طبقة التطبيقات

Application Layer

محتويات الفصل:

- مبادئ تطبيقات الشبكة
- شبكة الويب وبروتوكول **HTTP**
- نقل الملفات باستخدام بروتوكول **FTP**
- البريد الإلكتروني **(E-mail)**
- خدمة دليل الإنترنت لأسماء النطاقات **(DNS)**
- تطبيقات النظائر **(P2P)**
- برمجة مقابس بروتوكول **TCP**
- برمجة مقابس بروتوكول **UDP**
- الخلاصة

تعتبر تطبيقات الشبكة المبرر لوجود شبكة الحاسب الآلي، فإذا لم نستطع تخيل أية تطبيقات مفيدة لتلك الشبكة، فلن تكون هناك حاجة لتصميم أي بروتوكولات لدعمها. خلال الأربعين سنة الماضية تم ابتكار العديد من التطبيقات المبدعة والرائعة للشبكة. يشمل ذلك التطبيقات التقليدية النصية والتي كانت شائعة في السبعينيات والثمانينيات، كالبريد الإلكتروني النصي (text e-mail)، والوصول إلى الحاسبات عن بُعد (remote login)، ونقل الملفات (file transfer)، ومجموعات الأخبار (newsgroups)، والدردشة النصية (text chatting). كما تضمنت ذلك التطبيق الهام – الويب Web – الذى ظهر في منتصف التسعينيات مما يسّر المشاركة في المعلومات والبحث عنها والتجارة الإلكترونية. وتضمنت أيضاً التطبيقين الهامين اللذين ظهرا في نهاية الألفية: الرسائل الفورية (instant messaging) بقوائم المراسلة، ومشاركة النظائر للملفات (P2P file sharing). بالإضافة إلى العديد من تطبيقات الوسائط المتعددة (multimedia) كعرض شرائط الفيديو وراديو الإنترنت وهاتف الإنترنت ومؤتمرات الفيديو وتليفزيون الإنترنت (IPTV). وعلاوة على ذلك فإن انتشار تقنية الوصول ذات الحيز الترددي العريض للأماكن السكنية والاستخدام المتزايد للشبكات اللاسلكية يهيئ المسرح للمزيد من التطبيقات الجديدة والمثيرة في المستقبل.

في هذا الفصل سندرس جوانب تطوير تطبيقات الشبكة من حيث المفهوم والتطبيق. في البداية سنُعرِّف المفاهيم الرئيسة لطبقة التطبيقات (البرامج)، بما في ذلك بروتوكولات طبقة التطبيقات والزبائن (clients) والخادمات (servers) والعمليات (processes) ومقابس الاتصال (sockets) وواجهات التعامل (interfaces) مع طبقة النقل. بعد ذلك سنستعرض عدة أمثلة لتطبيقات الشبكة بالتفصيل كالويب والبريد الإلكتروني وخدمة الدليل لأسماء النطاقات (DNS) ومشاركة النظائر للملفات (P2P) وهاتف الإنترنت عن طريق النظائر. نتناول بعد ذلك تطوير تطبيقات للشبكة على كلٍّ من بروتوكول TCP وبروتوكول UDP. سندرس على التحديد واجهة المقابس لبرمجة التطبيقات (API)، ونتتبع تطبيقات بسيطة للزبون والخادم بلغة جافا (Java). وكمثال سندرس كيف يمكن تطوير خادم ويب

بسيط بتلك اللغة، كما سنقدم أيضاً عدة تدريبات مثيرة ومسلية على برمجة المقابس في نهاية الفصل.

تعتبر طبقة التطبيقات بصفة خاصة مكاناً جيداً لبدء دراسة البروتوكولات، فهي مألوفة لنا من خلال تعاملنا مع العديد من التطبيقات (البرامج) التي تعتمد على البروتوكولات التي سندرسها. هذا سيعطينا فكرةً جيدةً عن ماهية البروتوكولات ومدخلاً للعديد من القضايا التي ستقابلنا مرة أخرى عندما ندرس بروتوكولات طبقة النقل وطبقة الشبكة وطبقة ربط البيانات.

2-1 مبادئ تطبيقات الشبكة

لنفرض أن لديك فكرة لتطبيق جديد؛ قد يكون خدمة عظيمة للإنسانية أو لنيل مدح أستاذك أو لنيل ثروة عظيمة أو ببساطة قد يكون للمرح والتسلية. مهما يكن الحافز دعنا نستعرض الآن كيف تُحوِّل تلك الفكرة إلى تطبيق حقيقي للشبكة.

من صميم تطوير تطبيقات للشبكة كتابة البرامج التي تُنفَّذ على الأنظمة الطرفية (end systems) المختلفة وتتصل فيما بينها عبر الشبكة. على سبيل المثال في تطبيق الويب هناك برنامجان مُميَّزان يتصلان مع بعضهما: برنامج المتصفّح، وبرنامج خادم الويب. يعمل برنامج المتصفّح على مضيف المُستخدِم (مثلاً حاسب مكتبي (desktop) أو حاسب نقّال (laptop) أو مساعد رقمي شخصي (PDA) أو هاتف جوال (mobile phone) أو ما شابه ذلك)، ويعمل برنامج خادم الويب على مضيف خادم الويب. وكمثال آخر في نظام مشاركة النظائر للملفات يوجد برنامج في كل مضيف في المجموعة المشاركة للملفات، وفي هذه الحالة قد تكون البرامج في المضيفات المختلفة متشابهة أو متطابقة.

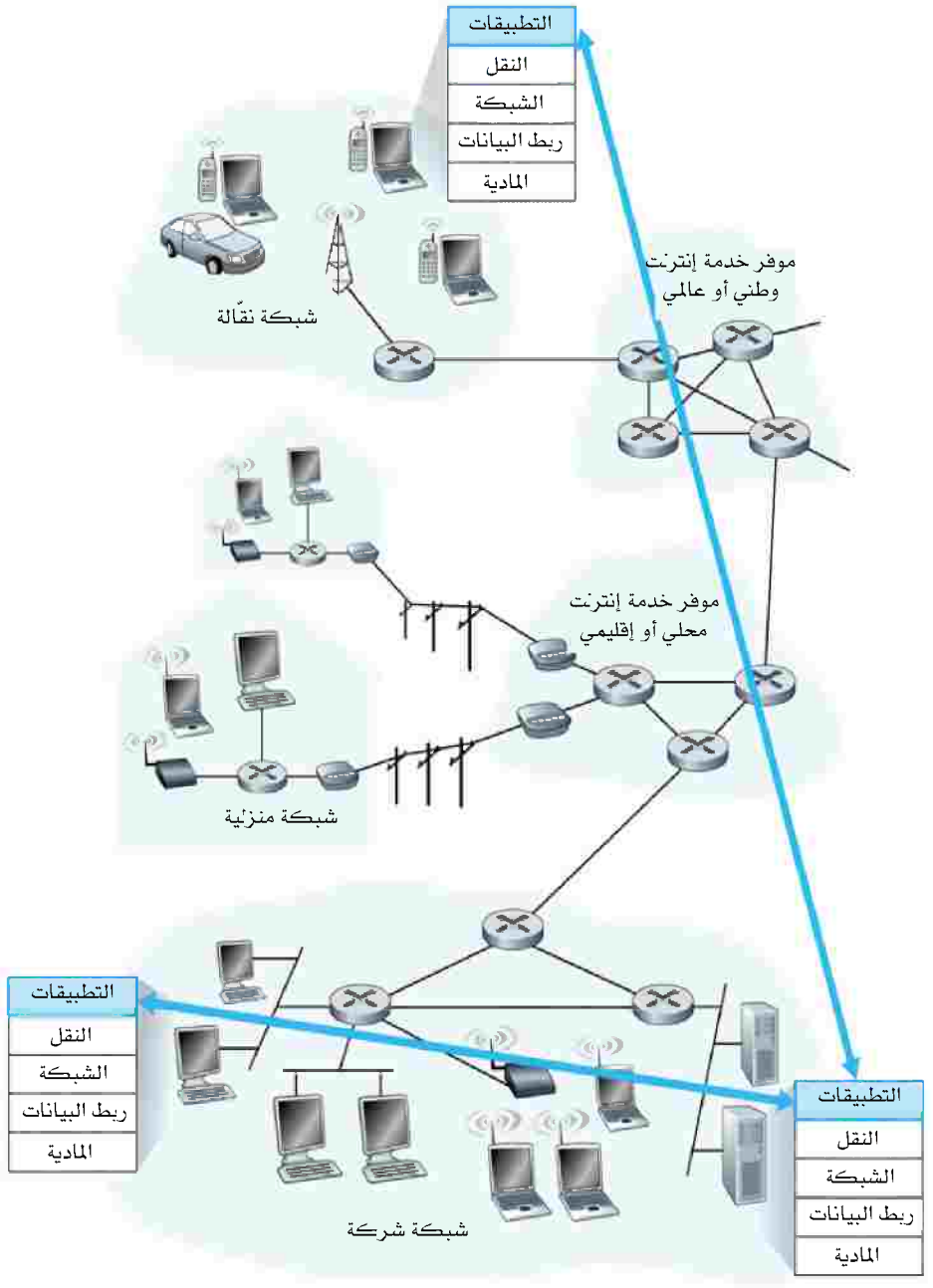
وهكذا فعند تطوير تطبيقك الجديد، ستحتاج إلى كتابة البرامج التي سيتم تنفيذها على أنظمة طرفية متعددة. ويمكنك كتابة هذه البرامج على سبيل

المثال بلغة C أو C++ أو جافا. المهم أنك لست بحاجة إلى أن تكتب برامج للأجهزة التي في قلب الشبكة (network core devices)، كالموجّهات (routers) أو محوّلات طبقة ربط البيانات (switches). وحتى إذا أردت كتابة برامج تطبيقات لتلك الأجهزة، فلن يكون بوسعك عمل ذلك. فكما تعلمنا في الفصل الأول (انظر الشكل 1-20) لا تعمل الأجهزة الموجودة في قلب الشبكة في طبقة التطبيقات، ولكنها تعمل في الطبقات الدنيا وتحديداً في طبقة الشبكة وما تحتها. لقد سهّل هذا التصميم البسيط – أي حصر برامج التطبيقات في الأنظمة الطرفية كما هو موضح في الشكل 1-2 – تطوير وانتشار تشكيلة واسعة من تطبيقات الشبكة بسرعة.

1-1-2 البنية المعمارية لتطبيقات الشبكة

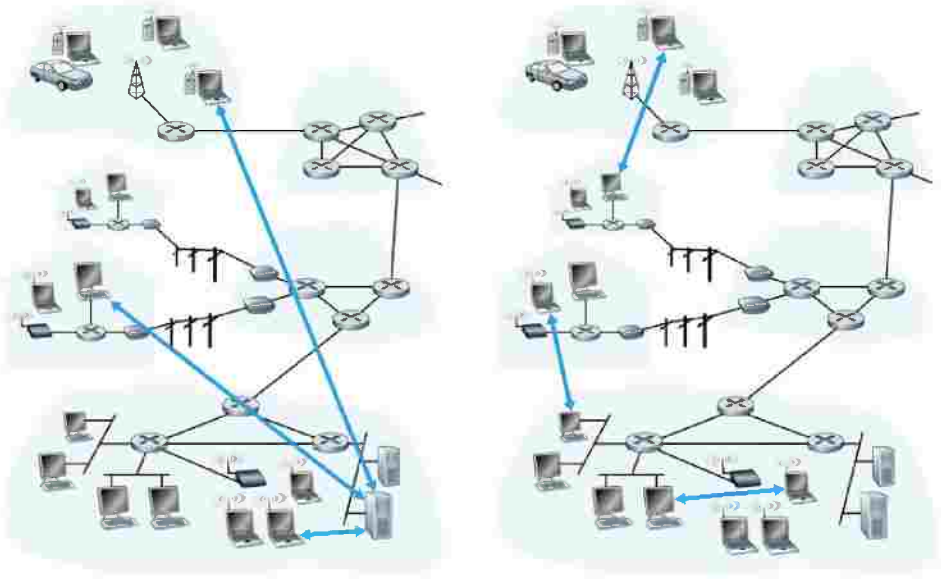
قبل الخوض في كتابة البرامج يجب أن يكون لديك خطة معمارية شاملة لبنية تطبيقك. تذكّر أن بنية التطبيق مختلفة تماماً عن بنية الشبكة (أي نموذج الإنترنت الذى يتكون من خمس طبقات كما تناولناه في الفصل الأول على سبيل المثال)، ومن منظور مطوِّر التطبيقات تعتبر بنية الشبكة ثابتة، وتستخدم لتوفير مجموعة معينة من الخدمات للتطبيقات. ومن ناحية أخرى يتم تصميم بنية التطبيق بواسطة المطوِّر والذي يحدد كيفية هيكلة التطبيق على الأنظمة الطرفية المختلفة. عند تحديد بنية التطبيق المعمارية، سيختار مطوِّر التطبيقات على الأرجح أحد النموذجين السائدين والمُستخدَمين في تطبيقات الشبكة الحديثة: بنية زبون/خادم أو بنية النظائر.

في بنية زبون/خادم يوجد مضيف يعمل على الدوام يسمى "الخادم" ليلبي طلبات العديد من المضيفات الأخرى تسمى "الزبائن". يمكن لمضيفات الزبائن أن تكون شغّالة أحياناً أو دائماً. المثال التقليدى لذلك هو تطبيق الويب الذى يعمل فيه خادم الويب بشكلٍ دائم في خدمة طلبات برامج المتصفّحات التي تعمل على مضيفات الزبائن. وعندما يتلقى خادم الويب طلباً لشيءٍ (كائن object) ما (مثل صفحة ويب) من مضيف الزبون، فإنه يستجيب بإرسال ذلك الشيء إليه إذا كان



الشكل 2-1 يحدث الاتصال بين برامج الشبكة بين النظم الطرفية في طبقة التطبيقات.

لدى الخادم نسخة منه، أو يستجيب بإرسال رسالة خطأ إذا لم يكن موجوداً لديه. لاحظ أنه في بنية زبون/خادم لا يتصل الزبائن مباشرة ببعضهم البعض، فمثلاً في تطبيق الويب لا يتصل متصفّحان مباشرة. الخاصية الأخرى لبنية زبون/خادم هي أن للخادم عنواناً ثابتاً ومعروفاً، يُعرَف بعنوان IP (سنتناوله فيما بعد). وبما أن للخادم عنواناً ثابتاً معروفاً وبما أنه يعمل باستمرار، يمكن أن يتصل الزبون بالخادم دائماً (في أي وقت) بإرسال رزمة بيانات إلى عنوان الخادم. من التطبيقات المعروفة جيداً التي تستخدم بنية زبون/خادم: تطبيقات الويب (Web)، ونقل الملفات (FTP)، والوصول إلى الحاسبات عن بُعد (Telnet)، والبريد الإلكتروني (e-mail). يوضح الشكل 2-2 (a) بنية زبون/خادم.



(a) بنية زبون/خادم

(b) بنية النظائر

الشكل 2-2 بنية التطبيقات: (a) بنية زبون/خادم، (b) بنية النظائر.

في أغلب الأحيان في تطبيقات زبون/خادم قد يعجز مضيف خادم واحد عن تلبية كل طلبات الزبائن. على سبيل المثال يمكن أن يزدحم موقع مشهور للتواصل الاجتماعي بسرعة إذا كان يستخدم خادماً واحداً لمعالجة كل الطلبات. لهذا السبب غالباً ما تُستخدَم مجموعة من الخادمات (cluster of servers)، يطلق عليها مزرعة خادمات (server farm)، للحصول على خادم افتراضي قوي في بنية زبون/خادم. غالباً ما تكون خدمات التطبيقات التي تعتمد أسلوب زبون/خادم مرتكزة بدرجة كبيرة على بنية تحتية (infrastructure)، لأنها تتطلب من موفّري الخدمة شراء وتركيب وصيانة مزارع الخادمات. إضافةً إلى ذلك فإن موفّري الخدمة يجب أن يدفعوا تكلفة متكررة للاتصال وللحيّز الترددي اللازمين لإرسال البيانات إلى الإنترنت وتلقيها منها. يلاحظ أن الخدمات الشائعة مثل محركات البحث (كـ Google مثلاً)، والتجارة من خلال الإنترنت (كـ Amazon وeBay)، والبريد الإلكتروني من خلال الويب (كبريد Yahoo)، والتشبيك الاجتماعي (كـ MySpace وFacebook)، ومشاركة الفيديو (كـ YouTube) ترتكز بدرجة كبيرة على بنية تحتية ويتطلب توفيرها كلفة عالية.

وعلى النقيض من ذلك تعتمد بنية النظائر بدرجة بسيطة على خادمات البنية التحتية التي تعمل دائماً أو قد لا تحتاج إليها على الإطلاق. وكبديل لذلك يستخدم التطبيق الاتصال المباشر بين أزواج المضيفات (والتي يطلق عليها النظائر) بشكلٍ متقطع. وجدير بالذكر أن تلك النظائر ليست ملكاً لموفّر الخدمة، ولكنها أجهزة حاسبات مكتبية وحاسبات نقّالة تحت سيطرة المستخدمين، حيث يوجد معظم النظائر في البيوت والجامعات والمكاتب. ولما كانت النظائر تتصل فيما بينها دون الحاجة للمرور عبر خادم مخصّص، فإنه يُطلق على تلك البنية "نظير إلى نظير" (peer-to-peer). يعتمد العديد من التطبيقات الأكثر شعبية والمزدحمة بحركة البيانات اليوم على بنية النظائر، بما في ذلك توزيع الملفات (كـ BitTorrent)، والمشاركة والبحث عن الملفات (كـ eMule وLimeWire)، وهاتف الإنترنت (كـ Skype)، وتليفزيون الإنترنت (كـ PPLive). يوضح الشكل 2-2 (b) بنية النظائر. يلاحظ أن لبعض التطبيقات بنية معمارية هجينة (hybrid) تجمع

عناصر من كلٍّ من بنية زبون/خادم وبنية النظائر. على سبيل المثال هناك العديد من تطبيقات الرسائل الفورية تستعمل خادمات لتعقب عناوين IP للمُستخدِمين، لكن ترسل الرسائل من مُستخدِم إلى آخر مباشرة بين المضيفات (بدون المرور عبر خادمات وسيطة).

من أهم مزايا بنية النظائر قدرتها الذاتية على التوسع (scalability). على سبيل المثال في تطبيق مشاركة الملفات بتلك البنية، رغم أن كل نظير يولّد حمل شغل (workload) إضافياً بطلب ملفات، فإن كل نظير يضيف مزيداً من قدرة الخدمة أيضاً إلى النظام بقيامه بتوزيع الملفات إلى النظائر الأخرى. تعتبر بنية النظائر طريقة فعالة من حيث التكلفة، فهي لا تتطلب عادةً بنية خادمات تحتية كبيرة أو حيّزاً ترددياً كبيراً للخادم. في سعيهم لخفض كلفة التشغيل، يزداد اهتمام موفّري الخدمة (مثل MSN، وYahoo، وغيرهما) باستخدام بنية النظائر لتطبيقاتهم. ولكن من ناحية أخرى وبسبب الطبيعة الموزّعة والمفتوحة جداً لتطبيقات بنية النظائر، يكون من الصعب توفير نظام فعّال لتأمينها [Doucer 2002; Yu 2006; Liang 2006; Naoumov 2006].

2-1-2 العمليات المُتصِلة فيما بينها (Communicating Processes)

قبل بناء تطبيق للشبكة تحتاج أيضاً لفهم أساسي لكيفية اتصال البرامج – والتي تعمل في أنظمة طرفية متعددة – مع بعضها البعض. في المصطلحات التخصصية لنظم التشغيل ليست البرامج في حقيقة الأمر هي التي تتصل فيما بينها ولكنها العمليات (processes) (ويمكن أن تعتبر العملية كبرنامج يعمل ضمن نظام طرفي). عندما تعمل العمليات على نفس النظام الطرفي، يتم الاتصال فيما بينها عن طريق الاتصال البيني للعمليات (interprocess communication)، باستخدام القواعد التي يحكمها نظام التشغيل على الوحدة الطرفية. لكننا في هذا الكتاب لسنا مهتمين بشكلٍ خاص بكيفية اتصال العمليات فيما بينها على نفس المضيف، وإنما بكيفية اتصال العمليات التي تعمل على مضيفين مختلفين (وقد تعمل تحت نظم تشغيل مختلفة).

تتصل العمليات على نظامين طرفيين مختلفين فيما بينها بتبادل الرسائل عبر شبكة الحاسب – فالعملية المُرسِلة تُنشئ وتُرسِل الرسائل إلى الشبكة، والعملية المُستقبِلة تستلم تلك الرسائل ومن المحتمل أن ترد عليها بإعادة رسائل للمُرسِل. كما هو موضح بالشكل 2-1 تتصل العمليات مع بعضها البعض ضمن طبقة التطبيقات في نموذج الخمس طبقات للبروتوكولات.

عمليتا الزبون والخادم

يتكون تطبيق الشبكة من أزواج من العمليات التي تتبادل الرسائل فيما بينها عبر الشبكة. على سبيل المثال في تطبيق الويب، تتبادل عملية المتصفّح الرسائل مع عملية خادم الويب، وفي نظام مشاركة النظائر للملفات يتم نقل الملف من عملية في نظيرٍ ما إلى عملية في نظيرٍ آخر. لكل زوج من العمليات المتصلة فيما بينها، تسمى إحدى العمليتين عادةً زبوناً والعملية الأخرى خادماً. فمثلاً في تطبيق الويب نعتبر المتصفّح عملية زبون وخادم الويب عملية خادم. وفي مشاركة النظائر للملفات، يُعتبَر النظير الذي يُنزِّل الملف (downloading) بمثابة الزبون، بينما يُعتبَر النظير الذي يُرسِل (يُحمِّل) الملف (uploading) بمثابة الخادم .

ولعلك لاحظت أنه في بعض التطبيقات، كمشاركة النظائر للملفات، يمكن أن تقوم عملية ما بدور الزبون والخادم في نفس الوقت. في الحقيقة يمكن أن تقوم عملية ما في نظام مشاركة الملفات بتحميل وتنزيل الملفات. ومع ذلك ففي سياق أي جلسة اتصال بين زوج من العمليات، يكون بوسعنا دائماً اعتبار إحدى العمليتين "زبون" والعملية الأخرى "خادم". نُعرِّف هنا عمليتي الزبون والخادم كالتالي:

> في جلسة اتصال بين زوج من العمليات، العملية التي تبدأ الاتصال (تتصل أولاً بالعملية الأخرى في بداية الجلسة) تُعَدّ الزبون، بينما العملية التي تنتظر لكي يُتصَل بها لتبدأ الجلسة تُعَدّ الخادم .

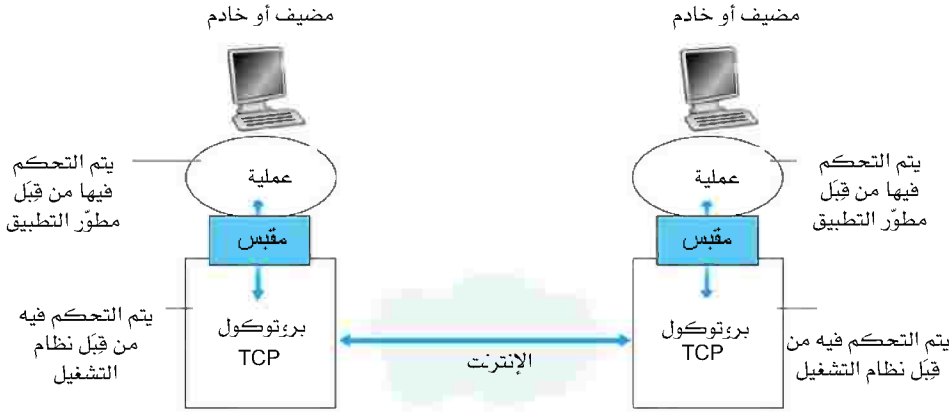
في الويب تبدأ عملية المتصفّح اتصالاً مع عملية خادم الويب؛ ولذا فعملية المتصفّح هي "الزبون" وعملية خادم الويب هي "الخادم". في مشاركة النظائر للملفات، عندما يسأل النظير A النظير B لإرسال ملف معين، فإنه في سياق جلسة الاتصال تلك يكون النظير A هو الزبون بينما النظير B هو الخادم. وعندما لا يكون هناك احتمال للبس أو عدم الوضوح في المعنى، سنستخدم أحياناً المصطلح "جانب الزبون" و"جانب الخادم" من التطبيق. في نهاية هذا الفصل سوف نستعرض خطوات برنامج بسيط لكلٍّ من جانبي الخادم والزبون لتطبيقات الشبكة.

الواجهة بين العملية وشبكة الحاسب

كما لاحظنا من قبل تتكون أكثر التطبيقات من أزواج من العمليات المتصلة فيما بينها، حيث تُرسِل كلٌّ من العمليتين الرسائل إلى العملية الأخرى. يجب أن تمر أي رسالة صادرة من عملية إلى أخرى عبر الشبكة التحتية. ترسل العملية الرسائل إلى الشبكة، وتتلقى الرسائل منها، خلال واجهة برمجة يطلق عليها مقبس (socket). دعنا نستعرض مثالاً يساعدنا على فهم العمليات والمقابس. تشبه العملية البيت، ومقبسها يمثل باب البيت. عندما تريد عملية إرسال رسالة إلى عملية أخرى تعمل على المضيف الآخر، تدفع العملية بالرسالة خارج بابها (مقبسها)، حيث تفترض العملية المُرسِلة هذه وجود بنية نقل تحتية على الجانب الآخر من بابها تقوم بنقل الرسالة إلى باب العملية المُرسَل إليها على مضيف الوجهة النهائية. وعندما تصل الرسالة إلى ذلك المضيف، فإنها تمر من خلال باب (مقبس) العملية المُستقبِلة، والتي تتصرف عندئذ بناءً على محتوى الرسالة.

يوضح الشكل 2-3 الاتصال بين مقبسي عمليتين تعملان على الإنترنت. يفترض الشكل أن نظام النقل التحتي المُستخدَم من قِبَل العمليتين هو بروتوكول التحكم في الإرسال (TCP)، وكما يوضح هذا الشكل، فإن مقبس الاتصال هو الواجهة بين طبقة التطبيقات وطبقة النقل على المضيف. كما يسمى المقبس أيضاً واجهة برمجة التطبيقات (API) بين التطبيق والشبكة، حيث إن المقبس هو واجهة البرمجة التي تُبنَى من خلالها تطبيقات الشبكة. وفي حين يمتلك مطوِّر التطبيقات

سيطرة كاملة على كل شيء على جانب طبقة التطبيق من المقبس، فإنه ليس لديه سوى القليل مما يمكنه عمله على جانب طبقة النقل من المقبس. تنحصر مجالات السيطرة الوحيدة التي يمتلكها مطوِّر التطبيقات على جانب طبقة النقل في: (1) اختيار نوع بروتوكول النقل، وربما (2) القدرة على تحديد قيمة بضعة متغيرات لطبقة النقل، كالسعة القصوى لذاكرة التخزين المؤقت (buffer) والحجم الأقصى لقطعة TCP (TCP segment) (سنتناولها في الفصل الثالث). وبمجرد اختيار مطوِّر التطبيقات بروتوكول النقل (في حالة توفر ذلك الخيار)، يُبنَى التطبيق باستعمال خدمات طبقة النقل المتوفرة مع ذلك البروتوكول. سنتناول المقابس بشيءٍ من التفصيل في الجزأين 2-7 و2-8.



الشكل 2-3 عمليتا التطبيق والمقابس وبروتوكول النقل بينهما.

2-1-3 خدمات النقل المتاحة للتطبيقات

تذكّر أن المقبس (socket) هو الواجهة بين التطبيق وبروتوكول طبقة النقل. على جانب الإرسال يدفع التطبيق بالرسائل خلال المقبس، وعلى الجهة الأخرى للمقبس يضطلع بروتوكول طبقة النقل بمسؤولية توصيل الرسائل إلى باب المقبس في جانب الاستقبال.

توفر العديد من الشبكات (بما فيها الإنترنت) أكثر من بروتوكول لطبقة النقل. وعندما تطور تطبيقاً ما يجب أن تختار أحد بروتوكولات طبقة النقل المتوفرة. كيف تقوم بهذا الاختيار؟ على الأغلب سوف تدرس الخدمات المقدمة من البروتوكولات المتوفرة في طبقة النقل، ثم تختار البروتوكول الذي يقدم خدمات أكثر ملاءمة لاحتياجات تطبيقك. هذا يشبه تماماً اختيارك وسيلة مواصلات (الطائرة أو القطار) للسفر بين مدينتين، حيث يجب أن تختار وسيلة دون الأخرى. كل وسيلة توفر خدمات مختلفة (على سبيل المثال محطات القطار كثيرة ويمكن أن تركب وتنزل من معظم أحياء المدينة، بينما تقطع الطائرة الرحلة في زمن أقصر).

ما هي الخدمات التي يمكن أن يقدمها بروتوكول طبقة النقل للتطبيقات التي تستخدمه؟ يمكن بشكلٍ عام تصنيف متطلبات الخدمة لتطبيق ما على أربعة محاور: النقل الموثوق للبيانات (reliable data transfer)، والطاقة الإنتاجية (throughput)، والتوقيت (timing)، والأمن (security).

النقل الموثوق للبيانات (Reliable Data Transfer)

كما تناولنا في الفصل الأول، من الممكن أن تفقد رزم البيانات في شبكة الحاسب. على سبيل المثال يمكن أن تُفقد البيانات نتيجة امتلاء ذاكرة المخزن المؤقت في موجّه، أو أن تُهمل عند الموجّه أو المضيف إثر اكتشاف خطأ في بعض بتاتها. وفي العديد من التطبيقات كالبريد الإلكتروني ونقل الملفات والوصول إلى المضيفات عن بُعد ونقل وثائق الويب والتطبيقات المالية، يمكن أن يؤدي فقد البيانات إلى نتائج وخيمة العاقبة (في الحالة الأخيرة إما للمصرف أو للعميل!). وهكذا لدعم تلك التطبيقات ينبغي اتخاذ الإجراءات اللازمة لضمان استقبال البيانات المُرسَلة من أحد طرفي التطبيق بشكلٍ صحيح وكامل على الطرف الآخر من التطبيق. إذا وفّر بروتوكول ما مثل تلك الخدمة للتوصيل المضمون للبيانات يُقال: إنه يوفّر نقلاً موثوقاً للبيانات. إحدى الخدمات المهمة التي يمكن أن يوفرها

بروتوكول طبقة النقل لتطبيق ما هي النقل الموثوق للبيانات بين عملية وأخرى. وعندما يوفر بروتوكول نقل هذه الخدمة، لن يكون على العملية المُرسِلة سوى تمرير بياناتها إلى المقبس مع ثقة تامة في أن تلك البيانات ستصل كاملةً وبدون أخطاء إلى العملية المُستقبِلة.

عندما لا يوفر بروتوكول طبقة النقل نقلاً موثوقاً للبيانات من الممكن ألا تصل البيانات المُرسَلة إلى العملية المُستقبِلة على الإطلاق. قد يكون هذا الأمر مقبولاً لدى التطبيقات التي تتسامح بعض الشيء مع فقد البيانات، ومنها بشكلٍ خاص تطبيقات الوسائط المتعددة كالصوت والفيديو الفوري أو تسجيلات الصوت/الفيديو المُخزّنة والتي يمكن أن تسمح ببعض الفقد في البيانات. في تطبيقات الوسائط المتعددة تلك قد يؤدي فقد البيانات إلى خلل طفيف يمكن تقبله أثناء تشغيل تسجيلات الصوت/الفيديو – وهو أمرٌ لا يمثل عيباً كبيراً يضر بجودة البيانات عند استعادتها على الطرف الآخر.

الطاقة الإنتاجية (Throughput)

في الفصل الأول قدّمنا مفهوم الطاقة الإنتاجية المتاحة والتي – في سياق جلسة اتصال بين عمليتين على طول مسار في الشبكة – تمثل معدل توصيل البيانات إلى عملية المُستقبِل. ولما كانت الجلسات الأخرى تشارك في الحيّز الترددي (bandwidth) على طول المسار في الشبكة، ونظراً لأن تلك الجلسات تجيء وتذهب، يمكن أن تتفاوت الطاقة الإنتاجية المتوفرة من وقت لآخر. تقود هذه الملاحظات إلى خدمة طبيعية أخرى يمكن أن يوفرها بروتوكول طبقة النقل، ألا وهي ضمان توفير طاقة إنتاجية بمعدل محدد. بموجب هذه الخدمة يمكن أن يطلب التطبيق ضمان طاقة إنتاجية تعادل r بت/ثانية، وعندئذ يضمن بروتوكول طبقة النقل توفير طاقة إنتاجية لا تقل أبداً عن r بت/ثانية. إن مثل تلك الخدمة – أي توفير طاقة إنتاجية مضمونة – تروق للعديد من التطبيقات. على سبيل المثال إذا كوّد تطبيق هاتف الإنترنت الصوت بمعدل 32 كيلوبت/ثانية،

فإنه يحتاج لإرسال البيانات إلى الشبكة وتسليم البيانات إلى التطبيق المُستقبل بنفس هذا المعدل. وإذا لم يستطع بروتوكول النقل توفير تلك الطاقة الإنتاجية فسيضطر التطبيق إما إلى تكويد البيانات بمعدل أقل (والحصول على طاقة إنتاجية كافية للمحافظة على معدل التكويد المنخفض هذا) أو التوقف تماماً، حيث إن تأمين نصف الطاقة الإنتاجية المطلوبة هو أمر قليل أو عديم الفائدة لتطبيق هاتف الإنترنت هذا. يُطلق على التطبيقات التي لها متطلبات تتعلق بالطاقة الإنتاجية تطبيقات حسّاسة للحيّز الترددي (bandwidth sensitive). يلاحظ أن العديد من التطبيقات الحالية للوسائط المتعددة هي من هذا النوع، رغم أن بعض تطبيقات الوسائط المتعددة قد تستعمل أساليب التكويد التكيفي، وذلك لتكويد البيانات بمعدل يجاري الطاقة الإنتاجية المتوفرة حالياً على الشبكة.

بينما تتميز التطبيقات الحسّاسة للحيّز الترددي بأن لها متطلبات محددة من الطاقة الإنتاجية، فإن التطبيقات المرنة (elastic applications) – على النقيض من ذلك – يمكن أن تستفيد من أي قدر من الطاقة الإنتاجية المتاحة قلّ ذلك أو كثر. وكمثال لذلك، فإن البريد الإلكتروني، ونقل الملفات، ونقل وثائق الويب، كلها تطبيقات من هذاالنوع المرن. بالطبع كلما زادت الطاقة الإنتاجية المتاحة كلما كان الوضع أفضل، وكما يقول المثل: لا يمكن أن تكون غنياً أكثر من اللازم، ولا نحيفاً أكثر من اللازم، ولا ذا طاقة إنتاجية أكثر من اللازم (أي لا يوجد حد أقصى للاكتفاء من هذه الأشياء)!

التوقيت (Timing)

يمكن أيضاً أن يوفر بروتوكول طبقة النقل ضمانات تتعلق بالتوقيت. وكما هو الحال مع ضمانات الطاقة الإنتاجية، فإن ضمانات التوقيت يمكن أن تأخذ العديد من الأشكال والصور. فمثلاً قد يشمل ذلك ضمان أن كل بت يرسلها المُرسِل إلى مقبس الإرسال تصل إلى مقبس المُستقبل خلال زمن لا يتعدى 100 ميللي ثانية لاحقاً. تروق مثل تلك الخدمة للتطبيقات الفورية التفاعلية كهاتف الإنترنت والبيئات الافتراضية والمؤتمرات عن بُعد والألعاب متعددة اللاعبين،

وجميعها تضع قيوداً صارمة على توقيت تسليم رزم البيانات للحصول على تطبيق فعّال (راجع الفصل السابع، و[Gauthier 1999] و[Ramiee 1994]). على سبيل المثال تؤدي التأخيرات الطويلة في نقل بيانات هاتف الإنترنت إلى توليد وقفات غير طبيعية في المحادثة، كما أن التأخير الطويل في لعبة متعددة اللاعبين أو بيئة تفاعلية افتراضية بين فعل شيء ورؤية الاستجابة له من البيئة (على سبيل المثال من لاعب آخر في نهاية توصيلة من طرف إلى طرف) يجعل التطبيق يبدو أقل واقعية. في حالة التطبيقات غير الفورية، يُفضَّل التأخير الأقل دائماً على التأخير الأكبر، ولكن بدون فرض قيود صارمة على التأخيرات من طرف إلى طرف.

الأمن (Security)

أخيراً يمكن أن يزوِّد بروتوكول النقل تطبيقاً ما بواحدٍ أو أكثر من خدمات الأمن. على سبيل المثال في مضيف الإرسال يمكن أن يُشفّر بروتوكول النقل كل البيانات الصادرة من العملية المُرسِلة. وفي المقابل وعلى مضيف الاستقبال يقوم بروتوكول النقل باسترجاع البيانات الأصلية قبل تسليمها إلى العملية المستقبلة. توفر مثل هذه الخدمة سريّة للتعامل بين العمليتين، حتى لو كانت البيانات تجرى مراقبتها بطريقة ما بين العمليتين على المُرسِل والمُستقبل. كما يمكن أن يوفر بروتوكول النقل خدمات أمن أخرى بالإضافة إلى السريّة كسلامة البيانات والتوثيق الطرفي، وسوف نتناول هذه الموضوعات بالتفصيل في الفصل الثامن.

4-1-2 خدمات النقل المتوفرة على الإنترنت

استعرضنا حتى الآن خدمات النقل التي يمكن أن توفرها شبكة الحاسب بصفة عامة. دعنا الآن نكون أكثر تحديداً بفحص أنواع الدعم المتاحة من شبكة الإنترنت للتطبيقات. توفر الإنترنت (وبعموم أكثر شبكات TCP/IP) بروتوكولي نقل لاستخدام التطبيقات، وهما: بروتوكول وحدة بيانات المُستخدِم (UDP) وبروتوكول التحكم في الإرسال (TCP). عندما تقوم (كمطوّر برامج)

بإنشاء تطبيق شبكة جديد للإنترنت، سيكون من أول القرارات التي يتعين عليك اتخاذها ما إذا كنت ستستخدم بروتوكول UDP أو TCP. يوفر كلٌّ من هذين البروتوكولين نموذج خدمة مختلف للتطبيقات التي تستخدمه. يوضح الشكل 2-4 متطلبات الخدمة لبعض التطبيقات المختارة.

التطبيق	فقد البيانات	الحيز الترددي	الحساسية للوقت
نقل الملفات	غير مسموح به	مرن	غير حساس
البريد الإلكتروني	غير مسموح به	مرن	غير حساس
الويب	غير مسموح به	مرن (بضعة كيلوبت/ثانية)	غير حساس
هاتف الإنترنت ومؤتمرات الفيديو	متساهل	الصوت: بضعة كيلوبت/ثانية – 1 ميجابت/ثانية الفيديو: 10 كيلوبت/ثانية – 5 ميجابت/ثانية	حساس (بضع مئات من الميللي ثانية)
ملفات الصوت والفيديو المخزنة	متساهل	الصوت: بضعة كيلوبت/ثانية – 1 ميجابت/ثانية الفيديو: 10 كيلوبت/ثانية – 5 ميجابت/ثانية	حساس (بضع ثوانٍ)
الألعاب التفاعلية	متساهل	بضعة كيلوبت/ثانية – 10 كيلوبت/ثانية	حساس (بضع مئات من الميللي ثانية)
المراسلة الفورية	غير مسموح به	مرن	تختلف فقد يكون بعضها حساس والبعض الآخر غير حساس

[1]الشكل 2-4 متطلبات بعض تطبيقات الشبكة.

[1] يشار إلى بعض "الجداول" في الكتاب الأصلي بالأشكال، لذا تركنا الإشارة إليها "بالأشكال" من أجل عدم إحداث تغيير بتسلسل ترقيم الأشكال والجداول مما يسهل الرجوع للكتاب الأصلي (لمن أراد ذلك).

خدمات بروتوكول TCP

يتضمن نموذج خدمة بروتوكول TCP خدمة نقل توصيلية وخدمة نقل موثوق للبيانات. عندما يستدعي تطبيق ما بروتوكول النقل TCP فإنه يستفيد من كلٍّ من هاتين الخدمتين.

- خدمة النقل التوصيلية: في بروتوكول TCP يتبادل كلٌّ من الزبون والخادم معلومات التحكم قبل أن تبدأ رسائل التطبيق بالتدفق. من شأن هذا الإجراء الذي يعرف بالمصافحة (handshaking) بين الطرفين تنبيه كلٍّ من الزبون والخادم للاستعداد لتدفق رزم البيانات. عقب مرحلة المصافحة يقال إن توصيلة TCP قائمة بين مقابس العمليتين (لدى الزبون والخادم). وهذه التوصيلة من النوع المزدوج الاتجاه (full-duplex) مما يُمكّن كلا العمليتين من إرسال الرسائل إلى بعضهما البعض في نفس الوقت وعلى نفس التوصيلة. عندما ينتهي التطبيق من إرسال الرسائل يجب عليه إغلاق التوصيلة. ويلاحظ أنه بالرغم من أننا أطلقنا على تلك الخدمة "خدمة توصيلية" إلا أننا نقصد في الواقع "خدمة مبنية على التوصيلة" حيث إن العمليتين موصلتان ببعضهما بطريقة فضفاضة للغاية. سنتناول في الفصل الثالث بالتفصيل الخدمة المبنية على التوصيل وكيفية تحقيقها.
- خدمة النقل الموثوق للبيانات: يمكن أن تعتمد العمليات المتصلة فيما بينها على بروتوكول التحكم في الإرسال (TCP) لتسليم كل البيانات التي أرسلت بدون خطأ أو تكرار وبالترتيب الصحيح. وعندما يرسل أحد طرفي التطبيق سلسلة من البايتات إلى مقبس اتصال، يمكن أن يعتمد على بروتوكول التحكم في الإرسال لتسليم نفس سلسلة البايتات إلى مقبس المُستقبل بدون فقد أو تكرار فيها.

يتضمن بروتوكول TCP أيضاً آلية للتحكم في الازدحام (congestion) في الشبكة، والتي تأخذ بعين الاعتبار الصالح العام لمُستخدِمي الإنترنت ككل وليس فقط المنفعة المباشرة والقريبة للعمليتين المعنيتين فقط. تقوم آلية التحكم في

الازدحام بالحد من معدل إرسال البيانات من عملية الإرسال (على زبون أو خادم) عندما تكون الشبكة مزدحمة بين المُرسِل والمُستقبل. وكما سنرى في الفصل الثالث، تحاول آلية التحكم في الازدحام في بروتوكول TCP تحقيق التوزيع العادل للحيّز الترددي المتاح بين توصيلات TCP. غير أن خَنْق معدل الإرسال يمكن أن يكون له تأثير ضار جداً على التطبيقات الفورية (تطبيقات الوقت الحقيقي (real-time applications)) كإرسال الصوت والفيديو، والتي تتطلب حداً أدنى من الحيّز الترددي. ولكن من ناحية أخرى، فإن التطبيقات الفورية متسامحة في فقد الرزم، ومن ثم فهي ليست بحاجة إلى خدمة نقل موثوقة تماماً. ولهذه الأسباب مجتمعة، فإن مطوّرى التطبيقات الفورية عادة ما يستخدمون بروتوكول UDP بدلاً من بروتوكول TCP.

خدمات بروتوكول UDP

يعتبر بروتوكول UDP بروتوكول نقل بسيط بأقل نموذج خدمة، فهو بروتوكول لاتوصيلي (connectionless)، حيث لا يتضمن إجراء مصافحة (handshaking) قبل بدأ الاتصال بين عمليتي التطبيق. يوفر هذا البروتوكول خدمة غير موثوقة لنقل البيانات. فعندما تُرسِل عملية ما رسالة إلى مقبس UDP، فإن البروتوكول لا يعطي أي ضمان لأن تصل الرسالة إلى عملية المُستقبل. علاوة على ذلك قد تصل الرسائل إلى عملية المُستقبل بترتيب مختلف.

لا يتضمن بروتوكول UDP أية آلية للتحكم في الازدحام في الشبكة، ولذا يمكن لعملية الإرسال بث البيانات إلى مقبس UDP بأي معدل تختاره (ولكن يجب ملاحظة أن الطاقة الإنتاجية المتحققة فعلياً من طرف إلى طرف قد تكون أقل من هذا المعدل بسبب الحيّز الترددي المحدود لوصلات الشبكة المُستخدَمة بين المُرسِل والمُستقبل أو بسبب الازدحام). نظراً لأن التطبيقات الفورية غالباً ما تسمح ببعض الفقد في البيانات ولكنها تتطلب ضمان حد أدنى لمعدل الإرسال لتكون فعالة، يختار مطوّرو التطبيقات الفورية أحياناً تشغيل تطبيقاتهم على UDP، ومن ثم تفادي الأعباء الإضافية في TCP للتحكم في الازدحام وفي تركيبة الرزم. من

ناحية أخرى ونظراً لأن العديد من برامج الحماية (الجدران النارية أو الحواجز المنيعة) (firewalls) تُضبط لحجب أكثر أنواع حركة مرور بيانات UDP، لجأ المصممون على نحو متزايد مؤخراً لتشغيل تطبيقات الوسائط المتعددة والفورية على TCP [Sripanidkulchai 2004].

نبذة عن الأمن (Focus on Security)

تأمين TCP

لا يوفر أيٌ من البروتوكولين TCP أو UDP تشفيراً للبيانات (encryption)، أي أن البيانات التى ترسلها عملية الإرسال خلال المقبس هى نفسها البيانات التي تنقل خلال الشبكة لعملية الوجهة. لذلك فإنه على سبيل المثال إذا أرسلت عملية الإرسال كلمة سر غير مشفرة خلال المقبس فإنها ستُرسَل بنفس الشكل (أي غير مشفرة) خلال كل الوصلات في الشبكة بين المُرسِل والمُستقبل. وبالتالي من المحتمل أن يتم التقاطها واكتشافها على أيٍّ من تلك الوصلات البينية. ولأن الأمن والخصوصية أصبحت ذات أهمية عالية للعديد من التطبيقات طوّر مجتمع الإنترنت تحسيناً لبروتوكول TCP يسمى طبقة المقابس الآمنة (SSL). يقدم بروتوكول TCP المحسّن بطبقة المقابس الآمنة (SSL) كل شيء يقدمه بروتوكول TCP التقليدي. بالإضافة إلى توفير خدمات الأمن من عملية إلى عملية كتشفير البيانات (data encryption)، وسلامة البيانات (data integrity)، والتحقق من هوية الأطراف المتصلة (authentication). ونؤكد القول بأن SSL ليس بروتوكول نقل مثل TCP أو UDP وإنما هوتحسين لبروتوكول TCP، وهذا التحسين مطبق في طبقة التطبيقات. وبالتحديد إذا أراد تطبيق أن يستخدم خدمات SSL فإن عليه أن يستخدم تعليمات SSL (توجد مكتبات (libraries) وفصائل (classes) على درجة عالية من الأداء) في جانبي الزبون والخادم للتطبيق. لدى SSL واجهة برمجة API خاصة به مشابهة لواجهة برمجة المقابس التقليدية الخاصة ببروتوكول TCP. عندما يستخدم تطبيقٌ ما SSL فإن عملية الإرسال ترسل البيانات غير مشفرة إلى مقبس SSL والذي يقوم بدوره بتشفير البيانات وإرسالها إلى مقبس TCP. تمر البيانات المشفرة خلال شبكة الإنترنت حتى تصل إلى مقبس TCP لدى عملية الاستقبال فيرسلها مقبس TCP إلى SSL والذي يقوم باسترجاع البيانات الأصلية وتمريرها خلال مقبس SSL إلى عملية الاستقبال. سوف نغطي بعض تفاصيل SSL في الفصل الثامن.

الخدمات التي لا توفرها بروتوكولات النقل على الإنترنت

لقد رتبنا خدمات بروتوكول النقل الممكنة على أربعة محاور: النقل الموثوق للبيانات، والطاقة الإنتاجية، والتوقيت، والأمن. أي من تلك الخدمات يوفرها بروتوكول TCP وأي منها يوفرها بروتوكول UDP؟ لاحظنا أن بروتوكول TCP يتيح نقلاً موثوقاً للبيانات من طرف إلى طرف. كما أنه يمكن بسهولة استخدام SSL لتحسين أداء بروتوكول TCP من منظور طبقة البرامج وذلك بتوفير خدمات لأمن البيانات. غير أنه في وصفنا المقتضب لبروتوكول TCP وبروتوكول UDP غاب بشكلٍ واضح أي ذِكر لخدمات ضمان الطاقة الإنتاجية أو التوقيت، حيث أن تلك الخدمات غير متوفرة في أي من بروتوكولات النقل الخاصة بالإنترنت اليوم. هل يعني ذلك أنه ليس بإمكان التطبيقات الحسّاسة للتوقيت كهاتف الإنترنت أن تعمل على إنترنت اليوم؟ واضح أن الجواب لا، فالإنترنت تستضيف تطبيقات حسّاسة للتوقيت لسنوات عديدة خلت. تعمل تلك التطبيقات بشكلٍ لا بأس به في أغلب الأحيان لأنها صُمِّمت لتحمُّل النقص في الضمان الذي تتطلبه إلى أقصى حد ممكن. سوف نناقش العديد من حيل التصميم المُستخدَمة في هذا المجال في الفصل السابع. ومع ذلك فالتصميم الماهر له حدوده عند الزيادة المفرطة في زمن التأخير كما هو الحال عادة في الإنترنت العامة. وخلاصة القول هي أن إنترنت اليوم يمكن أن تقدّم خدمة مُرضية في أغلب الأحيان للتطبيقات الحسّاسة للتوقيت لكنها لا تستطيع تقديم أي ضمانات فيما يتعلق بالحيّز الترددي أو التوقيت.

يبين الشكل 2-5 بروتوكولات النقل المستعملة من قِبَل بعض تطبيقات الإنترنت الشائعة. نلاحظ أن تطبيقات البريد الإلكتروني والوصول إلى الحاسبات عن بُعد والويب ونقل الملفات تستخدم بروتوكول TCP. لقد اختارت تلك التطبيقات TCP أساساً لأنه يوفر خدمة نقل موثوق للبيانات تضمن وصول كل البيانات إلى وجهتها النهائية في نهاية الأمر. ونلاحظ أيضاً أن هاتف الإنترنت يعمل عادة على بروتوكول UDP. يحتاج كل جانب من تطبيق هاتف الإنترنت لبث

البيانات عبر الشبكة بحدٍّ أدنى يجب توفره لمعدل الإرسال (راجع التسجيل الصوتي الفوري في الشكل 2-4)، واحتمال تحقق ذلك مع UDP أرجح منه مع TCP. كما أن تطبيقات هاتف الإنترنت متسامحة في فقد بعض البيانات، ومن ثم فهي ليست بحاجة ماسّة إلى خدمة النقل الموثوق للبيانات التي يوفرها بروتوكول TCP.

التطبيق	بروتوكول طبقة التطبيقات	بروتوكول طبقة النقل
البريد الإلكتروني	(RFC 2821) SMTP	TCP
الوصول إلى الحاسبات عن بُعد	(RFC 854) Telnet	TCP
الويب	(RFC 2616) HTTP	TCP
نقل الملفات	(RFC 959) FTP	TCP
تطبيقات الوسائط المتعددة	HTTP (كـ YouTube)، RTP	TCP أو UDP
هاتف الإنترنت	SIP، RTP، أو ذات ملكية خاصة كـ Skype	في العادة UDP

الشكل 2-5 تطبيقات الإنترنت الشائعة وبروتوكولات طبقتي التطبيقات والنقل التي تستخدمها.

عنونة العمليات (Addressing processes)

ركزت مناقشتنا حتى الآن على خدمات نقل البيانات بين عمليتين تتصلان فيما بينهما. لكن كيف تقوم عملية ما بتحديد العملية الأخرى التي تريد الاتصال معها بواسطة تلك الخدمات؟ كيف تحدد عملية تعمل على مضيف في مدينة أمهرست بولاية ماسوشوستس في الولايات المتحدة الأمريكية بأنها تريد الاتصال مع عملية معينة تعمل على مضيف في مدينة بانكوك بتايلند؟ لتعيين العملية التي ستتسلم البيانات، نحتاج إلى معلومتين: (1) اسم أو عنوان المضيف، (2) رقم معرّف يحدد العملية المُستلِمة على مضيف الوجهة.

يُميَّز المضيف في الإنترنت بعنوان IP. سنناقش عناوين IP بمزيد من التفصيل في الفصل الرابع. أما الآن فكل ما نحتاج لمعرفته هو أن عنوان IP هو رقم يتألف من 32 بتاً، ويمكن اعتباره كطريقة لتمييز المضيف عن غيره من عقد الشبكة تمييزاً فريداً. (ومع ذلك، وكما سنرى في الفصل الرابع، فمع استخدام مترجمات عناوين الشبكة (NATs) على نطاق واسع، أصبح عنوان IP المؤلف من 32 بتاً لا يكفي وحده من الناحية العملية لعنونة المضيف بطريقة فريدة).

بالإضافة إلى معرفة عنوان مضيف الوجهة النهائية، على المضيف المُرسِل أيضاً أن يُميِّز العملية المُستلِمة التي تعمل على مضيف الوجهة. هذه المعلومات مطلوبة لأنه بصفة عامة يمكن أن يُشغِّل مضيف الوجهة العديد من تطبيقات الشبكة في نفس الوقت. يؤدي هذا الغرض رقم منفذ الوجهة، وقد خُصّصت أرقام منافذ معينة للتطبيقات الشائعة. على سبيل المثال يُميَّز خادم الويب برقم المنفذ 80، بينما تُميَّزعملية خادم البريد (والتي تستخدم بروتوكول SMTP) برقم المنفذ 25. يمكنك الاطلاع على قائمة بأرقام المنافذ المشهورة لكل بروتوكولات الإنترنت المعيارية على الموقع http://www.iana.org. عندما يُنشئ مطوّر تطبيقات تطبيق شبكة جديد يجب أن يُخصّص له رقم منفذ جديد. وسوف نتناول أرقام المنافذ بالتفصيل في الفصل الثالث.

5-1-2 بروتوكولات طبقة التطبيقات

عرفنا قبل قليل أن عمليات الشبكة تتصل فيما بينها بإرسال الرسائل إلى المقابس. لكن كيف تصاغ هذه الرسائل، وما معاني الحقول المختلفة فيها؟ ومتى ترسلها العمليات؟ هذه الأسئلة تقودنا إلى عالم بروتوكولات طبقة التطبيقات. يحدد بروتوكول طبقة التطبيقات كيف ترسل عمليات التطبيق – والتي تعمل على الأنظمة الطرفية المختلفة – رسائل إلى بعضها البعض. وبالتحديد فإن بروتوكول طبقة التطبيقات يُعرِّف ما يلي:

- أنواع الرسائل المتبادلة، كرسائل الطلب ورسائل الرد.
- صيغ الرسائل المختلفة، كالحقول الموجودة بالرسالة والفواصل بينها.

- معاني الحقول، أي معنى المعلومة في كل حقل من حقول الرسالة.
- القواعد التي تحكم متى وكيف تُرسِل عملية ما الرسائل وترد عليها.

يتم توصيف بعض بروتوكولات طبقة التطبيقات في وثائق RFCs (طلبات تعليقات) متاحة للجمهور. على سبيل المثال يوجد بروتوكول طبقة التطبيقات الخاص بالويب HTTP (بروتوكول نقل مادة الإنترنت) في طلب التعليقات RFC 2616. وبالتالي فإذا ما اتبع مطوّر لمتصفّح جديد للإنترنت قواعد HTTP المذكورة في تلك الوثيقة، فإن المتصفّح الجديد سيتمكن من استجلاب صفحات الويب من أي خادم ويب يتبع نفس قواعد HTTP المذكورة في تلك الوثيقة. يلاحظ أن العديد من البروتوكولات الأخرى لطبقة التطبيقات تعتبر ذات ملكية خاصة للشركات (proprietary)، وغير متاحة للجمهور عن قصد. فعلى سبيل المثال يستخدم العديد من أنظمة مشاركة النظائر للملفات بروتوكولات خاصة لطبقة التطبيقات.

من المهم التمييز بين تطبيقات الشبكة وبروتوكولات طبقة التطبيقات، فبروتوكول طبقة التطبيقات يمثل فقط أحد أجزاء تطبيق الشبكة (رغم كونه جزءاً هاماً بلا شك). دعنا نستعرض مثالين:

- الأول: الويب هو تطبيق زبون/خادم يسمح للمُستخدِم بالحصول على الوثائق من خادمات الويب عند الطلب، ويتكون من العديد من المكونات التي تضم: معياراً لصياغة وثيقة الويب (مثل HTML)، ومتصفّحات الويب (كمتصفّح نيتسكاب (Netscape Navigator) ومتصفّح مايكروسوفت (Microsoft Internet Explorer))، وخادمات الويب (مثل Apache، مايكروسوفت، وخادمات نيتسكاب)، وبروتوكول طبقة التطبيقات (HTTP). يحدد بروتوكول طبقة التطبيقات للويب HTTP صيغة وتسلسل الرسائل المتبادلة بين متصفّح وخادم الويب. وهكذا فإن HTTP يمثل جزءاً واحداً من أجزاء تطبيق الويب.
- الثاني: تطبيق البريد الإلكتروني يتكون أيضاً من العديد من المكوّنات، تضم خادمات البريد التي تحوي صناديق بريد الوارد والصادر للمُستخدِمين، وقارئ البريد الذي يسمح للمُستخدِم بقراءة وإنشاء الرسائل،

ومعياراً لتحديد هيكل وصيغة رسائل البريد الإلكتروني، وبروتوكولات طبقة التطبيقات والتي تحدد كيفية تبادل الرسائل بين الخادمات، وكيفية تبادل الرسائل بين الخادمات وبرامج قراءة البريد، وكيفية تفسير محتويات أجزاء معينة من رسالة البريد (على سبيل المثال سطور الترويسة للرسالة (header lines)). إن بروتوكول طبقة التطبيقات الرئيسي للبريد الإلكتروني هو SMTP (بروتوكول نقل البريد البسيط) [RFC 2821]. وهكذا فإن بروتوكول طبقة التطبيقات الرئيس للبريد الإلكتروني SMTP هو مجرّد جزء واحد (رغم كونه جزءاً هاماً) من أجزاء تطبيق البريد الإلكتروني.

6-1-2 تطبيقات الشبكة التي سنتناولها في هذا الكتاب

يتم تطوير تطبيقات جديدة للإنترنت كل يوم، بعضها عام وبعضها خاص لشركات معينة. وبدلاً من تغطية عدد كبير من تطبيقات الإنترنت بأسلوب موجز، فقد اخترنا أن نركز على عددٍ محدود من التطبيقات الهامة واسعة الانتشار. سنناقش في هذا الفصل خمسة تطبيقات مهمة: الويب، ونقل الملفات، والبريد الإلكتروني، وخدمة الدليل للإنترنت، ومشاركة النظائر للملفات. سنبدأ بمناقشة الويب، ليس فقط لأنه تطبيق منتشر على نطاق واسع، ولكن أيضاً لأن بروتوكوله (HTTP) بسيط وسهل الفهم. بعد ذلك سنتناول بروتوكول FTP بإيجاز، لأنه يعطينا مقارنة تباينية لطيفة مع HTTP. ثم نناقش البريد الإلكتروني – أول التطبيقات الحيوية للإنترنت – وهو أكثر تعقيداً من الويب، حيث إنه لا يستخدم بروتوكولاً واحداً فقط، وإنما يستخدم عدة بروتوكولات في طبقة التطبيقات. بعد البريد الإلكتروني سنتناول بروتوكول DNS والذي يوفر خدمة دليل الإنترنت لأسماء النطاقات. لا يتعامل معظم مُستخدِمي الإنترنت مباشرة مع الـ DNS، وإنما يستخدمون الـ DNS بشكلٍ غير مباشر من خلال التطبيقات الأخرى (كالويب، ونقل الملفات، والبريد الإلكتروني). ويوضح الـ DNS بشكلٍ رائع كيف أن جزءاً من الوظائف الرئيسية لقلب الشبكة (الترجمة من "اسم على

الشبكة" إلى "عنوان على الشبكة") يمكن أن يتم في طبقة التطبيقات على الإنترنت. وأخيراً سنتناول مشاركة النظائر للملفات، والتي تمثل الفئة الأكثر انتشاراً من تطبيقات الإنترنت اليوم حسب بعض القياسات (كقياس حركة مرور البيانات على الشبكة).

2-2 شبكة الويب وبروتوكول HTTP

حتى أوائل التسعينيات كانت الإنترنت تُستخدم أساساً من قِبَل الباحثين والأكاديميين وطلاب الجامعات للوصول للحاسبات عن بُعد، ولنقل الملفات من المضيفات المحلية إلى المضيفات البعيدة والعكس بالعكس، ولاستقبال وإرسال الأخبار والبريد الإلكتروني. ورغم أن هذه التطبيقات كانت (وستظل) مفيدة جداً، إلا أن الإنترنت كانت في واقع الأمر أيامها مجهولة خارج نطاق المجتمع الأكاديمي والبحثي. بعد ذلك ظهرت الشبكة العنكبوتية العالمية (الويب) [Berners-Lee 1994] إلى حيّز الوجود في أوائل التسعينيات كتطبيق جديد وهام للإنترنت والذي شد انتباه جمهور الناس إلى الإنترنت. لقد غيّر الويب بشكلٍ ملحوظ كيفية تفاعل الناس داخل وخارج بيئات عملهم. لقد نقل هذا التطبيق الإنترنت بشكلٍ أساسي من كونها مجرد إحدى شبكات البيانات العديدة إلى كونها شبكة البيانات الواحدة والفريدة.

لعل ما يروق لأكثر مُستخدمي الويب هو كونه يعمل حسب الطلب، فهم يستقبلون ما يريدونه عندما يريدونه، بخلاف الإذاعة والتلفزيون حيث يُجبرون على توليف أجهزتهم لاستقبال البرامج عندما يقوم موفّرو المحتوى ببث تلك البرامج وإتاحة محتواها للجمهور. وبالإضافة إلى توفر المحتوى حسب الطلب، فللويب العديد من الميزات الرائعة الأخرى التي يحبها الناس ويقدرونها. فمن السهل جداً لأي فرد إضافة معلومات وجعلها متاحة عبر تلك الشبكة العنكبوتية، فبوسع كل شخص أن يصبح ناشراً بتكلفة زهيدة للغاية. فالوصلات التشعبية (hyperlinks) ومحركات البحث (search engines) تساعدنا على أن نبحر خلال محيط من مواقع الويب. والرسومات تحرك حواسنا، والأشكال وبرامج جافا

التفاعلية والعديد من الأدوات الأخرى تمكننا من التفاعل مع الصفحات والمواقع، كما يوفر الويب واجهة قوائم (menu interface) لِكَم هائل من المواد المسموعة والمرئية (audio and video) المخزنة على الإنترنت بصيغة الوسائط المتعددة (multimedia) والتي يمكن الوصول إليها حسب الطلب.

1-2-2 نظرة عامة على بروتوكول HTTP

يعتبر بروتوكول نقل مادة الإنترنت HTTP – بروتوكول طبقة التطبيقات للويب – بمثابة القلب من الشبكة العنكبوتية (الويب). ويوصّف هذا البروتوكول في RFC 1945 وRFC 2616، وينفّذ في برنامجين: برنامج الزبون وبرنامج الخادم. يتصل برنامجا الزبون والخادم (واللذان يعملان على نظامين طرفيين مختلفين) فيما بينهما بتبادل رسائل HTTP. ويتضمن بروتوكول HTTP تحديد الهيكل البنائي (النسق) لتلك الرسائل، وكيفية تبادلها بين برنامجي الزبون والخادم. قبل توضيح HTTP بالتفصيل ينبغي أن نراجع بعض مصطلحات الويب.

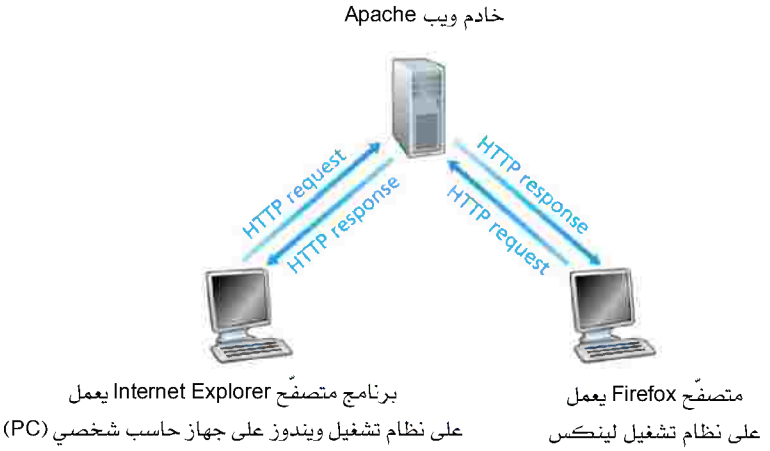
تتكون صفحة الويب – والتي تسمى أيضاً وثيقة (document) – من عناصر يطلق عليها كائنات (objects)، والكائن ببساطة هو ملف له عنوان URL وحيد، كملف HTML أو ملف صورة JPEG أو ملف برنامج جافا أو ملف مقطع فيديو (video clip). تشمل معظم صفحات الويب ملف HTML أساسي ومراجع (عناوين) لعدة كائنات. على سبيل المثال إذا كانت صفحة ويب تتضمن نص HTML وخمس صور JPEG فإنها تتكون من ستة كائنات: ملف HTML الأساسي بالإضافة إلى ملفات للصور الخمس. يرتبط ملف HTML الأساسي بالكائنات الأخرى في الصفحة عن طريق عناوين الـ URL للكائنات. يتألف عنوان URL من مكونين: اسم المضيف للخادم الذي يأوي الكائن، واسم المسارالخاص بالكائن (أي مكان تواجده على المضيف). على سبيل المثال يتكون العنوان:

http://www.someSchool.edu/someDepartment/picture.gif

من www.someSchool.edu والذي يمثل اسم المضيف للخادم، وsomeDepartment/picture.gif/ يمثل اسم المسار. نظراً لأن متصفّحات الويب (مثل

Explorer وFirefox) تطبق جانب الزبون لبروتوكول HTTP، فسنستخدم (في سياق الويب) مصطلحي المتصفّح (browser) والزبون بالتبادل للدلالة على نفس الشيء. أما خادمات الويب (Web servers) – والتي تطبق جانب الخادم لبروتوكول HTTP – فتأوي كائنات الويب، حيث يوجد لكلٍّ منها عنوان URL. من خادمات الويب الشائعة: Apache، وMicrosoft Internet Information Server (IIS).

يحدد بروتوكول HTTP كيفية طلب زبائن الويب صفحات الويب من خادمات الويب، وسوف نناقش التفاعل بين الزبون والخادم بالتفصيل لاحقاً، لكن الفكرة العامة موضحة في الشكل 2-6. فعندما يطلب مُستخدِم الإنترنت صفحة ويب (على سبيل المثال بالضغط على وصلة تشعبية (رابط) (hyperlink))، يرسل المتصفّح إلى الخادم رسائل طلب HTTP للحصول على الكائنات الموجودة في الصفحة المطلوبة. يستقبل الخادم بدوره تلك الطلبات ويستجيب برسائل رد HTTP تتضمن الكائنات المطلوبة.



الشكل 2-6 رسائل الطلب والرد لبروتوكول HTTP.

يعتمد HTTP على TCP كبروتوكول نقل أساسي (بدلاً من تشغيله فوق بروتوكول UDP). يبدأ زبون HTTP أولاً بالاتصال بالخادم عن طريق بروتوكول TCP، وعندما يتحقق الاتصال فإن برنامجيْ المتصفّح والخادم يستخدمان TCP من خلال واجهات المقبس. وكما وصفنا في الجزء 2-1، تُعتبر واجهة المقبس على جانب الزبون بمثابة المدخل بين عملية الزبون وتوصيلة TCP؛ وعلى جانب الخادم تعتبر واجهة المقبس المدخل بين عملية الخادم وتوصيلة TCP. يرسل الزبون رسائل طلب HTTP إلى واجهة مقبسه ويتلقى رسائل رد HTTP من خلال نفس الواجهة. وبنفس الطريقة يستقبل خادم HTTP رسائل طلب عبر واجهة مقبسه ويرسل رسائل الرد من خلالها أيضاً. وبمجرد أن يبعث الزبون رسالة إلى واجهة مقبسه فإن الرسالة تصبح خارج يده وتكون تحت سيطرة بروتوكول TCP. تذكّر من الجزء 2-1 أن بروتوكول TCP يوفر خدمة نقل موثوق للبيانات لبروتوكول HTTP. وهذا يعني ضمناً أن كل رسالة طلب HTTP أُرسِلت من عملية الزبون ستصل سليمة في النهاية إلى الخادم. وبنفس الطريقة فإن كل رسالة رد HTTP يرسلها الخادم ستصل للزبون سليمة في النهاية. تتضح هنا إحدى المزايا الهامة للبنية الطبقية لبروتوكولات الشبكة؛ فلا يلزم بروتوكول HTTP أن يكترث للبيانات التي قد تفقد، ولا بتفاصيل كيفية استعادة TCP لها، ولا بإعادة ترتيب البيانات ضمن الشبكة، فتلك مهمة بروتوكول TCP والبروتوكولات من أسفله في الطبقات الدنيا لرصة البروتوكولات (protocol stack).

من المهم ملاحظة أن الخادم يرسل الملفات المطلوبة إلى الزبائن بدون تخزين لأية معلومات عن الحالة (state information) فيما يتعلق بطلبات الزبائن، فمثلاً إذا طلب زبون ما نفس الكائن مرتين في غضون بضع ثوانٍ، فلن يرد الخادم بأنه قد سبق وأرسل ذلك الكائن إلى الزبون منذ قليل، بل سيرسل الخادم الكائن مرة أخرى لأنه قد نسي تماماً ما قام بعمله في السابق. ونظراً لأن خادم HTTP لا يحتفظ بأية معلومات عن الزبائن لذا يوصف بأنه بروتوكول بدون حالة (stateless). كما يجدر بنا الإشارة إلى أن الويب يستخدم بنية زبون/خادم كما

تقدّم وصفه في الجزء 2-1. كما أن خادم الويب يعمل دائماً وله عنوان IP ثابت، ويقوم على خدمة ملايين الطلبات المحتملة من المتصفّحات المختلفة للزبائن.

2-2-2 التوصيلات الدائمة والتوصيلات غير الدائمة (Non-Persistent and Persistent Connections)

يتواصل الزبون والخادم في العديد من تطبيقات الإنترنت لفترات طويلة بإرسال الزبون سلسلة من الطلبات واستجابة الخادم لكلٍّ منها. وعلى حسب التطبيق وكيفية استخدامه فإن سلسلة الطلبات قد ترسل واحداً تلو الآخر (back-to-back)، أو بشكلٍ دوري على فترات منتظمة، أو بشكلٍ متقطع. وعند حدوث هذا التفاعل بين الخادم والزبون على بروتوكول TCP يتعين على مطوّري التطبيق اتخاذ قرار هام ألا وهو هل ينبغي إرسال كل زوج من رسائل الطلب والرد باستخدام توصيلة TCP منفصلة أو إرسال كل الطلبات وردودها على نفس توصيلة TCP؟ في الطريقة الأولى يقال إن التطبيق يستخدم توصيلات غير دائمة (non-persistent connections)؛ وفي الطريقة الثانية يقال إنه يستخدم توصيلات دائمة (persistent connections). ولفهم هذه القضية بعمق، دعنا نستعرض ميزات وعيوب الاتصالات الدائمة في سياق تطبيق HTTP، والذي يمكن أن يستخدم كلتا الطريقتين (الدائمة وغير الدائمة) للاتصال. فرغم أن بروتوكول HTTP يستخدم التوصيلات الدائمة في النمط الاعتيادي (default mode)، فإنه من الممكن تغيير تهيئة الخادم والزبون لاستخدام التوصيلات غير الدائمة بدلاً من ذلك.

بروتوكول HTTP بتوصيلات غير دائمة

دعنا نتتبع خطوات نقل صفحة ويب من الخادم إلى المتصفّح في حالة التوصيلات غير الدائمة. افترض أن الصفحة تتكون من ملف HTML أساسي وعشر صور JPEG، وأن كلاً من تلك الكائنات الأحد عشر موجودة على نفس الخادم، ولو فرضنا أن عنوان URL لملف HTML الأساسي هو:

http://www.someSchool.edu/someDepartment/home.index

فإن الخطوات تتم كالتالي:

1. تُنشئ عملية زبون HTTP توصيلة TCP مع الخادم www.someSchool.edu من خلال المنفذ رقم 80 (منفذ HTTP المعتاد) وكنتيجة لذلك سيصبح هناك مقبس اتصال (socket) بالزبون ومقبس اتصال بالخادم.
2. يُرسل زبون HTTP رسالة طلب HTTP إلى الخادم عن طريق مقبس الاتصال لديه تتضمن مسار الملف /someDepartment/home.index (سنتناول بعض تفاصيل رسائل HTTP لاحقاً).
3. يستقبل خادم HTTP رسالة الطلب عن طريق المقبس لديه ويحصل على الملف /someDepartment/home.index من وحدة التخزين لديه (ذاكرة RAM أو القرص الصلب)، ثم يُضمِّنه في رسالة رد HTTP، ويرسلها إلى الزبون عن طريق مقبس الاتصال لديه.
4. يطلب خادم HTTP من بروتوكول TCP إغلاق توصيلة TCP. (ولكن بروتوكول TCP لا يغلق التوصيلة في واقع الأمر حتى يتأكد من أن الزبون تسلم رسالة الرد سليمة).
5. يستقبل زبون HTTP رسالة الرد وبعدها ينهي بروتوكول TCP الاتصال. وتشير رسالة الرد إلى أن الكائن عبارة عن ملف HTML، فيقوم الزبون باستخراج الملف من رسالة الرد ثم يفحص محتواه ليجد عناوين صور JPEG العشرة.
6. تتكرر الخطوات الأربع الأولى لكل عنوان من عناوين صور JPEG.

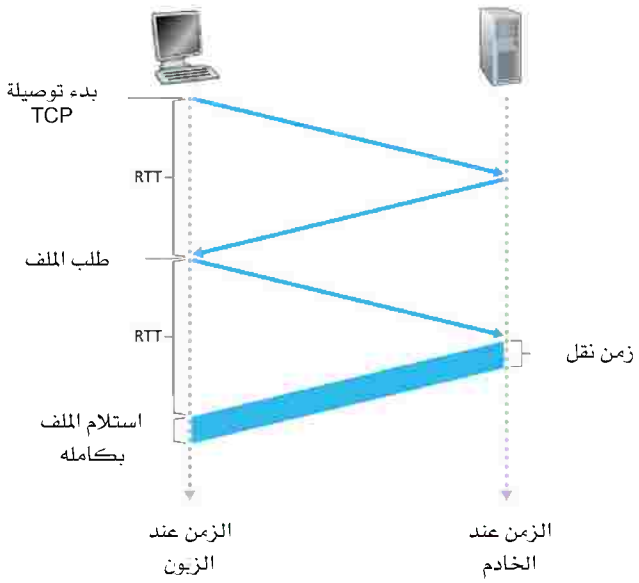
عندما يستقبل المتصفّح صفحة الويب يقوم بعرضها للمُستخدِم. ويمكن أن تختلف ترجمة متصفّحين مختلفين لصفحة ويب واحدة (أي يمكن أن يعرضاها للمُستخدِم بطرق مختلفة بعض الشيء). ليس لـ HTTP أية علاقة بكيفية تَرجمة صفحة الويب من قِبَل الزبون، فمواصفات HTTP المنشورة في RFC 1945 و RFC 2616 تُعرِّف فقط بروتوكول الاتصال بين برنامج زبون HTTP وبرنامج خادم HTTP.

توضح الخطوات المذكورة أعلاه استخدام التوصيلات غير الدائمة (non-persistent connections)، حيث إن كل توصيلة TCP تُغلَق بعد أن يرسل الخادم الكائن ولا تستمر التوصيلة لنقل الكائنات الأخرى. لاحظ أن كل توصيلة TCP تنقل فقط رسالة طلب واحدة ورسالة رد واحدة. وهكذا فعندما يطلب المُستخدِم صفحة الويب في المثال السابق فإنه يتم إنشاء إحدى عشرة توصيلة TCP.

في الخطوات المذكورة أعلاه، تعمّدنا عدم توضيح ما إذا كان الزبون يحصل على العشر صور باستخدام عشر توصيلات TCP متسلسلة، أو يحصل على البعض منها على توصيلات TCP متوازية. في الحقيقة يمكن أن يهيئ المُستخدِمون المتصفّحات الحديثة للتحكم في درجة التوازي. وفي أنماط الاستخدام المعتادة تفتح معظم المتصفّحات من خمس إلى عشر توصيلات TCP على التوازي، يقوم كلٌّ منها بالتعامل مع طلب واحد والرد عليه، وإذا أراد المُستخدِم يمكن ضبط العدد الأقصى لتوصيلات TCP المتوازية إلى واحد، وفي هذه الحالة يتم إنشاء التوصيلات العشر بشكلٍ متسلسل. وكما سنرى في الفصل القادم يُقصِّر استعمال التوصيلات المتوازية من زمن الاستجابة.

قبل الاستمرار في الشرح دعنا نقدر بشكلٍ تقريبي الوقت الذي ينقضي منذ أن يطلب الزبون ملف HTML الأساسي إلى أن يستقبله بالكامل. للقيام بذلك سنُعرِّف أولاً وقت الرحلة ذهاباً وإياباً (Round-Trip-Time (RTT)) على أنه الوقت المُستغرَق لنقل رزمة بيانات صغيرة من الزبون إلى الخادم ثم عودتها بعد ذلك إلى الزبون. يشمل هذا الوقت زمن انتقال الرزمة (propagation delay)، والزمن الذي تقضيه الرزمة في صفوف الانتظار (queuing delay) عند الموجّهات والمحوّلات (routers and switches)، وزمن معالجة الرزمة (processing delay)، وقد سبق مناقشة تلك الأزمنة في الجزء 1-4. لنر ما يحدث عندما ينقر المُستخدِم على وصلة تشعبية. كما يوضح الشكل 2-7 يُنشئ المتصفّح توصيلة TCP بينه وبين خادم الويب، ويتضمن ذلك "مصافحة ثلاثية"؛ حيث يرسل الزبون قطعة بيانات TCP صغيرة إلى الخادم فيقر الخادم باستلام الرسالة، ويرد بقطعة بيانات TCP صغيرة، وأخيراً يقر الزبون باستلامه القطعة من الخادم. يستغرق أول جزأين من المصافحة

الثلاثية زمن RTT واحد. ثم يرسل الزبون رسالة طلب HTTP مدمجة مع الجزء الثالث من المصافحة الثلاثية (إشعار الاستلام) عبر توصيلة TCP. فور وصول رسالة الطلب إلى الخادم، يرسل الخادم ملف HTML إلى توصيلة TCP، حيث يستغرق ذلك زمن RTT آخر. وهكذا فإن الزمن الكلي يعادل ضعف RTT تقريباً، بالإضافة إلى زمن إرسال ملف HTML عند الخادم.



الشكل 2-7 حساب الزمن المستغرق لطلب ملف HTML والحصول عليه.

بروتوكول HTTP بتوصيلات دائمة

إن للتوصيلات غير الدائمة بعض العيوب. أولاً: يجب إنشاء توصيلة جديدة والإبقاء عليها لكل كائن مطلوب. تتطلب كلٌّ من تلك التوصيلات تخصيص مساحة من الذاكرة للتخزين المؤقت والاحتفاظ بمتغيرات TCP في كلٍّ من الزبون والخادم. يمكن أن يُشكِّل ذلك عبئاً كبيراً على خادم الشبكة، والذي قد يقوم

على خدمة طلبات من مئات الزبائن المختلفين في نفس الوقت. ثانياً: يواجه كل كائن – كما وضحنا آنفاً – زمن تأخير للتوصيل يعادل ضعف RTT (واحد لإنشاء توصيلة TCP وواحد لطلب واستلام الكائن).

أما في حالة التوصيلات الدائمة، فيترك الخادم توصيلة TCP مفتوحة بعد إرسال الرد وذلك لإرسال الطلبات والردود اللاحقة بين نفس الزبون والخادم. وبالتحديد، يمكن إرسال صفحة ويب كاملة (في المثال السابق ملف HTML الأساسي والصور العشرة) على توصيلة TCP دائمة واحدة. وعلاوة على ذلك يمكن إرسال عدد من صفحات ويب الموجودة على نفس الخادم من الخادم إلى نفس الزبون عبر توصيلة TCP دائمة واحدة. ويمكن إرسال الطلبات الخاصة بتلك الكائنات متلاصقة الواحد تلو الآخر، بدون انتظار للرد على الطلبات الجاري تنفيذها بطريقة خط الأنابيب (pipelining). عادةً ما يغلق خادم HTTP التوصيلة عند بقائها غير مستعملة لوقت معين (مدة انقضاء فترة الموقت ويمكن تغييرها). وعندما يستلم الخادم طلبات الكائنات متلاصقة فإنه يُرسل أيضاً الكائنات متلاصقة الواحد تلو الآخر. يستخدم بروتوكول HTTP عادة توصيلات دائمة بطريقة خط الأنابيب. وسوف نقارن أداء التوصيلات غير الدائمة والتوصيلات الدائمة بشكلٍ كمّي في تمارين الفصل الثاني والثالث. كما نشجعك على مراجعة [Heidemann 1997; Nielsen 1997].

3-2-2 صيغة رسائل HTTP

تتضمن مواصفات HTTP (في RFC 2616) تحديداً لصيغ رسائل HTTP، وهي على نوعين: رسائل طلب ورسائل رد، وسيتم مناقشتهما فيما يلي.

رسائل طلب HTTP

يوضح المثال التالي نموذجاً نمطياً لرسالة طلب HTTP:

```
GET /somedir/page.html HTTP/1.1
Host: www.someschool.edu
Connection: close
User-agent: Mozilla/4.0
Accept-language: fr
```

يمكننا تعلم الكثير بإلقاء نظرة فاحصة على رسالة الطلب البسيطة تلك. أولاً: نلاحظ أن الرسالة مكتوبة بصيغة ASCII كي يتسنى للشخص الذي لديه دراية بالحاسب قراءتها. ثانياً: نلاحظ أن الرسالة تتكون من خمسة سطور، ينتهي كلٌّ منها برمز بداية سطر (carriage return) وتغذية سطر جديد (line feed). أما السطر الأخير من الرسالة فينتهي برمز آخر لبداية سطر وتغذية سطر جديد. بالرغم من أن رسالة الطلب المعينة هذه تتكون من خمسة أسطر، فإن رسالة الطلب عموماً يمكن أن تتضمن سطوراً أكثر أو أقل بحد أدنى سطر واحد. يسمى السطر الأول في رسالة طلب HTTP سطر الطلب (request line)، بينما تسمى السطور اللاحقة سطور الترويسة (header lines). يتكون سطر الطلب من ثلاثة حقول: حقل الأمر (method)، وحقل العنوان (URL)، وحقل رقم إصدار HTTP. يمكن أن يأخذ حقل الأمر عدة قيم مختلفة مثل: GET، POST، HEAD، PUT، DELETE. تستعمل الغالبية العظمى من رسائل طلب HTTP أمر GET والذي يستخدم عندما يريد المتصفّح طلب كائن حيث يضع مسار الكائن المطلوب في حقل العنوان URL. في هذا المثال يطلب المتصفّح الكائن /somedir/page.html (وهو عبارة عن صفحة ويب). أما رقم إصدار HTTP فواضح في هذا المثال أنه عبارة عن HTTP/1.1.

الآن دعنا ننظر إلى سطور الترويسة في المثال السابق. سطر الترويسة

Host: www.someschool.edu

يحدد هذا السطر المضيف الذي يوجد عليه الكائن. قد يبدو أن هذا السطر غير ضروري، حيث إن هناك توصيلة TCP سابقة مع المضيف، لكن كما سنرى في الجزء 2-2-5 فإن المعلومات في سطر الترويسة عن المضيف مطلوبة للذاكرة المخبأة في وكيل الويب – والذي يُعرَف أيضاً بالخادم المفوّض أو بروكسي الويب (web proxy). وبتضمين السطر

Connection: close

فإن المتصفّح يُخبر الخادم بأنه لا يريد التوصيلات الدائمة (persistent connections)، وإنما يريده أن يغلق التوصيلة بعد إرسال الكائن المطلوب. أما السطر

User-agent: Mozilla/4.0

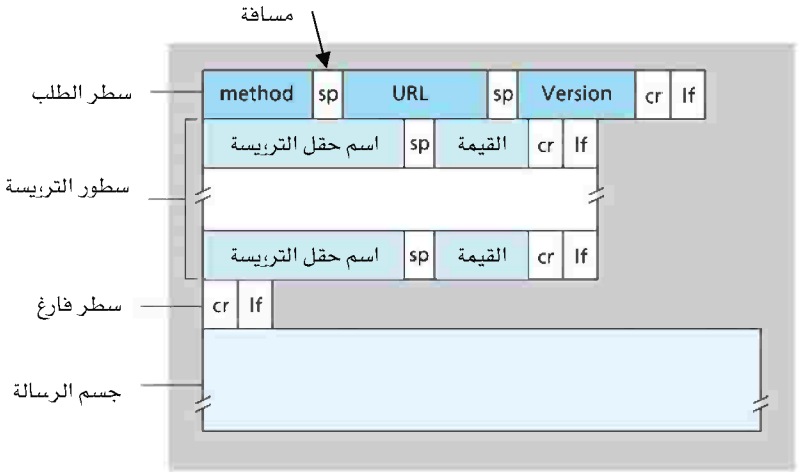
فيحدد نوع المتصفّح الذي أرسل الطلب إلى الخادم، ففي المثال السابق يأخذ القيمة Mozilla/4.0 (أحد المتصفّحات من شركة Netscape). وهذا السطر مفيد لأن الخادم يمكن أن يرسل نسخاً مختلفة من نفس الكائن إلى الأنواع المختلفة من المتصفّحات (كلٌّ منها معنون بنفس العنوان URL). وأخيراً يشير السطر

Accept-language: fr

إلى أن المُستخدِم يفضل استلام نسخة باللغة الفرنسية من الكائن (إذا وجدت على الخادم وإلا فسيرسل الخادم النسخة المعتادة لديه). وهذا السطر من سطور الترويسة هو مجرد مثال للعديد من المفاوضات (negotiations) المتوفرة في HTTP بخصوص المحتوى.

بعد أن استعرضنا المثال السابق دعنا نفحص الآن الصيغة العامة لرسالة طلب كما هي موضحة بالشكل 2-8، وسنرى أن الصيغة العامة قريبة جداً من المثال السابق. ومع ذلك فلعلك لاحظت أنه بعد سطور الترويسة (والسطر الفارغ) يوجد "محتوى الكيان" (entity body)، وهو حقل فارغ عند استخدام GET، ولكنه يُستخدَم مع POST. يستخدم زبون HTTP في أغلب الأحيان POST عندما يملأ المُستخدِم استمارة بيانات، مثلاً عندما يدخل المُستخدِم كلمات الدليلية

(keywords) إلى محرك البحث. في حالة رسالة POST يطلب المُستخدِم أيضاً صفحة ويب من الخادم لكن المحتويات المعينة لصفحة الويب تعتمد على ما أدخله المُستخدِم في حقول البيانات في الاستمارة والتي ترسل ضمن جسم الرسالة.



الشكل 2-8 الصيغة العامة لرسالة طلب HTTP.

وسنكون مقصرين إذا لم نذكر أن رسالة الطلب التي تتولد مع الاستمارة لا يتعين بالضرورة أن تستخدم POST، وإنما في أغلب الأحيان تستخدم طريقة GET وترسل البيانات المدخلة في حقول الاستمارة كجزء من عناوين URL المطلوبة. على سبيل المثال إذا استعملت استمارة بيانات طريقة GET، وبافتراض أن الاستمارة لها حقلان وكانت المُدخلات لهما monkeys وbananas، فإن حقل العنوان URL في الرسالة يصبح

www.somesite.com/animalsearch?monkeys&bananas

ولعلك لاحظت خلال تصفحك اليومي للويب عناوين URL طويلة فمن المحتمل أن تكون من هذا النوع. يشبه الأمر HEAD الأمر GET، فعندما يتلقى خادم طلباً يحتوي على HEAD يرد برسالة HTTP ولكن بدون الكائن المطلوب. وغالباً ما

يستخدم مطوّرو التطبيقات طريقة HEAD عند تعقب وتصحيح الأخطاء أثناء تطوير التطبيق. يُستخدم الأمر PUT في أغلب الأحيان مرتبطاً مع أدوات النشر على الويب، فهو يسمح للمُستخدِم بتحميل كائن إلى مسار معين (مجلد) على خادم ويب معين، كما يُستخدَم بواسطة التطبيقات التي تحتاج لتحميل كائنات على خادمات الويب. أما الأمر DELETE فيسمح لمُستخدِم أو تطبيق بحذف كائن موجود على خادم الويب.

رسائل رد HTTP

نورد فيما يلي رسالة رد HTTP نمطية يمكن أن تكون رسالة رد على رسالة الطلب في المثال الذى تناولناه سابقاً:

```
HTTP/1.1 200 OK
Connection: close
Date: Thu, 07 Jul 2007 12:00:15 GMT
Server: Apache/1.3.0 (Unix)
Last-Modified: Sun, 6 May 2007 09:23:24 GMT
Content-Length: 6821
Content-Type: text/html

(data data data data data ...)
```

دعنا ننظر بعناية في رسالة الرد تلك. تتكون الرسالة من ثلاثة أجزاء: سطر حالة (status line)، وستة سطور ترويسة (header lines)، ثم بعد ذلك جسم الكيان (entity body). إن جسم الكيان هو صُلب الرسالة ويحتوي على الكائن المطلوب نفسه (ممثلاً بـ ... data data data data data). يتكون سطر الحالة من ثلاثة حقول: حقل رقم نسخة بروتوكول HTTP، ورمز الحالة (status code)، ورسالة الحالة المناظرة. في هذا المثال يُشير سطر الحالة إلى أن الخادم يستخدم HTTP/1.1 وأن كل شيء على ما يرام (أي أن الخادم وجد الكيان المطلوب ويقوم بإرساله). دعنا الآن نفحص سطور الترويسة. يستخدم الخادم السطر:

Connection: close

ليخبر الزبون بأنه سيغلق توصيلة TCP بعد إرسال الرسالة.

Date: Thu, 07 Jul 2007 12:00:15 GMT

يُشير إلى وقت وتاريخ إنشاء الرسالة وإرسالها من الخادم. لاحظ أن هذا ليس وقت وتاريخ إنشاء الكائن أو آخر تعديل له، وإنما وقت جلب الخادم للكائن من نظام الملفات لديه، وإدراجه ضمن رسالة الرد، وإرسالها.

Server: Apache/1.3.0 (Unix)

يُشير إلى أن الرسالةَ أنشئت من خادم ويب Apache، وهي مماثلة لسطر العنوان

User-agent:

والتي تحدد نوع المتصفّح في رسالة طلب HTTP. أما السطر:

Last-Modified: Sun, 6 May 2007 09:23:24 GMT

فيُشير إلى وقت وتاريخ إنشاء الكائن أو آخر تعديل له، وسوف نتناوله قريباً بتفصيل أكثر نظراً لأهميته للتخزين المؤقت للكائن في الذاكرة المخبأة لدى كل من الزبون المحلي وخادمات الذاكرة المخبأة على الشبكة. أما السطر

Content-Length: 6821

فيُشير إلى حجم الكائن المُرسَل (بالبايتات). والسطر:

Content-Type: text/html

يُشير إلى أن الكائن المتضمن في الرسالة (جسم الكيان) نص HTML، (لاحظ أن نوع الكائن يحدد رسمياً بهذا السطر وليس بامتداد الملف).

بعد أن استعرضنا مثالاً دعنا الآن نفحص الصيغة العامة لرسالة الرد كما هي موضحة في الشكل 2-9، والتي تتطابق مع المثال السابق. دعنا نضيف بضع كلمات إضافية عن رموز الحالة (status code) وعباراتها (phrases) المناظرة. يُشير رمز الحالة والعبارة المصاحبة له إلى نتيجة الطلب. فيما يلي بعض رموز الحالة والعبارات المصاحبة لها:

- 200 OK

تعني نجاح العملية وإرسال المعلومات المطلوبة في الرسالة.

- 301 Moved Permanently

تعني أن الكائن المطلوب تم نقله بصفة دائمة للموقع المحدد في السطر Location من سطور الترويسة في رسالة الرد. سيقوم برنامج الزبون باسترجاع الكائن من الموقع الجديد تلقائياً.

- 400 Bad Request

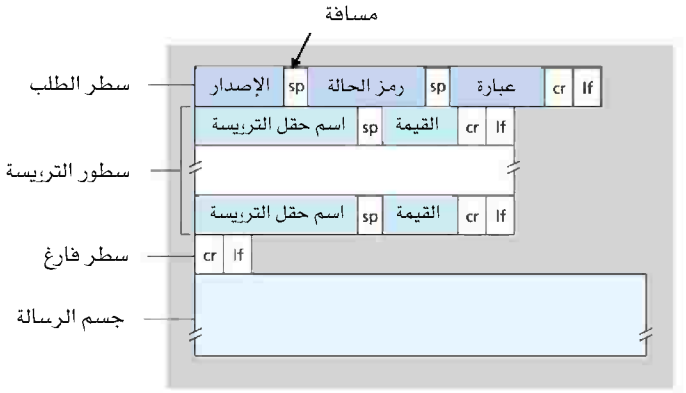
رسالة خطأ عامة تعني أن الخادم لم يفهم الطلب.

- 404 Not Found

تعني أن الكائن المطلوب غير موجود على الخادم.

- 505 HTTP Version Not Supported

تعني أن نسخة HTTP المُستخدَمة غير مدعومة من قِبَل الخادم.



الشكل 2-9 الصيغة العامة لرسالة رد HTTP.

هل تريد أن ترى رسالة رد HTTP حقيقية؟ هذا الأمر يوصى به بشدة كما أنه سهل المنال! أولاً استخدم telnet للوصول إلى خادم الويب المفضل لديك، ثم

اكتب رسالة طلب تتكون من سطر واحد لكائن ما على الخادم، على سبيل المثال عن طريق واجهة الأوامر (command prompt) اكتب:

```
telnet cis.poly.edu 80
GET /~ross/ HTTP/1.1
Host: cis.poly.edu
```

ثم اضغط على زر الإدخال مرتين بعد السطر الأخير. يتم فتح توصيلة TCP إلى منفذ 80 على المضيف cis.poly.edu، وبعد ذلك تُرسَل رسالة طلب HTTP. وكنتيجة لذلك سترى رسالة رد تتضمن ملف HTML الأساسي للصفحة الرئيسية للأستاذ Ross بجامعة بولي تكنك. أما إذا أردت أن ترى فقط سطور الترويسة لرسالة HTTP دون الحصول على الكائن نفسه فاستخدم HEAD بدلاً من GET. وأخيراً استبدل /~ross/ بـ /~banana/ لترى رسالة خطأ في الرد.

في هذا الجزء ناقشنا عدداً من سطور الترويسة التي تستخدم ضمن رسائل طلب ورد HTTP. وتُعرِّف مواصفات HTTP الكثير من سطور الترويسة والتي يمكن أن تقوم بإدخالها المتصفّحات وخادمات الويب وخادمات الذاكرة المخبأة على الشبكة. لقد تم تغطية عدد محدود فقط من مجموع سطور الترويسة، وسوف نغطي بضعة سطور أخرى فيا بعد وعدداً قليلاً آخر عندما نناقش استخدام الويب للذاكرة المخبأة في الجزء 2-2-5. توجد مناقشة شاملة وسهلة القراءة لنظام HTTP تتضمن رموز الحالة وسطور الترويسة في [Krishnamurty 2001]؛ راجع أيضاً [Luotonen 1998] لاستعراض الموضوع من وجهة نظر مطوّري التطبيقات.

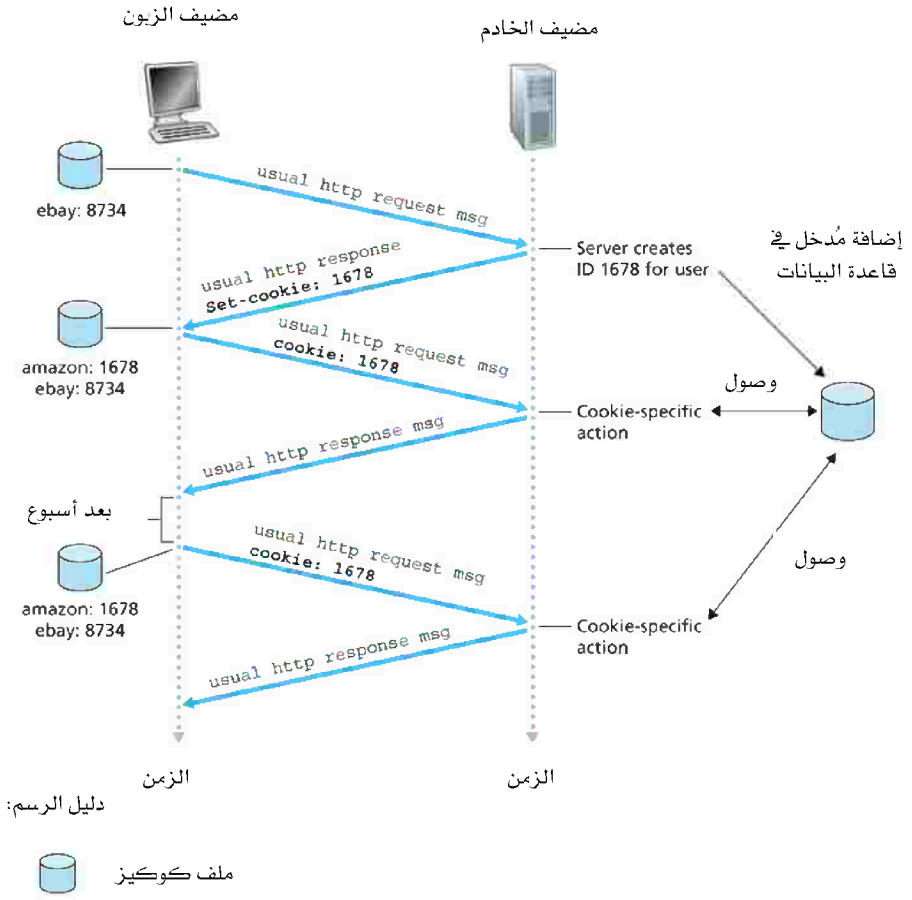
كيف يقرر متصفّحٌ ما سطور الترويسة التي يُضمِّنها في رسالة الطلب؟ وكيف يقرر خادم ما على الويب سطور الترويسة التي يُضمِّنها في رسالة الرد؟ يتم ذلك طبقاً لنوع المتصفّح ورقم إصداره (على سبيل المثال لا يولد متصفّح HTTP/1.0 أياً من سطور الترويسة الخاصة بمتصفّح HTTP/1.1)، وكذلك طبقاً لتهيئة أو إعداد المُستخدِم للمتصفّح (على سبيل المثال اللغة المُفضَّلة)، وأيضاً إذا ما كان

لديه حالياً نسخة من الكائن في ذاكرته المخبأة ولكنها قد تكون منتهية التاريخ. كذلك تتصرف خادمات الويب بنفس الطريقة: هناك العديد من المنتجات والإصدارات والإعدادات المختلفة، والتي تؤثر جميعها على سطور الترويسة المُتضمَّنة في رسائل الرد.

4-2-2 التفاعل بين المُستخدِم والخادم : الكوكيز (Cookies)

سبق أن ذكرنا أن خادم HTTP بلا ذاكرة للحالة (stateless)، مما يبسط تصميمه ويسمح للمهندسين بتطوير خادمات ويب عالية الأداء يمكن أن تعالج آلاف توصيلات TCP في نفس الوقت. ومع ذلك فغالباً ما يرغب موقع الويب في أن تكون لديه القدرة على تمييز المُستخدِمين، إما لأن الخادم يرغب في قصر الوصول إليه على مُستخدِمين بعينهم، أو لأنه يرغب في ربط خدمة المحتوى بهوية المُستخدِم. ولهذه الأغراض يستخدم بروتوكول HTTP الكوكيز (وهي مُعرّفة في طلب التعليقات RFC 2965) والتي تسمح لمواقع الويب بمتابعة المُستخدِمين. تستخدم أكثر مواقع الويب التجارية الرئيسة الكوكيزهذه الأيام.

كما هو موضح في الشكل 2-10 تتضمن تقنية الكوكيز أربعة مكوّنات: (1) سطر ترويسة للكوكيز في رسالة رد HTTP، (2) سطر ترويسة للكوكيز في رسالة طلب HTTP، (3) ملف كوكيز مُخزّن على النظام الطرفي للمُستخدِم ومُدار بمتصفّح المُستخدِم، (4) قاعدة بيانات في الخلفية على موقع الويب. بالاستعانة بشكل 2-10 دعنا نتتبع مثالاً يوضح كيفية عمل الكوكيز. افرض أن سوزان التي تستخدم دائماً المتصفّح إنترنت إكسبلورر من جهاز حاسبها الشخصي في بيتها، زارت موقع Amazon.com للمرة الأولى، ودعنا نفرض أنه قد سبق لها أن زارت موقع eBay. عندما يجيء الطلب إلى خادم ويب Amazon يقوم الخادم بإنشاء رقم فريد لتعريف هوية المُستخدِم ويضيف مدخلاً جديداً إلى قاعدة بياناته المفهرسة بذلك الرقم التعريفي. بعد ذلك يرد خادم ويب Amazon على متصفّح سوزان مُضمِّناً في الرد سطر الترويسة Set-cookie والذي يحتوي على الرقم التعريفي، على سبيل المثال: Set-cookie: 1678.



الشكل 2-10 الاحتفاظ بحالة المُستخدِم عن طريق الكوكيز.

وعندما يتسلم متصفّح سوزان رسالة الرد، يرى ذلك السطر.عندئذ يضيف المتصفّح سطراً إلى ملف الكوكيز الخاص الذي يحتفظ به لديه يتضمن اسم مضيف الخادم وذلك الرقم التعريفي. لاحظ أن ملف الكوكيز يحتوي أيضاً على مُدخل لموقع eBay منذ أن زارت سوزان ذلك الموقع في الماضي. وبينما تواصل سوزان تَصفُّح موقع Amazon، ففي كل مرة تطلب صفحة ويب، يستشير متصفّحها ملف

الكوكيز ويستخلص الرقم التعريفي لذلك الموقع، ويضع سطر ترويسة لرسالة الطلب يتضمن الرقم التعريفي. وبالتحديد ستتضمن كل طلباتها المُرسَلة إلى خادم Amazon سطر الترويسة:

Cookie: 1678

وبهذا الشكل يستطيع خادم Amazon تعقب نشاط سوزان في موقع Amazon بالرغم من أن موقع ويب Amazon لا يعرف بالضرورة اسم سوزان، وإنما يعرف بالضبط أي الصفحات زارها المُستخدِم رقم 1678، وبأي ترتيب وفي أي وقت! يستخدم موقع Amazon الكوكيز لتقديم خدمة عربة التسوّق (shopping cart) – حيث يمكن له الاحتفاظ بقائمة بكل البنود التي تنوي سوزان شراءها، لكي تتمكن من دفع ثمنهم مرة واحدة في نهاية الجلسة. وإذا عادت سوزان إلى موقع Amazon لاحقاً (بعد أسبوع مثلاً)، سيواصل متصفّحها وضع سطر الكوكيز

Set-cookie: 1678

في رسائل الطلب. كما يمكن أيضاً أن يقترح موقع Amazon بعض المنتجات لسوزان حسب صفحات الويب التي زارتها في الماضي. أمّا إذا سجّلت سوزان نفسها أيضاً بموقع Amazon بإعطاء اسمها بالكامل، وعنوان بريدها الإلكتروني، وعنوانها البريدي، ومعلومات بطاقتها الائتمانية – فإن موقع Amazon يمكن عندئذ أن تضيف تلك المعلومات إلى قاعدة بياناتها وبذلك يرتبط اسم سوزان بالرقم التعريفي لها (وكل الصفحات التي زارتها على الموقع في الماضي!). وهذه هي كيفية توفير موقع Amazon ومواقع التجارة الإلكترونية الأخرى خدمة "التسوّق بنقرة واحدة". فعندما تختار سوزان شراء مادة أثناء زيارة لاحقة، لن تحتاج إلى إعادة إدخال اسمها أو رقم بطاقة الائتمان لها أو عنوانها.

من هذه المناقشة نرى أن الكوكيز cookies يمكن أن تُستعمَل لتمييز المُستخدِم. فأول مرة يزور المُستخدِم موقعاً ما يُعطى رمزاً تعريفياً (مثلاً اسمه أو اسمها). وأثناء الجلسات اللاحقة يُرسِل المتصفّح سطر الكوكيز إلى الخادم، وبذلك يستطيع الخادم تمييز المُستخدِم. وهكذا يمكن استعمال الكوكيز

لإنشاء طبقة الجلسة للمُستخدِم فوق بروتوكول HTTP (الذى لا يتذكر الحالة). فعلى سبيل المثال عندما يدخل المُستخدِم إلى تطبيق البريد الإلكتروني على الإنترنت (كـ hotmail مثلاً) يُرسِل المتصفّح معلومات الكوكيز إلى الخادم مما يسمح للخادم بتمييز المُستخدِم طوال جلسة اتصاله بالتطبيق.

على الرغم من أن الكوكيز غالباً ما يبسّط عملية التسوّق من خلال الإنترنت فهناك جدل حولها، حيث يمكن أيضاً أن تضر بالخصوصية (privacy). فكما لاحظنا سابقاً يمكن لموقع الويب – باستخدام الكوكيز مع معلومات حساب المُستخدِم – أن يعرف الكثير عن المُستخدِم ثم يبيع تلك المعلومات إلى طرف ثالث. يتضمن موقع [Cookie Central 2007] معلومات شاملة عن الجدل القائم حالياً حول الكوكيز.

5-2-2 ذاكرة الويب المخبأة (Web Caching)

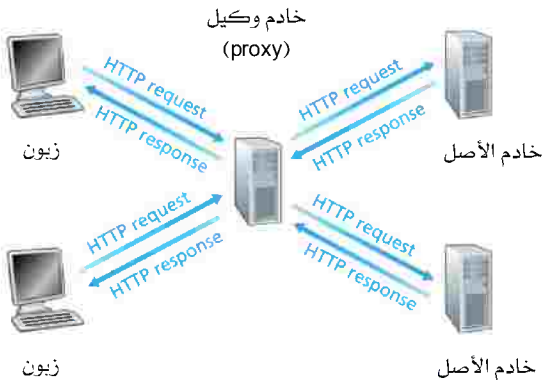
ذاكرة الويب المخبأة والتي تعرف أيضاً بالخادم المفوّض أو الوكيل (proxy server) هي أحد كيانات الشبكة التي تخدم طلبات HTTP نيابةً عن خادم الويب الأصلي، ولها أقراص تخزين (disk storage) خاصة بها تحتفظ فيها بنسخ من الكائنات المطلوبة حديثاً. كما يوضح الشكل 2-11، يمكن ضبط المتصفّح بحيث يُوجّه كل طلبات HTTP من المُستخدِم أولاً إلى ذاكرة الويب المخبأة. بمجرد إتمام عملية الضبط تلك، سيوجّه المتصفّح كل طلب أولاً إلى ذاكرة الويب المخبأة. كمثال افرض أن المتصفّح يطلب الكائن:

http:://www.someschool.edu/campus.gif

فسيحدث ما يلي:

1. يُنشئ المتصفّح توصيلة TCP إلى ذاكرة الويب المخبأة ويُرسِل إليها طلب HTTP للكائن.
2. تفحص ذاكرة الويب المخبأة لترى ما إذا كان لديها نسخة من الكائن مخزنة محلياً، فإذا وجدتها فإنها ترسلها ضمن رسالة رد HTTP إلى المتصفّح.

3. أما إذا لم يكن لديها الكائن، تفتح ذاكرة الويب المخبأة توصيلة TCP إلى الخادم الأصلي (المصدر) أي www.someschool.edu، وبعد ذلك ترسل ذاكرة الويب المخبأة طلب HTTP للكائن عبر توصيلة TCP إلى ذلك الخادم . وبعد استلام الخادم الأصلي لهذا الطلب، يُرسِل الكائن ضمن رسالة رد HTTP إلى ذاكرة الويب المخبأة.
4. عندما تتسلم ذاكرة الويب المخبأة الكائن تقوم بتخزين نسخة منه في وحدة التخزين المحلي لديها وتُرسِل نسخة ضمن رسالة رد HTTP إلى متصفّح الزبون (على توصيلة TCP الحالية بين متصفّح الزبون وذاكرة الويب المخبأة).



الشكل 2-11 الزبائن تطلب كائنات عن طريق ذاكرة الويب المخبأة.

لاحظ أن الذاكرة المخبأة تعمل كخادم وكزبون في نفس الوقت، فعندما تستلم طلبات من المتصفّح وتُرسل الردود إليه، فهي عندئذ تعمل كخادم، وعندما ترسل بالطلبات إلى الخادم الأصلي وتستلم الردود منه فإنها تعمل عند ذاك كزبون. عادةً ما تُشتَري ذاكرة الويب المخبأة وتُركَّب من قِبَل موفّر خدمة الإنترنت. على سبيل المثال قد تُركِّب جامعةٌ ما ذاكرة مخبأة على شبكة الحرم الجامعي، وتضبط كل متصفّحات الحرم الجامعي لتشير إلى تلك الذاكرة المخبأة. كما يمكن أن يُركِّب موفّر رئيس لخدمة الإنترنت السكني

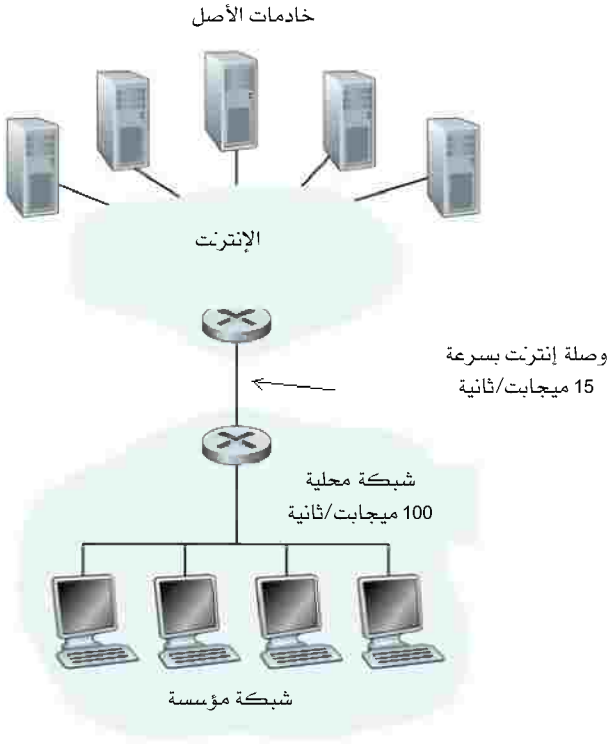
(residential ISP) (مثل AOL) واحدةً أو أكثر من تلك الذاكرات على شبكته، ويضبط متصفّحاته لتشير إليها.

يرجع انتشار استخدام ذاكرة الويب المخبأة في الإنترنت لسببين. أولاً: يمكن أن تقلل ذاكرة الويب المخبأة وقت الرد على طلب الزبون بشكلٍ ملحوظ، خاصة إذا كان الحيّز الترددي الذي يمثل عنق الزجاجة بين الزبون والخادم الأصلي أقل بكثير من الحيّز الترددي الذي يمثل عنق الزجاجة بين الزبون والذاكرة المخبأة. وإذا كانت الوصلة سريعة بين الزبون والذاكرة المخبأة (كما هو الحال في الغالب) وكانت الذاكرة المخبأة تحوي الكائن المطلوب، فحينئذ سيكون بوسع الذاكرة المخبأة تسليم الكائن بسرعة إلى الزبون. ثانياً: كما سنوضح قريباً بمثال يمكن أن تؤدي الذاكرة المخبأة إلى خفض حركة البيانات على وصلة اتصال مؤسسة (شركة أو جامعة مثلاً) بالإنترنت بشكلٍ جوهري. وبخفض حركة البيانات فإن المؤسسة (الشركة أو الجامعة) لن تكون مضطرة لترقية (توسّعة) (upgrade) الحيّز الترددي لوصلتها بالإنترنت بسرعة، مما يؤدي إلى خفض التكلفة. وعلاوةً على ذلك يمكن أن تقلل ذاكرة الويب المخبأة حركة بيانات الويب في الإنترنت ككل بشكلٍ ملحوظ مما يُحسِّن أداء كل التطبيقات.

ولفهم فوائد الذاكرة المخبأة بعمق دعنا نأخذ مثالاً ضمن سياق الشكل 2-12 والذي يبين شبكتين (شبكة المؤسّسة وبقيّة شبكة الإنترنت العامة). شبكة المؤسّسة هي شبكة اتصالات محلية بسرعة عالية (High-Speed LAN). يتصل الموجّه على تلك الشبكة بموجّه الإنترنت بوصلة سرعتها 15 ميجابت/ثانية. ترتبط خادمات المصدر بالإنترنت وهي موزّعة في جميع أنحاء الكرة الأرضية. افترض أن الحجم المتوسط للكائن هو 1 ميجابت، وأن المعدل المتوسط لطلبات الحصول على الكائنات من متصفّحات المؤسّسة إلى خادمات الأصل هو 15 طلب في الثانية. وافترض أن رسائل طلب HTTP صغيرة جداً (بدرجة يمكن إهمالها)، وبالتالي لن تُولّد أي حركة مرور تُذكر للبيانات في الشبكات أو في الوصلة من موجّه المؤسّسة إلى موجّه الإنترنت. افترض أيضاً أن الوقت الذي يمر من اللحظة التي يقوم فيها الموجه على جانب الإنترنت من الوصلة في الشكل 2-12 بتوجيه طلب

HTTP (ضمن وحدة بيانات IP) إلى أن يتلقى الرد (عادة على شكل عدة وحدات بيانات IP) يعادل ثانيتين في المتوسط. وبطريقة غير رسمية يطلق على زمن التأخير هذا "تأخير الإنترنت".

زمن الاستجابة الكلي (أي الفترة الزمنية من حين طلب المتصفّح الكائن حتى حصوله عليه) هو مجموع زمن التأخير على الشبكة المحلية وزمن التأخير على وصلة التوصل بالإنترنت (أي الوصلة بين الموجّهين) وزمن التأخير على الإنترنت. دعنا نجري بعض الحسابات التقريبية لتقدير ذلك الزمن.



الشكل 2-12 عنق الزجاجة بين شبكة مؤسّسة والإنترنت.

تبلغ كثافة المرور على الشبكة المحلية (راجع الجزء 1-4-2):

(15 requests/sec) × (1 Mbits/request) / (100 Mbps) = 0.15

بينما تبلغ كثافة المرورعلى الوصلة من موجّه الإنترنت إلى موجّه المؤسّسة:

(15 requests/sec) × (1 Mbits/request) / (15 Mbps) = 1

تؤدي كثافة المرور 0.15 على شبكة الاتصالات المحلية إلى تأخير يعادل (على الأغلب) عشرات الميللي ثانية، وبالتالي يمكننا إهمال التأخير على شبكة الاتصالات المحلية. ولكن (كما نوقش في الجزء 1-4-2) مع اقتراب كثافة المرور من واحد (كما هو الحال على الوصلة بين الموجّهين في الشكل 2-12) فإن التأخيرعلى الوصلة يصبح كبيراً جداً وينمو بدون حد. وهكذا فإن الزمن المتوسط لتحقيق الطلبات سيصبح في حدود الدقائق (إن لم يكن أكثر)، وهذا غير مقبول لمُستخدِمي المؤسّسة، مما يُحتِّم إيجاد حل بديل.

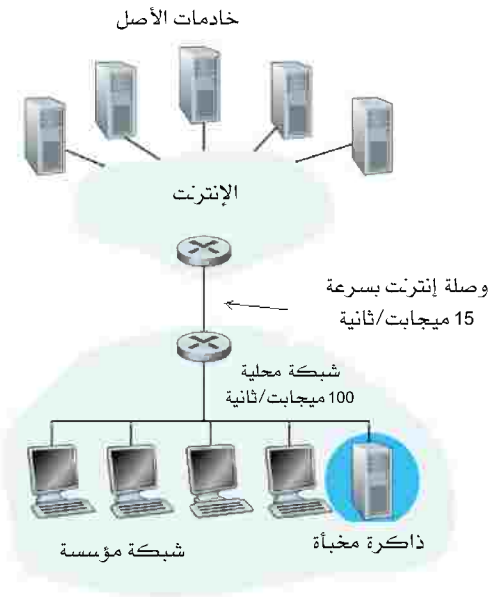
قد يكون أحد الحلول الممكنة زيادة سرعة وصلة التوصل بالإنترنت إلى 100 ميجابت/ثانية مثلاً بدلاً من 15 ميجابت/ثانية. سيؤدي ذلك إلى تخفيض كثافةَ المرور على وصلة التوصل إلى 0.15 والتي تؤدي إلى تأخير بسيط بين الموجّهين. في هذه الحالة سيكون وقت الرد الكلي ثانيتين تقريباً، أي مساوياً لتأخير الإنترنت. لكن هذا الحل يعني أيضاً أن المؤسّسة يجب أن تُرقِّي الوصلة التي تربطها بالإنترنت من 15 ميجابت/ثانية إلى 100 ميجابت/ثانية وهو حل مكلّف.

لننظر بعين الاعتبار لحل بديل لا يتطلب ترقيّة لوصلة الإنترنت، ولكن (بدلاً من ذلك) يتضمن تركيب ذاكرة ويب مخبأة على شبكة المؤسّسة، كما هو موضّح في الشكل 2-13. يترواح معدل إصابة الهدف (hit rate) (نسبة الطلبات التي تتمكن الذاكرة المخبأة من تلبيتها) ما بين 0.2 إلى 0.7 عملياً. ومن أجل الإيضاح دعنا نفترض أن الذاكرة المخبأة توفر معدل إصابة للهدف يعادل 0.4 لتلك المؤسّسة. ولأن الزبائن والذاكرة المخبأة متصلون بنفس شبكة الاتصالات المحلية السريعة فإن 40٪ من الطلبات تقريباً ستُلبى فوراً (مثلاً خلال 10 ميللي ثانية) من

الذاكرة المخبأة. أما نسبة الـ 60٪ المتبقية من الطلبات فلا يزال من الضروري إرسالها إلى خادمات الأصل. لكن نظراً لأن 60٪ فقط من الكائنات المطلوبة تعبر الآن وصلة الإنترنت، فإن كثافة المرورعلى الوصلة تنخفض من 1.0 إلى 0.6. وطبيعي أن كثافة مرور أقل من 0.8 تناظر تأخيراً صغيراً (مثلاً بضع عشرات ميللي ثانية) على الوصلة بسرعة 15 ميجابت/ثانية. هذا التأخير بسيط مقارنة بالثانيتين (زمن تأخير الإنترنت). ولذا فإن متوسط زمن التأخير تحت هذه الظروف يساوي:

$$0.4 \times (0.01 \text{ seconds}) + 0.6 \times (2.01 \text{ seconds})$$

وهذا يتجاوز 1.2 ثانية بقليل. وهكذا يوفر هذا الحل الثاني زمن استجابة أقل من الحل الأول، كما أنه لا يتطلب من المؤسّسة ترقية وصلة الإنترنت لديها. وبالطبع يجب على المؤسّسة شراء وتركيب ذاكرة ويب مخبأة، غير أن تكلفة ذلك منخفضة - فالعديد من ذاكرات الويب المخبأة تستخدم برامج عامة تعمل على حاسبات شخصية رخيصة.



الشكل 2-13 إضافة ذاكرة مخبأة إلى شبكة المؤسّسة.

6-2-2 أمر GET الشّرطي

رغم أن استخدام ذاكرة مخبأة يمكن أن يُخفّض أوقات الاستجابة المحسوسة للمُستخدِم، إلا أنها تأتي بمشكلة جديدة (فَقَد تتقادم نسخة الكائن الموجود على الذاكرة المخبأة). وبمعنى آخر ربما يكون قد جرى تعديل على الكائن الموجود على خادم الويب الأصلي بعد وضع نسخة من ذلك الكائن في الذاكرة المخبأة لدى الزبون. لحسن الحظ يتضمن بروتوكول HTTP آلية تسمح للذاكرة المخبأة بالتحقق من أن الكائنات حديثة حتى اللحظة. هذه الآلية هي أمر GET الشّرطي. تسمّى رسالة طلب HTTP برسالة GET الشّرطية إذا (1) استعملت رسالة الطلب أمر GET، (2) تضمنت الرسالة سطر الترويسة:

If-Modified-Since:

ولتوضيح كيفية عمل أمر GET الشّرطي دعنا نستعرض هذا المثال. أولاً: يُرسل خادم الذاكرة المخبأة نيابةً عن المتصفّح رسالة طلب إلى خادم الويب:

GET /fruit/kiwi.gif HTTP/1.1
Host: www.exotiquecuisine.com

ثانياً: يُرسِل خادم الويب رسالة رد بالكائن المطلوب إلى الذاكرة المخبأة:

HTTP/1.1 200 OK
Date: Thu, 7 Jul 2007 15:39:29
Server: Apache/1.3.0 (Unix)
Last-Modified: Mon, 4 Jul 2007 09:23:24
Content-Type: image/gif

(data data data data data ...)

تُرسِل الذاكرة المخبأة الكائن إلى المتصفّح الطالب لكن أيضاً مع الاحتفاظ بنسخة من الكائن محلياً. كما تخزن الذاكرة المخبأة تاريخ آخر تعديل سويّة مع الكائن. ثالثاً: بعد أسبوع من ذلك التاريخ يطلب متصفّح آخر نفس

الكائن عن طريق الذاكرة المخبأة، والكائن ما زال في الذاكرة المخبأة. نظراً لاحتمال كون هذا الكائن قد عُدِّل في خادم الويب خلال ذلك الأسبوع، تقوم الذاكرة المخبأة بالتأكد من ذلك عن طريق إصدار رسالة GET الشّرطية. وبالتحديد تُرسِل الذاكرة المخبأة:

```
GET /fruit/kiwi.gif HTTP/1.1
Host: www.exotiquecuisine.com
If-Modified-Since: Wed, 4 Jul 2007 09:23:24
```

لاحظ أن قيمة السطر If-Modified-Since تساوي بالضبط قيمة Last-Modified التي أرسلها الخادم قبل أسبوع مضى. يطلب هذا الامر الشّرطي من الخادم أن يُرسِل الكائن من جديد إذا كان قد عُدِّل منذ التاريخ المحدد. افترض أن الكائن لم يُعدَّل منذ 4 يوليو/تموز 2007 09:23:24. حينئذ يُرسِل خادم الويب رسالة رد إلى الذاكرة المخبأة:

```
HTTP/1.1 304 Not Modified
Date: Thu, 14 Jul 2007 15:39:29
Server: Apache/1.3.0 (Unix)

(empty entity body)
```

وكما نرى في الرد على رسالة GET الشّرطية، ما زال خادم الويب يُرسِل رسالة رد لا تتضمن الكائن المطلوب. فتضمين الكائن المطلوب سيؤدي إلى إهدار الحيّز الترددي المتاح، وزيادة زمن الاستجابة المحسوس للمُستخدِم (خصوصاً إذا كان حجم الكائن المطلوب كبيراً). لاحظ أن رسالة الرد الأخيرة هذه تحتوي على

```
344 Not Modified
```

في سطر الحالة (أي لم يحدث تعديل للكائن المطلوب منذ التاريخ المذكور)، وهذا يخبرالذاكرة المخبأة بأن بإمكانها المُضي قُدُماً في إرسال نسخة الكائن الموجودة عليها إلى المتصفّح الطالب.

بهذا تنتهي مناقشتنا لبروتوكول HTTP (أول بروتوكولات الإنترنت التي ندرسها بالتفصيل، وهو أحد بروتوكولات طبقة التطبيقات). ولقد استعرضنا من

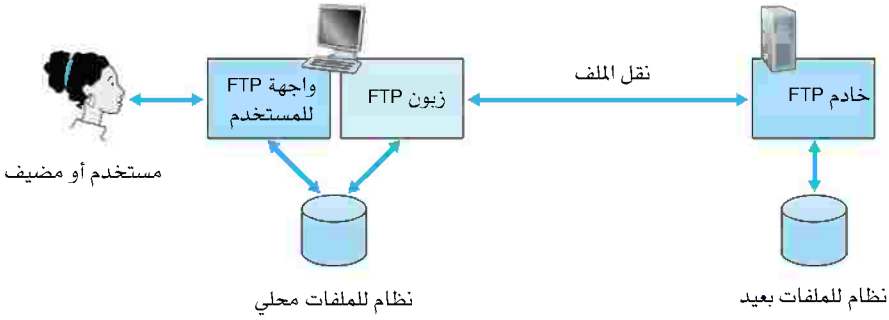
خلال تلك المناقشة صيغ رسائل HTTP والأعمال التي يقوم بها كلٌّ من زبون وخادم الويب عند إرسال الرسائل واستلامها. وتعرّضنا بعض الشيء للبنية التحتية لتطبيق الويب، بما في ذلك الذاكرة المخبأة والكوكيز وقواعد البيانات الخلفية والتي ترتبط جميعها بشكلٍ أو بآخر ببروتوكول HTTP.

3-2 نقل الملفات باستخدام بروتوكول FTP

في جلسة FTP المعتادة يجلس المُستخدِم أمام حاسبه (المضيف المحلي) ويريد نقل ملفات من مضيف بعيد وإليه. ولكي يتمكن المُستخدِم من الوصول للحاسب البعيد يجب عليه أولاً أن يُدخل تعريف المُستخدِم وكلمة المرور (كلمة السّر) الخاصين به. وبعد إعطاء معلومات التفويض هذه يمكنه نقل ملفات من نظام الملفات المحلي إلى نظام الملفات البعيد والعكس بالعكس. وكما هو موضح في الشكل 2-14 يتعامل المُستخدِم مع بروتوكول FTP من خلال برنامج زبون المُستخدِم للبروتوكول. يقوم المُستخدِم أولاً بإدخال اسم المضيف للجهاز البعيد وبالتالي تقوم عملية زبون FTP في المضيف المحلي بإنشاء توصيلة TCP مع عملية الخادم في المضيف البعيد. عندئذ يُدخِل المُستخدِم تعريف المُستخدِم وكلمة السّر الخاصين به، فتُرسَل عبر توصيلة TCP كجزء من أوامر FTP إلى عملية الخادم. بمجرد تأكد الخادم من صلاحية التعامل مع المُستخدِم يمكن للمُستخدِم أن ينسخ ملفاً أو أكثر من الملفات المخزّنة في نظام الملفات المحلي إلى نظام الملفات البعيد (أو العكس).

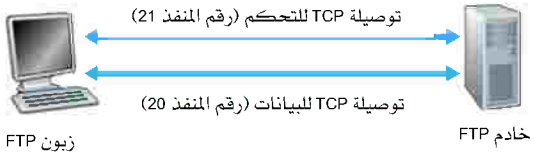
يعتبر HTTP وFTP بروتوكولات لنقل الملفات ولهما العديد من الخصائص المشتركة. على سبيل المثال كلاهما بروتوكول لطبقة التطبيقات يعمل فوق بروتوكول TCP. ومع ذلك توجد بعض الاختلافات المهمة بينهما. إن الاختلاف الأكثر وضوحاً هو أن FTP يستخدم توصيلتي TCP متوازيتين لنقل ملف ما، يُطلق على الأولى توصيلة التحكم (control connection) وعلى الثانية توصيلة البيانات (data connection). تُستخدَم توصيلة التحكم لإرسال معلومات التحكم بين المضيفين – كتعريف المُستخدِم، وكلمة السّر، وأوامر تغيير مجلد الملفات البعيد

(remote directory)، والأوامر الخاصة بـ "تحميل" أو"تنزيل" الملفات. أما توصيلة البيانات فتستخدم لإرسال الملف الحقيقي. وبسبب استخدام FTP توصيلة تحكم مستقلة يقال إن FTP يرسل معلومات التحكم خارج النطاق (out-of-band). سنرى في الفصل السابع أن بروتوكول RTSP (والذي يُستخدَم للتحكم في تشغيل الوسائط المتعددة كتسجيلات الصوت والفيديو) يُرسِل معلومات التحكم أيضاً خارج النطاق.



الشكل 2-14 نقل ملفات بين نظامي الملفات المحلي والبعيد باستخدام FTP.

أما HTTP (كما تتذكّر) فيُرسِل سطور الترويسة (header lines) لرسائل الطلب والرد على نفس توصيلة TCP التي تحمل الملف المنقول، ولهذا السبب يُقال إن HTTP يرسل معلومات التحكم داخل النطاق (in-band). في الجزء التالي سوف نرى أن بروتوكول SMTP (البروتوكول الأساسي للبريد الإلكتروني) يرسل معلومات التحكم أيضاً داخل النطاق. يوضح الشكل 2-15 توصيلة التحكم وتوصيلة البيانات لبروتوكول FTP.



الشكل 2-15 توصيلتا التحكم والبيانات لبروتوكول FTP.

عندما يبدأ المُستخدِم جلسة FTP مع مضيف بعيد فإن جانب زبون FTP (المُستخدِم) يبدأ بإنشاء توصيلة تحكم TCP إلى الخادم (المضيف البعيد) على منفذ الخادم رقم 21. يُرسِل جانب زبون FTP تعريف المُستخدِم وكلمة السّر على تلك التوصيلة، كما يُرسِل جانب زبون FTP أيضاً على توصيلة التحكم أوامر لتغيير الدليل البعيد. وعندما يستلم جانب الخادم أمراً على توصيلة التحكم لنقل ملف (سواءً إلى المضيف البعيد أو منه)، يبدأ جانب الخادم توصيلة بيانات TCP إلى الزبون. يُرسِل FTP ملفاً واحداً على توصيلة البيانات، وبعدها يغلق توصيلة البيانات تلك. وإذا أراد المُستخدِم أثناء نفس الجلسة نقل ملف آخر، يفتح FTP توصيلة بيانات أخرى. وهكذا فإن FTP يُبقِي توصيلة التحكم مفتوحة طوال مدّة جلسة المُستخدِم، لكنه يُنشئ توصيلة بيانات جديدة لكل ملف يتم نقله أثناء الجلسة (أي أن توصيلات البيانات في FTP غير دائمة (non-persistent).

يجب أن يحتفظ الخادم بمعلومات عن حالة المُستخدِم أثناء الجلسة. وبشكلٍ خاص يجب أن يقرن الخادم توصيلة التحكم مع حساب مُستخدِم (user account) معين، كما يجب أن يتتبع الخادم دليل الملفات الحالي للمُستخدِم أثناء تجوّله خلال شجرة الدليل (الفهرس) على المضيف البعيد. يلاحظ أن متابعة تلك المعلومات الرسمية لكل جلسة مستمرة يُحِد بشكلٍ ملحوظ من العدد الكلي للجلسات التي يمكن أن يقوم بها بروتوكول FTP في نفس الوقت. تذكّر في المقابل أن بروتوكول HTTP عديم الحالة (stateless)، بمعنى أنه لا يتتبع أية حالة للمُستخدِم.

1-3-2 أوامر وردود بروتوكول FTP

دعنا ننهي هذا الجزء بمناقشة قصيرة لبعض أوامر وردود FTP الأكثر شيوعاً. ترسل الأوامر (من الزبون إلى الخادم)، والردود (من الخادم إلى الزبون) عبر توصيلة التحكم في صيغة ASCII بطول 7 بتات. وهكذا فإن أوامر FTP – مثله في ذلك مثل HTTP – يمكن قراءتها من قِبَل الناس. ولفصل الأوامر المتعاقبة يُستخدَم رمز بداية سطر (carriage return) ورمز تغذية سطر جديد (line feed).

يتكون كل أمر من أربعة حروف ASCII كبيرة كما أن بعض الأوامر لها معاملات اختيارية. وفيما يلي وصف لبعض الأوامر الأكثر شيوعاً:

- USER username

يُستخدَم لإرسال تعريف المُستخدِم إلى الخادم.

- PASS password

يُستخدَم لإرسال كلمة السّر الخاصة بالمُستخدِم إلى الخادم .

- LIST

يُستخدَم لطلب إرسال قائمة بكل الملفات الموجودة في الدليل البعيد الحالي من الخادم. تُرسَل قائمة الملفات على توصيلة بيانات (جديدة وغير دائمة) بدلاً من توصيلة TCP الخاصة بالتحكم.

- RETR filename

يُستخدَم لاستجلاب ملف من الدليل الحالي على المضيف البعيد. عند تلقي هذا الأمر يُنشئ المضيف البعيد توصيلة بيانات ويرسل الملف المطلوب عليها.

- STOR filename

يُستخدَم لتخزين (وضع) ملف في الدليل الحالي للمضيف البعيد.

يوجد عادةً تناظر (واحد لواحد) بين الأمر الذي يصدره المُستخدِم وأمر FTP الذي يُرسَل عبر توصيلة التحكم. ويتبع كل أمر يُرسَل من الزبون إلى الخادم رد يُرسَل من الخادم إلى الزبون. يتكون هذا الرد من رقم يتألف من ثلاث خانات مع رسالة اختيارية بعده. لاحظ وجه الشبه في هذا التركيب مع رمز الحالة وعبارة الحالة في سطر الحالة ضمن رسالة رد HTTP. فيما يلي أمثلة لبعض الأجوبة مع رسائلها المحتملة:

- 331 Username OK, password required

اسم المُستخدِم صحيح ومطلوب كلمة السّر.

- 125 Data connection already open; transfer starting

توصيلة البيانات مفتوحة فعلاً والنقل على وشك البدء.

- 425 Can't open data connection

تعذّر فتح توصيلة البيانات.

- 452 Error writing file

حدث خطأ عند كتابة (تخزين) الملف.

وندعو القرّاء المهتمّين بتعلم أوامر وردود FTP الأخرى للاطلاع على RFC 959.

4-2 البريد الإلكتروني (E-mail)

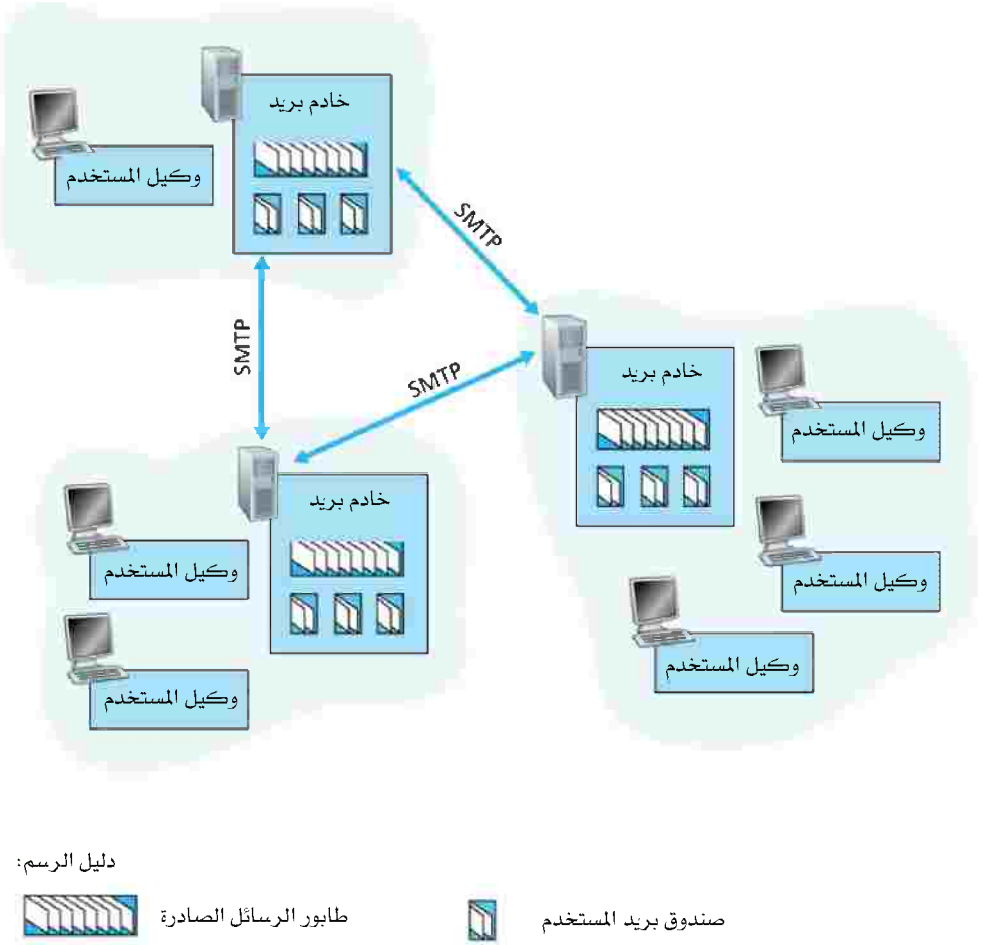
لقد ظهر البريد الإلكتروني منذ بداية الإنترنت، وكان التطبيق الأكثر شعبية عندما كانت الإنترنت في مهدها [Segaller 1998]، على مرِّ السنين أصبح هذا التطبيق أكثر إتقاناً وقوة، ويبقى اليوم أحد أكثر تطبيقات الإنترنت أهمية وانتشاراً. كما هو الحال مع البريد العادي، يوفّرالبريد الإلكتروني وسط اتصال لاتزامني، فالناس تُرسِل وتقرأ الرسائل متى تيسر لهم ذلك بدون الحاجة للتنسيق مع جداول مواعيد الناس الآخرين. يمتاز البريد الإلكتروني مقارنةً بالبريد العادى بسرعته وسهولة توزيعه ورخص كلفته. وللبريد الإلكتروني الحديث العديد من الميزات الكبيرة كاستخدام قوائم المراسلة (العناوين) والتي عن طريقها يمكن إرسال رسائل البريد الإلكتروني ورسائل الدعاية إلى آلاف المُستخدِمين في نفس الوقت. كما تتضمن رسائل البريد الإلكتروني الحديثة الملحقات في أغلب الأحيان، والروابط التشعبية، والنصوص المنسّقة بطريقة HTML، والصور.

سنتناول في هذا الجزء بروتوكولات طبقة البرامج التي تقع في قلب بريد الإنترنت الإلكتروني. لكن قبل أن نقفز إلى مناقشة مفصّلة لتلك البروتوكولات، دعنا نلقِ نظرة مبسطة على نظام بريد الإنترنت ومكوّناته الرئيسة. يوضح الشكل 2-16 نظاماً بسيطاً لبريد الإنترنت والذي يتألف من ثلاثة مكوّنات رئيسة كما يظهر من الشكل: وكلاء المُستخدِمين، وخادمات البريد، وبروتوكول نقل البريد البسيط SMTP. نصف الآن كلاً من تلك المكوّنات ضمن سياق المُرسِل

أليس (Alice) التي ترسل رسالة بريد إلكتروني إلى المُستقبل بوب (Bob). يسمح وكلاء المُستخدِمين لهم بقراءة الرسائل والرد عليها وإعادة توجيهها وتخزينها وتأليف رسائل جديدة. (أحياناً يُطلَق على وكلاء المُستخدِمين للبريد الإلكتروني قرّاء البريد، رغم أننا سنتفادى استخدام هذا التعبير عموماً في هذا الكتاب).

عندما تُنهي أليس إعداد رسالتها يُرسِل وكيل المُستخدِم لديها الرسالة إلى خادم البريد، حيث توضع في طابور الرسائل الخارجة. وعندما يريد بوب قراءة رسالة، يسترجع وكيل المُستخدِم لديه الرسالة من صندوق بريده في خادم البريد. في أواخر التسعينيات شاع استعمال واجهات المُستخدِم الرسومية GUI، مما سمح للمُستخدِمين بمشاهدة وتأليف رسائل الوسائط المتعددة. حالياً تعتبر برامج Microsoft Outlook، Apple Mail، Mozilla Thunderbird ضمن واجهات المُستخدِم الرسومية الشائعة للبريد الإلكتروني. كما يوجد أيضاً العديد من الواجهات النصية للبريد الإلكتروني (مثل mail، pine، elm) والمتوفرة في ساحة البرامج العامة بالإضافة إلى الواجهات المبنيّة على الويب، كما سنرى بعد قليل.

تُشكِّل خادمات البريد لب البنية التحتية للبريد الإلكتروني. لكل مُستلِم مثل بوب صندوق بريد موجود على أحد خادمات البريد تلك. ويُدير صندوق بريد بوب الرسائل التي أُرسِلت إليه ويحتفظ بها. عادةً تبدأ الرسالة رحلتها في وكيل المُستخدِم المُرسِل وتسافر إلى خادم البريد المُرسِل، ثم إلى خادم البريد المُستقبل، حيث تُودَع في صندوق بريد المُستقبل. وعندما يريد بوب الوصول للرسائل في صندوق بريده، يتحقق خادم البريد الذي يحتوي هذا الصندوق من شخصية بوب (عن طريق معرّف المُستخدِم وكلمة السّر). يجب أيضاً أن يتعامل خادم بريد أليس مع حالات تعطل خادم بريد بوب. إذا لم يستطع خادم أليس توصيل البريد إلى خادم بوب، يحتفظ خادم أليس بالرسالة في طابور ويحاول توصيلها لاحقاً. تُعاد المحاولة من جديد مرة كل 30 دقيقة تقريباً. وإذا لم ينجح خادم المُرسِل في ذلك بعد مُضيّ عدة أيام، يقوم الخادم بحذف الرسالة ويحيط المُرسِل عِلماً بذلك عن طريق رسالة بريد إلكتروني.



الشكل 2-16 نظرة مبسطة لنظام البريد الإلكتروني.

يعتبر SMTP بروتوكول طبقة البرامج الرئيس لبريد الإنترنت الإلكتروني. يستخدم SMTP خدمة النقل الموثوق للبيانات التي يوفرها بروتوكول TCP لنقل البريد من خادم البريد المُرسِل إلى خادم البريد المُستقبِل. وكما هو الحال مع معظم بروتوكولات طبقة البرامج، لبروتوكول SMTP جانبان: جانب الزبون (الذي يعمل على خادم البريد المُرسِل) وجانب الخادم (والذي يعمل على خادم البريد المُستقبِل). يعمل كلٌّ من جانبي الخادم والزبون لبروتوكول SMTP على كل خادم بريد،

وعندما يرسل خادم البريد رسالة إلى خادمات البريد الأخرى، فإنه يتعامل معها كزبون SMTP، وعندما يستلم خادم البريد رسالة من خادمات البريد الأخرى فإنه يتعامل معها كخادم SMTP.

2-4-1 بروتوكول نقل البريد البسيط (SMTP)

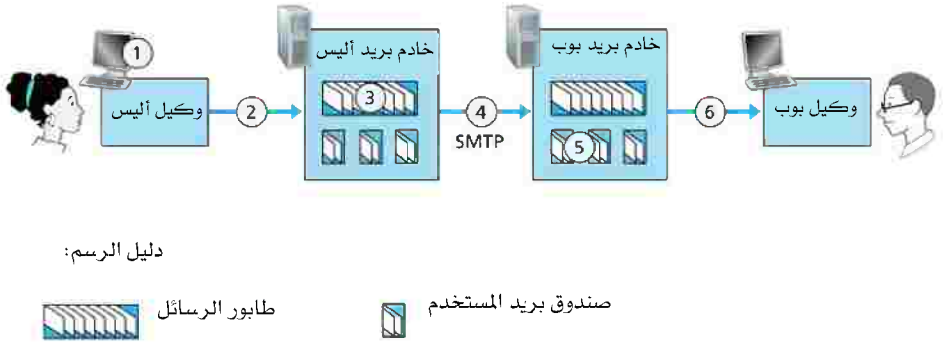
يُشكِّل بروتوكول SMTP والمُعرّف في طلب التعليقات RFC 2821 قلب بريد الإنترنت الإلكتروني. وكما ذكرنا سابقاً ينقل SMTP الرسائل من خادمات البريد المُرسِلة إلى خادمات البريد المُستقبِلة. ويعتبر SMTP أقدم بكثير من HTTP (يعود RFC الأصلي لـ SMTP إلى عام 1982 وكان SMTP موجوداً قبل ذلك بفترة طويلة). بالرغم من العديد من الميزات الرائعة لبروتوكول SMTP، والتي أدت إلى انتشاره المطلق على الإنترنت، فإنه يعتبر تقنية تراثية تعاني من بعض القيود القديمة. على سبيل المثال يتطلب البروتوكول استخدام صيغة ASCII البسيطة بطول 7 بتات لكتابة الرسالة كلها وليس فقط سطور الترويسة. كان هذا التقييد منطقياً في أوائل الثمانينيات عندما كانت قدرة الإرسال ضئيلة ولم يكن أحد يرسل مع البريد الإلكتروني ملحقات أو صوراً كبيرة، أو تسجيلات صوتية، أو ملفات فيديو. أما اليوم في عصر الوسائط المتعددة يُشكِّل استخدام ASCII بطول 7 بتات بعض المتاعب، حيث يتطلب تكويد بيانات الوسائط المتعددة الثنائية (binary) بصيغة ASCII قبل أن ترسل على SMTP؛ كما يتطلب الأمر استعادة الرسالة الثنائية من رسالة ASCII الواصلة بعد نقلها بواسطة SMTP بتكويد معاكس. تذكّر من الجزء 2-2 أن HTTP لا يتطلب تكويد بيانات الوسائط المتعددة بنمط ASCII قبل نقلها.

لتوضيح طريقة عمل SMTP الأساسية دعنا نتتبع سيناريو شائعاً. افترض أن أليس تريد أن تُرسِل رسالة ASCII بسيطة إلى بوب، ومن ثَمَّ:

1. تستدعي أليس وكيل المُستخدِم للبريد الإلكتروني لديها وتعطي عنوان البريد الإلكتروني لبوب (على سبيل المثال bob@someschool.edu) ثم تُعِد الرسالة وتعطي تعليماتها للوكيل لإرسال الرسالة.

2. يُرسِل وكيل المُستخدِم لدى أليس الرسالة إلى خادم البريد، حيث توضع في طابور الرسائل.
3. يرى جانب زبون SMTP (الذي يعمل على خادم بريد أليس) الرسالةَ في طابور الرسائل، فيفتح توصيلة TCP إلى خادم SMTP الذي يعمل على خادم بريد بوب.
4. بعد الانتهاء من خطوات المصافحة الأولية يُرسل زبون SMTP رسالة أليس إلى توصيلة TCP.
5. يستلم جانب خادم SMTP على خادم بريد بوب الرسالة، ثم يضعها في صندوق بريد بوب.
6. يستدعي بوب وكيل المُستخدِم لديه لقراءة الرسالة عندما يرغب في ذلك.

يوضح الشكل 2-17 ملخصاً لهذا السيناريو.



الشكل 2-17 إرسال رسالة بريد إلكتروني من أليس إلى بوب.

من المهم ملاحظة أن SMTP لا يستخدم عادةً خادمات البريد الوسيطة لإرسال البريد، حتى عندما يقع خادما البريد في طرفين متقابلين من العالم. فإذا كان خادم أليس في هونج كونج وخادم بوب في سانت لويس، فإن توصيلة TCP تُشكِّل توصيلةً مباشرةً بين هونج كونج وخادمات سانت لويس. وعلى وجه الخصوص إذا تعطل خادم بريد بوب تبقى الرسالة في خادم بريد أليس بانتظار محاولة إرسال جديدة ولا توضع الرسالة في بعض خادمات البريد الوسيطة.

دعنا الآن نُلقي نظرةً أكثر تفحصاً على كيفية نقل SMTP لرسالة من خادم البريد المُرسِل إلى خادم البريد المُستقبِل. سوف نلاحظ الشبه الكبير بين بروتوكول SMTP والبروتوكولات التي تُستعمَل للتفاعل الإنساني وجهاً لوجه. أولاً: يطلب زبون SMTP (والذي يعمل على مضيف خادم بريد الإرسال) من TCP إنشاء توصيلة إلى المنفذ رقم 25 في خادم SMTP (الذي يعمل على مضيف خادم بريد الاستقبال). وإذا كان خادم الاستلام المطلوب لا يعمل، يعاود الزبون المحاولة مرة أخرى لاحقاً. بمجرد إنشاء تلك التوصيلة يقوم الخادم والزبون بخطوات المصافحة (handshaking) المطلوبة في طبقة التطبيقات. تماماً كما يفعل البشر غالباً عند تقديم أنفسهم قبل تبادل المعلومات من واحد إلى آخر، يقدّم زبائن وخادمات SMTP أنفسهم قبل تبادل المعلومات. أثناء مرحلة المصافحة تلك يُشير زبون SMTP إلى عنوان البريد الإلكتروني للمُرسِل (الشخص الذي أنشأ الرسالة) وعنوان البريد الإلكتروني للمُستقبِل. وبمجرد تقديم زبون وخادم SMTP أنفسهما إلى بعضهما البعض يُرسِل الزبون الرسالة. يمكن أن يعتمد SMTP على خدمة نقل البيانات الموثوق لبروتوكول TCP لتوصيل الرسالة إلى الخادم بدون أخطاء. ثم يكرر الزبون تلك العملية على نفس توصيلة TCP إذا كان لديه رسائل أخرى يود إرسالها إلى الخادم؛ وإلا فإنه يأمر TCP بإغلاق التوصيلة.

دعنا نتناول مثالاً للرسائل المتبادلة بين زبون SMTP (C) وخادم SMTP (S). اسم مضيف الزبون هو crepes.fr واسم مضيف الخادم هو hamburger.edu. السطور التي تبدأ بـ C هي بالضبط السطور التي يرسلها الزبون إلى مقبس TCP والسطور التي تبدأ بـ S هي بالضبط السطور التي يرسلها الخادم إلى مقبس TCP. يبدأ الحوار التالي بمجرد إنشاء توصيلة TCP:

```
S: 220 hamburger.edu
C: HELO crepes.fr
S: 250 Hello crepes.fr, pleased to meet you
C: MAIL FROM: <alice@crepes.fr>
S: 250 alice@crepes.fr ... Sender ok
C: RCPT TO: <bob@hamburger.edu>
S: 250 bob@hamburger.edu ... Recipient ok
C: DATA
S: 354 Enter mail, end with "." on a line by itself
C: Do you like ketchup?
C: How about pickles?
C: .
S: 250 Message accepted for delivery
C: QUIT
S: 221 hamburger.edu closing connection
```

في هذا المثال يرسل الزبون رسالة " Do you like ketchup? How about pickles؟" من خادم البريد crepes.fr إلى خادم البريد hamburger.edu. وكجزء من الحوار يصدر الزبون خمسة أوامر: HELO (اختصار HELLO)، و MAIL FROM، وRCPT TO، وQUIT، وهي أوامر واضحة المعنى. يرسل الزبون إلى الخادم أيضاً سطراً يتكون من نقطة واحدة تُشير إلى نهاية الرسالة. (في مفردات ASCII تنتهي كل رسالة بـ CRLF، حيث تعني CR أول السطر (Carriage Return) وتعني LF تغذية سطر جديد (Line Feed)). يُصدِر الخادم رداً لكل أمر ومع كل رد رمز رد وبعض التفسير باللغة الإنجليزية (اختياري). نضيف هنا أن SMTP يستخدم توصيلات دائمة: فإذا كان لدى خادم البريد المُرسِل عدة رسائل للإرسال إلى نفس خادم البريد المُستقبل، فإنه يمكنه إرسالها كلها على نفس توصيلة TCP. ومع كل رسالة يبدأ الزبون العملية بـ

MAIL FROM: crepes.fr

ويحدد نهاية الرسالة بنقطة على سطر مستقل، ويصدر أمر QUIT فقط بعد الانتهاء من بث كل الرسائل.

ننصحك أن تجرب بنفسك استخدام Telnet لإجراء حوار مباشر مع خادم SMTP. ولعمل ذلك أدخل الأمر

telnet serverName 25

حيث يُمثِل serverName اسم خادم البريد المحلي. وعند قيامك بذلك فإنك ببساطة تُنشئ توصيلة TCP بين مضيفك وخادم البريد المحلي على شبكتك. بعد كتابة هذا السطر يجب أن تتلقى فوراً الرد 220 من الخادم. بعد ذلك يمكنك إصدار أوامر SMTP التالية في الأوقات الملاءمة:

HELO, MAIL FROM, RCPT TO, DATA, CRLF.CRLF, QUIT

نوصي القارئ أيضاً بحل تمرين البرمجة الثانى الموجود في نهاية هذا الفصل، وفيه سيتم بناء وكيل مُستخدِم بسيط يطبّق جانب الزبون من بروتوكول SMTP، ويمكن عن طريقه إرسال رسالة بريد إلكتروني إلى مُستقبل اعتباطي عن طريق خادم البريد المحلي.

تاريخ حالة (Case History)

بريد الهوتميل (Hotmail)

في ديسمبر/كانون الأول عام 1995 قام صابر باتيا (Sabeer Bhatia) وجاك سميث (Jack Smith) بزيارة شركة درابر فيشر جيرفيستون (Draper Fisher Jurveston) للاستثمار في مجال الإنترنت واقترحا تطوير نظام لتوفير البريد الإلكتروني مجاني من خلال الويب. كانت الفكرة هي إعطاء حساب بريد إلكتروني مجاني لأي شخص يطلبه، وجعل الحسابات بحيث يسهل الوصول إليها عن طريق الويب. فمن خلال هذا الحساب يستطيع الشخص الموصّل بالويب (مثلاً من مكتبة المدرسة أو مركز المجتمع) أن يقرأ ويرسل رسائل البريد الإلكتروني. وعلاوة على ذلك يسمح البريد الإلكتروني من خلال الويب بقابلية حركة عظيمة إلى مشتركيه. في مقابل 15٪ من الشركة قام جيرفيستون بتمويل باتيا وسميث لتأسيس شركة جديدة أطلقوا عليها الهوتميل (Hotmail) . وتمكن باتيا وسميث مع ثلاثة موظفين دائمين آخرين وحوالي 12 إلى 14 موظفاً يعملون بدوام جزئي (ولهم نسب في أسهم الشركة) من تطوير وإطلاق الخدمة في يوليو/تموز عام 1996. خلال شهر بعد إطلاق الخدمة أصبح عدد المشتركين 100 ألف مشترك. واستمر عدد المشتركين في النمو بسرعة مع عروض إعلانات الدعاية التى تظهر أثناء قراءة البريد الإلكتروني. في ديسمبر/كانون الأول عام 1997 – أي بعد أقل من 18 شهراً من إطلاق الخدمة – أصبح عدد مشتركي الهوتميل أكثر من 12 مليون مشترك عندما اشترتها شركة مايكروسوفت مقابل 400 مليون دولار.

يُنسَب نجاح الهوتميل في أغلب الأحيان إلى "سابقة ظهوره" وإلى "تسويق فيروسي" (viral marketing) متأصّل. فقد كانت شركة الهوتميل الأولى من نوعها التي وفرّت خدمة البريد الإلكتروني من خلال الويب. بالطبع قلّدت شركات أخرى فكرة الهوتميل لكن ظل الهوتميل متقدّماً بستّة شهور عليها. يتحقق الشرط الأول لنجاح مشروع بامتلاك فكرة أصلية وتطويرها بسرعة وبسرية تامة. أما الشرط الثاني فيقال: إنّ خدمةً أو منتجاً لديه تسويق فيروسي إذا سوّق المنتج نفسه. يُعتبر البريد الإلكتروني مثالاً كلاسيكياً لخدمة ذات تسويق فيروسي حيث يرسل المُرسِل رسالة إلى واحد أو أكثر من المُستقبلين، ومن ثم يصبح كل المُستقبلين على علم بالخدمة. برهن الهوتميل أن تحقق هذين الشرطين يمكن أن يقود إلى تطبيق ناجح. وربما سيكون بعض الطلاب الذين يقرأون هذا الكتاب من بين رجال الأعمال الجدد الذين يتخيلون ويطوّرون خدمات للإنترنت تحقق هذين الشرطين.

2-4-2 مقارنة مع بروتوكول HTTP

دعنا الآن نعقد مقارنة سريعة بين SMTP وHTTP. يُستخدَم كلا البروتوكولين لنقل الملفات من مضيف إلى آخر: ينقل HTTP الملفات (تسمى أيضاً كائنات) من خادم الويب إلى زبون الويب (المتصفّح)، بينما ينقل SMTP الملفات (أي رسائل البريد الإلكتروني) من خادم بريد إلى خادم بريد آخر. وعند نقل تلك الملفات يستخدم كلٌّ من HTTP الدائم وSMTP توصيلات دائمة. وهكذا فللبروتوكولين خصائص مشتركة. ومع ذلك فهناك أيضاً اختلافات هامة بينهما. أولاً: يُعتبر HTTP بشكلٍ أساسي بروتوكول سحب (pull protocol) – حيث يُحمِّل شخصٌ ما المعلومات على خادم الويب ويستعمل المُستخدِمون HTTP لسحب المعلومات من الخادم حسب رغبتهم. وبشكلٍ خاص فإن توصيلة TCP يتم إنشاؤها بواسطة الجهاز الذي يريد استلام الملف. في المقابل يُعتبر SMTP بروتوكول دفع (push protocol)، حيث يدفع خادم البريد المُرسِل الملف إلى خادم البريد المُستقبل. وبشكلٍ خاص فتوصيلة TCP يتم إنشاؤها بواسطة الجهاز الذي يريد إرسال الملف أو الرسالة.

ثمة اختلاف ثانٍ كنا قد ألمحنا إليه في وقت سابق، وهو أن SMTP يتطلب أن تكون كل الرسالة – بما في ذلك جسم الرسالة – بصيغة ASCII بطول 7 بتات. فإذا كانت الرسالة تتضمن حروفاً غير ذلك (على سبيل المثال حروفاً فرنسية ذات علامات) أو تحتوي على بيانات ثنائية (كملف صورة)، فإنه يتعين تكويد الرسالة بصيغة ASCII بطول 7 بتات، في حين لا يفرض بروتوكول HTTP مثل هذه القيود على البيانات.

هناك اختلاف ثالث هام يتعلق بكيفية مناولة وثيقة تتضمن نصوصاً وصوراً (وربما مع بعض مواد الوسائط المتعددة الأخرى). كما رأينا في الجزء 2-2 يقوم HTTP بتغليف كل كائن في رسالة رد HTTP الخاصة به، بينما يضع بريد الإنترنت كل كائنات الرسالة في رسالة واحدة، كما سنرى بتفصيل أكثر لاحقاً.

3-4-2 صيغ رسائل البريد والامتداد MIME

عندما تكتب أليس رسالة بريد عادية إلى بوب قد تضمنها معلومات إضافية في أعلى الرسالة، كعنوان بوب وعنوان أليس للبريد الراجع والتاريخ. وبنفس الطريقة فعند إرسال رسالة بريد إلكتروني من شخص لآخر، فسوف يسبق جسم الرسالة نفسها مجموعة سطور تحتوي على المعلومات الإضافية (يحتوي طلب التعليقات RFC 822 على توصيف لتلك المعلومات الإضافية). يفصل بين تلك السطور وجسم الرسالة سطر فارغ (أي CRLF). كما هو الحال مع HTTP، يحتوي كل سطر ترويسة نصاً مقروءاً يتكون من كلمة دليلية تتبعها النقطتان (:) تتبعهما قيمة. بعض الكلمات الدليلية مطلوبة وبعضها اختياري. فكل ترويسة يجب أن تتضمن سطر From وسطر To، وقد تحتوي على Subject، بالإضافة إلى سطور الترويسة الاختيارية الأخرى. من المهم ملاحظة أن هذه السطور مختلفة عن أوامر SMTP التي درسناها في الجزء 2-4-1 (رغم وجود بعض الكلمات المشتركة مثل 'From' و'To'). كانت الأوامر في ذلك الجزء تُشكِّل جزءًا من بروتوكول المصافحة لـ SMTP، في حين تُشكِّل سطور الترويسة التي ذكرناها هنا جزءاً من رسالة البريد نفسها.

وكمثال تبدو سطور الترويسة لرسالة كالتالي:

From: alice@crepes.fr
To: bob@hamburger.edu
Subject: Searching for the meaning of life.

يلى سطور الترويسة سطر فارغ ثم جسم الرسالة (بصيغة ASCII). الآن جرّب استخدام Telenet لإرسال رسالة إلى خادم البريد تحتوي على بعض سطور الترويسة بما في ذلك Subject، ولعمل ذلك استخدم الأمر

telnet serverName 25

كما سبق ذكره في الجزء 2-4-1.

امتدادات MIME للبيانات بغير صيغة ASCII

بينما تعد سطور ترويسة الرسائل التي تم توصيفها في طلب التعليقات RFC 822 كافية لإرسال نص ASCII عادي، فإنها ليست غنية بدرجة كافية لرسائل الوسائط المتعددة (كالرسائل التي تتضمن صوراً وتسجيلات صوت وفيديو) أو لنقل صيغ النص غير ASCII (كحروف اللغات الأخرى غير الإنجليزية). لإرسال المحتويات غير نصوص ASCII يجب على وكيل المُستخدِم المُرسِل استخدام سطور ترويسة إضافية في الرسالة. تم توصيف تلك السطور الإضافية في RFC 2045 وRFC 2046، ويطلق عليها "امتدادات بريد الإنترنت متعددة الأغراض" Multipurpose Internet Mail Extensions والمعروفة بالاختصار MIME وهي امتداد لطلب التعليقات RFC 822.

من بين سطور MIME الإضافية الهامة لدعم رسائل الوسائط المتعددة السطران:

Content-Type:
Content-Transfer-Encoding:

يسمح Content-Type لوكيل المُستخدِم المُستقبِل باتخاذ الإجراء المناسب لمحتوى الرسالة. على سبيل المثال بتحديد أن جسم الرسالة يحتوي على صورة JPEG يمكن لوكيل المُستخدِم المُستقبِل توجيه جسم الرسالة إلى برنامج لاسترجاع الصور المضغوطة بطريقة JPEG. ولفهم الحاجة إلى السطر الثاني Content-Transfer-Encoding تذكّر أن رسائل النصوص غير ASCII يجب أن تكوّد بصيغة ASCII لكي يفهمها SMTP، يُنبِّه هذا السطر وكيل المُستخدِم المُستقبِل إلى أن جسم الرسالة يتكون من صيغة ASCII مُكوّدة ويشير إلى طريقة التكويد المُستخدَمة. وهكذا فعندما يستلم وكيل المُستخدِم رسالة تتضمن سطري الترويسة هذين، فإنه يستخدم أولاً قيمة السطر Content-Transfer-Encoding في تحويل جسم الرسالة إلى الشكل الأصلي غير ASCII لها، وبعد ذلك يستخدم السطر Content-Type لتحديد الإجراءات المطلوب القيام بها على جسم الرسالة.

لنلقِ نظرة على مثال محدد. افترض أن أليس تريد إرسال صورة JPEG إلى بوب. ولعمل ذلك تستدعى أليس وكيل المُستخدِم للبريد الإلكتروني لديها، وتحدد عنوان البريد الإلكتروني لبوب، وتحدد موضوع الرسالة، وتدخل الصورة إلى جسم الرسالة. (وعلى حسب وكيل المُستخدِم الذي تستعمله أليس، يمكن أن تدخل الصورة إلى الرسالة كملف مرفق). وعندما تنتهي أليس من إعداد رسالتها، تنقر على "Send" (أي "أرسل")، وعندها يقوم وكيل المُستخدِم لدى أليس بتوليد رسالة MIME يمكن أن تبدو كالتالي:

```
From: alice@crepes.fr
To: bob@hamburger.edu
Subject: Picture of yummy crepe.
MIME-Version: 1.0
Content-Transfer-Encoding: base64
Content-Type: image/jpeg

base64 encoded data .....
.....
.....base64 encoded data
```

نلاحظ من رسالة MIME تلك أن وكيل أليس قام بتكويد صورة JPEG بطريقة base64، وهي أحد أساليب التكويد الموصوفة في MIME [RFC 2045] للتحويل إلى صيغة ASCII بطول 7 بتات والمقبولة لدى بروتوكولSMTP . هناك أسلوب تكويد آخر شائع الاستعمال هو quoted-printable، والذي يستخدم لتحويل رسالة بصيغة ASCII بطول 8 بتات (قد تتضمن حروفاً غير إنجليزية) إلى ASCII بطول 7 بتات.

عندما يقرأ بوب بريده باستعمال وكيل المُستخدِم لديه يقوم وكيل بوب بملاحظة سطر الترويسة

Content-Transfer-Encoding: base64

ومن ثم يشرع في فك التكويد لجسم الرسالة. كما تتضمن الرسالة أيضاً سطر الترويسة

Content-Type: image/jpeg

والذي يُنبِّه وكيل المُستخدِم إلى أن جسم الرسالة يجب أن يكون JPEG بعد إزالة عملية ضغط البيانات (decompression). وأخيراً تتضمن الرسالة السطر

MIME-Version: 1.0

والذي يُشير بالطبع إلى رقم إصدر MIME المُستخدَم. لاحظ أن الرسالة فيما عدا ذلك تتبع معيار RFC 822 لصيغة الرسالة. وبالتحديد بعد سطور ترويسة الرسالة يأتي سطر فارغ يتبعه بعد ذلك جسم الرسالة.

الرسائل المُستلَمة

سنكون مقصرين إذا أغفلنا ذكر نوع آخر من سطور الترويسة التي يتم إدخالها من قِبَل خادم SMTP المُستقبل. فور استلام ذلك الخادم رسالة تحتوي على سطور ترويسة من RFC 822 وMIME يقوم الخادم بإلحاق السطر Received في بداية الرسالة. يحدد هذا السطر اسم خادم SMTP الذي أرسل الرسالة (from)، واسم خادم SMTP الذي استلم الرسالة (by)، ووقت استلام الخادم المُستقبل للرسالة. وهكذا فإن الرسالة التي يتلقاها المُستخدِم عند الوجهة النهائية تأخذ الشكل التالي:

```
Received: from crepes.fr by hamburger.edu; 12 Oct 98 15:27:39 GMT
From: alice@crepes.fr
To: bob@hamburger.edu
Subject: Picture of yummy crepe.
MIME-Version: 1.0
Content-Transfer-Encoding: base64
Content-Type: image/jpeg

base64 encoded data .....
.....
.....base64 encoded data
```

كل من استعمل البريد الإلكتروني تقريباً قد رأى سطر العنوان Received يسبق الرسالة (مع سطور الترويسة الأخرى). (وهذا السطر يرى في أغلب الأحيان مباشرة على الشاشة أو عندما تُرسَل الرسالة إلى طابعة). وربما لاحظت أن رسالة واحدة قد تتضمن أحياناً عدة سطور Received، وذلك لأن الرسالة قد تُرسَل إلى أكثر من خادم SMTP في المسار بين المُرسِل والمُستلِم. على سبيل المثال إذا طلب بوب من خادم بريده الإلكتروني hamburger.edu توجيه كل رسائله إلى sushi.jp، فإن الرسالة التي يقرؤها وكيل المُستخدِم لدى بوب قد تبدأ بشيء مثل:

Received: from hamburger.edu by sushi.jp; 3 Jul 01 15:30.01 GMT
Received: from crepes.fr by hamburger.edu; 3 Jul 01 15:17:39 GMT

وتوفر سطور الترويسة تلك لوكيل المُستخدِم المُستقبِل أثراً لتتبع خادمات SMTP التي زارتها الرسالة في طريقها للوجهة، بالإضافة إلى الوقت الذي تمت فيه تلك الزيارات.

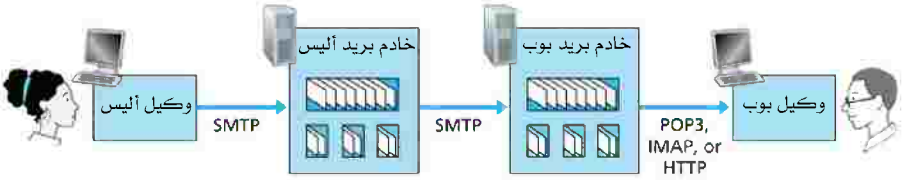
4-4-2 بروتوكولات التوصل لرسائل البريد

عندما يوصل SMTP الرسالة من خادم بريد أليس إلى خادم بريد بوب توضع الرسالة في صندوق بريد بوب. خلال هذه المناقشة افترضنا ضمنياً أن بوب يقرأ بريده بالدخول إلى مضيف الخادم ثم يشغل قارئ البريد على ذلك المضيف. حتى أوائل التسعينيات كانت تلك هي الطريقة القياسية لعمل ذلك، أمّا اليوم فيتبع التوصل للبريد الإلكتروني البنية المعمارية من طراز زبون/خادم – حيث يقرأ المُستخدِم العادي البريد الإلكتروني باستخدام زبون يُنفَّذ على النظام الطرفي لديه، كحاسب شخصي مكتبي، أو حاسب نقّال، أو مساعد رقمي شخصي. وبتنفيذ زبون البريد على الحاسب الشخصي المحلي يتمتع المُستخدِمون بحزمة غنية من الميزات، منها القدرة على مشاهدة رسائل وملحقات الوسائط المتعددة.

افترض أن بوب (المُستقبِل) يُشغِّل وكيل المُستخدِم على حاسبه الشخصي، ومن الطبيعي اعتبار وجود خادم البريد على نفس الحاسب أيضاً. وبهذه النظرة فإن خادم بريد أليس يدير حواراً مباشراً مع حاسب بوب. ولكن هناك مشكلة

تكتنف هذه الطريقة. تذكّر أن خادم البريد يدير صناديق البريد ويُشغِّل جانبى خادم وزبون من بروتوكول SMTP. فإذا كان خادم بريد بوب يستقر على حاسبه الشخصي، يجب أن يبقى ذلك الحاسب دائماً في وضع التشغيل ومتصلاً بالإنترنت لكي يستلم البريد الجديد الذي يمكن أن يصل في أي وقت، وهذا غير عملي للعديد من مُستخدِمي الإنترنت. وبدلاً من ذلك يُشغِّل المُستخدِم عادةً وكيل المُستخدِم فقط على حاسبه الشخصي، لكنه يخزّن صندوق بريده على خادم البريد المشترك والذي يبقى دائماً في وضع تشغيل. ويدار خادم البريد المشترك هذا، والذي يشترك في استعماله المُستخدِمون الآخرون، عادةً من قِبَل موفّر خدمات الإنترنت للمُستخدِم (مثلاً جامعة أو شركة).

دعنا الآن نأخذ في الاعتبار المسار الذي تسلكه رسالة بريد إلكتروني عندما تُرسَل من أليس إلى بوب. عرفنا للتو أنه في وقتٍ ما على طول المسار من الضّروري إيداع رسالة البريد الإلكتروني في خادم بريد بوب. هذا يمكن أن يحدث ببساطة بجعل وكيل المُستخدِم أليس يُرسِل الرسالة مباشرة إلى خادم بريد بوب. ويمكن أن يحدث ذلك باستخدام SMTP (في الحقيقة صُمِّم SMTP لدفع البريد الإلكتروني من مضيف إلى آخر). ومع ذلك فإنه في العادة لا يدير وكيل المُستخدِم المُرسِل حواراً مباشراً مع خادم البريد المُستلم. بدلاً من ذلك وكما هو موضح في الشكل 2-18 يستخدم وكيل المُستخدِم لدى أليس بروتوكول SMTP لدفع رسالة البريد الإلكتروني إلى خادم البريد لديها، ثم يستخدم خادم بريد أليس بروتوكول SMTP (كزبون SMTP) لترحيل رسالة البريد الإلكتروني إلى خادم بريد بوب. لماذا إجراء خطوتين؟ بالدرجة الأولى لأنه بدون الترحيل من خلال خادم بريد أليس ليس لوكيل المُستخدِم لدى أليس أي إمكانية للوصول إلى خادم بريد الوجهة النهائية إذا كان من المتعذر الوصول إليه في الوقت الحالي. وبجعل أليس تُودِع البريد الإلكتروني أولاً في خادم بريدها الخاص، يمكن أن يعاود خادم بريد أليس محاولة إرسال الرسالة إلى خادم بريد بوب مراراً وتكراراً (مثلاً كل 30 دقيقة) لحين أن يصبح خادم بريد بوب شغّالاً. يحدد طلب التعليقات الخاص ببروتوكول SMTP كيفية استخدام أوامر SMTP لترحيل رسالة عبر عدة خادمات SMTP.



الشكل 2-18 بروتوكولات البريد الإلكتروني وكياناتها المتصلة.

لكن ما زالت هناك قطعة واحدة مفقودة من اللغز! كيف يحصل مُستقبِل مثل بوب يُشغّل وكيل المُستخدِم على حاسبه الشخصي على رسائله الموجودة في خادم البريد عند موفّر خدمة الإنترنت له؟ لاحظ أن وكيل المُستخدِم لدى بوب لا يستطيع استخدام SMTP للحصول على تلك الرسائل، لأن الحصول على الرسائل عملية سحب بينما SMTP بروتوكول دفع. يكتمل اللغز بتوفير نظام خاص للوصول للبريد بنقل الرسائل من خادم بريد بوب إلى حاسبه الشخصي. يوجد حالياً عدد من بروتوكولات الوصول للبريد شائعة الاستخدام تشمل الإصدار الثالث لبروتوكول مكتب البريد (Post Office Protocol--Version 3 (POP3))، وبروتوكول الوصول لبريد الإنترنت (Internet Mail Access Protocol (IMAP))، وبروتوكول HTTP.

يلخص الشكل 2-18 البروتوكولات المستخدمة مع بريد الإنترنت: يُستخدَم SMTP لنقل البريد من خادم البريد المُرسِل إلى خادم البريد المُستقبِل؛ ويُستخدَم SMTP أيضاً لنقل البريد من وكيل المُستخدِم المُرسِل إلى خادم البريد المُرسِل. ويُستخدَم بروتوكول الوصول للبريد مثل POP3 لنقل البريد من خادم البريد المُستقبِل إلى وكيل المُستخدِم المُستقبِل.

بروتوكول POP3

يُعتبر POP3 بروتوكولاً بسيطاً جداً للوصول للبريد، وهو مُعرَّف في طلب التعليقات RFC 1939 (وهو مستند قصير وسهل القراءة). ولأن البروتوكول بسيط

جداً فوظيفته أيضاً محدودة. يبدأ POP3 العمل عندما يفتح وكيل المُستخدِم (الزبون) توصيلة TCP إلى خادم البريد (الخادم) على منفذ رقم 110. بعد ذلك يمر POP3 بثلاث مراحل: التحقق (authorization)، وتنفيذ العملية (transaction)، والتحديث (update). في المرحلة الأولى (التحقق) يُرسِل وكيل المُستخدِم اسم المُستخدِم وكلمة السّر (بشكلٍ مقروء) للتحقق من شخصية المُستخدِم والترخيص له. وأثناء المرحلة الثانية (تنفيذ العملية) يسترجع وكيل المُستخدِم الرسائل؛ يمكن لوكيل المُستخدِم أثناء هذه المرحلة أيضاً أن يُؤشِّر على رسائل للحذف، أو يلغي علامات الحذف، أو يحصل على إحصائيات عن بريده. المرحلة الثالثة (التحديث) وتحدث بعد أن يُصدِر الزبون الأمر QUIT ليُنهِي جلسة POP3، ومنها يحذف خادم البريد الرسائل التي أُشِّرت للحذف.

في أثناء المرحلة الثانية (تنفيذ العملية) يُصدِر وكيل المُستخدِم الأوامر ويرد الخادم على كل أمر. هناك إجابتان محتملتان: "+OK" (يتبعها أحياناً بيانات من الخادم إلى الزبون) ويستخدمها الخادم للإشارة إلى سلامة الأمر السابق على ما يرام؛ و"-ERR" ويستخدمها الخادم للإشارة إلى حدوث خطأ عند تنفيذ الأمر السابق.

تتضمن مرحلة التحقق أمرين رئيسين:

user <*username*>
pass <*password*>

ونقترح أن تستخدم Telnet للاتصال مباشرة مع خادم POP3 على منفذ رقم 110، وأن تجري الحوار التالي على افترض أن mailServer هو اسم خادم البريد:

```
telnet mailServer 110
+OK POP3 server ready
user bob
+OK
pass hungry
+OK user successfully logged on
```

إذا أخطأت في كتابة أمر فسيجيب خادم POP3 برسالة خطأ "-ERR".

الآن لنلقِ نظرة على مرحلة تنفيذ العملية. في أغلب الأحيان يمكن تجهيز وكيل المُستخدِم الذي يستعمل POP3 لكي "يُنزّل ويحذف" (download and delete) أو "يُنزّل ويحتفظ" (download and keep). تعتمد سلسلة الأوامر التي تصدرمن قِبَل وكيل مُستخدِم POP3 على نمط التشغيل المستخدم من بين هذين النمطين. في النمط "download and delete" سوف يُصدِر وكيل المُستخدِم الأوامر:

list, retr, dele

وكمثال افترض أن المُستخدِم لديه رسالتان في صندوق بريده وأجرى الحوار التالي، حيث ترمز C لوكيل المُستخدِم (الزبون) وS لخادم البريد (الخادم):

```
C: list
S: 1 498
S: 2 912
S:.
C: retr 1
S: (blah blah ...
S:  .................
S: ..... blah)
S:.
C: dele 1
C: retr 2
S: (blah blah ...
S:  .................
S: .....blah)
S:.
C: dele 2
C: quit
S: +OK POP3 server signing off
```

في هذا الحوار يطلب وكيل المُستخدِم أولاً من خادم البريد عرض حجم كلٍّ من الرسائل المُخزَّنة، ثم يسترجع ويحذف كل رسالة من الخادم. لاحظ أنه بعد مرحلة التوثيق يستعمل وكيل المُستخدِم أربعة أوامر فقط:

list, retr, dele, quit

وصيغة هذه الأوامر مُعرَّفة في طلب التعليقات RFC 1939. بعد تنفيذ الأمر quit يدخل خادم POP3 مرحلة التحديث ويحذف الرسائل رقم 1 و2 من صندوق البريد.

من مشاكل نمط "download-and-delete" هذا أن المُستقبِل بوب قد يكون متجولاً ويريد الوصول إلى رسائل بريده من عدة أجهزة، على سبيل المثال من حاسب مكتبه وحاسب بيته وحاسبه النقّال. هذا النمط يقسم رسائل بريد بوب على هذه الأجهزة الثلاثة. وبالتحديد إذا قرأ بوب رسالة على حاسب مكتبه فلن يكون قادراً على إعادة قراءتها من حاسبه النقّال في البيت في وقت لاحق. أما في نمط "download-and-keep" فيترك وكيل المُستخدِم الرسائل على خادم البريد بعد تحميلها. في هذه الحالة يمكن أن يعيد بوب قراءة الرسائل من الأجهزة المختلفة؛ ويمكن أن يستعرض رسالة في مكتبه ثم يستعرضها ثانيةً في وقت لاحق من البيت.

يحتفظ خادم POP3 ببعض المعلومات عن الحالة أثناء نفس الجلسة بين وكيل المُستخدِم وخادم البريد؛ وبصفة خاصة يتتبع أي رسائل للمُستخدِم قد أُشّرت للحذف. لكن لا يحتفظ خادم POP3 بمعلومات عن الحالة من جلسة لأخرى مما يبسط كثيراً عملية تطوير برنامج الخادم.

بروتوكول IMAP

في بروتوكول POP3 للوصول للبريد يمكن أن يُنشئ بوب مجلدات للبريد (mail folders) على حاسبه الشخصي ويخزن رسائله فيها. كما يمكنه بعد ذلك حذف الرسائل أو نقلها بين المجلدات أو البحث عن رسائل بعينها (باسم المُرسِل أو الموضوع). لكن هذا الأسلوب (أي وجود المجلدات والرسائل في الحاسب المحلي) يُشكِّل مشكلة للمُستخدِم المتنقل الذي يُفضِّل إبقاء المجلدات والرسائل على الخادم البعيد ليتمكن من الوصول إليها من أي حاسب آخر. هذا الأمر غير ممكن مع بروتوكول POP3، فهو لا يوفر أية وسيلة للمُستخدِم لإنشاء المجلدات البعيدة وتخزين الرسائل فيها.

لحل هذه المشكلة وغيرها من المشاكل، تم استحداث بروتوكول IMAP والمُعرَّف في RFC 3501، وهو بروتوكول للوصول للبريد مثل POP3، غير أنه يتميز عنه بميزاتٍ عديدة ولكنه أيضاً أكثر تعقيداً. يقرن خادم IMAP كل رسالة بمجلد، وعندما تصل رسالة إلى الخادم لأول مرة ترتبط بمجلد صندوق الوارد (INBOX). عندئذ يمكن للمُستخدِم نقل الرسالة إلى مجلد جديد تم إنشاؤه من قبل، أو قراءة الرسالة، أو حذفها، وهكذا. يوفر بروتوكول IMAP أوامر تسمح للمُستخدِمين بإنشاء المجلدات ونقل الرسائل من مجلد لآخر. كما يوفر الأوامر التي تُمكّن المُستخدِمين من البحث في المجلدات البعيدة عن رسائل تطابق معايير معينة. لاحظ أنه بخلاف POP3 يحتفظ خادم IMAP بمعلومات عن حالة المُستخدِم عبر جلسات IMAP كأسماء المجلدات والرسائل بكلٍّ منها.

من الميزات الهامة الأخرى لبروتوكول IMAP أنه يتضمن الأوامر التي تسمح لوكيل المُستخدِم بالحصول على مكوّنات الرسائل. على سبيل المثال يمكن أن يحصل وكيل المُستخدِم على سطور الترويسة فقط من الرسالة أو على جزء واحد فقط من رسالة MIME متعددة الأجزاء. هذه الميزة مفيدة عندما تكون وصلة الاتصال ذات سعة إرسال منخفضة بين وكيل المُستخدِم وخادم البريد (على سبيل المثال وصلة مودم بطيئة السرعة). في هذه الحالة قد لا يريد المُستخدِم تنزيل كل الرسائل في صندوق بريده ليتفادى الرسائل الطويلة خصوصاً التي قد تحتوي مثلاً على تسجيلات صوتية أو لقطات فيديو. يمكن أن تقرأ كل شيء عن IMAP من على موقعه الرسمي على الويب [IMAP 2007].

البريد الإلكتروني من خلال الويب (Web-Based E-mail)

اليوم يرسل العديد من المُستخدِمين بشكلٍ متزايد رسائل البريد الإلكتروني ويستعرضونه من خلال متصفّحات الويب. قدّم الهوتميل (Hotmail) خدمة بريد الويب في منتصف التسعينيات، كما يتوفر البريد الإلكتروني على الويب الآن أيضاً من خلال ياهوو (Yahoo) وجوجل (Google) بالإضافة إلى كل جامعة وشركة كبرى تقريباً. وبهذه الخدمة يكون وكيل المُستخدِم متصفّح ويب

عادي، ويتصل المُستخدِم بصندوق بريده البعيد عن طريق HTTP. وعندما يريد مُستخدِم مثل بوب الوصول لرسالة في صندوق بريده، تُرسَل الرسالة من خادم بريد بوب إلى متصفّح بوب بواسطة HTTP بدلاً من POP3 أو IMAP. عندما يريد مُرسِل مثل أليس إرسال رسالة بريد إلكتروني، تُرسَل الرسالة من متصفّحها إلى خادم البريد على HTTP بدلاً من SMTP. ومع ذلك فإن خادم بريد أليس لا يزال يُرسِل رسائل إلى خادمات البريد الأخرى ويتلقّى الرسائل منها باستخدام SMTP.

5-2 خدمة دليل الإنترنت لأسماء النطاقات (DNS)

يمكن أن نُميَّز – نحن البشر – بعدة طرق. على سبيل المثال يمكن أن نُميَّز بالأسماء في شهادات الميلاد، أو بأرقام الضمان الاجتماعي، أو أرقام رخص القيادة. ورغم أن كل أشكال التعريف هذه يمكن أن تُستعمَل لتمييز الناس، قد يكون أحد الطرق أكثر ملاءمة من الآخر ضمن سياق معين. على سبيل المثال تفضل الحاسبات في وكالة IRS (وكالة تحصيل الضرائب المشهورة في الولايات المتّحدة) استخدام أرقام الضمان الاجتماعي ثابتة الطول بدلاً من أسماء شهادات الميلاد، في حين يُفضِّل الناس استخدام أسماء شهادات الميلاد لسهولة تذكرها بدلاً من أرقام الضمان الاجتماعي.

ومثلما يمكننا تمييز البشر بعدة طرق، يمكننا أن نُميِّز مضيفات الإنترنت بعدة طرق. أحد طرق التعريف هو اسم المضيف (hostname) مثل: cnn.com، www.yahoo.com، cis.poly.edu، gala.cs.umass.edu. وهي سهلة التذكر ولذا يفضلها الناس. ومع ذلك فإن أسماء المضيفات تعطي معلومات قليلة عن مواقع المضيفات على شبكة الإنترنت. مثلاً اسم المضيف www.eurecom.fr والذي ينتهي برمز البلد fr، يخبرنا بأنه من المحتمل أن يكون المضيف في فرنسا، لكن لا يمكننا القول أكثر من ذلك. وعلاوة على ذلك يمكن أن تكون أسماء المضيفات بأطوال مختلفة (حروف وأرقام) مما يُصعِّب معالجتها بالموجّهات. لهذه الأسباب تُميَّز المضيفات أيضاً بما يسمّى عناوين IP.

سوف نناقش عناوين IP ببعض التفصيل في الفصل الرابع، ولكن من المفيد تناولها ببضع كلمات قصيرة الآن. يتكون عنوان IP من أربعة بايتات، وله تركيب هرمي ثابت. يأخذ عنوان IP شكلاً كالمثال التالي: 121.7.106.83، حيث يفصل كل بايت عن الآخر بنقطة ويكتب بطريقة عشرية من 0 إلى 255. عنوان IP ذو تركيب هرمي لأنه عندما نقرأ العنوان من اليسار إلى اليمين، نحصل على معلومات معينة أكثر فأكثر حول مكان المضيف في الإنترنت (أي ضمن أية شبكة في شبكة الشبكات العالمية). بنفس الطريقة عندما نقرأ العنوان البريدي من أسفل إلى أعلى نحصل على معلومات معينة أكثر فأكثر حول مكان المُرسَل إليه.

1-5-2 الخدمات المتوفرة من قِبَل DNS

رأينا للتو أن هناك طريقتين لتمييز المضيف: اسم المضيف وعنوان IP. يفضل الناس اسم المضيف لسهولة تذكره، بينما تفضل أجهزة الموجّهات عناوين IP ثابتة الطول وذات التركيب الهرمي. ولكي نوفق بين هذه التفضيلات المختلفة، نحتاج إلى خدمة الدليل التي تترجم أسماء المضيفات إلى عناوين IP، وتلك هي المهمة الرئيسة لنظام أسماء النطاقات في الإنترنت ((Domain Name System (DNS)). فنظام DNS هو (1) قاعدة بيانات موزّعة مُنفّذة في خادمات DNS ذات ترتيب هرمي، (2) أحد بروتوكولات طبقة التطبيقات والذي يسمح للمضيفات بالبحث في قاعدة البيانات الموزّعة تلك. غالباً ما تعمل خادمات DNS على أجهزة يونيكس بنظام ((Berkeley Internet Name Domain (BIND)). ويستخدم بروتوكول DNS بروتوكول UDP على منفذ 53.

في الغالب يُستخدَم DNS عن طريق بروتوكولات طبقة البرامج الأخرى بما فيها HTTP، وSMTP، وFTP لترجمة أسماء المضيفات إلى عناوين IP. كمثال لنأخذ في الاعتبار ما يحدث عندما يقوم متصفّح يعمل على مضيف ما بطلب صفحة الويب:

www.someschool.edu/index.html

لكي يتمكن المضيف من إرسال رسالة طلب HTTP إلى خادم الويب www.someschool.edu يجب أن يحصل أولاً على عنوان IP لهذا الخادم. ويتم ذلك كالتالي:

1. يشغّل نفس جهاز المُستخدِم برنامج الزبون لتطبيق DNS.
2. يقتبس المتصفّح اسم المضيف www.someschool.edu من عنوان URL ويرسله إلى برنامج الزبون لتطبيق DNS.
3. يرسل برنامج زبون DNS استفساراً يحتوي على اسم المضيف إلى خادم DNS.
4. يستقبل زبون DNS في النهاية إجابة تتضمن عنوان IP لاسم المضيف.
5. عندما يستلم المتصفّح عنوان IP من DNS، يمكنه أن يُنشئ توصيلة TCP مع عملية خادم HTTP والموجودة على منفذ رقم 80 في ذلك العنوان.

نرى من هذا المثال أن DNS يتسبب في زمن تأخير إضافي – كبير أحياناً – لتطبيقات الإنترنت التي تستخدمه. لحسن الحظ – كما سنناقش فيما بعد – يُحفَظ عنوان IP المطلوب في أغلب الأحيان في الذاكرة المخبأة لخادم DNS قريب، مما يساعد في تخفيض حركة مرور بيانات DNS على الشبكة وكذلك تخفيض زمن تأخير DNS في المتوسط.

يوفر DNS بضع خدمات أخرى مهمة بالإضافة لترجمة أسماء المضيفات إلى عناوين IP:

- أسماء المضيف البديلة (Host aliases): يمكن أن يحمل مضيف له اسم صعب واحداً أو أكثر من الأسماء البديلة. على سبيل المثال يمكن أن يأخذ المضيف relay1.west-coast.enterprise.com اسمين أكثر شهرة مثل

enterprise.com
www.enterprise.com

في هذه الحالة يقال إن اسم المضيف

relay1.west-coast.enterprise.com

هو الاسم القانوني (أو الرسمي) (canonical name). وتُعتبر أسماء المضيفات البديلة أكثر سهولة للتذكر من أسماء المضيفات القانونية. ويمكن أن يستخدم تطبيق DNS للحصول على اسم المضيف القانوني وعنوان IP المناظرين لعنوان بديل للمضيف.

- أسماء خادم البريد البديلة (Mail server aliasing): لأسباب واضحة من المرغوب فيه جداً أن تكون عناوين البريد الإلكتروني سهلة التذكر. على سبيل المثال إذا كان بوب لديه حساب هوتميل، فإن عنوان بريده الإلكتروني قد يكون بسيطاً مثل bob@hotmail.com. ومع ذلك فإن اسم مضيف خادم البريد الهوتميل أكثر تعقيداً وأصعب في تذكره من الاسم البسيط hotmail.com (فقد يكون اسم المضيف القانوني مثلاً relayl.west-coast.hotmail.com). يمكن أن يستخدم DNS من قبل تطبيق بريدي للحصول على اسم المضيف القانوني لاسم مضيف بديل بالإضافة إلى عنوان IP للمضيف. في الحقيقة يسمح سجل MX (كما سترى فيما بعد) أن يكون لخادم البريد وخادم الويب لشركةٍ ما أسماء مضيفات بديلة متطابقة. على سبيل المثال يمكن أن يسمى كلٌّ من خادم الويب وخادم البريد للشركة enterprise.com.
- توازن الأحمال (Load balancing): يُستخدَم DNS أيضاً للقيام بتوزيع الأحمال بين الخادمات المكررة بشكلٍ متوازن، كخادمات الويب المكررة (replicated web servers). يتم تكرار خدمات الويب للمواقع ذات الأحمال العالية (التي يكثر عليها الطلب) مثل enn.com على خادمات متعددة، ويعمل كلٌّ منها على نظام طرفي مستقل وله عنوان IP مختلف. ولهذه الخادمات مجموعة من عناوين IP المرتبطة باسم مضيف قانوني واحد. تحتوي قاعدة بيانات DNS على مجموعة عناوين IP تلك. وعندما ترسل زبائن DNS استفساراً عن اسم مرتبط بتلك المجموعة من العناوين، يرد الخادم بمجموعة عناوين IP كاملة ولكن بترتيب مختلف للعناوين في كل مرة. لأن الزبون عادة ما يرسل رسالة طلب HTTP إلى

عنوان IP المدرج أولاً في المجموعة، فإن إعادة ترتيب DNS للعناوين يُوزِّع حركة البيانات بين الخادمات المكررة. يستخدم أيضاً تدوير DNS للعناوين في البريد الإلكتروني لكي يتسنى استخدام خادمات بريد متعددة لنفس الاسم البديل. ومؤخراً استخدمت شركات توزيع المحتوى (مثل [Akamai 2007]) بروتوكول DNS بطرق أكثر تطوراً للقيام بتوزيع محتوى الويب (انظر الفصل السابع).

تم توصيف بروتوكول DNS في RFC 1034 وRFC 1035 وعُدِّل في عدة طلبات تعليقات أخرى. سنتناول هنا باختصار السّمات الرئيسة فقط لطريقة عمله، ونحيل القارئ المهتم لبعض المصادر الإضافية كبعض طلبات التعليقات الأخرى والكتاب [Abitz 1993] والأبحاث [Mockapetris 1988; Mockapetris 2005].

المبادئ في الواقع العملي (Principles in Practice)

بروتوكول DNS وأداء وظائف حرجة للشبكة عن طريق بنية زبون/خادم

يعتبر بروتوكول DNS أحد بروتوكولات طبقة التطبيقات مثله في ذلك مثل HTTP وFTP وSMTP، وذلك للأسباب التالية: (1) أنه يستخدم بين أنظمة طرفية تتصل فيما بينها باستخدام أسلوب زبون/خادم، و(2) أنه يعتمد على بروتوكولات النقل التحتية لتبادل رسائل DNS بين الأنظمة الطرفية المتصلة. ولكن من وجهة نظر أخرى يختلف دور DNS تماماً عن الويب، ونقل الملفات، وتطبيقات البريد الإلكتروني، فهو ليس تطبيقاً يتعامل معه المُستخدِم مباشرة، ولكن DNS يوفر خدمة رئيسية للإنترنت – ألا وهي خدمة ترجمة أسماء المضيفات إلى عناوين IP المناظرة لتطبيقات المُستخدِم والبرامج الأخرى في الإنترنت. لاحظنا في الجزء 1-2 أن معظم التعقيد في البنية المعمارية للإنترنت يوجد على "حواف" الشبكة. ويعتبر DNS الذي يقوم بعملية ترجمة الاسم إلى العنوان والذي يستخدم زبائن وخادمات موجودة على حافة الشبكة مثالاً آخر لفلسفة التصميم تلك.

2-5-2 نظرة عامة على كيفية عمل DNS

لنلقِ الآن نظرة عامة مبسطة على كيفية عمل DNS. ستُركِّز مناقشتنا على خدمة ترجمة اسم المضيف إلى عنوان IP. افترض أن تطبيقاً ما (مثل متصفّح الويب أو قارئ بريد)، يجري تشغيله على مضيف يحتاج لترجمة اسم مضيف إلى عنوان IP، سوف يستدعي التطبيق جانب الزبون لـ DNS ويحدد اسم المضيف الذي يحتاج لترجمته. (على العديد من أجهزة يونيكس، يُستخدم الإجراء gethostbyname() للحصول على الترجمة. سنوضح في الجزء 2-7 كيفية استدعاء DNS من تطبيقات جافا). عندئذ يبدأ DNS العمل في مضيف المُستخدِم بإرسال رسالة استفسار إلى الشبكة. ترسل كل رسائل DNS للاستفسار والرد ضمن وحدة بيانات UDP إلى منفذ رقم 53. بعد زمن تأخير يتراوح من عدة ميللي ثانية إلى عدة ثوانٍ، يتلقى DNS في مضيف المُستخدِم رسالة رد DNS تتضمن المطابقة المطلوبة بين الاسم والعنوان. ترسل تلك المطابقة إلى التطبيق الطالب. وهكذا – من منظور التطبيق الطالب في مضيف المُستخدِم – يبدو DNS كصندوق أسود يوفر خدمة ترجمة مباشرة وبسيطة. ولكن في حقيقة الأمر هذا الصندوق الأسود الذي يوفر تلك الخدمة معقّد، ويتكون من عدد كبير من خادمات DNS الموزّعة حول العالم، بالإضافة إلى بروتوكول في طبقة التطبيقات يحدد كيف تتصل خادمات DNS بالمضيفات المستفسرة.

وكتصميم بسيط لـ DNS يُستخدم خادم DNS واحد يحتوي على كل المطابقات بين الأسماء والعناوين. في هذا التصميم المركزي يوجّه الزبائن كل الاستفسارات إلى ذلك الخادم الذي يتولى الردّ مباشرة عليها. على الرغم من بساطة هذا التصميم وجاذبيته، فهو للأسف غير ملائم للإنترنت اليوم لاتساعها وتزايد عدد المضيفات. ومن بين المشاكل الأخرى لهذا التصميم المركزي:

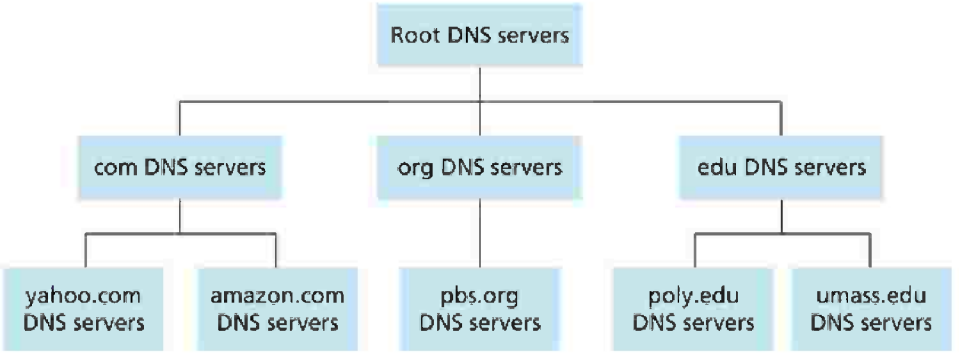
- نقطة وحيدة للعطل: إذا تعطل خادم DNS فسوف تتعطل أيضاً الإنترنت بالكامل!

- زيادة حجم حركة المرور: يجب أن يعالج خادم DNS الوحيد كل استفسارات DNS (لكل طلبات HTTP ورسائل البريد الإلكتروني الناشئة من مئات الملايين من المضيفات).
- قاعدة البيانات مركزية وبعيدة: لا يمكن أن يكون خادم DNS الوحيد "قريباً من" كل الزبائن المستفسرة. فإذا وضعنا خادم DNS الوحيد في مدينة نيويورك، فعلى كل الاستفسارات من استراليا أن تسافر إلى الجانب الآخر من المعمورة، وربما على وصلات اتصال بطيئة ومزدحمة، مما يؤدي إلى زيادة في التأخير.
- الصيانة: يجب أن يحتفظ خادم DNS الوحيد بالسجلات لكل مضيفات الإنترنت. ليست المشكلة فقط في أن قاعدة البيانات المركزية هذه ستكون ضخمة، ولكن سيتعين أيضاً تحديثها باستمرار لتشمل كل مضيف جديد.

وباختصار فإن قاعدة البيانات المركزية في خادم DNS وحيد لا تتناسب ببساطة مع التوسع الكبير في حجم الشبكة. وفي الواقع يعتبر DNS مثالاً رائعاً لكيفية تحقيق قاعدة بيانات موزّعة على الإنترنت.

قاعدة بيانات هرمية وموزّعة

لكي يتعامل DNS مع قضية التوسع المضطرد في الإنترنت، فإنه يستخدم عدداً كبيراً من الخادمات منظّمة بترتيب هرمي وموزّعة حول العالم. في هذا التصميم لا يحتوي خادم DNS وحيد على كل المطابقات لكل المضيفات على الإنترنت وإنما توجد موزّعة عبر عدة خادمات. كتبسيط أولي توجد ثلاثة أصناف من تلك الخادمات مرتبة بشكلٍ هرمي (كما هو مبين في الشكل 2-19): خادمات DNS الجذرية (Root DNS Servers)، خادمات DNS لنطاق المستوى الأعلى ((Top-Level Domain (TLD)، وخادمات DNS المسؤولة (Authoritative DNS Servers).



الشكل 2-19 جزء من التوزيع الهرمي لخادمات DNS.

ولفهم كيفية تعامل هذه الأصناف الثلاثة من الخادمات مع بعضها، افترض أن زبون DNS يريد تحديد عنوان IP لاسم المضيف www.amazon.com في أول تبسيط تتم الأحداث التالية: يتصل الزبون بأحد خادمات الجذر أولاً، والتي ترجع عناوين IP لخادمات TLD لنطاق المستوى الأعلى com. ثم يتصل الزبون بأحد خادمات TLD تلك، والتي ترجع عنوان IP لخادم مسؤول عن amazon.com. أخيراً يتصل الزبون بأحد الخادمات المسؤولة عن amazon.com والذي يرجع عنوان IP لاسم المضيف www.amazon.com. سوف نفحص لاحقاً عملية البحث هذه لدى DNS (DNS lookup) بتفصيل أكثر، ولكن دعنا أولاً نلقي نظرة أدق على تلك الأصناف الثلاثة من خادمات DNS:

- خادمات DNS الجذرية: يوجد على الإنترنت 13 خادم DNS جذري (مسماة من A إلى M)، أغلبها في أمريكا الشمالية. يوضح الشكل 2-20 التوزيع الجغرافي لها في أكتوبر/تشرين عام 2007. وتوجد قائمة بهذه الخادمات في [Root-servers 2007]. وبالرغم من أننا أشرنا إلى كلٍّ منها كما لو كان خادماً وحيداً، فإن كل "خادم" في الحقيقة عبارة عن مجموعة (cluster) من الخادمات المكررة من أجل زيادة الاعتمادية (reliability) والأمن (security).

الشكل 2-20 التوزيع الجغرافي لخادمات DNS الجذرية في عام 2007.

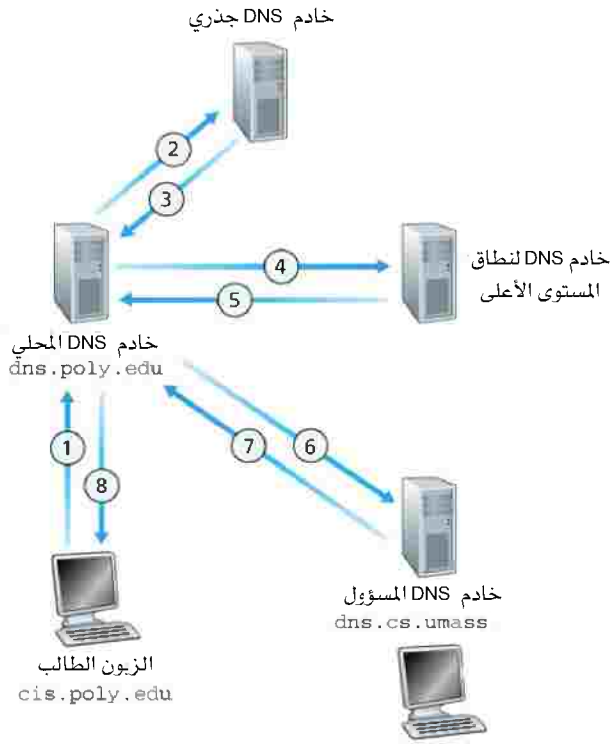
- خادمات نطاق المستوى الأعلى (TLD): هذه الخادمات مسؤولة عن نطاقات المستوى الأعلى مثل com، org، net، edu، gov وكذلك كل نطاقات المستوى الأعلى التي تمثل البلدان مثل uk، fr، ca، jp. وفي وقت كتابة هذا الكتاب (ربيع عام 2007) كانت شركة حلول الشبكات (Network Solutions) مسؤولة عن خادمات TLD لنطاق المستوى الأعلى com، وشركة Educause عن خادمات TLD لنطاق المستوى الأعلى edu.
- خادمات DNS المسؤولة: يتعين على كل منظمة لها مضيفات متاحة للوصول العام على الإنترنت (مثل خادمات الويب وخادمات البريد) توفير سجلات DNS سهلة الوصول بشكلٍ عام لتحويل أسماء تلك المضيفات إلى عناوين IP، ويحتفظ خادم DNS المسؤول الخاص بتلك المنظمة بسجلات DNS تلك. يمكن أن تختار المنظمة أن توفر بنفسها خادم DNS المسؤول الخاص بها لحفظ تلك السجلات. وكبديل لذلك يمكن أن تدفع المنظمة رسوماً لبعض موفّري الخدمة (service providers) مقابل تخزين السجلات على خادم DNS مسؤول لديها. معظم الجامعات والشركات

الكبيرة تحتفظ بخادمات DNS مسؤولة أساسية وثانوية (احتياطية) خاصة بها.

تُنظَم كل خادمات DNS (الجذرية وTLD والمسؤولة) في تركيب هرمي (شجري) كما هو مبين في الشكل 2-19. هناك نوع آخر مهم من خادمات DNS يطلق عليه خادمات DNS المحلية. وللدقة لا ينتمى خادم DNS المحلي لشجرة الخادمات، ولكنه مع ذلك يلعب دوراً محورياً في بنية DNS المعمارية. يوجد لدى كل موفّر خدمة إنترنت (كجامعة، أو قسم أكاديمي، أو شركة، أو موفّر خدمة الإنترنت السكني) خادم DNS محلي (يسمى خادم الاسم الاعتيادي (default name server)). عندما يتصل مضيف مع موفّر خدمة إنترنت فإن موفّر الخدمة يعطي المضيف عنوان IP لواحد أو أكثر من خادمات DNS المحلية لديه (عادةً من خلال بروتوكول DHCP والذي سنتناوله في الفصل الرابع). يمكنك معرفة عنوان IP لخادم DNS المحلي لديك بسهولة بفتح نوافذ حالة الشبكة في نظام تشغيل ويندوز أو يونيكس. ويكون خادم DNS المحلي عادة "قريباً من" المضيف. في حالة موفّر خدمة إنترنت لمؤسّسة قد يكون خادم DNS المحلي على نفس شبكة الاتصالات المحلية LAN كالمضيف. أما في حالة موفّر خدمة الإنترنت السكني فعادة ما يفصل بينه وبين المضيف ما لا يزيد عن بضعة موجّهات (routers). وعندما يُرسِل مضيف استفسار DNS، يُرسَل الاستفسار إلى خادم DNS المحلي، والذي يتصرف كـ "وكيل" (proxy) فيُرسِل الاستفسار إلى شجرة خادمات DNS كما سنناقش بتفصيل أكثرفيما بعد.

لنلقِ نظرة على مثال بسيط. افترض أن المضيف cis.poly.edu يرغب في الحصول على عنوان IP للمضيف gaia.cs.umass.edu، وافترض أيضاً أن خادم DNS المحلي لجامعة Polytechnic يُدعى dns.poly.edu وأن خادم DNS المسؤول عن gaia.cs.umass.edu يُدعى dns.umass.edu. كما هو موضح في الشكل 2-21 يرسل المضيف cis.poly.edu أولاً رسالة استفسار DNS إلى خادم DNS المحلي dns.poly.edu تتضمن اسم المضيف الذي يريد ترجمته (أي gaia.cs.umass.edu). يوجه خادم DNS المحلي رسالة الاستفسار إلى خادم DNS

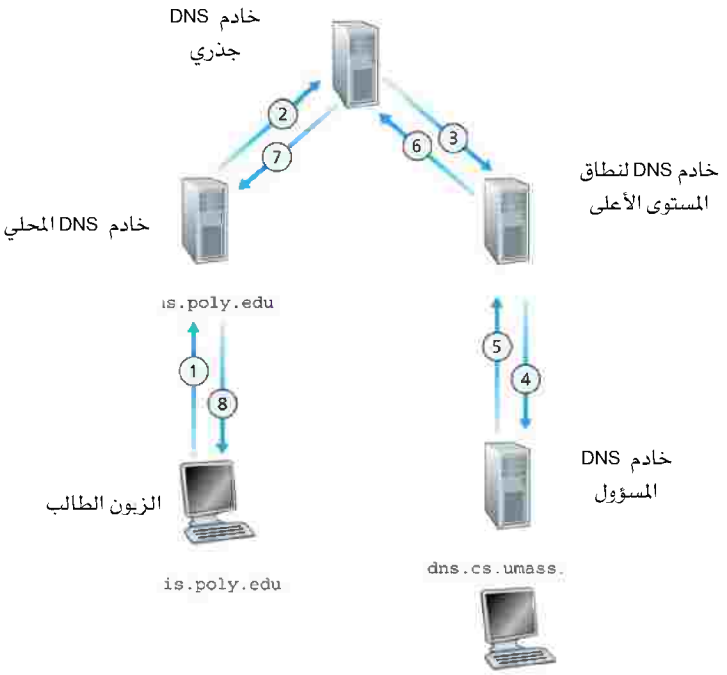
الجذري والذي يلاحظ اللاحقة edu ويرجع إلى خادم DNS المحلي قائمة عناوين IP لخادمات TLD المسؤولة عن نطاق edu. بعد ذلك يرسل خادم DNS المحلي رسالة الاستفسار مرة ثانية إلى أحد خادمات TLD تلك، والذي يلاحظ اللاحقة umass.edu، فيرد بعنوان IP لخادم DNS المسؤول لجامعة ماسوشوستس (أي dns.umass.edu). أخيراً يُرسِل خادم DNS المحلي رسالة الاستفسار مرة ثانية مباشرةً إلى dns.umass.edu والذي يرد بعنوان IP لـ gaia.cs.umass.edu. لاحظ في هذا المثال أنه لكي نحصل على العنوان المرتبط باسم مضيف واحد، تم إرسال ثماني رسائل DNS: أربع رسائل استفسار وأربع رسائل رد! وسنرى قريباً كيف يستخدم DNS الذاكرة المخبأة ليُخفّض حركة المرور الناجمة عن تلك الاستفسارات.



الشكل 2-21 التفاعل بين خادمات DNS المختلفة.

في مثالنا السابق افترضنا أن خادم TLD يعرف خادم DNS المسؤول الذي يتضمن اسم المضيف، ولكن ذلك لن يكون صحيحاً دائماً. بدلاً من ذلك فإن خادم TLD قد يعرف فقط خادم DNS وسيط والذي يعرف بدوره خادم DNS المسؤول الذي يتضمن اسم المضيف. على سبيل المثال افترض ثانية أن جامعة ماسوشوستس لها خادم DNS يسمى dns.umass.edu وافترض أيضاً أن كل قسم من أقسام الجامعة له خادم DNS خاص به وأن ذلك الخادم مسؤول عن كل المضيفات في القسم. في هذه الحالة عندما يستلم خادم DNS الوسيط dns.umass.edu استفساراً عن مضيف ينتهي اسمه بـ cs.umass.edu فسوف يرجع إلى dns.poly.edu عنوان IP لـ dns.cs.umass.edu والذي هو مسؤول عن كل أسماء المضيفات التي تنتهي بـ cs.umass.edu ثم يُرسِل خادم DNS المحلي dns.poly.edu الاستفسار إلى خادم DNS المسؤول الذي يُرجع العنوان المطلوب إلى خادم DNS المحلي والذي يُرجع بدوره العنوان إلى المضيف الطالب. في هذه الحالة تُرسَل عشر رسائل DNS!

يستخدم المثال الموضح في الشكل 2-21 كلاً من الاستفسارات التتابعية (recursive) والاستفسارات التكرارية (iterative). الاستفسار الذي أُرسِل من المضيف cis.poly.edu إلى الخادم dns.poly.edu استفسار تتابعي، وفيه يطلب المضيف من الخادم أن يحصل له على العنوان نيابة عنه. لكن الاستفسارات الثلاثة التالية تكرارية لأن كل الردود عليها تعود مباشرةً إلى dns.poly.edu. نظرياً يمكن أن يكون أي استفسار DNS تتابعي أو تكراري. على سبيل المثال يوضح الشكل 2-22 سلسلة استفسارات DNS كلها تتابعية. عملياً عادة ما تتبع الاستفسارات النمط الموضح في الشكل 2-21، أي يكون الاستفسار من المضيف الطالب إلى خادم DNS المحلي تتابعياً، بينما بقية الاستفسارات تكون تكرارية.



الشكل 2-22 استفسارات DNS التتابعية.

ذاكرة DNS المخبأة

أهملنا في مناقشتنا حتى الآن استخدام DNS للذاكرة المخبأة والتي تمثل ميزة هامة جداً لنظام DNS. في الحقيقة يستخدم DNS الذاكرة المخبأة على نطاق واسع لكي يُحسّن الأداء بتقليل زمن التأخير وتخفيض عدد رسائل DNS التي تجوب الإنترنت. الفكرة وراء استخدام DNS للذاكرة المخبأة بسيطة للغاية. ففي سلسلة استفسارات عندما يستلم خادم DNS رد DNS (يتضمن مثلاً مطابقة بين اسم مضيف وعنوان IP)، يمكن أن يحتفظ بنسخة منه في ذاكرته المحلية. على سبيل المثال في الشكل 2-21 في كل مرة يتلقى خادم DNS المحلي dns.poly.edu رداً من خادم DNS يمكن أن يُخزّن أياً من المعلومات الواردة في الرد في تلك الذاكرة.

إذا وُجد زوج البيانات المكون من اسم المضيف وعنوان IP المناظر مخزناً على خادم DNS ووصل استفسار جديد إلى الخادم لنفس اسم المضيف، يمكن أن يرد الخادم بعنوان IP المطلوب، حتى إذ لم يكن هو الخادم المسؤول عن اسم المضيف. ولأن المضيفات والمطابقة بين أسمائها وعناوين IP المقابلة لها ليست ثابتة على الإطلاق (نظراً لتخصيص العناوين ديناميكياً) فإن خادمات DNS تحذف المعلومات المخبأة بعد فترة معينة (تحدد غالباً بيومين).

وكمثال افترض أن مضيف apricot.poly.edu يريد أن يسأل dns.poly.edu عن عنوان IP للمضيف cnn.com. علاوة على ذلك افترض أنه بعد ساعات قليلة يقوم مضيف آخر من جامعة بولي تكنك مثلاً kiwi.poly.edu بالاستفسار من dns.poly.edu عن اسم المضيف نفسه. مع استخدام الذاكرة المخبأة، سيكون خادم DNS المحلي قادراً على إرجاع عنوان IP لـ cnn.com فوراً للمضيف الثانى المستفسر بدون الحاجة للاستفسار من أي من خادمات DNS الأخرى. ويمكن أيضاً لخادم DNS المحلي أن يُخزّن عناوين IP لخادمات TLD مما يسمح لخادم DNS المحلي بتخطى خادمات DNS الجذرية في سلسلة استفسار (وهذا ما يحدث في أغلب الأحيان).

3-5-2 سجلات ورسائل DNS

تقوم خادمات DNS (والتي تكوّن فيما بينها قاعدة بيانات DNS الموزّعة) بتخزين سجلات الموارد ((Resource Records (RRs بما في ذلك السجلات الخاصة بالمطابقة ما بين اسم المضيف وعنوان IP المناظر. تحمل كل رسالة رد DNS سجلاً واحداً أو أكثر. سنلقي في هذا الجزء والجزء الذي يليه نظرة عامة وسريعة عن سجلات ورسائل DNS، ويمكن الحصول على تفاصيل أكثر من [Abitz 1993] أو من طلبات التعليقات حول DNS [RFC 1034; RFC 1035].

يتكون سجل المورد من الأربعة حقول التالية:

(Name, Value, Type, TTL)

يُمثل الحقل TTL فترة العمر لسجل المورد، وهو يحدد متى يجب أن يحذف السجل من الذاكرة المخبأة. في الأمثلة التي نقدّمها فيما يلي أهملنا حقل TTL. ويعتمد معنى الحقل Name والحقل Value على قيمة حقل Type كما يلي:

- إذا كانت قيمة Type = A ، فعندئذ تمثل قيمة الحقل Name اسم المضيف، بينما تمثل قيمة الحقل Value عنوان IP المناظر لذلك الاسم. وهكذا يعطي السجل من نوع A المطابقة القياسية بين اسم المضيف وعنوان IP المناظر. وكمثال لهذا النوع من السجلات:

(relayl.bar.foo.com, 145.37.93.126, A)

- إذا كانت قيمة Type = NS فإن قيمة الحقل Name تمثل اسم نطاق (مثل foo.com) وقيمة الحقل Value تمثل اسم خادم DNS المسؤول عن هذا النطاق والذي يعرف كيف يحصل على عناوين IP للمضيفات في ذلك النطاق. يُستخدَم هذا السجل لإعادة توجيه استفسارات DNS في سلسلة الاستفسار. وكمثال لهذا النوع:

(foo.com, dns.foo.com, NS)

- إذا كانت قيمة Type = CNAME فإن قيمة الحقل Name تمثل اسم المضيف القانوني لاسم بديل للمضيف والتي تذكر في حقل Value. يمكن أن يعطي هذا السجل رداً للاستفسار عن الاسم القانوني لاسم مضيف ما. وكمثال لهذا النوع:

(foo.com, relayl.bar.foo.com, CNAME)

- إذا كانت قيمة حقل Type = MX فإن قيمة حقل Name تمثل الاسم القانوني لخادم البريد للاسم البديل للمضيف الموجود في حقل Value. وكمثال لهذا النوع:

(foo.com, mail.bar.foo.com, MX)

- تسمح سجلات MX بإعطاء أسماء شهرة بديلة بسيطة لأسماء مضيفات خادمات البريد. لاحظ أنه باستعمال السجل MX يمكن أن تستخدم

شركة نفس الاسم البديل لخادم البريد ولأحد خادماتها الأخرى (مثل خادم الويب لديها). وللحصول على الاسم القانوني لخادم البريد سوف يستفسر زبون DNS عن سجل MX، وللحصول على الاسم القانوني للخادم الآخر سوف يستفسر زبون DNS عن سجل CNAME.

إذا كان خادم DNS هو الخادم المسؤول عن اسم مضيف معين فإن خادم DNS سيتضمن سجلاً من النوع A لاسم المضيف. (حتى إذا كان خادم DNS ليس الخادم المسؤول فقد يتضمن أيضاً سجلاً من النوع A في ذاكرته المخبأة). إذا كان الخادم ليس مسؤولاً عن اسم المضيف فإنه سيتضمن سجلاً من النوع NS للنطاق الذي يتضمن اسم المضيف، وكذلك أيضاً سجلاً من النوع A يوفر عنوان IP لخادم DNS الموجود في حقل Value لسجل NS. كمثال افترض أن خادم TLD مسؤول عن النطاق edu ولكنّه ليس مسؤولاً عن المضيف gaia.cs.umass.edu عندئذ سيتضمن هذا الخادم سجلاً للنطاق الذي يضم المضيف cs.umass.edu على سبيل المثال:

(umass.edu, dns.umass.edu, NS)

وسيتضمن خادم TLD الخاص بنطاق edu أيضاً سجلاً من النوع A يحدد عنوان IP المناظر لاسم خادم أسماء النطاقات dns.umass.edu على سبيل المثال:

(dns.umass.edu, 128.119.40.111, A)

رسائل DNS

أشرنا سابقاً في هذا الجزء إلى رسائل DNS الخاصة بالاستفسار والرد، وهذان هما النوعان الوحيدان من أنواع رسائل DNS. كما أن كلاً من رسائل الاستفسار والرد لهما نفس الصيغة كما هو موضح في الشكل 2-23.

تتلخص معاني الحقول المختلفة في رسالة DNS فيما يلي:

يمثل أول 12 بايتاً الجزء الخاص بسطور الترويسة، والذي يتضمن بدوره عدداً من الحقول. الحقل الأول هو رقم تعريفي (identifier) يميز

الاستفسار ويتكون من 16 بتاً. ينسخ هذا المعرّف في رسالة الرد على ذلك الاستفسار، مما يسمح للزبون بربط الردود التي يتلقاها بالاستفسارات التي أرسلها. يوجد عدد من الأعلام (flags) في حقل الأعلام. يشير أحد الأعلام إلى نوع الرسالة (0 يعني رسالة استفسار، و1 يعني رسالة رد)، ويشير آخر في رسالة الرد إلى أن الخادم هو الخادم المسؤول عن اسم المضيف الجاري الاستفسار عنه. تكون قيمة عَلم البحث التتابعي (recursion) 1 عندما يرغب الزبون (مضيف أو خادم DNS) في أن يقوم خادم DNS بالبحث التتابعي في حالة عدم وجود السجل المطلوب لديه. وإذا كان الخادم يدعم البحث التتابعي فإنه يضع القيمة 1 في ذلك العَلم في رسالة الرد. تتضمن الترويسة أيضاً أربعة حقول يشير كلٌّ منها إلى طول كل قسم من الأقسام الأربعة التي تلي الترويسة.

الشكل 2-23 صيغة رسالة DNS.

- يحتوي "قسم السؤال" (question section) على معلومات عن الاستفسار المطلوب. ويتضمن هذا القسم: (1) حقل "Name" ويحتوي على الاسم المستفسر عنه، (2) حقل "Type" ويشير إلى نوع الاستفسار المطلوب، على سبيل المثال عنوان المضيف المناظر لاسم (النوع A) أو خادم البريد المناظر لاسم (النوع MX).
- في رسالة رد من خادم DNS، يحتوي "قسم الرد" (answer section) على سجلات الموارد للاسم المستفسر عنه في الأصل. تذكّر أنه يوجد في كل سجل مورد حقل النوع (على سبيل المثال: A، NS، CNAME، MX)، وحقل القيمة، وفترة العمر TTL. يمكن أن ترجع رسالة الرد العديد من سجلات المورد (RR) في الجواب، لأن اسم المضيف يمكن أن يكون له عدة عناوين IP (كما في حالة خادمات الويب المكررة كما ناقشنا سابقاً في هذا الجزء).
- يحتوي "قسم المسؤولية" (authority section) على سجلات الخادمات المسؤولة (authoritative servers) الأخرى.
- يحتوي "القسم الإضافي" (additional section) على سجلات أخرى مساعدة. على سبيل المثال يحتوي حقل الرد في رسالة الرد لاستفسار MX على سجل مورد يعطي اسم المضيف القانوني لخادم البريد. كما يحتوي القسم الإضافي على سجل من النوع A يعطي عنوان IP لاسم المضيف القانوني لخادم البريد.

كيف ترسل رسالة استفسار DNS مباشرةً من المضيف الذي تعمل عليه إلى خادم DNS؟ يمكن القيام بذلك بسهولة باستخدام برنامج البحث nslookup، والمتوفر على معظم أجهزة ويندوز ويونيكس. مثلاً من مضيف ويندوز افتح واجهة الأوامر (command prompt) واستدع برنامج nslookup، ويتم ذلك ببساطة بكتابة nslookup. بعد استدعاء البرنامج يمكن أن ترسل استفسار DNS إلى أي خادم DNS (جذر، أو TLD، أو مسؤول). بعد استلام رسالة الرد من خادم DNS تستعرض أداة البحث السجلات المُتضمّنة في الرد (بشكلٍ يسهل على المُستخدِم

قراءته). وكبديل لتشغيل أداة البحث nslookup من مضيفك الخاص، يمكن أن تزور أحد مواقع الويب العديدة التي تسمح لك باستخدام أداة البحث عن بُعد. (فقط اكتب "nslookup" في أي محرك بحث على شبكة الويب وستجلب إليك أحد تلك المواقع).

إدخال سجلات إلى قاعدة بيانات DNS

ركزت المناقشة السابقة على كيفية استرجاع سجلات من قاعدة بيانات DNS، وقد تتساءل وكيف أدخلت تلك السجلات في قاعدة البيانات في المقام الأول؟ دعنا ننظر إلى كيفية تحقيق ذلك في سياق مثال معين. افترض أنك أنشأت شركة جديدة تسمى شبكة Utopia، فإن أول شيء تريده بالتأكيد هو أن تسجل اسم النطاق networkutopia.com لدى أحد المسجِّلين (registrars). والمسجِّل هو كيان تجاري يقوم بالتحقق من تفرد اسم النطاق (أي عدم استخدامه من قِبَل جهة أخرى)، وكذلك إضافة أسماء النطاقات إلى قاعدة بيانات DNS (كما سنناقش فيما بعد) وذلك نظير أجر بسيط مقابل خدماته. قبل عام 1999 كان المسجِّل الوحيد هو شركة Network Solutions، وكان لديها احتكار على تسجيل النطاقات com، net، org. ولكن الآن هناك العديد من المسجِّلين يتنافسون على الزبائن، وتقوم شركة الإنترنت للأسماء والأعداد المُخصَّصة (Internet Corporation for Assigned Names and Numbers (ICANN)) بترخيص المسجِّلين المعتمدين. توجد قائمة كاملة بالمسجِّلين المعتمدين على الموقع .http://www.internic.net

عندما تسجل اسم النطاق networkutopia.com مع مسجِّل ما ستحتاج أيضاً لتزويد المسجِّل بالأسماء وعناوين IP لخادمات DNS الأساسية والثانوية المسؤولة. افترض أن الأسماء وعناوين IP هي:

dns1.networkutopia.com
dns2.networkutopia.com
212.212.212.1
212.212.212.2

سوف يتأكد المسجِّل من أن سجلاً من نوع NS وآخر من نوع A قد أدخلت ضمن خادمات TLD للنطاق com، وذلك لكلٍّ من هذين الخادمين المسؤولين. وبالتحديد، يقوم المسجِّل بإدخال السجلين التاليين إلى نظام DNS بخصوص الخادم المسؤول الأساسي للنطاق networkutopia.com:

(networkutopia.com, dnsl.networkutopia.com, NS)
(dnsl.networkutopia.com, 212.212.212.1, A)

يجب أيضاً التأكد من إضافة سجل من النوع A لخادم الويب www.networkutopia.com وسجل من النوع MX لخادم البريد mail.networkutopia.com إلى خادمات DNS المسؤولة. لوقت قريب كانت محتويات كلٍّ من خادمات DNS تضبط بطريقة ثابتة (على سبيل المثال من ملف تهيئة يقوم بإعداده مدير النظام). ومؤخراً أُضيف خيار UPDATE إلى نظام DNS للسماح بالإضافة أو الحذف من قاعدة البيانات عن طريق رسائل DNS بطريقة ديناميكية. ويوجد توصيف التعديل الديناميكى لنظام DNS في RFC 2136 و RFC 3007.

بمجرد الانتهاء من كل تلك الخطوات، سيكون بوسع الجمهور زيارة موقع الشركة وإرسال البريد الإلكتروني إلى موظفيها. دعنا نختم مناقشتنا عن DNS بإثبات صحة هذه الجملة. يساعد هذا الإثبات أيضاً في تقوية ما تعلمناه عن DNS.

افترض أن أليس في استراليا أرادت استعراض صفحة الويب www.networkutopia.com. كما ناقشنا في وقت سابق سيرسل مضيفها أولاً استفسار DNS إلى خادم DNS المحلي. بعد ذلك سيتصل خادم DNS المحلي بخادم TLD المسؤول عن النطاق com (يجب أيضاً أن يتصل خادم DNS المحلي بخادم DNS الجذرى إذا كان عنوان خادم TLD المسؤول عن النطاق com غير موجود بالذاكرة المخبأة). يحتوي خادم TLD هذا على سجل من النوع NS وآخر من النوع A كما عرضنا سابقاً، حيث تأكد المسجِّل من إدخال تلك السجلات إلى كل خادمات TLD للنطاق com . يرسل خادم TLD للنطاق com رداً لخادم DNS المحلي لدى أليس يحتوي على السجلين. ثم يرسل خادم DNS المحلي استفسار DNS إلى

212.212.212.1 يسأل عن سجل من النوع A المناظر لـ www.networkutopia.com. يعطي هذا السجل عنوان IP لخادم الويب المطلوب (افترض أنه 212.212.71.4) والذي يرسله خادم DNS المحلي إلى مضيف أليس. يمكن أن يبدأ متصفّح أليس الآن توصيلة TCP إلى المضيف 212.212.71.4 ويرسل طلب HTTP على التوصيلة. وكما ترى فإن ما يجرى خلف كواليس الإنترنت أكثر بكثير مما تراه العين عند تصفح الويب!

نبذة عن الأمن (Focus on Security)

نقاط ضعف DNS (الثغرات الأمنية في DNS)

رأينا أن DNS هو أحد المكونات الرئيسة للبنية التحتية للإنترنت حيث لا يمكن أن تعمل بدونه العديد من الخدمات الهامة مثل الويب والبريد الإلكتروني. والسؤال الطبيعي هو كيف يمكن الهجوم على DNS؟ هل يقبع DNS منتظراً أن يُبعَد عن الخدمة متسبباً في تعطل الكثير من تطبيقات الإنترنت؟

أول ما يخطر على الذهن هو هجوم فيضان الحيّز الترددي الموزّع لحجب الخدمة (Distributed Denial of Service (DDoS)) (انظر الجزء 1-6) ضد خادمات DNS. على سبيل المثال يمكن أن يحاول مهاجم إرسال سيل من الرزم إلى كل خادم من خادمات DNS الجذرية بحيث لا يستطيع الرد على العديد من استفسارات DNS الشرعية. في الحقيقة حدث مثل هذا الهجوم على نطاق واسع ضد خادمات DNS الجذرية في 21 أكتوبر/تشرين الأول عام 2002، حيث قام المهاجمون باستخدام شبكة الروبوت (botnet) لإرسال كم هائل من رسائل ICMP للبينج إلى كلٍّ من خادمات DNS الجذرية الثلاثة عشر (سنناقش رسائل ICMP في الفصل الرابع لكن يكفي الآن أن تعرف أنها أنواع خاصة من وحدات بيانات IP). لحسن الحظ تسبب هذا الهجوم الواسع النطاق في إحداث أقل ما يمكن من الأضرار والتى لم يتأثر بها الكثير من مُستخدمي الإنترنت، فبرغم نجاح المهاجمين في توجيه سيل الرزم إلى الخادمات الجذرية إلا أن العديد منها كانت محمية بمرشحات الرزم (packet filters) والمهيئة لمنع كل رسائل ICMP للبينج الموجهة نحو خادمات الجذر. ساعد ذلك على استمرار الخادمات المحمية في العمل بشكلٍ طبيعي. علاوة على ذلك تحتفظ معظم خادمات DNS المحلية بعناوين IP لخادمات نطاق المستوى الأعلى (TLD) مما يسمح لعملية الاستفسار

بتجاوز خادمات DNS الجذرية في أغلب الأحيان.

هجوم آخر قد يكون أكثر فعّالية هو شن هجوم DDoS ضد خادمات نطاق المستوى الأعلى (TLD) بإرسال سيل من استفسارات DNS لتلك الخادمات، كإرسال كم هائل من استفسارات DNS إلى كل خادمات المستوى الأعلى للنطاق com سيكون ترشيح استفسارات DNS الموجّهة إلى خادمات DNS أكثر صعوبة وكذلك لا يمكن تجاوز تلك الخادمات بنفس السهولة التي يتم بها تجاوز خادمات DNS الجذرية. لكن يمكن الحد من شدة مثل هذا الهجوم باستخدام ذاكرة مخبأة في خادمات DNS المحلية.

من الممكن مهاجمة DNS بطرق أخرى. مثلاً في هجوم "رجل في الوسط" (man-in-the-middle) يعترض المهاجم الاستفسارات من المضيفات ويرجع ردوداً مزيفة. وفي هجوم تسميم DNS يرسل المهاجم إجابات مزيفة إلى خادم DNS والذى ينخدع ويضيف السجلات المزيفة إلى ذاكرته المخبأة. يمكن أن تستغل تلك الهجمات على سبيل المثال لتوجيه مُستخدم ويب إلى موقع الويب للمهاجم. ومع ذلك فهذه الهجمات صعبة التطبيق حيث تتطلب القدرة على اعتراض الرزم أو خنق الخادمات [Skoudis 2006].

وهناك هجوم DNS آخر هام لكنه ليس هجوماً على خدمة DNS في حد ذاته وإنما يستغل البنية التحتية لـ DNS لشنّ هجوم DDoS ضد مضيف مستهدف (على سبيل المثال خادم البريد الإلكتروني لجامعتك). في هذا الهجوم يرسل المهاجم استفسارات DNS إلى العديد من خادمات DNS المسؤولة بكلٍّ منها عنوان مصدر مغشوش للمضيف المستهدف. حينئذ ستقوم خادمات DNS بإرسال الأجوبة مباشرة إلى المضيف المستهدف (لاحظ أن المضيف المستهدف لم يرسل أي استفسار في حقيقة الأمر). إذا أمكن تشكيل الاستفسارات بحيث يكون حجم رسائل الرد أكبر بكثير (في عدد البايتات) من الاستفسار (وهو ما يسمى بالتضخيم (amplification)) ففي هذه الحالة يمكن أن يغمر الهدف لدرجة تمنعه من توليد معظم حركة مرور بياناته الخاصة. إلا أن نجاح مثل تلك الهجمات الإنعكاسية التي تستغل DNS ما زال محدوداً حتى الآن [Mirkovic 2005].

وخلاصة القول أن DNS قد أثبت قدرته على صد مثل تلك الهجمات ضده. حتى الآن لم ينجح هجوم في عرقلة خدمة DNS، ورغم ذلك كانت هناك بعض الهجمات الإنعكاسية الناجحة ولكن أمكن التصدي لها بالإعداد المناسب لخادمات DNS.

6-2 تطبيقات النظائر (Peer-to-Peer Applications)

ذكرنا في الجزء 2-1-1 أنه بشكلٍ عام يمكن تصميم التطبيق باستخدام بنية معمارية تعتمد على مفهوم "زبون/خادم" أو باستخدام بنية معمارية تعتمد على مفهوم "النظائر". تستخدم كل التطبيقات التي وصفناها حتى الآن في هذا الفصل (بما في ذلك الويب، والبريد الإلكتروني، وخدمة أسماء النطاقات DNS) البنية المعمارية "زبون/خادم" حيث تعتمد أساساً على خادمات البنية التحتية التي تعمل دائماً (always-on). في حين أن بنية النظائر P2P – كما ذكرنا – تعتمد بأقل قدر ممكن (أو لا تعتمد على الإطلاق) على خادمات البنية التحتية التي تعمل دائماً. بدلاً من ذلك تتصل مباشرةً أزواج من المضيفات (تسمى النظائر) بشكلٍ متقطع مع بعضها البعض. وتلك النظائر ليست ملكاً لموفّر الخدمة وإنما هي حاسبات مكاتب وحاسبات نقالة تحت سيطرة المُستخدِمين.

وسوف نتناول في هذا الجزء ثلاثة تطبيقات مختلفة تلائم تماماً تصميم بنية النظائر. التطبيق الأول هو توزيع الملفات، وفيه يتم توزيع ملف من مصدر وحيد إلى عدد كبير من النظائر. ويعتبر توزيع الملفات تطبيقاً ملائماً لبدء دراسة بنية النظائر، حيث يُظهِر بشكلٍ واضح القدرة الذاتية على التوسع (self-scalability) لتلك البنية. وكمثال محدد لتوزيع الملفات سوف نصف نظام BitTorrent الشائع. أما تطبيق النظائر الثاني الذي سنتناوله فهو تنظيم المعلومات والبحث عنها في مجتمع من النظائر، حيث سنستكشف عدة أنظمة مختلفة يطبق كلٌّ منها في نظم مشاركة النظائر للملفات على نطاق واسع (large-scale file sharing). في التطبيق الثالث والأخير سنستكشف سكايب (Skype)، وهو تطبيق نظائر ناجح بشكلٍ هائل لهاتف الإنترنت.

1-6-2 توزيع النظائر للملفات

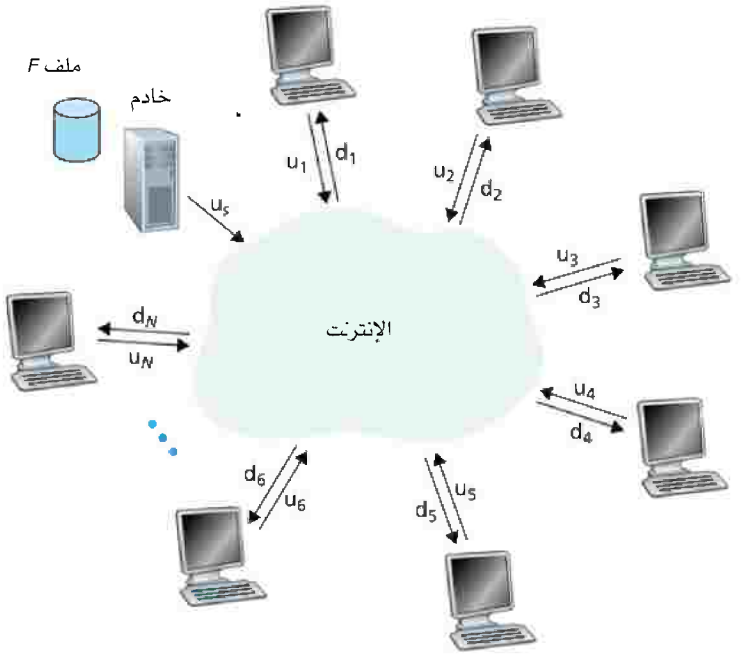
لنبدأ رحلتنا مع بنية النظائر بدراسة تطبيق طبيعي جداً، ألا وهو توزيع ملف كبير الحجم من خادم وحيد إلى عدد كبير من المضيفات (النظائر). قد يكون

هذا الملف نسخة جديدة من نظام تشغيل لاينكس (Linux)، أو ترميم (patch) لنظام التشغيل الحالي أو لبرنامج تطبيق حالي، أو ملف موسيقى MP3، أو ملف فيديو MPEG. في توزيع الملفات باستخدام بنية "زبون/خادم"، يجب أن يرسل الخادم نسخة من الملف إلى كل زبون من الزبائن المعنية، مما يُشكِّل عبئاً هائلاً على الخادم ويستهلك كمية كبيرة من الحيّز الترددي المخصص له. أما في توزيع الملفات ببنية النظائر، يمكن أن يعيد كل نظير توزيع أي جزء من الملف الذي استلمه إلى أيٍّ من النظائر الأخرى، وبذلك يساعد الخادم في عملية التوزيع. وفي وقت تأليف هذا الكتاب (ربيع عام 2007) كان البروتوكول الأكثر شعبية لتوزيع النظائر للملفات هو BitTorrent. وببعض التقديرات يمثل BitTorrent حوالي 30٪ من حركة مرور البيانات على شبكة العمود الفقري الرئيسة للإنترنت [CacheLogic 2007]. لقد طُوِّرهذا البروتوكول في الأصل من قِبَل برام كوهين (Bram Cohen)، وهناك الآن العديد من زبائن BitTorrent المستقلين والمتوافقين مع بروتوكول BitTorrent تماماً مثلما يوجد العديد من متصفّحات الويب المتوافقة مع بروتوكول HTTP. في هذا الجزء سنفحص أولاً القدرة الذاتية على التوسع (self-scalability) بسهولة لبنية النظائر في سياق توزيع الملفات، وبعدها سنصف بعض تفاصيل BitTorrent مبرزين خصائصه وميزاته الهامة.

قدرة بنية النظائر على التوسع

لمقارنة البنية المعمارية "زبون/خادم" ببنية النظائر المعمارية، وتوضيح القدرة الذاتية الطبيعية على التوسع في بنية النظائر، سنأخذ في الاعتبار الآن نموذجاً كمّياً بسيطاً لتوزيع ملف على مجموعة من النظائر عن طريق كلا النوعين للبنية المعمارية. يبين الشكل 2-24 اتصال الخادم والنظائر بالإنترنت. لِنَرمُز لمعدل التحميل لوصلة الوصول للخادم بـ u_s، ومعدل التحميل لوصلة الوصول للنظير i بـ u_i، ومعدل التنزيل لوصلة الوصول للنظير i بـ d_i. نرمز أيضاً لحجم الملف المراد توزيعه بـ F (بتات) ولعدد النظائر التي تريد الحصول على نسخة من الملف بـ N. نعرّف "زمن التوزيع" (distribution time) بأنه الزمن اللازم ليحصل كل نظير على

نسخة من الملف. في تحليلنا التالي لزمن التوزيع لكلٍّ من بنية "زبون/خادم" وبنية "النظائر" سنفترض لتبسيط التحليل أن "لُب" الإنترنت له حيّز ترددي وفير (وعموماً هذا الافتراض صحيح [Akella 2003])، مما يعني ضمناً أن عنق الزجاجة (bottleneck) يكمن في التوصل إلى الشبكة. نفترض أيضاً أن الخادم والزبائن لا يشاركون في أي تطبيقات شبكة أخرى، كي يكرّس كل الحيّز الترددي المتاح لهم بالكامل للتنزيل والتحميل اللازمين لتوزيع ذلك الملف.



الشكل 2-24 توضيح لمشكلة توزيع الملفات.

دعنا نحسب أولاً زمن التوزيع لبنية "زبون/خادم"، والذي نرمز له بـ D_{cs}. في بنية زبون/خادم لا تساعد النظائر في توزيع الملف. لنُبدِ الملاحظات التالية:

- يجب أن يرسل الخادم نسخة واحدة من الملف إلى كلٍّ من النظائر. وهكذا يجب أن يرسل الخادم NF بت. ولأن معدل إرسال الخادم u_s، فإن زمن توزيع الملف يجب أن يكون على الأقل $\frac{NF}{u_s}$.
- لنرمز لمعدل التنزيل للزبون صاحب أقل معدل بـ $d_{\min}$ أي أن

$$d_{\min} = \min(d_1, d_2, \ldots, d_N)$$

لا يستطيع النظير صاحب معدل التنزيل الأدنى الحصول على كل بتات الملف (F بت) في أقل من $\frac{F}{d_{\min}}$ ثانية. وهكذا يكون زمن التوزيع الأدنى على الأقل $\frac{F}{d_{\min}}$.

وبدمج كلتا الملاحظتين معاً نحصل على

$$D_{cs} \geq \max\left\{\frac{NF}{u_s}, \frac{F}{d_{\min}}\right\}$$

وهو يمثل حداً أدنى للقيمة الصغرى لزمن التوزيع لبنية "زبون/خادم". في التمارين الموجودة في نهاية الفصل سوف يُطلَب منك إثبات أن الخادم يمكن أن يجدول إرساله لكي يتحقق الحد الأدنى في الواقع. لذا دعنا نأخذ هذا الحد الأدنى كزمن التوزيع الفعلي، أي أن:

$$D_{cs} = \max\left\{\frac{NF}{u_s}, \frac{F}{d_{\min}}\right\}, \quad (2\text{-}1)$$

من هذه المعادلة يتبين أنه لقيم N الكبيرة بما فيه الكفاية، يكون زمن التوزيع لبنية "زبون/خادم" $\frac{NF}{u_s}$. وهكذا يزيد زمن التوزيع بشكلٍ خطي مع عدد النظائر N، فمثلاً إذا زاد عدد النظائر من أسبوع لآخر من ألف إلى مليون (أي ألف ضعف)، فإن زمن توزيع الملف إلى كل النظائر يزيد ألف ضعف كذلك.

دعنا نتناول الآن التحليل المناظر في حالة بنية النظائر، حيث يمكن أن يساعد كل نظير الخادم في توزيع الملف. بالتحديد عندما يتسلم نظير بعض بيانات الملف، يمكن أن يستخدم قدرته الخاصة على التحميل لإعادة توزيع البيانات إلى النظائر الأخرى. يُعتبر حساب زمن التوزيع لبنية النظائر أكثر تعقيداً بعض الشيء مقارنة ببنية "زبون/خادم"، لأن زمن التوزيع يعتمد على الكيفية التي يوزّع بها كل نظير أجزاء الملف إلى النظائر الأخرى. على الرغم من ذلك يمكن الحصول على تعبير بسيط للقيمة الصغرى لزمن التوزيع [Kumar 2006]. وسعياً لهذا الهدف دعنا نُبدي الملاحظات التالية أولاً:

- في بداية التوزيع يكون الخادم فقط هو الذي يمتلك الملف. ولتوزيع ذلك الملف إلى مجموعة النظائر، يجب أن يرسل الخادم كل بت من الملف مرة واحدة على الأقل إلى وصلة الوصول للشبكة لديه. وهكذا تكون القيمة الصغرى لزمن التوزيع على الأقل $\frac{F}{u_s}$. (على خلاف بنية "زبون/خادم"، قد لا يحتاج الخادم لإرسال كل بت أكثر من مرة، لأن النظائر قد تعيد توزيع البت فيما بينها).
- كما هو الحال مع بنية "زبون/خادم"، لا يستطيع النظير الذي له أقل معدل تنزيل الحصول على كل بتات الملف (F بت) في أقل من $\frac{F}{d_{\min}}$ ثانية. وهكذا تكون القيمة الصغرى لزمن التوزيع تساوي $\frac{F}{d_{\min}}$ على الأقل.
- أخيراً لاحظ أن قدرة الإرسال الكلية للنظام ككل تساوي معدل التحميل من الخادم بالإضافة إلى معدل إرسال كل فرد من النظائر، أي

$$u_{total} = u_s + u_1 + \ldots + u_N$$

يجب أن يُحمِّل النظام F بت إلى كلٍّ من النظائر التي عددها N، ومن ثم يُحمِّل النظام ما مجموعه NF بت. وهذا لا يمكن تحقيقه بمعدل أسرع من u_{total}. وهكذا تكون القيمة الصغرى لزمن التوزيع أيضاً على الأقل

$$\frac{NF}{u_s + u_1 + \ldots + u_N}$$

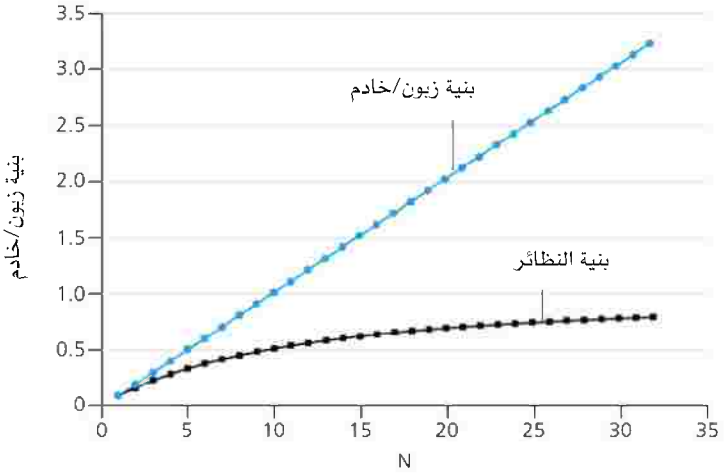
بدمج هذه الملاحظات الثلاث معاً نحصل على القيمة الصغرى لزمن التوزيع لبنية النظائر، (نشير إليه بـ D_{P2P}).

$$D_{P2P} \geq \max\left\{\frac{F}{u_s}, \frac{F}{d_{\min}}, \frac{NF}{u_s + \sum_{i=1}^{N} u_i}\right\}, \quad (2\text{-}2)$$

تعطي هذه المعادلة حداً أدنى للقيمة الصغرى لزمن التوزيع لبنية النظائر. وبالتالي إذا تخيلنا أن كل نظير يمكن أن يعيد توزيع البت بمجرد تسلّمها، فعندئذ توجد طريقة لإعادة التوزيع تحقق هذا الحد الأدنى في واقع الأمر [Kumar 2006]. (سوف نُثبت حالة خاصة لهذه النتيجة في تمارين نهاية الفصل). في الواقع عندما يعاد توزيع قطع من الملف بدلاً من البتات كلٍّ على حدة، تعتبر المعادلة 2-2 تقريباً جيداً للقيمة الفعلية الصغرى لزمن التوزيع. ولذا سنعتبر أن الحد الأدنى المعطى بالمعادلة 2-2 يمثل القيمة الفعلية الصغرى لزمن التوزيع، أي

$$D_{P2P} = \max\left\{\frac{F}{u_s}, \frac{F}{d_{\min}}, \frac{NF}{u_s + \sum_{i=1}^{N} u_i}\right\}, \quad (2\text{-}3)$$

يُقارن الشكل 2-25 القيمة الصغرى لزمن التوزيع لكلٍّ من بنية "زبون/خادم" وبنية النظائرعلى افتراض أن كل النظائر لها نفس معدل التحميل u. في الشكل 2-25 افترضنا أن $\frac{F}{u}$ = 1 ساعة، $u_s = 10u$، $d_{\min} \geq u_s$. أي أنه بوسع النظيرإرسال الملف بكامله في ساعة واحدة، وأن معدل إرسال الخادم يساوي عشر مرات معدل إرسال النظير، وللتبسيط افترضنا أن معدل التنزيل للنظائر كبير بما فيه الكفاية بحيث لا يكون له تأثير. نرى من الشكل 2-25 أن زمن التوزيع يزيد بشكلٍ خطي لبنية "زبون/خادم"، وبدون حد مع زيادة عدد النظائر. ولكن زمن التوزيع الأدنى لبنية النظائر ليس فقط دائماً أقل من زمن التوزيع لبنية "زبون/خادم"، وإنما أيضاً أقل من ساعة واحدة لأي عدد من النظائر N. ومن ثم يمكن أن تكون تطبيقات النظائر ذاتية التوسع بسهولة. إن هذه القدرة على التعامل الجيد مع التوسع هي نتيجة مباشرة لكون النظائر بمثابة نقاط لإعادة توزيع (redistributors) البيانات بالإضافة إلى كونها مستهلكة (consumers) لتلك البيانات في نفس الوقت.



الشكل 2-25 زمن التوزيع لبنية زبون/خادم وبنية النظائر.

بروتوكول BitTorrent

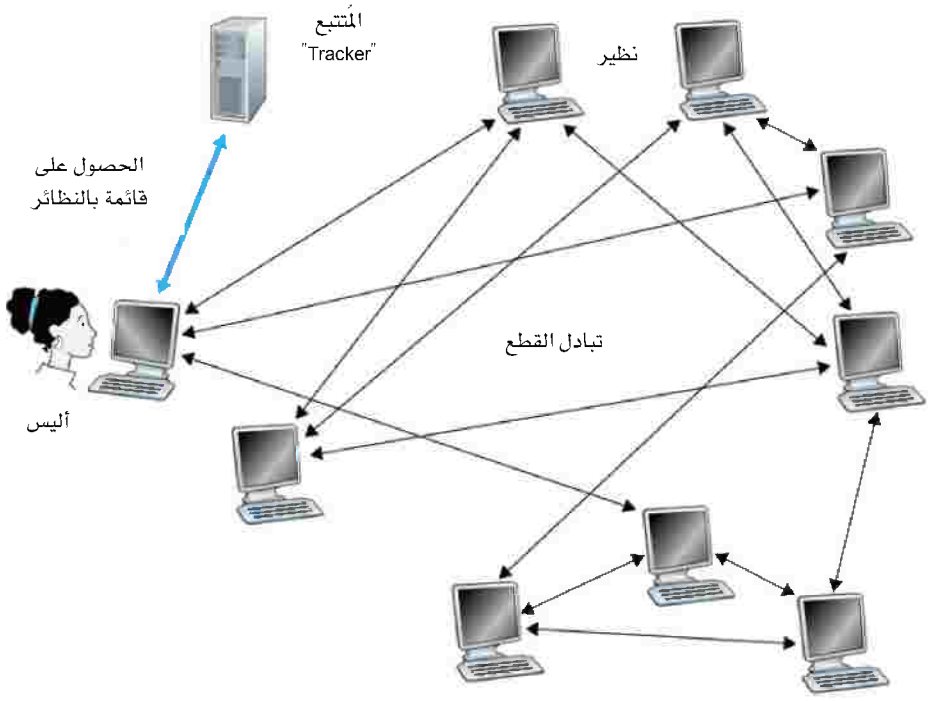
يُعتبر BitTorrent بروتوكول نظائر شائع الاستعمال لتوزيع الملفات [BitTorrent 2007]. في مصطلحات BitTorrent تسمى مجموعة النظائر التي تشارك في توزيع ملف معين "سيلاً" (torrent). تُنزِّل النظائر الموجودة في "السيل" قطعاً متساوية من الملف من أحدها للآخر، في العادة يكون حجم القطعة 256 كيلوبايت. وعندما ينضم نظير إلى "السيل" في البداية، لا يكون لديه قطع من الملف، ولكن بمرورالوقت يقوم بتجميع المزيد والمزيد من القطع. وبينما يُنزّل قطعاً فإنه يُحمِّل أيضاً قطعاً إلى النظائر الأخرى. عندما يحصل نظير على كامل الملف، قَد يغادر السيل (بشكلٍ أناني)، أو يبقى في السيل (بدافع مساعدة الغير) ويُواصل إرسال القطع إلى النظائر الأخرى. أيضاً قد يترك أي نظير السيل في أي وقت بعد تنزيل مجموعة جزئية من القطع فقط، ثم ينضم ثانية إلى السيل لاحقاً.

دعنا الآن نلقي نظرة أكثر تفحصاً على كيفية عمل BitTorrent. نظراً لأن BitTorrent بروتوكول معقّد نوعاً ما، فسوف نصف آلياته الهامة فقط، ونغض

الطرف عن بعض التفاصيل الأخرى لنتمكن من رؤية الغابة من خلال الأشجار. يتضمن كل سيل عقدة بنية تحتية يطلق عليها "المقتفي" (tracker). عندما ينضم نظير إلى السيل، يُسجّل نفسه بالمقتفي ويُخبر المقتفي بشكلٍ دوري بأنه ما زال في السيل. بهذه الطريقة يتابع المقتفي النظائر المشاركة في السيل. وربما يضم سيلٌ ما المئات أو الآلاف من النظائر التي تشارك فيه في وقت من الأوقات.

كما هو موضح في الشكل 2-26، عندما ينضم نظير جديد مثل أليس إلى السيل، يختار المقتفي بشكلٍ عشوائي مجموعة جزئية من النظائر (لنقُل 50 نظيراً مثلاً) من مجموعة النظائر المشاركة، ويُرسل عناوين IP الخاصة بتلك النظائر الـ 50 إلى أليس. مع امتلاك هذه القائمة من النظائر، تحاول أليس إنشاء توصيلات TCP في نفس الوقت مع كل النظائر على تلك القائمة. دعنا ندعو كل النظائر التي نجحت أليس في إنشاء توصيلة معها "النظائر المجاورة". (في الشكل 2-26 لدى أليس ثلاثة نظائر مجاورة فقط، وعادةً يكون لديها أكثر من ذلك). ومع مرور الوقت قد تغادر بعض تلك النظائر بينما قد تحاول نظائر أخرى (خارج الخمسين الأولى) إنشاء توصيلات TCP مع أليس، لذا ستتغير مجموعة النظائر المجاورة بمرور الوقت.

في أي وقت سيكون لدى كل نظير مجموعة جزئية من قطع الملف، ويكون عند النظائر المختلفة مجموعات جزئية مختلفة. وبشكلٍ دوري ستسأل أليس كلاً من نظائرها المجاورة (على توصيلات TCP) عن قائمة القطع التي وصلتهم. وإذا كانت أليس لديها عدة جيران عددهم L، فستحصل على عدد L من قوائم القطع. وبناءً على تلك المعرفة، ستصدر أليس الطلبات (مرة ثانية على توصيلات TCP) للحصول على القطع التي ليست لديها حالياً.



الشكل 2-26 توزيع الملفات باستخدام بروتوكول BitTorrent.

لذلك فإنه في أي لحظة سيكون لدى أليس مجموعة جزئية من القطع وستعرف أي قطع تتوافر عند جيرانها. وبهذه المعلومات سيكون عليها اتخاذ قرارين مهمين هما: ما القطع التي يجب أن تطلبها أولاً من جيرانها؟، وإلى أي جيرانها يجب أن تُرسِل القطع المطلوبة؟ لتحديد القطع التي تطلبها أليس تستخدم أسلوباً يطلق عليه "الأندر أولاً" (rarest first). تتلخص الفكرة في أن تحدد من بين القطع التي ليست لديها القطع "الأندر" بين جيرانها (القطع ذات النسخ الأقل تكرراً بين جيرانها)، وبعد ذلك تطلب تلك القطع أولاً. بهذا الأسلوب يعاد توزيع القطع الأندر بسرعة أكبر، مما يؤدي إلى تساوي أعداد نسخ كل قطعة في السيل تقريباً. ولتحديد أي الطلبات ترد عليها أليس، يستخدم BitTorrent خوارزمية تجارية ذكية تتلخص فكرتها الأساسية في أن تعطي أليس أولوية للجيران الذين

يزودونها بالبيانات حالياً بأعلى معدل. وبشكلٍ محدد تقيس أليس بشكلٍ مستمر المعدل الذي تستلم به البيانات من كلٍّ من جيرانها، ثم تختار النظائر الأربعة التي تغذيها بالبيانات بأعلى معدل، وتتبادل معها القطع. وبعد كل عشر ثوانٍ، تعيد أليس حساب المعدلات ومن المحتمل إجراء تعديل على مجموعة النظائر الأربعة الحالية. من المهم ملاحظة أن كل 30 ثانية تلتقط أليس جاراً إضافياً واحداً بطريقة عشوائية، وترسل له قطعاً من الملف. دعنا نطلق على ذلك النظير المختارعشوائياً بوب. نظراً لأن أليس ترسل البيانات إلى بوب، فقد تصبح واحداً من النظائر الأربعة المُحمِّلة (uploaders) على القمة لدى بوب. وفي هذه الحالة يبدأ بوب بإرسال البيانات إلى أليس. إذا كان المعدل الذي يرسل به بوب البيانات إلى أليس عالياً بما فيه الكفاية، يمكن أن يصبح بوب بدوره واحداً من النظائر الأربعة المُحمِّلة على القمة لدى أليس. أي أن كل 30 ثانية ستختار أليس بشكلٍ عشوائي شريكاً جديداً لتبادل البيانات معه، ثم تستهل التعامل معه. وإذا أصبح النظيران راضيين عن هذا التعامل، فسوف يضع كلٌّ منهم الآخر في قوائم "الأربعة العليا" لديه ويستمر بالتعامل معه إلى أن يجد أحدهما شريكاً أفضل. ونتيجة لذلك تتجه النظائر المتوافقة في معدلات الإرسال إلى العثور على بعضها البعض. في نفس الوقت يسمح اختيار الجار العشوائي أيضاً لنظائر جديدة بالحصول على قطع، ليكون لديهم شيء يتبادلونه. كل النظائر المتجاورة الأخرى غير النظائر الخمسة (أربعة نظائر "عليا" ونظير المجس (probing peer) تصبح "مخنوقة"، بمعنى أنها لا تتلقى أي قطع من أليس .

توجد مشكلة شائعة في مشاركة النظائر للملفات، ألا وهي مشكلة "الركوب المجاني" (free-riding)، وفيها يُنزِّل نظير ملفات من نظام مشاركة الملفات دون أن يرسل هو ملفات. من المفترض أن بروتوكول BitTorrent يتغلب على المشكلة باستخدام خوارزمية "التبادل التجاري"، لأنه لكي تتمكن أليس من تنزيل القطع من بوب بمعدل مقبول لفترة زمنية طويلة، يجب في نفس الوقت أن ترسل القطع إلى بوب بمعدل مقبول. يمتلك BitTorrent عدة آليات مثيرة أخرى لن نناقشها هنا مثل تقطيع الملف إلى أجزاء، والمعالجة بطريقة خط الأنابيب

(pipelining)، والاختيار العشوائي الأول (random first selection)، ونمط نهاية اللعبة (end game mode)، ومانع الوقف (anti-snubbing) [Cohen 2003].

2-6-2 البحث عن معلومات في مجتمع النظائر

يُشكِّل "فهرس المعلومات" أحد المكونات الهامة في العديد من تطبيقات النظائر – بمعنى ربط المعلومات مع مواقع المضيفات. في مثل تلك التطبيقات تقوم النظائر بتعديل الفهرس والبحث فيه بطريقة ديناميكية. نظراً لأن فكرة "ربط المعلومات مع مواقع المضيفات" قد تبدو مجرّدة نوعاً ما، لنُلقِ نظرة على مثالين محددين.

- يوجد في نظام مشاركة النظائر للملفات عادةً عدد كبير من النظائر المشاركة، وكل نظير لديه ملفات يشارك بها كملفات MP3 والفيديو والصور والبرامج. يتضمن نظام مشاركة النظائر للملفات فهرساً يتعقب بطريقة ديناميكية الملفات التي توفرها النظائر للمشاركة. ولكل نسخة من كل ملف متاح للمشاركة بين مجموعة النظائر، يحتفظ الفهرس بسجل يربط بين المعلومات عن النسخة (مثلاً إذا كان ملف MP3 لأغنية، تكون المعلومات عنوان الأغنية، واسم الفنان، وهكذا) وعنوان IP للنظير الذي لديه تلك النسخة. يُعدّل الفهرس بطريقة ديناميكية بينما تأتي النظائر وتروح، وتحصل النظائر على نسخ جديدة من الملفات. على سبيل المثال عندما ينضم نظير إلى النظام يُخبر الفهرس بالملفات الموجودة لديه. وعندما يريد مُستخدِم معين مثل أليس الحصول على ملف معين تبحث في الفهرس لتحديد النظائر التي لديها نسخ من الملف المطلوب. بعد الحصول على مواقع تلك النظائر سيكون بوسع أليس تنزيل الملف منها. وعندما يصبح لديها الملف بكامله، يتم تعديل الفهرس ليشمل نسخة أليس الجديدة من الملف.
- في تطبيق الرسائل الفورية (instant messaging) يوجد فهرس يربط ما بين أسماء المُستخدِمين ومواقعهم (أي عناوين IP لهم). ولفهم أهمية الفهرس في

هذا التطبيق، افترض أن كلاً من المُستخدِمين BeautifulAlice وHandsomeBob موجود على قائمة "الرفقاء" للآخر. عندما يبدأ HandsomeBob برنامج زبون الرسائل الفورية على مضيف بعنوان IP = X، سوف يخبر زبونه الفهرس بأن HandsomeBob على الإنترنت وعنوانه X. لاحقاً عندما تبدأ BeautifulAlice برنامج المراسلة الفورية لديها، حيث إن HandsomeBob على قائمة رفقائها، يبحث الزبون لديها في الفهرس عن HandsomeBob ويكتشف بأن HandsomeBob على الإنترنت على العنوان X. يمكن أن تنشئ BeautifulAlice عندئذ توصيلة TCP إلى المضيف في العنوان X وتبدأ بالمراسلة الفورية مع HandsomeBob. وبالإضافة إلى المراسلة الفورية تستخدم الكثير من التطبيقات الأخرى اليوم فهرساً لتتبع التواجد، بما في ذلك أنظمة هواتف الإنترنت (انظر الجزء 2-6-3).

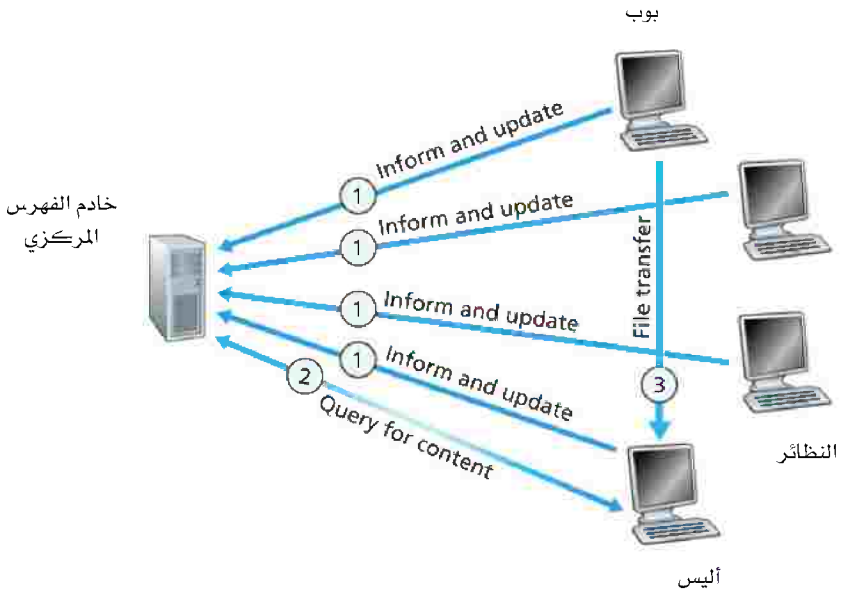
ونذكر باختصار هنا أن نظام BitTorrent يمثل بروتوكولاً لتوزيع الملفات فقط، ولا يوفر أي وظائف لفهرسة الملفات والبحث عنها.

سنناقش فيما يلي ثلاث طرق لتنظيم الفهرس والبحث فيه في مجتمع النظائر. ولنكون أكثر تحديداً، سوف نقوم بذلك في سياق البحث عن ملف في نظام مشاركة النظائر للملفات. ولكننا نؤكد على أن هذه المناقشة تنطبق على حدٍ سواء على البحث عن أي نوع من المعلومات في مجتمع النظائر.

الفهرس المركزي

يعتبر استخدام فهرس مركزي أحد الطرق البسيطة لتحديد مكان ملف (كما في Napster والذي يمثل الاستخدام التجاري الأول لمشاركة النظائر للملفات على نطاق واسع). في هذا التصميم يتم توفير خدمة الفهرس من قِبَل خادم كبير (أو مزرعة خادمات). كما هو موضح في الشكل 2-27، عندما يبدأ مُستخدِم تطبيق مشاركة النظائر للملفات، فإنه يخبر خادم الفهرس بعنوان IP له وأسماء الملفات المتوفرة لديه للمشاركة (على سبيل المثال عناوين كل ملفات MP3 المُخزَّنة عليه). يجمع خادم الفهرس تلك المعلومات من كل نظير فعّال ومن ثم يُنشئ فهرساً

مركزياً ديناميكياً يربط ما بين كل نسخة ملف ومجموعة من عناوين IP. لاحظ أن نظام النظائر لمشاركة الملفات الذي يستخدم فهرساً مركزياً هو في الحقيقة نظام "هجين" من بنية النظائر وبنية زبون/خادم. يستخدم توزيع الملف بنية النظائر ولكن يستخدم البحث عنه بنية "زبون/خادم". يمكن أن توجد مثل هذه البنية المعمارية الهجينة في عدد من التطبيقات اليوم، بما في ذلك العديد من برامج المراسلة الفورية.



الشكل 2-27 الفهرس المركزي.

يُعدُّ استخدام "فهرس مركزي" لتحديد مكان المعلومات طريقة مباشرة ذات مفهوم واضح، غير أن لها عدة عيوب:

- نقطة فشل وحيدة: إذا تعطل خادم الفهرس فإن التطبيق بكامله يتعطل. حتى إذا استخدمنا "مزرعة خادمات"، يمكن أن تتعطل توصيلات الإنترنت إلى مزرعة الخادمات، مما يسبب تعطل التطبيق بأكمله.

- أداء عنق الزجاجة وبنية تحتية مكلفة: في نظام نظائر كبير يضم مئات الآلاف من المُستخدِمين المُوصَّلين، يجب أن يحتفظ الخادم المركزي بفهرس ضخم ويجب أن يرد على آلاف الاستفسارات كل ثانية. في الحقيقة في عام 2000 عندما كان تطبيق Napster الأكثر شعبية عانى من مشاكل ازدحام حركة المرورعند خادمه المركزي.
- انتهاك حقوق النشر: رغم أن هذا الموضوع يقع خارج مجال هذا الكتاب، فإننا نذكر باختصار هنا أن شركات تسجيلات الصوت والفيديو كانت مشغولة وقلقة بشأن استخدام نظام مشاركة النظائر للملفات لأنه يسمح للمُستخدِمين بالحصول بسهولة على "المحتوى المحفوظ الحقوق" مجاناً. ولمناقشة ممتازة حول قوانين حقوق النشر المرتبطة بنظام النظائر، طالع [von Lohmann 2003]. عندما تملك شركة لمشاركة النظائر للملفات خادم فهرس مركزي، قد يؤدي اتخاذ إجراء قانوني ما إلى إغلاق خادم الفهرس. في حين يصعب ذلك في البنية اللامركزية.

فيضان الاستفسار

على الطرف المعاكس للفهارس المركزية توجد الطريقة اللامركزية تماماً وهي "فيضان الاستفسار". وقد تجسدت هذه الطريقة في بروتوكول Gnutella الأصلي، وفيها يكون الفهرس موزعاً بالكامل على مجتمع النظائر حيث يحتوي كل نظير فقط على فهرس للملفات التي يوفرها للمشاركة وليس أي ملفات أخرى.

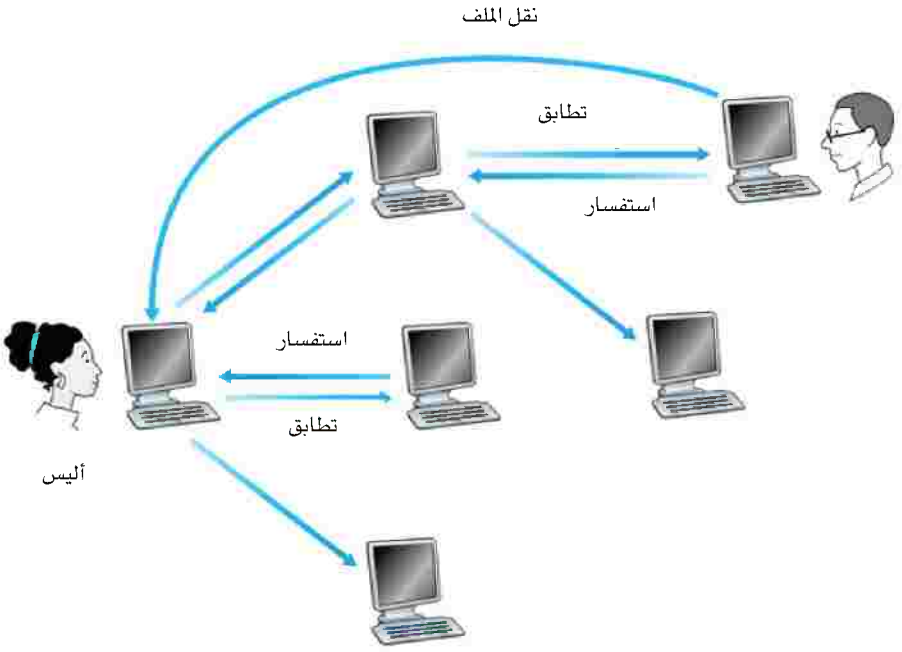
وتُشكِّل النظائر شبكة منطقية مجرّدة يطلق عليها "شبكة إضافية" (overlay network)، ويمكن تعريفها باستخدام مصطلحات "نظرية الأشكال البيانية" (graph theory) كالتالي: إذا كان النظير X يحتفظ بتوصيلة TCP مع نظير آخر Y، نقول بأن هناك "رابط" (edge) بين X وY. ويُعرَف الشكل البياني الذي يضم كل النظائر النشطة وروابط التوصيل (توصيلات TCP الحالية) بالشبكة الإضافية. لاحظ أن "الرابط" ليس وصلة اتصال مادية، وإنما هو وصلة

مجرّدة قد تشمل تحتها عدة وصلات مادية. فعلى سبيل المثال قد يمثل "رابط" في الشبكة الإضافية توصيلة TCP بين نظير في ليثوانيا وآخر في البرازيل.

ورغم أن مثل تلك الشبكة الإضافية قد تتكون من مئات الآلاف من النظائر المشاركة، فإنه في العادة يتصل النظير بعدد قليل من النظائر الأخرى (عادة أقل من عشرة) ضمن الشبكة الإضافية، كما هو موضح في الشكل 2-28. سوف نوضّح لاحقاً كيف يمكن أن تُبنَى الشبكة الإضافية وتعدل بينما تنضم إليها نظائر وتغادر أخرى. دعنا الآن نفترض أن الشبكة الإضافية موجودة ولنركز على كيفية تحديد نظير لمكان محتوى وكيفية حصوله على ذلك المحتوى.

في هذا التصميم ترسل النظائر الرسائل إلى النظائر المجاورة في الشبكة الإضافية على توصيلات TCP الموجودة مسبقاً. مثلاً عندما تريد أليس تحديد مكان ملف "Network Love"، سوف يرسل برنامج الزبون لديها رسالة استفسار تتضمن الكلمات الدليلية "Network Love" إلى كل جيرانها، ومن ثم يسلّم كل واحد منهم الاستفسار إلى كل جيرانه، وهكذا. يوضح الشكل 2-28 عملية "فيضان الاستفسار" هذه. عندما يتلقى نظير استفساراً، يفحص ليرى ما إذا كانت الكلمة الدليلية تطابق أياً من الملفات المتوفرة لديه للمشاركة. وإذا حدث تطابق يرسل النظير رسالة "query-hit" إلى أليس تتضمن اسم وحجم ذلك الملف. تتبع تلك الرسالة الطريق العكسي لرسالة الاستفسار، مستخدمة بذلك توصيلات TCP الموجودة مسبقاً. وبهذا الأسلوب تكتشف أليس النظائر التي تمتلك نسخة من الملف الذي تريده.

رغم أن هذا التصميم اللامركزي بسيط ورائع، إلا أنه عادةً ما يُنتقَد لعدم قابليته للتوسع. وبالتحديد حينما يبدأ نظير استفساراً، يتدفق الاستفسار لكل نظير آخر في الشبكة الإضافية بكاملها، مما يولد كمية هائلة من حركة البيانات بين النظائر في الشبكة التحتية (كالإنترنت) التي تصل ما بين النظائر. طوّر مصممو Gnutella حلاً لهذه المشكلة باستعمال "فيضان الاستفسار محدود المجال". وفيه عندما تبعث أليس رسالة الاستفسار الأولى تُضبَط قيمة حقل عدد



الشكل 2-28 فيضان الاستفسار.

النظائر (peer count) في الرسالة عند حد معين (مثلاً 7). كل مرة تصل رسالة الاستفسار إلى نظير جديد ينقص النظير قيمة هذا الحقل بمقدار واحد قبل إعادة إرساله إلى جيرانه في الشبكة الإضافية. وعندما يستلم نظير استفساراً وتكون قيمة هذا الحقل قد أصبحت "صفراً"، فإنه يتوقف عن إرسال هذا الاستفسار. بهذا الأسلوب يتم حصر الفيضان في منطقة الشبكة الإضافية المحيطة بالنظير الذي بدأ الاستفسار محدود المجال. واضح أن هذا من شأنه الحد من حركة مرور الاستفسارات، لكنه يقلل أيضاً من مجال البحث، وعليه فمن المحتمل ألا يتمكن نظير يريد محتوى معين من تحديد مكانه، بالرغم من وجود ذلك المحتوى في مكان ما في مجتمع النظائر.

تعتبر معالجة النظائر التي تنضم إلى الشبكة أو تغادرها قضية أساسية في الشبكات الإضافية. باستخدام تصميم Gnutella الأصلي كمثال فإن الإجراءات التي تُتخَذ لتعديل الشبكة عندما تنضم نظائر جديدة إليها كالنظير X:

1. يجب أولاً أن يجد النظير X نظيراً آخر موجوداً حالياً في الشبكة الإضافية. لحل معضلة البدء (bootstrap) هذه يمكن أن يحتفظ X بقائمة النظائر (عناوين IP) التي "تعمل" في أغلب الأحيان في الشبكة الإضافية. كبديل لذلك يمكن أن يتصل X بموقع "المقتفي" (tracker) والذي يحتفظ بمثل تلك القائمة (كما في BitTorrent).
2. بمجرد أن يحصل X على مثل تلك القائمة، يحاول بدء توصيلة TCP مع النظائر التي على القائمة الواحد تلو الآخر حتى يتحقق اتصال مع واحد منها، وليكن Y.
3. بعد إنشاء توصيلة TCP بين X وY، يمكن أن يرسل النظير X رسالة "ping" إلى Y تتضمن حقل عدد النظائر. وفور استلام Y لتلك الرسالة، يرسلها إلى كل جيرانه في الشبكة الإضافية. وتواصل النظائر إرسال الرسالة حتى يصبح حقل عدد النظائر صفراً.
4. وحينما يستلم نظير Z رسالة البينج، يرد بإرسال رسالة "pong" (تتضمن عنوان IP لـ Z) خلال الشبكة الإضافية إلى X.
5. بعد استلام X رسائل pong، يعرف عناوين IP للعديد من النظائر في الشبكة الإضافية. عندئذ يمكنه أن يبدأ توصيلات TCP ببعض تلك النظائر الأخرى، وبذلك يُنشئ "روابط" متعددة بينه وبين الشبكة الإضافية.

سوف نستكشف الإجراءات التي يمكن أن تتخذها الشبكة الإضافية فور مغادرة نظير في تمارين نهاية الفصل.

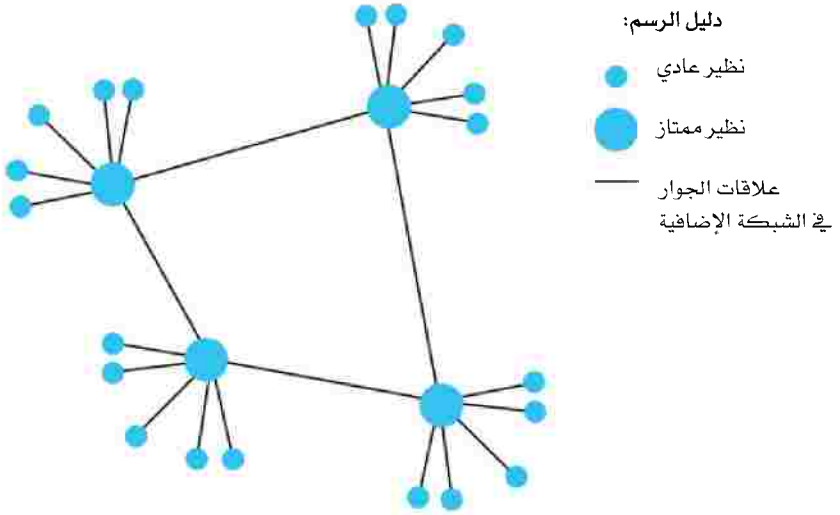
حتى الآن غطينا الخصائص الضرورية لفيضان الاستفسار وبناء الشبكة الإضافية ديناميكياً. وباختصار فإن "فيضان الاستفسار" أسلوب بسيط وموزّع بين النظائر يسمح لمُستخدِم بالسؤال عن المعلومات التي توجد في النظائر القريبة (أي

ضمن عدد صغير من القفزات (hops) في الشبكة الإضافية). طبّق تصميم بروتوكول Gnutella الأصلي "فيضان الاستفسار" كما وصفنا أعلاه. وعلى مرِّ السنين تطوّر البروتوكول بشكلٍ ملحوظ، والآن يستغل عدم تجانس النظائر في نظام النظائر لمشاركة الملفات. ويبقى بروتوكول Gnutella شائعاً جداً اليوم، كما أنه يُستخدَم في زبون نظام النظائرالشائع LimeWire.

البناء الهرمي للشبكات الإضافية

عرفنا أن "الفهرس المركزي" و"فيضان الاستفسار" طريقتان متعارضتان تماماً لتحديد مكان المعلومات. هناك طريقة ثالثة سوف نشير إليها "بالتصميم الهرمي للشبكات الإضافية"، وهي تجمع أفضل ميزات الطريقتين السابقتين. ابتُكر "التصميم الهرمي للشبكات الإضافية" أولاً من قِبَل FastTrack (وهو نظام نظائر لمشاركة الملفات استخدم في عدد من التطبيقات على مرِّ السنين، مثل Kazaa وMorpheus). كما يستخدمه أيضاً نظام Gnutella الحديث، رغم أنه يختلف عن النظام الموصوف هنا.

كما هو الحال مع "فيضان الاستفسار" لا يكرس "التصميم الهرمي للشبكات الإضافية" خادماً (أو مزرعة خادمات) لتتبع وفهرسة الملفات. ومع ذلك فبخلاف فيضان الاستفسار ليست كل النظائر متماثلة في التصميم الهرمي للشبكات الإضافية. بالتحديد يطلق على النظائر المتاحة غالباً والتي تتصل بالإنترنت بوصلات لها سعة إرسال عالية نظائر ممتازة (عليا) (super peers) ويكون لها مسؤوليات أكبر. وكما هو موضح في الشكل 2-29، إذا لم يكن النظير "نظيراً ممتازاً"، فإنه يكون "نظيراً عادياً" وينسب ابناً لنظير ممتاز. قد يكون للنظير الممتاز بضعة مئات من النظائر العادية كأبناء.



الشكل 2-29 البناء الهرمي للشبكات الإضافية.

يُنشئ النظير الجديد أولاً توصيلة TCP بأحد النظائر الممتازة، ثم يُخطره بكل الملفات التي يوفرها للمشاركة. يسمح ذلك للنظير الممتاز بالاحتفاظ بفهرس يتضمن هوية كل ملفات النظائر الأبناء له، كما يتضمن بيانات ما وراء البيانات (meta-data) (أي بيانات عن بيانات أخرى) حول تلك الملفات، وعناوين IP المناظرة للأبناء الذين لديهم تلك الملفات. بهذه الطريقة يصبح كل نظير ممتاز فهرساً "مصغراً" (mini-index).

بالمقارنة مع الفهرس المركزي الذي ناقشناه في بداية هذا الجزء، لا يعتبر النظير الممتاز خادماً مكرّساً، وإنما هو نظير عادي، وعادة ما يوجد في حي سكني أو حرم جامعي. وإذا ما عزل كل نظير ممتاز وأبناؤه عن الشبكة فإن ذلك سيحد من كمية المحتوى المتوفرة لأي نظير بشكلٍ ملحوظ. لعلاج هذا التقييد ترتبط النظائر الممتازة فيما بينها بتوصيلات TCP لتكوين شبكة إضافية بينها. وبهذا يمكن أن ترسل النظائر الممتازة الاستفسارات إلى النظائر

الممتازة المجاورة. هذه النظرة مشابهة لفيضان الاستفسار، لكنه فيضان محدود المجال يتم داخل الشبكة الإضافية بين النظائر الممتازة.

عندما يريد نظير إجراء بحث باستخدام كلمات دليلية (keywords)، يرسل استفساراً يتضمن تلك الكلمات إلى نظيره الممتاز. فيرد النظير الممتاز بعناوين IP لنظائره الأبناء الذين لديهم ملفات تتوافق موصّفاتها (descriptors) مع تلك الكلمات الدليلية (مع مُعرِّفات (identifiers) لتلك الملفات). قد يرسل النظير الممتاز الاستفسار أيضاً إلى واحد أو أكثر من النظائر الممتازة الأخرى المجاورة. إذا تلقى نظير مجاور مثل هذا الاستفسار، فإنه يرد أيضاً بعناوين IP لنظائره الأبناء الذين لديهم ملفات مطابقة. تتبع الردود من النظائر الممتازة الطريق العكسي في الشبكة الإضافية.

يستغل التصميم الهرمي للشبكات الإضافية عدم التجانس الطبيعي للنظائر بتعيين عدد قليل من النظائر ذات القدرات الأكبر كنظائر ممتازة، لتكوين الطبقة العليا (top tier) لهرم الشبكة الإضافية، كما هو موضح في الشكل 2-29. بالمقارنة مع فيضان الاستفسار محدود المجال (كما في تصميم Gnutella الأصلي)، يسمح التصميم الهرمي للشبكات الإضافية بفحص "التطابق" مع عدد كبير جداً من النظائر، بدون توليد كمية مفرطة من حركة مرور الاستفسار [Liang 2005].

قبل أن ننهي مناقشتنا للبحث عن المعلومات في تطبيقات النظائر، نذكر باختصار تصميماً مهماً آخر يطلق عليه الجدول الهاش المُوزَّع (Distributed (DHT) Hash Table) [Stoica 2001; Rowstron 2001; Ratnasamy 2001; Zhao 2004; Maymounkov 2002; Garces-Erce 2003]. إن المناقشة المستفيضة لهذا الموضوع تقع خارج مجال هذا الكتاب. لكننا نذكر هنا أن (1) يُنشئ DHT فهرساً غير مركزي تماماً يربط ما بين مُعرِّفات الملفات ومواقعها، (2) يسمح للمُستخدِم بتحديد كل مواقع الملف (من حيث المبدأ) بدون توليد كمية مفرطة من حركة مرورالبحث. وقد لاقى DHT اهتماماً كبيراً في المحيط البحثي، وهو مستخدم في

Overnet والذي يمثل جزءاً محورياً من تطبيق مشاركة الملفات الشائع eMule [Liang 2006].

3-6-2 دراسة حالة: الاتصال الهاتفي للنظائر عبر الإنترنت باستخدام سكايب (Skype)

يُعتبر سكايب تطبيق نظائر شائع الاستخدام بشكلٍ كبير، حيث يستخدمه في أغلب الأحيان من سبعة إلى ثمانية ملايين مُستخدِم متصلون به في أي لحظة. وبالإضافة إلى أنه يوفر خدمة اتصال هاتفي عبر الإنترنت بين أجهزة الحاسب (PC-to-PC)، فهو يقدم خدمات الاتصال الهاتفي من الحاسب للهاتف (PC-to-Phone)، والاتصال الهاتفي من الهاتف للحاسب (Phone-to-PC)، ومؤتمرات الفيديو بين أجهزة الحاسب (PC-to-PC). قام بتأسيس سكايب نفس الأشخاص الذين بنوا FastTrack وKazaa وقد اشترته eBay عام 2005 مقابل 2.6 بليون دولار.

يستخدم سكايب تقنيات النظائر في عدد من الطرق الإبداعية، ويُصوّر بشكلٍ رائع كيف يمكن استخدام النظائر في التطبيقات التي تتجاوز "توزيع المحتوى" و"مشاركة الملفات". كما هو الحال مع المراسلة الفورية، فإن الاتصال الهاتفي عبر الإنترنت PC-to-PC هو في الأصل نظام نظائر، ذلك لأنه في قلب التطبيق يتصل أزواج من المُستخدِمين (وبمعنى آخر النظائر) مع بعضهم في الوقت الحقيقي. لكن سكايب يستخدم أيضاً تقنيات النظائر لوظيفتين مهمتين أخريين: تحديد موقع المُستخدِم واجتياز الـ NAT.

ليست بروتوكولات سكايب فقط ذات ملكية خاصة، ولكن أيضاً كل رزم سكايب المُرسَلة (رزم الصوت والتحكم) مشفّرة تشفيراً سرياً. وعلى الرغم من ذلك فمن خلال موقع الويب الخاص به وعدد من دراسات القياسات تعلم الباحثون كيف يعمل سكايب عموماً [Baset 2006; Guha 2006; Chen 2006; Suh 2006; Ren 2006]. وكما هو الحال مع FastTrack، تنتظم العُقَد في Skype في ترتيب هرمي للشبكة الإضافية، وكل نظير مصنّف كنظير ممتاز أو نظير عادي. يتضمن Skype فهرساً يربط أسماء مُستخدِمي سكايب مع عناوين IP

الحالية (وأرقام المنافذ)، وهذا الفهرس موزّع على النظائر الممتازة. عندما تريد أليس محادثة بوب، فسوف يبحث زبون Skype لديها في الفهرس الموزّع لتحديد عنوان IP الحالي لبوب. ولأن نظام سكايب ذو ملكية خاصة (proprietary) فهو حالياً لا يوضّح كيف ينظم الفهرس عبر النظائر الممتازة، بالرغم من أن نوعاً ما من تنظيمات DHT محتملة جداً.

تُستخدَم تقنيات النظائر أيضاً في مرحِّلات سكايب (Skype Relays) لعمل مكالمات (calls) بين المضيفات في الشبكات المنزلية. توفر العديد من ترتيبات الشبكات المنزلية اتصالاً بالإنترنت من خلال "موجّه" (عادة يكون موجّهاً لاسلكياً). وهذه "الموجّهات" في الحقيقة أكثر من كونها "موجّهات"، فهي تتضمن عادة ما يطلق عليه مترجم عناوين الشبكة (Network Address (NAT) Translator). وسوف ندرس NAT بالتفصيل في الفصل الرابع. كل ما نحتاج لمعرفته هنا هو أن NAT يمنع مضيفاً من خارج الشبكة المنزلية من بدء الاتصال مع مضيف ضمن الشبكة المنزلية. إذا كان كلا الشخصين المتصلين من خلال سكايب لديهما NAT، فهناك مشكلة حيث لا يستطيع أحدهما تلبية نداء بدأه الشخص الآخر، وبالتالي يبدو الاتصال مستحيلاً. إلا أن الاستعمال الذكي للنظائر الممتازة (super peers) والمُرحِّلات (relays) يحل هذه المشكلة بشكلٍ رائع. افترض أنه عندما تبدأ أليس الاتصال تُنسَب لأحد النظائر الممتازة خارج NAT. تستطيع أليس أن تبدأ جلسة مع نظيرها الممتاز، لأن مترجم عنوان شبكتها NAT يرفض الجلسات التي تبدأ من خارج شبكة منزلها، وهذا يسمح لها ولنظيرها الممتاز بتبادل "رسائل تحكم" على هذه الجلسة. يحدث نفس الشيء لدى بوب. الآن وعندما تريد أليس محادثة بوب، تُخطِر نظيرها الممتاز، والذي يُخطِر بدوره نظير بوب الممتاز، والذي يقوم بعد ذلك بإخطار بوب بالدعوة القادمة من أليس. إذا قَبِل بوب الدعوة لمحادثة أليس، يختار النظيران الممتازان نظيراً ممتازاً ثالثاً خارج NAT (يطلق عليه المرحِّل) تكون وظيفته نقل البيانات بين أليس وبوب. عندئذ يُخطِر النظيران الممتازان أليس وبوب لبدء الاتصال عبر المرحِّل. ترسل أليس حزم بيانات الصوت إلى المرحِّل على التوصيلة Alice-to-relay (والتي أنشئت من

قِبَل أليس)، ومن ثم يُرسِل المرحِّل تلك الرزم على التوصيلة relay-to-Bob (والتي أنشئت من قِبَل بوب) إلى بوب. تتدفق الرزم من بوب إلى أليس في الاتجاه المعاكس على نفس توصيلتي الترحيل هاتين. وبهذه الطريقة ينجح بوب وأليس في الحصول على توصيلة حسب الطلب من طرف إلى طرف رغم أنه لا يمكن لأحدهما أن يقبل جلسة يتم إنشاؤها من خارج شبكة الاتصالات المحلية LAN الخاصة به. يوضح استخدام المُرحِّلات التصميم المتطوّر جداً لأنظمة النظائر، حيث تؤدّي النظائر خدمات نظام رئيسة للآخرين (كخدمات الفهرسة والترحيل)، بينما تستخدم هي نفسها في الوقت ذاته خدمات المُستخدِم النهائي (كتحميل الملفات وهاتف الإنترنت) والمتاحة من نظام النظائر.

يعتبر سكايب تطبيق إنترنت ناجح وواسع الانتشار جداً، حيث امتدت خدماته لتشمل عشرات الملايين من المُستخدِمين. إن التبني السريع والواسع الانتشار بشكلٍ مدهش لسكايب، بالإضافة إلى مشاركة النظائر للملفات والويب والمراسلة الفورية قبل ذلك، لشاهد على حكمة التصميم المعماري العام للإنترنت. ذلك التصميم الذي لم يكن له أن يتوقع تطبيقات الإنترنت الغنية والدائمة التوسع التي تم تطويرها على مدى السنوات الثلاثين التالية لظهوره. إن الخدمات التي توفرها الشبكة لتطبيقات الإنترنت (كالنقل اللاتوصيلي لوحدات البيانات (من خلال UDP)، والنقل التوصيلي الموثوق لوحدات البيانات (من خلال TCP)، وواجهة برمجة المقابس (socket programming interface)، والعنونة (addressing)، ودليل أسماء النطاقات (DNS)، وغيرها من الخدمات الأخرى) قد أثبتت أنها كافية للسماح بتطوير الآلاف من التطبيقات. ونظراً لأن كل تلك التطبيقات تعمل فوق الطبقات الأربع السفلى الحالية من رصّة بروتوكولات الإنترنت، فإنها تتطلب فقط تطوير برامج "زبون/خادم" أو "نظير لنظير" جديدة تستخدم على الأنظمة الطرفية، مما ساعد على انتشار تلك التطبيقات وتبنيها بسرعة.

7-2 برمجة مقابس بروتوكول TCP

الآن وبعد أن استعرضنا عدداً من تطبيقات الشبكة الهامة، دعنا نستكشف كيف تُكتَب برامج تطبيقات الشبكة في الواقع. في هذا الجزء سنكتب برامج تطبيقات تستخدم بروتوكول TCP؛ وفي الجزء التالي سنكتب برامج تطبيقات تستخدم بروتوكول UDP.

تذكّر من الجزء 2-1 أن العديد من تطبيقات الشبكة تتكون من زوج من البرامج – برنامج زبون وبرنامج خادم – موجودين على نظامين طرفيين مختلفين. وعند تشغيل هذين البرنامجين، يتم إنشاء عملية زبون وعملية خادم، وتتصل هاتان العمليتان فيما بينهما عن طريق الكتابة إلى المقابس والقراءة منها. عند تطوير تطبيق للشبكة تتلخص المهمّة الرئيسة لمطوّر التطبيق في كتابة الكود لبرنامجي الخادم والزبون .

يوجد نوعان من تطبيقات الشبكة. نوع يطبق معيار بروتوكول (قياسي) مُعرَّف على سبيل المثال في طلبات التعليقات (RFCs). ولهذا التطبيق يجب أن يتوافق كلٌّ من برنامجي الخادم والزبون مع القواعد التي يمليها طلب التعليقات. على سبيل المثال يمكن أن يكون برنامج الزبون تحقيقاً لجانب الزبون لبروتوكول FTP (والذي تناولناه في الجزء 2-3 والمُعرّف بشكلٍ واضح في طلب التعليقات RFC 959)، وبنفس الطريقة يمكن أن يكون برنامج الخادم تحقيقاً لجانب خادم FTP (والمُعرّف أيضاً بشكلٍ واضح في طلب التعليقات RFC 959). وإذا كتب أحد مطوّري التطبيقات الكود لبرنامج الزبون، وكتب مطوّر آخر مستقل الكود لبرنامج الخادم، واتبع كلا المطوّرين قواعد RFC بعناية، فسوف يكون البرنامجان قادرين على التعامل مع بعضهما البعض. وفي الحقيقة يتضمن العديد من تطبيقات الشبكة اليوم اتصالاً بين برامج الزبون والخادم والتي طورت من قِبَل مطوّرين مستقلين. على سبيل المثال يتصل متصفّح Firefox بخادم ويب Apache، أو يُحمِّل برنامج زبون FTP (موجود على جهاز حاسب شخصي) ملفاً إلى خادم لاينكس FTP. عندما يطبق زبون أو خادم بروتوكولاً مُعرّفاً في RFC، يجب

استخدام رقم المنفذ المقترن بالبروتوكول. (نوقشت أرقام المنافذ باختصار في الجزء 2-1، وسوف تُغطى بتفصيل أكثر في الفصل الثالث). النوع الآخر لتطبيق الشبكة هو تطبيق ذو ملكية خاصة (proprietary)، وفي هذه الحالة لا يتوافق بروتوكول طبقة التطبيقات المستخدم من قِبَل برامج الخادم والزبون بالضرورة مع طلبات تعليقات موجودة حالياً. عند تطوير كلا البرنامجين للخادم والزبون يكون للمطوّر سيطرة كاملة على ما يكتب في الكود. ولكن نظراً لأن الكود لا يُطبّق بروتوكول ذا ملكية عامة (public-domain protocol)، فلن يستطيع مطوّرون مستقلون آخرون تطوير كود آخر يمكنه التعامل مع ذلك التطبيق. وعند تطوير تطبيق ذي ملكية خاصة يجب أن يتوخى المطوّر الحذر وألا يستخدم أحد أرقام المنافذ المشهورة والمُعرّفة في طلبات التعليقات.

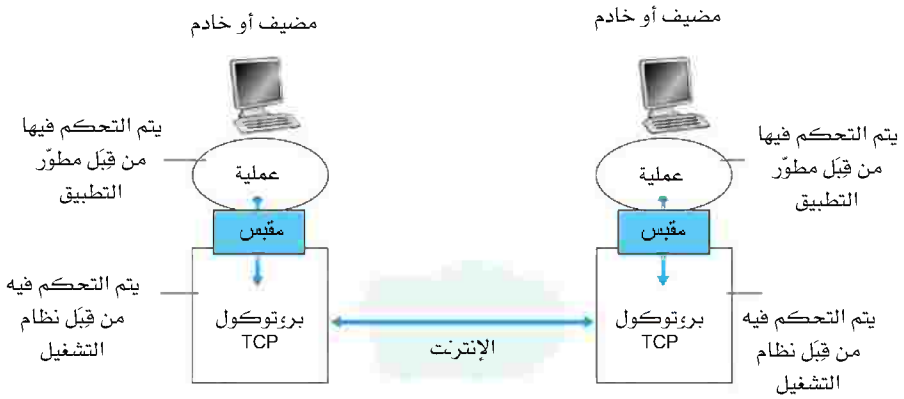
في هذا الجزء والجزء التالي سوف نتناول القضايا الرئيسة في تطوير تطبيق "زبون/خادم" خاص. من أول القرارات التي يجب أن يتخذها المطوّر هو "هل يعمل التطبيق على بروتوكول TCP أو على بروتوكول UDP؟". تذكّر أن بروتوكول TCP توصيلي ويوفر قناة مجرى للتدفق الموثوق لبايتات البيانات (reliable byte-stream channel) بين نظامين طرفيين. أما بروتوكول UDP فغير توصيلي ويرسل رزم بيانات مستقلة من نظام طرفي إلى آخر بدون أي ضمانات فيما يتعلق بالتوصيل.

في هذا الجزء سنُطوِّر تطبيق زبون بسيط يعمل على بروتوكول TCP، وفي الجزء التالي سنُطوّر برنامج زبون بسيط يعمل على بروتوكول UDP. وسوف نكتب هذه البرامج البسيطة بلغة البرمجة جافا. رغم أنه كان من الممكن أن نكتب الكود بلغة C أو لغة C++، لكننا اخترنا لغة جافا لأن التطبيقات تكتب فيها على نحو نظيف وبشكلٍ أكثر تنسيقاً. فمع لغة جافا توجد سطور أقل في كود البرنامج، وكل سطر يسهل توضيحه للمبرمج المبتدئ بدون صعوبة كبيرة. لكن ليس هناك حاجة لكي تخاف إذا كنت غير ملم بلغة جافا. ستصبح قادراً على تتبع كود جافا إذا كان لديك خبرة في البرمجة بلغة أخرى. وتوجد عدة

مراجع جيدة للقرّاء المهتمّين ببرمجة تطبيقات زبون/خادم بلغة C [Donahoo 2001; Stevens 1997; Frost 1994; Kurose 1996].

1-7-2 برمجة مقابس TCP

ذكرنا في الجزء 1-2 أن العمليات تنفذ على حاسبات مختلفة تتصل فيما بينها بإرسال الرسائل إلى المقابس. وقلنا أن كل عملية تناظر البيت ومقبس العملية يناظر باب البيت. وكما هو موضح في الشكل 2-30 يُعتبر المقبس بمثابة الباب بين العملية وبروتوكول TCP. ولمطوّر البرنامج سيطرة كاملة على كل شيء على جانب طبقة التطبيقات من المقبس، ولكن ليس له سيطرة تذكر على جانب طبقة النقل من المقبس (فعلى الأكثر يكون له القدرة على ضبط بضعة بارامترات لبروتوكول TCP كالحجم الأقصى للذاكرة المؤقتة (buffer) والحجم الأقصى لقطعة بيانات TCP).



الشكل 2-30 اتصال العمليات عن طريق مقابس TCP.

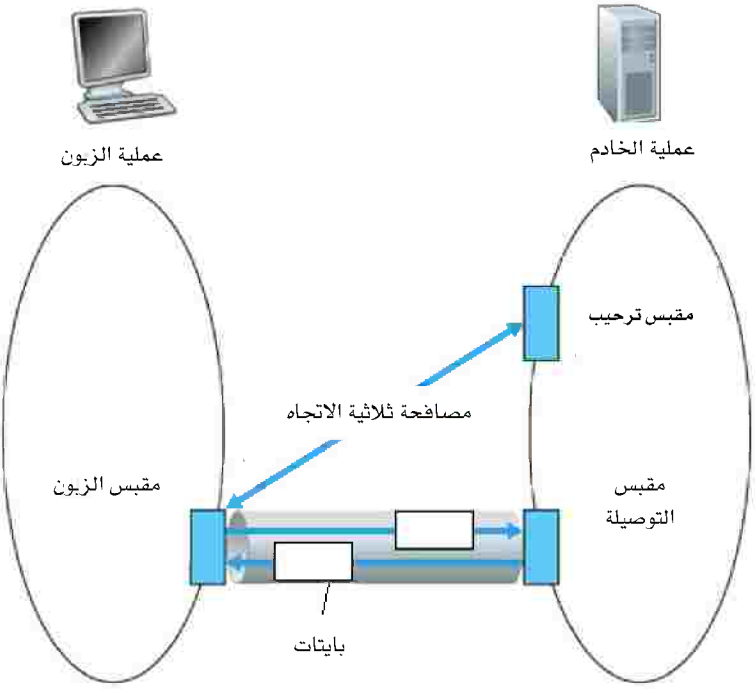
دعنا الآن نلقي نظرة أدق على التفاعل ما بين برنامجي الخادم والزبون. يتحمل الزبون مسؤولية بدء الاتصال بالخادم، ولكي يتمكّن الخادم من الرد على اتصالِ الزبون الأولي، يجب أن يكون الخادم جاهزاً، وهذا يتطلب شيئين. أولاً: لا

يمكن أن يكون برنامج الخادم خاملاً (dormant)، فعليه أن يكون شغّالاً قبل أن تحاول عملية الزبون بدء الاتصال. ثانياً: يجب أن يكون لدى برنامج الخادم نوع من الباب (وبدقة أكثر مقبس) يُرحّب ببعض الاتصال الأولي من عملية زبون تنفّذ على مضيف اعتباطي. باستخدام التناظر بين "بيت ↔ عملية" و"باب ↔ مقبس" سنشير أحياناً إلى اتصال الزبون الأولي على أنه "طرْق على باب الترحيب".

عندما تكون عملية الخادم شغّالة، يمكن أن تبدأ عملية الزبون توصيلة TCP إلى الخادم. وهذا يتم في برنامج الزبون بإنشاء مقبس. عندما يُنشئ زبونٌ ما مقبساً، يحدد عنوان عملية الخادم الذي سيتم الاتصال به (والذي يتضمن عنوان IP لمضيف الخادم ورقم منفذ عملية الخادم). وبمجرد إنشاء المقبس في برنامج الزبون، يقوم TCP في الزبون ببدء مصافحة ثلاثية (3-way handshaking) ويُنشئ توصيلة TCP إلى الخادم. إن المصافحة الثلاثية التي تحدث في طبقة النقل غير مرئية تماماً لبرامج الخادم والزبون.

أثناء المصافحة الثلاثية تطرق عملية الزبون على باب الترحيب لعملية الخادم. وعندما "يسمع" الخادم الطرْق، يُنشئ باباً جديداً (بالأحرى مقبساً جديداً) يُكرَّس لذلك الزبون. في المثال الذي سنتاوله باب الترحيب هو كائن من نوع ServerSoeket ونطلق عليه welcomeSocket. وعندما يطرق زبونٌ ما على هذا الباب، يستدعي البرنامج الوظيفة welcomeSocket.accept()، والتي تنشئ باباً جديداً للزبون. في نهاية مرحلة المصافحة تؤسس توصيلة TCP بين مقبس الزبون ومقبس الخادم الجديد. ومن الآن فصاعداً سنشير إلى مقبس الخادم الجديد بمقبس توصيلة الخادم (connection socket). وهذه المقابس موضحة في الشكل 2-31.

من منظور التطبيق تمثل توصيلة TCP أنبوباً افتراضياً بين مقبس الزبون ومقبس توصيلة الخادم. يمكن أن ترسل عملية الزبون بايتات اعتباطيةَ إلى مقبسها، وسوف يضمن TCP توصيل كل بايت أُرسِل في الطلب إلى عملية الخادم (خلال مقبس التوصيل) بنفس ترتيب إرساله. وهكذا يوفر TCP خدمة مجرى



الشكل 2-31 ثلاث مقابس: مقبس الزبون، ومقبس الترحيب، ومقبس التوصيلة.

للتدفق الموثوق لبايتات البيانات بين عمليتي الخادم والزبون. علاوة على ذلك (وكما يمكن للناس الدخول والخروج من نفس الباب) فإن عملية الزبون لا ترسل البايتات فقط إلى مقبسها، ولكنها تستلم البايتات أيضاً من نفس المقبس. بنفس الطريقة فإن عملية الخادم لا تستلم البايتات فقط من مقبس توصيلتها، ولكنها ترسل البايتات أيضاً عبر ذلك المقبس. ولما كانت المقابس تلعب دوراً محورياً في تطوير تطبيقات زبون/خادم فإن عملية التطوير تُعرَف ببرمجة المقابس (socket programming).

قبل إعطاء مثال لتطبيق زبون/خادم، من المفيد مناقشة المصطلح "مجرى" (stream)، فهو عبارة عن سيل من الحروف يتدفق داخل أو خارج عملية. وكل "مجرى" إما أن يكون "مجرى إدخال" (input stream) أو "مجرى إخراج" (output

(stream للعملية. يرتبط مجرى الإدخال بوحدة إدخال للعملية كوحدة الإدخال القياسية (لوحة المفاتيح (keyboard)) أو كمقبس تتدفق البيانات عبره من الإنترنت، في حين يرتبط مجرى الإخراج بوحدة إخراج للعملية كوحدة الإخراج القياسية (شاشة العرض (monitor)) أوكمقبس تتدفق البيانات عبره إلى الإنترنت.

2-7-2 مثال لتطبيق "زبون/خادم" بلغة جافا

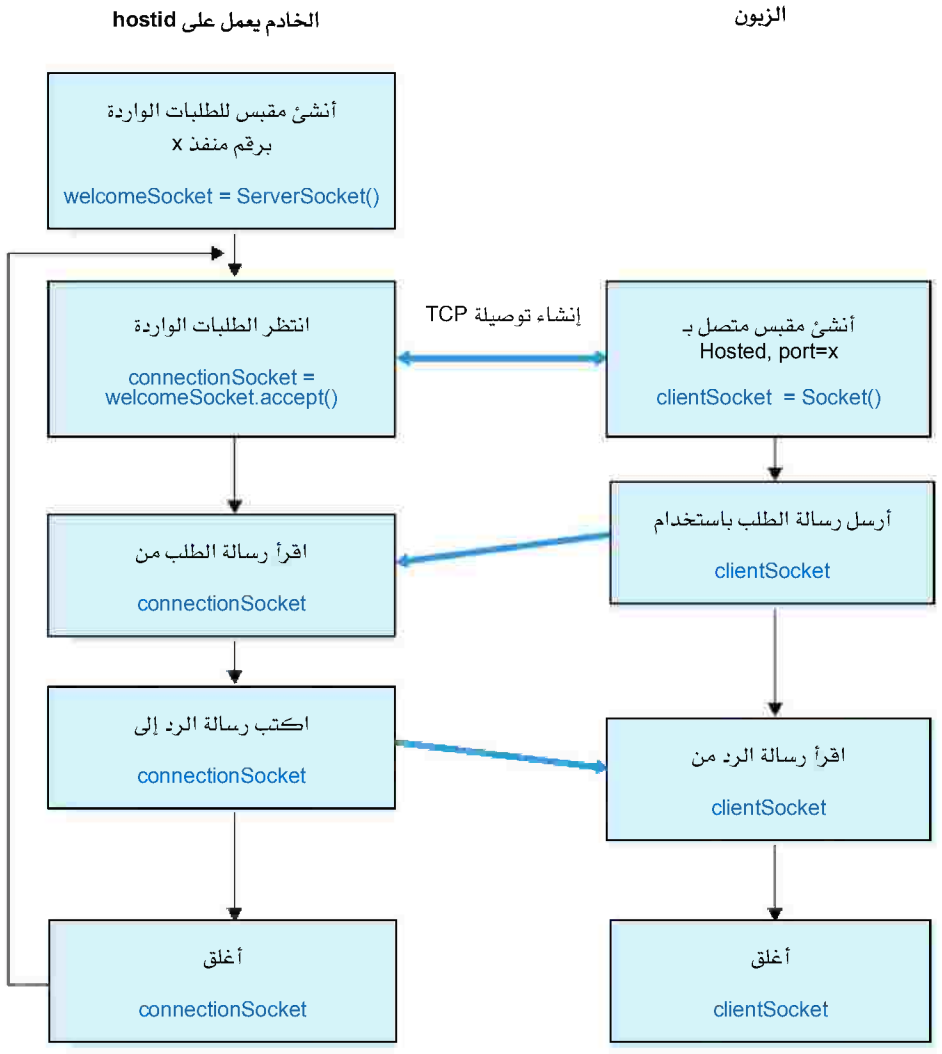
سنستخدم تطبيق زبون/خادم البسيط التالي لتوضيح برمجة المقابس لكلٍّ من TCP وUDP:

1. يقرأ الزبون سطراً من وحدة الإدخال القياسية (لوحة المفاتيح) ويرسله خارج مقبسه إلى الخادم .
2. يقرأ الخادم السطر من مقبس توصيلته مع الزبون.
3. يُحوِّل الخادم حروف السطر إلى حروف كبيرة (uppercase).
4. يُرسِل الخادم السطر المُعدَّل خارج مقبس توصيلته إلى الزبون.
5. يقرأ الزبون السطر المُعدَّل من مقبسه ويعرضه على وحدة الإخراج القياسية (شاشة العرض).

يبين الشكل 2-32 الأنشطة الرئيسة المتعلقة بالمقبس لدى كلٍّ من الزبون والخادم.

نورد فيما يلي زوج البرامج للزبون والخادم مبني على بروتوكول TCP، مع تحليل مفصّل (سطر بسطر) لكل برنامج. يسمى برنامج الزبون TCPClient.java، ويسمى برنامج الخادم TCPServer.java. ولكي نؤكّد على القضايا الرئيسة، سنعطي عمداً كوداً واضحاً للغاية ويفي بالغرض لكنه غير مثالي بالضرورة. سوف يحتوي "الكود الجيد" بالتأكيد على بضعة سطور إضافية للمساعدة. وبمجرد إتمام عملية الترجمة (compilation) للبرنامجين على مضيفاتهم الخاصة، يُنفَّذ برنامج الخادم أولاً في مضيف الخادم ليُنشئ عملية خادم. كما ناقشنا سابقاً تنتظر عملية الخادم لكي تتصل بها عملية زبون. في هذا المثال عندما يُنفّذ برنامج الزبون يتم إنشاء عملية في مضيف الزبون، وهذه العملية تتصل بالخادم فوراً

وتنشئ توصيلة TCP معه. بعد ذلك يمكن للمُستخدِم عند الزبون استخدام التطبيق لإرسال سطر وبعد ذلك يتلقى نسخة من السطر المعدّل من الخادم.



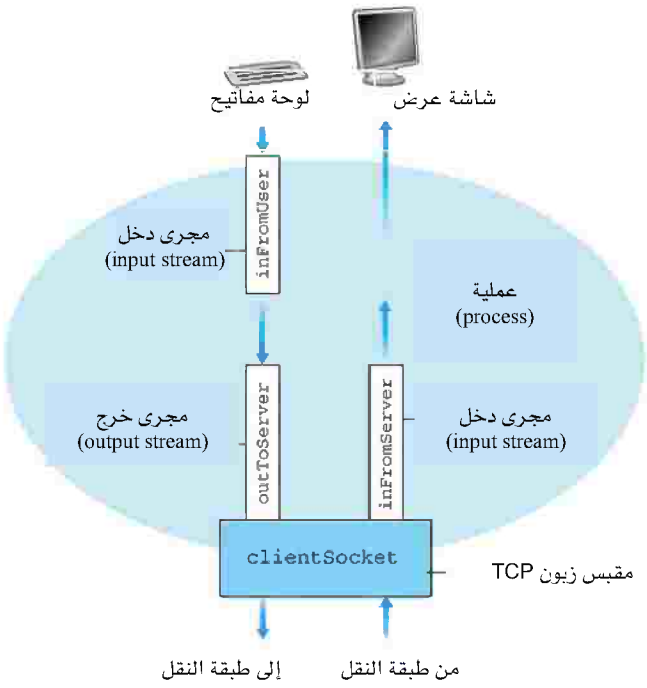
الشكل 2-32 تطبيق "زبون/خادم" باستخدام خدمة النقل التوصيلية (TCP).

برنامج الزبون TCPClient.java

فيما يلي الكود الخاص بجانب الزبون من التطبيق:

```
import java.io.*;
import java.net.*;
class TCPClient {
    public static void main(String argv[]) throws Exception {
        String sentence;
        String modifiedSentence;
        BufferedReader inFromUser = new new BufferedReader(new
        InputStreamReader(System.in));
        Socket clientSocket = new Socket("hostname", 6789);
        DataOutputStream outToServer = new DataOutputStream(
        clientSocket.getOutputStream());
        BufferedReader inFromServer = new BufferedReader(new
        InputStreamReader(clientSocket.getInputStream()));
        sentence = inFromUser.readLine();
        outToServer.writeBytes(sentence+'\n');
        modifiedSentence = inFromServer.readLine();
        System.out.println("FROM SERVER: "+modifiedSentence);
        clientSocket.close();
    }
}
```

يُنشئ برنامج TCPClient.java مقبساً واحداً وثلاثة مجارٍ (streams) كما هو موضح في الشكل 2-33. يُسمى المقبس clientSocket. يُستخدَم المجرى inFromUser لمدخلات البرنامج ويرتبط بوحدة الإدخال القياسية (لوحة المفاتيح).عندما يكتب المُستخدِم حروفاً على لوحة المفاتيح، تتدفق الحروف من خلال المجرى inFromUser. أمّا المجرى inFromServer فهو مجرى آخر للمدخلات إلى البرنامج مرتبط بالمقبس. تصب الحروف التي تصل من شبكة الإنترنت في مجرى inFromServer. أخيراً يُستخدم المجرى outToServer لمخرجات برنامج الزبون إلى الخادم وهو مرتبط بالمقبس أيضاً. تتدفق الحروف التي يرسلها الزبون إلى الشبكة في المجرى outToServer.



الشكل 2-33 يمتلك الزبون TCPCleint ثلاثة مجارٍ تمر خلالها الحروف.

دعنا الآن نلقي نظرة على السطور المختلفة في الكود.

```
import java.io.*;
import java.net.*;
```

تمثل java.io وjava.net حزمتين من حزم لغة جافا. تحتوي حزمة java.io على فصائل (classes) تشمل النوع "مجرى" للإدخال والإخراج. وبشكلٍ خاص تتضمن الحزمة java.io BufferedReader وDataOutputStream التي يستخدمها البرنامج لإنشاء أنواع "المجرى" الثلاثة التي سبق توضيحها. أمّا حزمة java.net فتحتوي على فصائل لدعم الشبكة. وبشكلٍ خاص تحتوي على Socket وServerSocket. يعرّف البرنامج الكائن clientSocket من النوع Socket. أما السطور التالية:

```
class TCPClient{
public static void main(String argv[]) throws Exception{ ..... }
```

فهي جزء قياسي يوجد في مقدمة معظم برامج جافا. تبدأ الكلمة الدليلية "class" تعريف فصيلة تسمى TCPClient. تحتوي الفصيلة على متغيرات (variables) ووظائف (methods). تحدد بداية ونهاية كتلة تعريف الفصيلة بالأقواس {}. الفصيلة TCPClient ليس لها متغيرات وتتضمن وظيفة واحدة فقط تسمى main() تشبه الوظائف في لغة جافا الإجراءات في لغات أخرى كلغة C، كما تشبه الوظيفة main() نظيراتها في لغات C و++C. عندما يُنفِّذ مترجم جافا (Java Interpreter) تطبيقاً (باستدعائه من قِبَل فصيلة التحكم في التطبيق)، يبدأ باستدعاء الوظيفة main() في الفصيلة، والتي تستدعى بدورها الوظائف الأخرى المطلوبة لتنفيذ التطبيق. في هذه المقدمة لبرمجة المقابس في لغة جافا، يمكنك إهمال الكلمات الدليلية

public, static, void, main, throws Exceptions

(رغم ضرورة تضمينها في الكود).

```
String sentence;
String modifiedSentence;
```

يُعرِّف هذان السطران كائنين من نوع سلسلة الحروف (String). يستخدم الكائن الأول sentence لحفظ سلسلة الحروف التي تكتب من قِبَل المُستخدِم والمراد إرسالها إلى الخادم. أما الكائن الثاني modifiedSentence فيُستخدَم لحفظ سلسلة الحروف التي يحصل عليها الزبون من الخادم والتي ترسل لوحدة الإخراج القياسية لدى المُستخدِم.

```
BufferedReader inFromUser = new BufferedReader(new
InputStreamReader(System.in());
```

يُنشئ هذا السطر كائن inFromUser من النوع BufferedReader، وبدايةً يأخذ القيمة System.in والتي تربط "مجرى الإدخال" بوحدة الإدخال القياسية، مما يسمح للزبون بقراءة النص من لوحة المفاتيح لديه.

```
Socket client Socket = new Socket ("hostname", 6789);
```

يُنشئ هذا السطر الكائن clientSocket من النوع Socket، كما يبدأ أيضاً توصيلة TCP بين الزبون والخادم. ويجب أن تستبدل سلسلة الحروف "hostname" باسم مضيف الخادم الحقيقي (على سبيل المثال"apple.poly.edu"). وقبل أن يبدأ تشغيل توصيلة TCP في الواقع، يقوم الزبون بالحصول على عنوان IP المناظر لاسم المضيف عن طريق DNS. يمثل العدد 6789 رقم منفذ الخادم، ويمكنك استعمال رقم منفذ مختلف، لكن يجب أن تتأكد من أنك تستعمل نفس رقم المنفذ في جانب الخادم من التطبيق. كما ذكرنا في السابق، يستخدم عنوان IP للمضيف مع رقم منفذ التطبيق في تمييز عملية الخادم .

```
DataOutputStream outToServer  = new
DataOutputStream(clientSocket.getOutputStream());

BufferedReader inFromServer= new BufferedReader(new
inputStreamReader(clientSocket.getInputStream()));
```

يُنشئ هذان السطران كائنات مجرى ترتبط بالمقبس. يُستخدَم outToServer لإرسال ناتج العملية إلى الخادم، بينما يُستخدَم inFromServer للاستقبال من الخادم (انظر الشكل 2-33).

```
Sentence = inFromUser.readLine();
```

يضع هذا السطر ما كتبه المُستخدِم على لوحة المفاتيح في الكائن sentence. ويستمر هذا الكائن في استقبال الحروف التي يكتبها المُستخدِم إلى أن ينهي المُستخدِم الكتابة بالضغط على رمز "بداية السطر". تمر تلك الحروف من وحدة الإدخال القياسية (لوحة المفاتيح) خلال المجرى inFromUser إلى الكائن sentence.

```
outToServer.writeBytes(sentence + '\n');
```

يُرسِل هذا السطر النص الموجود بالكائن sentence مع رمز "بداية السطر" إلى المجرى outToServer. تتدفق الحروف خلال مقبس الزبون إلى أنبوب TCP ومن ثم إلى الخادم .

modifiedSentence = inFromServer.readLine();

يسبب هذا السطر انتظار عملية الزبون لحين تلقي رسالة من الخادم. عندما تصل الحروف من الخادم، تتدفق خلال المجرى inFromServer وتوضع في الكائن modifiedSentence. يستمر تجميع الحروف في modifiedSentence حتى ينتهي السطر برمز "بداية السطر".

System.out.println("FROM SERVER" + modifiedSentence);

يعرض هذا السطر محتوى الكائن modifiedSentence على الشاشة.

clientSocket.close();

يغلق هذا السطر الأخير المقبس، وبالتالي يغلق توصيلة TCP بين الزبون والخادم، مما يجعل TCP في الزبون يرسل رسالة TCP إلى TCP في الخادم لإنهاء التوصيلة (انظر الجزء 3-5).

برنامج الخادم TCPServer.java

لنلقِ الآن نظرة على برنامج الخادم. يشبه برنامج الخادم TCPServer.java برنامج الزبون TCPClient.java في عدة جوانب. دعنا الآن نلقي نظرة على سطور البرنامج TCPServer.java، مع ملاحظة أننا لن نُعَلِّق على السطور المطابقة أو المشابهة للأوامر في برنامج الزبون TCPClient.java.

يختلف السطر الأول في TCPServer.java بشكلٍ جوهري عن نظيره في TCPClient.java:

ServerSocket welcomeSocket = new ServerSocket(6789);

يُنشئ هذا السطر الكائن welcomeSocket من نوع ServerSocket، وهو بمثابة باب يستمع الخادم للطرق عليه من بعض الزبائن، وهو معرّف برقم المنفذ 6789.

```
import java.io.*;
import java.net.*;
class TCPServer {
      public static void main(String argv[]) throws Exception{
            String clientSentence;
            String capitalizedSentence;
            ServerSocket welcomeSocket = new ServerSocket(6789);
            while(true){
                  Socket connectionSocket = welcomeSocket.accept();
                  BufferedReader inFromSocket = new BufferedReader(new
                  InputStreamReader(connectionSocket.getInputStream()));
                  DataOutputStream outToClient = new DataOutputStream(
                  connectionSocket.getOutputStream());
                  clientSentence = inFromClient.readLine();
                  capitalizedSentence = clientSentence.toUpperCase() + '\n';
                  outToClient.writeBytes(capitalizedSentence);
            }
      }
}
```

Socket connectionSocket = welcomeSocket.accept();

يُنشئ هذا السطر مقبساً جديداً يسمّى connectionSocket عندما يطرق زبون ما على welcomeSocket. يستخدم المقبس منفذ رقم 6789 أيضاً (وسوف نوضّح في الفصل الثالث لماذا يكون لكلا المقبسين نفس رقم المنفذ). بعد ذلك يُنشئ TCP أنبوباً افتراضياً بين clientSocket في الزبون وconnectionSocket في الخادم. عندئذ يمكن أن يرسل كلٌّ من الزبون والخادم البايتات إلى بعضهم البعض عبر هذا الأنبوب، وتصل كل البايتات المُرسَلة إلى الجانب الآخر بنفس الترتيب. بعد إنشاء connectionSocket يمكن للخادم أن يواصل الاستماع للطلبات من عملاء التطبيق الآخرين باستعمال welcomeSocket، إلا أن هذه النسخة من البرنامج لا تستمع في الحقيقة لمزيد من طلبات الاتصال ولكن يمكن تعديل البرنامج باستخدام threads

للقيام بذلك. بعد ذلك يُنشئ البرنامج عدة كائنات من نوع "مجرى" مماثلة لتلك الكائنات التي تم إنشاؤها في clientSocket.

```
capitalizedSentence = clientSentence.toUpperCase() + '\n';
```

يمثل هذا الأمر قلب التطبيق، فهو يأخذ السطر (سلسلة الحروف) الذي أرسل من قِبَل الزبون، ويحول الحروف إلى حروف كبيرة. باستعمال الوظيفة ()toUpperCase، ثم يضيف رمز "بداية السطر". تعتبر بقية الأوامر الأخرى في البرنامج ثانوية فهي تُستخدَم أساساً للاتصال مع الزبون .لاختبار زوج البرامج قم بإنشاء الملف TCPClient.java على مضيف والملف TCPServer.java على مضيف آخر. تأكّد من تضمين اسم المضيف الصحيح للخادم في برنامج الزبون TCPClient.java، ثم قم بترجمة كلا البرنامجين لتحصل على الملفين المناظرين TCPServer.class وTCPClient.class القابلين للتنفيذ. نفّذ البرنامج TCPServer.class على الخادم فتتكوّن عملية خادم تبقى خاملة إلى أن يتصل بها زبون ما. ثم نفذ البرنامج TCPClient.class فتتكوّن عملية زبون يتم إنشاء توصيلة TCP بين عمليتي الخادم والزبون. أخيراً لاستعمال التطبيق اكتب جملة تليها علامة "بداية السطر" (بالضغط على مفتاح الإدخال) على الزبون.

لتطوير تطبيقات زبون/خادم خاصة بك يمكنك أن تبدأ بإدخال بعض التعديلات على البرنامجين السابقين. على سبيل المثال بدلاً من أن يُحوّل برنامج الخادم كل الحروف إلى حروف كبيرة، يمكن أن يحسب عدد مرات ظهور الحرف "S" في الرسالة ثم يُرجع ذلك العدد.

8-2 برمجة مقابس بروتوكول UDP

عرفنا في الجزء السابق أنه عندما تتصل عمليتان على بروتوكول TCP فكأننا قد مدّدنا أنبوباً لنقل البايتات بينهما، ويبقى مفعول هذا الأنبوب سارياً حتى تغلقه إحدى العمليتين. عندما تريد إحدى العمليات إرسال بعض البايتات إلى العملية الأخرى، فإنها ببساطة تقوم بإدخال البايتات إلى الأنبوب. لا يتعين على عملية الإرسال أن تربط عنوان "الوجهة النهائية" بالبايتات؛ حيث أن الأنبوب متصل

منطقياً بالوجهة النهائية. وعلاوة على ذلك يوفر الأنبوب قناة لمجرى بايتات يوفر نقلاً موثوقاً للبيانات – أي أن سلسلة البايتات التي تتلقاها عملية الاستقبال لدى المُستقبِل هي بالضبط نفس سلسلة البايتات التي أدخلها المُرسِل إلى الأنبوب.

يسمح بروتوكول UDP أيضاً لعمليتين أو أكثر تعملان على مضيفين مختلفين بالاتصال، ولكنه يختلف عن بروتوكول TCP في العديد من الجوانب الأساسية. أولاً: يوفر بروتوكول UDP خدمة غير توصيلية – فلا توجد مرحلة مصافحة في البداية ولا يتم إنشاء أنبوب اتصال بين العمليتين. ولذا فعندما تريد عملية إرسال دفعة من البايتات إلى العملية الأخرى، يجب أن تربط عملية "المُرسِل" عنوان العملية المقصودة في الوجهة النهائية بدفعة البايتات تلك. ويجب أن يتم ذلك لكل دفعة من البايتات ترسلها عملية المُرسِل. وكمثال لنأخذ بعين الاعتبار مجموعة من عشرين شخصاً تستقل خمس سيارات أجرة إلى وجهة نهائية مشتركة؛ بينما يدخل الأشخاص سيارات الأجرة التي ستقلهم، يجب أن يُخطَر كل سائق سيارة أجرة على حدة بالوجهة المطلوبة. يشبه نموذج خدمة UDP سيارة الأجرة تلك. يشمل عنوان الوجهة النهائية عنوان IP لمضيف الوجهة ورقم منفذ العملية على ذلك المضيف. يطلق على دفعة بايتات المعلومات مع عنوان IP لمضيف الوجهة ورقم المنفذ "رزمة" بيانات. يوفر UDP نموذج خدمة مبنياً على الرسائل (message-oriented)، وهذا يعني أن البايتات التي ترسل كدفعة واحدة على جانب الإرسال ويتم توصيلها معاً إلى جانب الاستلام. وهذا يختلف عن طريقة TCP الذي يرسل سيلاً من البايتات المنفصلة في مجرى توصيل موثوق. تُعتبر خدمة UDP خدمة من نوع "أفضل جُهد"، فلا يوجد في الواقع أي ضمان لتوصيل دفعة البايتات المُرسَلة. وهكذا تختلف خدمة UDP بشدة (في عدة نواحٍ) عن نموذج خدمة TCP لمجرى البايتات الذي يحقق نقلاً موثوقاً للبيانات.

بعد تكوين "رزمة" البيانات، تدفع عملية الإرسال بالرزمة إلى الشبكة عبر مقبس. واستمراراً مع مثال سيارة الأجرة، توجد على الجانب الآخر من مقبس الإرسال سيارة أجرة تنتظر الرزمة. تحمل سيارة الأجرة الرزمة عندئذ في اتجاه عنوان الوجهة النهائية للرزمة. ومع ذلك لا تضمن سيارة الأجرة توصيل الرزمة في

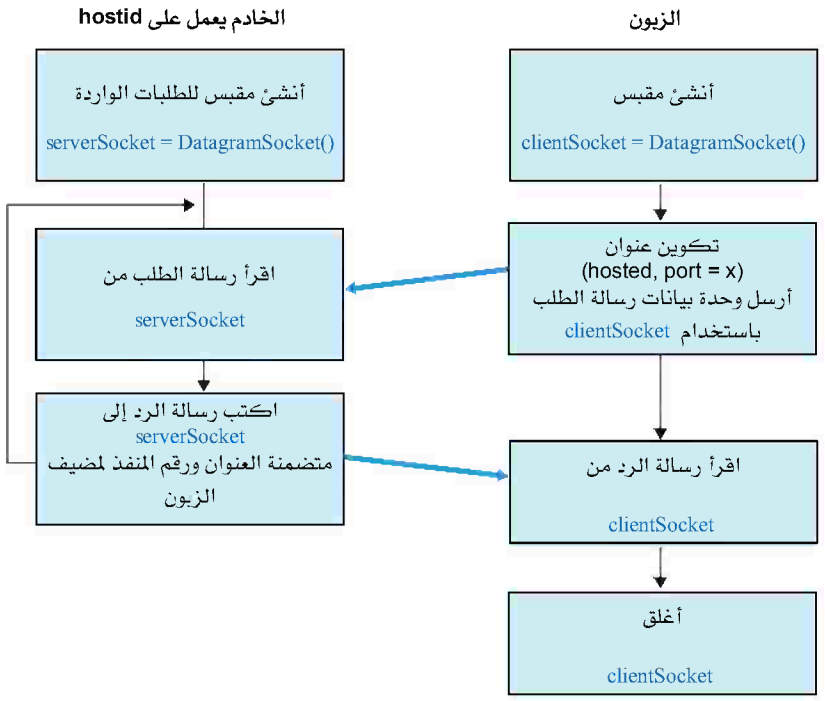
النهاية إلى وجهتها النهائية، حيث يمكن أن تتعطل سيارة الأجرة أو تعاني من مشكلة أخرى غير متوقعة. وبمعنى آخر يوفر بروتوكول UDP خدمة نقل غير موثوقة للعمليات التي تستخدمه للاتصال فيما بينها، أي بدون ضمانات لتوصيل الرزمة إلى غايتها النهائية.

سنتناول في هذا الجزء برمجة المقابس بإعادة تطوير نفس التطبيق في الجزء السابق، ولكن هذه المرة على UDP. سوف نرى أن الكود لبروتوكول UDP مختلف عن الكود لبروتوكول TCP في العديد من الجوانب المهمة. وعلى وجه التحديد: (1) لا توجد مصافحة في البداية بين العمليتين ولذا فلا حاجة لمقبس ترحيب، (2) لا توجد كائنات "مجرى" مرتبطة بالمقبس، (3) تكوّن مضيفات الإرسال الرزم بربط عنوان الوجهة النهائية ورقم المنفذ مع كل دفعة بايتات يتم إرسالها، (4) يجب أن تقوم عملية الاستقبال بفض كل رزمة يتم استلامها للحصول على بايتات المعلومات الموجودة داخل الرزمة.

تذكّر مرة أخرى تطبيقنا البسيط:

1. يقرأ الزبون سطراً من وحدة الإدخال القياسية (لوحة المفاتيح) ويرسله خارج مقبسه إلى الخادم.
2. يقرأ الخادم السطر من مقبسه.
3. يُحوّل الخادم حروف السطر إلى حروف كبيرة (uppercase).
4. يُرسِل الخادم السطر المُعدَّل خارج مقبسه إلى الزبون .
5. يقرأ الزبون السطر المُعدَّل من مقبسه ويعرضه على وحدة الإخراج القياسية (شاشة العرض).

يوضح الشكل 2-34 الأنشطة الرئيسة بين الزبون والخادم اللذين يتصلان بخدمة نقل غير توصيلية (UDP).



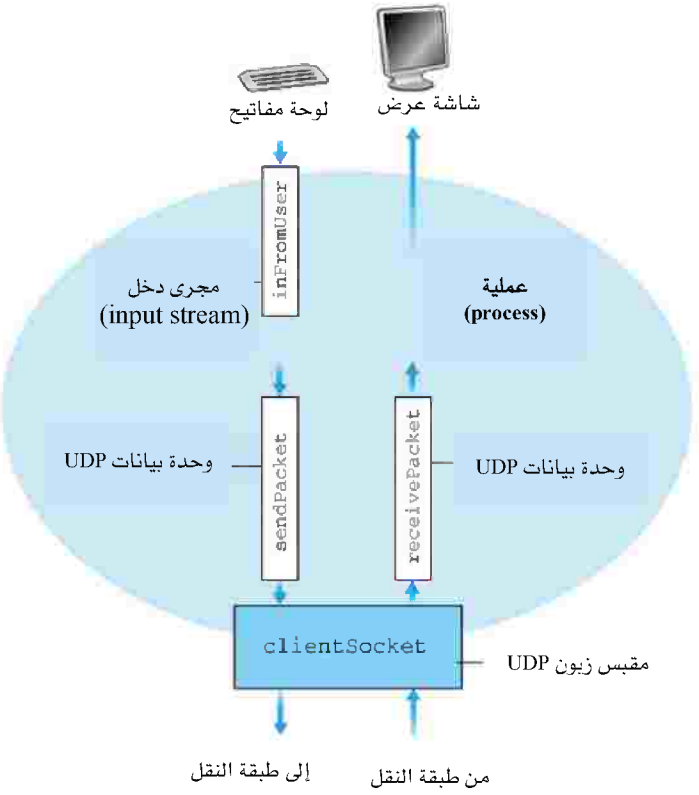
الشكل 2-34 تطبيق "زبون/خادم" باستخدام خدمات نقل غير توصيلية (UDP).

برنامج الزبون UDPClient.java

يُنشئ البرنامج UDPClient.java الخاص بجانب الزبون من التطبيق مجرى واحداً ومقبساً واحداً، كما هو موضح في الشكل 2-35. يُدعى المقبس clientSocket وهو من نوع DatagramSocket. لاحظ أن UDP في الزبون يستخدم نوعاً مختلفاً من المقابس عن TCP. بالتحديد مع UDP يستخدم الزبون مقبساً من نوع DatagramSocket، بينما يستخدم زبون TCP مقبساً من نوع Socket. المجرى inFromUser هو مجرى إدخال إلى البرنامج ويرتبط بوحدة الإدخال القياسية (لوحة المفاتيح) (كان لدينا "مجرى" مكافئ في نسخة TCP من البرنامج). ولكن بالمقارنة مع TCP لايوجد مجرى (إدخال أو إخراج) مرتبط بالمقبس. يدفع UDP

رزماً منفصلة خلال كائن من النوع DatagramSocket بدلاً من أن يغذي البايتات إلى مجرى مرتبط بكائن من النوع Socket (كما كان الحال مع TCP).

```
import java.io.*;
import java.net.*;
class UDPClient {
      public static void main(String argv[]) throws Exception {
            BufferedReader inFromUser = new new BufferedReader(new
            InputStreamReader(System.in));
            DatagramSocket clientSocket = new DatagramSocket();
            InetAddress IPAddress = InetAddress.getByName("hostname");
            byte[] sendData = new byte[1024];
            byte[] receiveData = new byte[1024];
            String sentence = inFromUser.readLine();
            sendData = sentence.getBytes();
            DatagramPacket sendPacket = new DatagramPacket(sendData,
            sendData.length, IPAddress, 9876);
            clientSocket.send(sendPacket);
            DatagramPacket receivePacket = new DatagramPacket(
            receiveData, receiveData.length);
            clientSocket.receive(receivePacket);
            String modifiedSentence = new String(receivePacket.getData());
            System.out.println("FROM SERVER: "+modifiedSentence);
            clientSocket.close();
      }
}
```



الشكل 2-35 لدى الزبون UDPClient.java مجرى واحد؛ يستلم المقبس رزماً من العملية ويسلمها رزماً.

دعنا الآن نلقي نظرة على سطور الكود التي تختلف بشكلٍ ملحوظ عن برنامج TCPClient.java.

```
DatagramSocket clientSocket = new DatagramSocket();
```

يُنشئ هذا السطر الكائن clientSocket من النوع DatagramSocket. بالمقارنة مع TCPClient.java لا يُنشئ هذا السطر توصيلة TCP. وبالتحديد فإن مضيف الزبون لا يتصل بمضيف الخادم بعد تنفيذ هذا السطر. ولهذا السبب لا تأخذ الوظيفة DatagramSocket() اسم أو رقم منفذ مضيف الخادم كمعاملات للوظيفة

(arguments). باستخدام التناظر "باب ↔ مقبس"، يؤدي تنفيذ السطر أعلاه إلى إنشاء "باب" لعملية الزبون ولكنه لا يُنشئ "أنبوب" اتصال بين العمليتين.

```
InetAddress IPAddress = InetAddress.getByName("hostname") ;
```

لإرسال البايتات إلى عملية الوجهة النهائية، نحتاج إلى عنوان العملية. يمثل عنوان IP لمضيف الوجهة النهائية جزءاً من هذا العنوان. يستدعي السطر أعلاه بروتوكول DNS الذي يُترجِم اسم المضيف (والمزوّد في هذا المثال من قِبَل مطوّر البرنامج) إلى عنوان IP. في نسخة TCP من برنامج الزبون أُستخدِم DNS أيضاً، غير أن ذلك تم ضمنياً وليس بشكلٍ صريح. تأخذ الوظيفة getByName() اسم مضيف الخادم كمعامل وترجع عنوان IP لهذا الخادم نفسه، ويوضع هذا العنوان في كائن يسمى IPAddress من النوع InetAddress.

```
byte[] sendData = new byte[1024];
byte[] receiveData = new byte[1024];
```

تستخدم مصفوفتا البايتات sendData وreceiveData في تخزين البيانات التي يرسلها ويستلمها الزبون على التوالي.

```
sendData = sentence.getBytes();
```

يقوم هذا السطر أساساً بتحويل نوع البيانات، حيث يأخذ كائناً نصياً (سلسلة حروف) يسمى sentence ويحوله إلى مصفوفة بايتات تخزّن في sendData.

```
DatagramPacket sendPacket = new DatagramPacket(sendData,
sendData.length, IPAddress, 9876);
```

يبني هذا السطر الرزمة sendPacket، التي سيدفعها الزبون إلى شبكة الإنترنت عبر مقبسه. تتضمن تلك الرزمة البيانات الموجودة في sendData، وطول هذه البيانات، وعنوان IP للخادم، ورقم منفذ الخادم (والذي وضعناه 9876 في هذا المثال). لاحظ أن sendPacket كائن من النوع DatagramPacket.

```
clientSocket.send(sendPacket);
```

في هذا السطر تأخذ الوظيفة send() للكائن clientSocket الرزمة (التي بُنيَت للتو) وترسلها إلى الشبكة عبر clientSocket. ومرة أخرى لاحظ أن UDP يُرسِل

سطراً من الحروف بأسلوب مختلف جداً عن أسلوب TCP. ببساطة قام TCP بإدخال سلسلة الحروف إلى "المجرى" الذي له اتصال منطقي مباشر بالخادم؛ أما UDP فإنه يُكوِّن رزمة تتضمن عنوان الخادم. بعد إرسال الرزمة ينتظر الزبون استلام رزمة من الخادم.

```
DatagramPacket receivePacket = new DatagramPacket(receiveData,
receiveData.length);
```

في هذا السطر (بينما ينتظر الزبون وصول رزمة من الخادم) يُنشئ الزبون مكاناً لحفظ الرزمة (placeholder) يسمى receivePacket من النوع DatagramPacket. ثم يدخل الزبون طور خمول إلى أن يتلقى رزمة؛ عند ذلك يضع الرزمة في receivePacket باستخدام الأمر التالي:

```
clientSocket.receive(receivePacket);
String modifiedSentence = new String(receivePacket.getData());
```

يستخلص هذا السطر البيانات من receivePacket ويجري تحويلاً للنوع (type conversion) من مصفوفة بايتات إلى كائن نصي يسمى modifiedSentence.

```
System.out.println("FROM SERVER:" + modifiedSentence);
```

يعرض هذا السطر (والموجود أيضاً في نسخة TCPClient) محتويات modifiedSentence على شاشة الزبون.

```
clientSocket.close();
```

يغلق هذا السطرالأخير المقبس. ولأن UDP غير توصيلي، لا يؤدي ذلك إلى إرسال الزبون رسالة من طبقة النقل إلى الخادم (على النقيض من TCPClient).

برنامج الخادم: UDPServer.java

دعنا الآن نلقي نظرة على جانب خادم التطبيق:

```
import java.io.*;
import java.net.*;
class UDPServer {
      public static void main(String argv[]) throws Exception {
            DatagramSocket serverSocket = new DatagramSocket(9876);
            byte[] receiveData = new byte[1024];
            byte[] sendData = new byte[1024];
            while(true) {
                DatagramPacket receivePacket = new DatagramPacket(
                receiveData, receiveData.length);
                serverSocket .receive(receivePacket);
                String sentence = new String(receivePacket.getData());
                InetAddress IPAddress = receivePacket.getAddress();
                int port = receivePacket.getPort();
                String capitalizedSentence = sentence.toUpperCase();
                sendData = capitalizedSentence.getBytes();
                DatagramPacket sendPacket = new DatagramPacket(sendData,
                sendData.length, IPAddress, port);
                serverSocket .send(sendPacket);
            }
      }
}
```

يُنشئ برنامج UDPServer.java مقبساً واحداً (كما هو موضح في الشكل 2-36). يسمى المقبس serverSocket، وهو كائن من نوع DatagramSocket تماماً كما كان المقبس في جانب الزبون. ومرةً أخرى لا توجد كائنات "مجرى" مرتبطة بالمقبس. دعنا الآن نلقي نظرة على سطور الكود التي تختلف عن TCPServer.java.

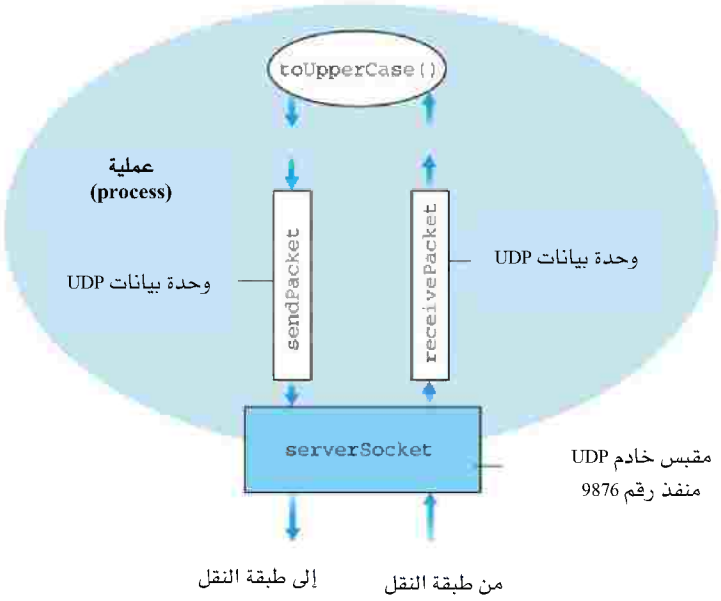
DatagramSocket serverSocket = new DatagramSocket(9876);

يُنشئ هذا السطر كائن serverSocket من النوع DatagramSocket على المنفذ رقم 9876. سوف تمر كل البيانات التي ترسل أو تستقبل خلال ذلك المقبس. ولأن UDP بروتوكول غير توصيلي، فليس من الضروري إنشاء مقبس جديد والاستمرار في الإنصات لطلبات اتصال جديدة (كما فعلنا في TCPServer.java). وإذا اتصل عدد من الزبائن بذلك التطبيق، فسيُرسل الجميع رزمهم عبر هذا الباب الوحيد serverSocket.

```
String sentence = new String(receivePacket.getData());
InetAddress IPAddress = receivePacket.getAddress();
int port = receivePacket.getPort();
```

تستخرج السطور الثلاثة أعلاه الرزمة التي تصل من الزبون. في السطر الأول، تُنتزع البيانات من الرزمة وتوضع في سلسلة الحروف sentence (وهذا السطر له مماثل في UDPClient). ينتزع السطر الثاني عنوان IP؛ وينتزع السطر الثالث رقم منفذ الزبون الذي يُختار اعتباطياً من قبل الزبون وهو يختلف عن رقم منفذ الخادم 9876 (سوف نُناقش أرقام منافذ الزبون بشيء من التفصيل في الفصلِ القادم). من الضروري للخادم أن يحصل على عنوان الزبون (عنوان IP ورقم المنفذ) لكي يمكنه الرد على رسالة الزبون.

بهذا ينتهي تحليلنا لزوج برامج UDP. ولكي تختبر التطبيق قم بإنشاء الملفات UDPClient.java على مضيف وUDPServer.java على مضيف آخر. (تأكّد من تضمين اسم المضيف الصحيح للخادم في برنامج الزبون UDPClient.java). قم بترجمة وتنفيذ البرنامجين على مضيفاتهما الخاصة. بخلاف TCP يمكنك تنفيذ جانب الزبون قبل تنفيذ جانب الخادم، وذلك لأن عملية الزبون لا تحاول بدء الاتصال بالخادم عند تنفيذ برنامج الزبون. ويمكنك استخدام التطبيق بكتابة جملة من برنامج الزبون، فيرسلها إلى الخادم، ثم يستلم الرد ويطبعه على الشاشة.



الشكل 2-36 ليس لدى الخادم UDPServer أي مجارٍ؛ يستلم المقبس رزماً من العملية ويسلمها رزماً.

9-2 الخلاصة

درسنا في هذا الفصل مفاهيم وسمات تطوير تطبيقات الشبكة. تعلمنا عن البنية المعمارية "زبون/خادم" والموجودة في كل مكان والمستعملة في العديد من تطبيقات الإنترنت، ورأينا استخداماتها في بروتوكولات HTTP، وFTP، وSMTP، وPOP3، وDNS. كما درسنا تلك البروتوكولات الهامة وتطبيقاتها المناظرة كالويب ونقل الملفات والبريد الإلكتروني وDNS بشيء من التفصيل. تناولنا أيضاً بنية النظائر التي يزداد انتشارها باضطراد وكيفية استخدامها في العديد من التطبيقات. كما استعرضنا كيف يمكن استخدام واجهة برمجة المقبس API لبناء تطبيقات الشبكة. كما تناولنا استخدام المقابس لتوفير خدمات

النقل من طرف إلى طرف سواء التوصيلي منها (TCP) أو اللاتوصيلي (UDP). إن الخطوة الأولى في رحلتنا خلال البنية المعمارية الطبقية للشبكة قد اكتملت الآن!

في بداياتنا الأولى مع هذا الكتاب (وبالتحديد في الجزء 1-1) أوردنا تعريفاً مبسطاً وغير واضح بعض الشيء للبروتوكول على أنه "صيغ الرسائل التي يتم تبادلها بين اثنين أو أكثر من الكيانات المتصلة، بالإضافة إلى الخطوات التي تتخذ عند إرسال أو استقبال رسالة أو حصول حدث آخر." لا شك أن المادة العلمية في هذا الفصل وبشكلٍ خاص دراستنا المُفصَّلة لبروتوكولات التطبيقات المختلفة قد أعطت الآن الكثير من المعاني لهذا التعريف، فالبروتوكولات مفهوم أساسي في مجال الشبكات، ودراستنا لبروتوكولات التطبيقات أعطتنا الفرصة الآن لتطوير فهم أكثر وعياً ودراية بماهية البروتوكولات وأهميتها.

تناولنا في الجزء 2-1 نماذج الخدمة التي يقدمها بروتوكولا طبقة النقل TCP وUDP للتطبيقات التي تستخدمهما، وألقينا نظرة عن قرب على تلك النماذج عندما طوَّرنا تطبيقات بسيطة تعمل على TCP وUDP في الجزء 2-7 والجزء 2-8 على الترتيب. ولكننا لم نسهب القول حول كيفية توفير TCP وUDP لتلك النماذج. على سبيل المثال عرفنا أن TCP يوفر خدمة نقل للبيانات موثوقة، ولكننا لم نذكر بعد كيفية تحقيق ذلك. في الفصل القادم سوف نُلقي نظرة أدق نستعرض من خلالها ليس فقط "ماذا" ولكن أيضاً "كيف" و"لماذا" فيما يتعلق ببروتوكولات النقل. بعد تلك المعرفة حول تركيب تطبيقات الإنترنت وبروتوكولات طبقة التطبيقات، يمكننا الآن أن نغوص أعمق في رصّة بروتوكولات الإنترنت وفحص طبقة النقل في الفصل القادم.

أسئلة وتمارين وتدريبات الفصل الثاني

❖ أسئلة مراجعة

• الجزء 2-1

1. اذكر أسماء خمسة تطبيقات ذات ملكية عامة للإنترنت واذكر أسماء بروتوكولات طبقة التطبيقات التي يستخدمونها.
2. ما الفرق بين بنية الشبكة المعمارية وبنية التطبيقات؟
3. في جلسة اتصال بين عمليتين، أي منهما يمثل الزبون وأي منهما يمثل الخادم؟
4. في تطبيق مشاركة النظائر للملفات، هل توافق على أنه "لا يوجد ما يمثل جانبي الزبون والخادم في جلسة الاتصال"؟ بيّن سبب الموافقة أو الرفض.
5. ما المعلومات التي تستخدمها عملية يتم تشغيلها على أحد المضيفات لتعريف عملية أخرى يجري تنفيذها على مضيف آخر؟
6. افترض أنك تريد أن تقوم بتنفيذ معاملة تجارية (transaction) معينة على خادم ما من مضيف بعيد بأقصى سرعة ممكنة، فهل ستستخدم بروتوكول TCP أو UDP؟ ولماذا؟
7. بالرجوع إلى الشكل 2-4 نرى أنه لا توجد تطبيقات مذكورة بالشكل تضع قيوداً على كلٍّ من التوقيت وفقد البيانات معاً، فهل تستطيع تخيل أحد تلك التطبيقات التي تتطلب عدم فقد للبيانات وتكون شديدة الحساسية للتوقيت؟
8. اذكر الأربع فئات العامة للخدمات التي يمكن أن يوفرها بروتوكول طبقة النقل؛ مع بيان اسم البروتوكول المستخدم مع كل فئة من فئات الخدمة هل هو TCP أو UDP أو كلاهما؟
9. تذكر أنه يمكن تحسين بروتوكول TCP باستخدام بروتوكول SSL لتوفير خدمات الأمن (بما في ذلك التشفير) بين العمليتين المتصلتين؛ فهل يعمل بروتوكول SSL في طبقة النقل أم في طبقة التطبيقات؟ وماذا يجب على مطوّر التطبيقات أن يفعل إذا أراد أن يستخدم SSL؟

• الأجزاء 2-2 حتى 2-5

10. ما المقصود ببروتوكول مصافحة؟

11. لماذا يستخدم بروتوكول TCP وليس UDP مع كلٍّ من البروتوكولات التالية: HTTP، FTP، SMTTP، POP3؟
12. افترض أن أحد مواقع التجارة الإلكترونية يريد الاحتفاظ بسجل المشتريات لكل زبون؛ صف كيف يمكن تنفيذ ذلك باستخدام الكوكيز.
13. صف كيف يمكن أن يؤدي استخدام ذاكرة الويب المخبأة إلى تخفيض تأخير استلام المتصفّح للكائن المطلوب. هل يؤدي استخدام ذاكرة الويب المخبأة إلى تخفيض التأخير لكل الكائنات المطلوبة أم لبعض هذه الكائنات فقط؟ ولماذا؟
14. استخدم Telnet للاتصال بخادم ويب وأرسل رسالة طلب متعددة السطور وتتضمن سطر التريسة If-Modified-Since لتتسبب في ظهور 304 Not Modified في سطر الحالة في رسالة الرد.
15. لماذا يقال أن FTP يرسل يرسل معلومات التحكم "خارج النطاق" (out-of-band)؟
16. افترض أن أليس ترسل رسالة بريد إلكتروني من خلال حسابها على نظام للبريد الإلكتروني مبني على الويب (كـ Hotmail وgmail) إلى بوب والذي يستخدم POP3 للوصول لبريده. ناقش كيف تنتقل الرسالة من مضيف أليس إلى مضيف بوب؛ مع ذكر سلسلة بروتوكولات طبقة التطبيقات المستخدمة أثناء انتقال الرسالة بين المضيفين.
17. استعرض سطور التريسة لإحدى رسائل البريد الإلكتروني التي تلقيتها حديثاً. ما عدد سطور التريسة التي تتضمن Received؟ حلل كل سطر من سطور التريسة في تلك الرسالة.
18. من منظور المستخدم ما الفرق بين نمط download-and-delete ونمط download-and-keep في بروتوكول POP3؟
19. هل من الممكن أن يستخدم خادم الويب لمؤسّسة وخادم البريد الإلكتروني لها نفس الاسم البديل للمضيف (مثلاً foo.com)؟ ما نوع سجل المورد (RR) لمضيف البريد الإلكتروني؟

- **الجزء 2-6**

20. في بروتوكول BitTorrent افترض أن أليس تزود بوب بالقطع على أساس 30 ثانية؛ فهل من الضروري أن يقوم بوب برد الجميل بتزويدها القطع إلى أليس بنفس تلك المدة؟ بيّن السبب؟

21. افترض أن أليس تلتحق كنظير جديد مع BitTorrent بدون معالجة لأيٍّ من القطع. ولذا لا يمكنها أن تصبح واحداً من النظائر الأربعة على قمة قائمة النظائر المُحمِّلة (uploaders) لدى أيٍّ من النظائر الأخرى. بين كيف يمكنها عندئذ أن تحصل على أول قطعة رغم عدم وجود أي شيء لتحميله؟
22. ما المقصود بالشبكة الإضافية (overlay network)؟ هل تتضمن موجّهات؟ ما الروابط (edges) فيها؟ وكيف يتم إنشاء وتعديل الشبكة الإضافية لفيضان الاستفسار؟
23. من أي وجه يكون نظام الرسائل الفورية بفهرس مركزي نظاماً هجيناً من بنية زبون/خادم وبنية النظائر؟
24. تستخدم معظم أنظمة الرسائل الفورية اليوم فهرساً مركزياً لتحديد أماكن المُستخدِمين. افترض أنه بدل من ذلك استخدمت شبكة إضافية بفيضان الاستفسار (مثل Gnutella) لتحديد أماكن المُستخدِمين. صف كيف يمكن أن يتم ذلك، وناقش مميزات وعيوب مثل هذا التصميم.
25. يستخدم بروتوكول Skype طرق النظائر لوظيفتين هامتين؛ ما هما؟
26. اذكر على الأقل أربعة تطبيقات ذات طبيعة تناسبها بيئة النظائر (أمثلة على ذلك توزيع الملفات والرسائل الفورية).

- **الأجزاء 2-7 حتى 2-8**

27. يحتاج خادم UDP الموصوف في جزء 2-8 لمقبس واحد فقط بينما يحتاج خادم TCP الموصوف في الجزء 2-7 لمقبسين؛ لماذا؟ وإذا كان على خادم TCP أن يدعم n من التوصيلات في نفس الوقت (كلٌّ منها من مضيف مختلف)، فكم عدد المقابس التي سيحتاجها خادم TCP؟
28. في تطبيق زبون/خادم على بروتوكول TCP الموصوف في الجزء 2-7، لماذا يجب أن يتم تشغيل الخادم قبل الزبون؟ ولماذا لا نحتاج لنفس الشرط في حالة تطبيق زبون/خادم على بروتوكول UDP الموصوف في الجزء 2-8؟

❖ تدريبات

1. صح أم خطأ
   a. يطلب مستخدم صفحة ويب تتضمن بعض النصوص وصورتين. لهذه الصفحة سيرسل الزبون رسالة طلب واحدة ويستقبل ثلاث رسائل رد.
   b. يمكن أن تُرسَل صفحتا ويب مختلفتان (مثلاً: www.mit.edu/research.html وwww.mit.edu/students.html) على نفس التوصيلة الدائمة.
   c. من الممكن لقطعة TCP أن تحمل رسالتي طلب HTTP مختلفتين عند وجود توصيلات غير دائمة بين المتصفّح وخادم الأصل.
   d. يدل سطر الترويسة Date في رسالة رد HTTP على زمن آخر تعديل للكائن المُتضمَّن في تلك الرسالة.
2. اقرأ RFC 959 عن بروتوكول FTP ثم اذكر كل أوامر الزبون التي يدعمها.
3. اعتبر زبون HTTP يريد أن يسترجع مستند ويب من عنوان URL معين. وبافتراض أن عنوان IP لخادم HTTP غير معروف في البداية. ما هي البروتوكولات المطلوبة في كلٍّ من طبقة النقل وطبقة التطبيقات بالإضافة إلى HTTP لتنفيذ هذه المهمة؟
4. اعتبر سلسلة حروف الأسكي (ASCII) التالية والتي تم إلتقاطها ببرنامج Ethereal عندما أرسل المتصفّح رسالة GET (أي أن هذا النص هو المحتوى الفعلي لرسالة GET). تمثل الأحرف <cr><lf> بداية السطر.

```
GET /cs453/index.html HTTP/1.1<cr><lf>Host: gai
a.cs.umass.edu<cr><lf>User-Agent: Mozilla/5.0 (
Windows;U; Windows NT 5.1; en-US; rv:1.7.2) Gec
Ko/20040804 Netscape/7.2 (ax) <cr><lf>Accept: ex
t/xml,application/xml,application/xhtml+xml,text
/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
<cr><lf>Accept-Language: en-us,en;q=0.5<cr><lf>Accept-
Encoding: zip,deflate<cr><lf>Accept-Charset: ISO
-8859-1,utf-8;q=0.7,*;q=0.7<cr><lf>Keep-Alive: 300<cr>
<lf>Connection: keep-alive<cr><lf><cr><lf>
```

أجب على الأسئلة التالية مع بيان مكان الإجابة في رسالة GET:

a. ما عنوان URL للمستند المطلوب من المتصفّح؟
b. ما هو رقم الإصدار (النسخة) لبروتوكول HTTP الذي يستخدمه المتصفّح؟
c. ما نوع التوصيلة التي يطلبها المتصفّح هل هي دائمة (persistent) أم غير دائمة (non-persistent)؟

d. ما هو عنوان IP للمضيف الذي يجري تشغيل المتصفّح عليه؟

5. يبيّن النص التالي الرسالة المُرسَلة من الخادم رداً على رسالة GET في السؤال السابق.

```
HTTP/1.1 200 OK<cr><,f>Date: Tue, 07 Mar 2006
12:39:45GMT<cr><,f>Server: Apache/2.0.52 (Fedora)
<cr><,f>Last-Modified: Sat, 10 Dec 2005 18:27:46
GMT<cr><,f>ETag: "526c3-f22-a88a4c80"<cr><,f>Accept-
Range: bytes<cr><,f>Content-Length: 3874<cr><,f>
Keep-Alive: timeout=max=100<cr><,f>Connection:
keep-alive<cr><,f>Content-Type: text/html; charset=
ISO-8859-1<cr><,f><cr><,f><!doctype html public "-
//w3c//dtdd html 4.0 transitional//en"><,f><html><,f>
<head><,f> <met http-equiv="Content-Type"
content="text/html; charset=iso-8859-1"<,f><meta
name="GENERATOR" content= "Mozilla/4.79 [en] (Windows NT
5.0; U) Netscape]"><,f> <title>CMPSCI 453 / 591 /
NTU-ST550A Spring 2005 homepage</title><,f></head><,f>
        <much more document text following here (not shown)>
```

أجب على الأسئلة التالية مع بيان مكان الإجابة في الرسالة:

a. هل تمكن الخادم من أن يجد المستند المطلوب بنجاح أم لا؟ ومتى تم إرسال الرد؟

b. متى تم آخر تعديل للمستند؟

c. ما حجم (عدد البايتات) المستند الذي تم تضمينه في رسالة الرد؟

d. ما أول خمسة بايتات في المستند الذي تم تضمينه في رسالة الرد؟ هل وافق الخادم أن تكون التوصيلة دائمة؟

6. احصل على المواصفات HTTP/1.1 (من طلب التعليقات RFC 2616)، وأجب على ما يلي:

a. وضح الإجراء المُستخدَم للتأشير (signaling) بين الزبون والخادم لبيان أن توصيلة دائمة يتم إغلاقها. هل يمكن أن يقوم الزبون أو الخادم أو كلاهما بإرسال إشارة لإغلاق توصيلة؟

b. ما هي خدمات التشفير التي يوفرها HTTP؟

7. افترض أنك ضغطت على رابط داخل المتصفّح للحصول على صفحة ويب، وكان عنوان IP المرتبط بعنوان URL لها غير موجود بالذاكرة المخبأة على المضيف المحلي، ولذا يلزم البحث في دليل DNS للحصول على عنوان IP. افترض أنه تم زيارة n خادم DNS قبل الحصول على عنوان IP، وأن الزيارات المتتابعة استغرقت RTT_1، RTT_2، ....، RTT_n على التوالي. وافترض أيضاً أن الصفحة المطلوبة تتضمن كائناً واحداً يتألف من نصاً بسيطاً بصيغة HTML. افرض أن RTT_0 تمثل زمن الرحلة ذهاباً وإياباً بين المتصفّح

والخادم. وافترض زمن نقل الكائن يساوي صفراً. ما مقدار الوقت المنقضي من لحظة ضغط المُستخدِم على الرابط حتى لحظة حصوله على الكائن؟

8. بالإشارة إلى السؤال السابق وبافترض أن ملف HTML يتضمن عناوين لثلاث كائنات صغيرة على نفس الخادم، فما مقدار الوقت اللازم في الحالات التالية:
   a. توصيلات HTTP غير دائمة وتوصيلات TCP غير متوازية؟
   b. توصيلات HTTP غير دائمة وتوصيلات TCP متوازية؟
   c. توصيلة HTTP دائمة؟

9. بالإشارة إلى الشكل 2-12 والذي فيه شبكة مؤسّسة موصلة بالإنترنت. افترض أن الحجم المتوسط للكائن 900000 بت وأن معدل الطلب المتوسط من متصفّحات المؤسّسة للخادمات الأصل 15 طلب/ثانية. افترض أيضاً أن الوقت المستغرق من لحظة توجيه الموجّه على جانب الإنترنت لطلب HTTP إلى لحظة حصوله على الرد يعادل ثانيتين في المتوسط (انظر الجزء 2-2-5). اعتبر زمن الاستجابة المتوسط الكلي يُمثَّل بمجموع زمن التأخير المتوسط للوصول (أي التأخير بين موجّه الإنترنت وموجّه المؤسّسة) وزمن تأخير الإنترنت المتوسط. استخدم المعادلة التالية لحساب زمن تأخير الوصول المتوسط:

$$t = \frac{\Delta}{1 - \Delta\beta}$$

حيث Δ تمثل الزمن المتوسط اللازم لإرسال كائن على وصلة الوصول وβ تمثل معدل وصول الكائنات لوصلة الوصول.

   a. احسب زمن الاستجابة المتوسط الكلي.
   b. الآن افترض استخدام ذاكرة مخبأة في شبكة المؤسّسة المحلية وأن معدل إصابة الهدف (hit rate) 0.4، ثم احسب زمن الاستجابة المتوسط الكلي في هذه الحالة.

10. افترض وجود وصلة قصيرة طولها 10 أمتار ومعدل الإرسال عليها 150 بت/ثانية في كلا الإتجاهين. افترض أن رزم البيانات طولها 100000 بت وأن رزم التحكم (مثل إشعار الاستلام والمصافحة) طولها 200 بت. افترض وجود N من التوصيلات المتوازية خلال تلك الوصلة وأن الحيز الترددي للوصلة يقسم بالتساوي بين تلك التوصيلات. افترض الآن استخدام بروتوكول HTTP وأن حجم كل كائن يتم تنزيله 100 كيلوبت وأن الكائن الذي يتم تنزيله في البداية يتضمن 10 مراجع لكائنات أخرى على نفس الخادم. هل استخدام التوصيلات المتوازية له مغزى في هذه الحالة؟ والآن افترض استخدام HTTP الدائم، هل تتوقع تحسن ملحوظ عما سبق؟ وضح إجابتك وبيّن السبب.

11. اكتب برنامج TCP بسيطاً لخادم يقبل سطور مدخلات من الزبون ويقوم بعرض تلك السطور على الشاشة (وحدة الإخراج القياسية). ويمكنك القيام بذلك بتعديل برنامج TCPServer.java الذي سبق شرحه في الجزء 2-7-2. قم بترجمة البرنامج وتنفيذه. على أي جهاز يحتوي برنامج المتصفّح قم بإعداد خادم الوكيل (proxy) في المتصفّح ليشير إلى المضيف الذي يتم تشغيل برنامخ الخادم الذي أعددته وكذلك قم بضبط رقم المنفذ أيضاً. سيرسل المتصفّح الآن رسائل GET إلى هذا الخادم والذي سيقوم بعرض الرسائل على الشاشة. باستخدام هذا النظام حدّد ما إذا كان المتصفّح يستخدم GET الشرطية للكائنات المُخزَّنة بالذاكرة المخبأة محلياً أم لا.
12. ما الفرق بين MAIL FROM في SMTP وFROM في رسالة البريد نفسها؟
13. قم بقراءة طلب التعليقات 1939 لبروتوكول POP3 ووضح الغرض من أمر UIDL.
14. افترض أنك تستخدم POP3 للوصول إلي بريدك الإلكتروني.

a. افترض أنك أعددت زبون POP للعمل في نمط "نزّل واحذف"، ثم أكمل الإجراء التالي:

C: list
S: 1 498
S: 2 912
S: .
C: retr 1
S: blah blah …
S: …….. blah
S: .
?
?

b. افترض أنك أعددت زبون POP للعمل في نمط "نزّل واحتفظ"، ثم أكمل الإجراء التالي:

C: list
S: 1 498
S: 2 912
S: .
C: retr 1
S: blah blah …
S: …….. blah
S: .
?
?

c. افترض أنك أعددت زبون POP للعمل في نمط "نزّل واحتفظ"، استخدم الجزء (b) من السؤال وافترض أنك استعدت الرسائل 1 و2، ثم خرجت من POP وبعد خمس دقائق قمت بالاتصال مرة أخرى بخادم POP لاستعادة الرسائل الجديدة التي أرسلت لك. اكتب الرسائل والتعليمات التي سيتم تبادلها بين الخادم والزبون لتحقيق هذه المهمة.

15. أجب عن الأسئلة التالية:

a. ما هي قاعدة بيانات whois؟

b. استخدم قواعد بيانات whois على الإنترنت لتحصل على أسماء خادمين DNS، واذكر قاعدة بيانات whois التي استخدمتها.

c. استخدم الأمر nslookup على جهازك المحلي لإرسال استفسارات DNS إلى ثلاثة خادمات DNS: أحدهم يمثل خادم DNS المحلي والاثنان الآخران يمثلان ما وجدته في الجزء (b). جرب الاستفسار عن سجلات من أنواع مختلفة: A، NS، MX. لخص ما تحصل عليه.

d. استخدم الأمر nslookup للبحث عن خادم ويب له عناوين IP متعددة. هل خادم الويب لمؤسستك (مدرسة أو شركة) له عناوين IP متعددة؟

e. استخدم قاعدة بيانات ARIN لتحديد مدى عناوين IP المُستخدَم في جامعتك.

f. صف كيف يستخدم المهاجم قواعد بيانات whois وأداة nslookup للقيام بعمليات استطلاعية لشبكة مؤسستك قبل أن يشن هجومه عليها.

g. ناقش لماذا يجب أن تكون قواعد بيانات whois متاحة علناً.

16. افترض أن ملفاً حجمه 10 جيجابت يتم توزيعه إلى N من النظائر. إذا كان معدل تحميل الملف من الخادم u_s يساوي 20 ميجابت/ثانية، ومعدل تنزيل الملف لكل نظير d_i يساوي 1 ميجابت/ ثانية ومعدل تحميل الملف لكل نظير يساوي u. قم بإعداد رسم بياني يوضح أدنى زمن توزيع لكلٍ من بنية زبون/خادم وبنية النظائرعند كل زوج من القيم لـ N وu عندما N = 10، 100، 1000 وu = 200 كيلوبت/ثانية، 600 كيلوبت/ثانية، 1 ميجابت/ثانية.

17. افترض أن ملفاً حجمه F بت يتم توزيعه إلى N من النظائر باستخدام بنية زبون/خادم. افترض نموذج حركة الموائع (fluid model) أي أن الخادم يمكنه الإرسال لعدة نظائر في نفس الوقت وبمعدلات مختلفة طالما أن المعدل الكلي لا يزيد عن us.

c. افترض أن $u_s / N \le d_{\min}$، حدد أسلوباً للتوزيع بحيث يكون زمن التوزيع NF/u_s.

d. افترض أن $u_s / N \ge d_{\min}$، حدد أسلوباً للتوزيع بحيث يكون زمن التوزيع $F/d_{\min}$.

e. استنتج أنه بشكلٍ عام يحسب أدنى زمن توزيع من $\max\{NF/u_s, F/d_{\min}\}$.

18. افترض أن ملفاً حجمه F بت يتم توزيعه إلى N من النظائر باستخدام بنية النظائر، وافترض النموذج السائل (fluid model). وللتبسيط افترض أن d_{min} كبيرة جداً بحيث لا يمكن أن يمثل الحيز الترددي لتنزيل الملف أي عائق.
   a. افترض $u_s \le (u_s + u_1 + ... + u_N)/N$؛ حدد أسلوباً للتوزيع بحيث يكون زمن التوزيع F/u_s.
   b. افترض $u_s \ge (u_s + u_1 + ... + u_N)/N$؛ حدّد أسلوباً للتوزيع بحيث يكون زمن التوزيع $NF/(u_s + u_1 + ... + u_N)$.
   c. استنتج أنه بشكلٍ عام يحسب أدنى زمن توزيع من $\max\{F/u_s, NF/(u_s + u_1 + ... + u_N)\}$.
19. افترض أن شبكة إضافية (overlay network) بها عدد N من النظائر النشطة وبين كل زوج منها توصيلة TCP فعالة. افترض أيضاً أن توصيلة TCP تمر خلال M موجّه. ما عدد العقد والأحرف المناظرة في تلك الشبكة؟
20. في مناقشتنا للشبكات الإضافية باستخدام فيضان الاستفسارات في الجزء 2-6 وصفنا ببعض التفصيل كيف يلتحق نظير جديد بتلك الشبكة. في هذا السؤال نريد استكشاف ما سيحدث عندما يغادر نظير تلك الشبكة. افترض أن كل نظير مشترك يحتفظ في أي لحظة بتوصيلات TCP لأربعة نظائر مختلفة على الأقل. افترض أن النظير X والذي له خمس توصيلات TCP مع النظائر الأخرى يريد أن يغادر.
   a. في البداية اعتبر حالة المغادرة رشيقة (graceful leave) أي أن النظير X يغلق بشكلٍ واضح تطبيقه وبالتالي ينهي بشكلٍ واضح توصيلاته الخمسة. ماذا سيفعل كلٌّ من النظائر الخمسة التي كان يتصل بها النظير X؟
   b. افترض الآن أن النظير X يقطع اتصاله بالإنترنت بشكلٍ مفاجئ دون انذار سابق لجيرانه الخمسة أنه سينهي توصيلات TCP معهم؛ ماذا يحدث في هذه الحالة؟
21. في هذا السؤال سنستكشف التوجيه على المسار العكسي (reverse-path routing) لرسائل الإصابة للاستفسار (query hit) في فيضان الاستفسار. افترض أن أليس أرسلت رسالة استفسار، وأن بوب تلقى رسالة الاستفسار (والتي قد تكون قد تم توجيهها خلال عدة نظائر بينية) ولديه ملف يطابق الاستفسار.
   a. تذكر أنه عندما يمتلك نظير ملفاً مطابق للاستفسار فإنه سيرسل رسالة إصابة خلال المسار العكسي لرسالة الاستفسار المناظرة. كتصميم بديل يمكن أن يؤسس بوب توصيلة TCP مباشرة مع أليس ويرسل رسالة الإصابة للاستفسار خلال تلك التوصيلة. ما هي مميزات وعيوب هذا التصميم البديل؟

b. عندما يقوم النظير أليس بتوليد رسالة استفسار، يقوم بإدخال رقم تعريفي فريد في حقل MessageID (معرّف الرسالة). عندما يمتلك النظير بوب تطابق، يُولّد رسالة إصابة مستخدماً نفس الرقم التعريفي في حقل MessageID. صف كيفية استخدام النظائر لحقل MessageID وجداول التوجيه المحلية لتحقيق توجيه المسار العكسي.

c. تصميم آخر بديل لا يستخدم أرقام تعريف: عندما تصل رسالة استفسار إلى نظيرٍ ما، يقوم النظير بتضمين عنوان IP له في الرسالة قبل أن يقوم بتوجيهها. صف كيفية استخدام النظائر لهذه الآلية لتحقيق توجيه المسار العكسي.

22. في هذا السؤال تصميم شبكة إضافية هرمية فيها نظائر عادية ونظائر ممتازة ونظائر فوق الممتاز (super-duper).

a. افترض أن كل نظير فوق الممتاز مسؤول تقريباً عن 200 نظير ممتاز وأن كل نظير ممتاز مسؤول تقريباً عن 200 نظير عادي. ما عدد النظائر فوق الممتاز المطلوب لشبكة بها 4 ملايين نظير؟

b. ما المعلومات التي يحتمل أن يخزنها كل نظير ممتاز؟ ما المعلومات التي يحتمل أن يخزنها كل نظير فوق الممتاز؟ وكيف يتم البحث في مثل هذا التصميم ثلاثي المستوى؟

23. في فيضان الاستفسار الذي نوقش في الجزء 2-6 افترض أن كل نظير متصل بعدد N من الجيران في الشبكة الإضافية. افترض أيضاً أن قيمة حقل عدد العقد (node-count) في البداية K. افترض أن أليس أرسلت استفساراً. احسب الحد الأعلى لعدد رسائل الاستفسارات التي سترسل خلال الشبكة الإضافية.

24. قم بإعداد وترجمة برامج جافا TCPClient وUDPClient على مضيفٍ ما وبرامج TCPServer وUDPServer على مضيفٍ آخر.

a. افترض أنك قمت بتشغيل TCPClient قبل تشغيل TCPServer؛ فماذا سيحدث؟ ولماذا؟

b. افترض أنك قمت بتشغيل UDPClient قبل تشغيل UDPServer؛ فماذا سيحدث؟ ولماذا؟

c. ماذا يحدث إذا استخدمت أرقام منافذ مختلفة لكلٍّ من الزبون والخادم؟

25. افترض أننا قمنا باستبدال السطر

```
DatagramSocket clientSocket = new DatagramSocket();
```

بالسطر

```
DatagramSocket clientSocket = new DatagramSocket(5432);
```

في برنامج UDPClient.java؛ فهل يلزم تغيير UDPServer.java؟ ما أرقام المنافذ لمقابس الزبون والخادم؟ وماذا كانت تلك الأرقام قبل القيام بهذا التغيير؟

❖ أسئلة للمناقشة

1. ما سبب انتشار تطبيقات النظائر لمشاركة الملفات في رأيك؟ هل لأنها توزع ملفات الموسيقى والفيديو مجاناً (رغم كونه قد يكون بشكلٍ مخالف أو غير رسمي)؟ أم لأن العدد الكبير من خادماتها يستجيب بكفاءة للطلب الهائل للميجابايتات من البيانات؟ أم كل هذه الأسباب مجتمعة؟
2. اقرأ البحث [Biddle 2003] عن شبكة Darknet ومستقبل توزيع المحتوى. هل توافق على كل أراء المؤلفين؟ بين أسباب ذلك.
3. غالباً ما تستخدم مواقع التجارة الإلكترونية ومواقع الويب الأخرى قواعد بيانات خلفية. كيف تتصل خادمات HTTP مع قواعد البيانات الخلفية تلك؟
4. كيف يمكنك إعداد برنامج المتصفّح لديك لاستعمال الذاكرة المخبأة المحلية؟ وما الاختيارات المتاحة لديك لتلك الذاكرة؟
5. هل يمكنك إعداد برنامج المتصفّح لديك ليفتح عدة توصيلات لموقع ويب على التوازي في نفس الوقت؟ ما هي ميزات وعيوب وجود عدد كبير من توصيلات TCP تلك في نفس الوقت؟
6. رأينا أن مقابس TCP في الإنترنت تتعامل مع البيانات المُرسَلة كتدفق من البايتات (byte stream) في حين تتعرف مقابس UDP على فواصل الرسائل. اذكر ميزة وعيب لواجهة برمجة التطبيقات التي تتعامل مع البيانات المُرسَلة كتدفق من البايتات وتلك التي تتعرف وتحتفظ بفواصل الرسائل.
7. ما هو خادم Apache للويب؟ وماذا يكلف؟ وما هي الوظائف التي يوفرها حالياً؟
8. افترض أن المنظمات التي تضع معايير الويب قررت أن تغير اصطلاحات التسمية لكي يسمى كل كائن ويشار إليه باسم فريد لا يعتمد على موقعه (وهو ما يُعرف بـ URN). ناقش بعض القضايا المتعلقة بذلك.
9. هل توجد اليوم أي شركات لتوزيع برامج البث التلفزيوني الحية على الإنترنت؟ وإذا كانت هناك أي من تلك الشركات فهل تستخدم بنية زبون/خادم أم بنية النظائر؟

10. هل الشركات التي توفر خدمة الفيديو عند الطلب (video-on-demand) على الإنترنت اليوم تستخدم بنية النظائر؟
11. كيف يوفر سكايب (Skype) خدمة اتصال من الحاسب للهاتف (PC-to-Phone) للعديد من البلدان المختلفة؟
12. ما هي بعض زبائن BitTorrent المنتشرة اليوم؟

❖ تدريبات على برمجة المقابس

1. خادم ويب متعدد التفريعات (Multi-threaded Web Server): في نهاية هذا التدريب ستكون قد طورت خادم ويب متعدد التفريعات في لغة جافا يكون قادراً على خدمة طلبات متعددة على التوازي. ستحقق الإصدار 1.0 لبروتوكول HTTP كما هو مُعرّف في RFC 1945. يُنشئ بروتوكول HTTP/1.0 توصيلة TCP منفصلة لكل زوج من الطلب والرد. تقوم تفريعة (thread) منفصلة بمعالجة كلٍّ من هذه التوصيلات. توجد أيضاً تفريعة رئيسة يتم فيها استماع الخادم للزبائن التي تريد تأسيس توصيلات معه. ولتبسيط مهمة البرمجة، سنطور البرنامج على مرحلتين. في المرحلة الأولى ستكتب خادم متعدد التفريعات يقوم ببساطة بإظهار محتويات طلبات HTTP التي يتلقاها. بعد أن تتأكد من أن هذا البرنامج يعمل بشكلٍ صحيح ستضيف الكود المطلوب لتوليد الرد المناسب. وأثناء تطويرك للبرنامج يمكنك اختبار هذا الخادم مستعيناً بمتصفّح الويب لديك (مثل متصفّح اكسبلورر من مايكروسوفت)؛ لكن عليك أن تتذكر أن هذا الخادم لا يعمل على المنفذ المعياري 80 ولذا عليك أن تحدد رقم المنفذ المستخدم ضمن عنوان URL الذي تكتبه للمتصفّح. على سبيل المثال إذا كان اسم المضيف الذي يجري تشغيل هذا الخادم عليه host.someschool.edu وكنت مستخدماً المنفذ رقم 6789 للخادم وتريد أن تستعرض الملف index.html، عليك أن تكتب العنوان كالآتي:

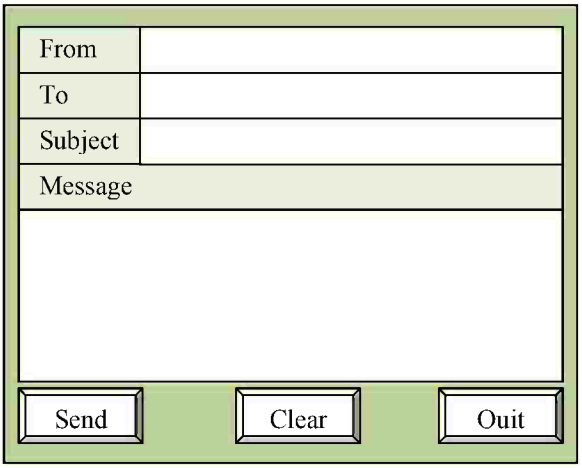
http://host.someschool.edu:6789/index.html

عندما يصادف هذا الخادم خطأً يجب أن يرد برسالة تبين هذا الخطأ بصيغة HTML المناسبة لاستعراضها ضمن نافذة المتصفّح. يمكنك أن تجد التفاصيل الكاملة لهذا التدريب وكذلك أجزاء من البرنامج في لغة جافا من خلال موقع الويب لهذا الكتاب
http://www.awl.com/kurose-ross

2. زبون البريد (Mail Client): في هذا التدريب ستقوم بتطوير وكيل مُستخدِم للبريد الإلكتروني في لغة جافا بالخصائص التالية:

- يوفر واجهة رسومية للمُستخدِم (GUI) تضم حقولاً لخادم البريد المحلي وعنوان البريد للمُرسِل وعنوان البريد للمُستقبل وعنوان الرسالة والرسالة نفسها.
- يؤسس توصيلة TCP بين الزبون وخادم البريد المحلي، ويرسل أوامر SMTP لهذا الخادم، ويستقبل ويعالج أوامر SMTP من هذا الخادم.

يجب أن يكون شكل واجهة المُستخدِم كالآتي:



ستطور وكيل المُستخدِم لكي يرسل رسالة البريد الإلكتروني لمُستقبل واحد على الأكثر في كل مرة. وعلاوة على ذلك سيفترض وكيل المُستخدِم أن الجزء الخاص بالنطاق لعنوان البريد للمُستقبل هو نفسه الاسم القانوني (canonical name) لخادم البريد للمُستقبل (أي أن وكيل المُستخدِم لن يقوم بالبحث في دليل أسماء النطاقات عن سجل خادم البريد ولذا يجب أن يُدخِل المُستخدِم الاسم الحقيقي لخادم البريد). يمكنك الاطلاع على التفاصيل الكاملة لهذا التطبيق وأجزاء من البرامج بلغة جافا في موقع كتابنا هذا على الويب http://www.awl.com/kurose-ross.

3. تطبيق UDP Pinger: في هذا التدريب ستطور زبوناً وخادماً للبنج ويستخدمان بروتوكول UDP. تشبه الوظائف المتوفرة مع هذه البرامج برنامج البنج المعياري المتاح ضمن أنظمة التشغيل الحديثة. يعمل برنامج البنج المعياري بإرسال رسائل صدى ICMP والتي يقوم الجهاز البعيد بإرجاعها للمُرسِل. عندئذ يحدد المُرسِل زمن رحلة الذهاب والإياب (RTT) بينه وبين ذلك الجهاز المراد اختبار الاتصال به. لا توفر لغة جافا أية وظائف لإرسال واستقبال رسائل ICMP وهذا هو السبب أنه يلزمك تطوير هذه البرامج في طبقة

التطبيقات باستخدام مقابس ورسائل بروتوكول UDP. توجد التفاصيل الكاملة لهذا التطبيق وأجزاء من البرامج بلغة جافا في موقع كتابنا هذا على الويب .http://www.awl.com/kurose-ross

4. خادم وكيل الويب: في هذا التدريب ستطور خادماً بسيطاً لوكيل الويب وفيه أيضاً يتم تخزين صفحات الويب في ذاكرة مخبأة. يستقبل هذا الخادم رسائل GET من متصفّح الويب، ويوجه تلك الرسائل لخادم الويب للوجهة، ثم يستقبل رد HTTP من هذا الخادم ويوجهه للمتصفّح. يعتبر هذا الخادم بسيط جداً لوكيل الويب، فهو يتعامل فقط مع رسائل GET. ومع ذلك يستطيع هذا الخادم أن يتعامل مع كل أنواع الكائنات بما في ذلك ملفات الصور وليس فقط صفحات HTML. توجد التفاصيل الكاملة لهذا التطبيق وأجزاء من البرامج بلغة جافا في موقع كتابنا هذا على الويب .http://www.awl.com/kurose-ross

❖ تدريبات معملية على استخدام برنامج Ethereal

1. بروتوكول HTTP: بعد أن عرفنا مبادئ تشغيل برنامج Ethereal لالتقاط الرزم في التدريب الأول في نهاية الفصل الأول، يمكننا الآن استخدامه لفحص البروتوكولات أثناء تشغيلها. في هذا التدريب سنفحص العديد من خصائص بروتوكول HTTP مثل: رسائل GET الأساسية والرد عليها، وصيغة رسائل HTTP، واسترجاع ملفات HTML كبيرة، واسترجاع ملفات HTML متضمنة عناوين URL، والتوصيلات الدائمة وغير الدائمة، والتحقق والأمن في بروتوكول HTTP. وكما هو الحال مع كل تدريبات Ethereal يوجد الوصف الكامل لهذا التدريب من خلال موقع الكتاب .http://www.awl.com/kurose-ross

2. دليل أسماء النطاقات (DNS): في هذا التدريب سنفحص عن كثب جانب الزبون لخدمة دليل أسماء النطاقات (والتي تقوم بالتحويل ما بين أسماء المضيفات وعناوين IP لها). تذكر من الجزء 2-5 أن دور الزبون في DNS يعتبر بسيطاً نسبياً (يقوم الزبون بإرسال استفسار إلى خادم DNS المحلي ويستقبل الرد منه). يمكن إخفاء الكثير من التفاصيل عن زبون DNS بينما تتفاعل خادمات أسماء النطاقات ذات التنظيم الهرمي مع بعضها بطريقة تتابعية أو تكرارية للحصول على عنوان IP المناظر لاسم المضيف المعني. ومع ذلك يعتبر البروتوكول من منظور زبون DNS بسيطاً نوعاً ما. سنرى طريقة تشغيل DNS في هذا التدريب والذي يوجد الوصف الكامل له من خلال موقع الكتاب .http://www.awl.com/kurose-ross