‌سازد، و بهمین دلیل پاکت و نامه با هم مخلوط می‌شوند.

مهمترین فیلدهای سرآیند که نقش مهمی در انتقال پیام دارند، در شکل ۷-۹ نشان داده شده است. در فیلد *TO:* آدرس DNS گیرندهٔ اصلی پیام نوشته می‌شود. نوشتن چندین گیرنده در این فیلد مجاز است. در فیلد *Cc:* آدرس گیرنده‌های ثانویهٔ پیام نوشته می‌شود. از نظر تحویل پیام در سیستم ایمیل، تفاوتی بین گیرنده‌های اصلی و ثانویه وجود ندارد؛ این فیلد بیشتر برای انسانها مهم است تا برای ماشین. اصطلاح *Cc* (کپی کربنی) قدری قدیمی است، چون در کامپیوترها چیزی بنام کاغذ کپی وجود ندارد، اما این اصطلاح دیگر کاملاً جا افتاده است. فیلد *Bcc:* (کپی کربنی ناپیدا) شبیه *Cc:* است، با این تفاوت که این فیلد در نامه‌های کپی شده حذف می‌شود، و گیرندهٔ نامه از هویت سایر گیرنده‌ها (و حتی وجود چنین گیرنده‌هایی) مطلع نخواهد شد.

سرآیند	مفهوم
To:	آدرس ایمیل گیرنده های اصلی
Cc:	آدرس ایمیل گیرنده های ثانویه (کپی)
Bcc:	آدرس ایمیل گیرنده های کپی‌های ناشناس
From:	فرستنده پیام
Sender:	آدرس ایمیل فرستنده
Received:	هر عامل انتقال واسطه در بین راه مشخصات خود را اضافه می کند
Return-Path:	می توان از آن برای مشخص کردن مسیر برگشت به فرستنده استفاده کرد

شکل ۷-۹. فیلدهای سرآیند RFC 822 برای انتقال پیام.

دو فیلد بعدی، یعنی *From:* و *Sender:* ، بترتیب نویسنده و فرستندهٔ نامه را مشخص می‌کنند. این دو الزاماً یکی نیستند (ولی اغلب چنین است). برای مثال، نویسندهٔ نامه می‌تواند مدیر عامل باشد *(From:)* ، ولی منشی شرکت آنرا بفرستد *(Sender:)* . فیلد *From:* حتماً باید پُر شود، ولی فیلد *Sender:* (اگر با *From:* یکی باشد) می‌تواند خالی رها شود. اگر احتمال می‌دهید نامه بدست گیرنده نمی‌رسد و برگشت می‌خورد، حتماً فیلد *Sender:* را پُر کنید، چون نامه‌های غیرقابل‌تحویل به این آدرس برگشت داده می‌شوند.

وقتی یک نامه از واسطه‌های مختلفی عبور می‌کند تا به دست گیرنده برسد، نام هر واسطه در یک فیلد *Received:* جداگانه نوشته می‌شود. در این فیلد نام عامل گیرنده، تاریخ و زمان دریافت پیام، و اطلاعات دیگر ثبت می‌شود. از این اطلاعات می‌توان برای رفع اشکالاتی که در طی تحویل نامه‌ها پیش می‌آید، استفاده کرد.

فیلد *Return-Path:* که توسط آخرین عامل انتقال پیام اضافه می‌شود، نشان می‌دهد که مسیر برگشت به فرستنده چگونه است. از نظر تئوری این اطلاعات باید شامل تمام سرآیندهای *Received:* (بجز صندوق پستی فرستنده) باشد، ولی بندرت چنین است و معمولاً فقط آدرس فرستنده در آن نوشته می‌شود.

علاوه بر فیلدهای شکل ۷-۹، پیامهای RFC 822 دارای سرآیندهای دیگری نیز هستند که به بیشتر کار عامل کاربر یا شخص گیرنده می‌آیند. در شکل ۷-۱۰ برخی از این سرآیندها را می‌بینید. کارکرد اغلب این فیلدها از روی نامشان پیداست، و نیازی به توضیح زیاد ندارند.

سرآیند	مفهوم
Date:	زمان و تاریخ ارسال
Reply-To:	آدرس ایمیل برای پاسخ نامه
Message-Id:	عدد منحصر بفرد شناسایی پیام
In-Reply-To:	شماره پیامی که این پاسخ پیام پاسخ آن است
References:	سایر شماره های مربوطه
Keywords:	کلمات کلیدی انتخاب شده توسط کاربر
Subject:	موضوع پیام

شکل ۷-۱۰. برخی از فیلدهای سرآیند RFC 822 .

فیلد *Reply-To:* برای مواقعیست که نویسنده و گیرندهٔ نامه هیچکدام نمی‌خواهند گیرندهٔ پاسخ نامه باشند. برای مثال، وقتی مدیر بازاریابی نامه‌ای دربارهٔ محصولات جدید شرکت به یک مشتری می‌نویسد، و منشی هم آنرا می‌فرستد، فیلد *Reply-To:* می‌تواند به آدرس قسمت فروش شرکت، که پاسخگوی سفارشات هستند، اشاره کند. این فیلد برای مواردی که فرستنده دو آدرس ایمیل دارد، و مایل است پاسخها را از طریق آدرس دیگرش بگیرد، نیز مفید است.

استاندارد RFC 822 اجازه می‌دهد تا کاربران سرآیندهای دلخواهشان را به نامه‌ها اضافه کنند، مشروط باینکه این سرآیندها با *X-* شروع شوند (هیچیک از سرآیندهای رسمی این استاندارد با *X-* شروع نمی‌شوند، و در آینده نیز نخواهند شد). این قبیل سرآیندها می‌توانند اطلاعات اضافی را با خود حمل کنند.

بعد از سرآیند، بدنهٔ نامه می‌آید. کاربران می‌توانند هر چیزی در این بدنه بنویسند. برخی افراد در انتهای نامه‌هایشان اختتامیه‌های ماهرانه‌ای می‌آورند، مانند اشکال کارتونی جالب و خنده‌دار، کلمات قصار مشاهیر، و یا عبارات قانونی سلب مسئولیت (مثلاً، «شرکت فلان و بهمان هیچگونه مسئولیتی را دربارهٔ محتویات این نامه نمی‌پذیرد.»).

MIME - الحاقات چندمنظورهٔ پُست اینترنت

در روزهای اولیهٔ آرپانت، ایمیل‌ها فقط متن ساده بود که به زبان انگلیسی و با فرمت ASCII نوشته می‌شد. استاندارد RFC 822 با این وضعیت هیچ مشکلی نداشت: کاربر می‌توانست هر چیزی که می‌خواست در بدنهٔ نامه بنویسد. اما این روش دیگر برای دنیای امروز کافی نیست. برخی از مشکلات ذاتی این سیستم عبارتند از:

۱. ارسال پیام به زبانهایی که اِعراب دارند (مانند فرانسه و آلمانی).
۲. ارسال پیام به زبانهای غیرلاتین (مانند عربی و روسی).
۳. ارسال پیام به زبانهای غیرالفبایی (مانند چینی و ژاپنی).
۴. ارسال پیامهایی که اصلاً متن نیستند (مانند صدا و تصویر).

راه حل این مشکلات در RFC 1341 ارئه شد، و بعدها در RFC 2045-49 به روز در آمد. این راه حل که MIME (الحاقات چندمنظورهٔ پُست‌اینترنت – Multipurpose Internet Mail Extensions) نام دارد، امروزه بطور گسترده‌ای رواج یافته است.

ایدهٔ اصلی MIME عبارتست از: ادامهٔ استفاده از فرمت RFC 822 ، و اضافه کردن ساختاری جدید به بدنهٔ پیام و تعریف قواعد درج پیامهای غیرمتنی. پیامهای MIME با برنامه‌ها و پروتکلهای موجود ایمیل کاملاً سازگارند، چون از استاندارد RFC 822 تخطی نمی‌کنند. فقط برنامه‌های عامل کاربر باید عوض شوند، که این کار را هم کاربران می‌توانند براحتی انجام دهند.

MIME پنج سرآیند جدید تعریف می‌کند، که آنها را در شکل ۷-۱۱ مشاهده می‌کنید. اولین سرآیند به عامل کاربر دریافت کنندهٔ پیام می‌گوید که با یک پیام MIME سروکار دارد، و ویرایش آن هم اعلام می‌شود. هر پیامی که سرآیند *MIME-Version:* نداشته باشد، متن ساده تلقی شده و به همان طریق پردازش می‌شود.

سرآیند	مفهوم
MIME-Version:	ویرایش MIME پیام
Content-Description:	جمله ای درباره محتویات پیام
Content-Id:	عدد منحصر بفرد شناسایی محتویات پیام
Content-Transfer-Encoding:	نحوه کدگذاری پیام
Content-Type:	نوع و فرمت محتویات پیام

شکل ۷-۱۱. سرآیندهای اضافی MIME در RFC 822 .

سرآیند *Content-Description:* یک عبارت متنی است که می‌گوید چه چیزی در پیام وجود دارد. از روی این سرآیند است که گیرنده تشخیص می‌دهد آیا محتویات پیام ارزش خواندن دارد یا خیر. برای مثال، اگر سرآیند *Content-Description:* بگوید: «این عکس یک موش است»، و گیرنده علاقه‌ای به دیدن عکس موش نداشته باشد، برنامه پیام را دور انداخته و تلاشی برای نمایش این عکس نخواهد کرد.

سرآیند *Content-Id:* محتویات پیام را مشخص می‌کند. فرمت این سرآیند مانند *Message-Id:* است.

سرآیند *Content-Transfer-Encoding:* نحوهٔ کُد شدن محتویات پیام (برای گذر از شبکه‌هایی که فقط به متن ساده اجازهٔ عبور می‌دهند) را مشخص می‌کند. پنج نوع کُدگذاری پیام وجود دارد. نوع اول فقط متن سادهٔ ASCII است. کاراکترهای ASCII هفت‌بیتی هستند، و تمام پروتکلهای ایمیل می‌توانند خطوط متن را (مشروط بر اینکه هر خط از ۱۰۰۰ حرف تجاوز نکند) منتقل کنند.

نوع دوم در واقع همان نوع قبلیست، که فقط از کاراکترهای ASCII ۸بیتی (از 0 تا 255) استفاده می‌کند.

برخی از بخشهای اینترنت از این کُد برای ارسال متن (و نمایش کاراکترهای خاص) استفاده می‌کنند. این کُدگذاری در پروتکلهای ایمیل اینترنتی مجاز نیست (و این تعریف هم باعث مجاز شدن آن نمی‌شود)، ولی حداقل توضیح می‌دهد که اشکال کار از کجاست. پیامهای دارای کُدگذاری ۸بیتی هم محدود به خطهای ۱۰۰۰ حرفی هستند.

بدتر از آن پیامهایی هستند که کُدگذاری باینری دارند. اینها فایلهای باینری هستند، که نه تنها از تمام حالات ممکنهٔ ۸ بیت استفاده می‌کنند، بلکه به محدودیت ۱۰۰۰ بایت نیز وقعی نمی‌گذارند. برنامه‌های اجرایی در این دسته قرار می‌گیرند. هیچ تضمینی وجود ندارد که پیامهای باینری درست به مقصد برسند، ولی بسیاری افراد همچنان کار خودشان را می‌کنند.

یکی از روشهای کُدگذاری صحیح پیامهای باینری روش **کُدگذاری base64** (که گاهی ASCII armor نیز گفته می‌شود) است. در این روش، هر دستهٔ ۲۴ بیتی به چهار واحد ۶بیتی تقسیم شده، و هر واحد بعنوان یک کاراکتر معتبر ASCII فرستاده می‌شود. در این روش "A" معادل 0 است، "B" معادل 1 ، ... و بالاخره "Z" معادل 26 ؛ پس از آن ۲۶ حرف کوچک انگلیسی می‌آیند، و پس از آن ارقام 0 تا 9 ؛ 62 و 63 نیز بترتیب معادل + و / هستند. توالیهای == و = بترتیب نشان می‌دهند که آخرین گروه ۸ یا ۱۶ بیتی است. کاراکترهای «برگشت سر خط» (carriage return) و «خط بعدی» (line feed) نیز بکلی نادیده گرفته می‌شود، بنابراین می‌توان برای کوتاه کردن خطوط از آنها استفاده کرد. با این روش می‌توان فایلهای باینری را بطور صحیح ارسال کرد.

برای پیامهایی که تقریباً بطور کامل ASCII هستند و فقط چند کاراکتر غیر ASCII دارند، کُدگذاری base64 کارایی مطلوبی ندارد. برای این قبیل پیامها از **کُدگذاری quoted-printable** استفاده می‌شود. این در واقع همان روش ASCII ۷بیتی است، که در آن کاراکترهای بالای 127 با علامت = و یک عدد هگزادسیمال دو رقمی مشخص می‌شوند.

اطلاعات باینری همواره باید با یکی از روشهای base64 یا quoted-printable فرستاده شوند. اگر دلیل موجهی برای استفاده نکردن از هر یک از روشها وجود داشته باشد، می‌توان از کُدگذاریهای خاص (که با سرآیند *Content-Transfer-Encoding:* مشخص می‌شوند) استفاده کرد.

آخرین آیتم شکل ۷-۱۱ در واقع مهمترین سرآیند MIME است. این سرآیند خصلت واقعی محتویات پیام را مشخص می‌کند. در RFC 2045 هفت نوع (type) تعریف شده، که هر کدام از آنها می‌توانند چندین زیرنوع (subtype) داشته باشند. نوع و زیرنوع با یک / از هم جدا می‌شوند:

```
Content-Type: video/mpeg
```

زیرنوع باید صریحاً در سرآیند مشخص شود: هیچ مقداری بعنوان پیش‌فرض وجود ندارد. لیست اولیهٔ نوع‌ها و زیرنوع‌های RFC 2045 را در شکل ۷-۱۲ ملاحظه می‌کنید. از آن زمان تاکنون آیتمهای دیگری اضافه شده است، و در آینده باز هم اضافه خواهد شد.

اجازه دهید این لیست را مختصراً بررسی کنیم. نوع *text* همان متن ASCII ساده است. زیرنوع *text/plain* به گیرنده می‌گوید پیام را بهمان صورتی که دریافت کرده (بدون هیچ فرمت یا پردازشی) نمایش دهد. این گزینه اجازه می‌دهد تا پیامهای معمولی بدون هیچ اقدام اضافه‌ای به پیامهای MIME تبدیل شوند.

زیرنوع *text/enriched* اجازه می‌دهد تا از زبانهای علامتگذاری ساده برای فرمت کردن متن (تعیین فونت، اندازه، رنگ و صفحه‌بندی) استفاده شود. این زیرنوع زیرمجموعه‌ای از SGML (زبان علامتگذاری عمومی استاندارد - که زبان استاندارد وب یعنی HTML نیز جزیی از آن محسوب می‌شود) است. برای مثال پیام

```
The <bold> time </bold> has come the <italic> walrus </italic> said ...
```

به این صورت به نمایش در خواهد آمد:

نوع	زیرنوع	توضیح
Text	Plain	متن فرمت نشده
	Enriched	متن با فرمت ساده
Image	Gif	تصاویر با فرمت GIF
	Jpeg	تصاویر با فرمت JPEG
Audio	Basic	صدا
Video	Mpeg	تنظیم با فرمتMPEG
Application	Octet-stream	توالی بایت تفسیر نشده
	Postscript	سند چاپی با فرمت پست اسکریپت
Message	Rfc822	پیام RFC 822
	Partial	سند چند تکه شده
	External-body	پیام باید از اینترنت گرفته شود
Multipart	Mixed	پیامی با چند بخش مستقل
	Alternative	یک پیام با فرمتهای مختلف
	Parallel	بخشهای پیام باید همزمان دیده شوند
	Digest	هر بخش یک پیام RFC 822 کامل است

شکل ۷-۱۲. نوع‌ها و زیرنوع‌های MIME تعریف شده در RFC 2045 .

The **time** has come the *walrus* said ...

فرمت کردن پیام بطور کامل بر عهدهٔ سیستم گیرنده است، و نباید انتظار داشته باشید چنین پیامی در تمام سیستمها یکسان (و آنطوری که شما تصور می‌کنید) دیده شود. گاهی یک سیستم معنای کاری را که شما خواسته‌اید نمی‌فهمد، و بجای آن کار دیگری انجام می‌دهد.

بعد از رواج وب، زیرنوع جدیدی بنام *text/html* (در RFC 2854) اضافه شد، که اجازه می‌داد صفحات وب از طریق ایمیل‌های RFC 822 فرستاده شوند. در RFC 3023 نیز زیرنوع دیگری (*text/xml*) برای ارسال پیامهای XML تعریف شده است. در ادامهٔ این فصل با HTML و XML بیشتر آشنا خواهید شد.

نوع MIME بعدی *image* است، که برای ارسال تصاویر ثابت بکار می‌رود. امروزه فرمتهای بسیار متنوعی برای ذخیره و ارسال کردن تصاویر (با فشرده‌سازی یا بدون آن) وجود دارد، که از میان آنها فرمتهای GIF و JPEG بسیار معروفند و تقریباً تمام مرورگرهای اینترنت از آنها پشتیبانی می‌کنند. (البته بعدها فرمتهای دیگری نیز به این لیست اضافه شد.)

نوع‌های *audio* و *video* بترتیب برای ارسال صدا و تصاویر متحرک هستند. توجه داشته باشید که نوع *video* فقط برای ارسال اطلاعات تصویری است، و اگر فایل شما دارای تراک صوتی هم باشد، باید آنها را جداگانه بفرستید. اولین فرمت ویدئویی که قابلیت ارسال از طریق ایمیل را پیداکرد، فرمت MPEG (گروه تخصصی تصاویر متحرک – Moving Picture Expert Group) بود؛ بعدها فرمتهای دیگر نیز اضافه شد. علاوه بر زیرنوع *audio/basic* ، در RFC 3003 زیرنوع دیگری (*audio/mpeg*) برای ارسال فایلهای صوتی MP3 تعریف شده است.

هر نوع فایل باینری دیگری که در دستجات بالا قرار نگیرد (و سیستم ایمیل نداند آنرا چگونه پردازش کند)، در ذیل نوع *application* دسته‌بندی خواهد شد. زیرنوع *octet-stream* فقط استریمی از بایتهاست (و سیستم هیچگونه تفسیری روی آنها نخواهد کرد). عامل کاربر بعد از دریافت این استریم، تنها کاری که می‌تواند انجام دهد ذخیره کردن آن بصورت یک فایل است – پس باید نام فایل را از کاربر بگیرد. پردازشهای بعدی نیز بر عهدهٔ کاربر است.

زیرنوع تعریف شدهٔ دیگر *postscript* است، که به زبان پُست اسکریپت (زبان توصیف صفحات چاپی، از شرکت Adobe) مربوط می‌شود. بسیاری از چاپگرهای امروزی دارای مفسرهای پُست اسکریپت هستند. با اینکه عامل کاربر می‌تواند تفسیر فایلهای پُست اسکریپت را بر عهدهٔ برنامه‌های خارجی بگذارد، اما این کار خالی از خطر نیست. پُست اسکریپت یک زبان برنامه‌نویسی کامل است، و یک برنامه‌نویس (خودآزار) می‌تواند با صرف وقت کافی حتی یک کامپایلر C یا سیستم مدیریت پایگاه داده با آن بنویسد. عامل کاربر برای نمایش محتویات فایل پُست‌اسکریپت، در واقع برنامه‌ای راکه در دل پیام فرستاده شده اجرا می‌کند. چنین برنامه‌ای حتی می‌تواند (در کنار نمایش یک متن ساده) فایلهای کاربر را تغییر داده، یا آنها را پاک کند (و یا دهها کار ناجور دیگر انجام دهد).

نوع *message* اجازه می‌دهد تا یک پیام را بطور کامل در دل پیام دیگر جای دهیم. این نوع بویژه برای هدایت پیامها (message forwarding) مفید است. برای قرار دادن یک ایمیل RFC 822 در دل پیام دیگر، می‌توان از زیرنوع *message/rfc822* استفاده کرد.

با زیرنوع *partial* می‌توان یک پیام را چند تکه کرده، و هر تکه را در دل یک ایمیل جداگانه قرار داد (که این برای پیامهای خیلی بزرگ مناسب است). در مقصد سیستم ایمیل می‌تواند با استفاده از پارامترهای این زیرنوع، پیام تکه‌تکه شده را دوباره به هم بچسباند.

بالاخره، از زیرنوع *external-body* می‌توان برای پیامهای بسیار بزرگ (مانند فیلمهای ویدئویی) استفاده کرد؛ یعنی بجای قرار دادن فایل MPEG در دل پیام، آدرس FTP آن نامه در نوشته می‌شود، و عامل کاربر می‌تواند در موقع نیاز به این آدرس مراجعه کرده و فایل را بخواند. این زیرنوع بخصوص در مواردی که بخواهیم فایل بزرگی را برای یک لیست پستی بفرستیم، بسیار ایده‌آل است، چون احتمال اینکه همهٔ اعضاء لیست پستی این فایل را بخواهند چندان زیاد نیست (فرستادن آگهی‌های تبلیغاتی یکی از این موارد است).

نوع آخر *multipart* است، که اجازه می‌دهد یک پیام چندین بخش داشته باشد (بخشهایی که ابتدای و انتهای آنها کاملاً مشخص است). در زیرنوع *mixed* این بخشها می‌توانند کاملاً متفاوت باشند، بدون اینکه نیازی به ساختار اضافی وجود داشته باشد. در بسیاری از برنامه‌های ایمیل یک پیام ساده می‌تواند چندین پیوست (attachment) از انواع مختلف داشته باشد، که این پیوستها با استفاده از نوع *multipart* فرستاده می‌شوند.

بر خلاف *multipart* ، در زیرنوع *alternative* یک پیام واحد به چندین فرمت (مانند متن ساده، متن فرمت‌دار، و یا پست اسکریپت) فرستاده می‌شود، که عامل کاربر گیرنده می‌تواند بسته به امکانات خود از یکی از این انواع استفاده کند. این انواع باید از ساده‌ترین به پیچیده‌ترین مرتب شوند، تا حتی کاربران غیر MIME بتوانند پیامها را بخوانند.

از زیرنوع *alternative* برای ارسال پیام به زبانهای مختلف هم می‌توان استفاده کرد. (سنگ روزِتا را می‌توان یکی از قدیمیترین پیامهای *multipart/alternative* دانست – سنگ روزِتا سنگ‌نبشته‌ای است که توسط سپاهیان ناپلئون در منطقه‌ای به همین نام در مصر کشف شد، و در آن یک پیام واحد به زبانهای هیروگلیف و یونانی نوشته شده بود؛ با استفاده از همین سنگ‌نبشته بود که شامپولیون باستان‌شناس فرانسوی توانست معمای خط هیروگلیف را بعد از قرنها کشف کند.)

در شکل ۷–۱۳ یک پیام *multipart* را ملاحظه می‌کنید. در این مثال، یک پیام تبریک تولد به دو صورت متن و صوت فرستاده شده است. اگر کامپیوتر گیرنده کارت صوتی داشته باشد، برنامهٔ عامل کاربر فایل صوتی *birthday.snd* را از شبکه گرفته، و پخش می‌کند. اما اگر چنین نباشد، فقط متن شعر را نمایش خواهد داد. دقت کنید که بخشهای پیام با دو - (خط تیره) و رشته‌ای از حروف (که نرم‌افزار آنها را تولید کرده، و در قسمت *boundary* مشخص شده) از هم جدا می‌شوند.

همچنین توجه کنید که سرآیند *Content-Type* در سه نقطه از این پیام آمده است. در بالای پیام، سرآیند

```
From: elinor@abcd.com
To: carolyn@xyz.com
MIME-Version: 1.0
Message-Id: <0704760941.AA00747@abcd.com>
Content-Type: multipart/alternative; boundary=qwertyuiopasdfghjklzxcvbnm
Subject: Earth orbits sun integral number of times

This is the preamble. The user agent ignores it. Have a nice day.

--qwertyuiopasdfghjklzxcvbnm
Content-Type: text/enriched

Happy birthday to you
Happy birthday to you
Happy birthday dear <bold> Carolyn </bold>
Happy birthday to you

--qwertyuiopasdfghjklzxcvbnm
Content-Type: message/external-body;
        access-type="anon-ftp";
        site="bicycle.abcd.com";
        directory="pub";
        name="birthday.snd"

content-type: audio/basic
content-transfer-encoding: base64
--qwertyuiopasdfghjklzxcvbnm--
```

شکل ۷-۱۳. یک پیام *multipart*، محتوی متن و صوت.

Content-Type می‌گوید که این یک پیام *multipart/alternative* است. در دو *Content-Type* بعدی نوع و زیرنوع هر بخش مشخص می‌شود. در آخرین *Content-Type* نیز کُدگذاری فایل صوتی را مشخص کرده‌ایم، چون هر چیزی که متن ASCII ۷بیتی نباشد، باید دارای کُدگذاری مشخص باشد.

پیامهای *multipart* دارای دو زیرنوع دیگر نیز می‌توانند باشند. از زیرنوع *parallel* وقتی استفاده می‌کنیم که بخواهیم تمام بخشهای پیام همزمان «مشاهده» شوند. برای مثال، فیلمهای ویدئویی دارای کانالهای تصویری و صوتی مجزا هستند، که باید همزمان پخش شوند (نه بدنبال هم).

زیرنوع *digest* وقتی بکار برده می‌شود که بخواهیم تعدادی پیام را در یک پیام مرکب بسته‌بندی کنیم. برای مثال، در **گروههای مباحثه** (discussion group) اینترنتی معمولاً چندین پیام که از اعضای مختلف جمع‌آوری شده، در یک پیام *multipart/digest* به سایر اعضای گروه فرستاده می‌شود.

۷-۲-۴ انتقال پیام

سیستم **انتقال پیام** (message transfer) با ارسال پیام از فرستنده به گیرنده سروکار دارد. ساده‌ترین راه برای این کار، برقراری یک اتصال مستقیم از ماشین فرستنده به ماشین گیرنده، و انتقال پیام است. بعد از بررسی این روش، به مواردی می‌پردازیم که چنین کاری امکان ندارد، و سپس برای آن موارد نیز راه‌حل‌هایی نشان خواهیم داد.

SMTP - پروتکل سادهٔ انتقال نامه

در اینترنت، انتقال ایمیل با برقراری یک اتصال TCP از ماشین مبدأ به پورت 25 ماشین مقصد صورت می‌گیرد. برنامه‌ای که به این پورت گوش می‌کند، دیمون SMTP (پروتکل سادهٔ انتقال نامه – Simple Mail Transfer Protocol) نام دارد. این دیمون اتصالات ورودی را پذیرفته، و پیامها را در صندوق پستی مربوطه کپی می‌کند. اگر گیرنده نتواند پیامی را تحویل گیرد، یک گزارش خطا حاوی اولین بخش از پیام مزبور به فرستنده برمی‌گرداند.

SMTP یک پروتکل سادهٔ ASCII است. بعد از برقراری اتصال TCP ، ماشین فرستنده (که نقش مشتری را بازی می‌کند) منتظر می‌ماند تا ماشین گیرنده (که نقش سرویس‌دهنده را بازی خواهد کرد) شروع به صحبت کند. در شروع، سرویس‌دهنده یک خط متن فرستاده و ضمن معرفی خود، اعلام می‌کند که آیا آماده دریافت ایمیل هست یا خیر. اگر سرویس‌دهنده آماده نباشد، مشتری ارتباط را قطع کرده، و بعداً دوباره سعی خواهد کرد.

اگر سرویس‌دهنده آمادهٔ دریافت باشد، مشتری اعلام می‌کند که پیام از چه کسی می‌آید و به چه کسی باید تحویل شود. اگر چنین دریافت‌کننده‌ای در ماشین سرویس‌دهنده وجود داشته باشد، سرویس‌دهنده از مشتری می‌خواهد که پیام را بفرستد. پس از آن که مشتری پیام را فرستاد، سرویس‌دهنده دریافت آن را تصدیق می‌کند. سرویس‌دهنده هیچ تلاشی برای چک کردن جمع تطبیقی انجام نخواهد داد، چون TCP سالم بودن ارتباط را تضمین می‌کند. اگر باز هم پیامی وجود داشته باشد، پس از آن فرستاده می‌شود. وقتی تمام ایمیل‌ها (در هر دو جهت) مبادله شد، ارتباط قطع می‌شود. در شکل ۷-۱۴ مکالمهٔ سرویس‌دهنده و مشتری (منجمله کُدهای عددی SMTP) برای ارسال ایمیل شکل ۷-۱۳ را ملاحظه می‌کنید. خطوطی که توسط مشتری ارسال شده‌اند، را با C: ، و آنهایی که توسط سرویس‌دهنده فرستاده شده‌اند، را با S: مشخص کرده‌ایم.

کمی توضیح دربارهٔ شکل ۷-۱۴ می‌تواند مفید باشد. اولین پیام مشتری *HELO* است. این کلمه در واقع مخفف چهارحرفی «سلام» *(HELLO)* است، و پیداست که از همهٔ چهارحرفی‌ها به آن شبیه‌تر است. اینکه چرا باید از کلمه‌های چهارحرفی استفاده کنیم، در غبار زمان گم شده، ولی این رسم همچنان پا برجاست.

از آنجائیکه فقط یک گیرنده وجود دارد، در اینجا فقط یک دستور *RCPT* دیده می‌شود، ولی می‌توان در آن واحد یک پیام را به چندین گیرنده فرستاد (که قبول یا رد درخواست برای هر یک جداگانه انجام خواهد شد).

با اینکه دستورات چهارحرفی از طرف مشتری ثابت و مشخص هستند، پاسخ سرویس‌دهنده فقط یک سری کُدهای عددی است. در واقع همین کُدهاست که اهمیت دارد، و هر سیستم می‌تواند برای هر کُد توضیح خاص خود را داشته باشد.

اجازه دهید برای درک بهتر پروتکل SMTP ، کمی دربارهٔ فرآیند کار توضیح دهیم. قبل از هر چیز سراغ کامپیوتری بروید که به اینترنت دسترسی دارد. سپس دستور زیر را در خط فرمان وارد کنید (در این مثال فرض کرده‌ایم سیستم عامل یونیکس است)

```
telnet mail.isp.com 25
```

(بجای mail.isp.com آدرس IP یا نام DNS سیستم ایمیل خود را وارد کنید.) در سیستمهای ویندوز، پنجرهٔ Start|Run را باز کرده و دستور فوق را در آنجا وارد کنید. این دستور یک اتصال telnet (یعنی، TCP) با پورت 25 ماشین مشخص شده برقرار می‌کند. پورت 25 پورت SMTP است (شکل ۶-۲۷ را ببینید). پاسخی که دریافت خواهید کرد، احتمالاً چیزی شبیه زیر است:

```
Trying 192.30.200.66...
Connection to mail.isp.com
Escape character is '^]'
220 mail.isp.com Smail #74 ready at Thu, 25 Mar 2003 13:26 +0200
```

```
                S: 220 xyz.com SMTP service ready
C: HELO abcd.com
                S: 250 xyz.com says hello to abcd.com
C: MAIL FROM: <elinor@abcd.com>
                S: 250 sender ok
C: RCPT TO: <carolyn@xyz.com>
                S: 250 recipient ok
C: DATA
                S: 354 Send mail; end with "." on a line by itself
C: From: elinor@abcd.com
C: To: carolyn@xyz.com
C: MIME-Version: 1.0
C: Message-Id: <0704760941.AA00747@abcd.com>
C: Content-Type: multipart/alternative; boundary=qwertyuiopasdfghjklzxcvbnm
C: Subject: Earth orbits sun integral number of times
C:
C: This is the preamble. The user agent ignores it. Have a nice day.
C:
C: --qwertyuiopasdfghjklzxcvbnm
C: Content-Type: text/enriched
C:
C: Happy birthday to you
C: Happy birthday to you
C: Happy birthday dear <bold> Carolyn </bold>
C: Happy birthday to you
C:
C: --qwertyuiopasdfghjklzxcvbnm
C: Content-Type: message/external-body;
C:         access-type="anon-ftp";
C:         site="bicycle.abcd.com";
C:         directory="pub";
C:         name="birthday.snd"
C:
C: content-type: audio/basic
C: content-transfer-encoding: base64
C: --qwertyuiopasdfghjklzxcvbnm
C: .
                S: 250 message accepted
C: QUIT
                S: 221 xyz.com closing connection
```

شکل ۷-۱۴. انتقال یک پیام از *elinor@abcd.com* به *carolyn@xyz.com* .

سه خط اول مربوط به برنامهٔ telnet هستند، و می‌گویند چه اتفاقی در حال افتادن است. خط آخر از سرویس‌دهنده SMTP آمده، و آمادگی آنرا برای دریافت ایمیل از طرف شما اعلام می‌کند. برای اینکه ببینید سرویس‌دهنده ایمیل چه فرمانهایی را قبول می‌کند، دستور زیر را وارد کنید

HELP

از اینجا به بعد با چیزی شبیه شکل ۷-۱۴ روبرو خواهید بود. پروتکلهای ASCII در اینترنت بسیار رایج هستند، چون تست و دیباگ آنها بسیار ساده است. فرمان دادن به این پروتکلها بسیار آسان است، و پاسخ آنها را نیز براحتی می‌توان درک کرد.

با اینکه پروتکل SMTP بخوبی تعریف شده است، اما برخی مشکلات کوچک نیز می‌تواند بروز کند. یکی از این مشکلات طول پیام است: در برخی از سیستمهای ایمیل قدیمی طول پیام نباید از 64 KB تجاوز کند. مشکل دیگر زمانهای متفاوت انتظار برای پاسخ در دو سمت مقابل است. اگر زمان انتظار برای دریافت پاسخ (timeout) در سمت سرویس‌دهنده و مشتری متفاوت باشد، این احتمال هست که یکی از آنها تسلیم شده (در حالیکه دیگری هنوز منتظر است) و ارتباط را بطور نامنتظره قطع کند. مشکل دیگر بروز توفانهای ایمیل است. اگر ماشین ۱ دارای یک لیست پستی بنام A ، و ماشین ۲ دارای یک لیست پستی بنام B باشند، و هر لیست عضو لیست مقابل باشد، توفانی پایان‌ناپذیر از ایمیلهای تکراری به راه خواهد افتاد (که فقط با دخالت سرپرست سیستم می‌تواند قطع شود).

برای حل این قبیل مشکلات، ویرایش گسترش‌یافتهٔ SMTP (موسوم به ESMTP) در RFC 2821 تعریف شده است. اگر یک مشتری بخواهد بجای SMTP از این پروتکل استفاده کند، باید در شروع کار بجای *HELO* دستور *EHLO* را بکار ببرد. اگر دستور *EHLO* از طرف سرویس‌دهنده پذیرفته نشود، مشتری می‌فهمد که با یک سرویس‌دهنده *SMTP* معمولی سروکار دارد و باید از همان روش سابق استفاده کند. اما اگر *EHLO* پذیرفته شد، مشتری اجازه دارد دستورات این پروتکل را بکار ببرد.

۵-۲-۷ تحویل نهایی

تا اینجا فرض ما بر این بود که تمام کاربران شبکه ماشینهایی با قابلیت ارسال و دریافت ایمیل دارند. همانطور که دیدید، فرستنده باید یک اتصال TCP به ماشین گیرنده برقرار کرده، و ایمیل خود را به آن بفرستد. این روش سالها بخوبی کار می‌کرد، چون تمام ماشینهای آرپانت (و بعدها اینترنت) کامپیوترهایی بودند که همیشه روی خط بودند و می‌توانستند اتصالات TCP را بپذیرند.

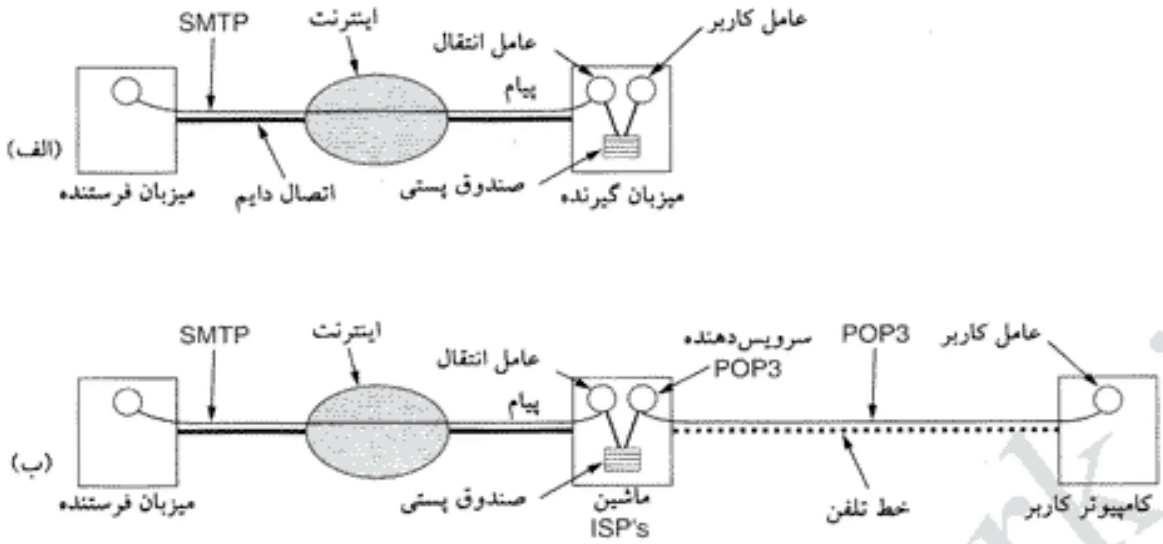
این وضعیت با ظهور کاربرانی که برای دسترسی اینترنت مجبور بودند از طریق مودم و با واسطهٔ یک ISP اقدام کنند، تغییر کرد. مشکل این بود که: اگر در لحظه‌ای که Elinor می‌خواهد برای Carolyn ایمیل بفرستد، Carolyn روی خط نباشد، چه خواهد شد؟ در این حالت Elinor قادر به برقراری ارتباط TCP با Carolyn نیست، و نمی‌تواند پروتکل SMTP را اجرا کند.

ساده‌ترین راه‌حل آن است که یک عامل انتقال پیام در محل ISP ایجاد شود، و ایمیلها را پس از دریافت از فرستنده در صندوق پستی گیرنده (که در همان ISP قرار دارد) ذخیره کند. از آنجائیکه این عامل انتقال پیام می‌تواند ۲۴ ساعته روی خط باشد، همیشه می‌توان به آن ایمیل فرستاد.

POP3

متأسفانه این راه‌حل یک مشکل کوچک دارد: کاربران چگونه باید ایمیل‌های خود را از عامل انتقال مستقر در ISP بگیرند؟ برای حل این مسئله به پروتکل جدیدی نیاز داریم که PC کاربر را به یک عامل انتقال پیام (از ISP) تبدیل کند. یکی از این پروتکل‌ها، که در RFC 1939 تعریف شده، POP3 (**پروتکل دفترپستی‌ویرایش ۳** – Post Office Protocol Version 3) نام دارد.

به شکل ۷-۱۵ نگاه کنید؛ در تصویر (الف) وضعیتی که باید وجود داشته باشد، را می‌بینید: فرستنده و گیرنده



شکل ۷-۱۵. (الف) وضعیتی که فرستنده و گیرنده دسترسی دائم به اینترنت دارند، و عامل کاربر و عامل انتقال پیام هر دو روی یک ماشین اجرا می‌شوند. (ب) خواندن ایمیل در حالتی که گیرنده از طریق تلفن و با واسطهٔ یک ISP به اینترنت وصل می‌شود.

هر دو دائماً روی خط هستند. در شکل ۷-۱۵ (ب) اوضاع به این خوبی نیست: فرستنده همیشه روی خط است، ولی گیرنده چنین نیست.

پروتکل POP3 کار خود را زمانی شروع می‌کند که کاربر برنامهٔ ایمیل‌خوان را باز می‌کند. برنامهٔ ایمیل‌خوان با ISP تماس گرفته (البته اگر این تماس از قبل وجود نداشته باشد)، و یک اتصال TCP به پورت 110 ماشین عامل انتقال پیام برقرار می‌کند. بعد از برقراری ارتباط، پروتکل POP3 سه مرحله را طی می‌کند:

۱. احراز هویت (authorization) .
۲. تراکنش (transaction) .
۳. به روز در آوردن (update) .

در مرحلهٔ احراز هویت کاربر باید هویت واقعی خود را به عامل انتقال پیام بشناساند. در مرحلهٔ تراکنش، کاربر ایمیل‌های خود را از صندوق پستی خوانده، و آنها را برای حذف شدن علامتگذاری می‌کند. این ایمیل‌ها در مرحلهٔ بعد (به روز در آوردن) حذف می‌شوند.

برای دیدن مراحل کار، فرمان زیر را اجرا کنید:

```
telnet mail.isp.com 110
```

(که در آن *mail.isp.com* نام DNS سرویس‌دهنده ایمیل ISP است.) برنامهٔ telnet یک اتصال TCP با پورت 110 (که سرویس‌دهنده POP3 به آن گوش می‌کند) برقرار می‌سازد. بعد از برقراری ارتباط، سرویس‌دهنده یک پیام ASCII فرستاده، و آمادگی خود را اعلام می‌کند. این پیام معمولاً با +OK شروع می‌شود، و جملهٔ کوتاه دیگری بعد از آن می‌آید. در شکل ۷-۱۶ مکالمهٔ یک مشتری با سرویس‌دهنده POP3 را ملاحظه می‌کنید (در اینجا هم پیامهای مشتری را با C: و پاسخهای سرویس‌دهنده را با S: مشخص کرده‌ایم).

در مرحلهٔ احراز هویت، مشتری ابتدا نام‌کاربر (username) و سپس کلمهٔ‌عبور (password) خود را می‌فرستد. اگر مشتری بتواند این مرحله را با موفقیت پشت سر گذارد، می‌تواند با ارسال دستور *LIST* فهرستی از پیامهای موجود در صندوق پستی خود را دریافت کند (یک پیام در هر خط، که طول آنها نیز مشخص شده است).

پایان این لیست با یک نقطه (.) مشخص می‌شود.

```
                S: +OK POP3 server ready
C: USER carolyn
                S: +OK
C: PASS vegetables
                S: +OK login successful
C: LIST
                S: 1 2505
                S: 2 14302
                S: 3 8122
                S: .
C: RETR 1
                S: (sends message 1)
C: DELE 1
C: RETR 2
                S: (sends message 2)
C: DELE 2
C: RETR 3
                S: (sends message 3)
C: DELE 3
C: QUIT
                S: +OK POP3 server disconnecting
```

شکل ۷-۱۶. استفاده از POP3 برای گرفتن پیامهای ایمیل.

پس از آن مشتری می‌تواند برای دریافت پیامها از فرمان *RETR* استفاده کرده، و آنها را با *DELE* برای حذف علامتگذاری کند. وقتی تمام پیامها دریافت (و احتمالاً برای حذف علامتگذاری) شدند، مشتری می‌تواند با فرمان *QUIT* وارد مرحلهٔ بعدی (به روز در آوردن) شود. بعد از آن که سرویس‌دهنده پیامهای علامتگذاری شده را حذف کرد، یک پیام تصدیق به مشتری فرستاده و ارتباط TCP را قطع می‌کند.

با اینکه پروتکل POP3 می‌تواند ایمیل‌ها را بصورت دسته‌جمعی یا تکی گرفته و آنها را روی سرویس‌دهنده‌باقی بگذارد، اغلب برنامه‌های ایمیل همه چیز را از روی سرویس‌دهنده خوانده و سپس صندوق پستی را خالی می‌کنند. در این حالت تنها کپی پیامها در دست خود کاربر است، و اگر روزی کامپیوتر وی صدمه ببیند، همهٔ ایمیل‌ها از بین خواهند رفت.

اجازه دهید یک بار دیگر روش ارسال و دریافت ایمیل را برای کاربران ISP بطور خلاصه مرور کنیم. Elinor در برنامهٔ ایمیل خود (یعنی، عامل کاربر) یک نامه برای Carolyn می‌نویسد، و روی دکمهٔ Send کلیک می‌کند. برنامهٔ ایمیل این پیام را گرفته و به عامل انتقال پیام ISP (که Elinor مشتری آن است) تحویل می‌دهد. عامل انتقال پیام آدرس گیرندهٔ نامه (*carolyn@xyz.com*) را خوانده، و از یک DNS کمک می‌گیرد تا رکورد *MX* ناحیهٔ *xyz.com* را پیدا کند. درپاسخ به این درخواست، DNS نام DNS سرویس‌دهنده ایمیل ناحیهٔ *xyz.com* را برمی‌گرداند. عامل انتقال پیام دوباره به کمک همان DNS آدرس IP این سرویس‌دهنده را درخواست می‌کند. پس از دریافت این آدرس IP ، عامل انتقال پیام یک اتصال TCP به پورت 25 (سرویس‌دهنده SMTP) برقرار می‌کند، و با استفاده از دستورات SMTP (مانند آنچه در شکل ۷-۱۴ دیدید) پیام را به صندوق پستی Carolyn فرستاده، و سپس ارتباط TCP را قطع می‌کند.

پس از مدتی، Carolyn (از خواب بیدار شده و) PC خود را روشن کرده، و مثل همیشه قبل از هر کاری به ISP وصل می‌شود تا ایمیل‌هایش را چک کند. برنامهٔ ایمیل در بدو شروع یک اتصال TCP به پورت 110 سرویس‌دهنده ایمیل ISP (سرویس‌دهنده POP3) برقرار می‌کند. نام DNS یا آدرس IP این ماشین در همان بدو عضویت Carolyn در ISP در اختیار وی قرار داده می‌شود، تا آنرا در برنامهٔ ایمیل خود وارد کند (و معمولاً نیازی به پا در میانی DNS نیست). پس از برقراری اتصال به پورت 110 ، برنامهٔ ایمیل پروتکل POP3 را اجرا کرده، و (مانند آنچه در شکل ۷-۱۶ دیدید) پیامهای رسیده را از صندوق پستی Carolyn خوانده و در کامپیوتر وی ذخیره می‌کند (در پایان کار هم، اتصال TCP را قطع می‌کند). در اینجا می‌توان ارتباط تلفنی با ISP را هم قطع کرد (که البته نباید در این کار عجله کنید، چون برای فرستادن پاسخ نامه‌ها باز هم به آن احتیاج پیدا خواهد کرد).

IMAP

برای کاربری که فقط با یک ISP کار می‌کند و همیشه هم از یک PC به آن وصل می‌شود، POP3 بهترین گزینه است (بخصوص که ساده و کارآمد هم هست). ولی در دنیای کامپیوتر اصلی بدیهیست که می‌گوید، «وقتی چیزی دارد خوب کار می‌کند، بلافاصله یکی پیدا می‌شود که بیشتر می‌خواهد.» برای ایمیل هم همین اتفاق افتاد. برای مثال، خیلی از مردم هستند که فقط یک آدرس ایمیل دارند، و می‌خواهند هر کجا که هستند (در خانه، در مدرسه، در محل کار، و یا در سفر) با همان آدرس کار کنند. با اینکه POP3 می‌تواند از عهدهٔ این کار بر آید، اما کاربر بزودی متوجه می‌شود که ایمیل‌هایش روی چندین کامپیوتر پراکنده شده است (کامپیوترهایی که شاید بعضی از آنها حتی متعلق به وی نباشد).

این مشکل POP3 منجر به ارائهٔ راه‌حلی بنام IMAP (پروتکل دسترسی پیام‌اینترنتی – Internet Message Access Protocol) شد، که در RFC 2060 تعریف شده است. بر خلاف POP3 که اساساً فرض می‌کند کاربر تمام پیامهایش را به کامپیوتر خود منتقل کرده و سپس ارتباط با اینترنت را قطع می‌کند، IMAP پیامها را برای همیشه روی کامپیوتر سرویس‌دهنده نگه داشته و آنها را در چندین صندوق پستی حفظ می‌کند. IMAP دارای مکانیزمهای پیشرفته‌ای برای خواندن پیامهاست، که حتی اجازه می‌دهند کاربر فقط بخشهایی از یک پیام بخواند؛ فقط تصور کنید که یک پیام مهم برای کاربر بیچارهٔ ما – که با یک مودم کُند عهد بوق به اینترنت وصل می‌شود – آمده، و این پیام ضمن داشتن متنی مهم دارای یک پیوست بزرگ نیز هست – IMAP به کاربر ما اجازه می‌دهد تا فقط متن پیام را خوانده و از خیر پیوست آن بگذرد. از آنجائیکه در IMAP فرض بر آن است که پیامها به کامپیوتر کاربر منتقل نمی‌شوند، مکانیزمهایی برای نوشتن ایمیل، از بین بردن آنها، و یا مدیریت پیامهای رسیده (مانند دسته‌بندی آنها بر حسب فرستنده) روی سرویس‌دهنده ایمیل در نظر گرفته شده است.

یکی از قابلیتهای جالب IMAP دسته‌بندی و نمایش پیامهای رسیده بر حسب فرستندهٔ ایمیل – یا ویژگیهای دیگر – است (بر خلاف شکل ۷-۸ که تمام ایمیل‌ها پشت سر هم ردیف می‌شوند). IMAP (برخلاف POP3) فقط پروتکلی برای دریافت ایمیل نیست، بلکه می‌تواند ارسال نامه‌ها را هم انجام دهد. روش کار IMAP بسیار شبیه POP3 (شکل ۷-۱۶) است، با این تفاوت که دستورات بسیار متنوعتری دارد. سرویس‌دهنده IMAP به پورت 143 گوش می‌کند. در شکل ۷-۱۷ مقایسه‌ای بین POP3 و IMAP آورده شده است. اما همین جا باید تذکر داد که تمام ISP ها (و همچنین برنامه‌های ایمیل) از هر دو پروتکل پشتیبانی نمی‌کنند (هنگام انتخاب ISP و برنامهٔ ایمیل دقت کنید که از چه پروتکلهایی پشتیبانی می‌کنند).

امکانات سیستم تحویل نامه

اغلب سیستمهای ایمیل، صرفنظر از اینکه از POP3 یا IMAP برای گرفتن نامه‌ها استفاده کنید، امکاناتی برای

پردازش پیامها دارند. یکی از این امکانات فیلتر کردن پیامهاست. این فیلترها قواعدی هستند که روی ایمیلهای رسیده عمل می‌کنند. هر قاعده یک شرط دارد و عملی را انجام می‌دهد. برای مثال، یک قاعده می‌تواند بگوید «اگر پیامی از رئیس رسید، آنرا در صندوق شماره ۱ قرار بده.» یا «اگر پیام از گروهی مشخصی از دوستان بود، آنرا در صندوق شماره ۲ قرار بده.» و یا «اگر کلمهٔ خاصی در موضوع پیام بود، آنرا بکلی دور بینداز.»

ویژگی	POP3	IMAP
سطح تعریف پروتکل	RFC 1939	RFC 2060
پورت	110	143
محل ذخیره شدن ایمیل	کاربر PC	سرویس دهنده
محل خوانده شدن ایمیل	خارج خط	روی خط
زمان اتصال	کم	زیاد
استفاده از منابع سرویس دهنده	حداقل	گسترده
صندوق پستی های متعدد	خیر	بلی
مسئول گرفتن پشتیبان	کاربر	ISP
مناسب برای کاربران سیار	خیر	بلی
کنترل بار کردن محتویات	کم	زیاد
بار کردن قسمتی از پیامها	خیر	بلی
مشکل محدودیت دیسک	خیر	گاهی
پیاده سازی ساده است	بلی	خیر
پشتیبانی گسترده	بلی	در حال و شد

شکل ۷-۱۷. مقایسه‌ای بین POP3 و IMAP .

برخی از ISP ها فیلترهایی دارند که بطور خودکار ایمیلهای رسیده را به دو دستهٔ «مهم» و «هرز» (spam یا junk - نامه‌هایی که فقط بدرد سطل آشغال می‌خورند) تقسیم کرده، و آنها را در صندوقهای مخصوص قرار می‌دهند. این فیلترها ابتدا با چک کردن فرستندهٔ نامه‌ها شروع می‌کنند (تا مطمئن شوند از مردم‌آزارهای حرفه‌ای نباشند). اگر چند صد کاربر یک ISP نامه‌هایی با یک موضوع دریافت کنند، فرستندهٔ آن باحتمال زیاد یک هرزنویس است. برای تشخیص این قبیل نامه‌ها تکنیکهای دیگری نیز وجود دارد.

یک دیگر از امکانات سیستمهای تحویل ایمیل هدایت (موقتی) نامه‌های رسیده برای یک کاربر به آدرسی دیگر است. این آدرس حتی می‌تواند شماره‌ای در یک سیستم فراخوان (pager) باشد، که در این حالت کاربر بلافاصله بعد از دریافت هر ایمیل موضوع آنرا در دستگاه فراخوان خود مشاهده خواهد کرد.

امکان دیگر پاسخ خودکار در صورت عدم حضور در محل است (که به آن دیمون تعطیلات - vacation daemon - می‌گویند). با این ویژگی فرد می‌تواند هنگام رفتن به تعطیلات یا مأموریت بطور خودکار به ایمیلهای رسیده پاسخی مانند زیر بدهد:

سلام، من تا ۲۴ ام آگوست در تعطیلات تابستانی هستم. امیدوارم شما هم تعطیلات خوبی داشته باشد.

در این قبیل پیامها حتی می‌توانید مشخص کنید که در صورت اضطرار با کجا باید تماس گرفته شود. در برخی از سیستمهای ایمیل، دیمون تعطیلات می‌تواند تشخیص دهد که قبلاً برای چه کسانی پیام فرستاده، تا از ارسال پیام تکراری برای آنها اجتناب شود. برخی از این دیمون‌ها حتی می‌توانند تشخیص دهند که نامهٔ وارده به یک لیست پستی فرستاده شده، که در این صورت از فرستادن جواب آماده خودداری خواهند کرد.

نویسندهٔ این کتاب شخصاً با یک سیستم پاسخ خودکار فوق‌العاده جالب از جانب شخصی روبرو شده است،

که ادعا می‌کرد روزی ۶۰۰ ایمیل می‌گیرد. (اجازه دهید برای حفظ حریم شخصی افراد، این فرد را با نام مستعار جان معرفی کنیم.)

جان یک روبات ایمیل در کامپیوتر خود نصب کرده که تمام پیامهای رسیده را چک می‌کند. اگر این ایمیل از کسی باشد که قبلاً با جان تماس نداشته (و بعبارت دیگر تازه‌وارد باشد)، پیام خودکاری برای وی ارسال می‌شود که (ضمن عذرخواهی از عدم امکان پاسخ فردی) سندی حاوی تمام اطلاعات موردنیاز (از جمله آدرس، شماره تلفن، شماره فکس، و طریقهٔ تماس با شرکت جان) در آن آمده است. از اطلاعات مفصل دیگری که جان دربارهٔ خود در این پاسخ خودکار آورده حرفی نمی‌زنم، اما بنظر می‌رسد که روش جان یکی از نمونه‌های افراط در استفاده از امکانات باشد.

پُست وب

آخرین مبحثی که در سیستمهای ایمیل می‌توان به آن اشاره کرد، **پُست‌وب** (Webmail) است. همانطور که شاید قبلاً دیده باشید، برخی از سایتهای معروف و بزرگ مانند هات‌میل (Hotmail.com) و یاهو (Yahoo.com) سرویسهای ایمیل مجانی در اختیار بازدیدکنندگان قرار می‌دهند. در این سیستمها یک عامل انتقال پیام معمولی وجود دارد، که به پورت 25 (پورت SMTP) گوش می‌کند. برای وصل شدن به، مثلاً، هات‌میل باید ابتدا رکورد *MX* آنرا بدست آورید؛ در یک سیستم یونیکس می‌توانید از دستور زیر استفاده کنید:

```
host -a -v hotmail.com
```

پس از آن، با فرض اینکه سرویس‌دهنده ایمیل هات‌میل *mx10.hotmail.com* نام داشته باشد، می‌توانید با دستور

```
telnet mx10.hotmail.com 25
```

یک اتصال TCP به پورت 25 آن برقرار کرده، و از فرمانهای SMTP برای ارسال پیامهای خود استفاده کنید. (تا اینجا که چیز غیرعادی وجود ندارد؛ فقط مواظب باشید که سر این کامپیوترها خیلی شلوغ است، و معمولاً باید چند بار سعی کنید تا بتوانید با آنها تماس بگیرید.)

بخش جالب در این سرویسها قسمت تحویل نامه است. معمولاً وقتی وارد صفحهٔ وب سرویس ایمیل می‌شوید، فرمی ظاهر می‌شود که باید نام‌کاربر و کلمهٔ عبور خود را در آن وارد کنید. وقتی دکمهٔ Sign In را کلیک می‌کنید، این اطلاعات به سرویس‌دهنده فرستاده شده و در آنجا با اطلاعات موجود تطبیق داده می‌شود. اگر هویت شما تأئید شود، سرویس‌دهنده صندوق پستی را یافته و محتویات آنرا بصورت یک صفحه‌ای شبیه شکل ۷-۸، ولی با فرمت HTML ، نمایش می‌دهد. اغلب امکانات یک برنامهٔ ایمیل (مانند خواندن نامه، نوشتن نامه، و حذف آنها) در این صفحهٔ وب نیز وجود دارد.

۳-۷ تارنمای جهانی - وب

تارنمای جهانی (World Wide Web) ، یا بطور مختصر **وب**، ساختاریست برای دسترسی به **سندهای پیوندشده** (linked documents) در اینترنت. ظرف ده سال، وب از یک ابزار ارتباطی فیزیکدانها به چیزی تبدیل شده که بسیاری از مردم آنرا همان «اینترنت» می‌دانند. علت اصلی محبوبیت وب در ظاهر گرافیکی آن ریشه دارد، که باعث شده تا میلیونها کاربر تازه‌کار بتوانند بسادگی از آن استفاده کنند. حجم فوق‌العادهٔ اطلاعات موجود در وب را نیز می‌توان یکی دیگر از علل موفقیت آن دانست.

وب (که به WWW نیز معروف است) به سال ۱۹۸۹ در مرکز اروپایی فیزیک هسته‌ای موسوم به CERN متولد شد. در این مرکز دهها تیم تحقیقاتی از سراسر اروپا به کار روی فرضیه‌های فیزیک ذرات مشغول هستند.

اغلب این آزمایشات چنان پیچیده‌اند که دهها دانشمند از چندین کشور مختلف سالها روی آنها کار می‌کنند. همین پراکندگی دانشمندان این مرکز در کشورهای مختلف، و لزوم ارتباط پیوستهٔ آنها منجر به تولد وب شد.

ایدهٔ سندهای پیوند شده را اولین بار یکی از فیزیکدانان CERN بنام تیم برنرز-لی در مارس ۱۹۸۹ مطرح کرد. اولین نمونهٔ این برنامه ۱۸ ماه بعد عملیاتی شد، و در کنفرانس Hypertext '91 که در دسامبر ۱۹۹۱ در سان‌آنتونیو، تگزاس بر پا شده بود، به نمایش در آمد.

این نمایش و قابلیتهای بالقوهٔ یک **مرورگرآبرمتن** (hypertext browser) نظر محققان را بخود جلب کرد، و یکی از همین افراد بنام مارک آندرسن از دانشگاه ایلینویز اولین مرورگر گرافیکی را در فوریه ۱۹۹۳ به بازار عرضه کرد - این مرورگر **موزائیک** (Mosaic) نام داشت. موزائیک چنان موفقیتی بدست آورد که یک سال بعد آندرسن شرکتی بنام **نت‌اسکیپ**(Netscape) ، با هدف توسعهٔ نرم‌افزارهای وب، تأسیس کرد. وقتی نت‌اسکیپ در سال ۱۹۹۵ وارد بورس شد، سرمایه‌گذاران به تصور تولد یک میکروسافت دیگر ۱/۵ میلیارد دلار بابت خرید سهام آن پول پرداخت کردند - و این همه پول (که یک رکورد محسوب می‌شد) در شرایطی پرداخت شد که نت‌اسکیپ شرکت کوچکی بود با فقط یک محصول، و حتی اعلام کرده بود که تا مدتها هیچگونه سوددهی نخواهد داشت. طی سه سال بعد نت‌اسکیپ (با مرورگر ناویگیتور - Navigator) و میکروسافت (با مرورگر اینترنت‌اکسپلورر - Internet Expolrer) درگیر جنگی بودند که به «جنگ مرورگرها» معروف شد، و در آن هر شرکت سعی می‌کرد با ارائهٔ محصول بهتر دیگری را از میدان بدر کند. در سال ۱۹۹۸، شرکت America Online نت‌اسکیپ را به قیمت ۴/۲ میلیارد دلار خرید، و به عمر کوتاه آن بعنوان یک شرکت مستقل پایان داد.

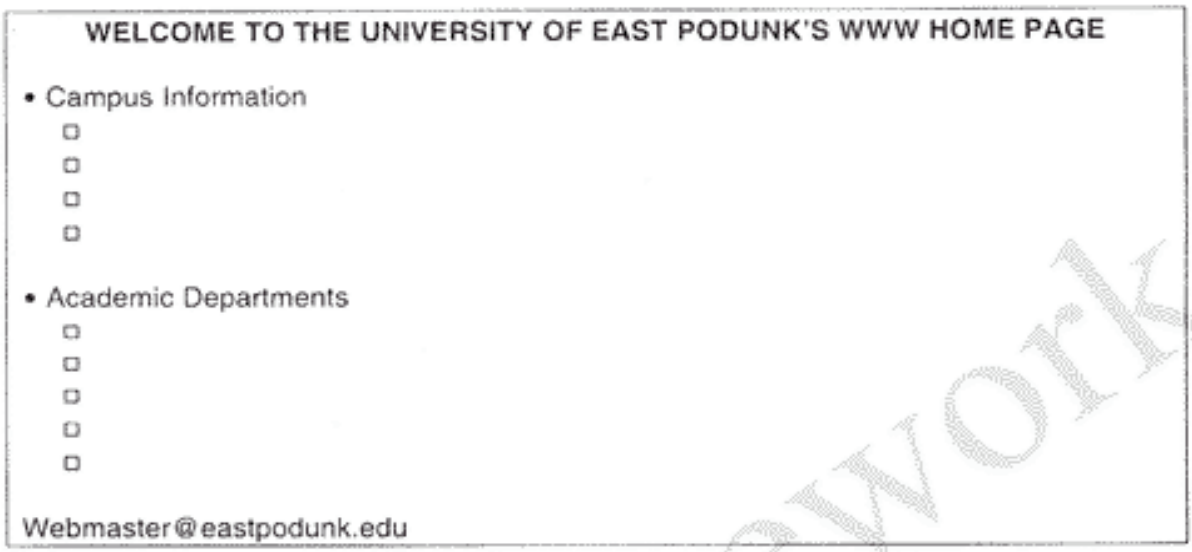
در ۱۹۹۴، CERN و M.I.T موافقتنامه‌ای برای تأسیس کنسرسیوم WWW (که به W3C معروف است) امضا کردند. وظیفهٔ این سازمان توسعهٔ وب، استاندارد کردن پروتکلها، و تسهیل ارتباط بین سایت‌ها بود، و برنرز-لی نیز بعنوان اولین رئیس آن انتخاب شد. از آن زمان تاکنون صدها دانشگاه و شرکت بزرگ به این کنسرسیوم پیوسته‌اند. با اینکه صدها کتاب خوب دربارهٔ وب چاپ شده، اما بهترین محل برای کسب اطلاعات دربارهٔ وب خود وب است. آدرس صفحهٔ اصلی سایت کنسرسیوم وب *www.w3.org* است. در این سایت می‌توانید صدها صفحه، سند و لینک دربارهٔ کنسرسیوم وب و فعالیتهای آن ببینید.

۱-۳-۷ بررسی ساختاری

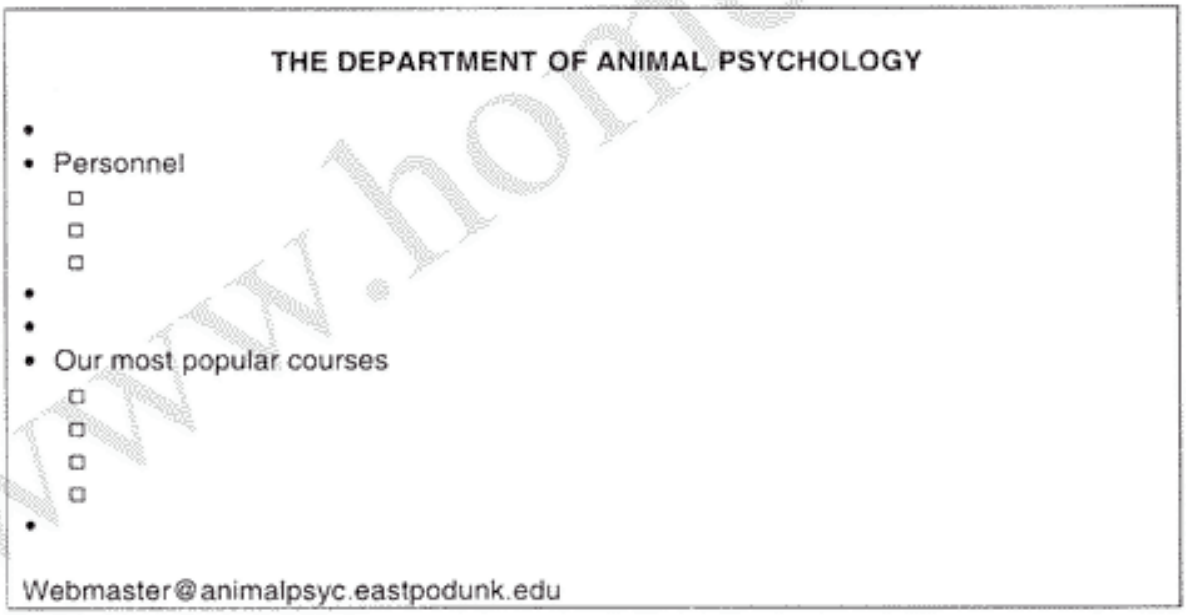
از دید یک کاربر، وب عبارتست از مجموعه‌ای فوق‌العاده بزرگ از میلیونها سند یا **صفحهٔوب** (web page) . هر صفحهٔ وب می‌تواند لینکهایی به صفحات دیگر (در سراسر دنیا) داشته باشد، که برای رفتن به این صفحات کافیست روی لینک آنها کلیک کنید. این کار را می‌توان بدفعات نامحدود تکرار کرد. ایدهٔ صفحاتی که به یکدیگر اشاره می‌کنند، و اکنون آن را با عنوان آبرمتن (hypertext) می‌شناسیم، مدتها قبل از اختراع اینترنت و در سال ۱۹۴۵ توسط وانوار بوش (استاد مهندسی برق دانشگاه M.I.T) ابداع شده بود.

برای دیدن صفحات وب از برنامه‌ای بنام **مرورگر** (browser) استفاده می‌کنیم؛ معروفترین مرورگرهای موجود اینترنت اکسپلورر و نت‌اسکیپ ناویگیتور هستند. مرورگر بعد از آوردن صفحهٔ خواسته شده از وب، محتویات آنرا تفسیر کرده و با فرمت صحیح نمایش می‌دهد. در شکل ۷-۱۸ (الف) نمونه‌ای از یک صفحهٔ وب ساده را ملاحظه می‌کنید. مانند بسیاری از صفحات وب، این صفحه هم با یک عنوان ساده شروع شده، و بعد از مقداری اطلاعات (محتویات صفحه) به یک آدرس ایمیل ختم می‌شود. مرورگر لینکهای موجود در صفحه را، که به آنها **آبرلینک** (hyperlink) گفته می‌شود، بگونه‌ای از سایر قسمتهای متن متمایز می‌کند (با زیرخط، رنگ متفاوت، و یا هر دو). برای دنبال کردن یک لینک، کرسر ماوس را روی لینک موردنظر برده (که معمولاً با اینکار شکل کرسر عوض

می‌شود)، و کلیک کنید. با اینکه مرورگرهای کاملاً متنی هم وجود دارد (مانند لینکس – Lynx)، اما بعلت محبوبیت فراگیر مرورگرهای گرافیکی در ادامهٔ این بخش فقط دربارهٔ آنها صحبت خواهیم کرد. (اخیراً مرورگرهای کلامی نیز توسعه داده شده‌اند، که به دستورات شفاهی کاربر عکس‌العمل نشان می‌دهند.)



(الف)



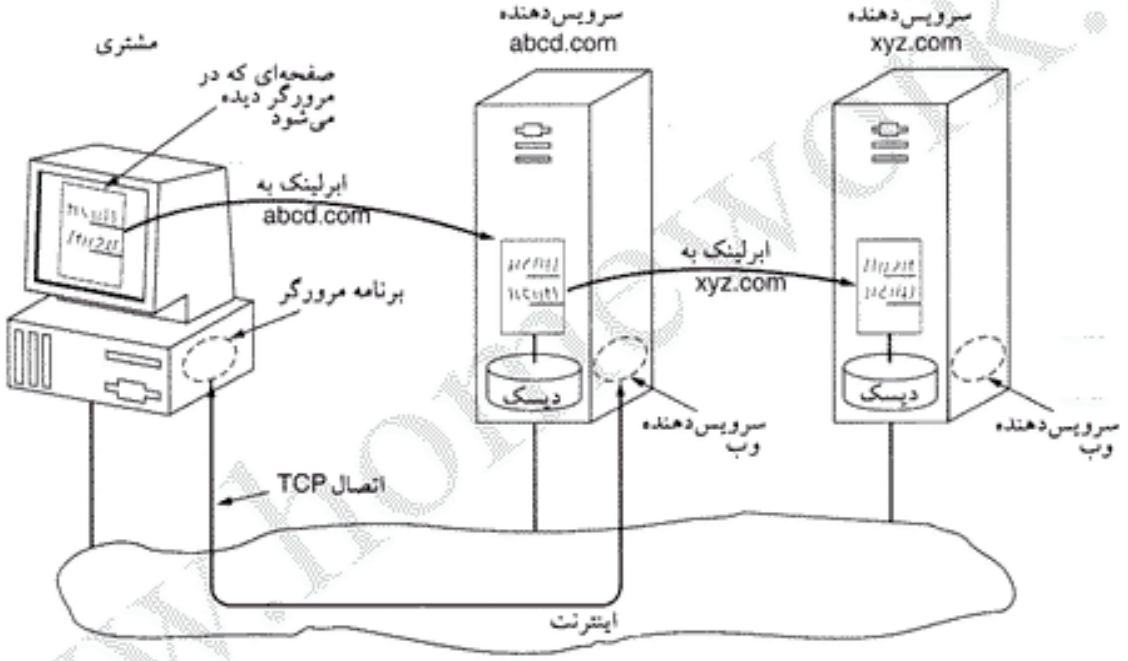
(ب)

شکل ۷-۱۸. (الف) یک صفحهٔ وب. (ب) وقتی کاربر روی لینک Department of Animal Psychology کلیک کند، به اینجا خواهد رسید.

اگر به روانشناسی حیوانات علاقمند باشید، می‌توانید روی لینک Department of Animal Psychology کلیک کنید تا به این صفحه برسید. با این کار، مرورگر صفحهٔ مشخص شده را از سایت وب آورده و نمایش می‌دهد (شکل ۷-۱۸ ب). در این صفحه هم جملاتی که زیرخط‌دار دارند لینک هستند، و با کلیک کردن آنها می‌توان به صفحات دیگر رفت. صفحه‌ای که یک لینک به آن اشاره می‌کند، می‌تواند روی همان ماشین باشد یا در

کامپیوتری آن سوی دنیا - حدس آن برای کاربر ممکن نیست، چون مرورگر بدون کمک وی صفحات را می‌آورد. اگر بعد از دیدن چند صفحه به صفحهٔ اصلی برگردید، لینکهایی که دنبال شده‌اند را با رنگ دیگری مشاهده خواهید کرد. دقت کنید که با کلیک کردن جملهٔ *Campus Information* در صفحهٔ اصلی هیچ اتفاقی نخواهد افتاد، چون این یک لینک نیست (زیرخط ندارد).

طرز کار مدل وب در شکل ۷-۱۹ نشان داده شده است. در اینجا، مرورگر در حال نمایش یک صفحه روی کامپیوتر مشتری است. وقتی کاربر روی یک جمله که به صفحه‌ای روی ماشین *abcd.com* لینک شده کلیک می‌کند، مرورگر لینک را دنبال کرده و از ماشین *abcd.com* می‌خواهد که آن صفحه را برایش بفرستد. اگر در همین صفحه لینک دیگری به یک صفحه در ماشین *xyz.com* وجود داشته باشد و کاربر آنرا کلیک کند، مرورگر درخواست خود را به سرویس‌دهنده *xyz.com* می‌فرستد (و الی آخر).



شکل ۷-۱۹. بخشی از مدل وب.

سمت مشتری

اجازه دهید فعل و انفعالات **سمت‌مشتری** (client-side) را با جزئیات بیشتر بررسی کنیم. یک مرورگر در واقع برنامه‌ایست که می‌تواند حرکات ماوس و کلیک‌های آنرا تشخیص داده و صفحات وب را نمایش دهد. وقتی یک آیتم انتخاب می‌شود، مرورگر اَبَرلینک را دنبال کرده و صفحهٔ وب را می‌آورد. بنابراین، اَبَرلینک‌ها باید راهی برای نامگذاری و مشخص کردن صفحات وب داشته باشند. صفحات وب با استفاده از URL (**پابند:همسان‌منابع** - Uniform Resource Locator) نامگذاری می‌شوند. در زیر یک URL نوعی را می‌بینید:

http://www.abcd.com/products.html

در قسمتهای آینده بیشتر دربارهٔ URLها توضیح خواهیم داد. فعلاً کافیست بدانید که، یک URL سه بخش دارد: نام پروتکل (*http*)، نام DNS ماشینی که صفحه روی آن قرار دارد (*www.abcd.com*)، و نام فایل صفحهٔ وب (*products.html*).

وقتی کاربر روی اَبَرلینک کلیک می‌کند، مرورگر بایستی مراحلی را برای آوردن صفحهٔ وب طی کند. فرض

کنید کاربر بعد از کمی گشت و گذار در وب به لینک صفحهٔ اصلی ITU (که URL آن *http://www.itu.org/home/index.html* است) رسیده و می‌خواهد آنرا ببیند. اجازه دهید ببینیم بعد از انتخاب این لینک، مرورگر چه کارهایی انجام می‌دهد.

۱. مرورگر با بررسی آیتم انتخاب شده، URL آنرا بدست می‌آورد.
۲. مرورگر از DNS خود آدرس IP سایت *www.itu.org* را می‌پرسد.
۳. DNS آدرس 156.106.192.32 را برمی‌گرداند.
۴. مرورگر یک اتصال TCP به پورت 80 ماشین 156.106.192.32 برقرار می‌کند.
۵. سپس روی این لینک درخواستی برای فایل */home/index.html* به آن می‌فرستد.
۶. سرویس‌دهنده *www.itu.org* فایل */home/index.html* را به مرورگر می‌گرداند.
۷. اتصال TCP قطع می‌شود.
۸. مرورگر متن صفحهٔ */home/index.html* را نمایش می‌دهد.
۹. مرورگر تصاویر صفحه را هم آورده و نمایش می‌دهد.

اغلب مرورگرها مراحل کار خود را در خط وضعیت (پائین پنجرهٔ مرورگر) نشان می‌دهند. بدین‌ترتیب در مواقعی که مشکلی پیش می‌آید، کاربر می‌تواند ببیند که کدام مرحله مشکل‌ساز شده است (DNS دیر جواب می‌دهد، سرویس‌دهنده www سرش شلوغ است، یا ترافیک شبکه زیاد است).

برای آن که مرورگر بتواند صفحات وب را نمایش دهد، باید فرمت آنها را بداند. بهمین دلیل تمام صفحات وب با یک زبان استاندارد بنام HTML نوشته می‌شوند، تا هر مرورگری بتواند آنها را تفسیر کند. در ادامه دربارهٔ زبان HTML بیشتر صحبت خواهیم کرد.

با اینکه مرورگرها اساساً چیزی نیستند جز مفسر فایلهای HTML ، اغلب آنها دکمه‌های دیگری نیز دارند که به کاربر در **وبگردی** (web surfing) کمک می‌کند. دکمه‌های عقب (Back - برای برگشت به صفحهٔ قبلی)، جلو (Forward - برای برگشت به صفحهٔ بعدی) و خانه (Home - برای برگشت به صفحهٔ شروع) در اغلب مرورگرها وجود دارد. بسیاری از مرورگرها دارای دکمه‌ها یا منوهایی برای علامتگذاری صفحات (بمنظور ردیابی سریع آنها) نیز هستند. ذخیره و چاپ صفحات وب از دیگر امکانات مرورگرهاست، و در ضمن کاربر می‌تواند محیط کار مرورگر را مطابق نیاز خود تنظیم کند.

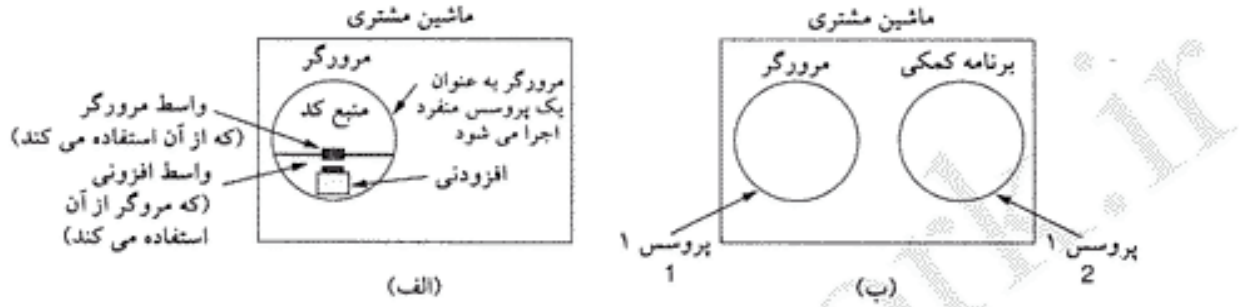
صفحات وب علاوه بر متن معمولی و اَبَرلینک، می‌توانند آیکون، طرح، و تصویر نیز داشته باشند، که هر یک از آنها می‌تواند به صفحات دیگر لینک شده باشد (و با کلیک کردن آنها، مرورگر صفحهٔ مشخص شده را باز خواهد کرد). حتی در برخی موارد قسمتهای مختلف یک تصویر می‌تواند به صفحات مختلفی لینک شده باشد.

تمام صفحات وب HTML نیستند؛ یک صفحهٔ وب می‌تواند حاوی سند PDF ، تصاویری با فرمتهای GIF و JPEG ، موسیقی با فرمت MP3 ، ویدئو با فرمت MPEG ، و یا صدها نوع فایل دیگر باشد. از آنجائیکه یک صفحهٔ استاندارد HTML می‌تواند لینکهایی به هر کدام از این فایلها داشته باشد، مرورگر در برخورد با صفحاتی که فرمت آنها را نمی‌شناسد، مشکل خواهد داشت.

بجای اینکه هر روز مرورگرهای بزرگتری بسازیم که بتوانند فرمتهای جدید را بخوانند، اکثر مرورگرها از روش کلی‌تر و مؤثرتری استفاده می‌کنند. وقتی سرویس‌دهنده وب صفحه‌ای را برمی‌گرداند، اطلاعات اضافه‌ای را نیز دربارهٔ آن می‌دهد، که این اطلاعات شامل نوع MIME صفحه هم می‌شود (شکل ۷-۱۲ را ببینید). صفحات *text/html* (و چند نوع سادهٔ دیگر) مستقیماً نمایش داده می‌شوند. اگر فرمت فایل یکی از این انواع شناخته شده نباشد، مرورگر برای نمایش آن با جدول انواع MIME مشورت می‌کند. در این جدول برای هر نوع MIME یک

نمایش‌دهنده (viewer) معرفی شده است.

نمایش‌دهنده‌ها بر دو نوعند: افزودنی‌ها، و برنامه‌های کمکی. **افزودنی** (plug-in) یک قطعه کُد است، که مرورگر آنرا از روی دیسک خوانده و به قابلیتهای خود اضافه می‌کند (شکل ۷-۲۰ الف). از آنجائیکه افزودنی در داخل مرورگر اجرا می‌شود، به صفحهٔ وب دسترسی داشته و می‌تواند ظاهر آنرا عوض کند. بعد از آن که افزودنی کارش را انجام داد (معمولاً بعد از اینکه کاربر به صفحهٔ دیگری رفت)، از حافظهٔ مرورگر پاک می‌شود.



شکل ۷-۲۰. (الف) یک افزودنی. (ب) یک برنامهٔ کمکی.

هر مرورگر دارای مجموعه‌ای از روالهاست که افزودنی باید آنها را پیاده‌سازی کند تا مرورگر بتواند با آن تماس برقرار کند. برای مثال، روالی باید وجود داشته باشد که مرورگر بتواند از طریق آن داده‌ها را به افزودنی بدهد. به این مجموعه از روالها **واسط افزودنی** (plug-in interface) گفته می‌شود، و برای هر مرورگر باید واسطی خاص نوشته شود. علاوه بر آن، مرورگر نیز تعدادی از روالهای خود را در اختیار افزودنی می‌گذارد تا بتواند به آن سرویس بدهد (روالهای تخصیص حافظه، نمایش وضعیت، و گرفتن پارامترها از این دسته‌اند). به این روالها **واسط مرورگر** (browser interface) می‌گویند.

قبل از استفاده از افزودنی باید آنرا نصب کرد، و این کار بر عهدهٔ کاربر است (که به سایت وب افزودنی رفته، آنرا بار کرده و نصب کند). در ویندوز، این فایل معمولاً یک فایل فشردهٔ خود-استخراج (با پسوند *exe*.) است. وقتی کاربر روی این فایل دو-کلیک کند، برنامهٔ کوچکی که در دل فایل فشرده تعبیه شده، اجرا شده و بعد از باز کردن فایل افزودنی آنرا در دایرکتوری افزودنی‌های مرورگر کپی می‌کند. سپس ارتباط ارگانیک نوع MIME افزودنی با آن را برقرار می‌کند. در سیستمهای یونیکس، این فرآیند اغلب بر عهدهٔ یک **اسکریپت پوسته** (shell script) گذاشته می‌شود.

روش دیگر گسترش دادن قابلیتهای مرورگر استفاده از **برنامهٔ کمکی** (helper application) است؛ این یک برنامهٔ کامل است که بعنوان یک پروسس مستقل اجرا می‌شود (شکل ۷-۲۰ ب). از آنجائیکه برنامهٔ کمکی یک برنامهٔ مستقل است، هیچ واسطی در اختیار مرورگر نمی‌گذارد و از سرویسهای آن هم استفاده نمی‌کند. بجای آن، نام فضای موقتی که فایل در آن ذخیره شده را از مرورگر گرفته، محتویات صفحه را خوانده و نمایش می‌دهد. برنامه‌های کمکی معمولاً برنامه‌های بزرگ و مستقلی مانند Adobe Acrobat Reader (برای نمایش فایلهای PDF) و Microsoft Word (برای نمایش فایلهای DOC) هستند. برخی از برنامه‌های کمکی برای اجرا شدن به یک افزودنی کوچک نیاز دارند.

اغلب برنامه‌های کمکی از نوع MIME *application* استفاده می‌کنند. در این نوع MIME زیرنوع‌های بسیاری تعریف شده، که *application/pdf* (برای فایلهای PDF) و *application/msword* (برای فایلهای DOC) از آن جمله‌اند. بدین ترتیب، یک URL می‌تواند مستقیماً به فایل PDF یا DOC اشاره کند، و وقتی کاربر چنین

لینکی را کلیک کند، Acrobat یا Word بطور خودکار اجرا شده و محتویات فایل را نمایش می‌دهند. با این روش می‌توان بدون کوچکترین تغییر در مرورگر، کاری کرد که بتواند انواع نامحدودی از فایلها را نمایش دهد. اغلب سرویس‌دهنده‌های وب برای خواندن صدها نوع/زیرنوع MIME پیکربندی شده‌اند، که روز به روز هم بر تعداد آنها افزوده می‌شود. البته برنامه‌های کمکی به نوع *application* محدود نیستند؛ برای مثال، برنامهٔ Adobe Photoshop از زیرنوع *image/x-photoshop* و برنامهٔ RealOne Player از زیرنوع audio/mp3 استفاده می‌کنند.

وقتی برنامه‌ای در ویندوز نصب می‌شود، خود را برای نمایش فایل MIME وابسته ثبت می‌کند. این مکانیزم وقتی چند برنامه برای نمایش یک نوع فایل (مثلاً، *video/mpg*) وجود داشته باشد، می‌تواند مشکل‌ساز شود (معمولاً برنامه‌ای که آخر نصب شده نمایش فایل را به انحصار خود در می‌آورد). در نتیجه، نصب برنامه‌های جدید می‌تواند رفتار مرورگر را در نمایش فایلها تغییر دهد.

در یونیکس، معمولاً این فرآیند خودکار نیست، و این کاربر است که فایلهای پیکربندی را به روز درمی‌آورد. این روش کار بیشتری می‌برد، ولی دردسر آن هم کمتر است.

مرورگرها (علاوه بر صفحات وب) فایلهای محلی را نیز می‌توانند باز کنند، اما از آنجائیکه این فایلها دارای نوع MIME نیستند، مرورگر باید راهی برای تشخیص برنامهٔ کمکی یا افزودنی نمایش‌دهندهٔ آنها داشته باشد. برای این کار معمولاً از پسوند نام فایل استفاده می‌شود. برخی مرورگرها (علاوه بر نوع MIME و پسوند فایل) از محتویات خود فایل نیز برای تشخیص نوع آن استفاده می‌کنند. برای مثال، اینترنت اکسپلورر بیش از نوع MIME به پسوند فایل متکی است.

در این حالت هم برنامه‌های مختلفی که قادر به نمایش یک نوع فایل هستند، بر سر باز کردن آن با هم رقابت می‌کنند. در برنامه‌های حرفه‌ای کاربر می‌تواند وابستگی برنامه به انواع MIME را فعال یا غیرفعال کند. اما برنامه‌های زیادی هم هستند که هیچ اهمیتی به این مسائل نمی‌دهند، و خیلی ساده وابستگی فایل را به خود اختصاص می‌دهند.

توانایی مرورگر در نمایش انواع مختلف فایلها بسیار جالب است، اما در عین حال می‌تواند دردسرساز باشد. برای مثال، وقتی اینترنت اکسپلورر یک فایل *exe* را می‌آورد، متوجه می‌شود که این فایل یک برنامه است و هیچ برنامهٔ کمکی برای آن ندارد: محتملترین گزینه اجرای این برنامه است. اما این می‌تواند یک رخنهٔ امنیتی بزرگ باشد. تنها کاری که یک سایت وب خرابکار باید انجام دهد، پنهان کردن لینک کُد ویروس پشت تصویری از یک ستارهٔ سینما یا قهرمان ورزشی است. کاربر از همه جا بی خبر روی تصویر موردعلاقه‌اش کلیک می‌کند، و با این کار ویروس مخرب را وارد کامپیوتر خود کرده و اجرا می‌کند. البته می‌توان اینترنت اکسپلورر را بگونه‌ای تنظیم کرد که هر برنامه‌ای را بطور خودکار اجرا نکند (و یا حداقل قبل از اجرای برنامه تأئید کاربر را بگیرد)، ولی بسیاری از کاربران مبتدی نمی‌دانند چگونه باید چنین تنظیمی را انجام دهند.

در سیستمهای یونیکس هم این اتفاق می‌تواند بیفتد، مشروط بر اینکه کاربر آگاهانه پوسته را بعنوان یک برنامهٔ کمکی تعریف کرده باشد. خوشبختانه، این کار بقدری پیچیده است که احتمال اینکه تصادفی چنین اتفاقی بیفتد، بسیار ناچیز است (در واقع، فقط عدهٔ کمی هستند که می‌توانند چنین کاری را انجام دهند).

سمت سرویس‌دهنده

کمی هم از فعل و انفعالات سمت سرویس‌دهنده بگوئیم. همانطور که قبلاً دیدید، وقتی کاربر روی یک URL (یا اَبَرلینک) کلیک می‌کند، مرورگر هر آنچه را که بین *http://* و / بعدی قرار دارد یک نام DNS تلقی کرده، و بدنبال آدرس IP آن می‌رود. بعد از بدست آوردن آدرس IP سرویس‌دهنده، مرورگر یک اتصال TCP به پورت 80 آن

برقرار می‌کند و بقیهٔ URL را (که نام فایل صفحهٔ وب است) به سرویس‌دهنده می‌فرستد؛ سرویس‌دهنده هم در جواب صفحهٔ خواسته شده را به مرورگر برمی‌گرداند.

یک سرویس‌دهنده وب (با کمی تسامح) شبیه سرویس‌دهنده شکل ۶-۶ است. در هر دوی آنها مراحلی که سرویس‌دهنده (در حلقهٔ اصلی خود) طی می‌کند، مانند زیر است:

۱. اتصال TCP را از مشتری (مرورگر) قبول می‌کند.
۲. نام فایل درخواست شده را می‌گیرد.
۳. فایل را (از روی دیسک) می‌خواند.
۴. فایل را به مشتری برمی‌گرداند.
۵. اتصال TCP را قطع می‌کند.

سرویس‌دهنده‌های وب مدرن ویژگیهای بسیار بیشتری دارند، ولی وظیفهٔ اصلی آنها همان است که در بالا گفتیم.

یکی از مشکلات روش فوق اینست که سرویس‌دهنده برای هر درخواست باید به دیسک مراجعه کند. در نتیجه تعداد درخواستهایی که یک سرویس‌دهنده وب می‌تواند پاسخ دهد، به توانایی آن در مراجعه به دیسک بستگی دارد. زمان دسترسی در جدیدترین دیسکهای SCSI در حدود 5 msec است، و این یعنی حداکثر 200 درخواست در ثانیه (که البته اگر فایل بزرگ باشد، کمتر هم خواهد شد). این مقدار حتی برای یک سرویس‌دهنده وب متوسط هم بسیار کم است.

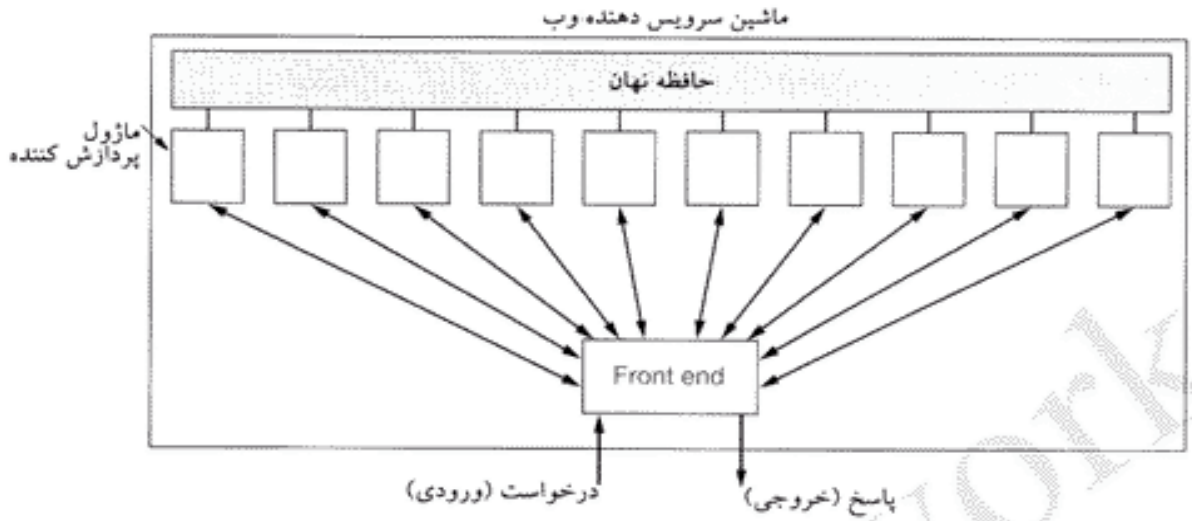
ساده‌ترین روش بهبود پاسخ سرویس‌دهنده وب (که در تمام سرویس‌دهنده‌های وب هم از آن استفاده می‌شود)، نگهداری آخرین n فایل درخواست شده در حافظهٔ RAM است. سرویس‌دهنده وب قبل از رفتن به سراغ دیسک، این **حافظهٔ‌نهان** (cache) را چک می‌کند. اگر فایل خواسته شده در حافظهٔ نهان باشد، از همان جا برداشته شده و به مشتری تحویل داده می‌شود (و دسترسی دیسک انجام نخواهد شد). با اینکه این تکنیک به حافظهٔ زیادی نیاز دارد و زمانی هم صرف جستجوی حافظهٔ نهان می‌شود، اما تقریباً همیشه سریعتر از دسترسی مستقیم دیسک است.

قدم بعدی برای سریعتر کردن سرویس‌دهنده وب، طراحی آن بصورت چندریسمانی (multithreaded) است. در این تکنیک سرویس‌دهنده وب یک ماژول جلودار (front-end – که درخواستهای رسیده را می‌پذیرد) و k ماژول **پردازش‌کننده** (processing) دارد (شکل ۷-۲۱ را ببینید). تمام این $k + 1$ ریسمان متعلق به یک پروسس هستند، بنابراین به فضای آدرس آن دسترسی دارند. وقتی یک درخواست جدید به سرویس‌دهنده وب می‌رسد، ماژول جلودار آن را پذیرفته، مشخصات آنرا ثبت می‌کند، و سپس به یکی از ماژولهای پردازش‌کننده تحویل می‌دهد. (تکنیک دیگری نیز وجود دارد که در آن ماژول جلودار حذف شده، و ماژولهای پردازش‌کننده مستقیماً درخواستها را دریافت می‌کنند. اما در این روش بایستی مکانیزمی برای جلوگیری از تداخل پروسسها در نظر گرفته شود.)

ماژول پردازش‌کننده ابتدا حافظهٔ نهان را چک می‌کند تا ببیند که فایل موردنظر در آنجا هست یا خیر. اگر آنجا باشد، اشاره‌گری را که به آن فایل اشاره می‌کند، به روز در می‌آورد. اگر در حافظهٔ نهان نباشد، سراغ دیسک می‌رود و مقداری از آن را در حافظهٔ نهان کپی می‌کند (که باحتمال زیاد مقداری از محتویات قدیمی حافظهٔ نهان را از بین می‌برد). بعد از کپی کردن فایل در حافظهٔ نهان، آنرا برای مشتری می‌فرستد.

مزیت این روش آن است که در زمانهایی که یکی از پروسسها در حال خواندن دیسک است (و کاری با CPU ندارد)، سایر پروسسها می‌توانند با استفاده از حافظهٔ نهان به درخواستهای رسیده پاسخ دهند. برای بهبود واقعی کارایی در سیستمهای چندریسمانی، لازم است که ماشین سرویس‌دهنده وب بجای یک دیسک چندین دیسک (و

چندین کنترلر دیسک) داشته باشد. با داشتن k دیسک در سرویس‌دهنده‌ای با k ریسمان، کارایی سیستم k برابر سیستمهای یک دیسکی خواهد بود.



شکل ۷-۲۱. یک سرویس‌دهنده وب چندریسمانی، با یک ماژول جلودار و چند ماژول پردازش‌کننده.

از نظر تئوری، یک سیستم تک‌ریسمانی با k دیسک نیز می‌تواند به همان میزان از کارایی دست یابد، ولی چنین سیستمی بسیار پیچیده‌تر خواهد بود، چون ریسمانی که در حال خواندن دیسک است، هیچ کار دیگری نمی‌تواند انجام دهد (در حالیکه این محدودیت در سیستمهای چندریسمانی وجود ندارد).

سرویس‌دهنده‌های وب جدید فقط فایل عرضه نمی‌کنند، بلکه کارهای بیشتری (که می‌توانند بسیار پیچیده باشند) انجام می‌دهند. به همین دلیل، در اغلب سرویس‌دهنده‌ها هر ماژول پردازش‌کننده (پس از دریافت درخواست از ماژول جلودار، و بسته به نوع درخواست) مراحل مختلفی را طی می‌کند.

۱. تعیین نام صفحهٔ وب خواسته شده.
۲. احراز هویت مشتری.
۳. کنترل دسترسی مشتری.
۴. کنترل دسترسی صفحهٔ وب.
۵. چک کردن حافظهٔ نهان.
۶. آوردن صفحهٔ خواسته شده از دیسک.
۷. تعیین نوع MIME فایل و نوشتن آن در پاسخ مشتری.
۸. انجام جنبه‌های دیگر درخواست.
۹. برگرداندن پاسخ به مشتری.
۱۰. نوشتن یک رکورد در دفتر ثبت وقایع (log) سرویس‌دهنده.

شاید فکر کنید مرحلهٔ ۱ اضافیست، چون نام صفحهٔ وب همیشه در درخواست مشتری قید می‌شود، اما موارد زیادی هست که نام صفحه در درخواست مشتری وجود ندارد. برای مثال، *http://www.cs.vu.nl* یک URL معتبر است که نام صفحه ندارد. در اینجا باید یک صفحهٔ پیش‌فرض وجود داشته باشد که سرویس‌دهنده (در صورت

عدم وجود نام صفحه) از آن استفاده کند. در سرویس‌دهنده‌های وب جدید حتی می‌توان زبان پیش‌فرض (مانند انگلیسی، یا ایتالیایی) هم داشت، که صفحات وب (در صورت وجود) به این زبان برگردانده می‌شوند. بعلت وجود این مکانیزمهای پیش‌فرض، آوردن نام صفحات دیگر چندان الزامی نیست.

در مرحلهٔ ۲ هویت کاربر تأئید می‌شود. این مرحله بیشتر برای صفحاتی است که در معرض دسترسی عموم قرار ندارند، و فقط افراد خاصی می‌توانند از آنها استفاده کنند. در قسمتهای آینده یکی از روشهای احراز هویت مشتری را مورد بررسی قرار خواهیم داد.

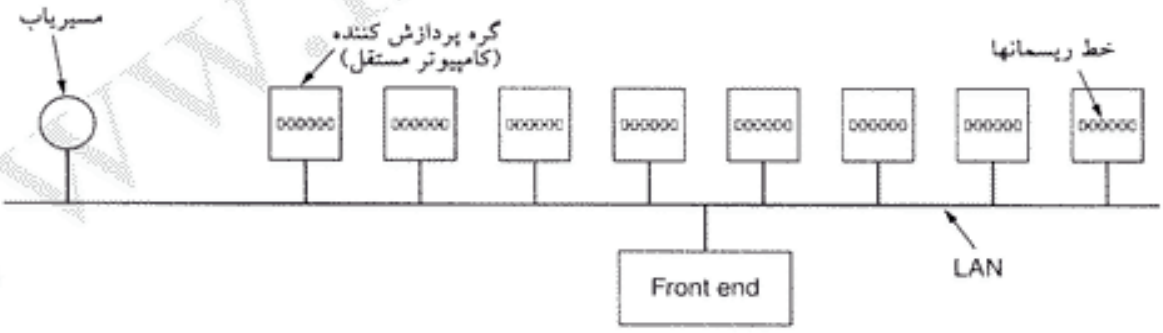
در مرحلهٔ ۳ مجوزهای کاربر در دسترسی به صفحهٔ خواسته شده چک می‌شود. مرحلهٔ بعد (مرحلهٔ ۴) این قبیل مجوزها را نسبت به خود صفحه بررسی می‌کند. مجوزهای دسترسی صفحات در فایلهای خاصی (در همان دایرکتوری که صفحه در آن قرار دارد) تعیین می‌شود - فایل *htaccess* یکی از این نوع فایلهاست.

صفحهٔ خواسته شده در مراحل ۵ و ۶ گرفته می‌شود. روتین مرحلهٔ ۶ باید بگونه‌ای طراحی شود که قادر به کار همزمان با چندین دیسک باشد.

در مرحلهٔ ۷ نوع MIME فایل (از روی پسوند فایل، عبارتی در ابتدای فایل، یا از طریق فایلهای پیکربندی) تعیین می‌شود. در مرحلهٔ ۸ کارهای متفرقه‌ای از قبیل ایجاد پروفایل کاربر یا جمع‌آوری داده‌های آماری، صورت می‌گیرد.

صفحه در مرحلهٔ ۹ برای مشتری فرستاده شده، و در مرحلهٔ ۱۰ اقدامات انجام شده در یک فایل ثبت می‌شود (که این اطلاعات معمولاً بدرد سرپرست سایت می‌خورد).

اگر تعداد درخواستهای رسیده در واحد زمان بسیار زیاد باشد، CPU (صرفنظر از تعداد دیسکهای موازی) نمی‌تواند از عهدهٔ انجام آنها برآید. راه حل این مشکل استفاده از کامپیوترهای متعدد برای توزیع بار سرویس‌دهنده است. این مدل را که **مزرعهٔ سرویس‌دهنده** (server farm) نام دارد، در شکل ۷-۲۲ ملاحظه می‌کنید. در این روش هم یک ماژول جلودار وجود دارد که در خواستها را گرفته، ولی این بار (بجای پروسسهای مختلف) در گره‌ها یا کامپیوترهای مختلف - که هر کدام می‌توانند ماشینهای چندریسمانی با چندین دیسک باشند - توزیع می‌کند.

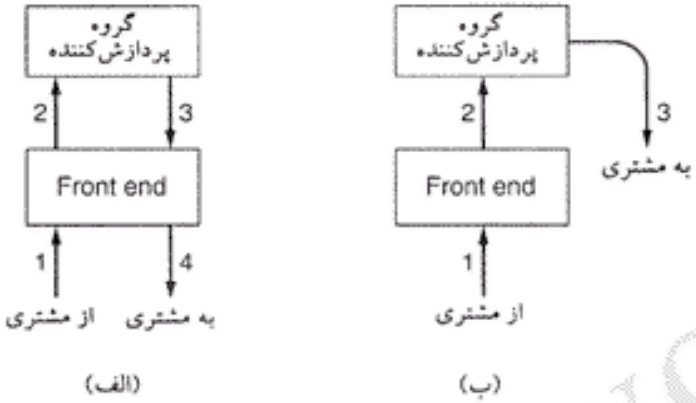


شکل ۷-۲۲. مزرعهٔ سرویس‌دهنده.

یکی از مشکلات مزرعهٔ سرویس‌دهنده این است که (بعلت جدا بودن حافظهٔ کامپیوترها) حافظهٔ نهان واحد نمی‌تواند وجود داشته باشد - مگر اینکه از کامپیوتری با چندین CPU و حافظهٔ مشترک استفاده کنیم. برای غلبه بر این مشکل و بالا بردن کارایی سیستم، ماژول جلودار می‌تواند مسیر درخواستهای مختلف را ثبت کرده و درخواستهای بعدی مرتبط با یک صفحهٔ مشخص را به همان کامپیوتر قبلی بفرستد. این تکنیک باعث می‌شود تا هر گره (کامپیوتر) به ارائهٔ صفحات خاصی اختصاص یابد، و از هرز رفتن حافظهٔ نهان جلوگیری شود.

مشکل دیگر مزرعهٔ سرویس‌دهنده این است که اتصالهای TCP به ماژول جلودار می‌رسند، و پاسخ نیز باید از همان مسیر برگردد. به شکل ۷-۲۳ (الف) نگاه کنید: در اینجا درخواست ورودی (1) و پاسخ سرویس‌دهنده وب

(4) هر دو از ماژول جلودار عبور کرده‌اند. برای دور زدن این مشکل، گاهی از تکنیکی بنام پاس‌دادن **TCP** (TCP handoff) استفاده می‌شود. در این روش اتصال TCP مشتری نیز به گره پردازش‌کننده منتقل می‌شود، تا این گره بتواند پاسخ را مستقیماً به مشتری برگرداند - مسیر (3) در شکل ۷-۲۳ (ب). کل فرآیند پاس دادن TCP از دید مشتری مخفی است.



شکل ۷-۲۳. (الف) فرآیند عادی درخواست-پاسخ. (ب) درخواست-پاسخ با استفاده از تکنیک پاس دادن TCP.

URL

بارها گفته‌ایم که یک صفحهٔ وب می‌تواند لینکهایی به صفحات دیگر داشته باشد؛ اکنون خواهید دید که این کار چگونه صورت می‌گیرد. وقتی وب اختراع می‌شد، بلافاصله روشن شد که صفحات وب باید مکانیزمی برای نامگذاری و مکان‌یابی صفحاتی که به آنها لینک دارند، داشته باشند. این مکانیزم، بویژه، باید به سه سئوال اساسی دربارهٔ صفحهٔ انتخاب شده جواب دهد:

۱. نام این صفحه چیست؟
۲. کجا قرار دارد؟
۳. چگونه می‌توان آنرا خواند؟

اگر هر صفحه دارای یک نام منحصربفرد باشد، پیدا کردن آن قاعدتاً نباید مشکلی داشته باشد. اما این راه حل هم مشکل ما را برطرف نخواهد کرد. با یک مثال موضوع روشنتر خواهد شد. در ایالات متحدهٔ آمریکا هر فرد یک شمارهٔ تامین اجتماعی منحصربفرد دارد، اما آیا فقط با داشتن این شماره می‌توان فرد موردنظر را پیدا کرد. آدرس این فرد کجاست؟ با چه زبانی باید با او تماس گرفت؟ همین مسائل برای وب هم مصداق دارند.

راه حلی که برای این موضوع پیدا شد، به هر سه سئوال فوق یکجا جواب می‌دهد. در این راه حل، هر صفحهٔ وب یک URL (یابندهٔ‌همسان‌منابع - Uniform Resource Locator) دارد، که آنرا بگونه‌ای منحصربفرد در تمام دنیا مشخص می‌کند. هر URL سه بخش دارد: پروتکل (سئوال ۳)، نام DNS ماشینی که صفحه روی آن قرار دارد (سئوال ۲)، و نام صفحه در آن ماشین (سئوال ۱). بعنوان نمونه در سایت وب مؤلف کتاب چندین فایل ویدئویی از شهر آمستردام و دانشگاه آن وجود دارد، که در صفحه‌ای با URL زیر قرار دارند:

http://www.cs.vu.nl/video/index-en.html

سه بخش این URL عبارتند از: پروتکل (*http*)، نام DNS ماشین سرویس‌دهنده (*www.cs.vu.nl*)، و نام صفحهٔ وب (*video/index-en.html*) که با / از هم جدا شده‌اند. نام صفحهٔ وب عبارتست از مسیر نسبی فایل در دایرکتوری وب پیش‌فرض کامپیوتر *cs.vu.nl*.

در اغلب سایتهای وب از نامهای کوتاه شده برای فایلها استفاده می‌شود. اگر نام فایل در URL وجود نداشته باشد، معمولاً صفحهٔ اصلی سایت (home page) برگردانده می‌شود. اگر فقط دایرکتوری صفحه مشخص شده باشد، سرویس‌دهنده وب فایل پیش‌فرض آن دایرکتوری (که معمولاً *index.html* نام دارد) را به مشتری برمی‌گرداند. در برخی از سرویس‌دهنده‌های وب نیز از نامهای کوتاهی مانند */user/~* برای مشخص کردن دایرکتوری اختصاصی اشخاص استفاده می‌شود. بعنوان مثال، برای دسترسی به صفحهٔ اصلی فضای وب مؤلف می‌توانید از URL زیر استفاده کنید:

http://www.cs.vu.nl/~ast/

اکنون اجازه دهید ببینیم آبرمتن (hypertext) چگونه کار می‌کند. برای اینکه یک عبارت قابل‌کلیک باشد، نویسندهٔ متن باید دو چیز را مشخص کند: متنی که باید دیده شود، و URL صفحه‌ای که بعد از کلیک شدن عبارت باید باز شود (بعداً در همین فصل روش ایجاد چنین عبارتهایی را خواهید دید). وقتی این عبارت کلیک شود، مرورگر آدرس سرویس‌دهنده صفحه را با استفاده از DNS پیدا کرده، یک اتصال TCP به آن برقرار می‌کند و نام فایل و پروتکل را به سرویس‌دهنده می‌دهد. سرویس‌دهنده هم صفحهٔ خواسته شده را برمی‌گرداند.

یکی از ویژگیهای URL این است که می‌توان براحتی پروتکل مورد استفاده را عوض کرده و به منابع مختلف دسترسی پیدا کرد. در حقیقت تعداد زیادی از این پروتکلها تعریف شده، که برخی از معروفترین آنها را در شکل ۷-۲۴ ملاحظه می‌کنید.

نامه	کاربرد	مثال
http	ابرمتن (HTML)	http://www.cs.vu.nl/~ast/
ftp	FTP	ftp://ftp.cs.vu.nl/pub/minix/README
file	فایل محلی	file:///usr/suzanne/prog.c
news	گروه خبری	news:comp.os.minix
news	مقاله خبری	news:AA0134223112@cs.utah.edu
gopher	گوفر	gopher://gopher.tc.umn.edu/11/Libraries
mailto	ارسال ایمیل	mailto:JohnUser@acm.org
telnet	ورود از راه دور	telnet://www.w3.org:80

شکل ۷-۲۴. چند URL معروف.

اجازه دهید این فهرست را مختصراً بررسی کنیم. پروتکل *http* زبان وب است، زبانی که سرویس‌دهنده‌های وب با آن صحبت می‌کنند. HTTP مخفف HyperText Transfer Protocol (**پروتکل انتقال آبرمتن**) است. بعداً دربارهٔ این پروتکل بیشتر صحبت خواهیم کرد.

پروتکل *ftp* (**پروتکل انتقال فایل** – File Transfer Protocol) برای انتقال فایل در اینترنت مورد استفاده قرار می‌گیرد. FTP بیش از دو دهه است که در اینترنت حضور دارد، و پروتکلی کاملاً جا افتاده است. در سراسر دنیا تعداد زیادی سرویس‌دهنده FTP وجود دارد که کاربران اینترنت می‌توانند وارد آنها شده و فایلهای موجود در آنها را بار کنند. وب تغییر چندانی در این وضعیت نداد، فقط کارها را ساده‌تر کرد (چون واسط کاربر FTP کمی کهنه و قدیمی شده بود). FTP بسیار قویتر از HTTP است، چون اجازه می‌دهد کاربر ماشین *A* فایلی را از ماشین *B* به ماشین *C* منتقل کند.

مرورگرها می‌توانند با استفاده از پروتکل *file* مستقیماً با فایلها نیز (مانند صفحات وب) کار کنند، البته فقط با فایلهای محلی (روی همان کامپیوتر) نه فایلهای کامپیوترهای دیگر. روش کار شبیه FTP است، با این تفاوت که

نیازی به سرویس‌دهنده FTP ندارد.

مدتها قبل از اینکه اینترنت بدنیا بیاید، یک سیستم خبری وجود داشت بنام USENET . این سیستم میلیونها کاربر را در بیش از ۳۰,۰۰۰ **گروه‌خبری** (newsgroup) گرد هم آورده بود، کاربرانی که دربارهٔ موضوعات مختلف و متنوع بحث کرده و مقاله رد و بدل می‌کردند. برای ارتباط با این گروههای خبری می‌توان از پروتکل *news* استفاده کرد (یعنی، مرورگرها می‌توانند خبرخوان هم باشند، و در واقع بعضی از آنها حتی از برنامه‌های تخصصی خبرخوان نیز بهتر هستند).

پروتکل *news* از دو فرمت پشتیبانی می‌کند. در فرمت اول بعد از مشخص کردن گروه خبری، می‌توان از میان فهرست مقالات آن مقالهٔ مورد نظر را انتخاب کرد. در فرمت دوم شمارهٔ شناسایی مقاله را باید مشخص کرد (AA0134223112@cs.utah.edu در شکل ۷-۲۴). مرورگرها برای آوردن مقالات گروه خبری از پروتکل NNTP(**پروتکل انتقال اخبار شبکه** – Network News Transfer Protocol) استفاده می‌کنند. در این کتاب دربارهٔ NNTP صحبت نخواهیم کرد، فقط کافیست یادآور شویم که این پروتکل تا حدی شبیه SMTP است و مانند آن عمل می‌کند.

از پروتکل *gopher* در سیستمهای **گوفر** استفاده می‌شود؛ این سیستم در دانشگاه مینه‌سوتا اختراع شد، و نام آن از تیم ورزشی این دانشگاه (**گوفرهای طلایی** – گوفر نوعی موش خرمای بزرگ است، و در ضمن لقبی است که به اهالی ایالت مینه‌سوتا نیز داده شده) گرفته شده است. گوفر سالها قبل از وب وجود داشت، و نوعی سیستم بازیابی اطلاعات محسوب می‌شود (که متأسفانه فقط متنی است، و قابلیتهای گرافیکی ندارد). این سیستم سالهاست که منسوخ شده، و دیگر بندرت کسی از آن استفاده می‌کند.

دو پروتکل آخر شکل ۷-۲۴ ارتباطی با صفحات وب ندارند، و کاربردهای دیگری دارند. پروتکل *mailto* اجازه می‌دهد که کاربر از داخل مرورگر ایمیل بفرستد. برای این کار کافیست روی عبارتی که URL آن *mailto:somebody@abcd.com* است، کلیک کنید. اغلب مرورگرها برای ارسال ایمیل به این آدرس از برنامهٔ ایمیل پیش‌فرض سیستم استفاده می‌کنند. پروتکل *telnet* نیز برای ایجاد ارتباط با کامپیوترهای دیگر بکار می‌رود، و طرز کار آن بسیار شبیه برنامهٔ telnet است.

استفاده از پروتکلهای مختلف به کاربر اجازه می‌دهد تا یک برنامهٔ واحد (مرورگر وب) را برای کارهای مختلف بکار گیرد. اگر نمی‌دانستیم که وب و مرورگر را یک فیزیکدان اختراع کرده، قطعاً آنرا دستپخت بخش تبلیغات یک شرکت بزرگ نرم‌افزاری تصور می‌کردیم.

با وجود تمام این ویژگیهای جالب URL ، رشد سرسام‌آور وب ضعف ذاتی آن را آشکار کرد: هر URL به یک سرویس‌دهنده مشخص اشاره می‌کند. برای کاهش ترافیک صفحاتی که تعداد مراجعان آن زیاد است، بهتر است یک URL چندین سرویس‌دهنده داشته باشد. اما مشکل اینجاست که، URLها هیچ راهی ندارند که بدون ذکر آدرس دقیق صفحهٔ وب، به آن اشاره کنند. برای مثال، یک URL نمی‌تواند بگوید: «من صفحهٔ xyz را می‌خواهم، اهمیتی هم نمی‌دهم آنرا از کجا می‌آوری.» برای حل این مشکل و امکان تکثیر صفحات وب، IETF در حال کار روی سیستمی از URN (**نام‌جهانی‌منابع** – Universal Resource Name) هاست. در کل، URN را می‌توان یک URL عمومی دانست. تحقیقات در این زمینه همچنان ادامه دارد، اما پیشنهادهایی تحت RFC 2141 نیز ارائه شده است.

بدون‌حالتی و کوکی

همانطور که بارها خاطر نشان کردیم، وی اساساً **بدون‌حالت** (stateless) است، و چیزی مانند **ورود‌به‌سیستم** (login) در آن وجود ندارد. مرورگر درخواست خود را به سرویس‌دهنده می‌فرستد، سرویس‌دهنده هم یک فایل

به آن برمی‌گرداند، و بعد بکلی فراموش می‌کند که اصلاً چنین مشتریی را دیده است.

در اوایل کار، زمانیکه وب فقط یک محیط عمومی بود، این مدل کاملاً کفایت می‌کرد. ولی با گسترش کاربردهای وب این موضوع مشکل‌ساز شد. بعنوان مثال، برای استفاده از برخی سایتهای وب کاربران باید ثبت نام کنند، و احیاناً پولی بپردازند. این وضعیت سئوالی را پیش می‌آورد: «سرویس‌دهنده وب چگونه باید بین درخواستهای کاربرانی که ثبت‌نام کرده‌اند، و کاربران عادی فرق قائل شود؟» مثال دیگر مربوط به **تجارت‌الکترونیک** (e-commerce) می‌شود: «کاربری در یک فروشگاه الکترونیکی می‌چرخد و اجناس موردنیازش را در یک سبد **خریدالکترونیکی** (shopping cart) قرار می‌دهد؛ سئوال اینست که سرویس‌دهنده وب چگونه محتویات هر سبد خرید را ردگیری می‌کند؟» مثال سوم به سایتهایی که اجازه می‌دهند کاربر فضاهای اختصاصی داشته باشد (مانند Yahoo)، مربوط می‌شود: «سرویس‌دهنده وب چگونه کاربران را شناسایی می‌کند، و بخاطر می‌سپارد که هر کاربر صفحه‌اش را چطور تنظیم کرده است؟»

در نگاه اول شاید فکر کنید که سرویس‌دهنده وب می‌تواند از آدرس IP کاربر برای شناسایی وی استفاده کند. اما این روش کار نمی‌کند. اول اینکه، بسیاری کاربران از کامپیوترهای مشترک استفاده می‌کنند (بویژه کارمندان شرکتها)، و آدرس IP فقط بدرد شناسایی کامپیوترها می‌خورد، نه کاربران. دیگر اینکه، اغلب ISPها از تکنیک NAT استفاده می‌کنند، و تمام بسته‌هایی که از این ISPها خارج می‌شود یک آدرس IP دارند. از دیدگاه سرویس‌دهنده وب تمام کاربران این ISP یک آدرس IP بیشتر ندارند.

برای حل این مشکل، نت‌اسکیپ تکنیکی اختراع کرد بنام **کوکی** (cookie) ، که انتقادات زیادی به آن وارد شد. این نام از نوعی برنامهٔ خاص گرفته شده، که کاری را انجام می‌دهد و آنرا در جایی ثبت می‌کند، تا بعدها بتواند دوباره بیاد بیاورد چه کار کرده است (در سیستم عامل هم چنین مفهومی بکرات مورد استفاده قرار می‌گیرد). کوکی بعدها در RFC 2109 جنبهٔ رسمی بخود گرفت.

وقتی مشتری یک صفحهٔ وب درخواست می‌کند، سرویس‌دهنده مقداری اطلاعات اضافی همراه آن می‌فرستد که کوکی هم می‌تواند جزئی از آن باشد. کوکی معمولاً یک فایل کوچک (حداکثر 4 KB) یا یک رشتهٔ متنی است. مرورگر این کوکی‌ها را در یک دایرکتوری خاص روی کامپیوتر مشتری ذخیره می‌کند (البته اگر کاربر این ویژگی را غیرفعال نکرده باشد). کوکی‌ها فقط فایل یا رشته‌های متنی هستند، نه برنامه‌های اجرایی. در حقیقت، کوکی حتی می‌تواند حاوی کُد یک ویروس باشد، اما از آنجائیکه کوکی‌ها اساساً داده تلقی می‌شوند، چنین ویروسی هیچ راهی برای اجرا شدن در کامپیوتر مشتری (و صدمه زدن به آن) ندارد. با این حال، همیشه احتمال آن هست که بالاخره یک هکر بتواند راهی برای این کار پیدا کند.

هر کوکی می‌تواند تا پنج فیلد داشته باشد (شکل ۷-۲۵ را ببینید). فیلد *Domain* مشخص می‌کند که کوکی از کجا آمده است. مرورگرها مکانیزمهایی دارند تا مطمئن شوند که سرویس‌دهنده دروغ نگفته است. هر ناحیه می‌تواند حداکثر ۲۰ کوکی (برای هر مشتری) در یک کامپیوتر ذخیره کند. فیلد *Path* مشخص می‌کند که کدام بخش از سیستم فایل سرویس‌دهنده وب می‌تواند از کوکی استفاده کند. این فیلد اغلب / است، که به معنای کل سیستم فایل می‌باشد.

ناحیه	مسیر	محتویات	زمان انقضا	ایمنی
toms-casino.com	/	CustomerID=497793521	15-10-02 17:00	بلی
joes-store.com	/	Cart=1-00501;1-07031;2-13721	11-10-02 14:22	خیر
aportal.com	/	Prefs=Stk:SUNW+ORCL;Spt:Jets	31-12-10 23:59	خیر
sneaky.com	/	UserID=3627239101	31-12-12 23:59	خیر

شکل ۷-۲۵. چند نمونه کوکی.

فیلد *Content* به شکل *name = value* است، که *name* و *value* می‌توانند هر چیزی (که سرویس‌دهنده می‌خواهد) باشند. این فیلدیست که محتویات کوکی در آن ذخیره می‌شود.

فیلد *Expires* مشخص می‌کند که کوکی تا چه زمانی اعتبار دارد. اگر در این فیلد مقدار وجود نداشته باشد، مرورگر هنگام خروج کوکی را دور می‌اندازد. به این قبیل کوکی‌ها **کوکی بی‌دوام** (nonpersistent cookie) می‌گویند. اگر فیلد *Expires* کوکی تاریخ و ساعت داشته باشد، به آن **بادوام** (persistent) می‌گویند، و کوکی تا این زمان در سیستم حفظ خواهد شد. زمان انقضا بوقت گرینویچ است. اگر سرویس‌دهنده بخواهد یک کوکی را از روی سیستم مشتری پاک کند، کافیست همان کوکی را دوباره با تاریخ انقضایی در گذشته بفرستد.

و بالاخره فیلد *Secure* می‌گوید که مرورگر باید کوکی را فقط به یک سرویس‌دهنده امن (secure) پس بفرستد. این ویژگی بیشتر برای تجارت الکترونیک، عملیات بانکی و سایر کاربردهایی که به ایمنی بالا نیاز دارند، مورد استفاده قرار می‌گیرد.

دیدید که سرویس‌دهنده چطور کوکی‌ها را به مشتری می‌فرستد، اما طرز کار آنها چگونه است؟ درست قبل از اینکه مرورگر درخواستی را به یک سرویس‌دهنده وب بفرستد، دایرکتوری کوکی‌ها را چک می‌کند تا ببیند آیا ناحیهٔ مقصد در آنجا کوکی دارد یا خیر. اگر چنین باشد، مرورگر تمام آن کوکی‌ها را همراه با درخواست خود به سرویس‌دهنده وب می‌فرستد. وقتی سرویس‌دهنده این کوکی‌ها را دریافت کرد، خود می‌داند با آنها چه کند.

اجازه دهید چند نمونه از کاربرد کوکی‌ها را بررسی کنیم. در شکل ۷-۲۵، اولین کوکی متعلق به ناحیهٔ *toms-casino.com* است، و به کار شناسایی کاربر می‌آید. وقتی کاربر به خیال بردن مقداری پول وارد این سایت می‌شود، سرویس‌دهنده با استفاده از این کوکی وی را شناسایی می‌کند. برای مثال، سرویس‌دهنده وب می‌تواند به کمک این عدد مشخصات کاربر را از پایگاه دادهٔ خود استخراج کرده، و صفحه را به شکل مناسب (بسته به علائق قبلی کاربر) به نمایش در آورد.

کوکی دوم متعلق به سایت *joes-store.com* است. در این سناریو کاربر ما مشغول چرخیدن در یک فروشگاه مجازی و انتخاب کالاست. وقتی کاربر کالای مناسبی را یافت، روی آن کلیک می‌کند؛ با این کار، سرویس‌دهنده وب یک کوکی به کامپیوتر مشتری می‌فرستد، که در آن تعداد اجناس انتخاب شده و شمارهٔ آنها قید شده است. هر بار که کاربر صفحهٔ جدیدی را در این فروشگاه باز می‌کند (و در واقع درخواست جدیدی می‌فرستد)، این کوکی را هم به همراه آن به سرویس‌دهنده وب برمی‌گرداند. در مثال شکل ۷-۲۵ سه قلم جنس در سبد خرید مشتری وجود دارد، که از سومی دو عدد انتخاب شده است. در پایان هم وقتی مشتری پای صندوق می‌رود، سرویس‌دهنده با استفاده از همین کوکی می‌تواند قیمت کل خریدهای مشتری را به وی اعلام کند.

کوکی سوم متعلق به یکی از **درگاه‌های وب** (web portal) است. وقتی کاربر می‌خواهد وارد این سایت شود، مرورگر این کوکی را به همراه درخواست وی به سرویس‌دهنده وب می‌فرستد. سرویس‌دهنده وب هم صفحه‌ای به کاربر برمی‌گرداند که در آن نرخ سهام شرکتهای Sun Microsystems و Oracle ، و نتایج تیم فوتبال New Yprk Jets نشان داده شده است. از آنجائیکه یک کوکی می‌تواند تا 4 KB باشد، کاربر می‌تواند فهرست بلندبالایی از علائق خود را در این سایت ثبت کند.

یک سرویس‌دهنده وب می‌تواند از کوکی فقط برای مصارف داخلی خودش استفاده کند، مثلاً تعداد کاربرانش را بداند، و یا بداند که هر بازدیدکننده قبل از ترک سایت چند صفحه را دیده است. وقتی اولین بازدیدکننده وارد سایت می‌شود، هنوز هیچ کوکیی از آن در کامپیوترش ندارد، پس سرویس‌دهنده یک کوکی که در آن عبارت *Counter = 1* نوشته شده به مشتری می‌فرستد. کلیک بعدی کاربر باعث می‌شود تا همین کوکی به سرویس‌دهنده برگردد. سرویس‌دهنده هم مقدار *Counter* را یکی افزایش داده، و دوباره به مشتری برمی‌گرداند. بدین ترتیب

سرویس‌دهنده می‌تواند آماری از تعداد بازدیدکنندگان سایت (و اینکه هر کدام چند صفحه را دیده‌اند) بدست آورد.

امکان سوءاستفاده از کوکی‌ها نیز وجود دارد. در تئوری، مرورگر کوکی‌ها را فقط به سایتی که به آن تعلق دارند می‌فرستد، ولی برخی از هکرها توانسته‌اند با استفاده از باگهایی که در مرورگرها وجود دارد، کوکی‌هایی را که متعلق به آنها نیست بدست آورند. از آنجائیکه برخی از سایتهای تجارت الکترونیک شمارهٔ کارت اعتباری مشتریان خود را در کوکی‌ها ذخیره می‌کنند، می‌توانید خطرات بالقوهٔ این وضعیت را بروشنی دریابید.

نقطه ضعف دیگر کوکی‌ها (که بحثهای زیادی را هم بدنبال داشته) امکان استفاده از آنها برای جمع‌آوری مخفیانهٔ اطلاعات دربارهٔ کاربران و عادتهای وبگردی آنهاست. برای مثال، شرکت تبلیغاتی Sneaky Ads با سایتهای معروف تماس گرفته و از آنها می‌خواهد که (در ازای دریافت اجرت) اعلان تبلیغاتی این شرکت را بالای سایت خود نصب کنند (و همانطور که می‌دانید این بزرگترین منبع درآمد سایتهای وب است). اما Sneaky Ads بجای دادن تصویر GIF یا JPEG آگهی تبلیغاتی خود، یک URL به این شرکتها می‌دهد تا در صفحات خود قرار دهند. هر URL که این شرکت به طرفهای خود می‌دهد، یک شمارهٔ منحصربفرد دارد، مانند زیر

http://www.sneaky.com/382674902342.gif

وقتی کاربر وارد صفحهٔ یکی از این سایتها (مثلاً، صفحهٔ *P*) می‌شود، مرورگر فایل HTML آنرا می‌خواند، و بعد از آنکه متوجه شد در آن یک لینک به تصویری در سایت www.sneaky.com وجود دارد، درخواستی برای خواندن این تصویر به سایت مزبور می‌فرستد. سرویس‌دهنده این سایت به همراه تصویر آگهی، یک کوکی با شماره شناسایی منحصربفرد 3627239101 به مرورگر پس می‌فرستد (شکل ۷-۲۵ را ببینید). سپس، سایت Sneaky بازدید کاربر از صفحهٔ *P* را ثبت می‌کند (و این کاری ساده است، چون سرویس‌دهنده وب می‌داند که فایل *382674902342.gif* مربوط به صفحهٔ *P* است). البته احتمالاً آگهی همان است، فقط نام فایل آن در هر سایت فرق می‌کند.

بعدها وقتی کاربر وارد سایت دیگری که آگهی شرکت Sneaky Ads در آن قرار دارد، می‌شود مرورگر فایل HTML آنرا خوانده و سپس از سایت www.sneaky.com درخواست ارسال تصویر (مثلاً) *493654919923.gif* را می‌کند. اما از آنجائیکه مرورگر یک کوکی از سایت www.sneaky.com دارد، این کوکی را نیز همراه درخواست خود می‌فرستد؛ و اکنون Sneaky Ads دومین صفحه‌ای را که کاربر ما از آن بازدید کرده، می‌شناسد.

به مرور زمان، شرکت Sneaky Ads اطلاعات کاملی دربارهٔ تعداد زیادی از کاربران و عادتهای وبگردی آنها جمع آوری می‌کند (بدون اینکه آنها حتی یکبار روی آگهی‌های این شرکت کلیک کرده باشند!). البته این شرکت هنوز نام کاربر را نمی‌داند (اگرچه آدرس IP او را می‌داند، و همین برای بدست آوردن اطلاعات بعدی کافیست) - و اگر کاربر بیچاره حتی برای یک بار نامش را در اختیار یکی از شرکتهای همکار Sneaky Ads گذاشته باشد، که دیگر کار تمام است. امروزه فروش اطلاعات کاربران اینترنت و علائق آنها (به طالبان این قبیل اطلاعات) یکی از منابع سرشار درآمد در وب محسوب می‌شود. بدترین جنبهٔ این نوع جمع‌آوری اطلاعات آنست که کاربر حتی روحش از آن خبر ندارد، و فکر می‌کند چون روی هیچ آگهی اینترنتی کلیک نکرده کاملاً در امان است!

و اگر Sneaky Ads بخواهد پلیدی را به نهایت برساند، حتی لازم نیست یک آگهی کلاسیک بدهد؛ یک آگهی که فقط یک پیکسل (آن هم به رنگ زمینهٔ سایت!) باشد، برای کار او کافیست (مرورگر باز هم مجبور است برای این تصویر یک پیکسلی به سایت www.sneaky.com برود، و کوکی را بگیرد).

برخی از کاربران (برای حفظ حریم شخصی خود) مرورگر را طوری پیکربندی می‌کنند که تمام کوکی‌ها را رد کند. اما این کار در عملکرد سایتهای معتبری که به کوکی وابسته‌اند، نیز اختلال ایجاد می‌کند. برای حل این مشکل می‌توان از نرم‌افزارهایی که به کوکی‌خور (cookie-eating) معروفند، استفاده کرد. این نرم‌افزارها تمام کوکی‌هایی

را که به یک کامپیوتر می‌شوند، گرفته و فقط آنهایی را ذخیره می‌کنند که از نظر کاربر مجاز شناخته شده باشند. مرورگرهای جدید هم به کاربران امکان کنترل کامل کوکی‌ها را می‌دهند.

۷-۳-۲ سندهای وب استاتیک

اساس وب عبارتست از انتقال صفحات وب از سرویس‌دهنده به مشتری. صفحات وب، در ساده‌ترین شکل خود، استاتیک (ثابت و ایستا) هستند و فقط روی سرویس‌دهنده منتظرند تا کسی بیاید و آنها را بردارد. با این تعریف، حتی یک فیلم ویدئویی نیز استاتیک است، چون چیزی نیست جز یک فایل ساده. در این قسمت صفحات وب استاتیک را بتفصیل مورد بررسی قرار خواهیم داد.

HTML

صفحات وب با زبانی بنام HTML (زبان علامتگذاری آبرمتن – HyperText Markup Language) نوشته می‌شوند. با HTML می‌توان متن، گرافیک و لینک به صفحات وب اضافه کرد. HTML یک زبان علامتگذاریست، یعنی زبانی که نشان می‌دهد سند چگونه باید فرمت شود. «علامتگذاری» اصطلاحیست قدیمی که به دوران چاپ با حروف سربی برمی‌گردد، و در آن ویراستار با علامتگذاری متن نوشته به حروفچین نشان می‌داد که چگونه (با چه حروف و اندازه‌ای) باید صفحهٔ چاپی را بچینند. در زبانهای علامتگذاری فرمانهای مشخصی برای فرمت کردن سند وجود دارد. برای مثال، در HTML علامت فرمان شروع فرمت با فونت ضخیم، و فرمان پایان فونت ضخیم است. مزیت زبان علامتگذاری اینست که نوشتن مرورگر برای آن ساده است، چون فقط کافیست مرورگر فرمانهای علامتگذاری را بداند. زبانهای TeX و troff هم جزء زبانهای علامتگذاری معروف هستند.

با نوشتن فرمانهای علامتگذاری در فایل HTML و استاندارد کردن این فرمانها، تمام مرورگرها می‌توانند صفحات وب را خوانده و فرمت کنند. این ویژگی اهمیت بسیار زیادی در نمایش صحیح صفحات وب دارد، چون ممکنست یک صفحهٔ وب در کامپیوتری با وضوح 1200 × 1600 پیکسل و رنگ 24-bit ایجاد شده باشد، ولی کامپیوتر بازدیدکننده فقط وضوحی معادل 480 × 640 پیکسل و رنگ 8-bit داشته باشد.

در این قسمت HTML را بطور مختصر بررسی خواهیم کرد. یک سند HTML را می‌توان با هر ادیتوری نوشت، اما برای نوشتن صفحات HTML برنامه‌های خاصی نیز طراحی شده است، که امکانات بیشتری در اختیار فرد قرار می‌دهند (و در ضمن دست او را در ریزه‌کاریها می‌بندند).

هر صفحهٔ وب یک سر (head) و یک بدنه (body) دارد، که بین برچسبهای <html> و </html> قرار می‌گیرند، اگرچه اغلب مرورگرها نبودن این دو برچسب را نادیده می‌گیرند (برچسب – tag – فرمان فرمت HTML است که داخل < > نوشته می‌شود). همانطور که در شکل ۷-۲۶ (الف) می‌بینید، قسمت سر بین برچسبهای <head> و </head> ، و قسمت بدنه بین برچسبهای <body> و </body> محصور می‌شوند. دستورات HTML داخل این برچسبها نوشته می‌شود. اغلب فرمانهای HTML دارای چنین شکلی هستند، یعنی با <something> شروع، و به </something> ختم می‌شوند. در اغلب مرورگرها فرمانی وجود دارد بنام VIEW SOURCE (یا چیزی شبیه آن)، که به کمک آن می‌توان (بجای خروجی فرمت‌شده) محتویات فایل HTML را دید.

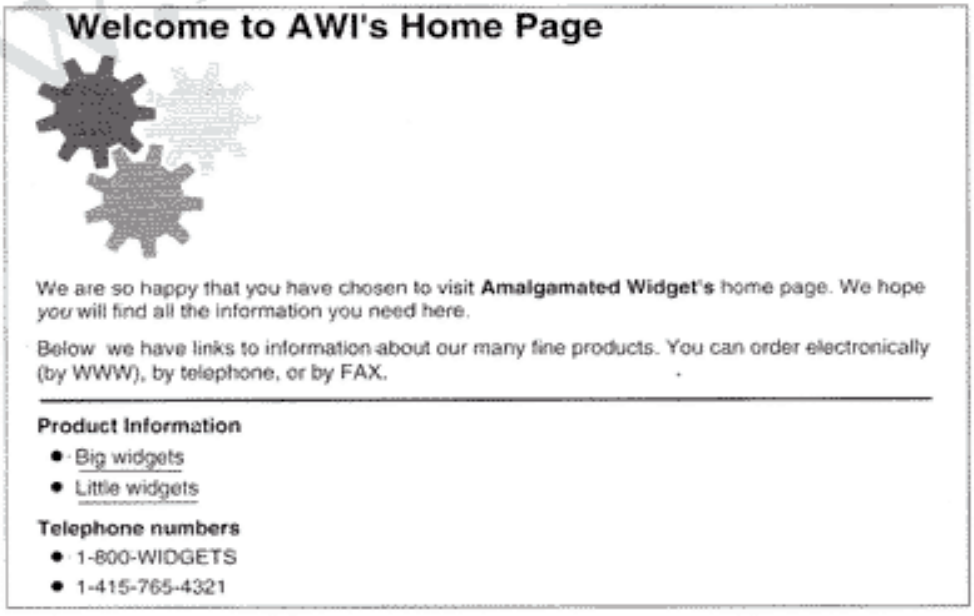
نوع حروف در برچسبهای HTML تفاوتی ندارد، بعبارت دیگر برچسبهای <head> و <HEAD> یکی هستند؛ البته در استانداردهای جدید فقط می‌توان از حروف کوچک برای نوشتن برچسبها استفاده کرد. فرمت خود فایل HTML هیچ تأثیری روی خروجی آن ندارد، چون مرورگرها مجبورند صفحات را در کامپیوترهای مختلف

نمایش دهند، و بهمین دلیل فاصله‌ها و فضاهای خالی را حذف کرده و صفحه را متناسب با تنظیمات کامپیوتر مقصد از نو فرمت می‌کنند. البته نویسندهٔ فایل HTML می‌تواند از فاصله و فضای خالی برای خواناتر کردن آن استفاده کند (که اتفاقاً چیز بسیار خوبی هم هست). در نتیجه، برای فاصله انداختن بین پاراگرافها نمی‌توانید از Enter استفاده کنید: برای این کار برچسب مخصوصی وجود دارد.

بعضی از برچسبها دارای پارامترهای نام‌دار (named parameter) هستند، که به صفت (attribute) معروفند.

```
<html>
<head> <title> AMALGAMATED WIDGET, INC. </title> </head>
<body> <h1> Welcome to AWI's Home Page </h1>
<img src="http://www.widget.com/images/logo.gif" ALT="AWI Logo"> <br>
We are so happy that you have chosen to visit <b> Amalgamated Widget's</b>
home page. We hope <i> you </i> will find all the information you need here.
<p>Below we have links to information about our many fine products.
You can order electronically (by WWW), by telephone, or by fax. </p>
<hr>
<h2> Product information </h2>
<ul>
   <li> <a href="http://widget.com/products/big"> Big widgets </a>
   <li> <a href="http://widget.com/products/little"> Little widgets </a>
</ul>
<h2> Telephone numbers </h2>
<ul>
   <li> By telephone: 1-800-WIDGETS
   <li> By fax: 1-415-765-4321
</ul>
</body>
</html>
```

(الف)



(ب)

شکل ۷-۲۶. (الف) کد HTML برای صفحه وب نمونه (ب) صفحه قالب‌دهی شده بر روی مرورگر.

مثلاً، در فرمان

```
<img src="abc" alt="foobar">
```

برچسب <img> دو پارامتر بنامهای *src* و *alt* دارد، که مقدار آنها بترتیب *abc* و *foobar* است. در استاندارد HTML فهرستی از پارامترهای مجاز هر برچسب و مفهوم آنها داده شده است. از آنجائیکه این پارامترها نام‌دار هستند، ترتیب نوشتن آنها اهمیتی ندارد.

از نظر فنی، سندهای HTML با مجموعه کاراکتر ISO 8859-1 Latin-1 نوشته می‌شوند، ولی از آنجائیکه اغلب کاربران از صفحه کلیدهای ASCII استفاده می‌کنند، برای نمایش کاراکترهای خاص (مانند è) می‌توان از توالی‌گریز (escape sequence) کمک گرفت. این کاراکترها (که فهرست آنها در استاندارد HTML آمده) با & شروع و به ; ختم می‌شوند. برای مثال، توالی گریز   معادل یک فاصله (space) ، توالی گریز è معادل è ، و توالی گریز é معادل é است. از آنجائیکه > ، < و & در فایلهای HTML معنای خاصی دارند، برای نمایش آنها نیز باید از توالی‌های گریز (بترتیب < ، > و &) استفاده کرد.

مهمترین چیزی که در قسمت سر نوشته می‌شود عنوان صفحهٔ وب است، که بین برچسبهای <title> و </title> قرار می‌گیرد (البته برخی اطلاعات دیگر، که به اطلاعات‌متا معروفند، نیز در قسمت سر نوشته می‌شوند). هر چیزی که در برچسب <title> نوشته شود، در خود صفحهٔ وب دیده نخواهد شد، بلکه در میلهٔ عنوان پنجرهٔ مرورگر ظاهر می‌شود.

اجازه دهید نگاهی به قسمتهای دیگر شکل ۷-۲۶ بیندازیم. برچسبهایی که در این فایل بکار رفته (و چندبرچسب دیگر) را در شکل ۷-۲۷ ملاحظه می‌کنید. برای نوشتن تیتر مطالب می‌توانید از برچسب <h*n*> ، که *n* عددی بین 1 تا 6 است، استفاده کنید: <h1> بزرگترین تیتر، و <h6> کوچکترین تیتر را تولید می‌کنند. فرمت کردن تیترها (اندازه و نوع فونت یا رنگ آنها) بر عهدهٔ مرورگر است، و معمولاً تیترهای درشت‌تر اعداد کوچکتری دارند. برچسب <h1> که بزرگترین تیتر را تولید می‌کند، دارای بیشترین فاصلهٔ خالی در بالا و پائین نیز هست، در حالیکه تیترهای کوچکتر فاصلهٔ کمتری با خطوط دیگر دارند.

برچسب	مفهوم
<html> ... </html>	صفحه وب را تعریف می کند
<head> ... </head>	محتویات سرصفحه را مشخص می کند
<title> ... </title>	عنوان صفحه را تعیین می کند (که البته در خود صفحه دیده نمی شود)
<body> ... </body>	بدنه صفحه را مشخص می کند
<h *n*> ... </h*n*>	سطح (اندازه) عنوان را تیتر را مشخص می کند
...	محتویات برچسب را ضخیم می کند
...	محتویات برچسب را کج می کند
...	محتویات برچسب را واسط صفحه وب قرار می دهد
...	یک لیست غیر منظم (گلوله دار) می سازد
...	یک لیست شماره دار می سازد
...	آیتمهای لیست را مشخص می کند
 	یک شکست خط در صفحه ایجاد می کند
<p>	یک پاراگراف جدید می سازد
<hr>	یک خط افقی روی صفحه رسم می کند
<img src="...">	یک تصویر روی صفحه نمایش می دهد
 ... 	یک لینک را تعریف می کند

شکل ۷-۲۷. چند برچسب HTML . برخی از این برچسبها می‌توانند پارامتر داشته باشند.

برچسبهای <b> و <i> بترتیب برای ضخیم و کج کردن فونت بکار می‌روند. اگر مرورگر نتواند فونتهای ضخیم یا کج را نمایش دهد، معمولاً آنها بگونه‌ای دیگر (رنگ متفاوت یا نگاتیو کردن حروف) از سایر قسمتها متمایز خواهد کرد.

در HTML مکانیزمها مختلفی برای ایجاد لیست (از جمله لیستهای تودرتو) وجود دارد. لیست‌ها با برچسب <ul> یا <ol> شروع می‌شوند، که در هر دو مورد برای مشخص کردن آیتمهای لیست از برچسب <li> استفاده می‌کنیم. برچسب <ul> یک لیست‌مرتب‌نشده (unordered list) می‌سازد، که آیتمهای آن با گلوله (bullet) (● - مشخص می‌شوند. برچسب <ol> یک لیست‌مرتب‌شده (ordered list) می‌سازد، که آیتمهای آن توسط مرورگر شماره‌گذاری خواهند شد.

برچسبهای
 ، و <hr> بین قسمتهای مختلف متن فاصله می‌اندازند. برای فرمت کردن دقیق متن می‌توان از شیوه‌نامه (style sheet) نیز استفاده کرد. برچسب
 باعث می‌شود تا ادامهٔ متن از سر خط شروع شود (معمولاً مرورگرها بعد از
 خط خالی اضافه نمی‌کنند). برچسب پاراگراف جدیدی را شروع می‌کند و بین آنها یک خط فاصله می‌اندازد (و حتی ممکنست تورفتگی – indent –ابتدای پاراگراف را هم رعایت کند). از نظر تئوری یک پاراگراف باید با شروع و به ختم شود، ولی برچسب بندرت بکار برده می‌شود، و حتی اغلب آنهایی که صفحات وب می‌نویسند از وجود آن خبر ندارد. برچسب <hr> نیز یک خط افقی روی صفحه رسم می‌کند.

صفحات وب می‌توانند تصویر نیز داشته باشند. یک برچسب <img> تصویر مشخص شده را در همان نقطه‌ای که این برچسب نوشته شده، نمایش می‌دهد. برچسب <img> می‌تواند پارامترهای متعددی بگیرد. پارامتر *src* همان URL تصویر موردنظر است. فرمت این تصاویر جزیی از استاندارد HTML نیست، بلکه جزء قابلیتهای مرورگر محسوب می‌شود. اغلب مرورگرها می‌توانند تصاویر GIF و JPEG را نمایش دهند. مرورگرها می‌توانند از فرمتهای دیگر نیز پشتیبانی کنند، ولی این یک شمشیر دو لبه است: اگر عادت کنید از فرمتهای دیگر (مثلاً، BMP) در صفحات وب خود استفاده کنید، شاید بعضی از کاربران (که مرورگرهای متفاوتی دارند) نتوانند خلاقیت هنری شما را تحسین کنند!

برخی دیگر از پارامترهای <img> عبارتند از: *align* که تصویر را با خط کرسی متن تراز می‌کند (مقادیر *top* ، *middle* ، و *bottom* بترتیب معادل تراز بالا، وسط و پائین هستند)؛ *alt* که متن جایگزین تصویر را (وقتی کاربر تصاویر را غیرفعال کرده باشد) مشخص می‌کند؛ و *ismap* که نشان می‌دهد تصویر دارای نقاط فعال (قابل‌کلیک) است.

برای ایجاد اَبَرلینک در صفحات وب از برچسب <a>... استفاده می‌کنیم. برچسب <a> هم پارامترهای مختلفی دارد، از جمله *href* (URL مقصد لینک)، و *name* (نام اَبَرلینک). متن (یا تصویری) که بین <a> و قرار دارد، در صفحهٔ وب دیده خواهد شد، و کاربر با کلیک کردن این متن (یا تصویر) می‌تواند به صفحهٔ مقصد لینک برود. در زیر نمونه‌ای از یک اَبَرلینک را می‌بینید.

```
<a href="http://www.nasa.gov">NASA's home page</a>
```

که کاربر آنرا در مرورگر به این صورت خواهد دید:

NASA's home page

و اگر کاربر کنجکاو ما به ناسا علاقمند باشد و روی این لینک کلیک کند، مرورگر بلافاصله صفحهٔ *http://www.nasa.gov* را آورده و نمایش می‌دهد.

مثال زیر شکل دیگری از همان لینک بالاست:

```
<a href="http://www.nasa.gov"><img src="shuttle.gif" alt="NASA"></a>
```

در اینجا بجای عبارت NASA's home page از تصویر یک شاتل فضایی استفاده کرده‌ایم، و کاربر با کلیک

کردن این تصویر به همان صفحهٔ *http://www.nasa.gov* خواهد رفت. اگر هم کاربر نمایش تصاویر را در مرورگر غیرفعال کرده باشد، کلمهٔ NASA نشان می‌دهد که موضوع از چه قرار است.

برچسب <a> پارامتری دارد بنام *name* ، که به کمک آن می‌توان لینک را به وسط یک صفحه نشانه رفت. با این پارامتر می‌توان صفحاتی بصورت فهرست مطالب ایجاد کرد، که کلیک کردن آیتمهای آن باعث رفتن کاربر به قسمت مربوطه در همان صفحه می‌شود.

استاندارد HTML از تغییر و تکامل در امان نبوده است. در ویرایشهای HTML 1.0 و HTML 2.0 جدول (table) وجود نداشت، ولی از HTML 3.0 این عنصر هم به صفحات وب اضافه شد. هر جدول HTML تعدادی سطر دارد، که خود از یک یا چند سلول (cell) تشکیل شده است. در یک سلول می‌توان هر چیزی قرار دارد: متن، عدد، تصویر، و حتی یک جدول دیگر. سلولها را می‌توان در هم ادغام کرد، برای مثال می‌توان با ادغام چند سلول برای جدول تیتر یا عنوان درست کرد. جدولها حرف آخر را در فرمت کردن صفحات وب (و کنترل خصوصیات ظاهری آن) می‌زنند.

در شکل ۷-۲۸ (الف) تعریف یک جدول HTML ، و در شکل ۷-۲۸ (ب) خروجی آنرا می‌بینید. این مثال فقط برخی از ویژگیهای ابتدایی جدول را نشان می‌دهد. جدولها با برچسب <table> شروع می‌شوند، و با دادن پارامترهای دلخواه می‌توان ویژگیهای کلی آنها را تعیین کرد.

با استفاده از برچسب <caption> می‌توان یک تیتر کلی به جدول داد. هر سطر جدول با برچسب <tr> (مخفف Table Row) شروع می‌شود، و برای مشخص کردن محتویات سلولها از برچسب‌های <th> (مخفف Table Header) یا <td> (مخفف Table Data) استفاده می‌شود (مرورگر برچسبهای <th> و <td> را بگونه‌ای متفاوت فرمت می‌کند). جدولها دارای صفات متعددی دیگری (از جمله نوع تراز افقی و عمودی سلولها، حاشیهٔ آنها، و ادغام سلولها در یکدیگر) نیز هستند.

در HTML 4.0 قابلیتهای جدیدی به این زبان اضافه شده است، که از میان آنها می‌توان به امکانات جدید برای سهولت و یگردی معلولین، قابیلت استفاده از شیء (object) در صفحات وب، پشتیبانی از زبانهای اسکریپت‌نویسی (برای ایجاد محتویات دینامیک) اشاره کرد.

در سایتهای بزرگ وب که طراحان متعددی در ایجاد صفحات آن شرکت دارند، یکی از مهمترین نکات حفظ یکدستی و یکنواختی ظاهر صفحات است. این مشکل را می‌توان با استفاده از **شیوه‌نامه** (style sheet) حل کرد. در این تکنیک طراحان صفحات وب بجای شیوه‌های فیزیکی (مانند فونت ضخیم یا کج)، از شیوه‌های منطقی مانند <dn> (تعریف)، (تأکید خفیف)، (تأکید شدید)، و <var> (متغیرهای برنامه) استفاده می‌کنند. این شیوه‌های منطقی در یک شیوه نامه (که در ابتدای هر صفحه به آن ارجاع می‌شود) تعریف می‌شوند. با استفاده از شیوه نامه، سرپرست تیم طراحی می‌تواند ظاهر یکنواخت کلیهٔ صفحات را تضمین کند. برای مثال، اگر روزی تصمیم گرفته شود که برچسب از فونت 14 کج آبی به فونت 18 ضخیم صورتی پُررنگ تبدیل شود، فقط کافیست تعریف این برچسب در شیوه نامه تغییر کند. شیوه نامه‌ها را می‌توان معادل فایلهای #include در برنامه‌های C (که محل تعریف ماکروهاست) دانست.

فرم

اولین ویرایش HTML (یعنی HTML 1.0) اساساً یک طرفه بود: کاربران می‌توانستند صفحات را درخواست کنند، ولی فرستادن اطلاعات به سرویس‌دهندهٔ وب بسیار مشکل بود. با رشد کاربردهای تجاری وب، تقاضا برای ترافیک دو طرفه بشدت فزونی گرفت. برای مثال، شرکتهای بسیاری میل داشتند از طریق وب هم سفارش بگیرند، یا شرکتهای نرم‌افزاری می‌خواستند محصولات خود را بصورت الکترونیکی بفروشند، و دهها مورد مانند آن.

```
<html>
<head> <title> A sample page with a table </title> </head>
<body>
<table border=1 rules=all>
<caption> Some Differences between HTML Versions </caption>
<col align=left>
<col align=center>
<col align=center>
<col align=center>
<col align=center>
<tr> <th>Item  <th>HTML 1.0  <th>HTML 2.0  <th>HTML 3.0 <th>HTML 4.0 </tr>
<tr> <th> Hyperlinks <td> x <td> x <td> x <td> x </tr>
<tr> <th> Images <td> x <td> x <td> x <td> x </tr>
<tr> <th> Lists <td> x <td> x <td> x <td> x </tr>
<tr> <th> Active Maps and Images <td>   <td> x <td> x <td> x </tr>
<tr> <th> Forms <td>   <td> x <td> x <td> x </tr>
<tr> <th> Equations <td>   <td>   <td> x <td> x </tr>
<tr> <th> Toolbars <td>   <td>   <td> x <td> x </tr>
<tr> <th> Tables <td>   <td>   <td> x <td> x </tr>
<tr> <th> Accessibility features <td>   <td>   <td>   <td> x </tr>
<tr> <th> Object embedding <td>   <td>   <td>   <td> x </tr>
<tr> <th> Scripting <td>   <td>   <td>   <td> x </tr>
</table>
</body>
</html>
```

(الف)

برخی از تفاوتهای در ویرایش HTML

آیتم	HTML 1.0	HTML 2.0	HTML 3.0	HTML 4.0
ابر لینک	X	X	X	X
تصویر	X	X	X	X
لیست	X	X	X	X
تصویر یا نقش فعال		X	X	X
فرم		X	X	X
معادله			X	X
میله ابزار			X	X
جدول			X	X
ویژگی معلولین				X
قرار دادن شی ء در صفحه				X
اسکریپت نویسی				X

(ب)

شکل ۷-۲۸. (الف) یک جدول HTML . (ب) خروجی جدول.

این تقاضاها باعث شد تا فرم (form) به ویرایش بعدی یعنی HTML 2.0 اضافه شود. فرم می‌تواند فیلدها (یا دکمه‌هایی) داشته باشد، که کاربر آنها را پُر کرده و به سرویس‌دهنده برگرداند. برای این منظور از برچسبی بنام <input> استفاده می‌کنیم، که دارای پارامترهای متعددی برای کنترل ظاهر و عملکرد آن است. رایجترین فرمها معمولاً دارای یک یا چند فیلد برای وارد کردن متن، یک یا چند جعبه برای انتخاب کردن، نگاشتهای فعال، و دکمه‌های *submit* باشند. در شکل ۷-۲۹ برخی از خصوصیات فرم نشان داده شده است.

اجازه دهید برای آشنایی بیشتر با فرمها، این مثال را دقیقتر بررسی کنیم. هر فرم با برچسبهای <form> و </form> مشخص می‌شود. در داخل یک فرم می‌توان از تمام برچسبهای HTML استفاده کرد، اما هر متنی که خارج این برچسبها نوشته شود، بهمان صورت دیده خواهد شد. سه نوع **جعبهٔ‌ورودی** (input box) وجود دارد که می‌توان از آنها در فرمها استفاده کرد.

در شکل ۷-۲۹، اولین جعبهٔ ورودی بعد از کلمهٔ "Name" آمده است. این جعبه ورودی رشته‌ایی بطول حداکثر ۴۶ کاراکتر می‌گیرد، و آنرا در متغیری بنام *customer* ذخیره می‌کند. برچسب <p> هم باعث می‌شود که مرورگر فیلدهای بعدی را در خط بعد نشان دهد. با استفاده از برچسب <p> طراح صفحه می‌تواند ظاهر آنرا بصورت دلخواه کنترل کند.

خط بعدی فرم یک آدرس (بطول ۴۰ کاراکتر) از کاربر می‌گیرد. فیلدهای شهر، استان، و کشور هم بدنبال آن آمده‌اند؛ بین این فیلدها <p> وجود ندارد، بنابراین مرورگر آنها را در یک خط نشان می‌دهد. تا آنجاکه به مرورگر مربوط است، این پاراگراف شش آیتم دارد (سه رشتهٔ متن، و سه جعبهٔ ورودی)، که باید آنها را یک خط نمایش دهد (و اگر نتوانست به خط بعدی می‌رود). اگر وضوح تصویر مانیتور زیاد باشد، این شش آیتم در یک خط دیده خواهند شد، اما اگر وضوح آن پائین باشد، احتمالاً به دو خط شکسته می‌شوند (حتی ممکنست کلمهٔ "Country" در آخر خط، و جعبهٔ ورودی مقابل آن در ابتدای خط بعدی بیفتد).

خط بعدی شمارهٔ کارت اعتباری و تاریخ انقضای آنرا از کاربر می‌گیرد (البته ارسال این قبیل اطلاعات روی اینترنت فقط باید در شرایط کاملاً امن صورت گیرد - در فصل ۸ در این باره بیشتر صحبت خواهیم کرد).

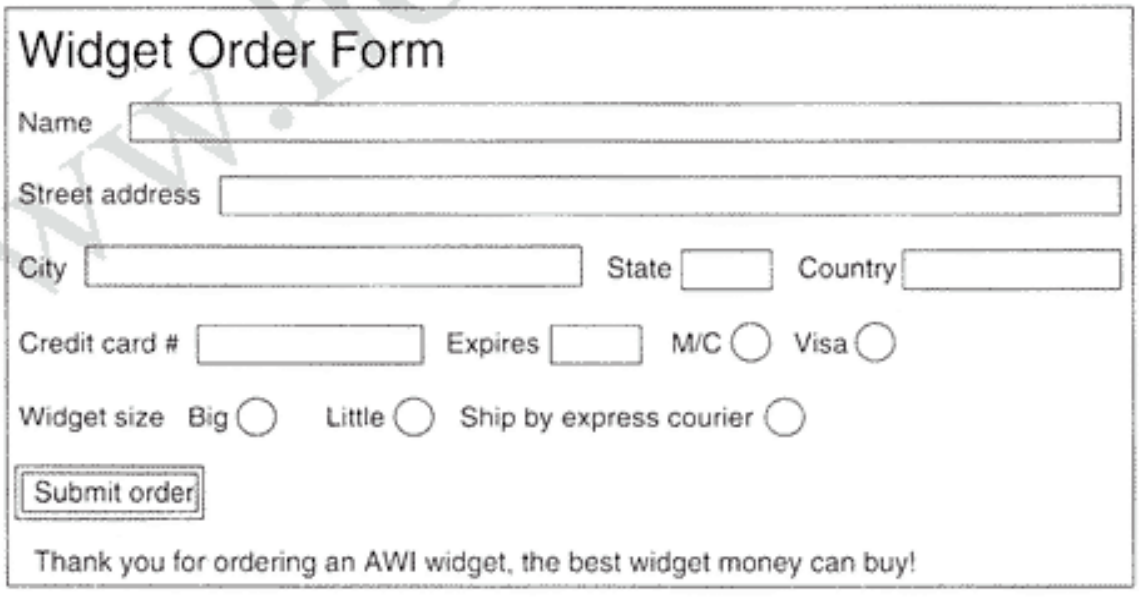
بعد از فیلد تاریخ انقضا، نوع دیگری از جعبه ورودی را می‌بینید: **دکمهٔ‌رادیویی** (radio button) . از دکمهٔ رادیویی وقتی استفاده می‌شود که بخواهیم انتخاب کاربر را به دو یا چند گزینه محدود کنیم (مانند رادیوی ماشین که با زدن هر دکمه می‌توان یک ایستگاه از پیش تعیین شده را انتخاب کرد). کاربر می‌تواند برای انتخاب هر یک از این دکمه‌ها روی آن کلیک کند (و یا از صفحه کلید استفاده کند). انتخاب هر دکمه، باعث می‌شود که دکمه‌های دیگر همان گروه خاموش شوند (نمایش ظاهری دکمه‌های خاموش و روشن بر عهدهٔ مرورگر است). گروه‌بندی دکمه‌های رادیویی (و ایجاد ارتباط منطقی بین آنها) توسط صفت *name* صورت می‌گیرد. برای مثال، Widget size گروه مستقلیست که آن هم دو دکمه دارد.

برای مشخص کردن اینکه در هر گروه کدام دکمه روشن است، از پارامتر *value* استفاده می‌کنیم. برای مثال، در گروه اول مقدار متغیر cc ("mastercard" یا "visacard") نشان می‌دهد که کاربر کدامیک از دکمه‌های M/C یا Visa را انتخاب کرده است.

بعد از این دکمه‌های رادیویی، نوع دیگری از جعبه ورودی را می‌بینید: **جعبه‌چک** (checkbox) . یک جعبه چک نیز مانند دکمهٔ رادیویی می‌تواند دو حالت داشته باشد: روشن، خاموش. اما بر خلاف دکمه‌های رادیویی، هر جعبه چک می‌تواند مستقلاً خاموش یا روشن باشد. برای مثال، وقتی می‌خواهید از طریق وب سفارش پیتزا بدهید. می‌توانید قارچ، گوشت و پیاز را همزمان بعنوان ترکیبات یک پیتزا انتخاب کنید، ولی امکان انتخاب همزمان اندازه‌های کوچک، متوسط و بزرگ را برای یک پیتزا ندارید. بهمین دلیل ترکیبات پیتزا را با استفاده از جعبه چک نشان می‌دهند، و اندازهٔ آنرا با دکمه‌های رادیویی.

```
<html>
<head> <title> AWI CUSTOMER ORDERING FORM </title> </head>
<body>
<h1> Widget Order Form </h1>
<form ACTION="http://widget.com/cgi-bin/widgetorder" method=POST>
<p> Name <input name="customer" size=46> </p>
<p> Street Address <input name="address" size=40> </p>
<p> City <input name="city" size=20> State <input name="state" size =4>
Country <input name="country" size=10> </p>
<p> Credit card # <input name="cardno" size=10>
Expires <input name="expires" size=4>
M/C <input name="cc" type=radio value="mastercard">
VISA <input name="cc" type=radio value="visacard"> </p>
<p> Widget size Big <input name="product" type=radio value="expensive">
Little <input name="product" type=radio value="cheap">
Ship by express courier <input name="express" type=checkbox> </p>
<p><input type=submit value="submit order"> </p>
Thank you for ordering an AWI widget, the best widget money can buy!
</form>
</body>
</html>
```

(الف)



(ب)

شکل ۷-۲۹. (الف) یک فرم HTML . (ب) خروجی فرمت شدهٔ فرم.

در مواردی که تعداد گزینه‌های موجود برای یک انتخاب بسیار زیاد باشد، استفاده از دکمه‌های رادیویی قدری مشکل است. در این موارد می‌توان از برچسب <select> استفاده کرد: این برچسب با پارامتر *multiple* مانند

جعبه چک عمل می‌کند، و بدون آن مانند دکمهٔ رادیویی. در اغلب مرورگرها برچسب <select> بصورت یک منوی بازشو (drop-down menu) نمایش داده می‌شود.

تا اینجا دو نوع <input> را دیدید: *radio* و *checkbox* . البته نوع سوم را هم قبل از آن دیده بودید: نوع *text* ؛ ولی از آنجائیکه این نوع پیش‌فرض <input> است، استفاده از صفت *type = text* ضرورتی ندارد. انواع دیگر جعبه ورودی عبارتند از: *password* و *textarea* . جعبهٔ *password* درست مانند *text* است، با این تفاوت که هر چیزی که در آن نوشته شود، فقط * نشان می‌دهد. جعبهٔ *textarea* نیز مانند *text* است، فقط چندخطی است.

در آخرین قسمت از فرم شکل ۷-۲۹ یک دکمهٔ *submit* می‌بینید. وقتی کاربر این دکمه را کلیک کند، اطلاعاتی که در سایر قسمتهای فرم وارد کرده به کامپیوتری که فرم روی آن قرار دارد، فرستاده می‌شود. در اینجا پارامتر *value* عبارتیست که روی دکمهٔ *submit* دیده خواهد شد. دکمهٔ *submit* تنها جعبه ورودی است که باید *value* داشته باشد، در سایر جعبه‌ها این پارامتر اختیاری است (چون کاربر می‌تواند متن موجود در آنها را بدلخواه تغییر دهد).

با کلیک شدن دکمهٔ *submit* ، مرورگر تمام اطلاعات فرم را در یک خط ترکیب کرده و به سرویس‌دهنده برمی‌گرداند. این فیلدها با & از هم جدا می‌شوند، و اگر در آنها فاصله وجود داشته باشد، مرورگر بجای آن + قرار می‌دهد. در شکل ۷-۳۰ نمونه‌ای از مقدار برگشتی فرم ۷-۲۹ به سرویس‌دهنده را می‌بینید (تمام این اطلاعات در واقع فقط یک خط است، که بدلیل محدودیت عرض صفحه به چند خط شکسته شده است).

```
customer=John+Doe&address=100+Main+St.&city=White+Plains&
state=NY&country=USA&cardno=1234567890&expires=6/98&cc=mastercard&
product=cheap&express=on
```

شکل ۷-۳۰. وردودیهای کاربر که مرورگر به سرویس‌دهنده برمی‌گرداند.

(اگر یک جعبه چک انتخاب نشده باشد، مرورگر آنرا به سرویس‌دهنده نمی‌فرستد.) تفسیر اطلاعات رسیده بر عهدهٔ سرویس‌دهنده است - در این باره بعداً بیشتر صحبت خواهیم کرد.

XML و XSL

HTML ، با فرم یا بدون آن، هیچگونه ساختاری برای صفحات وب فراهم نمی‌آورد. در HTML محتوای صفحه و فرمانهای فرمت‌کننده نیز مخلوط هستند. با گسترش روزافزون تجارت الکترونیک (و سایر کاربردها)، نیاز به نوعی ساختار برای صفحات وب (و تفکیک محتوا و فرمت) بالا گرفت. برای مثال، برنامه‌ای که می‌خواهد در وب بدنبال بهترین قیمت یک کتاب (یا CD) جستجو کند، باید صفحات متعددی را خوانده و در آنها بدنبال عنوان کتاب و قیمت آن بگردد. وقتی صفحات وب با HTML نوشته شده باشند، برای چنین برنامه‌ای دشوار است تشخیص دهد عنوان کتاب کجاست و قیمت آن کجا.

بهمین دلیل کنسرسیوم W3C استاندارد جدیدی برای HTML توسعه داده، که به کمک آن می‌توان به صفحات وب ساختاری مناسب برای پردازش خودکار داد. برای این هدف دو زبان جدید نیز توسعه داده شده است. اولی، XML (زبان علامتگذاری قابل توسعه - eXtensible Markup Language)، محتویات صفحهٔ وب را بصورت ساخت‌یافته توصیف می‌کند، و دومی، XSL (زبان شیوه‌نویسی قابل توسعه - eXtensible Style Language)، برای توصیف مستقل فرمت این محتواست. هر دوی این زبانها بسیار مفصل و پیچیده‌اند، و بحث این قسمت فقط ایده‌ای از طرز کار آنها به شما خواهد داد.

مثال شکل ۷-۳۱ را در نظر بگیرید. در این مثال ساختاری بنام book_list ، برای نگهداری مشخصات کتابها، تعریف شده است. هر کتاب سه فیلد دارد: عنوان، مؤلف، و سال انتشار. این ساختار بسیار ساده است، اما می‌توان ساختارهای پیچیده‌تری هم ایجاد کرد (مانند کتابهایی که چند مؤلف دارند، کتابهایی که CD پیوست دارند، و یا URL سایتهای عرضه و فروش کتاب).

در این مثال هر فیلد فقط یک قسمت دارند، اما فیلدها را می‌توان به فیلدهای کوچکتری نیز تقسیم کرد. بعنوان مثال، می‌توان فیلد <author> را به نام و نام خانوادگی تقسیم کرد، تا جستجوهای دقیقتر ممکن شود (این تقسیم را می‌توان تا هر درجه‌ای از عمق انجام داد):

```
<author>
    <first_name>Andrew</first_name>
    <last_name>Tanenbaum</last_name>
</author>
```

تمام کاری که فایل ۷-۳۱ انجام می‌دهد، ایجاد فهرستی از سه کتاب است. این فایل هیچ حرفی دربارهٔ نحوهٔ نمایش این اطلاعات نمی‌زند. برای فرمت کردن این اطلاعات به فایل دیگری نیاز داریم: فایل *book_list.xsl* ، که حاوی تعریفهای XSL لازم برای فرمت کردن فهرست *book_list.xml* است. این فایل یک شیوه‌نامه است که نحوهٔ فرمت کردن صفحهٔ وب را بیان می‌کند. (البته روشهای دیگری برای فرمت کردن فایلهای XML وجود دارد - مانند تبدیل XML به HTML - که از حوزهٔ بحث این کتاب خارج است.)

```
<?xml version="1.0" ?>
<?xml-stylesheet type="text/xsl" href="b5.xsl"?>
<book_list>
<book>
   <title> Computer Networks, 4/e </title>
   <author> Andrew S. Tanenbaum </author>
   <year> 2003 </year>
</book>
<book>
   <title> Modern Operating Systems, 2/e </title>
   <author> Andrew S. Tanenbaum </author>
   <year> 2001 </year>
</book>
<book>
   <title> Structured Computer Organization, 4/e </title>
   <author> Andrew S. Tanenbaum </author>
   <year> 1999 </year>
</book>
</book_list>
```

شکل ۷-۳۱. یک صفحهٔ وب که با XML نوشته شده است.

در شکل ۷-۳۲ یک فایل XSL نمونه برای فرمت کردن فایل ۷-۳۱ را ملاحظه می‌کنید. بعد از چند تعریف لازم (مانند URL استاندارد XSL)، برچسبهای <html> و <body> آمده‌اند، که اینها (مانند همیشه) شروع صفحهٔ وب را مشخص می‌کنند. پس از آن جدولی با یک تیتر و سه ستون تعریف کرده‌ایم. دقت کنید که در این

جدول علاوه بر برچسبهای <th> از برچسبهای </th> نیز استفاده کرده‌ایم، کاری که قبلاً نمی‌کردیم. استانداردهای XML و XSL بسیار سختگیرتر از HTML هستند، و رعایت اصول نوشتن برچسبها در آنها لازم است، حتی اگر مرورگر قادر به تشخیص نیت طراح صفحهٔ وب باشد. مرورگری که فایلهای غلط XML و XSL را قبول کند (و خود اشکالات آنها را رفع کند)، در امتحانات سازگاری نمرهٔ قبولی نخواهد گرفت - مرورگرها فقط می‌توانند خطاها را اعلام کنند. این مقررات سختگیرانه برای مقابله با افراد تنبلی که وب را پُر از صفحات شلخته کرده‌اند، لازم است.

```
<?xml version='1.0'?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
<xsl:template match="/">
<html>
<body>
<table border="2">
  <tr>
    <th> Title</th>
    <th> Author</th>
    <th> Year </th>
  </tr>

  <xsl:for-each select="book_list/book">
  <tr>
    <td> <xsl:value-of select="title"/> </td>
    <td> <xsl:value-of select="author"/> </td>
    <td> <xsl:value-of select="year"/> </td>
 </tr>
 </xsl:for-each>
</table>
</body>
</html>
</xsl:template>
</xsl:stylesheet>
```

شکل ۷-۳۲. یک شیوه‌نامهٔ XSL.

دستور

```
<xsl:for-each  select="book_list/book">
```

معادل دستور for در زبان C است. این دستور (که به </xsl:for-each> ختم می‌شود) در یک حلقه کتابها را بترتیب می‌خواند. در هر تکرار این حلقه پنج فرمان HTML در خروجی نوشته می‌شود: <tr> ، عنوان کتاب، مؤلف، سال انتشار کتاب، و </tr> . پس از اتمام حلقه و بستن جدول، برچسبهای </body> و </html> نیز در خروجی نوشته می‌شوند. حاصل کار یک جدول HTML است، که کاملاً شبیه جدولهای معمولی است. اما با این‌فرمت هر برنامه‌ای می‌تواند فایل XML را آنالیز کرده، و کتابهای موردنظر را استخراج کند. تأکید بر این نکته ضروریست که، اگر چه در فایل XSL نوعی حلقه وجود دارد ولی صفحات XML و XSL همچنان استاتیک هستند، چون فقط برای فرمت کردن صفحهٔ وب بکار می‌آیند. البته مرورگر باید توانایی تفسیر فایلهای XML و XSL را داشته باشد، ولی خبر خوب اینست که اغلب مرورگرهای امروزی چنین قابلیتی دارند. آیندهٔ XSL و اینکه آیا می‌تواند جایگزین شیوه‌نامه‌ها شود، هنوز در پرده‌ای از ابهام قرار دارد.

با اینکه روش کار را نشان ندادیم، XML به طراح سایت وب اجازه می‌دهد تا با تعریف فایلهای ساختاری و ضمیمه کردن آنها به صفحات وب، سایتهایی پیچیده‌ای ایجاد کند. کتابهای متنوع و بسیار خوبی در این زمینه نوشته شده، که برای نمونه می‌توان به (Livingston, 2002; and Williamson, 2001) اشاره کرد.

قبل از پایان دادن به بحث XML و XSL ، بد نیست به جنگ ایدئولوژیکی که بین کنسرسیوم WWW و طراحان وب در گرفته، هم اشاره کنیم. هدف اولیهٔ HTML مشخص کردن ساختار سند بود، نه ظاهر آن. برای مثال، فرمان

```
<h1>Deborah's Photos</h1>
```

فقط به مرورگر می‌گوید که این یک تیتر است، اما هیچ چیز دربارهٔ فرمت (نوع فونت، رنگ و یا اندازهٔ) آن نمی‌گوید. این وظیفه بر عهدهٔ مرورگر گذاشته شده است. اما از آنجائیکه بسیاری از طراحان وب مایلند ظاهر صفحه را خود کنترل کنند، برای این کار برچسبهای جدیدی به HTML اضافه شد:

```
<font face="helvetica" size="24" color="red">Deborah's Photos</font>
```

همچنین فرمانهایی برای کنترل دقیق محل اشیاء روی صفحه در نظر گرفته شد. اما مشکل این روش آن است که صفحاتی که با این دستورات نوشته می‌شوند، همه جا یکسان دیده نمی‌شوند (صفحه‌ای که در کامپیوتر طراح آن بسیار قشنگ دیده می‌شود، ممکنست در جای دیگر یک افتضاح واقعی باشد). XML تلاشی بود برای بازگشت به آن ایده‌های اولیه: مشخص کردن ساختار صفحه، نه ظاهر آن. اما هر ایدهٔ خوبی می‌تواند مورد سوءاستفاده قرار گیرد (در این حرف شک نکنید!).

علاوه بر توصیف صفحات وب ، XML کاربردهای دیگری نیز دارد. برای مثال، می‌توان از XML بعنوان رابط بین برنامه‌های کاربردی استفاده کرد: SOAP (پروتکل سادهٔ دسترسی شیء – Simple Object Access Protocol) یکی از راههای انجام RPC (فراخوانی از راه دور – Remote Procedure Call) بین برنامه‌ها (مستقل از زبان و سیستم عامل) است. در این روش، مشتری درخواست خود را بصورت یک پیام XML در آورده و با استفاده از پروتکل HTTP به سرویس‌دهنده می‌فرستد؛ پاسخ سرویس‌دهنده هم بصورت پیام XML به مشتری برگشت داده می‌شود. با این تکنیک برنامه‌هایی که روی سیستم عامل‌های متفاوت هستند، می‌توانند با یکدیگر ارتباط برقرار کنند.

XHTML

HTML برای پاسخ به نیازهای جدید به تحول خود ادامه می‌دهد. از هم اکنون می‌توان حدس زد که در آینده صفحات وب فقط به PC ها محدود نمی‌مانند، و راه خود را به دستگاههای تلفن همراه و PDA ها باز خواهند کرد. این قبیل دستگاهها حافظهٔ کمی دارند، و نمی‌توانند مرورگرهای حجیم و سنگین امروزی (که قسمت اعظم کُد آنها صرف حدس زدن و رفع و رجوع خرابکاری طراح صفحهٔ وب می‌شود) را اجرا کنند. بهمین دلیل، بعد از HTML 4 (بجای HTML 5) زبان جدیدی بنام XHTML (HTML توسعه‌یافته – eXtended HTML) به بازار آمد، که بسیار هم ناخن‌خشک است. این زبان جدید اساساً همان HTML 4 است، که با XML فرمول‌بندی شده است. بعبارت دیگر، در این زبان برچسبی مانند <h1> هیچ معنای خاصی ندارد، مگر اینکه در یک فایل XSL تعریف شده باشد. XHTML استاندارد جدید وب است، و اگر می‌خواهید بالاترین قابلیت جابجایی را در وب داشته باشید، باید از آن استفاده کنید.

بین HTML 4 و XHTML شش تفاوت عمده (و چند تفاوت جزئی) وجود دارد، که در اینجا آنها را فهرست‌وار مرور خواهیم کرد. اول اینکه، صفحات و مرورگرهای XHTML باید استانداردهای آن را دقیقاً رعایت کنند (دیگر جایی برای بنجل‌ها نیست!). XHMTL این ویژگی را از XML ارث برده است.

دوم، تمام برچسب‌ها و صفت‌های XHTML باید با حروف کوچک نوشته شوند. برچسبی مثل <HTML> دیگر در XHTML معتبر نیست، و باید آنرا بصورت <html> نوشت. دستور <img SRC="pic001.jpg"> نیز غلط است، چون XHTML صفتی بنام SRC نمی‌شناسد.

سوم، حذف برچسبهای انتهایی (حتی برچسبی مثل <p/>) قدغن است. در برچسبهایی که ذاتاً انتها ندارند (مانند
 ، <hr> ، و <img>)، نیز باید از /> استفاده کرد:

```
<img src="pic001.jpg" />
```

چهارم، مقدار تمام صفت‌ها باید در داخل گیومه نوشته شود (حتی اگر عدد باشد). مثلاً، دستور

```
<img src="pic001.jpg" height=500 />
```

مجاز نیست، چون 500 در داخل "" قرار ندارد.

پنجم، تودرتو کردن برچسبها باید بدرستی انجام شود. در گذشته این کار لزومی نداشت، چون مرورگر منظور طراح صفحه را حدس می‌زد. برای مثال، HTML 4 دستور زیر را بدرستی اجرا می‌کند:

```
<center><b>Vacation Pictures</center></b>
```

اما این دستور در XHTML مجاز نیست (جای و باید عوض شود).

و بالاخره ششم اینکه، نوع هر سند باید دقیقاً (در ابتدای آن) مشخص شده باشد. (برای دیدن سایر تفاوتهای XHTML و HTML می‌توانید به سایت *www.w3c.org* مراجعه کنید.)

۳-۳-۷ سندهای وب دینامیک

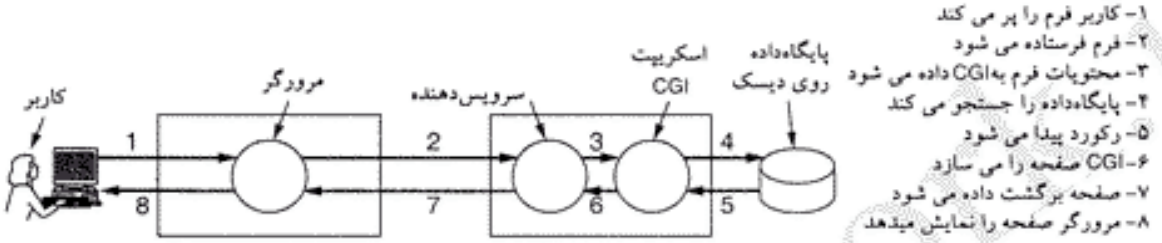
تا اینجا با مدلی که در شکل ۶-۶ دیدید سروکار داشتیم: مشتری فایلی را از سرویس‌دهنده درخواست می‌کند، و سرویس‌دهنده این فایل را برای او می‌فرستد. در روزهای اولیهٔ وب اوضاع همیشه بر همین منوال بود، یعنی صفحه‌ها ثابت بودند و هیچ تغییری نمی‌کردند. اما در سالهای اخیر روز به روز بر تعداد صفحاتی که محتویات آنها بصورت دینامیک تولید می‌شود (و از قبل فایلی روی دیسک سرویس‌دهنده وجود ندارد)، اضافه شده است. تولید محتوا می‌تواند در دو نقطه صورت گیرد: سمت سرویس‌دهنده، و سمت مشتری. اجازه دهید هر یک از آنها را بررسی کنیم.

تولید صفحات وب دینامیک سمت-سرویس‌دهنده

برای اینکه ببینید اساساً چرا تولید محتوا در سمت سرویس‌دهنده احتیاج است، فرمهایی را که در قسمت قبل دربارهٔ آنها صحبت کردیم، در نظر بگیرید. وقتی کاربر فرم را پُر کرده و دکمهٔ *submit* را کلیک می‌کند، پیامی به سرویس‌دهنده فرستاده شده و مقدار فیلدهای فرم را به آن اعلام می‌کند. این پیام هیچ چیز دربارهٔ اینکه سرویس‌دهنده چه فایلی باید برگرداند، نمی‌گوید - و در واقع سرویس‌دهنده قبل از هر کاری باید این اطلاعات را (توسط یک برنامه یا اسکریپت) پردازش کند. معمولاً از اطلاعات فرم برای جستجوی یک پایگاه داده (روی سرویس‌دهنده) و ایجاد یک صفحهٔ HTML خاص استفاده می‌شود. برای مثال، وقتی در یک برنامهٔ تجارت الکترونیک کاربر دکمهٔ *PROCEED TO CHECKOUT* را کلیک می‌کند، مرورگر کوکی محتوی اقلام موجود در سبد خرید کاربر را هم به همراه آن به سرویس‌دهنده می‌فرستد، و این سرویس‌دهنده است که باید با استفاده از اطلاعات این کوکی صفحهٔ HTML لازم را ایجاد کند. بعنوان مثال، این صفحه می‌تواند اقلام انتخاب شده را به کاربر نشان دهد، و ضمن نمایش اطلاعات دیگر (از قبیل آدرس خریدار، و شمارهٔ کارت اعتباری) تأئید نهایی وی را برای شروع عملیات مالی اخذ کند. در شکل ۷-۳۳ مراحل پردازش اطلاعات فرم HTML نشان داده شده است.

روش سنتی پردازش فرمها و دیگر صفحات تعاملی وب سیستمی است بنام CGI (**واسط مشترک دروازه** -

Common Gateway Interface). CGI یک واسط استاندارد شده است که به سرویس‌دهندهٔ وب اجازه می‌دهد تا با برنامه‌ها یا اسکریپت‌های پشت صحنه (که می‌توانند اطلاعات ورودی را از فرمها گرفته، و صفحات HTML تولید کنند) ارتباط برقرار کند. برنامه‌های پشت صحنه معمولاً به زبان اسکریپت‌نویسی پرل (Perl) نوشته می‌شوند، چون نوشتن آنها ساده‌تر و سریعتر است (البته اگر پرل بلد باشید). این قبیل برنامه‌ها را معمولاً در یک دایرکتوری بنام *cgi-bin* ذخیره می‌کنند (که در اغلب URL ها می‌توانید این موضوع را ببینید). زبانهای اسکریپت‌نویسی دیگری هم هست (مانند پایتون – Python)، که می‌توان بجای پرل از آنها استفاده کرد.



شکل ۷-۳۳. مراحل پردازش اطلاعات یک فرم HTML .

برای دیدن طرز کار CGI ، فرض کنید شرکت Truly Great Products Company محصولات خود را بدون کارت ضمانت‌نامه می‌فروشد، ولی از خریداران دعوت می‌کند که برای پُر کردن کارت ضمانت‌نامه به سایت *www.tgpc.com* مراجعه کنند. وقتی خریدار به این صفحه مراجعه می‌کند، لینک زیر را در آن می‌بیند:

Click here to register your product

این لینک به اسکریپت *www.tgpc.com/cgi-bin/reg.perl* اشاره می‌کند. وقتی این اسکریپت بدون هیچ پارامتری اجرا شود، یک صفحهٔ HTML شامل فرم ضمانت‌نامه به کاربر برمی‌گرداند. وقتی کاربر بعد از پُر کردن فرم دکمهٔ *submit* را کلیک کند، پیامی محتوی اطلاعات فرم به اسکریپت *reg.perl* فرستاده می‌شود. اسکریپت *reg.perl* پس از پردازش اطلاعات فرم، یک رکورد برای مشتری جدید در پایگاه دادهٔ مشتریان ایجاد کرده، و یک صفحهٔ HTML حاوی شمارهٔ ضمانت‌نامه محصول و تلفن پشتیبانی پس از فروش را به کاربر برمی‌گرداند. این تنها راه پردازش دینامیک اطلاعات نیست، اما متداولترین روش است. صدها کتاب دربارهٔ اسکریپت‌های CGI و برنامه‌نویسی پرل نوشته شده، که بعنوان نمونه می‌توانیم به (Hanegan, 2001; Lash, 2002; and Meltzer and Michalski, 2001) اشاره کنیم.

اسکریپت‌های CGI تنها روش تولید محتویات دینامیک در سمت سرویس‌دهنده نیست. روش متداول دیگر نوشتن اسکریپت‌های کوچک در داخل صفحات HTML و اجرای این اسکریپت‌ها توسط سرویس‌دهنده است. یکی از زبانهای رایج برای نوشتن این قبیل اسکریپت‌ها PHP (**صفحهٔ خانگی شخصی** – Personal Home Page) نام دارد. البته سرویس‌دهنده باید این زبان را بشناسد تا بتواند اسکریپت‌های آنرا اجرا کند (درست مثل مرورگر که باید XML را بشناسد تا بتواند صفحات XML را نمایش دهد). صفحاتی که حاوی کُد PHP هستند، معمولاً بجای *htm* یا *html* پسوند *php* دارند.

در شکل ۷-۳۴ یک اسکریپت کوچک PHP را ملاحظه می‌کنید؛ این اسکریپت با هر سرویس‌دهنده‌ای که PHP روی آن نصب شده باشد، کار می‌کند. در این صفحه (علاوه بر برچسبهای معمول HTML) یک دستور PHP که در داخل برچسب <? ... php?> قرار دارد، می‌بینید. این صفحه به کاربر اعلام می‌کند که از وی چه می‌داند. این قبیل اطلاعات معمولاً توسط مرورگر (بهمراه کوکی‌ها) به سرویس‌دهنده فرستاده می‌شوند (و

سرویس‌دهنده آنها را از طریق متغیر *HTTP_USER_AGENT* بدست می‌آورد). اگر این صفحه در فایلی بنام *test.php* (در سایت abcd.com) قرار داشته باشد، و کاربر روی لینک *www.abcd.com/test.php* کلیک کند، صفحه‌ای دریافت می‌کند که نوع مرورگر، زبان و سیستم عاملش را به وی اعلام خواهد کرد.

```
<html>
<body>
<h2> This is what I know about you </h2>
<?php echo $HTTP_USER_AGENT ?>
</body>
</html>
```

شکل ۷-۳۴. یک صفحهٔ HTML با دستورات PHP .

PHP از CGI ساده‌تر، و بویژه برای پردازش فرمها مناسبتر است. برای آشنایی بیشتر با طرز کار PHP مثال شکل ۷-۳۵ (الف) را در نظر بگیرید. در این شکل یک فرم HTML می‌بینید؛ تنها تفاوت این فرم با فرمهای قبلی در خط اول آن است: این فرم هنگام کلیک شدن دکمهٔ *submit* فایلی بنام *action.php* را اجرا می‌کند. در این فرم دو جعبهٔ ورودی از نوع *text* وجود دارد، که یکی نام و دیگری سن کاربر را می‌پرسد. بعد از کلیک شدن دکمهٔ *submit* و ارسال محتویات فرم به سرویس‌دهنده (مانند آنچه در شکل ۷-۳۰ دیدید)، سرویس‌دهنده مقدار فیلد اول را در متغیری بنام *name* و مقدار فیلد دوم را در متغیری بنام *age* قرار می‌دهد. پس از آن فایل *action.php* (شکل ۷-۳۵ ب) را پردازش کرده، و دستورات PHP آن را اجرا می‌کند. اگر کاربر در فیلدهای فرم ۷-۳۵ (الف) بترتیب "Barbara" و "24" رابعنوان نام و سن خود وارد کرده باشد، فایلی شبیه شکل ۷-۳۵ (ج) به وی برگردانده خواهد شد. همانطور که می‌بینید، پردازش فرمهای HTML با PHP بسیار ساده شده است.

با اینکه PHP نسبتاً آسان است، اما زبانی بسیار قوی برای ارتباط با پایگاه داده محسوب می‌شود. متغیرها، رشته‌ها، آرایه‌ها، و اغلب ساختارهای کنترلی PHP شباهت زیادی با C دارد (و I/O آن نیز بسیار قوی است). کُد PHP باز است و همه جا می‌توان آنرا مجانی بدست آورد. این زبان بویژه برای کار روی آپاچی (Apache - یکی از پُرطرفدارترین سرویس‌دهنده‌های وب) طراحی شده است. برای اطلاعات بیشتر دربارهٔ PHP به (Valade, 2002) مراجعه کنید.

تا اینجا دو روش مختلف برای ایجاد صفحات HTML دینامیک را بررسی کردیم: CGI و PHP . تکنیک سومی نیز وجود دارد، که JSP (**صفحات سرویس‌دهندهٔ جاوا** - JavaServer Pages) نام دارد؛ این تکنیک مشابه PHP است، با این تفاوت که بجای PHP از زبان برنامه‌نویسی جاوا (Java) استفاده می‌کند (صفحاتی که با این روش نوشته شده‌اند، پسوند *jsp* دارند). تکنیک چهارم یعنی ASP (**صفحات فعال سرویس‌دهنده** - Active Server Pages) معادلیست برای PHP و JSP از شرکت میکروسافت. در این تکنیک از زبان برنامه‌نویسی VBScript (که آن هم از محصولات میکروسافت است) استفاده می‌شود (صفحات ASP پسوند *asp* دارند). انتخاب یکی از تکنیکهای PHP ، JSP یا ASP بیشتر از آن که جنبهٔ فنی داشته باشد، سیاسی است (دعوای همیشگی طرفداران کُد باز، سان و میکروسافت)، چون همهٔ آنها تقریباً شبیه هم هستند.

به مجموعه تکنیکهای ایجاد محتویات دینامیک وب گاهی **HTML دینامیک** نیز گفته می‌شود.

تولید صفحات وب دینامیک سمت-مشتری

تکنیکهای CGI ، PHP ، JSP ، و ASP همگی برای پردازش اطلاعات فرمها (و ارتباط با پایگاه داده) در سمت

```
<html>
<body>
<form action="action.php" method="post">
<p> Please enter your name: <input type="text" name="name"> </p>
<p> Please enter your age: <input type="text" name="age"> </p>
<input type="submit">
</form>
</body>
</html>
```

(الف)

```
<html>
<body>
<h1> Reply: </h1>
Hello <?php echo $name; ?>.
Prediction: next year you will be <?php echo $age + 1; ?>
</body>
</html>
```

(ب)

```
<html>
<body>
<h1> Reply: </h1>
Hello Barbara.
Prediction: next year you will be 25
</body>
</html>
```

(د)

شکل ۷-۳۵. (الف) یک فرم HTML . (ب) اسکریپت PHP برای پردازش این فرم.
(ج) خروجی اسکریپت PHP برای ورودی‌های "Barbara" و "24" .

سرویس‌دهنده هستند. این تکنیک‌ها با پردازش اطلاعات فرم و جستجو در پایگاه داده، صفحات HTML مناسب را تولید و به مشتری برمی‌گردانند. اما هیچکدام از آنها نمی‌توانند حرکات ماوس را تشخیص داده، و یا مستقیماً با کاربر ارتباط برقرار کنند. برای این کار باید از اسکریپتهایی که در دل صفحات HTML نوشته شده و در کامپیوتر مشتری اجرا می‌شوند، استفاده کرد. پای این قبیل اسکریپتها، که با برچسب <script> مشخص می‌شوند، از HTML 4.0 به وب باز شد. متداولترین زبان اسکریپت‌نویسی سمت مشتری **جاوااسکریپت** (JavaScript) است، پس ما هم همین زبان را مختصراً بررسی خواهیم کرد.

جاوااسکریپت یک زبان اسکریپت‌نویسی است، که ارتباط دوری با جاوا دارد، اما مسلماً جاوا نیست. مانند سایر زبانهای اسکریپت‌نویسی، جاوااسکریپت زبانی سطح-بالا است. برای مثال، فقط با یک خط کُد می‌توان یک جعبه دیالوگ نمایش داد، ورودی کاربر را گرفت، و آنرا در یک متغیر ذخیره کرد. این ویژگی جاوااسکریپت را برای ایجاد صفحات وب **تعاملی** (interactive) بسیار مناسب کرده است. از طرف دیگر، این واقعیت که این زبان هنوز

استاندارد نشده (و با سرعتی بیشتر از یک مگس گرفتار آمده در ماشین اشعهٔ X جهش پیدا می‌کند)، نوشتن یک برنامهٔ جاوااسکریپت که بتواند روی هر سیستمی اجرا شود را فوق‌العاده مشکل کرده است.

برنامهٔ جاوااسکریپت شکل ۷-۳۶ را در نظر بگیرید (این برنامه هم مانند شکل ۷-۳۵(الف) نام و سن کاربر را گرفته، و پیش‌بینی می‌کند که وی سال آینده چند سال خواهد داشت!). قسمت <body> تقریباً با برنامهٔ PHP یکسان است؛ تنها جایی که تفاوت کرده، قسمت تعریف دکمهٔ *submit* است. در اینجا، صفت *onclick* به مرورگر می‌گوید که هنگام کلیک شدن دکمه باید اسکریپت *response* را اجرا کرده، و فرم *form* را بعنوان پارامتر به آن بدهد.

اما تعریف تابع جاوااسکریپت *response* در قسمت <head> این صفحه کاملاً جدید است. این تابع ابتدا مقدار فیلد *name* فرم را استخراج کرده و آنرا به همان صورت در متغیری بنام *person* ذخیره می‌کند؛ سپس مقدار فیلد *age* فرم را هم خوانده، و پس از تبدیل آن به عدد (با تابع *eval*) و اضافه کردن 1 به آن، آنرا در متغیر *years* ذخیره می‌کند. پس از آن یک سند (صفحهٔ وب) برای خروجی باز کرده، با دستور *writeln* چهار خط در آن می‌نویسد، و آنرا می‌بندد. بعد از کامل شدن این صفحه، مرورگر آنرا مانند یک فایل HTML معمولی نمایش می‌دهد.

```
<html>
<head>
<script language="javascript" type="text/javascript">
function response(test_form) {
   var person = test_form.name.value;
   var years = eval(test_form.age.value) + 1;
   document.open();
   document.writeln("<html> <body>");
   document.writeln("Hello " + person + ".<br>");
   document.writeln("Prediction: next year you will be " + years + ".");
   document.writeln("</body> </html>");
   document.close();
}
</script>
</head>
<body>
<form>
Please enter your name: <input type="text" name="name">
<p>
Please enter your age: <input type="text" name="age">
<p>
<input type="button" value="submit" onclick="response(this.form)">
</form>
</body>
</html>
```

شکل ۷-۳۶. استفاده از جاوااسکریپت برای پردازش فرم.

درک این نکته بسیار مهم است، که با وجود شباهت برنامه‌های ۷-۳۵ و ۷-۳۶، آنها بطریق کاملاً متفاوتی پردازش می‌شوند. در شکل ۷-۳۵، بعد از آن که کاربر دکمهٔ *submit* را کلیک کرد، مرورگر اطلاعات فرم را در رشته‌ای مانند شکل ۷-۳۰ جمع‌آوری کرده، و به سرویس‌دهنده‌ای که صفحه روی آن قرار دارد، می‌فرستد. سرویس‌دهنده بعد از دیدن نام فایل PHP، این اسکریپت را اجرا می‌کند. اسکریپت PHP مزبور نیز با استفاده از اطلاعات فرم، یک صفحهٔ HTML ایجاد کرده و به مشتری برمی‌گرداند. اما وقتی در شکل ۷-۳۶ دکمهٔ *submit*

کلیک می‌شود، مرورگر اسکریپت درون آنرا اجرا می‌کند - تمام کارها در داخل مرورگر انجام می‌شود، و هیچ تماسی با سرویس‌دهنده وجود ندارد. بهمین دلیل نتیجهٔ کار معمولاً بلافاصله است، برخلاف اسکریپت PHP که رفت و برگشت صفحه ممکنست چندین ثانیه طول بکشد. تفاوت اسکریپت سمت-سرویس‌دهنده و سمت-مشتری در شکل ۷-۳۷ نشان داده شده است. در هر دو شکل، مرحلهٔ 1 گرفتن اطلاعات از کاربر است؛ تفاوت این دو روش در شکل بخوبی مشخص است.

شکل ۷-۳۷. (الف) اسکریپت سمت-سرویس‌دهنده با PHP . (ب) اسکریپت سمت-مشتری با جاوااسکریپت.

اما این تفاوت بدان معنا نیست که جاوااسکریپت بهتر از PHP است - آنها کاربردهای کاملاً متفاوتی دارند. PHP (و زبانهای مشابه آن) اساساً برای پردازش اطلاعات پایگاه داده روی وب مناسب هستند، در حالیکه جاوااسکریپت بدرد نوشتن برنامه‌های تعاملی می‌خورد. نوشتن صفحه‌ای که PHP و جاوااسکریپت را با هم داشته باشد، کاملاً ممکن است چون آنها کارهای متفاوتی انجام می‌دهند.

جاوااسکریپت یک زبان کامل است با تمام توانائیهای C و جاوا، و علاوه بر آن امکانات خاصی نیز برای پردازش صفحات وب (مانند کار با پنجره‌ها و فریمها، ست کردن و خواندن کوکی، پردازش فرم و لینک) دارد. در شکل ۷-۳۸ یک برنامهٔ جاوااسکریپت می‌بینید که در آن از یک تابع بازگشتی (recursive) استفاده شده است.

```
<html>
<head>
<script language="javascript" type="text/javascript">
function response(test_form) {
   function factorial(n) {if (n == 0) return 1; else return n * factorial(n – 1);}
   var r = eval(test_form.number.value);       //  r = typed in argument
   document.myform.mytext.value = "Here are the results.\n";
   for (var i = 1; i <= r; i++)     // print one line from 1 to r
      document.myform.mytext.value += (i + "! =  " + factorial(i) + "\n");
}
</script>
</head>
<body>
<form name="myform">
Please enter a number: <input type="text" name="number">
<input type="button" value="compute table of factorials" onclick="response(this.form)">
<p>
<textarea name="mytext" rows=25 cols=50> </textarea>
</form>
</body>
</html>
```

شکل ۷-۳۸. یک برنامهٔ جاوااسکریپت برای محاسبه و چاپ فاکتوریل.

جاوااسکریپت همچنین می‌تواند حرکت ماوس روی اشیاء مختلف صفحه را دنبال کند. بدین ترتیب می‌توان کاری کرد که وقتی کاربر ماوس را روی قسمتهای خاصی از صفحه می‌برد، اتفاق خاصی بیفتد (مثلاً، باز شدن یک منو، عوض شدن تصویر یا رنگ نوشته‌ها). این قبیل برنامه‌ها (که نمونه‌ای از آنرا در شکل ۷-۳۹می‌بینید) شادابی و سرزندگی خاصی به صفحات وب می‌دهند.

```
<html>
<head>
<script language="javascript" type="text/javascript">
if (!document.myurl) document.myurl = new Array();
document.myurl[0] = "http://www.cs.vu.nl/ ast/im/kitten.jpg";
document.myurl[1] = "http://www.cs.vu.nl/ ast/im/puppy.jpg";
document.myurl[2] = "http://www.cs.vu.nl/ ast/im/bunny.jpg";
function pop(m) {
   var urx = "http://www.cs.vu.nl/ ast/im/cat.jpg";
   popupwin = window.open(document.myurl[m],"mywind","width=250,height=250");
}
</script>
</head>
<body>
<p> <a href="#" onMouseover="pop(0); return false;" > Kitten  </a> </p>
<p> <a href="#" onMouseover="pop(1); return false;" > Puppy  </a> </p>
<p> <a href="#" onMouseover="pop(2); return false;" > Bunny  </a> </p>
</body>
</html>
```

شکل ۷-۳۹. یک صفحهٔ وب تعاملی، که به حرکات ماوس عکس‌العمل نشان می‌دهد.

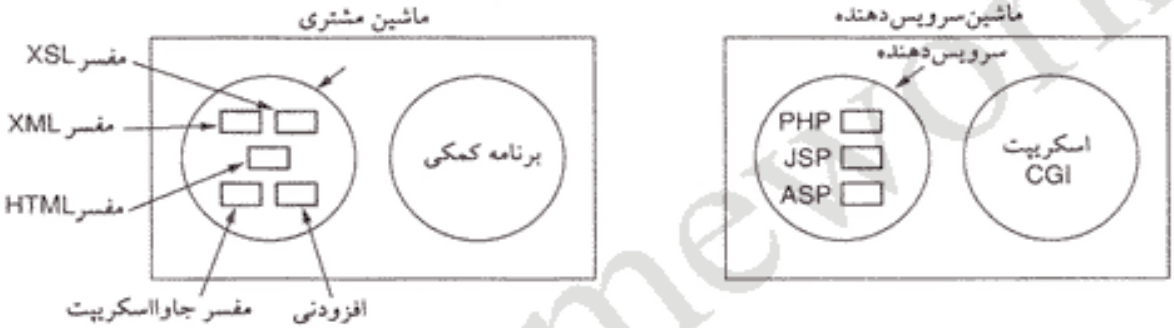
جاوااسکریپت تنها راه نوشتن صفحات وب تعاملی نیست؛ روش دیگر استفاده از اپلت (applet) است. اپلت عبارتست از یک برنامهٔ کوچک جاوا، که برای یک ماشین مجازی بنام JVM (ماشین‌مجازی‌جاوا – Java Virtual Machine) کامپایل می‌شود. این اپلت‌ها را می‌توان (با استفاده از برچسب <applet>) در صفحات وب قرار داد، و مرورگرهایی که JVM را داشته باشند، آنها را اجرا خواهند کرد. از آنجائیکه اپلت‌های جاوا بطور مستقیم اجرا نمی‌شوند، مفسر جاوا می‌تواند جلوی اعمال مخرّب آنها را بگیرد؛ حداقل در تئوری که اینطور گفته می‌شود. اما در عمل برنامه‌نویسان باهوش جاوا نارسایی‌های بیشماری در سیستم I/O جاوا شناسایی کرده‌اند، که می‌توان به کمک آنها در سیستم قربانی نفوذ کرد.

میکروسافت در پاسخ به اپلت‌های جاوا (که از ابداعات شرکت سان میکروسیستمز است) کنترل اکتیوایکس (ActiveX control) را معرفی کرد. کنترلهای اکتیوایکس برنامه‌های کامپایل شدهٔ پنتیوم هستند، که مستقیماً روی سخت‌افزار کامپیوتر مشتری اجرا می‌شوند. سرعت و توانائیهای کنترلهای اکتیوایکس بسیار بیشتر و بهتر از اپلت‌های جاواست، چون آنها برنامه‌های واقعی هستند. وقتی مرورگر اینترنت اکسپلورر در صفحهٔ وب به یک کنترل اکتیوایکس برخورد می‌کند، آنرا بار کرده، و (پس از تعیین هویت) اجرا می‌کند. اما، همانطور که می‌توان حدس زد، بار کردن و اجرای برنامه‌های خارجی مشکلات امنیتی زیادی بهمراه دارد، که در فصل آینده به آنها اشاره خواهیم کرد.

از آنجائیکه برنامه‌های جاوا و جاوااسکریپت روی (تقریباً) تمام مرورگرها اجرا می‌شوند، بهترین گزینه برای نوشتن صفحات تعاملی وب هستند، ولی اگر فقط مرورگرهای میکروسافت را هدف قرار داده‌اید، اکتیوایکس هم

به لیست امکانات شما اضافه خواهد شد. بطور کلی، نوشتن برنامه‌های جاواسکریپت از همه ساده‌تر است، اپلت‌های جاوا سریعتر اجرا می‌شوند، و کنترلهای اکتیوایکس از هر دوی آنها سریعترند. از طرف دیگر، از آنجائیکه اکثر قریب به اتفاق مرورگرها از یک JVM یکسان استفاده می‌کنند (در حالیکه پیاده‌سازی جاواسکریپت در هیچ دو مرورگری یکسان نیست)، قابلیت انتقال اپلت‌های جاوا بیشتر از برنامه‌های جاواسکریپت است. در زمینهٔ برنامه‌نویسی جاواسکریپت کتابهای متعدد و مفصلی وجود دارد، که از میان آنها می‌توان به (Easttom, 2001; Harris, 2001; and McFedries, 2001) اشاره کرد.

قبل از خاتمه دادن به بحث صفحات وب دینامیک، اجازه دهید یک بار دیگر آنچه را تا اینجا دیدیم مرور کنیم. با استفاده از اسکریپت‌ها می‌توان صفحات وب را بصورت کاملاً دینامیک در سرویس‌دهنده ایجاد کرد. این صفحات در کامپیوتر مشتری درست مثل صفحات معمولی HTML نمایش داده می‌شوند، و هیچ تفاوتی با آنها نخواهند داشت. اسکریپت‌ها را می‌توان با پرل، PHP ، JSP ، یا ASP نوشت (شکل ۷-۴۰ را ببینید).



شکل ۷-۴۰. روشهای مختلف ایجاد محتویات دینامیک و نمایش آنها.

تولید محتویات دینامیک در سمت مشتری نیز امکانپذیر است. صفحات وب را می‌توان با XML نوشت، و سپس برای تبدیل آنها به HTML از فایلهای XSL استفاده کرد. با برنامه‌های جاواسکریپت هم می‌توان محاسبات موردنیاز را انجام داد. و بالاخره، با استفاده از برنامه‌های کمکی می‌توان محتویات صفحه را بطرق مختلف فرمت کرد.

۷-۳-۴ پروتکل انتقال اَبَرمتن - HTTP

پروتکل انتقال در سراسر وب HTTP (پروتکل انتقال‌اَبَرمتن - HyperText Transfer Protocol) است. این پروتکل مشخص می‌کند که مشتری چه چیزهایی می‌تواند به سرویس‌دهنده بفرستد، و سرویس‌دهنده چه پاسخهایی می‌تواند به آن بدهد. در خواست‌ها از نوع متنی (ASCII) هستند، و پاسخ سرویس‌دهنده یکی از انواع MIME RFC 822 . تمام مشتری‌ها و سرویس‌دهنده‌ها باید از این پروتکل پیروی کنند. پروتکل HTTP در RFC 2616 تعریف شده است. در این قسمت مهمترین ویژگیهای این پروتکل را مورد بررسی قرار خواهیم داد.

اتصال

مرورگرها معمولاً از طریق اتصال TCP به پورت 80 سرویس‌دهنده با آن ارتباط برقرار می‌کند، اگرچه این الزامی رسمی نیست. خوبی اتصال TCP آنست که مرورگر یا سرویس‌دهنده هیچکدام لازم نیست نگران گم شدن پیامها، پیامهای تکراری یا خیلی بلند، و یا برگرداندن تصدیق دریافت باشند، چون تمام این کارها را TCP انجام می‌دهد.

در HTTP 1.0 ، بعد از برقراری اتصال یک درخواست فرستاده شده و یک پاسخ دریافت می‌شود، و پس از آن اتصال قطع خواهد شد. در آن زمانی که صفحات HTML فقط متن بودند، این روش کاملاً کفایت می‌کرد. اما

خیلی زود صفحات وب پر شد از تصویر، آیکون و چیزهایی مانند آن، که برقرای یک اتصال TCP برای انتقال هر کدام از آنها اصلاً مقرون بصرفه نبود.

برای حل این مشکل HTTP 1.1 عرضه شد، که از اتصال پایدار (persistent connection) پشتیبانی می‌کرد. اتصال پایدار، اتصالیست که می‌توان روی آن چندین بار درخواست و پاسخ ردوبدل کرد. بدین ترتیب، سربارهٔ TCP برای هر صفحهٔ وب بسیار کمتر خواهد شد. در این روش امکان استفاده از تکنیک خط‌لوله (pipeline – ارسال درخواست ۲ قبل از برگشت پاسخ درخواست ۱) هم وجود دارد، که سرعت بار کردن صفحات را بسیار بهتر می‌کند.

متد

با اینکه HTTP برای استفاده در وب طراحی شد، اما طراحان آن (هوشمندانه) نگاهی به آیندهٔ برنامه‌های شیء-گرا نیز داشتند. بهمین دلیل در HTTP عملیاتی غیر از درخواست صفحات وب نیز پیش‌بینی شده است، که به آنها متد (method) می‌گویند. از همین جاست که پروتکل SOAP امکان وجود پیدا کرد. هر درخواست HTTP یک (یا چند) خط متن ASCII است، که با نام متد شروع می‌شود. در شکل ۷-۴۱ متدهایی که HTTP پشتیبانی می‌کند، را می‌بینید. اضافه کردن متدهای جدید به HTTP (برای انجام کارهای جدید) نیز امکانپذیر است. نام متدها در HTTP به نوع حروف حساس است، بنابراین متد *GET* را نمی‌توان بصورت *get* نوشت.

متد	توضیح
GET	تقاضای خواندن صفحه وب
HEAD	تقاضای خواندن سر صفحه وب
PUT	تقاضای ذخیره کردن صفحه وب
POST	اضافه کردن به یک منبع نام‌دار (یعنی صفحه وب)
DELETE	حذف صفحه وب
TRACE	برگرداندن درخواست ورودی
CONNECT	برای آینده رزرو شده است
OPTIONS	جستجوی گزینه‌های خاص

شکل ۷-۴۱. متدهای HTTP .

برای درخواست ارسال یک صفحهٔ وب از سرویس‌دهنده (یا هر شیء دیگر، و بعبارت کلی‌تر، یک فایل) از متد *GET* استفاده می‌کنیم. صفحهٔ ارسال شده بصورت MIME کُد می‌شود. بیشترین درخواستها از سرویس‌دهنده‌های وب همین متد *GET* است، که شکل کلی آن چنین است:

```
GET filename HTTP/1.1
```

که در آن filename نام شیء موردنظر، و 1.1 ویرایش پروتکل مورداستفاده است.

متد *HEAD* فقط سرآیند صفحه (و نه خود آن) را درخواست می‌کند. از این متد می‌توان برای تست به روز بودن صفحات، و یا تست معتبر بودن URL ها استفاده کرد.

متد *PUT* عکس *GET* است: این متد یک صفحه را به سرویس‌دهنده می‌فرستد. محتویات صفحهٔ موردنظر (که می‌تواند بصورت MIME کُد شده باشد) در بدنهٔ متد *PUT* می‌آید؛ در این متد سرآیند احرازهویت (authentication) – برای اثبات اینکه مشتری اجازهٔ نوشتن محتویات در سرویس‌دهنده را دارد – نیز می‌تواند وجود داشته باشد.

متد *POST* تا حدی شبیه *PUT* است، با این تفاوت که محتویات جدید به انتهای داده‌های قبلی اضافه می‌شود. از این متد اغلب برای ارسال پیام به گروههای خبری یا سیستمهای BBS (Bulletin Board System) استفاده می‌شود. در عمل، متدهای *PUT* و *POST* کاربرد بسیار محدودی دارند.

متد *DELETE* همان کاری را می‌کند، که از آن انتظار می‌رود: حذف صفحه (البته برای این کار هم مانند *PUT* داشتن مجوز ضروریست). هیچ تضمینی نیست که متد *DELETE* با موفقیت انجام شود، چون بسیار امکان دارد که وقتی سرویس‌دهندهٔ HTTP می‌خواهد صفحه را حذف کند، فایل در وضعیتی باشد که حذف آن ممکن نباشد.

متد *TRACE* برای دیباگ بکار می‌رود: این متد به سرویس‌دهنده می‌گوید که درخواست را به همان شکلی که دریافت کرده، برگرداند. در مواردی که یک درخواست بدرستی پاسخ داده نمی‌شود، و می‌خواهیم بدانیم علت آن چیست، متد *TRACE* می‌تواند نشان دهد که سرویس‌دهنده درخواست را چگونه دریافت کرده است.

متد *CONNECT* در حال حاضر استفاده نمی‌شود، و برای آینده پیش‌بینی شده است.

متد *OPTIONS* اجازه می‌دهد تا مشتری اطلاعاتی از سرویس‌دهنده (یا یکی از فایلهای آن) بدست آورد.

هر درخواست یک پاسخ شامل یک خط وضعیت و احتمالاً مقداری اطلاعات دیگر (که می‌تواند صفحهٔ وب یا بخشی از آن باشد) دریافت می‌کند. **خط‌وضعیت** (status line) شامل یک کُد سه رقمی است که نشان می‌دهد آیا درخواست انجام شده، و اگر نشده، چرا. رقم اول این کُد برای دسته‌بندی پاسخها بکار می‌رود؛ همانطور که در شکل ۷-۴۲ می‌بینید، پنج نوع پاسخ وجود دارد. کُدهای 1xx در عمل بندرت مورد استفاده قرار می‌گیرند. کُدهای 2xx می‌گویند که درخواست پذیرفته شده، و اطلاعات خواسته شده در حال فرستاده شدن است. کُدهای 3xx مشتری را بجای دیگری (یک URL دیگر، یا حافظهٔ نهان خود مشتری) حواله می‌دهند. کُدهای 4xx حاکی از عدم موفقیت درخواست (به علت خطا در درخواست مشتری، یا عدم وجود صفحهٔ خواسته شده) هستند. کُدهای 5xx هم حاکی از آن هستند که خود سرویس‌دهنده با خطا مواجه شده است (خطا در اجرای کُد خواسته شده، یا ترافیک بیش از حد و عدم توانایی در پاسخ به موقع).

سرآیند پیام

معمولاً بدنبال خط درخواست (خطی که مثلاً متد *GET* در آن آمده) خطهای دیگری (با اطلاعات بیشتر) نیز می‌آیند. به این خطها (که می‌توان آنها را شبیه پارامتر دانست) **سرآیند‌درخواست** (request header) می‌گویند. پاسخها هم می‌توانند **سرآیند‌پاسخ** (response header) داشته باشند. برخی از این سرآیندها (که تعدادی از آنها را در شکل ۷-۴۳ می‌بینید) در هر دو جهت کاربرد دارند.

کد	مفهوم	مثال
1xx	Information	۱۰۰ = سرویس دهنده با درخواست مشتری موافق است
2xx	Success	۲۰۰ = درخواست موفق بوده است ، ۲۰۴ = محتویات موجود نیست
3xx	Redirection	۳۰۱ = صفحه جابجا شده ، ۳۰۴ = صفحه موجود در حافظه نهان همچنان معتبر است
4xx	Client error	۴۰۳ = صفحه ممنوع ، ۴۰۴ = پیدا نشد
5xx	Server error	۵۰۰ = خطای داخلی سرویس دهنده ، ۵۰۳ = دوباره سعی کنید

شکل ۷-۴۲. کُدهای پاسخ سرویس‌دهنده.

با سرآیند *User-Agent* مشتری می‌تواند اطلاعاتی دربارهٔ مرورگر و سیستم عامل خود (و چیزهایی از این قبیل) به سرویس‌دهنده بدهد. در اسکریپت شکل ۷-۳۴ دیدید که سرویس‌دهنده چگونه می‌تواند با استفاده از این سرآیند اطلاعات جالبی از مشتری نمایش دهد.

چهار سرآیند *Accept* به سرویس‌دهنده می‌گویند که مشتری چه چیزهایی را می‌تواند بپذیرد. اولین *Accept* نوع MIME موردپذیرش مشتری (مثلاً، *text/html*) را مشخص می‌کند، و دومی مجموعه کاراکتر آن را (مثلاً، ISO-8859-5 یا Unicode-1-1). سومین *Accept* نوع فشرده‌سازی (مثل، *gzip*)، و چهارمی زبان مشتری (مثلاً، ایتالیایی یا اسپانیایی) را اعلام می‌کنند (تا اگر سرویس‌دهنده امکان انتخاب زبان را فراهم کرده باشد، صفحهٔ مناسب را بفرستد). اگر سرویس‌دهنده نتواند خواسته‌های مشتری را اجابت کند، یک کُد خطا برمی‌گرداند.

سرآیند	نوع	محتویات
User-Agent	درخواست	اطلاعات درباره مرورگر و سیستم عامل آن
Accept	درخواست	نوع صفحاتی که مشتری می تواند بپذیرد
Accept-Charset	درخواست	مجموعه کارکترهای قابل قبول مشتری
Accept-Encoding	درخواست	کد گذاری های صفحه قابل قبول مشتری
Accept-Language	درخواست	زبانهای طبیعی قابل قبول مشتری
Host	درخواست	نام DNS سرویس‌دهنده
Authorization	درخواست	کوکی را به سرویس‌دهنده باز پس می فرستد
Cookie	درخواست	لیست احراز هویت مشتری
Date	هردو	زمان و تاریخ ارسال پیام
Upgrade	هردو	پروتکلی که فرستنده می خواهد به آن ارتقاء دهد
Server	پاسخ	اطلاعاتی درباره سرویس دهنده
Content-Encoding	پاسخ	نحوه کدگذاری محتویات صفحه
Content-Language	پاسخ	زبان طبیعی صفحه
Content-Length	پاسخ	طول صفحه (بر حسب بایت)
Content-Type	پاسخ	نوع MIME صفحه
Last-Modified	پاسخ	زمان و تاریخ آخرین تغییر صفحه
Location	پاسخ	فرمان به مشتری برای ارسال درخواست به جای دیگر
Accept-Ranges	پاسخ	سرویس دهنده محدوده بایت را پذیرفت
Set-Cookie	پاسخ	سرویس دهنده از مشتری می خواهد یک کوکی ذخیره کند

شکل ۷-۴۳. چند سرآیند پیام HTTP .

سرآیند *Host* نام سرویس‌دهنده را مشخص می‌کند، و از URL گرفته می‌شود. این سرآیند اجباریست، چون ممکنست چندین نام DNS دارای یک آدرس IP مشترک باشند، و سرویس‌دهنده باید بداند که درخواست مربوط به *کدام میزبان است*.

سرآیند *Authorization* برای صفحاتی که حفاظت شده هستند، لازم است (در این قبیل موارد، مشتری باید ثابت کند که اجازهٔ دریافت صفحهٔ خواسته شده را دارد).

با اینکه کوکی‌ها در RFC 2109 تعریف شده‌اند، در RFC 2616 نیز دو سرآیند برای کوکی‌ها در نظر گرفته شده است. مشتری برای ارسال کوکی به سرویس‌دهنده‌ای که قبلاً کوکی را از آن گرفته، از سرآیند *Cookie* استفاده می‌کند.

سرآیند *Date* یکی از سرآیندهای دو طرفه است، و برای مشخص کردن زمان ارسال پیام بکار می‌رود. سرآیند *Upgrade* برای تسهیل فرآیند ارتقاء (ویرایشهای ناسازگار) پروتکل HTTP در نظر گرفته شده است. این سرآیند به مشتری (یا سرویس‌دهنده) اجازه می‌دهد تا اعلام کند از چه ویرایشی پشتیبانی می‌کند.

سرآیندهای بعدی همگی خاص سرویس‌دهنده هستند، که در پاسخها از آنها استفاده می‌کند. با اولین آنها، یعنی *Server* ، سرویس‌دهنده خود را معرفی کرده و برخی از مشخصاتش را اعلام می‌کند.

چهار سرآیند بعدی، که همگی با *Content-* شروع می‌شوند، به سرویس‌دهنده اجازه می‌دهند تا صفحه‌ای را

که در حال فرستادن آن است، توصیف کند.

سرآیند *Last-Modified* مشخص می‌کند که تاریخ آخرین تغییر صفحه چه زمانی بوده است. این سرآیند نقش مهمی در عملکرد حافظهٔ نهان صفحات وب در مشتری دارد.

سرویس‌دهنده با استفاده از سرآیند *Location* به مشتری می‌گوید که باید به URL دیگری مراجعه کند (احتمالاً برای اینکه صفحهٔ خواسته شده به جای دیگری منتقل شده، یا هدایت مشتری به سرویس‌دهنده‌های کم ترافیک‌تر). در مواردی هم که یک شرکت دارای شعبه‌های متعددی در سراسر دنیاست، با استفاده از این سرآیند مشتری را به سرویس‌دهنده‌ای که خاص منطقهٔ وی در نظر گرفته شده، هدایت می‌کند. سرویس‌دهنده می‌تواند برای تشخیص مکان جغرافیایی مشتری از آدرس IP یا زبان درخواستی وی استفاده کند.

گاهی که یک صفحه خیلی بزرگ باشد، مشتریهای کوچک مایلند آنرا بصورت قطعه قطعه دریافت کنند. برخی از سرویس‌دهنده‌ها می‌توانند تعداد بایتهای درخواستی یک مشتری را پذیرفته، و فقط همان مقدار را برای وی بفرستند. سرویس‌دهنده با سرآیند *Accept-Range* آمادگی خود را برای ارسال جزئی صفحات اعلام می‌کند.

دومین سرآیند کوکی، یعنی *Set-Cookie* ، وسیله‌ایست که سرویس‌دهنده به کمک آن کوکی‌ها را به مشتری می‌فرستد. مشتری ملزم است این کوکی‌ها را ذخیره کرده، و همراه با درخواستهای بعدی به سرویس‌دهنده پس بفرستد.

نمونه‌ای از کاربرد HTTP

از آنجائیکه HTTP یک پروتکل متنی است، هیچ نیازی نیست مرورگر باشید تا بتوانید با یک سرویس‌دهندهٔ وب تماس بگیرید: فقط کافیست یک اتصال TCP به پورت 80 سرویس‌دهنده داشته باشید. اکیداً به خواننده توصیه می‌کنیم این قسمت را شخصاً امتحان کند (البته UNIX برای این منظور بهتر است، چون برخی از سیستمهای دیگر وضعیت اتصال را برنمی‌گردانند). برای شروع فرمان زیر را وارد کنید:

```
telnet www.ietf.org 80 >log
GET /rfc.html HTTP/1.1
Host: www.ietf.org
close
```

این فرمان یک اتصال TCP به پورت 80 سرویس‌دهندهٔ وب IETF (www.ietf.org) برقرار می‌کند. نتیجهٔ کار به فایل *log* هدایت می‌شود، تا بعداً بتوانیم آنرا بهتر بررسی کنیم. پس از آن فرمان *GET* (بهمراه نام فایل و پروتکل) می‌آید، و خط بعدی سرآیند اجباری *Host* است. خط خالی بعدی نیز اجباریست: این خط خالی به سرویس‌دهنده می‌گوید که ارسال سرآیندها از طرف ما تمام شده است. با فرمان *close* هم به برنامهٔ telnet می‌گوئیم که ارتباط را قطع کند.

فایل *log* یک فایل متنی است، که با هر ادیتوری می‌توان آنرا مشاهده کرد. ابتدای این فایل باید چیزی شبیه شکل ۷-۴۴ باشد (البته اگر IETF تازگیها آنرا عوض نکرده باشد).

سه خط اول خروجی برنامهٔ telnet هستند، نه سرویس‌دهندهٔ www.ietf.org . خط چهارم، که با HTTP/1.1 شروع شده، پاسخ IETF است و می‌گوید که این سرویس‌دهنده مایل است با پروتکل HTTP/1.1 با شما صحبت کند. پس از آن تعدادی سرآیند، و سپس محتویات صفحهٔ www.ietf.org/rfc.html می‌آید. قبلاً همهٔ این سرآیندها را دیده‌اید، بجز دو تا از آنها: سرآیند *ETag* (که یک شمارهٔ منحصربفرد برای شناسایی صفحه، مخصوص حافظهٔ نهان، است)، و *X-Pad* (که جزء سرآیندهای استاندارد نیست، و احتمالاً برای مقابله با مشکلات برخی از مرورگرها بکار می‌آید).

۷-۳-۵ بهبود کارایی

احتمالاً بزرگترین نقطه ضعف وب همان محبوبیت آن است. سرویس‌دهنده‌ها، مسیریابها، و خطوط مخابراتی