1: الأسس

LibreTexts™

1: الأسس

> يجب عليّ أن أبتكر نظامًا، وإلا سأكون عبدًا لنظام إنسان آخر؛ لن أستخدم
> المنطق والمقارنة: عملي هو الإبداع - William Blake

المسألة: بناء الشبكة

في بناء شبكة حاسوبية computer network، شبكة قادرة على النمو لتغطية عالميَّة ودعم تطبيقات متنوعة مثل المؤتمرات عن بُعد teleconferencing، والفيديو حسب الطلب video on demand، والتجارة الإلكترونية electronic commerce، والحوسبة الموزعة distributed computing، والمكتبات الرقمية digital library. ما التقنيات المتاحة التي ستُشكل لبنات البناء [الأساسية] ، وما نوع لبنات البرمجيات software blocks التي ستصممها لدمج هذه اللبنات في خدمة اتصالات فعّالة؟ الإجابة على هذا السؤال هي الهدف الرئيسي لهذا الكتاب - وصف مواد البناء المتاحة، ثم توضيح كيفية استخدامها لبناء شبكة من الصفر.

نقبل أن نفهم كيفية تصميم شبكة حاسوبية، يجب أن نتفق أولًا على ماهية شبكة الحاسوب. في السابق، كان مصطلح "شبكة" يعني مجموعة الخطوط التسلسلية serial lines المستخدمة لتوصيل أجهزة طرفية غبية [بسيطة] dumb terminals لوصلها بأجهزة الحاسوب المركزية terminals. ومن الشبكات المهمة الأخرى شبكة الهاتف الصوتي voice telephone network وشبكة تلفزيون الكابل cable TV networks المستخدمة لنشر إشارات الفيديو. تشترك هذه الشبكات في أنها متخصصة في التعامل مع نوع وحيد من البيانات (ضغطات المفاتيح keystrokes، الصوت voice، أو الفيديو video)، وعادةً ما تتصل بأجهزة مخصصة (الأجهزة الطرفية terminals، أجهزة الاستقبال اليدوية hand receivers، وأجهزة التلفزيون).

ما الذي يميز شبكة الحاسوب عن غيرها من أنواع الشبكات؟ لعل أهم ما يميزها هو عموميتها. تُبنى شبكات الحاسوب أساسًا من عتاد قابل للبرمجة programmable hardware لأغراض عامة، وليست مُصممة لتكون مثالية لتطبيق مُحدد مثل إجراء المكالمات الهاتفية أو بث إشارات التلفزيون. بل إنها قادرة على نقل أنواع مُختلفة من البيانات، وتدعم نطاقًا واسعًا ومتناميًا من التطبيقات. وقد استحوذت شبكات الحاسوب اليوم إلى حد كبير على الوظائف التي كانت تؤديها سابقًا شبكات الاستخدام الواحد. يتناول هذا الفصل بعض التطبيقات النموذجية لشبكات الحاسوب، ويناقش المتطلبات التي يجب أن يكون مُصمم الشبكة الراغب في دعم هذه التطبيقات على دراية بها.

بعد فهم المتطلبات، كيف نمضي قدمًا؟ لحسن الحظ، لن نكون أول من يبني الشبكة. فقد سبقنا آخرون، وأبرزهم مجتمع الباحثين المسؤولين عن الإنترنت. سنستخدم الخبرة الواسعة المكتسبة من الإنترنت لتوجيه تصميمنا. تتجسد هذه الخبرة في تكوين بنية هيكلية للشبكة network architecture تحددها مكونات components الأجهزة hardware والبرمجيات software المتاحة، وتوضح كيفية ترتيبها لتشكيل نظام شبكة كاملة [متكاملة].

بالإضافة إلى فهم كيفية بناء الشبكات، تزداد أهمية فهم كيفية تشغيلها وإدارتها وكيفية تطوير تطبيقاتها. يمتلك جميعنا تقريبًا الآن شبكات حاسوبية في منازلنا ومكاتبنا، وفي بعض الحالات في سياراتنا، لذا لم يعد تشغيل الشبكات حكرًا على قلة من المتخصصين. ومع انتشار الهواتف الذكية smartphones، أصبح عدد أكبر من هذا الجيل يطور تطبيقات شبكية أكثر من ذي قبل. لذلك، علينا النظر إلى الشبكات من وجهات نظر متعددة: المطورون، والمشغلون، ومطورو التطبيقات.

يبدأ هذا الفصل رحلتنا نحو فهم كيفية بناء وتشغيل وبرمجة شبكة، ويتناول أربعة أمور. أولًا، يستكشف متطلبات مختلف التطبيقات وفئات المستخدمين على الشبكة. ثانيًا، يُقدّم فكرة بنية الشبكة، التي تُشكّل الأساس لبقية الكتاب. ثالثًا، يُقدّم بعض العناصر الرئيسية في تنفيذ شبكات الحاسوب. وأخيرًا، يُحدّد المقاييس الرئيسية المُستخدمة لتقييم أداء شبكات الحاسوب.

1.1: التطبيقات

يعرف معظم الناس الإنترنت Internet من خلال تطبيقاته: الشبكة العنكبوتية العالمية [WWW] World Wide Web، والبريد الإلكتروني email، ووسائل التواصل الاجتماعي social media، وبث streaming الموسيقى أو الأفلام، ومؤتمرات الفيديو videoconferencing، والمراسلة الفورية instant messaging، ومشاركة الملفات file sharing، على سبيل المثال لا الحصر. أي أننا نتفاعل مع الإنترنت كمستخدمين له. يمثل مستخدمو الإنترنت أكبر فئة من الأشخاص الذين يتفاعلون معه بطريقة ما، ولكن هناك فئات مهمة أخرى.

هناك مجموعة من الأشخاص الذين يقومون بإنشاء التطبيقات - وهي مجموعة توسعت بشكل كبير في السنوات الأخيرة حيث خلقت منصات البرمجة programming platform القوية والأجهزة الجديدة مثل الهواتف الذكية [e.g. iphone] smart phone فرصًا جديدة لتطوير التطبيقات بسرعة وإحضارها إلى السوق الكبيرة.

ثم هناك من يُشغّلون الشبكات أو يُديرونها - وهي غالبًا وظيفة خلف الكواليس، لكنها بالغة الأهمية، وغالبًا ما تكون مُعقّدة للغاية. مع انتشار الشبكات المنزلية، أصبح عدد متزايد من الناس، وإن كان محدودًا، مُشغّلي شبكات.

وأخيرًا، هناك أولئك الذين يصممون ويبنون الأجهزة devices والبروتوكولات protocols التي تُشكل مجتمعةً شبكة الإنترنت. وهذه الفئة [المجموعة] الأخيرة هي الهدف التقليدي لكتب الشبكات، مثل هذا الكتاب، وستظل محور تركيزنا الرئيسي. ومع ذلك، سنتناول في هذا الكتاب أيضًا وجهات نظر [منظورات] مطوري التطبيقات application developers ومشغلي الشبكات network operators.

إن مراعاة هذه المنظورات ستمكننا من فهم المتطلبات المتنوعة التي يجب أن تلبيها الشبكة بشكل أفضل. كما سيتمكن مطورو التطبيقات من تطوير تطبيقات تعمل بشكل أفضل إذا فهموا كيفية عمل التكنولوجيا الأساسية [التحتية] وتفاعلها مع التطبيقات. لذا، قبل أن نبدأ في فهم كيفية بناء شبكة، دعونا نلقي نظرة فاحصة على أنواع التطبيقات التي تدعمها شبكات اليوم.

فئات التطبيقات

شبكة الويب العالمية هي تطبيق الإنترنت الذي حوّل الإنترنت من أداة غامضة نوعًا ما، يستخدمها في الغالب العلماء والمهندسون، إلى الظاهرة السائدة التي هي عليها اليوم. وقد أصبحت شبكة الويب Web نفسها منصة قوية لدرجة أن الكثيرين يخلطون بينها وبين الإنترنت، ومن المبالغة القول إنها تطبيق واحد.

يُقدم الويب في جوهره واجهة استخدام بسيطة وبديهية. يتصفح المستخدمون صفحات مليئة الأهداف objects النصية textual والرسومية graphical، وينقرون على الأهداف التي يرغبون في معرفة المزيد عنها، فتظهر صفحة جديدة. يدرك معظم الناس أيضًا أن كل هدف قابل للتحديد في الصفحة مرتبط بمعرف identifier للصفحة التالية أو الكائن التالي المراد عرضه. يُسمى هذا المعرف مُحدد الموارد الموحد Uniform Resource Locator (URL)، وهو يُتيح تحديد جميع الأهداف المُحتملة التي يُمكن عرضها من مُتصفح الويب. على سبيل المثال،

```
http://www.cs.princeton.edu/llp/index.html
```

هو عنوان URL لصفحة تُقدّم معلومات عن أحد مؤلفي هذا الكتاب: يشير الرمز http إلى أنه يجب استخدام بروتوكول نقل النص االترابطي (HTTP) Hypertext Transfer Protocol لتنزيل الصفحة page، و www.cs.princeton.edu هو اسم الجهاز المخدم الذي يُخدّم الصفحة، و/llp/index.html يُحدّد بشكل فريد صفحة السيد Larry على هذا الموقع.

ما يجهله معظم مستخدمي الإنترنت هو أنه بمجرد النقر على رابط URL واحد، قد يتم تبادل أكثر من اثنتي عشرة رسالة عبر الإنترنت، وأكثر من ذلك بكثير إذا كانت صفحة الويب معقدة وتحتوي على الكثير من العناصر المضمنة embedded objects. يشمل تبادل الرسائل هذا ما يصل إلى ست رسائل لترجمة اسم المخدم (www.cs.princeton.edu) server إلى عنوان بروتوكول الإنترنت Internet Protocol (IP) الخاص به (128.112.136.35)، وثلاث رسائل لإعداد اتصال بروتوكول التحكم في الإرسال Transmission Control Protocol (TCP) بين متصفحك وهذا الخادم، وأربع رسائل لمتصفحك لإرسال طلب "GET" HTTP وللمخدم للرد بالصفحة المطلوبة (ولكي يُقر كل طرف باستلام هذه الرسالة)، وأربع رسائل لإنهاء tear down اتصال TCP. وبالطبع، لا يشمل هذا ملايين الرسائل التي تتبادلها عُقد الإنترنت على مدار اليوم، لمجرد إعلام بعضها البعض بوجودها واستعدادها لخدمة صفحات الويب، وترجمة الأسماء إلى عناوين، وإعادة توجيه الرسائل إلى وجهتها النهائية.

من التطبيقات الشائعة الأخرى للإنترنت، هو "البث المباشر" "streaming" للصوت والفيديو. تستخدم خدمات مثل الفيديو عند الطلب video on demand (VoD) وراديو الإنترنت Internet radio هذه التقنية. وبينما نبدأ عادةً من موقع ويب website لبدء جلسة بث، فإن توفير الصوت والفيديو يختلف اختلافًا جوهريًا عن جلب صفحة ويب بسيطة من النصوص والصور. على سبيل المثال، غالبًا ما لا ترغب في تنزيل ملف فيديو كامل - وهي عملية قد تستغرق بضع دقائق - قبل مشاهدة المشهد الأول.

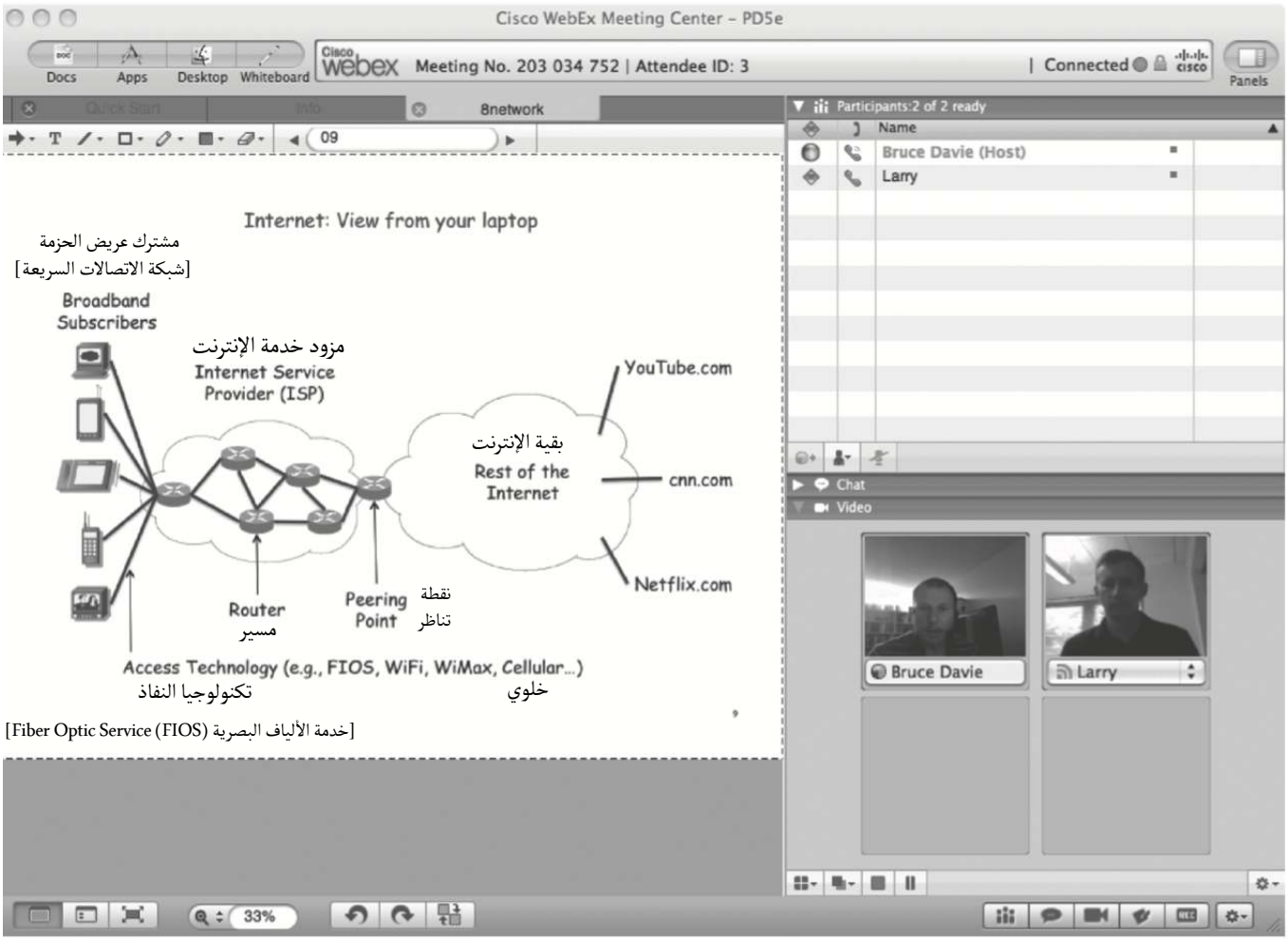
بث الصوت والفيديو يعني نقل الرسائل في الوقت المناسب من المرسل إلى المستقبل، حيث يعرض المستقبل الفيديو أو يشغل الصوت فور وصوله تقريبًا.

تجدر الإشارة إلى أن الفرق بين تطبيقات البث والطريقة التقليدية لنقل صفحة نصية أو صور ثابتة هو أن البشر يستهلكون تدفقات الصوت والفيديو بشكل مستمر، وأن الانقطاع - على شكل أصوات متخطاة أو مقاطع فيديو متقطعة - غير مقبول. في المقابل، يمكن عرض صفحة نصية وقراءتها على شكل أجزاء متفرقة. يؤثر هذا الاختلاف على كيفية دعم الشبكة لهذه الفئات المختلفة من التطبيقات.

تجدر الإشارة إلى أن الفرق بين تطبيقات البث والطريقة التقليدية لنقل صفحة نصية أو صور ثابتة هو أن البشر يستهلكون تدفقات الصوت والفيديو بشكل مستمر، وأن الانقطاع - على شكل أصوات متخطاة أو مقاطع فيديو متقطعة - غير مقبول. في المقابل، يمكن عرض صفحة نصية وقراءتها على شكل أجزاء متفرقة. يؤثر هذا الاختلاف على كيفية دعم الشبكة لهذه الفئات المختلفة من التطبيقات.

> ليس بالضرورة "في أقرب وقت ممكن ... تشير أبحاث العوامل البشرية إلى أن 300 مللي ثانية هو الحد الأقصى المعقول لمدى التأخير ذهابًا وإيابًا round-trip delay الذي يمكن تحمله في مكالمة هاتفية قبل أن يشتكي البشر، ويبدو تأخير لمدة 100 مللي ثانية يعتبر جيدًا جدًا لجودة السمع.

عندما يحاول شخص ما مقاطعة شخص آخر، يجب على الشخص الذي تمت مقاطعته سماع ذلك في أسرع وقت ممكن، ثم يقرر ما إذا كان سيسمح بالمقاطعة أم سيستمر في الحديث مع المقاطع. التأخير المفرط في مثل هذه البيئة يجعل النظام غير صالح للاستخدام. قارن هذا بخدمة الفيديو عند الطلب، حيث إذا استغرقت بضع ثوانٍ من بدء المستخدم تشغيل الفيديو حتى ظهور الصورة الأولى، فإن الخدمة تُعتبر مُرضية. كذلك، عادةً ما تتضمن التطبيقات التفاعلية interactive تدفقات صوتية و/أو مرئية في كلا الاتجاهين، بينما يُرجح أن يُرسل تطبيق البث الفيديو أو الصوت في اتجاه واحد فقط.



الشكل 1: تطبيق الوسائط المتعددة multimedia بما في ذلك مؤتمرات الفيديو

تتوفر أدوات [اجهزة] مؤتمرات الفيديو عبر الإنترنت منذ أوائل التسعينيات، ولكنها حققت انتشارًا واسعًا في السنوات القليلة الماضية، مع طرح العديد من المنتجات التجارية في السوق. يوضح الشكل 1 مثالًا على أحد هذه الأنظمة. وكما أن تنزيل صفحة ويب يتطلب أكثر مما يبدو، فإن الأمر نفسه ينطبق على تطبيقات الفيديو. فعلى سبيل المثال، يُعدّ وضع محتوى الفيديو في شبكة ذات سرعة تدفق منخفضة* low throughput نسبيًا، أو التأكد من تزامن synchronization الفيديو والصوت ووصولهما في الوقت المناسب لتوفير تجربة مستخدم جيدة، من المشكلات التي يجب على مصممي الشبكات والبروتوكولات الاهتمام بها. سنتناول هذه المشكلات وغيرها الكثير من المشكلات المتعلقة بتطبيقات الوسائط المتعددة لاحقًا في هذا الكتاب.

على الرغم من أنهما مجرد مثالين، إلا أن تنزيل الصفحات من الويب web والمشاركة في مؤتمر فيديو يُظهران تنوع التطبيقات التي يُمكن بناؤها على الإنترنت، ويُشيران إلى تعقيد تصميم شبكة الانترنت. سنُطوّر لاحقًا في هذا الكتاب تصنيفًا taxonomy أكثر شمولًا لأنواع التطبيقات للمساعدة في توجيه مناقشتنا لقرارات التصميم الرئيسية أثناء سعينا إلى بناء وتشغيل واستخدام شبكات تُغطي نطاقًا واسعًا من التطبيقات. ويختتم الكتاب بمراجعة تطبيقين محددين، بالإضافة إلى تطبيقات أخرى تُوضح اتساع ماهو ممكن اليوم على الإنترنت.

في الوقت الحالي، ستكون هذه النظرة السريعة على بعض التطبيقات النموذجية كافية لتمكيننا من البدء في النظر في المشكلات التي يتوجب معالجتها to be addressed إذا أردنا بناء شبكة تدعم مثل هذا التنوع في التطبيقات.

---

*[يستعمل المؤلف تعبير حزمة/عرض الطيف bandwidth للتعبير عن سرعة التدفق/الإرسال throughput في شبكة الحاسوب. وهذا شائع نظرا لأن سرعة التدفق في روابط الشبكة تكون متناسبة مع عرض طيفها bandwidth].

1.2: المتطلبات

المتطلبات

لقد وضعنا لأنفسنا هدفًا طموحًا: فهم كيفية بناء شبكة حاسوبية من الصفر. يتمثل نهجنا لتحقيق هذا الهدف في البدء بالمبادئ الأساسية، ثم طرح الأسئلة التي نطرحها عادةً عند بناء شبكة فعلية. في كل خطوة، سنستخدم بروتوكولات اليوم لتوضيح خيارات التصميم المختلفة المتاحة لنا، لكننا لن نقبل هذه الانظمة الحالية على أنها حقائق ثابتة. بدلًا من ذلك، سنطرح (ونجيب على) سؤال لماذا صُممت الشبكات بهذه الطريقة. مع أن الاكتفاء بفهم الطريقة المتبعة اليوم قد يبدو مغريًا، إلا أنه من المهم إدراك المفاهيم الأساسية التحتية، لأن الشبكات تتغير باستمرار مع تطور التكنولوجيا وابتكار تطبيقات جديدة. بناءً على خبرتنا، بمجرد فهم الأفكار الأساسية، سيكون أي بروتوكول جديد تواجهه سهل الاستيعاب نسبيًا.

أصحاب المصلحة

كما ذكرنا سابقًا، يمكن لدارس الشبكات أن يأخذ مناظير متعددة. عندما كتبنا الطبعة الأولى من هذا الكتاب، لم يكن لدى غالبية السكان اتصال بالإنترنت على الإطلاق، أما من كان متصلاً بالإنترنت فقد حصل عليه أثناء العمل أو الجامعة أو عبر مودم الاتصال الهاتفي في المنزل. كانت التطبيقات الشائعة تُعدّ على أصابع اليد. لذلك، وكما هو الحال مع معظم الكتب في ذلك الوقت، ركز كتابنا على منظور من سيصمم معدات وبروتوكولات الشبكات. وما زلنا نركز على هذا المنظور، ونأمل أن تتعرفوا بعد قراءة هذا الكتاب على كيفية تصميم معدات وبروتوكولات الشبكات في المستقبل.

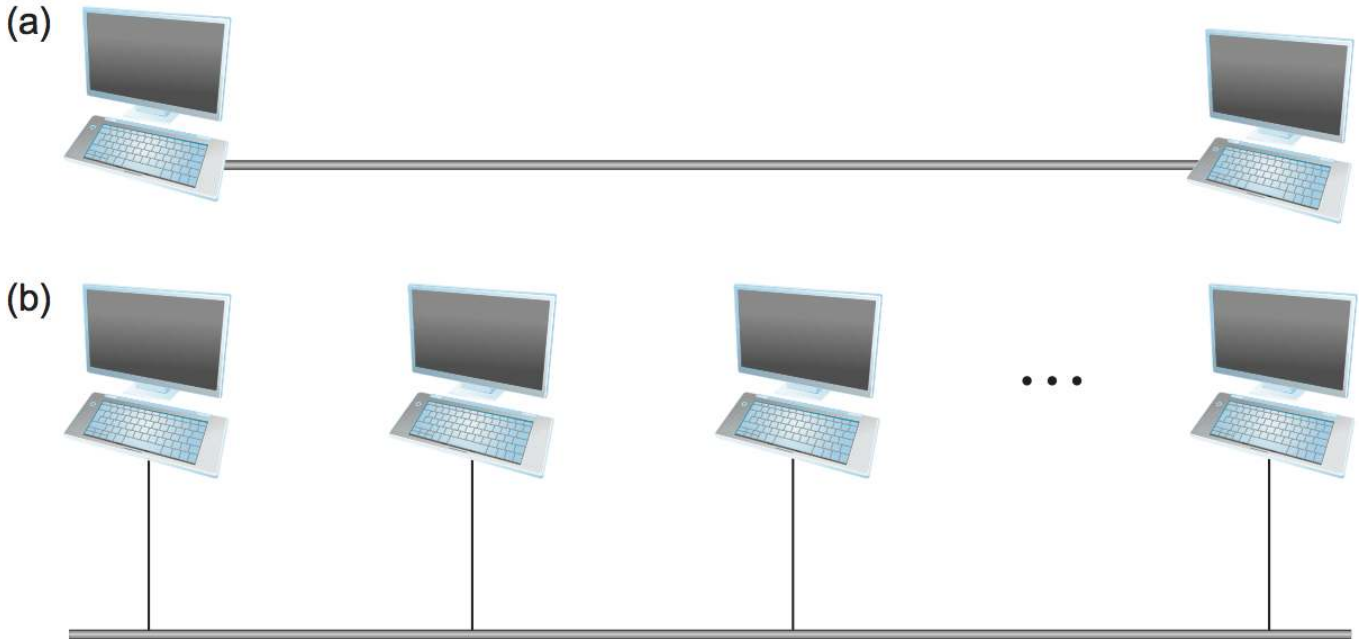
ومع ذلك، نود أيضًا تناول وجهات نظر إثنين إضافيين من أصحاب المصالح: مطورو التطبيقات الشبكية، ومديرو الشبكات أو مشغلوها. دعونا نفكر في كيفية سرد هذه الجهات الثلاث لمتطلباتها المتعلقة بالشبكة:

يحاول هذا القسم تقطير [تلخيص وتنقية] متطلبات أصحاب المصلحة المختلفين في تعريفهم إلى الإعتبارات الرئيسية التي تحرك تصميم الشبكة، ومن خلال القيام بذلك، تحديد التحديات التي سيتم مناقشتها في بقية هذا الكتاب.

قدرة توصيلية [إتصال] قابلة للتوسع Scalable Connectivity

بدءًا من البديهي، يجب أن توفر الشبكة اتصالًا بين مجموعة من أجهزة الكمبيوتر. أحيانًا، يكفي بناء شبكة محدودة تربط عددًا قليلًا من الأجهزة المختارة. في الواقع، ولأسباب تتعلق بالخصوصية والأمان، الهدف الصريح للعديد من الشبكات الخاصة (الشركات) إلى توصيل عدد قليل من الأجهزة. في المقابل، صُممت شبكات أخرى (والإنترنت مثالها الرئيسي) للنمو بطريقة تتيح لها إمكانية ربط جميع أجهزة الكمبيوتر في العالم. ويُقال إن النظام المُصمم لدعم النمو إلى حجم كبير جدًا قابل للتوسع. وباستخدام الإنترنت كنموذج، يتناول هذا الكتاب تحدي قابلية التوسع.

لفهم متطلبات الاتصال بشكل أعمق، علينا أن نلقي نظرة فاحصة على كيفية اتصال أجهزة الكمبيوتر في الشبكة. يحدث الاتصال [القدرة التوصيلية] على مستويات مختلفة. في أدنى مستوى، يمكن أن تتكون الشبكة من جهازي كمبيوتر أو أكثر متصلين مباشرةً بوسيط مادي physical medium، مثل كابل محوري coaxial cable أو[كابل] ليف بصري optical fiber. نُطلق على هذا الوسيط المادي اسم "وصلة" link، وغالبًا ما نُشير إلى أجهزة الكمبيوتر التي يربطها بالعقد nodes. (أحيانًا تكون العقدة قطعةً أكثر تخصصًا من جهاز كمبيوتر، لكننا نتجاهل هذا التمييز لأغراض هذه المناقشة). كما هو موضح في الشكل 1، تقتصر الروابط المادية أحيانًا على زوج من العقد (يُقال إن هذا الرابط من نقطة إلى نقطة point-to-point)، بينما في حالات أخرى، قد تتشارك أكثر من عقدتين رابطًا ماديًا واحدًا (يُقال إن هذا الرابط متعدد الوصول multiple-access). تُعد الروابط اللاسلكية، مثل تلك التي توفرها الشبكات الخلوية وشبكات Wi-Fi، فئة مهمة من روابط الوصول المتعدد. ودائمًا ما تكون روابط الوصول المتعدد محدودة الحجم، من حيث المسافة الجغرافية التي تغطيها وعدد العقد التي يمكنها الاتصال بها.

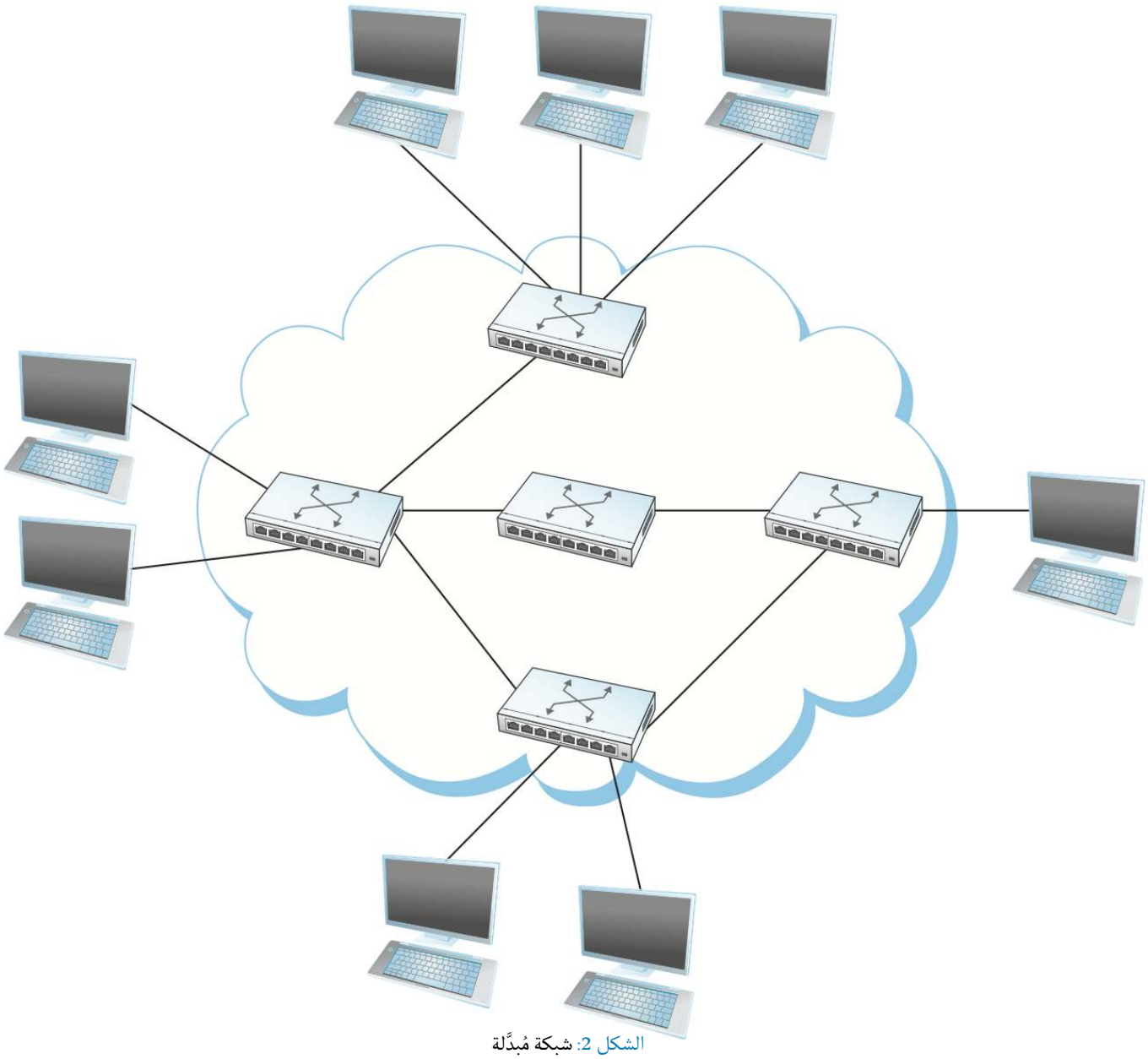


الشكل 1: الروابط المباشرة: (a) من نقطة إلى نقطة؛ (b) متعدد الوصول

إذا اقتصرت شبكات الحاسوب على الحالات التي تتصل فيها جميع العقد ببعضها البعض مباشرةً عبر وسيط مادي مشترك، فإما أن تكون الشبكات محدودة للغاية في عدد الحواسيب التي يمكنها توصيلها، أو أن يصبح عدد الأسلاك الخارجة من الجزء الخلفي لكل عقدة سريعًا غير قابل للإدارة ومكلفًا للغاية. لحسن الحظ، لا يعني الاتصال بين عقدتين بالضرورة وجود اتصال مادي مباشر بينهما - إذ يمكن تحقيق اتصال غير مباشر بين مجموعة من العقد المتعاونة. لننظر إلى المثالين التاليين لكيفية ربط مجموعة من الحواسيب بشكل غير مباشر.

يوضح الشكل 2 زوجًا من العقد، كل منها متصل برابط واحد أو أكثر من روابط نقطة إلى نقطة point-to-point. تُشغّل هذه العقد المتصلة برابطين على الأقل برنامجًا software يُعيد توجيه forward البيانات data المُستلَمة received من رابط link لتخرج من رابط آخر. إذا نُظِّمت هذه العقد المُوجِّهة بطريقة منهجية systematic، فإنها تُشكِّل شبكة مُبدَّلة switched-network. هناك أنواع عديدة من الشبكات المُبدَّلة، وأكثرها شيوعًا هما تبديل االدارات circuit-switched والتبديل الحزمي packet-switched. يُستخدم النوع الأول [تبديل الدارات] بشكل بارز في نظام الهاتف، بينما يُستخدم النوع الثاني في الغالبية العظمى من شبكات الحاسوب، وسيكون محور هذا الكتاب. (مع ذلك، يشهد تبديل الدارات عودةً قويةً في مجال الشبكات الضوئية، وهو أمرٌ يتبين أهميته مع تزايد الطلب على سعة الشبكة باستمرار).

تكمن الميزة المهمة للشبكات المُبدَّلة بالحزم في أن العقد في هذه الشبكة تُرسل كتلًا مُنفصلة من البيانات discrete blocks إلى بعضها البعض. لنعتبر هذه الكتل من البيانات مُقابلةً لبعض بيانات التطبيق application data، مثل ملف file أو رسالة بريد إلكتروني email أو صورة image. نطلق على كل كتلة من البيانات اسم حزمة packet أو رسالة message، وفي الوقت الحالي نستخدم هذين المصطلحين بالتبادل.

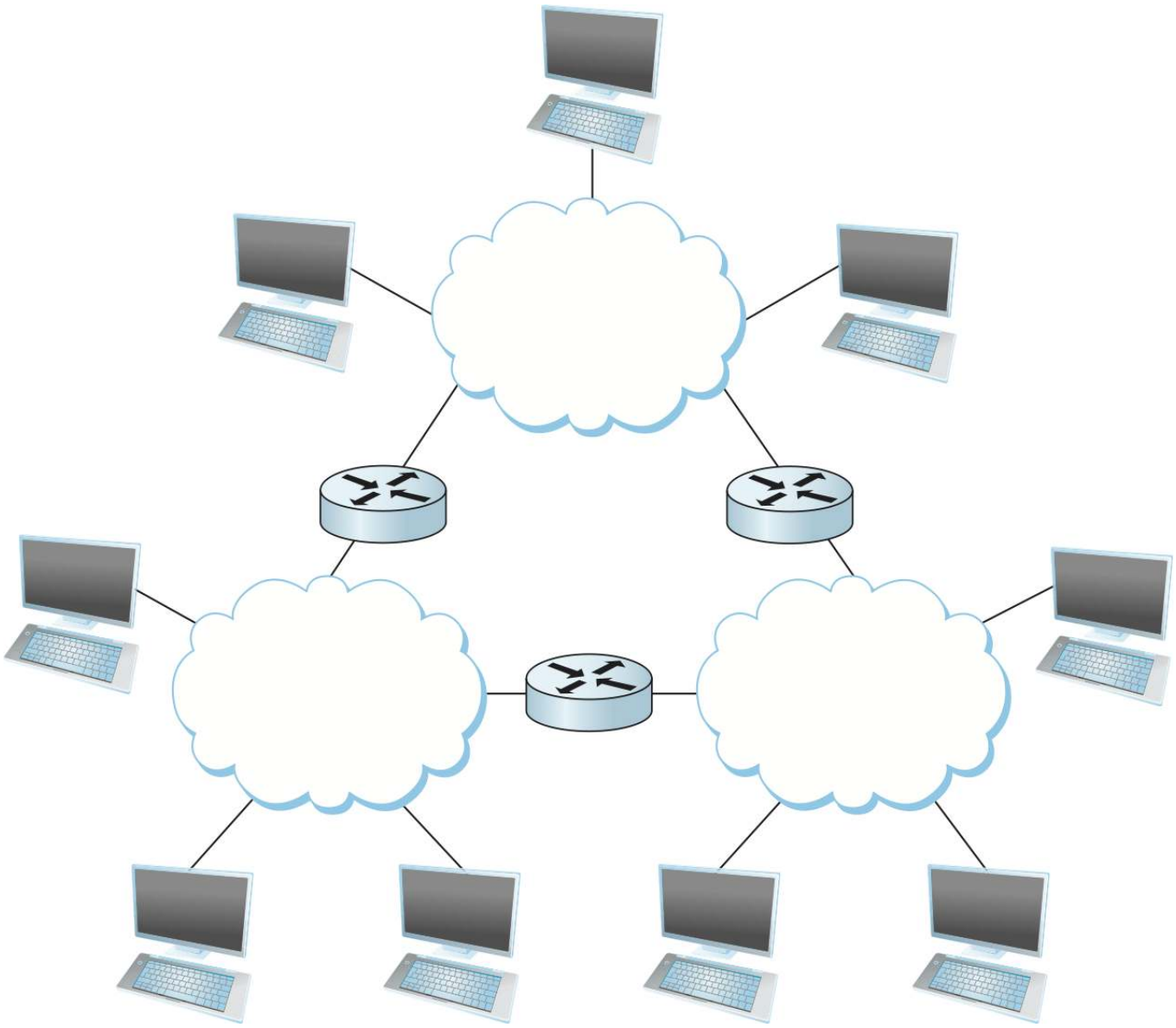


الشكل 2: شبكة مُبدَّلة

تستخدم شبكات تبديل الحزم عادةً استراتيجية تُسمى التخزين والتوجيه store-and-forward. وكما يوحي الاسم، تستقبل كل عقدة في شبكة التخزين والتوجيه أولًا حزمة كاملة عبر رابط ما، وتخزنها في ذاكرتها الداخلية، ثم تُعيد توجيهها إلى العقدة التالية. في المقابل، تُنشئ شبكة تبديل الدارات أولًا دارة circuit مخصصة عبر سلسلة من الروابط، ثم تسمح للعقدة المصدر بإرسال تدفق stream من البتات bits عبر هذه الدارة إلى العقدة الوجهة destination node. السبب الرئيسي لاستخدام تبديل الحزم بدلًا من تبديل الدارات في شبكات الحاسوب هو الكفاءة، وهو ما سنناقشه في القسم [الفصل] الفرعي التالي.

تميز السحابة cloud في الشكل 2 بين العقد الداخلية التي تُنفّذ الشبكة (تُسمى عادةً المُبدّلات switches، ووظيفتها الأساسية هي تخزين الحزم وإعادة توجيهها) والعقد الخارجية التي تستخدم الشبكة (تُسمى عادةً المُضيفات، وهي تدعم المستخدمين وتُشغّل برامج التطبيقات). تجدر الإشارة أيضًا إلى أن السحابة تُعدّ من أهم رموز شبكات الحاسوب. بشكل عام، نستخدم كلمة "سحابة" للإشارة إلى أي نوع من الشبكات، سواءً كانت رابطًا واحدًا من نقطة إلى نقطة، أو رابطًا متعدد الوصول، أو شبكة مُبدّلة. وبالتالي، عندما ترى كلمة "سحابة" مُستخدمة في أي شكل، يُمكنك اعتبارها رمزًا لأيٍّ من تقنيات الشبكات التي يتناولها هذا الكتاب.

> ومن المثير للاهتمام أن استخدام مصطلح السحاب بهذه الطريقة يسبق مصطلح الحوسبة السحابية cloud computing بعقدين من الزمن على الأقل، ولكن هناك صلة بين هذين الاستخدامين، والتي سنناقشها لاحقًا.



الشكل 3: شبكات مترابطة Interconnecting networks

يوضح الشكل 3 طريقةً ثانيةً لتوصيل مجموعة من أجهزة الكمبيوتر بشكل غير مباشر في هذه الحالة، تترابط مجموعة من الشبكات المستقلة (السحابات) لتكوين شبكة الشبكات internetwork, أو ما يُعرف اختصارًا بـ "الإنترنت" Internet. نعتمد على مصطلح الإنترنت للإشارة إلى الشبكة العامة للشبكات بحرف i الصغير، وشبكة TCP/IP العاملة حاليًا بحرف I الكبير. تُسمى العقدة المتصلة بشبكتين أو أكثر عادةً مسير [جهاز توجيه] router أو بوابة gateway، وتلعب نفس دور المُبدِّل switch تقريبًا - فهي تُعيد توجيه الرسائل من شبكة إلى أخرى. تجدر الإشارة إلى أنه يمكن اعتبار الإنترنت بحد ذاته نوعًا آخر من الشبكات، مما يعني أنه يمكن بناء الإنترنت من شبكة إنترنت واحدة. وبالتالي، يمكننا بناء شبكات كبيرة بشكل متكرر عن طريق ربط السحابات لتشكيل السحابات أكبر. يمكن القول بشكل معقول أن فكرة ربط الشبكات المختلفة على نطاق واسع كانت الابتكار الأساسي للإنترنت وأن النمو الناجح للإنترنت إلى حجم عالمي ومليارات العقد كان نتيجة لبعض قرارات التصميم الجيدة للغاية التي اتخذها مهندسو الإنترنت الأوائل، والتي سنناقشها لاحقًا.

إن مجرد اتصال مجموعة من المضيفين hosts ببعضهم البعض، بشكل مباشر أو غير مباشر، لا يعني بالضرورة نجاحنا في توفير اتصال بين المضيفين host-to-host connectivity. الشرط الأخير هو أن تتمكن كل عقدة من تحديد أي من العقد الأخرى على الشبكة تريد التواصل معها. يتم ذلك بتعيين عنوان address لكل عقدة. العنوان هو سلسلة بايتات تُعرّف العقدة؛ أي أن الشبكة يمكنها استخدام عنوان العقدة لتمييزها عن العقد الأخرى المتصلة بها. عندما تريد عقدة مصدر source node من الشبكة توصيل رسالة إلى عقدة وجهة destination node معينة، فإنها تُحدد عنوان عقدة الوجهة.

إذا لم تكن عقدتا الإرسال والاستقبال متصلتين مباشرةً، فإن مبدلات [بوابات gateways] الشبكة وأجهزة التوجيه تستخدم هذا العنوان لتحديد كيفية توجيه الرسالة إلى الوجهة. وتُسمى عملية تحديد كيفية توجيه الرسائل بشكل منهجي إلى عقدة الوجهة بناءً على عنوانها التوجيه routing.

افترضت هذه المقدمة الموجزة للتوجيه والعنونة addressing أن العقدة المصدر ترغب في إرسال رسالة أحادية الوجهة [إلى عقدة وجهة واحدة] unicast message [أحادي البث]. وبينما يُعد هذا السيناريو الأكثر شيوعًا، فمن الممكن أيضًا أن ترغب العقدة المصدر بارسال رسالة بث شامل broadcast message إلى جميع العقد على الشبكة. أو قد ترغب العقدة المصدر في إرسال رسالة إلى مجموعة فرعية من العقد الأخرى، وليس جميعها، وهو ما يُسمى برسالة البث المتعدد multicast message. وبالتالي، بالإضافة إلى العناوين الخاصة بكل عقدة، فإن من متطلبات الشبكة أيضًا دعم عناوين البث المتعدد والبث الشامل.

النقطة الرئيسية

الفكرة الرئيسية المستخلصة من هذه المناقشة هي أنه يُمكن تعريف الشبكة بشكل عودي [متكرر] recursively على أنها تتكون من عقدتين أو أكثر متصلة بوصلة مادية، أو كشبكتين أو أكثر متصلة بعقدة. بمعنى آخر، يُمكن بناء شبكة من خلال تداخل الشبكات nesting of networks، حيث تُنفذ الشبكة في المستوى الأدنى بواسطة وسيط مادي. من بين التحديات الرئيسية في توفير اتصال الشبكة تحديد عنوان لكل عقدة يُمكن الوصول إليه على الشبكة (بما في ذلك دعم البث الشامل والبث المتعدد)، واستخدام هذه العناوين لإعادة توجيه الرسائل نحو عقدة (عقد) الوجهة المناسبة.

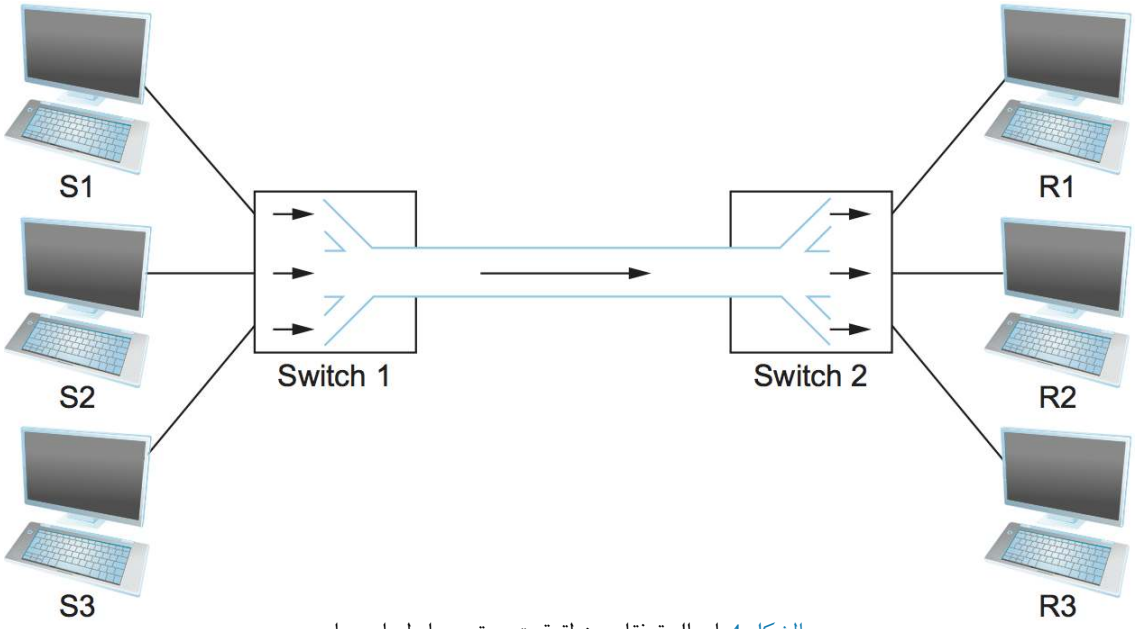
تقاسم فعال إقتصاديا للموارد Cost-Effective Resource Sharing

كما ذُكر سابقًا، يُركز هذا الكتاب على شبكات تحويل الحزم. ويشرح هذا القسم الشرط الأساسي لشبكات الحاسوب، ألا وهو الكفاءة، والذي يدفعنا لاختيار تحويل الحزم كاستراتيجية مُثلى.

إذا كانت هناك مجموعة من العقد متصلة بشكل غير مباشر عبر شبكة متداخلة، فمن الممكن لأي زوج من المضيفين hosts إرسال رسائل إلى بعضهما البعض عبر سلسلة من الروابط والعقد. بالطبع، لا نريد دعم زوج واحد فقط من المضيفين المتصلين، بل نريد تزويد جميع أزواج المضيفين بالقدرة على تبادل الرسائل. السؤال إذن هو كيف تتشارك جميع المضيفات الراغبة في التواصل الشبكة، خاصةً إذا كانت ترغب في استخدامها في الوقت نفسه ؟ وكما لو أن هذه المشكلة ليست صعبة بما فيه الكفاية، فالسؤال كيف تتشارك عدة مضيفات نفس الرابط عندما يرغبون جميعًا في استخدامه في الوقت نفسه ؟

لفهم كيفية مشاركة الأجهزة المضيفة لشبكة، علينا تقديم مفهوم أساسي، وهو التضميم multiplexing، والذي يعني مشاركة موارد النظام بين عدة مستخدمين. بديهيًا، يمكن تفسير االتضميم بمقارنة نظام حاسوبي قائم على تقاسم الوقت، حيث تتم مشاركة sharing معالج مادي physical processor واحد (التضميم) بين عدة وظائف، كل منها يعتقد أن لديه معالجًا خاصًا به. وبالمثل، يمكن إرسال البيانات المرسلة data من عدة مستخدمين عبر الروابط المادية التي تُشكل الشبكة.

لمعرفة كيفية عمل ذلك، لننظر إلى الشبكة البسيطة الموضحة في الشكل 4، حيث تُرسل الأجهزة المضيفة الثلاثة على يسار الشبكة (المرسلون S1-S3) البيانات إلى الأجهزة المضيفة الثلاثة على اليمين (المستقبلون R1-R3) من خلال مشاركة شبكة مبتدلة [مُبدّلة] switched network تحتوي على رابط مادي واحد فقط. (للتبسيط، افترض أن الجهاز المضيف S1 يُرسل البيانات إلى الجهاز المضيف R1، وهكذا). في هذه الحالة، يتم إرسال ثلاثة تدفقات بيانات - تُقابل ثلاثة أزواج من الأجهزة المضيفة - إلى رابط مادي واحد بواسطة االمبدلة switch 1، ثم يُعاد إرسالها إلى تدفقات منفصلة بواسطة المبدلة switch 2. لاحظ أننا نُتعمّد الغموض بشأن ما يعني "تدفق البيانات". لأغراض هذه المناقشة، افترض أن كل جهاز مضيف على اليسار لديه كمية كبيرة من البيانات التي يُريد إرسالها إلى نظيره على اليمين.



الشكل 4: إرسال تدفقات منطقية متعددة عبر رابط مادي واحد.

هناك عدة طرق مختلفة للتضميم تدفقات flows متعددة إلى رابط مادي واحد. إحدى الطرق الشائعة هي الإرسال المتعدد بتقسيم الزمن المتزامن synchronous time-division multiplexing (STDM). تتمثل فكرة STDM في تقسيم الزمن إلى فترات متساوية [تدعى] بالكم الزمني quanta [حصة زمنية time-slot]، وإعطاء كل تدفق فرصة لإرسال بياناته عبر الرابط المادي بطريقة دورية round-robin fashion. بمعنى آخر، خلال الكم الزمني 1، تُنقل البيانات من S1 إلى R1؛ وخلال الكم الزمني 2، تُنقل البيانات من S2 إلى R2؛ وفي الكم الزمني 3، يُرسل S3 البيانات إلى R3. عند هذه النقطة، يُعاد إرسال التدفق الأول (S1 إلى R1)، وتتكرر العملية. هناك طريقة أخرى هي التضمين [الإرسال المتعدد] بتقسيم التردد frequency-division multiplexing (FDM). تتمثل فكرة FDM في إرسال كل تدفق عبر الرابط المادي بتردد frequency مختلف، بشكل مماثل للطريقة التي تُنقل بها إشارات محطات التلفزيون المختلفة بتردد مختلف عبر موجات الأثيرairwaves أو عبر وصلة تلفزيونية بكابل محوري coaxial cable.

على الرغم من سهولة فهم كلٍّ من STDM وFDM، إلا أنهما محدودان بطريقتين. أولًا، إذا لم يكن لدى أحد التدفقين (أزواج المضيفين host pairs) أي بيانات لإرسالها، فإن حصته من الرابط المادي - أي كميته الزمنية أو تردده - تظل خاملة idle [غير مستعملة]، حتى لو كان لدى أحد التدفقين الآخرين بيانات لإرسالها. على سبيل المثال، اضطر S3 إلى انتظار دوره خلف S1 وS2 في الفقرة السابقة، حتى لو لم يكن لدى S1 وS2 أي بيانات لإرسالها. بالنسبة لاتصالات الكمبيوتر، يمكن أن يكون مقدار الوقت الذي يكون فيه الرابط خاملاً كبيرًا جدًا - على سبيل المثال، خذ في الاعتبار مقدار الوقت الذي تقضيه في قراءة صفحة ويب web page (ترك الرابط خاملاً) مقارنةً بالوقت الذي تقضيه في جلب fetching الصفحة. ثانيًا، يقتصر كلٌّ من STDM وFDM على الحالات التي يكون فيها الحد الأقصى لعدد التدفقات ثابتًا ومعروفًا مسبقًا. ليس من العملي تغيير حجم الكم أو إضافة كميات إضافية في حالة STDM أو إضافة ترددات جديدة في حالة FDM.

إن شكل الإرسال االمضمم الذي يعالج هذه العيوب، والذي نستعين به كثيرًا في هذا الكتاب، يُسمى الإرسال المضمم الإحصائي statistical multiplexing. على الرغم من أن الاسم ليس مفيدًا تمامًا لفهم االمبدأ concept، إلا أن الإرسال المضمم الإحصائي بسيط للغاية، ويقوم على فكرتين رئيسيتين. أولًا، يُشبه STDM من حيث مشاركة الرابط المادي عبر الوقت - أولًا تُنقل البيانات من تدفق واحد عبر الرابط المادي، ثم تُنقل البيانات من تدفق آخر، وهكذا. ولكن بخلاف STDM، تُنقل البيانات من كل تدفق عند الطلب on demand بدلاً من خلال فترة زمنية محددة مسبقًا. وبالتالي، إذا كان لدى تدفق واحد فقط بيانات لإرسالها، فإنه يُرسلها دون انتظار وصول الكم الخاص به، وبالتالي دون الحاجة إلى مراقبة مرور الكميات المخصصة للتدفقات الأخرى دون استخدامها. إن تجنب وقت الخمول هذا هو ما يمنح تبديل الحزم كفاءته.

وضمن تعريفنا حتى الآن، لا يمتلك الإرسال المضمم الإحصائي آليةً تضمن وصول جميع التدفقات إلى دورها في الإرسال عبر الرابط المادي. أي أنه بمجرد أن يبدأ تدفقٌ ما بإرسال البيانات، نحتاج إلى طريقةٍ ما لتحديد الإرسال، بحيث تتمكن التدفقات الأخرى من الوصول إلى دورها. ولتلبية هذه الحاجة، يُحدد الإرسال المتعدد الإحصائي حدًا أقصى upper bound لحجم كتلة البيانات data block التي يُسمح لكل تدفق بإرسالها في أي وقتٍ مُحدد. عادةً ما يُشار إلى هذه الكتلة المحدودة من البيانات باسم "الحزمة"، لتمييزها عن الرسالة الكبيرة التي قد يرغب برنامج تطبيقي في إرسالها. ولأن شبكة تبديل الحزم تُحدد الحد الأقصى لحجم الحزم، فقد لا يتمكن المُضيف من إرسال رسالة كاملة في حزمة واحدة. قد يحتاج المصدر إلى تجزئة fragment الرسالة إلى عدة حزم، وإعادة تجميع reassembling الحزم في الرسالة الأصلية في الوجهة [النهائية].

بمعنى آخر، يُرسل كل تدفق سلسلة من الحزم عبر الرابط المادي، مع اتخاذ قرار بشأن كل حزمة على حدة بشأن حزمة التدفق التي سيُرسلها. لاحظ أنه إذا كان هناك تدفق واحد فقط للبيانات المرسلة، فيمكن إرسال سلسلة متتالية من الحزم؛ أما إذا كان هناك أكثر من تدفق بيانات للإرسال، فسيتم تداخل حزمها على الرابط. يوضح الشكل 4 مُبدِّلًا يُرسل حزمًا متعددة من مصادر متعددة إلى رابط مشترك واحد.

يمكن اتخاذ قرار تحديد الحزمة التالية اللإرسال عبر رابط مُشترك shared link بعدة طرق. على سبيل المثال، في شبكة مُكوّنة من مُبدِّلات مُترابطة interconnected switches بروابط، مثل تلك المُوضّحة في الشكل 4، يُتّخذ القرار من قِبَل المُبدِّل الذي يُرسِل الحزم إلى الرابط المُشترك. (كما سنرى لاحقًا، لا تتضمن جميع شبكات تبديل الحزم مُبدِّلات بالفعل، وقد تستخدم آليات أخرى لتحديد الحزمة التي ستُرسَل إلى الرابط بعد ذلك). يتخذ كل مُبدِّل في شبكات تبديل الحزم packet switched networks هذا القرار بشكل مُستقل، على أساس كل حزمة على حدة. إحدى المُشكلات التي تُواجه مُصمّم الشبكة هي كيفية اتخاذ هذا القرار بطريقة عادلة. على سبيل المثال، يُمكن تصميم مُبدِّل لخدمة الحزم على أساس "الداخل أولاً يخرج أولاً" first-in first-out (FIFO) basis. وهناك نهج آخر يتمثل في إرسال الحزم من كل تدفق من التدفقات المُختلفة عبر المُبدِّل بطريقة "دورية" round-robin manner. قد يُجرى ذلك لضمان حصول تدفقات معينة على حصة معينة من سعة الموارد [الطيف] bandwidth للرابط ، أو لضمان عدم تأخر حزمها في المحول لأكثر من فترة زمنية محددة. يُقال أحيانًا إن الشبكة التي تحاول تخصيص سعة الموارد لتدفقات معينة لدعم [مرتبة] **جودة الخدمة quality of service (QoS)**.

لاحظ أيضًا في الشكل 4 أنه نظرًا لأن المبدل "switch 1" مُلزمٌ بإرسال ثلاثة تدفقات حزم واردة إلى رابط صادر واحد، فمن المُحتمل أن يستقبل االمبدل حزمًا أسرع من قدرة للاستيعاب للرابط الصادر المُشترك. في هذه الحالة، يُجبر المبدل على تخزين مؤقت buffer هذه الحزم في ذاكرته memory. إذا استقبل االمبدل حزمًا أسرع من قدرته على إرسالها لفترة طويلة، فسيستنفد المحول في النهاية مساحة التخزين المؤقتة buffer space، وسيُضطر إلى إسقاط drop بعض الحزم. عندما يعمل المحول في هذه الحالة state، يُقال إنه مُزدحم congested.

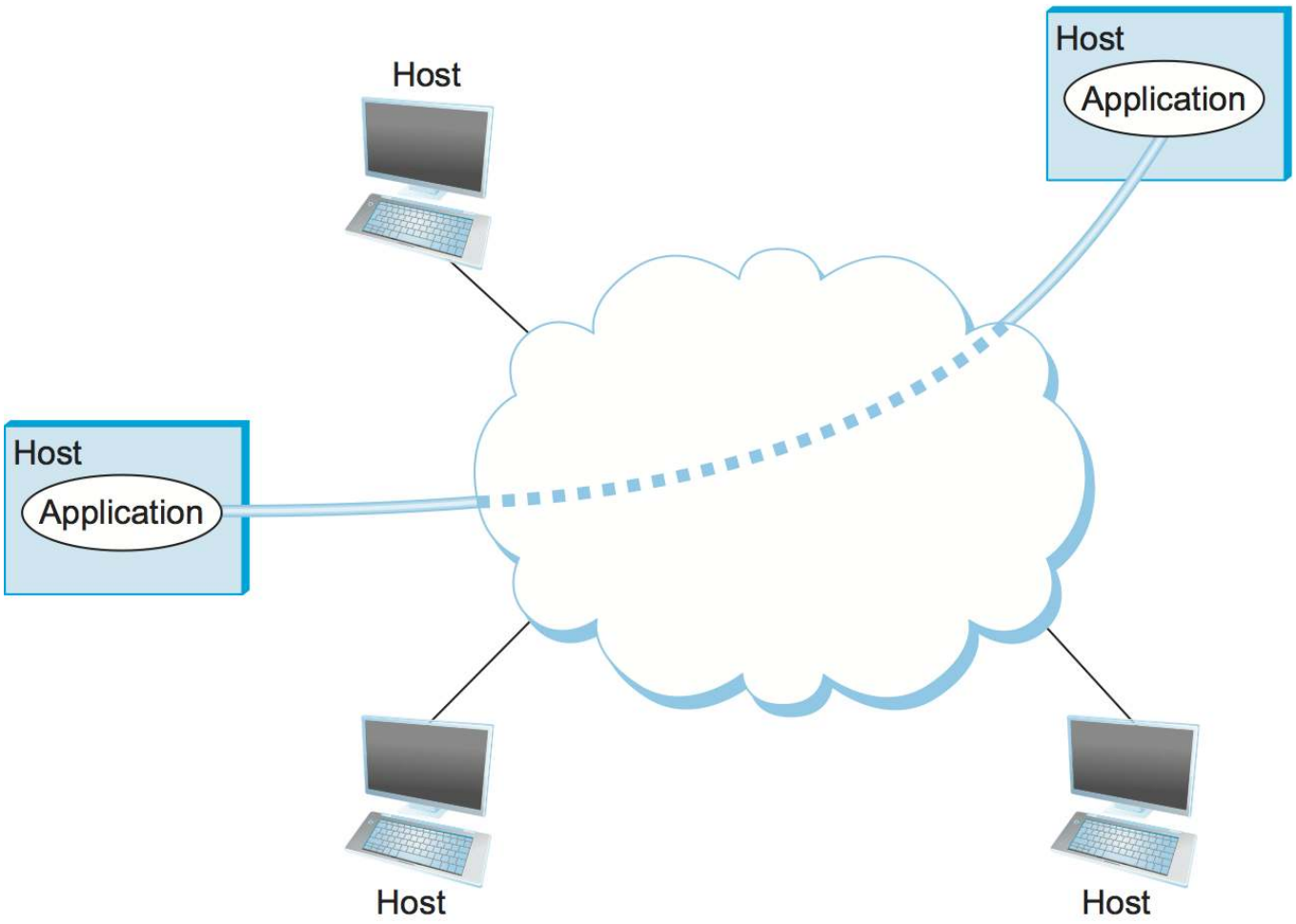
النقطة الرئيسية

خلاصة القول [في المحصلة] هي أن التضميم [المتعدد] الإحصائي يبعد طريقةً [فعالة] اقتصادية cost effective لخدمة مستخدمين متعددين (مثل تدفقات البيانات من مضيف إلى مضيف) لمشاركة موارد الشبكة (الروابط والعقد) بطريقة دقيقة. ويُعرّف الحزمة بأنها مستوى التفصيل granularity الذي تُخصص به روابط الشبكة [مواردها] لتدفقات مختلفة، حيث يتمكن كل مُبدّل من جدولة استخدام الروابط المادية المتصلة به على أساس كل حزمة. ويُعدّ توزيع سعة الرابط link capacity بشكل عادل على التدفقات المختلفة وويعد التعامل مع الازدحام عند حدوثه من التحديات الرئيسية للتضميم الإحصائي.

دعم الخدمات المعتادة

ركزت المناقشة السابقة على التحديات التي ينطوي عليها توفير اتصال/ترابط connectivity فعال من حيث التكلفة بين مجموعة من الأجهزة المضيفة، ولكن ذلك مبالغة في التبسيط في اعتبار شبكة الحاسوب مجرد توصيل حزم البيانات بين مجموعة من الأجهزة. من الأدق اعتبار الشبكة وسيلةً للتواصل بين مجموعة من عمليات التطبيقات الموزعة على تلك الأجهزة. بعبارة أخرى، يتمثل الشرط التالي لشبكة الحاسوب في أن تكون برامج التطبيقات التي تعمل على الأجهزة المضيفة المتصلة بالشبكة قادرة على التواصل بشيء له مغزى [بفعالية]. من منظور [وجهة نظر] مطور التطبيقات يجب أن تُسهّل الشبكة حياته [عمله].

عندما يحتاج إثنان من البرامج التطبيقية application programs للتواصل مع بعضهما البعض، لا بد من حدوث أمور معقدة تتجاوز مجرد إرسال رسالة من مضيف إلى آخر. أحد الخيارات هو أن يقوم مصممو التطبيقات ببناء كل هذه الوظائف المعقدة في كل برنامج تطبيق. ولكن، نظرًا لأن العديد من التطبيقات تحتاج إلى خدمات مشتركة [ذات الخدمات في كل تطبيق]، فمن المنطقي أكثر بكثير تنفيذ هذه الخدمات المشتركة مرة واحدة ثم السماح لمصمم التطبيق باستخدامها عند بناء التطبيقات. يكمن التحدي الذي يواجه مصمم الشبكة في تحديد المجموعة الصحيحة من الخدمات المشتركة. الهدف هو إخفاء تعقيد الشبكة عن التطبيق دون إثقال كاهل مصمم التطبيق.



الشكل 5: تواصل بين العمليات processes عبر قناة مجردة abstract channel

بديهيًا، ننظر إلى الشبكة على أنها توفر قنوات منطقية تتواصل من خلالها عمليات [إجرائيات] طبقة التطبيق application-level processes مع بعضها البعض؛ وتوفر كل قناة مجموعة الخدمات services التي يتطلبها هذا التطبيق. بمعنى آخر، كما نستخدم السحابة لتمثيل الاتصال بين مجموعة من أجهزة الكمبيوتر بشكل تجريدي، فإننا نعتبر القناة channel الآن بمثابة ربط عملية process بأخرى. يوضح الشكل 5 زوجًا من عمليات طبقة التطبيق تتواصل عبر قناة منطقية logical channel، والتي بدورها تُنفذ فوق سحابة تربط مجموعة من الأجهزة المضيفة. يمكننا اعتبار القناة بمثابة أنبوب pipe يربط بين تطبيقين، بحيث يمكن لتطبيق الإرسال وضع البيانات في أحد طرفيه وتوقع تسليمها عبر الشبكة إلى التطبيق في الطرف الآخر من الأنبوب.

يكمن التحدي في تحديد الوظائف التي ينبغي أن توفرها القنوات لبرامج التطبيقات. على سبيل المثال، هل يتطلب التطبيق ضمانًا بتسليم الرسائل المرسلة عبر القناة، أم أنه من المقبول عدم وصول بعض الرسائل؟ هل من الضروري أن تصل الرسائل إلى جهاز الاستقبال بنفس ترتيب إرسالها، أم أن جهاز الاستقبال لا يكترث بترتيب وصول الرسائل؟ هل تحتاج الشبكة إلى ضمان عدم قدرة أي طرف ثالث على التنصت eavesdrop على القناة، أم أن الخصوصية privacy ليست محل اهتمام؟ بشكل عام، توفر الشبكة أنواعًا مختلفة من القنوات، حيث يختار كل تطبيق النوع الذي يلبي احتياجاته على أفضل وجه. يوضح الجزء المتبقي من هذا القسم التفكير المتبع في تحديد القنوات المفيدة.

تحديد أنماط الاتصال الشائعة

يتضمن تصميم القنوات المجردة أولاً فهم احتياجات الاتصالات لمجموعة تمثيلية من التطبيقات، ثم استخراج متطلبات الاتصال المشتركة بينها، وأخيراً دمج الوظيفة functionality التي تلبي هذه المتطلبات في الشبكة.

من أقدم التطبيقات المدعومة على أي شبكة برامج الوصول إلى الملفات file access مثل بروتوكول نقل الملفات File Transfer Protocol (FTP) و نظام ملفات الشبكة Network File System (NFS). ورغم اختلاف العديد من التفاصيل - على سبيل المثال، ما إذا كانت الملفات تُنقل كاملةً عبر الشبكة أم تُقرأ/تُكتب read/write أجزاءً single blocks منها فقط في وقتٍ معين - إلا أن عنصر الاتصال في الوصول إلى الملفات عن بُعد remote file access يتميز بزوجٍ من العمليات، إحداها تطلب قراءة أو كتابة ملف، والأخرى تُلبّي هذا الطلب. تُسمى العملية process التي تطلب الوصول إلى الملف "الزبون" client، بينما تُسمى العملية التي تدعم الوصول إليه "المخدم" server.

تتضمن قراءة ملف إرسال الزبون رسالة طلب صغيرة إلى الخادم، فيرد الخادم برسالة كبيرة تحتوي على البيانات الموجودة في الملف. أما الكتابة فتتم بطريقة معاكسة، حيث يرسل الزبون رسالة كبيرة تحتوي على البيانات المراد كتابتها إلى الخادم، فيرد المخدم برسالة صغيرة تؤكد إتمام عملية الكتابة على القرص [تخزين].

المكتبة الرقمية digital library تطبيق أكثر تطورًا من [تطبيق] نقل الملفات، لكنها تتطلب خدمات اتصال مماثلة. على سبيل المثال، تدير جمعية آلات الحوسبة Association of Computing Machinery (ACM) مكتبة رقمية ضخمة لأدبيات علوم الحاسوب في

```
http://portal.acm.org/dl.cfm
```

تتمتع هذه المكتبة بمجموعة واسعة من ميزات البحث searching والتصفح browsing لمساعدة المستخدمين في العثور على المقالات التي يريدونها، ولكن في نهاية المطاف فإن معظم ما تفعله هو الاستجابة لطلبات المستخدمين للحصول على الملفات، مثل النسخ الإلكترونية من المقالات الصحفية.

باستخدام قدرة النفاذ إلى الملفات، والمكتبة الرقمية digital library، وتطبيقي الفيديو الموصوفين في المقدمة (مؤتمرات الفيديو والفيديو عند الطلب) كعينة تمثيلية، قد نقرر توفير النوعين التاليين من القنوات: قنوات الطلب/الرد request/reply channels وقنوات تدفق الرسائل message stream channels. تُستخدم قناة الطلب/الرد من قِبل تطبيقات نقل الملفات والمكتبة الرقمية. تضمن هذه القناة استلام الطرف الآخر لكل رسالة يرسلها أحد الطرفين، وتسليم نسخة واحدة فقط من كل رسالة. كما تحمي قناة الطلب/الرد خصوصية وسلامة البيانات التي تتدفق عبرها، بحيث لا تتمكن الأطراف غير المصرح لها من قراءة أو تعديل البيانات المتبادلة بين عمليات العميل والخادم.

يمكن استخدام قناة تدفق الرسائل من قِبل كلٍّ من تطبيقات الفيديو عند الطلب ومؤتمرات الفيديو، شريطة أن تكون مُعَيَّنة لدعم حركة المرور traffic أحادية الاتجاه one-way وثنائية الاتجاه two-way، وأن تدعم خصائص تأخير delay متنوعة. قد لا تحتاج قناة تدفق الرسائل إلى ضمان تسليم جميع الرسائل، إذ يُمكن لتطبيق الفيديو العمل بكفاءة حتى في حال عدم استلام بعض إطارات frames الفيديو. مع ذلك، يجب ضمان وصول الرسائل المُسلَّمة بنفس ترتيب إرسالها، لتجنب عرض إطارات خارج التسلسل out of sequence. وكما هو الحال مع قناة الطلب/الرد، قد ترغب قناة تدفق الرسائل في ضمان خصوصية بيانات الفيديو وسلامتها. وأخيرًا، قد تحتاج قناة تدفق الرسائل إلى دعم البث المتعدد multicast، بحيث يُمكن لأطراف متعددة المشاركة في المؤتمر الهاتفي أو مشاهدة الفيديو.

في حين أنه من الشائع أن يسعى مصممو الشبكات إلى أقل عدد من أنواع القنوات المجردة التي يمكنها خدمة أكبر عدد من التطبيقات، إلا أن هناك خطرًا في محاولة الإفلات بعدد قليل جدًا من القنوات التجريدية. ببساطة، إذا كان لديك مطرقة، فسيبدو كل شيء مسمارًا. على سبيل المثال، إذا كان كل ما لديك هو قنوات تدفق الرسائل والطلب/الرد، فسيكون من المغري استخدامها للتطبيق التالي الذي يظهر، حتى لو لم يوفر أيٌّ من النوعين الدلالات semantics التي يحتاجها التطبيق بالضبط. وبالتالي، من المرجح أن يبتكر مصممو الشبكات أنواعًا جديدة من القنوات - ويضيفون خيارات إلى القنوات الحالية - طالما أن مبرمجي التطبيقات يبتكرون تطبيقات جديدة.

لاحظ أيضًا أنه بغض النظر عن الوظيفة التي توفرها قناة معينة، يبقى السؤال: أين تُطبّق هذه الوظيفة؟ في كثير من الحالات، يكون من الأسهل اعتبار اتصال المضيف بالمضيف في الشبكة الأساسية مجرد توفير لقناة بت bit pipe، مع توفير أية دلالات اتصال عالية المستوى في المضيفين النهائيين [على الطرفين]. تكمن ميزة هذا النهج في أنه يُبقي المُبدِّلات في منتصف الشبكة بسيطة قدر الإمكان - فهي ببساطة تُعيد توجيه الحزم - ولكنه يُلزم المضيفين النهائيين بتحمّل جزء كبير من عبء دعم قنوات إجراء إلى إجراء [عملية إلى عملية] process-to-process channels غنية دلاليًا semantically rich. البديل هو إضافة وظائف إضافية إلى المُبدِّلات، مما يسمح للمضيفين النهائيين بأن يكونوا أجهزة غبية [بسيطة] (مثل أجهزة الهاتف). سنرى أن هذا السؤال المتعلق بكيفية تقسيم خدمات الشبكة المختلفة بين مُبدِّلات الحزم والمضيفين النهائيين (الأجهزة) يُمثل مشكلة متكررة recurring في تصميم الشبكة.

إيصال الرسائل Message Delivery بشكل موثوق

كما هو موضح في الأمثلة التي تناولناها، يُعدّ توصيل الرسائل بشكل موثوق إحدى أهم الوظائف التي يمكن أن توفرها الشبكة بشكل موثوق. ومع ذلك، من الصعب تحديد كيفية توفير هذه الموثوقية قبل فهم كيفية تعطل الشبكات. أول ما يجب إدراكه هو أن شبكات الحاسوب لا توجد في عالم مثالي. تتعطل الأجهزة ثم يُعاد تشغيلها reboot، وتنقطع الألياف [الضوئية]، وتفسد corrupt الواجهات الكهربائية electrical interfaces بتات البيانات المُرسلة، وتنفد مساحة التخزين المؤقت للمبدلات، وكما لو أن هذه الأنواع من المشاكل المادية ليست كافية للقلق، فقد يحتوي البرنامج الذي يُدير الأجهزة على أخطاء، وأحيانًا يُعيد توجيه الحزم إلى حالة من النسيان [العدم]. وبالتالي، فإن أحد المتطلبات الرئيسية للشبكة هو التعافي من أنواع معينة من الأعطال، حتى لا تضطر برامج التطبيقات إلى التعامل معها أو حتى أن تكون على دراية بهم.

هناك ثلاثة فصائل classes عامة من الأعطال التي يجب على مصممي الشبكات القلق بشأنها. أولاً، عند نقل حزمة عبر رابط مادي، قد تظهر أخطاء في بتات البيانات؛ أي أن القيمة 1 تتحول إلى 0 أو العكس. في بعض الأحيان، تتلف بتات منفردة single، ولكن في أغلب الأحيان يحدث رشقة أخطاء burst error - أي تلف عدة بتات متتالية. تحدث أخطاء البت عادةً بسبب تداخل القوى الخارجية مع نقل البيانات، مثل الصواعق، والاندفاعات surges المفاجئة في التيار الكهربائي، وأفران الميكروويف.

الخبر السار هو أن أخطاء البت هذه نادرة نسبيًا، إذ تؤثر في المتوسط على بت واحد فقط من كل 10^6 بت إلى 10^7 على كابل نحاسي copper-based نموذجي، وعلى بت واحد من كل 10^{12} بت إلى 10^{14} على الألياف الضوئية optical fibers النموذجية. وكما سنرى، هناك تقنيات تكشف أخطاء البت هذه باحتمالية عالية. بمجرد اكتشافها، من الممكن أحيانًا تصحيحها - إذا عرفنا البت أو البتات التالفة، يمكننا ببساطة قلبها - بينما في حالات أخرى، يكون الضرر بالغًا لدرجة أنه يستدعي التخلص من الحزمة بأكملها. في هذه الحالة، يُتوقع من المُرسِل إعادة إرسال الحزمة.

النوع الثاني من الأعطال يكون على مستوى الحزمة، لا على مستوى البت؛ أي أن الشبكة تفقد حزمة كاملة. أحد أسباب ذلك هو احتواء الحزمة على خطأ بت غير قابل للتصحيح، مما يستدعي التخلص منها. أما السبب الأكثر ترجيحًا فهو أن إحدى العقد التي يتعين عليها التعامل مع الحزمة - على سبيل المثال، مُبدِّل يُعيد توجيهها من رابط إلى آخر - مُثقلة جدًا بحيث لا تجد مكانًا لتخزينها، مما يُجبرها على حذفها. هذه هي مشكلة الازدحام التي ناقشناها سابقا. في حالات أقل شيوعًا، يرتكب البرنامج الذي يعمل على إحدى العقد التي تُعالج الحزمة خطأً. على سبيل المثال، قد يُعيد توجيه حزمة بشكل غير صحيح على الرابط الخطأ، مما يؤدي إلى عدم وصول الحزمة إلى وجهتها النهائية. كما سنرى، تتمثل إحدى الصعوبات الرئيسية في التعامل مع الحزم المفقودة في التمييز بين الحزمة المفقودة بالفعل والحزمة التي فقط تأخرت في الوصول إلى وجهتها.

أما النوع الثالث من الأعطال، فيحدث على مستوى العقدة والرابط؛ أي انقطاع الرابط المادي، أو تعطل الحاسوب المتصل به. قد يكون سبب ذلك تعطل برنامج software crash، أو انقطاع التيار الكهربائي، أو إهمال مشغل في شبكة backbone network مزود الانترنت [Internet Service Provider (ISP)]. كما أن الأعطال الناتجة عن سوء تهيئة misconfigurations أجهزة الشبكة شائعة أيضًا. ورغم إمكانية تصحيح أي من هذه الأعطال في نهاية المطاف، إلا أنها قد تؤثر بشكل درامي [عظيم] على الشبكة لفترة طويلة. ومع ذلك، ليس من الضرورة أن يؤدو إلى تعطيل كامل للشبكة. ففي شبكة تبديل الحزم، على سبيل المثال، من الممكن أحيانًا تسييرالحزم بعيدا عن route around عقدة أو رابط معطل. ومن صعوبات التعامل مع هذا النوع الثالث من الأعطال التمييز بين حاسوب معطل وآخر بطيء، أو في حالة الرابط [المعطل] [التمييز] بين حاسوب معطل وآخر ضعيف جدًا، مما يؤدي إلى عدد كبير من أخطاء البت.

النقطة الرئيسية

الفكرة الرئيسية المستخلصة من هذه المناقشة هي أن تحديد القنوات المفيدة يتطلب فهم متطلبات التطبيقات وإدراك حدود التقنية الأساسية [التحتية]. يكمن التحدي في سد الفجوة بين ما يتوقعه التطبيق وما يمكن أن توفره التقنية الأساسية. يُطلق على هذا أحيانًا اسم الفجوة الدلالية semantic gap.

القدرة على الإدارة

متطلب أخير، يبدو أنه يُهمَل أو يُترك للنهاية في كثير من الأحيان (كما هو الحال هنا)، هو ضرورة إدارة الشبكات network management. تشمل إدارة الشبكة ترقية المعدات upgrading equipment مع نمو الشبكة لاستيعاب المزيد من حركة المرور traffic أو الوصول إلى عدد أكبر من المستخدمين users، واستكشاف أخطاء الشبكة وإصلاحها network troubleshooting عند حدوث أي خلل أو انخفاض الأداء عن المستوى المطلوب، وإضافة ميزات features جديدة لدعم التطبيقات الجديدة.

يرتبط هذا المطلب جزئيًا بمسألة قابلية التوسع scalability التي نوقشت سابقا - فمع توسع الإنترنت لدعم مليارات المستخدمين ومئات الملايين من المضيفين على الأقل، أصبحت تحديات الحفاظ على التشغيل الصحيح للنظام وتشكيل configuring الأجهزة الجديدة عند إضافتها أكثر صعوبة. غالبًا ما يكون تشكيل مسير [جهاز توجيه] router واحد في شبكة مهمةً عمل لخبير مُدرَّب؛ أما تشكيل آلاف أجهزة التوجيه ومعرفة سبب عدم أداء شبكة بهذا الحجم كما هو متوقع، يمكن أن يصبح مهمةً تتجاوز قدرات أي شخص. علاوةً على ذلك، لجعل تشغيل الشبكة قابلًا للتوسع وفعّالًا إقتصاديا، يحتاج مشغلو الشبكات عادةً إلى الأتمتة automation لعديد من مهام الإدارة [بشكل كامل] أو على الأقل [لدرجة قدرة] تنفيذها بواسطة موظفين غير مؤهلين نسبيًا.

إحدى طرق تسهيل إدارة الشبكة هي تجنب التغيير. بمجرد أن تعمل الشبكة، ببساطة لا تلمسها! تكشف هذه العقلية عن التوتر الأساسي بين الاستقرار وسرعة [إدخال] الميزات: أي معدل إدخال قدرات جديدة إلى الشبكة. يُعدّ تفضيل الاستقرار النهج الذي اتبعته صناعة الاتصالات telecommunications industry (ناهيك عن مسؤولي أنظمة تكنولوجيا المعلومات IT system administrators في الجامعات والشركات) لسنوات عديدة، مما يجعلها واحدة من أكثر الصناعات بطئًا في التطور وتجنبًا للمخاطر risk averse. لكن الانتشار الواسع للسحابة الإلكترونية مؤخرًا غيّر هذه الديناميكية dynamics، مما جعل من الضروري تحقيق توازن أكبر بين الاستقرار وسرعة [إدخال] الميزات. يُعدّ تأثير السحابة الإلكترونية على الشبكة موضوعًا يتكرر مرارًا وتكرارًا في جميع أنحاء الكتاب، وهو موضوع نوليه اهتمامًا خاصًا في قسم "وجهات النظر" في نهاية كل فصل. في الوقت الحالي، يكفي القول إن إدارة شبكة سريعة التطور تُمثّل بلا شك التحدي الرئيسي في مجال الشبكات اليوم.

- سيقوم مبرمج التطبيقات application programmer بإدراج الخدمات التي يحتاجها تطبيقه: على سبيل المثال، ضمان تسليم كل رسالة يرسلها التطبيق دون أخطاء في غضون فترة زمنية معينة أو القدرة على التبديل بسلاسة بين الاتصالات المختلفة بالشبكة أثناء تنقل المستخدم user.

- يقوم مشغل الشبكة بإدراج خصائص النظام الذي يسهل إدارته والتحكم فيه: على سبيل المثال، ماهي الأخطاء االتي يمكن عزلها بسهولة، ماهي الأجهزة الجديدة التي يمكن إضافتها إلى الشبكة وتشكيلها بشكل صحيح، كيف يمكن حساب الاستخدام [درجة استخدام الشبكة] بسهولة.
- سيقوم مصمم الشبكة بإدراج خصائص التصميم الفعّال من حيث االفعالية الإقتصادية: على سبيل المثال، درجة استخدام موارد الشبكة بكفاءة وتوزيعها بشكل عادل على مختلف المستخدمين. ومن المرجح أيضًا أن تكون مسائل الأداء مهمة.

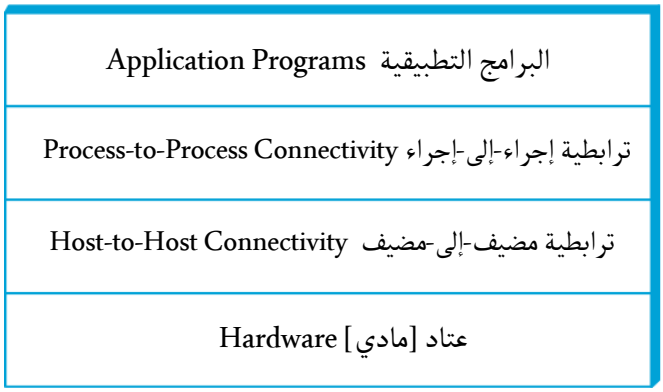
الأسس

في حال لم تلاحظ، فقد وضع القسم السابق مجموعةً واسعةً من المتطلبات لتصميم الشبكات - يجب أن توفر شبكة الحاسوب اتصالاً عامًا، واقتصاديًا، وعادلًا، ومتينا بين عدد كبير من الحواسيب. كما لو أن هذا لم يكن كافيًا، فالشبكات لا تبقى ثابتة في نقطة زمنية واحدة، بل يجب أن تتطور لاستيعاب التغيرات في كلٍّ من التقنيات الأساسية التي تعتمد عليها، وكذلك في متطلبات برامج التطبيقات. علاوةً على ذلك، يجب أن تكون الشبكات سهلة الإدارة من قِبل بشرٍ على اختلاف مستويات مهارة كلٍّ منهم. إن تصميم شبكة لتلبية هذه المتطلبات ليس مهمة صغيرة.

للمساعدة في التعامل مع هذا التعقيد، وضع مصممو الشبكات مخططات عامة - تُسمى عادةً بنيان الشبكات network architectures - تُرشد تصميم الشبكات وتنفيذها. يُعرّف هذا القسم بدقة أكبر ما نعنيه بهيكلية الشبكة من خلال طرح الأفكار المركزية المشتركة بين جميع هياكل الشبكات. كما يُقدم اثنتين من أكثر البنى المرجعية شيوعًا - هيكلية OSI (أو سبع طبقات) وهيكلية الإنترنت.

[التنظيم] بالطبقات والبروتوكولات Layering and Protocols

التجريد abstraction -[هو] إخفاء التفاصيل خلف واجهة interface مُحددة جيدًا - وهو الأداة الأساسية التي يستخدمها مصممو الأنظمة لإدارة التعقيد. تكمن فكرة التجريد في تحديد نموذج يُمكّن من التقاط بعض الجوانب المهمة للنظام، وتغليف encapsulate هذا النموذج في غرض object يُوفر واجهة يُمكن لمكونات النظام الأخرى إحداث تغيرات [تحت السيطرة] من خلالها مع إخفاء تفاصيل كيفية تنفيذ الغرض عن مستخدميه. يكمن التحدي في تحديد التجريدات التي تُوفر خدمةً تُثبت فائدتها في عدد كبير من الحالات في ذات الوقت مع إمكانية تنفيذها بكفاءة في النظام الأساسي. هذا بالضبط ما كنا نفعله عندما طرحنا فكرة القناة في القسم السابق: كنا نُقدم تجريدًا للتطبيقات application abstraction يُخفي تعقيد الشبكة عن مُطوري التطبيقات.



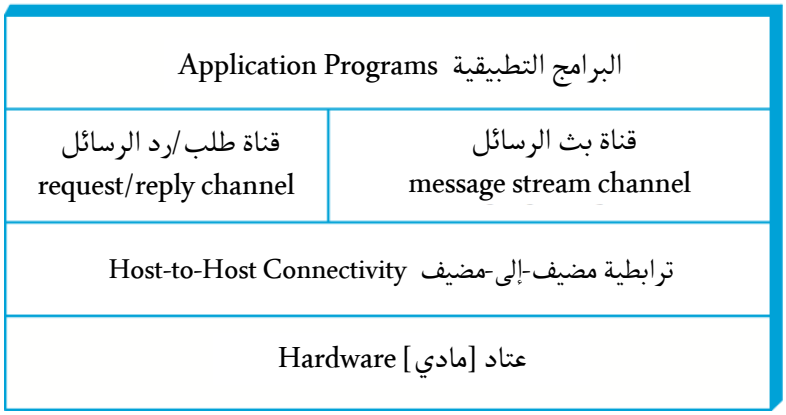
الشكل 1: مثال على نظام الشبكة الطبقي Layered Network System

تؤدي التجريدات بطبيعة الحال إلى الطبقات، خاصةً في أنظمة الشبكات. الفكرة العامة هي البدء بالخدمات التي تقدمها أجهزة العتاد [الأجهزة الفيزيائية الأساسية] ثم إضافة سلسلة من الطبقات، كل منها يوفر مستوى أعلى (أكثر تجريدًا) من الخدمات. تُنفذ الخدمات االمتوافرة في الطبقات العليا بإستعمال الخدمات التي تقدمها الطبقات الدنيا.على سبيل المثال، بالاعتماد على مناقشة متطلبات [الشبكة] الواردة في القسم السابق، يمكننا تخيل شبكة بسيطة تحتوي على طبقتي تجريد محصورتين بين برنامج التطبيق والأجهزة الأساسية، كما هو موضح في الشكل 1. قد توفر الطبقة التي تعلو الأجهزة مباشرة في هذه الحالة ترابطا بين مضيفين host-to-host connectivity، مما يستبعد وجود بنية شبكة معقدة بشكل اِعتِباطِيّ بين أي مضيفين. تعتمد الطبقة التالية على خدمة الترابط بين مضيفين وتوفر دعمًا لقنوات الإتصال بين عمليتين إجرائتين process-to-process channels، وبذلك تخفي [تعقيدات التعامل مع بعض الأحداث] مثل فقدان الشبكة للرسائل أحيانًا، على سبيل المثال.

يوفر التقسيم الطبقي ميزتين رائعتين. أولاً، يُحلل decomposes مشكلة بناء الشبكة إلى مكونات أكثر سهولة في الإدارة. فبدلاً من تطبيق برنامج مترابط [واحد] monolithic يُؤدي كل ما قد ترغب به، يُمكنك استخدام عدة طبقات [متعاونة]، كل منها يُحل جزءًا من المشكلة. ثانيًا، ذلك يُوفر تصميمًا اجتزائي [أكثر مرونة] modular. فإذا قررت إضافة خدمة جديدة، فقد تحتاج فقط إلى تعديل وظائف طبقة واحدة، وإعادة استخدام الوظائف functions المُقدمة في جميع الطبقات الأخرى.

مع ذلك، يُعدّ اعتبار النظام سلسلة خطية linear من الطبقات تبسيطًا مفرطًا. ففي كثير من الأحيان، هناك تجريدات متعددة في أي مستوى من مستويات النظام، كل منها يُقدّم خدمة service مختلفة للطبقات العليا، ولكنه يبني على ذات التجريدات منخفضة المستوى low-level abstractions [في الطبقات السفلية].

ولتوضيح ذلك، لننظر إلى نوعي القنوات اللذين ناقشناهما في القسم السابق. إحداهما تُقدّم خدمة طلب/رد request/reply، والأخرى تدعم خدمة تدفق الرسائل message stream service. قد تُمثّل هاتان القناتان عروضًا بديلة في مستوى ما من نظام الشبكات متعدد المستويات multilevel networking system، كما هو موضح في الشكل 2.

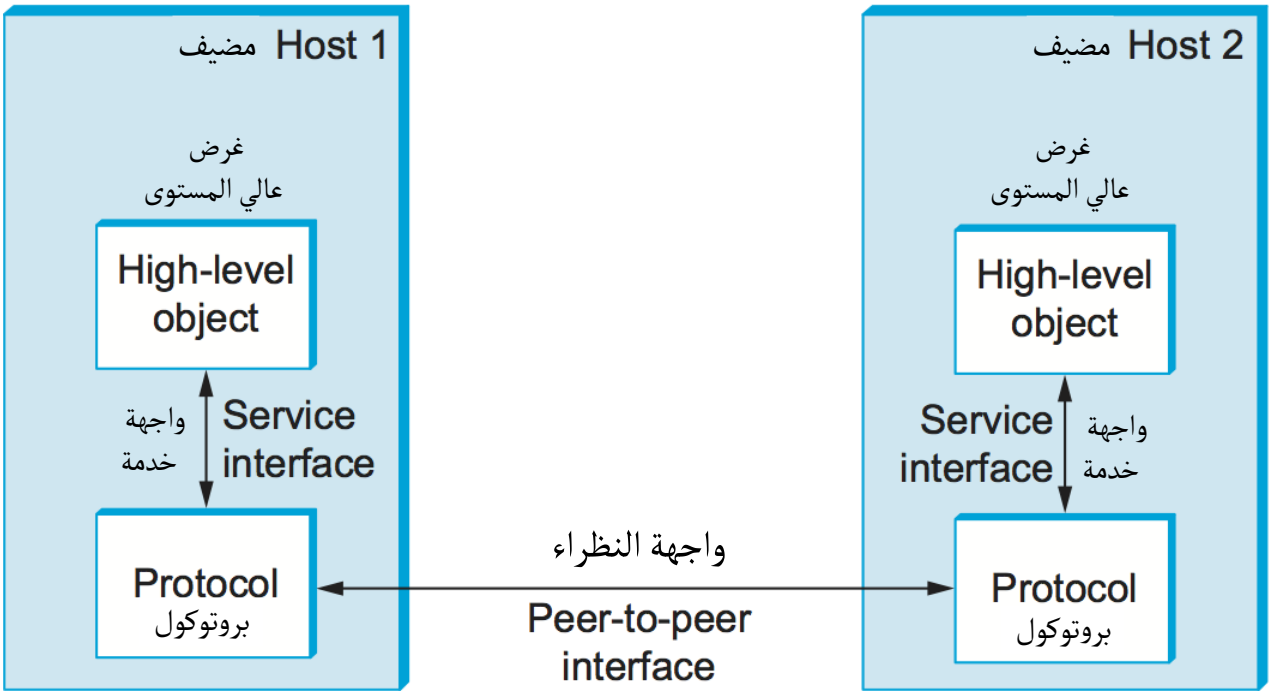


الشكل 2: نظام متعدد الطبقات مع تواجد تجريدات مختلفة في طبقة واحدة

بناءً على هذه المناقشة حول الطبقات، أصبحنا الآن جاهزين لمناقشة بنية الشبكة بشكل أكثر دقة. بدايةً، تُسمى الأغراض المجردة abstract objects التي تُشكل طبقات نظام الشبكة بالبروتوكولات protocols. أي أن البروتوكول يوفر خدمة اتصال تستخدمها الأغراض عالية المستوى (مثل إجرائيات [عمليات] التطبيقات application processes، أو ربما البروتوكولات من مستويات أعلى higher-level protocols) لتبادل الرسائل. على سبيل المثال، يمكننا تخيل شبكة تدعم بروتوكول طلب/رد request/reply وبروتوكول تدفق الرسائل message stream protocol، بما يتوافق مع قنوات الطلب/الرد وتدفق الرسائل المذكورة أعلاه.

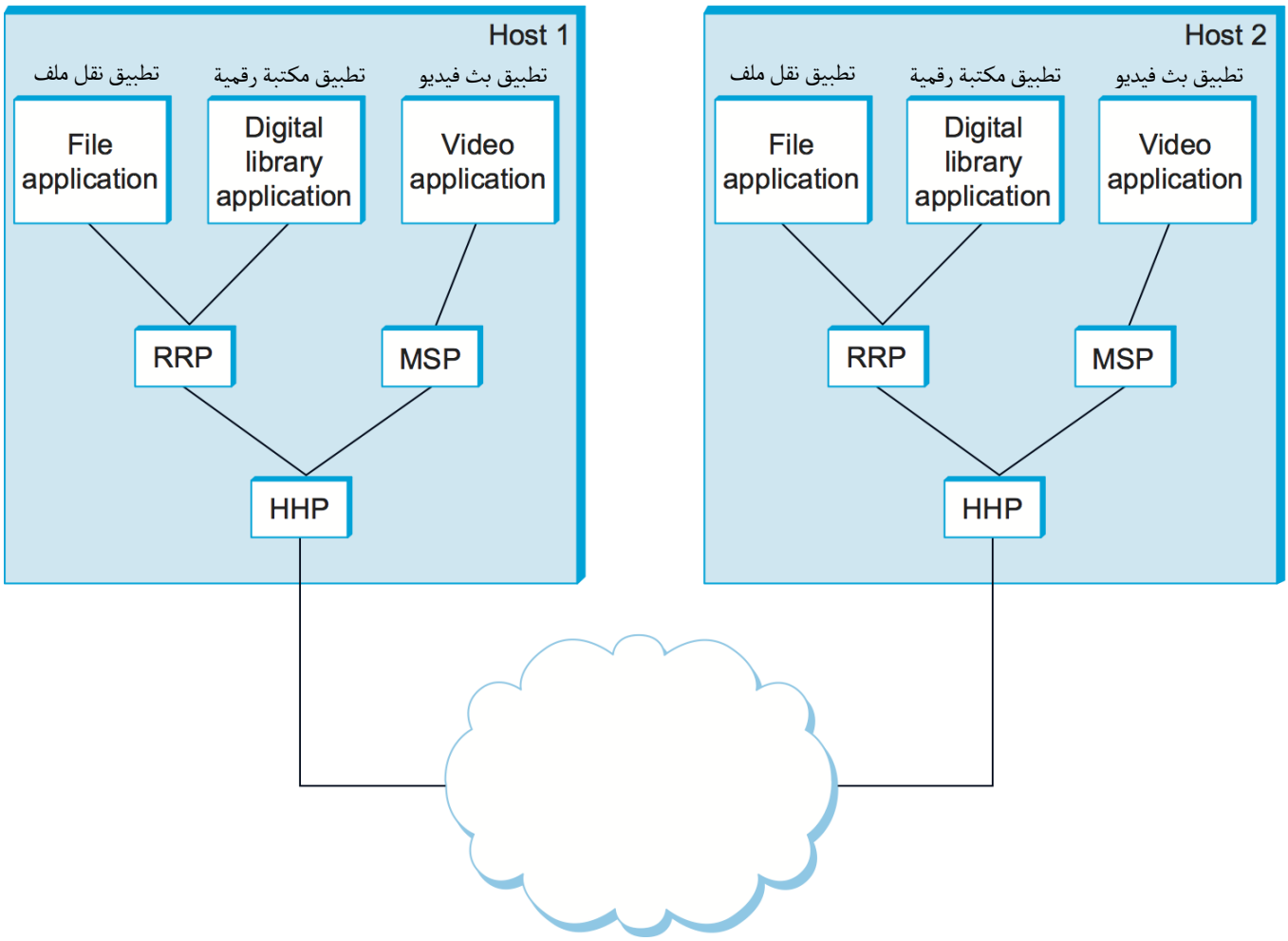
يُعرّف كل بروتوكول واجهتين مختلفتين. أولًا، تعرّف واجهة الخدمة service interface للأغراض الأخرى في ذات الحاسوب التي ترغب في استخدام خدمات الاتصال communication services الخاصة به. تُعرّف واجهة الخدمة هذه العمليات التي يُمكن للأغراض المحلية إجراؤها على البروتوكول. على سبيل المثال، يدعم بروتوكول الطلب/الرد العمليات التي يُمكن من خلالها للتطبيق إرسال واستقبال الرسائل. يُمكن لتطبيق بروتوكول HTTP دعم عملية جلب صفحة نص وب web page من خادم بعيد. يستدعي تطبيق، مثل متصفح الويب web browser، هذه العملية كلما احتاج المتصفح إلى الحصول على صفحة جديدة (مثلًا، عندما ينقر المستخدم على رابط weblink في الصفحة المعروضة حاليًا).

باختصار، يُعرّف البروتوكول خدمة اتصال communication service يُصدّرها محليًا (واجهة الخدمة the service interface)، بالإضافة إلى مجموعة من القواعد التي تُنظّم الرسائل التي يتبادلها البروتوكول مع نظرائه peer(s) لتنفيذ هذه الخدمة (واجهة النظير peer interface).هذا الوضع موضح في الشكل 3.



الشكل 3: واجهات الخدمة service interfaces وواجهات النظراء peer interfaces.

باستثناء مستوى الأجهزة، حيث يتواصل النظراء مباشرةً عبر وسيط مادي، يكون اتصال النظراء غير مباشر indirect، إذ يتواصل كل بروتوكول مع نظيره عبر تمرير رسائل إلى بروتوكول أدنى مستوى، والذي بدوره يُسلّم الرسالة إلى نظيره. إضافةً إلى ذلك، يُحتمل وجود أكثر من بروتوكول واحد في أي مستوى مُحدد، يُقدّم كلٌّ منها خدمة اتصال مُختلفة. لذلك، نُمثّل مجموعة البروتوكولات التي تُشكّل نظام شبكة بشكل بيان للبروتوكولات. تُمثّل عُقد البيان البروتوكولات، بينما تُمثّل الحواف edges علاقة [بين البروتوكولين على طرفيها]. على سبيل المثال، يُوضّح الشكل 4 رسمًا لبروتوكول النظام الطبقي الافتراضي الذي ناقشناه، حيث يُطبّق بروتوكولا RRP (بروتوكول الطلب/الرد Request/Reply Protocol) وMSP (بروتوكول تدفق الرسائل Message Streaming Protocol) نوعين مُختلفين من قنوات الاتصال بين العمليات، ويعتمد كلاهما على بروتوكول المضيف-إلى-المضيف Host-to-Host Protocol (HHP) الذي يُوفّر خدمة اتصال بين المضيفين.



الشكل 4: مثال على مخطط بروتوكول.

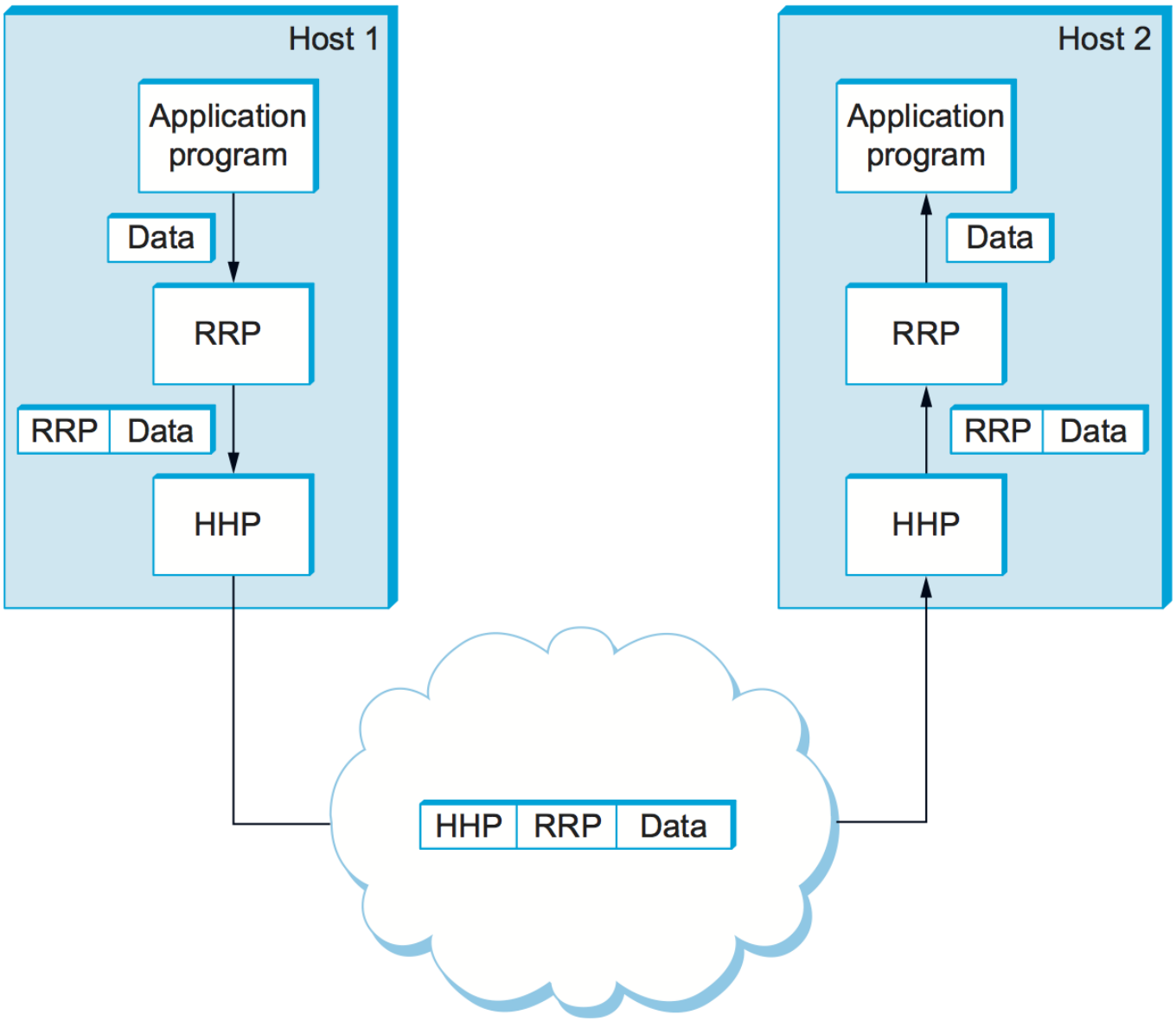
في هذا المثال، لنفترض أن برنامج الوصول إلى الملفات على المضيف 1 يريد إرسال رسالة إلى نظيره على المضيف 2 باستخدام خدمة الاتصال التي يوفرها RRP. في هذه الحالة، يطلب تطبيق الملف من RRP إرسال الرسالة نيابةً عنه. للتواصل مع نظيره، يستدعي RRP خدمات HHP، الذي بدوره ينقل الرسالة إلى نظيره على الجهاز الآخر. بمجرد وصول الرسالة إلى مثيل HHP على المضيف 2، يُمرر HHP الرسالة إلى RRP، الذي بدوره يُسلمها إلى تطبيق الملف. في هذه الحالة، يُقال إن التطبيق يستخدم خدمات كدسة بروتوكولات RRP/HHP protocol stack.

تجدر الإشارة إلى أن مصطلح البروتوكول يُستخدم بطريقتين مختلفتين. ففي بعض الأحيان، يشير إلى الواجهات المجردة - أي العمليات التي تُعرّفها واجهة الخدمة service interface وشكل ومعنى الرسائل المتبادلة بين النظراء، وفي أحيان أخرى إلى الوحدة module التي تُطبّق هاتين الواجهتين فعليًا. وللتمييز بين الواجهات والوحدة التي تُطبّقها، نشير عادةً إلى الأولى باسم مواصفات البروتوكول. وتُعبّر المواصفات عادةً عن نفسها باستخدام مزيج من النصوص [اللغة العادية] prose، والرموز الشبيهة [الإفتراضية] pseudocode، ومخططات انتقال الحالة state transition diagrams، وصور مصاغات الحزم pictures of packet formats، وغيرها من الرموز المجردة. ويُفترض أن إنجاز بروتوكول مُعيّن بطرق مختلفة من قِبَل مُبرمجين مُختلفين ممكن طالما التزم كلٌّ منهم بالمواصفات. ويتمثل التحدي في ضمان نجاح تطبيقين مختلفين لنفس المواصفات في تبادل الرسائل. ويُقال إن وحدتي بروتوكول أو أكثر تُطبّقان بدقة مواصفات بروتوكول ما، وبأنهم قابلين للعمل مع بعضهم البعض.

يمكننا تخيّل العديد من البروتوكولات والبيانات المختلفة للبروتوكولات التي تُلبّي متطلبات الاتصال لمجموعة من التطبيقات. ولحسن الحظ، توجد هيئات توحيد معايير standardization bodies، مثل فريق عمل هندسة الإنترنت Internet Engineering Task Force (IETF) والمنظمة الدولية للمعايير International Standards Organization (ISO)، التي تضع سياساتٍ لكل رسم بياني لبروتوكول مُحدّد. نُسمّي مجموعة القواعد rules التي تُنظّم شكل ومحتوى رسم البروتوكول "بنية الشبكة". على الرغم من أن هذا يتجاوز نطاق هذا الكتاب، فقد وضعت هيئات توحيد المعايير إجراءاتٍ مُحدّدة بدقة لإدخال البروتوكولات والتحقق من صحتها والموافقة عليها نهائيًا في بنياتها الخاصة. سنصف بإيجاز البنيات التي حدّدتها IETF وISO، ولكن أولاً، هناك أمران إضافيان نحتاج إلى شرحهما حول آليات طبقات البروتوكول.

التغليف Encapsulation

تخيل ما يحدث عندما يرسل أحد برامج التطبيق رسالة إلى نظيره عبر تمريرها إلى RRP. من وجهة نظر RRP، فإن الرسالة التي يتلقاها التطبيق هي سلسلة بايتات [بدون إنقطاع]. لا يُبالي RRP بأن تُمثل هذه البايتات مصفوفة من الأعداد الصحيحة، أو رسالة بريد إلكتروني، أو صورة رقمية، أو أي شيء آخر؛ فهو مُكلّف ببساطة بإرسالها إلى نظيره. ومع ذلك، يجب على RRP نقل معلومات التحكم إلى نظيره، مُعلّمًا إياه كيفية التعامل مع الرسالة عند استلامها. يقوم RRP بذلك عن طريق إرفاق ترويسة header بالرسالة. وبشكل عام، الترويسة هو بنية بيانات صغيرة - من بضعة بايتات إلى بضع عشرات من البايتات - تُستخدم بين النظراء للتواصل فيما بينهم. كما يوحي الاسم، عادةً ما تُرفق الترويسات بمقدمة الرسالة. ومع ذلك، في بعض الحالات، تُرسل معلومات التحكم هذه من نظير إلى نظير في نهاية الرسالة، وفي هذه الحالة تُسمى تذيلة [ملحقً] trailer. يُحدَّد الشكل الدقيق format للترويسة المُرفقة بواسطة RRP وفقًا لمواصفات بروتوكوله. ويُسمى باقي الرسالة - أي البيانات المُرسَلة نيابةً عن التطبيق - نص الرسالة أو حمولتها payload. ونُشير إلى أن بيانات التطبيق مُغلَّفة encapsulated في الرسالة الجديدة التي يُنشئها RRP.



الشكل 5: تغليف [تضمين] encapsulation الرسائل عالية المستوى high-level داخل الرسائل منخفضة المستوى low-level.

تتكرر عملية التغليف عند كل مستوى من مستويات مخطط البروتوكول؛ على سبيل المثال، يُغلّف HHP رسالة RRP بإرفاق ترويسة خاصة به. إذا افترضنا الآن أن HHP يُرسل الرسالة إلى نظيره عبر شبكة ما، فعند وصول الرسالة إلى المضيف الوجهة، تُعالَج بالترتيب المعاكس: يُفسّر HHP أولاً ترويسة HHP في مقدمة الرسالة (أي يتخذ الإجراء المناسب بناءً على محتوى االترويسة) ويُمرّر نص الرسالة (وليس ترويسة HHP) إلى RRP، الذي يتخذ الإجراء الذي تشير إليه ترويسة RRP الذي أرفقها نظيره، ويُمرّر نص الرسالة (وليس ترويسة RRP) إلى برنامج التطبيق.

SS

الرسالة المرسلة من RRP إلى التطبيق على المضيف 2 هي نفسها الرسالة المرسلة إلى RRP على المضيف 1؛ ولا يرى التطبيق أيًا من االترويسات المرفقة به لتنفيذ خدمات الاتصالات منخفضة المستوى. يوضح الشكل 5 هذه العملية برمتها. لاحظ أنه في هذا المثال، قد تفحص عُقد الشبكة (مثل المبدلات والموجهات) ترويسة HHP في مقدمة الرسالة.

تجدر الإشارة إلى أنه عندما نقول إن بروتوكولًا منخفض المستوى لا يُفسّر الرسالة المُقدّمة إليه من بروتوكول عالي المستوى، فإننا نعني أنه لا يعرف كيفية استخلاص extract أي معنى من البيانات المُتضمّنة في الرسالة. ومع ذلك، في بعض الأحيان، يُجري البروتوكول منخفض المستوى تحويلًا transformation بسيطًا على البيانات المُقدّمة إليه، مثل ضغطها compress أو تشفيرها encrypt. في هذه الحالة، يُحوّل البروتوكول نص الرسالة message body بالكامل، بما في ذلك بيانات التطبيق الأصلي وجميع الترويسات المُرفقة بها بواسطة بروتوكولات المستوى الأعلى higher-level protocols.

التضمين Multiplexing وفك التضمين Demultiplexing

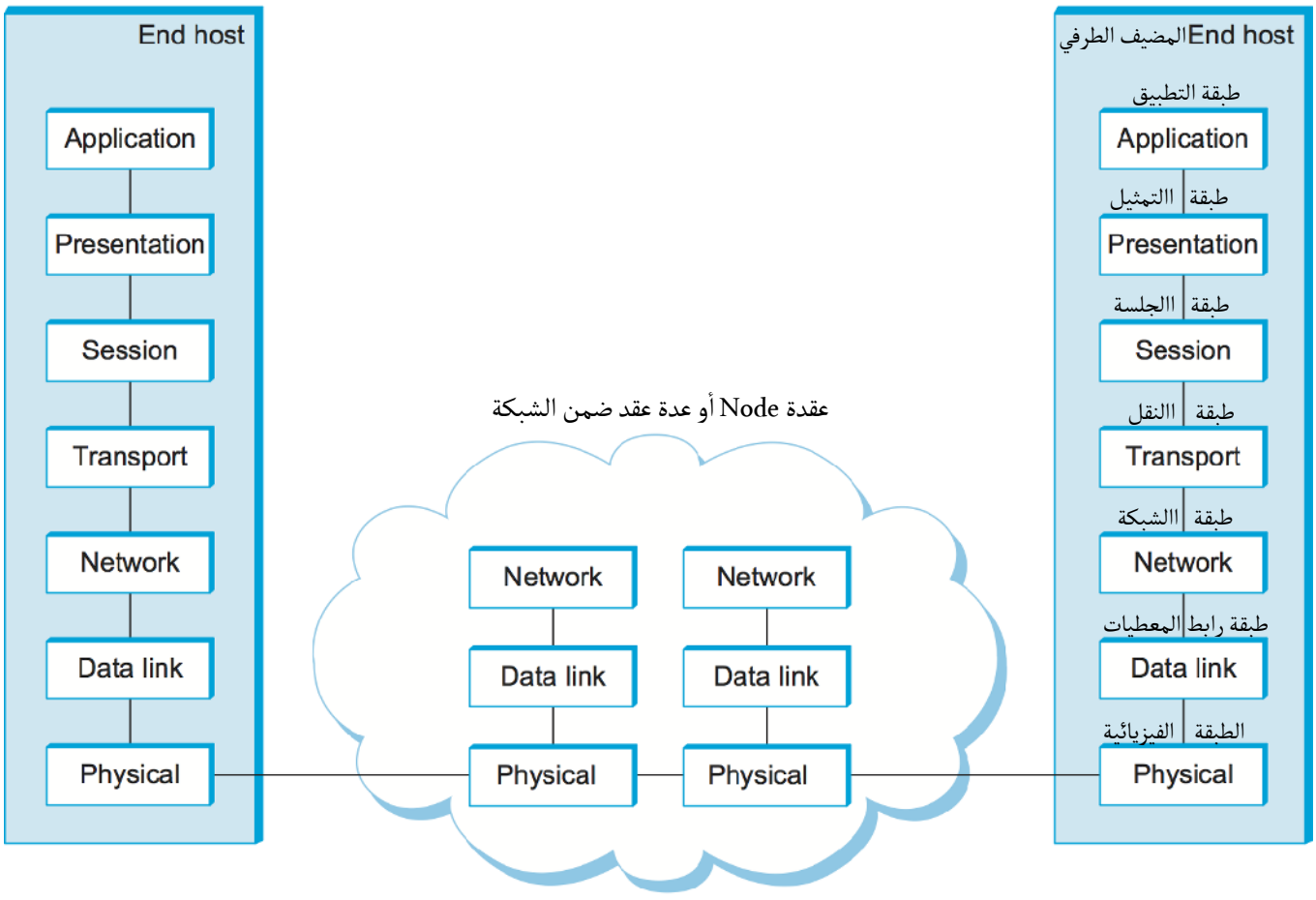
تذكر أن الفكرة الأساسية لتبديل الحزم هي تضمين تدفقات متعددة من البيانات عبر رابط مادي واحد. تنطبق هذه الفكرة نفسها على جميع أجزاء مخطط البروتوكول، وليس فقط على عُقد التبديل. في الشكل 4، على سبيل المثال، يمكننا اعتبار بروتوكول RRP بمثابة قناة اتصال منطقية logical communication channel، حيث تضمن رسائل من تطبيقين مختلفين عبر هذه القناة في المضيف المصدر، ثم يفك التضمين إلى التطبيق المناسب في المضيف الوجهة.

عمليًا، يعني هذا ببساطة أن الترويسة الذي يُرفقه بروتوكول RRP برسائله يحتوي على مُعرّف identifier يُسجل التطبيق الذي تنتمي إليه الرسالة. نُطلق على هذا المُعرّف اسم مفتاح فكّ التضمين demultiplexing key الخاص ببروتوكول RRP، أو demux key اختصارًا. في مُضيف المصدر، يُضمّن بروتوكول RRP مفتاح فكّ التضمين المُناسب في ترويسته. عند تسليم الرسالة إلى مُضيف الوجهة، يُزيل بروتوكول RRP ترويسته، ويفحص مفتاح فكّ التضمين، ثم يُفكّ التضمين إلى التطبيق المُناسب.

ليس بروتوكول RRP فريدًا في دعمه للتضمين المتعدد؛ فكل بروتوكول تقريبًا يُطبّق هذه الآلية. على سبيل المثال، يمتلك بروتوكول HHP مفتاح فك تتضمين خاص به لتحديد الرسائل التي يجب تمريرها إلى RRP وتلك التي يجب تمريرها إلى MSP. ولكن، لا يوجد اتفاق موحد بين البروتوكولات - حتى تلك الموجودة ضمن بنية شبكة واحدة - على ماهو بالضبط مفتاح فك التضمين. تستخدم بعض البروتوكولات حقلًا من 8 بت (أي أنها لا تدعم سوى 256 بروتوكولًا عالي المستوى)، بينما يستخدم البعض الآخر حقولًا من 16 أو 32 بت. كما تحتوي بعض البروتوكولات على حقل فك التضمين واحد في رتريويستها، بينما يحتوي البعض الآخر على زوج من حقول فك التضمين. في الحالة الأولى، يُستخدم مفتاح فك التضمين نفسه على كلا جانبي الاتصال، بينما في الحالة الثانية، يستخدم كل جانب مفتاحًا مختلفًا لتحديد البروتوكول عالي المستوى (أو برنامج التطبيق) الذي سيتم تسليم الرسالة إليه.

نموذج OSI ذو السبع طبقات

كانت منظمة ISO من أوائل المنظمات التي حددت رسميًا طريقةً مشتركةً لتوصيل أجهزة الكمبيوتر. تُعرّف بنيتها، المسماة بنية ربط الأنظمة المفتوحة Open Systems Interconnection (OSI) architecture، والموضحة في الشكل 6، بتقسيم وظائف الشبكة إلى سبع طبقات، حيث يُنفّذ بروتوكول واحد أو أكثر الوظائف المُخصصة لكل طبقة. وبهذا المعنى، فإن المخطط المُقدّم ليس رسمًا بيانيًا للبروتوكول بحد ذاته، بل هو نموذج مرجعي reference model له. ويُشار إليه غالبًا باسم نموذج الطبقات السبع 7-layer model.



الشكل 6: نموذج OSI ذو السبع طبقات 7-layer model

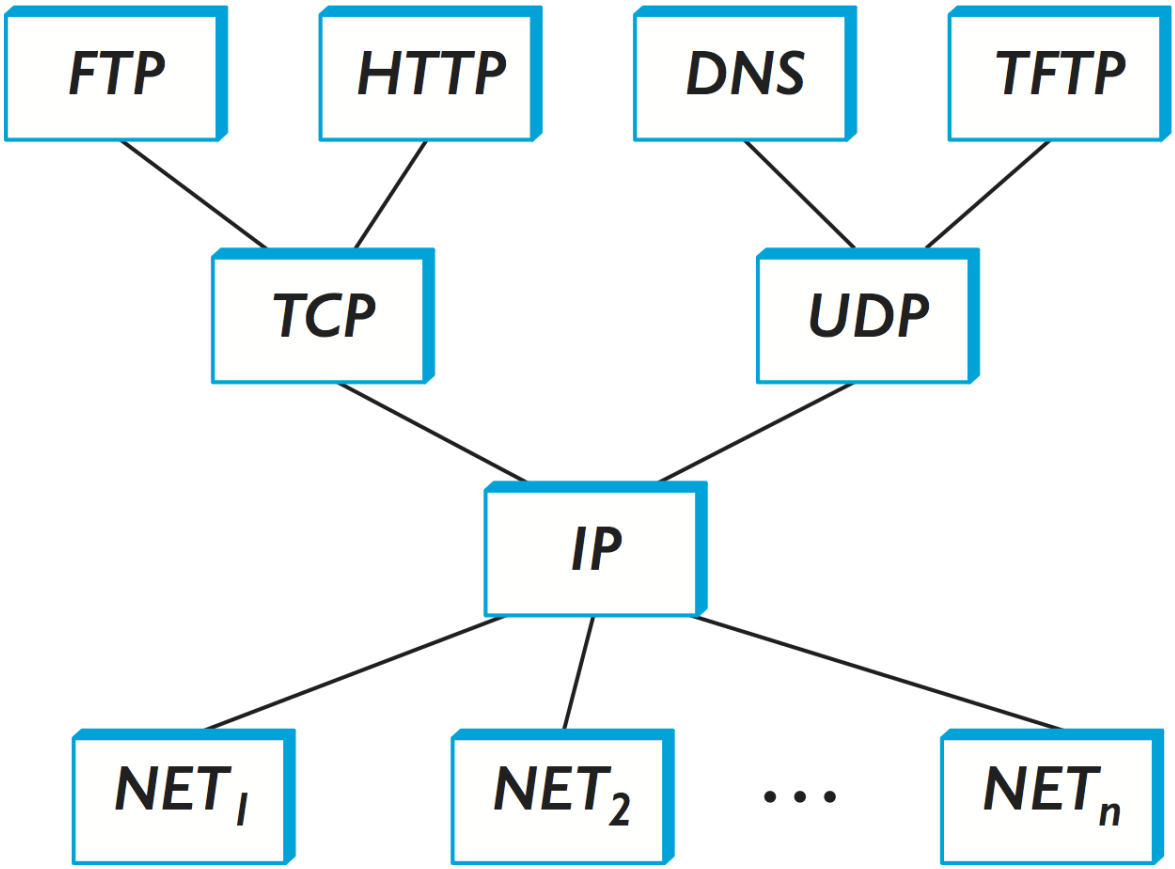
بدءًا من الأسفل وصولًا إلى الأعلى، تتولى الطبقة المادية physical layer نقل البتات الخام raw bits عبر رابط الاتصالات communications link. ثم تجمع طبقة رابط البيانات data link layer سيلًا stream من البتات في مجموعة أكبر aggregate تُسمى إطارًا frame. عادةً ما تُطبّق موائمات [محولات] الشبكة network adapters، إلى جانب [برامج] سواقات الأجهزة device drivers التي تعمل في نظام تشغيل العقدة node's operating system في مستوى رابط البيانات data link. هذا يعني أن الإطارات، وليس البتات الخام، هي التي تُسلّم فعليًا إلى الأجهزة المضيفة. تُعنى طبقة الشبكة بالتسيير routing بين العقد داخل شبكة تبديل الحزم. في هذه الطبقة، تُسمى وحدة البيانات المتبادلة بين العقد عادةً حزمةً packet بدلًا من إطار frame، مع أنهما في الأساس نفس الشيء. تُطبّق الطبقات الثلاث السفلى في جميع عقد الشبكة، بما في ذلك المبدلات switches داخل الشبكة وفي الأجهزة المضيفة المتصلة بخارج الشبكة. ثم تُطبّق طبقة النقل transport layer ما كنا نُطلق عليه حتى الآن قناةً الآن إجراء-إلى-إجراء process-to-process channel. عند هذه المرحلة، تُسمى وحدة البيانات المتبادلة رسالةً بدلًا حزمة أو إطار. عادةً ما تعمل طبقة النقل والطبقات الأعلى على الأجهزة المضيفة النهائية فقط، وليس في وسط الشبكة ضمن المبدلات أو أجهزة التيسر.

هناك اتفاق أقل حول تعريف الطبقات الثلاث العليا، ويعود ذلك جزئيًا إلى عدم تواجدها جميعًا دائمًا، كما سنرى لاحقًا. بالانتقال إلى الطبقة العليا (السابعة)، نجد طبقة التطبيق Application Layer. تتضمن بروتوكولات طبقة التطبيق بروتوكول نقل النص االترابطي Hypertext Transfer Protocol (HTTP)، وهو أساس شبكة الويب العالمية، وهو ما يُمكّن متصفحات الويب من طلب صفحات من خوادم الويب web servers. أسفلها، تُعنى طبقة العرض Presentation Layer بتنسيق البيانات المتبادلة بين الأقران peers - على سبيل المثال، ما إذا كان طول العدد الصحيح 16 أو 32 أو 64 بت، وما إذا كان البايت الأكثر أهمية يُرسل أولًا أم أخيرًا، أو كيفية تنسيق تدفق الفيديو video stream. وأخيرًا، تُوفر طبقة الجلسة Session Layer مساحة اسم name space تُستخدم لربط تدفقات النقل transport streams المختلفة التي تُشكل جزءًا من تطبيق واحد. على سبيل المثال، قد تُدير هذه الطبقة تدفقًا صوتيًا voice stream وتدفق فيديو video stream يتم دمجهما في تطبيق مؤتمرات عن بُعد teleconferencing application.

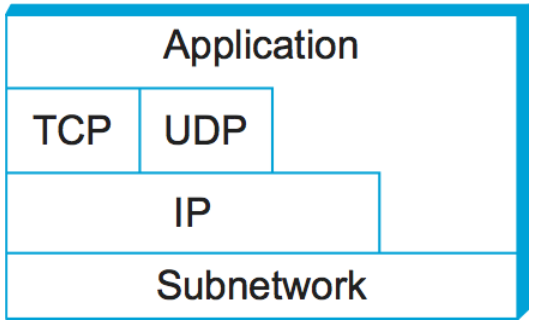
بنية الإنترنت الهندسية Internet Architecture

يظهر في الشكل 7 بنية الإنترنت، والتي تسمى أحيانًا بنية TCP/IP بعد بروتوكوليها الرئيسيين. ويرد تمثيل بديل في الشكل 8. تطورت بنية الإنترنت من الخبرات المكتسبة مع شبكة تبديل الحزم السابقة المسماة ARPANET.

تم تمويل كلٍّ من الإنترنت وشبكة ARPANET من قِبل وكالة مشاريع الأبحاث المتقدمة (ARPA)، إحدى وكالات تمويل البحث والتطوير التابعة لوزارة الدفاع الأمريكية. كان الإنترنت وشبكة ARPANET موجودين قبل ظهور بنية OSI، وكان للخبرة المكتسبة من بناءهما تأثير كبير على نموذج OSI المرجعي.



الشكل 7: رسم بنياني لبروتوكول الإنترنت



الشكل 8: نظرة بديلة على بنية الإنترنت.
كانت طبقة "الشبكة الفرعية" subnetwork تُسمى تاريخيًا "طبقة الشبكة" network layer، ويُشار إليها الآن غالبًا باسم "الطبقة الثانية" layer 2.

بينما يمكن تطبيق نموذج OSI ذي السبع طبقات، مع بعض الخيال، على الإنترنت، غالبًا ما يُستخدم نموذج رباعي الطبقات بدلاً منه. يوجد في أدنى مستوى مجموعة واسعة من بروتوكولات الشبكة، يُشار إليها باسم NET 1 وNET 2 وما إلى ذلك. عمليًا، تُنفذ هذه البروتوكولات من خلال مزيج من الأجهزة (مثل موائمات الشبكة network adaptors) والبرمجيات (مثل برنامج [برامج] سواقات الأجهزة الشبكية network device driver). على سبيل المثال، قد تجد بروتوكولات Ethernet أو بروتوكولات لاسلكية (مثل معايير Wi-Fi 802.11) في هذه الطبقة. (قد تتضمن هذه البروتوكولات بدورها عدة طبقات فرعية، ولكن بنية الإنترنت لا تفترض أي شيء عنها). تتكون الطبقة الثانية من بروتوكول واحد - بروتوكول الإنترنت Internet Protocol (IP). هذا هو البروتوكول الذي يدعم الترابط interconnection بين تقنيات الشبكات المتعددة في شبكة داخلية منطقية واحدة logical internetwork. تحتوي الطبقة الثالثة على بروتوكولين رئيسيين - بروتوكول التحكم في الإرسال Transmission Control Protocol (TCP) وبروتوكول برقيات معطيات المستخدم User Datagram Protocol (UDP).

يوفر بروتوكولا TCP وUDP قنوات منطقية logical channels بديلة لبرامج التطبيقات: يوفر TCP قناة موثوقة لتدفق البايتات reliable byte-stream، بينما يوفر UDP قناة غير موثوقة لتوصيل البرقيات [الرسائل القصيرة] unreliable datagram delivery channel. في لغة الإنترنت، يُطلق على TCP وUDP أحيانًا اسم بروتوكولات من طرف إلى طرف end-to-end protocols، مع أنه من الصحيح أيضًا تسميتهما ببروتوكولات النقل transport protocols.

تعمل فوق طبقة النقل مجموعة من بروتوكولات التطبيقات، مثل HTTP وFTP وTelnet (تسجيل الدخول عن بُعد remote login) وبروتوكول نقل البريد البسيط Simple Mail Transfer Protocol (SMTP)، والتي تُمكّن من التشغيل البيني interoperation بين التطبيقات الشائعة. لفهم الفرق بين بروتوكول طبقة التطبيقات والتطبيق، فكّر في جميع متصفحات الويب [web browsers] World Wide Web browsers المختلفة المتوفرة أو التي كانت متوفرة (مثل Firefox وChrome وSafari وNetscape وMosaic وInternet Explorer). هناك عدد كبير ومماثل من التطبيقات المختلفة لخوادم الويب web servers. والسبب في إمكانية استخدام أيٍّ من برامج التطبيقات هذه للوصول إلى موقع معين على الويب هو أنها جميعًا تتوافق مع بروتوكول طبقة التطبيقات نفسه: HTTP. ومن المُربك أن المصطلح نفسه ينطبق أحيانًا على كلٍّ من التطبيق وبروتوكول طبقة التطبيقات الذي يستخدمه (على سبيل المثال، غالبًا ما يُستخدم FTP كاسم للتطبيق الذي يُطبّق بروتوكول FTP).

معظم العاملين في مجال [حقل] الشبكات على دراية ببنية الإنترنت وبنية OSI ذات السبع طبقات، وهناك اتفاق عام على كيفية تقابل [ping]map الطبقات بين البنيات. تقع طبقة تطبيقات الإنترنت في الطبقة السابعة، وطبقة النقل في الطبقة الرابعة، وطبقة IP (تشبيك [الربط] الشبكي internetworking أو الشبكة network فقط) في الطبقة الثالثة، وطبقة الوصلة link أو الشبكة الفرعية subnet تقع في الطبقة الثانية أسفل طبقة IP.

تتميز بنية الإنترنت بثلاث خصائص جديرة بالاهتمام. أولاً، وكما هو موضح في الشكل 8، لا تتضمن بنية الإنترنت طبقاتٍ صارمة. فالتطبيق حر في تجاوز طبقات النقل المحددة واستخدام بروتوكول الإنترنت (IP) مباشرةً أو إحدى الشبكات الأساسية. في الواقع، يتمتع المبرمجون بحرية تعريف تجريدات قنوات channel abstractions جديدة أو تطبيقات تعمل فوق أيٍّ من البروتوكولات الحالية.

ثانيًا، إذا دققتَ النظر في رسم البروتوكول في الشكل 7، ستلاحظ شكل الساعة الرملية - عريض في الأعلى، ضيق في المنتصف، وعريض في الأسفل. يعكس هذا الشكل في الواقع الفلسفة الأساسية للبنية. أي أن بروتوكول الإنترنت Internet Protocol (IP) يُمثل النقطة المحورية focal point للبنية، إذ يُحدد طريقةً مشتركة لتبادل الحزم بين مجموعة واسعة من الشبكات. فوق بروتوكول الإنترنت، قد توجد بروتوكولات نقل متعددة، كل منها يُقدم تجريدًا مختلفًا للقنوات لبرامج التطبيقات. وبالتالي، فإن مسألة توصيل الرسائل من مضيف إلى مضيف منفصلة تمامًا عن مسألة توفير خدمة اتصال مفيدة بين العمليات. تحت بروتوكول الإنترنت، تسمح البنية بالعديد من تقنيات الشبكات المختلفة، بدءًا من الإيثرنت واللاسلكي وإلى الروابط الفردية من نقطة إلى نقطة.

من السمات الأخيرة لبنية الإنترنت (أو بالأحرى لثقافة IETF) أنه لكي يُدرج بروتوكول جديد رسميًا في البنية، يجب أن يكون هناك مواصفات بروتوكول ووجود تطبيق تمثيلي عامل واحد على الأقل (ويفضل تطبيقان) لها. ويشترط وجود تطبيقات عاملة لاعتماد المعايير من قِبل IETF. ويساعد هذا الافتراض الثقافي لمجتمع التصميم على ضمان إمكانية تنفيذ بروتوكولات البنية بكفاءة. ولعل أفضل مثال على أهمية البرمجيات العاملة في ثقافة الإنترنت هو الاقتباس الموجود على القمصان T-shirts التي تُرتدى عادةً في اجتماعات IETF والتي تحث على أهمية الوفاق ووجود برمجيات عاملة working code.

النقطة الرئيسية

من بين هذه السمات الثلاث لبنية الإنترنت، تُعد فلسفة تصميم الساعة الرملية بالغة الأهمية، مما يستدعي التكرار. يُمثل محيط الساعة الرملية الضيق مجموعةً محدودةً ومختارةً بعناية من القدرات العالمية، مما يسمح للتطبيقات عالية المستوى وتقنيات الاتصالات منخفضة المستوى بالتعايش، ومشاركة القدرات، والتطور السريع. يُعدّ هذا النموذج الضيق محوريًا critical لقدرة الإنترنت على التكيف بسرعة مع متطلبات المستخدمين الجديدة والتقنيات المتغيرة.

1.4:البرمجيات

تُعدّ بنى الشبكات ومواصفات البروتوكولات أمورًا أساسية، لكنّ المخطط التفصيلي الجيد لا يكفي لتفسير النجاح الباهر للإنترنت: فقد ازداد عدد أجهزة الكمبيوتر المتصلة بالإنترنت بشكل أسي [هائل] على مدى ثلاثة عقود تقريبًا (مع صعوبة الحصول على أرقام دقيقة). وقُدّر عدد مستخدمي الإنترنت بنحو 4.1 مليار [بليون] مستخدم بنهاية عام 2017، أي ما يقارب نصف سكان العالم.

ما الذي يُفسر نجاح الإنترنت؟ هناك بالتأكيد العديد من العوامل المساهمة (بما في ذلك بنيان جيد)، ولكن أحد العوامل التي جعلت الإنترنت يحقق هذا النجاح االمتعاظم [الخارج عن السيطرة] هو أن الكثير من وظائفه تُوفرها برمجيات تعمل على أجهزة حاسوب متعددة الاستخدامات. تكمن أهمية هذا في إمكانية إضافة وظائف [فعاليات] جديدة بسهولة "كبرمجة بسيطة". ونتيجة لذلك، ظهرت تطبيقات وخدمات جديدة بوتيرة مذهلة.

من العوامل ذات الصلة الزيادة الهائلة في القدرة الحاسوبية computing power المتاحة في الأجهزة التجارية [العادية]. فرغم أن شبكات الحاسوب كانت دائمًا قادرة، من حيث المبدأ، على نقل أي نوع من المعلومات، مثل عينات الصوت الرقمية digital voice والصور الرقمية digital images، وما إلى ذلك، إلا أن هذه الإمكانية لم تكن مثيرة للاهتمام بشكل خاص إذا كانت أجهزة الحاسوب المرسلة والمستقبلة لتلك البيانات بطيئة جدًا بحيث لا تتمكن من القيام بأي شيء مفيد بها. [ولكن] جميع أجهزة الحاسوب الحالية تقريبًا قادرة على تشغيل الصوت والفيديو الرقمي بسرعة ودقة تمييز عالية high resolution.

في السنوات التي تلت صدور الطبعة الأولى من هذا الكتاب، أصبحت كتابة التطبيقات الشبكية نشاطًا شائعًا أكثر بكثير، ولم تعد حكرًا على قلة من المتخصصين. وقد ساهمت عوامل عديدة في ذلك، منها توفر أدوات أفضل لتسهيل العمل على غير المتخصصين، وفتح أسواق جديدة مثل تطبيقات الهواتف الذكية.

تجدر الإشارة إلى أن معرفة كيفية تنفيذ برامج الشبكات جزء أساسي من فهم شبكات الحاسوب، ورغم أنه من غير المرجح أن تُكلف بتنفيذ بروتوكول منخفض المستوى مثل بروتوكول الإنترنت (IP)، إلا أنه من المرجح أن تجد مبررًا لتنفيذ بروتوكول على مستوى التطبيق - ذلك "التطبيق العبقري" "killer app [genius application]" الذي سيقودك إلى شهرة وثروة تفوق الخيال. للبدء، يقدم هذا القسم بعضًا من المشكلات المتعلقة بتنفيذ تطبيقات الشبكات على الإنترنت. عادةً ما تكون هذه البرامج تطبيقًا (أي مصممة للتفاعل مع المستخدمين) وبروتوكولًا (أي تتواصل مع أقرانها [من التطبيقات] عبر الشبكة).

واجهة البرمجيات التطبيقية Application Programming Interface (Sockets)

نقطة البداية عند تنفيذ تطبيق شبكة هي الواجهة التي تُصدّرها الشبكة. بما أن معظم بروتوكولات الشبكة موجودة في برمجيات (وخاصةً تلك التي تقع في أعلى مكدس البروتوكولات protocol stack)، وأن جميع أنظمة الحاسوب تقريبًا تُطبّق بروتوكولاتها الشبكية كجزء من نظام التشغيل operating system، فإننا عندما نشير إلى الواجهة "التي تُصدّرها الشبكة"، فإننا نشير عمومًا إلى الواجهة التي يُوفّرها نظام التشغيل لنظامه الفرعي للشبكات networking subsystem. تُسمى هذه الواجهة غالبًا واجهة برمجة تطبيقات الشبكة network application programming interface (API).

على الرغم من أن كل نظام تشغيل حر في تعريف واجهة برمجة التطبيقات (API) الخاصة به (ومعظمها كذلك)، إلا أن بعض هذه الواجهات حظيت بدعم واسع مع مرور الوقت؛ أي أنها نُقلت ported إلى أنظمة تشغيل [أخرى] غير نظامها الأصلي. هذا ما حدث مع واجهة المقبس socket التي وفرتها في الأصل توزيعة Berkeley لنظام يونكس Unix system، والتي تدعمها الآن جميع أنظمة التشغيل الشائعة تقريبًا، وهي أساس الواجهات الخاصة بلغات البرمجة، مثل مكتبة مقابس جافا أو بايثون socket library of Java or Python. تتمثل مزايا الدعم الشامل لواجهة برمجة تطبيقات API واحدة في سهولة نقل porting التطبيقات من نظام تشغيل إلى آخر، وإمكانية كتابة تطبيقات لأنظمة تشغيل متعددة multiple بسهولة.

> نظرًا لعدم رغبتنا في الانحياز إلى أي طرف في نقاش حول [لغات البرمجة] Java مقابل Python مقابل Go، ولأنها تظل لغة الاختيار للعناصر الداخلية للشبكة، فإن جميع أمثلة التعليمات البرمجية code examples الواردة في هذا الكتاب مكتوبة بلغة C نظرا لأنها تستخدم واجهات على مستوى نظام التشغيل operating system بشكل مباشر.

قبل وصف واجهة المقبس [المنفذ] socket interface، من المهم الفصل بين أمرين. يوفر كل بروتوكول مجموعة محددة من الخدمات، وتوفر واجهة برمجة التطبيقات (API) صيغةً لغويةً syntax تُمكّن من استدعاء invoke هذه الخدمات على نظام حاسوبي معين. بعد ذلك، يكون التنفيذ [البرمجيات المنفذة] مسؤولاً عن ربط مجموعة العمليات والأغراض الملموسة التي تُعرّفها واجهة برمجة التطبيقات API بمجموعة الخدمات المجردة abstract set of services التي يُعرّفها البروتوكول. إذا أتقنتَ تعريف الواجهة، فسيكون من الممكن استخدام صيغتها اللغوية لاستدعاء خدمات العديد من البروتوكولات المختلفة.

من المؤكد أن هذه العمومية كانت هدفًا لواجهة المقبس، على الرغم من أنها بعيدة عن الكمال.

ليس من المستغرب أن يكون المفهوم التجريدي الرئيسي لواجهة المقبس هو المقبس. ويمكن تعريف المقبس بأنه نقطة اتصال attachment عملية تطبيق محلية local application process بالشبكة. تُحدد الواجهة عمليات إنشاء المقبس، وربطه بالشبكة، وإرسال/استقبال الرسائل عبره، وإغلاقه. لتبسيط المناقشة، سنقتصر على توضيح كيفية استخدام المقابس مع بروتوكول TCP.

الخطوة الأولى هي إنشاء المقبس، ويتم ذلك من خلال العملية التالية:

```
int socket(int domain, int type, int protocol)
```

السبب في أن هذه العملية تتطلب ثلاثة محددات arguments هو أن واجهة المقبس مصممة لتكون عامة بما يكفي لدعم أي مجموعة بروتوكولات أساسية. بالتحديد، محدد النطاق domain يحدد عائلة البروتوكولات التي سيتم استخدامها: مثل PF_INET الذي يشير denote إلى عائلة الإنترنت، وPF_UNIX يشير إلى مرفق أنابيب يونكس UNIX pipe، وPF_PACKET تشير إلى الوصول المباشر direct access إلى واجهة الشبكة network interface (أي أنها تتجاوز حزمة بروتوكولات TCP/IP protocol stack). يشير محدد النوع type argument إلى دلالات الاتصال communication semantics. يُستخدم SOCK_STREAM للإشارة to denote إلى تدفق بايتات byte stream. يشير SOCK_DGRAM إلى بديل خدمة موجهة للرسائل message-oriented service، مثل تلك التي يوفرها UDP. يقوم محدد البروتوكول بتحديد [نوع] البروتوكول الذي سيتم استخدامه. في حالتنا هذه، UNSPEC هي المحدد لأن الجمع بين PF_INET وSOCK_STREAM يعني imply [البروتوكول] TCP. أخيرًا، قيمة الإرجاع return value من المقبس هي مقبض handle للمقبس المُنشأ حديثًا، أي مُعرِّف identifier يُمكننا من أن نشير إلى المقبس مستقبلًا، وتُعطى هذه القيمة كمُحدد للعمليات اللاحقة على المقبس

تعتمد الخطوة التالية على ما إذا كنتَ زبون client أم مخدم server. على جهاز المخدم، تُجري performs عملية التطبيق إجراء فتح سلبي "a passive open" - ويُعلن المخدم أنه جاهز لقبول الاتصالات prepared to accept connections، ولكنه لا يُنشئ اتصالاً فعلياً. يقوم المخدم بذلك من خلال استدعاء العمليات الثلاث التالية:

```
int bind(int socket, struct sockaddr *address, int addr_len)
int listen(int socket, int backlog)
int accept(int socket, struct sockaddr *address, int *addr_len)
```

عملية bind، كما يوحي اسمها، تربط المقبس المُنشأ حديثًا بعنوان address مُحدد. هذا هو عنوان شبكة network address الخاص بالمشارك المحلي local participant -أي المخدم. يُرجى ملاحظة أنه عند استخدامه مع بروتوكولات الإنترنت، يكون العنوان address بنية بيانات data structure تتضمن كلاً من عنوان IP المختص بالمخدم ورقم منفذ TCP port. تُستخدم المنافذ لتحديد العمليات [الإجرائيات] processes بشكل غير مباشر، وهي شكل من أشكال مفاتيح *demux*. عادةً ما يكون رقم المنفذ port number رقمًا معروفًا خاصًا بالخدمة المُقدمة؛ على سبيل المثال، تقبل خوادم الويب عادةً الاتصالات على المنفذ 80.

تُحدد عملية [تنصت] listen بعد ذلك عدد الاتصالات المعلقة على المقبس socket المُحدد. وأخيرًا، تُجري عملية [قبول] accept عملية الفتح السلبي passive open. وهي عملية حظر blocking operation لا تُعاد إلا بعد أن يُنشئ مُشارك بعيد remote participant اتصالاً connection، وعند اكتمالها، تُعيد مقبسًا جديدًا يُطابق هذا الاتصال المُنشأ حديثًا، ويحتوي المحدد address على عنوان المُشارك البعيد. يُرجى ملاحظة أنه عند عودة accept، يظل المقبس الأصلي المُعطى محددا argument موجودًا ويُطابق عملية الفتح السلبي passive open؛ ويُستخدم هذا في استدعاءات accept المُستقبلية.

على جهاز االزبون client machine، تقوم عملية التطبيق application process بإجراء فتح نشط active open؛ أي أنها تتحدد من تريد التواصل معه من خلال استدعاء العملية الفردية التالية:

```
int connect(int socket, struct sockaddr *address, int addr_len)
```

لا تعود [لا تكتمل] هذه العملية إلا بعد نجاح إنشاء اتصال TCP، وعندها يكون التطبيق حرًا في إرسال البيانات. في هذه الحالة، يحتوي على عنوان المشارك البعيد. عمليًا، عادةً ما يحدد الزبون عنوان المشارك البعيد فقط ويدع النظام يدخل المعلومات المحلية. بينما يستمع المخدم عادةً للرسائل على منفذ معروف، لا يهتم الزبون عادةً بالمنفذ الذي يستخدمه هو؛ ويختار نظام التشغيل ببساطة منفذًا غير مستخدم.

بمجرد إنشاء اتصال، تستدعي عمليات التطبيق العمليتين التاليتين لإرسال واستقبال البيانات:

```
int send(int socket, char *message, int msg_len, int flags)
int recv(int socket, char *buffer, int buf_len, int flags)
```

ترسل العملية الأولى الرسالة message المحددة عبر المقبس المحدد، بينما تستقبل العملية الثانية رسالة من المقبس المحدد إلى ذاكرة التخزين المؤقت buffer االمخصصة. تستقبل كلتا العمليتين مجموعة من الأعلام flags التي تتحكم في تفاصيل معينة للعملية.

نموذج لتطبيق Sample Application

نعرض الآن تطبيقًا لبرنامج بسيط للزبون/المخدم client/server يستخدم واجهة المقبس لإرسال الرسائل عبر اتصال TCP. يستخدم البرنامج أيضًا أدوات شبكات يونكس Unix الأخرى، والتي سنقدمها لاحقًا. يتيح تطبيقنا لمستخدم على جهاز [آلة حاسوبية] ما كتابة نص وإرساله إلى مستخدم على جهاز آخر. إنه نسخة مبسطة من برنامج الدردشة talk في Unix، وهو مشابه للبرنامج الأساسي لتطبيق المراسلة الفورية instant messaging.

نبدأ من جانب الزبون، الذي يأخذ اسم الجهاز البعيد remote device كمحدد. يستدعي أداة يونكس لترجمة هذا الاسم إلى عنوان IP الخاص بالمضيف البعيد. الخطوة التالية هي إنشاء بنية بيانات العنوان data structure (sin) المتوقعة من واجهة المقبس. لاحظ أن بنية البيانات هذه تحدد أننا سنستخدم المقبس للاتصال بالإنترنت (AF_INET). في مثالنا، نستخدم منفذ TCP port 5432 المعرف على المخدم؛ والذي لم يتم تعيينه لأي خدمة إنترنت أخرى. الخطوة الأخيرة في إعداد الاتصال connection setup هي استدعاء واجهة المقبس وتوصيل connect. بمجرد عودة العملية [وصول استجابة] operation return، يتم إنشاء الاتصال ويدخل برنامج االزبون حلقته الرئيسية main loop، والتي تقرأ النص text من الإدخال القياسي [من تجهيزات مثل لوحة المفاتيح] وترسله عبر المقبس.

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>    {مكتبات برمجة حاسوبية}
#include <netinet/in.h>
#include <netdb.h>

#define SERVER_PORT 5432
#define MAX_LINE 256

int
main(int argc, char * argv[])
{
  FILE *fp;
  struct hostent *hp;
  struct sockaddr_in sin;
  char *host;
  char buf[MAX_LINE];
  int s;
  int len;

  if (argc==2) {
    host = argv[1];
  }
  else {
    fprintf(stderr, "usage: simplex-talk host\n");
    exit(1);  {قيمة إرجاعية (1) من البرنامج function/program}
  }

  /* translate host name into peer's IP address */
  hp = gethostbyname(host);
  if (!hp) {
    fprintf(stderr, "simplex-talk: unknown host: %s\n", host);
    exit(1);
  }          {قيمة إرجاعية (1) من البرنامج function/program}
```

```
  }

  /* build address data structure */
  bzero((char *)&sin, sizeof(sin));
  sin.sin_family = AF_INET;
  bcopy(hp->h_addr, (char *)&sin.sin_addr, hp->h_length);
  sin.sin_port = htons(SERVER_PORT);

  /* active open */
  if ((s = socket(PF_INET, SOCK_STREAM, 0)) < 0) {
    perror("simplex-talk: socket");
    exit(1);
  }
  if (connect(s, (struct sockaddr *)&sin, sizeof(sin)) < 0)
  {
    perror("simplex-talk: connect");
    close(s);
    exit(1);
  }                    {الحلقة الرئيسية main loop للبرنامج}
  /* main loop: get and send lines of text */
  while (fgets(buf, sizeof(buf), stdin)) {
    buf[MAX_LINE-1] = '\0';
    len = strlen(buf) + 1;
    send(s, buf, len, 0);
  }
}
```

المخدم Server

ببساطة مماثلة, يُنشئ المخدم أولاً بنية بيانات العنوان بإدخال رقم منفذه (SERVER_PORT). بعدم تحديد عنوان IP، يكون برنامج التطبيق مستعدًا لقبول الاتصالات على أيٍّ من عناوين IP للمضيف المحلي. بعد ذلك، يُنفّذ المخدم الخطوات التمهيدية لعملية الفتح السلبي passive open؛ إذ يُنشئ المقبس، ويربطه بالعنوان المحلي، ويُحدّد الحد الأقصى لعدد الاتصالات المُعلّقة pending connections المسموح بها. وأخيرًا، تنتظر الحلقة الرئيسية main loop مُضيفًا بعيدًا ليحاول الاتصال، وعندما يفعل ذلك، تُستقبل الأحرف الواردة على الاتصال وإظهارها.

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>       {مكتبات برمجة حاسوبية}
#include <netinet/in.h>
#include <netdb.h>

#define SERVER_PORT  5432
#define MAX_PENDING  5
#define MAX_LINE     256

int
main()
{
  struct sockaddr_in sin;
  char buf[MAX_LINE];
  int len;
  int s, new_s;

  /* build address data structure */     {بناء بنية االعناوين}
  bzero((char *)&sin, sizeof(sin));
  sin.sin_family = AF_INET;
  sin.sin_addr.s_addr = INADDR_ANY;
  sin.sin_port = htons(SERVER_PORT);

  /* setup passive open */  {عملية الفتح السلبي passive open}
  if ((s = socket(PF_INET, SOCK_STREAM, 0)) < 0) {
    perror("simplex-talk: socket");
    exit(1);
  }
  if ((bind(s, (struct sockaddr *)&sin, sizeof(sin))) < 0) {
    perror("simplex-talk: bind");
    exit(1);
  }
  listen(s, MAX_PENDING);
                        {إنتظار إقامة الإتصال قبل استقبال الحروف وإظهارها}
 /* wait for connection, then receive and print text */
  while(1) {
    if ((new_s = accept(s, (struct sockaddr *)&sin, &len)) < 0) {
      perror("simplex-talk: accept");
      exit(1);
    }
    while (len = recv(new_s, buf, sizeof(buf), 0))
      fputs(buf, stdout);
    close(new_s);
  }
}
```

1.5: الأداء Performance

حتى هذه النقطة، ركزنا بشكل أساسي على الجوانب الوظيفية functional aspects للشبكات. ولكن، كما هو الحال مع أي نظام حاسوبي، يُتوقع من الشبكات الحاسوبية أيضًا أن تحقق أداءً جيدًا. ويرجع ذلك إلى أن فعالية العمليات الحسابية الموزعة distributed computing عبر الشبكة غالبًا ما تعتمد بشكل مباشر على كفاءة الشبكة في توصيل بيانات العمليات الحسابية. وبينما لا يزال القول المأثور في البرمجة "أنجز [البرنامج] بصورة صحيحة أولًا ثم إجعله سريعاً" صحيحا، إلا أنه في مجال الشبكات، غالبًا ما يكون من الضروري "التصميم من أجل الأداء". لذلك، من المهم فهم العوامل المختلفة التي تؤثر على أداء الشبكة.

معدل التدفق والتأخير Bandwidth and Latency

يُقاس أداء الشبكة بطريقتين أساسيتين: عرض النطاق/الحزمة الترددية (يُسمى أيضًا معدل التدفق throughput)* والتأخير latency (يُسمى أيضًا delay). يُحدد معدل التدفق للشبكة بعدد البتات التي يُمكن نقلها عبرها خلال فترة زمنية محددة. على سبيل المثال، قد يبلغ معدل التدفق للشبكة 10 ملايين بت/ثانية (Mbps)، مما يعني أنها قادرة على نقل 10 ملايين بت في الثانية. من المفيد أحيانًا التفكير في معدل التدفق من حيث المدة التي يستغرقها نقل كل بت من البيانات. على سبيل المثال، في شبكة بسرعة 10 ميجابت/ثانية، يستغرق نقل كل بت 0.1 ميكروثانية (µs) microseconds.

عرض الحزمة الترددية ومعدل نقل البيانات [معدل الإرسال الإفتراضي nominal transmission rate] مصطلحان مختلفان بشكل طفيفً [وصعب ملاحظته]. كبداية، عرض الحزمة الترددية bandwidth هو قياس للحزمة الترددية frequency band [التي تتواجد فيها الإشارة]. على سبيل المثال، كانت خطوط الهاتف الصوتية voice-grade telephone التقليدية legacy تدعم حزمة ترددية يتراوح بين 300 و3300 هرتز [Hertz] Hz؛ ولذلك فإن عرض حزمتها الترددية 300-3300=3000 Hz. إذا استُخدمت كلمة "عرض الحزمة الترددية" مقاسة بالهرتز، فربما ذلك يشير إلى مدى [أنواع] الإشارات التي يمكن استعمالها.**

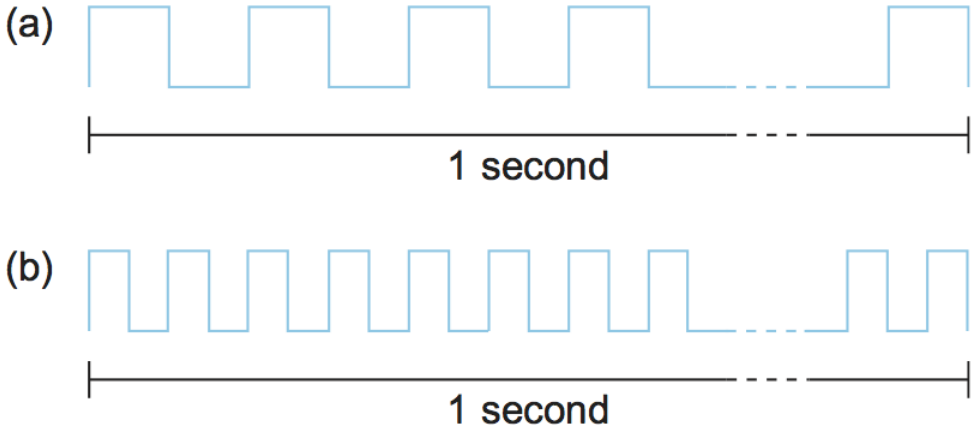
عند الحديث عن عرض الحزمة الترددية لرابط اتصال communication link، فإننا نشير عادةً إلى عدد البتات في الثانية bits per second التي يمكن إرسالها عبر الرابط. ويُسمى هذا أحيانًا معدل البيانات data rate. يمكننا القول إن عرض الحزمة الترددية لرابط إيثرنت Ethernet هو 10 ميجابت في الثانية Mbps. ومن المفيد، يمكن أيضًا التمييز بين أقصى معدل إرسال البيانات على الرابط وعدد البتات في الثانية التي يمكننا نقلها فعليًا عبر الرابط في الممارسة العملية. نميل إلى استخدام مصطلح "معدل التدفق" throughput للإشارة إلى الأداء المُقاس للنظام. وبالتالي، نظرًا لعوامل عدم كفاءة التنفيذ المختلفة، وهكذا قد يحقق زوج من العقد المتصلة عبر رابط بعرض حزمة ترددية يبلغ 10 ميجابت في الثانية معدل تدفق يبلغ 2 ميجابت في الثانية فقط. هذا يعني أن تطبيقًا على أحد المضيفين يمكنه إرسال البيانات إلى المضيف الآخر بسرعة [لاتزيد عن] 2 ميجابت في الثانية.

أخيرًا، كثيرًا ما نتحدث عن متطلبات عرض الحزمة الترددية لتطبيق ما. وهو عدد البتات في الثانية bits per second التي يحتاج التطبيق لنقلها عبر الشبكة ليعمل بشكل مقبول. بالنسبة لبعض التطبيقات، قد يكون هذا "كل ما يمكن الحصول عليه"؛ وبالنسبة لتطبيقات أخرى، قد يكون رقمًا ثابتًا (يفضل ألا يتجاوز عرض الحزمة الترددية المتاح للرابط)؛ وكذلك بالنسبة لتطبيقات أخرى، قد يكون رقمًا يتغير مع مرور الوقت. سنتناول هذا الموضوع بمزيد من التفصيل لاحقًا في هذا القسم.

بينما يمكنك التحدث عن عرض الحزمة الترددية للشبكة ككل، قد ترغب أحيانًا في أن تكون أكثر دقة، مع التركيز، على سبيل المثال، على عرض الحزمة الترددية لرابط مادي physical link واحد أو قناة منطقية logical channel من عملية إلى عملية process-to-process. على المستوى المادي physical level، يتحسن عرض النطاق الترددي باستمرار، دون نهاية في الأفق. بديهيًا، إذا كنت تفكر في ثانية من الزمن كمسافة يمكنك قياسها بمسطرة وعرض الحزمة الترددية على أنه عدد البتات التي يمكن حشرها في تلك المسافة، فيمكنك عندئذٍ التفكير في كل بت كنبضة ذات عرض ما. على سبيل المثال، يبلغ عرض كل بت على رابط بسرعة 1 ميجابت في الثانية 1 ميكروثانية، بينما يبلغ عرض كل بت على رابط بسرعة 2 ميجابت في الثانية 0.5 ميكروثانية، كما هو موضح في الشكل 1. كلما كانت تقنية الإرسال والاستقبال أكثر تطورًا، كلما أصبح كل بت أضيق، وبالتالي، زاد عرض الحزمة الترددية. بالنسبة للقنوات المنطقية من عملية إلى عملية، تتأثر الحزمة الترددية أيضًا بعوامل أخرى، بما في ذلك عدد المرات التي يتعين على البرنامج الذي ينفذ القناة التعامل فيها مع كل بت من البيانات، وربما تحويلها.

*[سرعة التدفق throughput في مجال شبكات الحاسوب تكون أقل من سرعة الإرسال الإفتراضية nominal transmission speed على الروابط نظرا لأن ارسال المعتطيات data transmission يتطلب تنسيق دائم مع الشبكة وكشف أخطاء المعطيات data error detection وإعادة الإرسال].

**[بشكل عام فإن إشارات حزمة الطيف االضيقة narrowband signals تتغير ببطء مع الوقت بينما إشارات حزمة الطيف الواسع wideband signals تتغير بسرعة مع الوقت. وبالتالي فإن إشارات حزمة الطيف الواسع تتمكن من إرسال معدلات معلومات عالية high information/bit rate, بينما إشارات حزمة الطيف االضيقة لا تتمكن من إرسال سوى معدلات معلومات منخفضة low information/bit rate].



الشكل 1: يمكن إعتبار أن بتات مرسلة على معدل ارسال معين على ان لها عرض معين: (a) في ارسال بمعدل 1 Mbps (كل بت يكون عرضها 1 ميكروثانية µs)؛ (b) في ارسال بمعدل 2 Mbps (كل بت يكون عرضها 0.5 ميكروثانية µs)

مقياس الأداء performance metric الثاني، وهو التأخير latency، يُشير إلى المدة التي تستغرقها الرسالة للانتقال من أحد طرفي الشبكة إلى الطرف الآخر. (كما هو الحال مع عرض النطاق الترددي، يُمكننا التركيز على زمن وصول رابط واحد أو قناة اتصال من طرف-إلى-طرف end-to-end channel). يُقاس التأخير بمقياس الوقت حصريا. على سبيل المثال، قد يبلغ زمن وصول شبكة عابرة للقارات transcontinental network 24 ملي ثانية (ms)؛ أي أن الرسالة تستغرق 24 ms للانتقال من أحد سواحل أمريكا الشمالية إلى الطرف الآخر. هناك العديد من الحالات التي يكون فيها معرفة المدة التي يستغرقها إرسال رسالة ذهابا وإيابا بين طرفي الشبكة أكثر أهمية من معرفة زمن الوصول في اتجاه واحد. نُطلق على هذا زمن الوصول ذهابًا وإيابًا round-trip time (RTT) للشبكة.

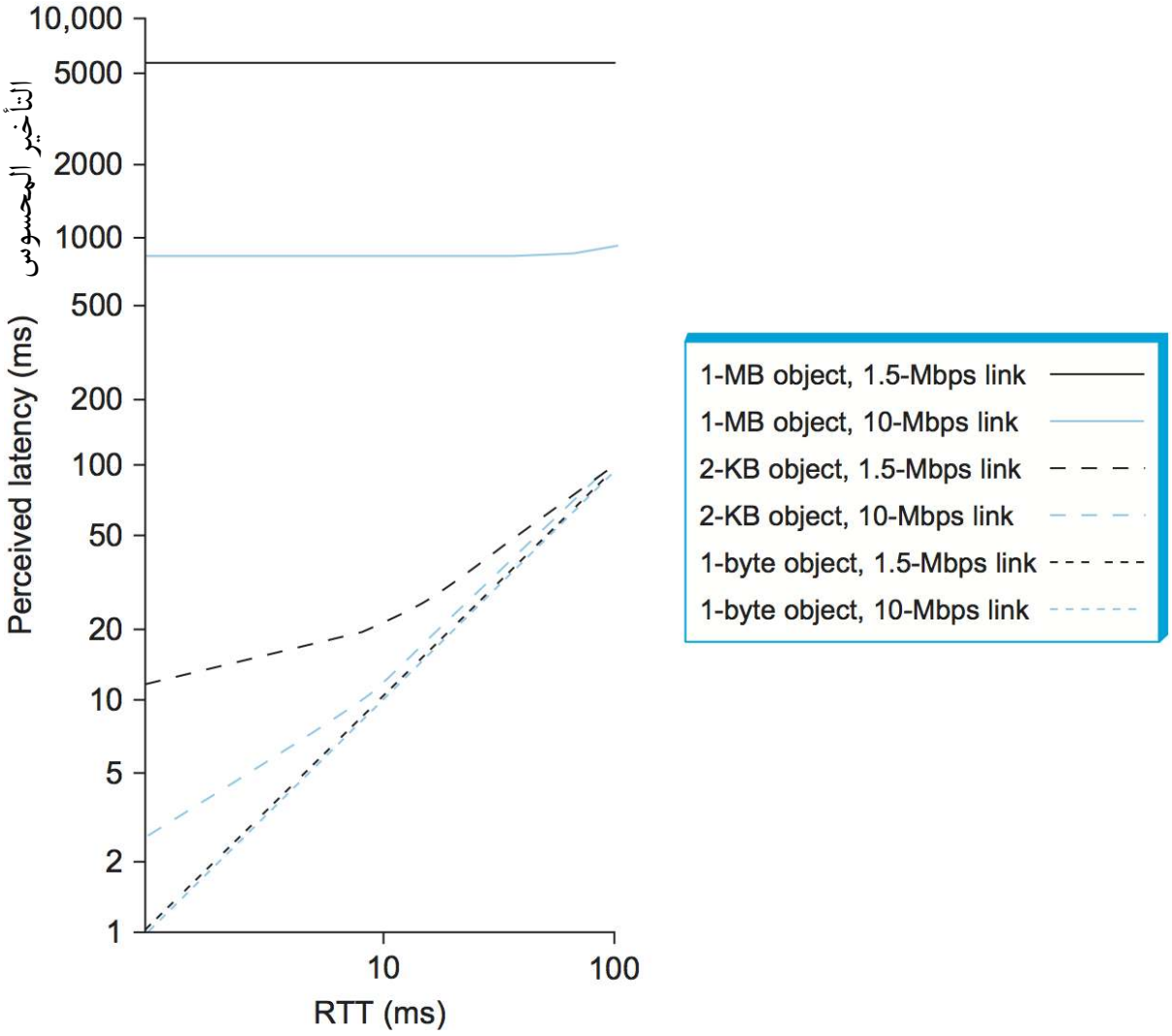
غالبًا ما نفكر في زمن التأخير بثلاثة مكونات. أولًا، هناك تأخير انتشار سرعة الضوء [بإفتراض استعمال الألياف الضوئية أو المكرويف] . يحدث هذا التأخير لأنه لا يمكن لأي شيء، بما في ذلك البتات على السلك، أن ينتقل أسرع من سرعة الضوء. إذا كنت تعرف المسافة بين نقطتين، يمكنك حساب زمن الوصول بسرعة الضوء، مع ضرورة توخي الحذر لأن الضوء ينتقل عبر وسائط مختلفة بسرعات مختلفة: ينتقل بسرعة 3.0×10^8 متر/ثانية في الفراغ، و 2.3×10^8 متر/ثانية في كابل نحاسي، و 2.0×10^8 متر/ثانية في الألياف الضوئية. ثانيًا، هناك مقدار الوقت اللازم للإرسال وحدة بيانات. ويعتمد هذا على عرض النطاق الترددي للشبكة وحجم الحزمة packet size التي تُحمل فيها البيانات. ثالثًا، قد تكون هناك تأخيرات في صفوف الانتظار queuing delays داخل الشبكة، لأن مبدلات الحزم تحتاج عادةً إلى تخزين store الحزم لبعض الوقت قبل إعادة توجيهها forwarding على رابط خرج outbound link. لذا، يمكننا تعريف زمن الوصول الإجمالي على أنه

```
Latency = Propagation + Transmit + Queue
Propagation =  Distance/SpeedOfLight
Transmit = Size/Bandwidth
```

حيث Distance هو طول السلك الذي ستنتقل عبره البيانات، وSpeedOfLight هي سرعة الضوء الفعلية عبر هذا السلك [وسط الإنتشار]، و Size هو قياس size الحزمة، وbandwidth هو عرض الحزمة الترددية [معدل التدفق] الذي تُرسل عبره الحزمة. يُرجى ملاحظة أنه إذا كانت الرسالة تحتوي على بت واحد فقط، ونحن نتحدث عن رابط واحد (بدلاً من شبكة كاملة)، فإن مصطلحي Transmit و Queue غير ذي صلة not relevant، ويتوافق زالتأخير مع تأخير الانتشار propagation فقط.

يتحد الحزمة الترددية والتأخير لتحديد خصائص أداء رابط أو قناة معينة. ومع ذلك، تعتمد أهميتهما النسبية على التطبيق. بالنسبة لبعض التطبيقات، يهيمن التأخير على الحزمة الترددية. على سبيل المثال، يكون الزبون الذي يرسل رسالة مكونة من بايت واحد إلى مخدم ويستقبل رسالة مكونة من بايت واحد في المقابل مقيدًا بالتأخير. وبافتراض عدم وجود أي معالجة حسابية تستغرق وقتا في إعداد الاستجابة، سيعمل التطبيق بشكل مختلف كثيرًا على قناة عبر القارات ذات تأخير للوقت يبلغ 100 مللي ثانية عن أدائه على قناة عبر الغرفة ذات تأخير للوقت يبلغ 1 مللي ثانية. ومع ذلك، فإن ما إذا كانت القناة 1 ميجابت في الثانية Mps أو 100 ميجابت في الثانية غير ذي أهمية نسبيًا، حيث أن الأول يعني أن وقت إرسال بايت هو 8 ميكروثانية µs والأخير يعني أن 0.08 = Transmit ميكروثانية µs.

على النقيض من ذلك، لنفترض أن برنامج مكتبة رقمية يُطلب منه جلب صورة بحجم 25 ميجابايت (MB) - كلما زاد عرض الحزمة الترددية [معدل التدفق] المتاحة، زادت سرعة إعادة الصورة إلى المستخدم. هنا، يُهيمن معدل التدفق للقناة على الأداء. لنفترض أن معدل التدفق للقناة 10 ميجابت في الثانية. سيستغرق إرسال الصورة 20 ثانية ($25 \times 10^6 \times 8 / 10 \times 10^6 = 20$)، مما يجعل الأمر غير مهم نسبيًا إذا كانت الصورة على الجانب الآخر من قناة مدتها 1 ms أو قناة مدتها 100 ms؛ فالفرق بين وقت الاستجابة 20.001 ثانية و20.1 ثانية ضئيل.



الشكل 2: التأخير المحسوس perceived latency (وقت الاستجابة response time) مقابل وقت الذهاب والإياب round-trip-time (RTT) لأحجام مختلفة من الأغراض [الرسائل المرسلة] وسرعات الارتباط link speeds

الشكل 2 يعطيك فكرة عن كيف أن يهيمن التآخير أو سرعة التدفق على الأداء في ظروف مختلفة. يوضح الرسم البياني المدة التي يستغرقها نقل غرض/رسائل بأحجام مختلفة (1 بايت، 2 كيلوبايت، 1 ميجابايت) عبر شبكات ذات round-trip-time (RTT) يتراوح من 1 إلى 100 ms وسرعات الرابط link speed إما 1.5 أو 10 ميجابايت في الثانية Mbps. نستخدم مقاييس لوغاريتمية logarithmic scales لإظهار الأداء النسبي relative performance. بالنسبة لكائن بحجم 1 بايت (مثل ضغطة مفتاح keystroke)، يظل التآخير مساويًا تقريبًا لزمن RTT، بحيث لا يمكنك التمييز بين شبكة بسرعة 1.5 ميجابايت في الثانية وشبكة بسرعة 10 ميجابايت في الثانية. بالنسبة لغرض بحجم 2 كيلوبايت (مثل رسالة بريد إلكتروني email message)، تُحدث سرعة الرابط فرقًا كبيرًا على التآخير في الشبكة ذات RTT يبلغ 1 ms ولكن فرقًا ضئيلًا على شبكة ذات RTT يبلغ 100 ms. أما بالنسبة لغرض بحجم 1 ميجابايت (مثل صورة رقمية digital image)، لا يُحدث RTT أي فرق - حيث أن سرعة الرابط هي التي تُهيمن على RTT في الأداء عبر النطاق الكامل full range.

تجدر الإشارة إلى أننا نستخدم مصطلحي "الزمن" و"التأخير" "latency" and "delay" في هذا الكتاب بشكل عام للإشارة إلى المدة اللازمة "التأخير" لأداء وظيفة معينة، مثل توصيل رسالة أو نقل غرض [ملف, صورة رقمية]. عند الإشارة إلى المدة الزمنية المحددة التي تستغرقها إشارة للانتشار من أحد طرفي الرابط إلى الطرف الآخر، نستخدم مصطلح "تأخير الانتشار" "propagation delay". كما نوضح في سياق المناقشة ما إذا كنا نشير إلى التأخير في اتجاه واحد latency أم التأخير ذهابًا وإيابًا (RTT).

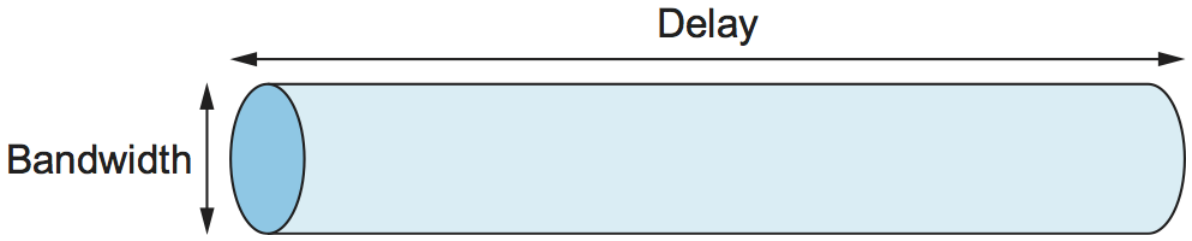
على هامش ذلك، أصبحت أجهزة الكمبيوتر سريعة للغاية لدرجة أنه من المفيد أحيانًا عند توصيلها بالشبكات، التفكير، ولو مجازيًا، في عدد التعليمات لكل ميل mile. تخيل ماذا يحدث عندما يرسل جهاز كمبيوتر قادر على تنفيذ 100 مليار تعليمة instruction في الثانية رسالة عبر قناة ذات زمن استجابة يبلغ 100 مللي ثانية. (ولتسهيل العملية الحسابية، افترض أن الرسالة تغطي مسافة 5000 ميل). إذا ظل هذا الجهاز خاملاً لمدة 100 ms كاملة في انتظار رسالة رد، فإنه يفقد القدرة على تنفيذ 10 مليارات تعليمة، أو مليوني تعليمة لكل ميل. ولذلك يجب أن يكون الأمر يستحق المرور عبر الشبكة [بما يعني ضرورة خدمة الشبكة] لتبرير هذا الهدر.

سرعة الإرسال x التأخير Delay x Bandwidth Product

من المفيد أيضًا الحديث عن حاصل ضرب هذين المقياسين metrics، والذي يُسمى غالبًا حاصل ضرب التأخير × سرعة الإرسال [معدل التدفق, عرض الحزمة الترددية]. بديهيًا، إذا اعتبرنا أن هناك قناة بين زوج من العمليات كأنبوب مجوف (انظر الشكل 3)، حيث يتوافق التأخير [زمن الوصول بين زوج العمليات] مع طول الأنبوب وتعطي سرعة الإرسال قطر الأنبوب، فإن حاصل ضرب التأخير × سرعة الإرسال يعطي حجم الأنبوب - وهو أقصى عدد من البتات التي يمكن أن تمر عبر الأنبوب في أي لحظة معينة. بعبارة أخرى، إذا كان التأخير [زمن الوصول] (مقاسًا بالوقت) يتوافق مع طول الأنبوب، فبموجب عرض كل بت (مقاسًا أيضًا بالوقت)، يمكنك حساب عدد البتات التي تتسع في الأنبوب. على سبيل المثال، قناة عابرة للقارات ذات زمن وصول أحادي الاتجاه يبلغ 50 ms وعرض نطاق ترددي يبلغ 45 Mbps قادرة على استيعاب

$$50 \times 10^3 \text{ sec} \times 45 \times 10^6 \text{ bits/sec} = 2.25 \times 10^6 \text{ bits}$$

أو ما يقارب 280 KB من البيانات. بمعنى آخر، تستوعب هذه القناة (الأنبوب) عددًا من البايتات يعادل ما تستوعبه ذاكرة حاسوب شخصي من أوائل ثمانينيات القرن العشرين.



الشكل 3: الشبكة كأنبوب

من المهم معرفة قيمة **التأخير**×**سرعة الإرسال** عند بناء شبكات عالية الأداء، لأنه يُمثل عدد البتات التي يجب على المُرسِل إرسالها قبل وصول البتة الأولى إلى المُستقبِل. إذا كان المُرسِل يتوقع من المُستقبِل إرسال إشارة ما تفيد ببدء وصول البتات، واستغرقت هذه الإشارة زمن انتقال [تأخير] قناة أخرى لانتشارها، فيمكن للمُرسِل إرسال بيانات تُعادل قيمة **RTT** × **سرعة الإرسال** قبل أن يسمع من المُستقبِل بأن كل شيء على ما يُرام. يُقال إن البتات في الأنبوب "قيد النقل"، مما يعني أنه إذا طلب المُستقبِل من المُرسِل إيقاف الإرسال، فقد يستقبل ما يصل إلى قيمة RTT × سرعة الإرسال قبل أن يتمكن المُرسِل من الاستجابة. في مثالنا أعلاه، تُعادل هذه الكمية 5.5×10^6 بت (671 كيلوبايت KB) من البيانات. من ناحية أخرى، إذا لم يُكمل المُرسِل الأنبوب - أي لم يُرسل قيمة RTT × سرعة الإرسال بالكامل قبل أن يتوقف لانتظار إشارة [من المستقبل] - فلن يُحقق المُرسِل الاستفادة الكاملة من الشبكة.

لاحظ أننا غالبًا ما نهتم بسيناريو زمن الاستجابة (RTT)، والذي نشير إليه ببساطة بحاصل ضرب التأخير في سرعة الإرسال، دون أن نحدد صراحةً أن "التأخير" delay هو زمن الاستجابة RTT (أي ضرب التأخير أحادي الاتجاه one-way-delay في اثنين). عادةً، يتضح من السياق ما إذا كان "التأخير" في زمن الاستجابة في اتجاه واحد أو زمن الاستجابة (RTT). يعرض الجدول 1 بعض الأمثلة على حاصل ضرب زمن الاستجابة RTT في سرعة الإرسال لبعض روابط الشبكة النموذجية.

الجدول 1: أمثلة التأخير× سرعة الإرسال RTT x Bandwidth

Link Type	Bandwidth	One-Way Distance	RTT	RTT x Bandwidth
Wireless LAN	54 Mbps	50 m	0.33 µs	18 bits
Satellite	1 Gbps	35,000 km	230 ms	230 Mb
Cross-country fiber	10 Gbps	4,000 km	40 ms	400 Mb

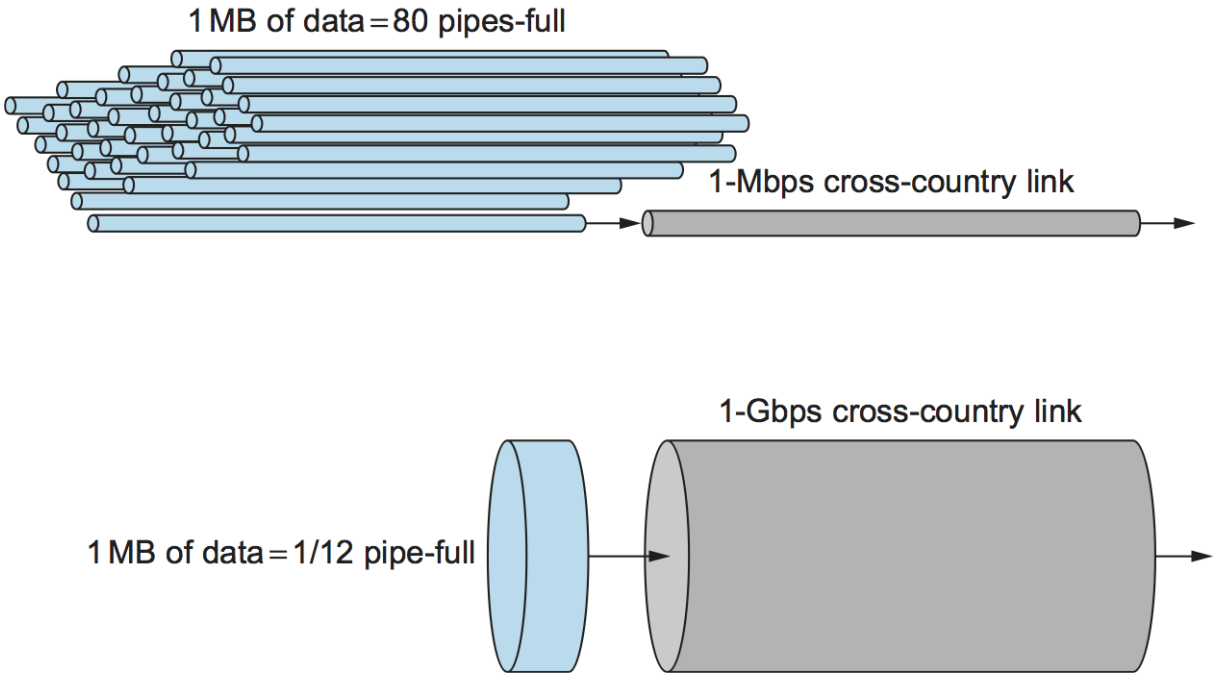
الشبكات عالية-السرعة High-Speed Networks

إن الزيادة الظاهرية المستمرة في سرعة الإرسال تدفع مصممي الشبكات إلى البدء في التفكير فيما يحدث في الحد الأقصى أو، بعبارة أخرى، ما هو تأثير توفر سرعة إرسال غير محدودة على تصميم الشبكة.

على الرغم من أن الشبكات عالية السرعة تُحدث تغييرًا دراماتيكيا في سرعة الإرسال المتاح للتطبيقات، إلا أن تأثيرها على نظرتنا للشبكات، من نواحٍ عديدة، يكمن في ما لا يتغير مع زيادة سرعة الإرسال [زيادة عرض الحزمة الترددية]: سرعة الضوء. بمعنى آخر، لا تعني "السرعة العالية" أن زمن الوصول يتحسن بنفس معدل تحسن عرض الحزمة الترددية؛ فزمن الوصول عبر القارات لرابط بسرعة 1 جيجابت في الثانية Gbps هو نفسه 100 ms لرابط بسرعة 1 ميجابت في الثانية Mbps.

لتقدير أهمية التزايد المستمر في سرعة الإرسال في ظل التأخير latency الثابت، لننظر إلى ما هو مطلوب لنقل ملف بحجم 1 ميجابايت عبر شبكة بسرعة 1 ميجابايت في الثانية Mbps، مقارنةً بشبكة بسرعة 1 جيجابت في الثانية Gbps، وكلاهما يتمتع بتأخير[ذهاباً وإياباً] RTT يبلغ 100 مللي ثانية. في حالة شبكة 1 Mbps، يستغرق نقل الملف 80 مرة ذهابًا وإيابًا؛ وخلال كل مرة، يتم إرسال 1.25% من الملف. في المقابل، لا يكاد ملف 1 ميجابايت MB نفسه يملأ زمن وصول واحد لرابط 1 Gbps، والذي يبلغ حاصل ضرب التأخير × سرعة الإرسال فيه 12.5 ميجابايت.

يوضح الشكل 4 الفرق بين الشبكتين. في الواقع، يبدو ملف 1 ميجابايت كتدفق بيانات يجب نقله عبر شبكة بسرعة 1 ميجابت في الثانية، بينما يبدو كحزمة بيانات واحدة على شبكة بسرعة 1 Gbps. ولتوضيح هذه النقطة، تجدر الإشارة إلى أن ملف 1 MB بالنسبة لشبكة بسرعة 1 Gbps يُعادل حزمة بيانات بحجم 1 KB بالنسبة لشبكة بسرعة 1 Mbps.



الشكل 4 العلاقة بين سرعة الإرسال [عرض االحزمة الترددية] والتأخير latency [ضمن الرابط]. ملف بحجم 1 MB سيملأ 80 مرة رابطًا بسرعة 1 Mbps، بينما سيملأ 1/12 فقط من رابط بسرعة 1 Gbps.

طريقة أخرى للتفكير في هذا الوضع هي إمكانية نقل بيانات أكثر خلال كل RTT على شبكة عالية السرعة، لدرجة أن زمن انتقال واحد يصبح وقتًا طويلاً. لذا، بينما لا يُفكر المرء مرتين في الفرق بين نقل ملف يستغرق 101 RTT بدلًا من 100 RTT (بفارق نسبي 1%)، فجأةً يصبح الفرق بين 1 RTT و 2 RTT كبيرًا - زيادة بنسبة 100%. بمعنى آخر، بدأ التأخير، وليس معدل االتدفق [الإرسال]، يُسيطر على تفكيرنا في تصميم الشبكات.

لعل أفضل طريقة لفهم العلاقة بين معدل التدفق [الإرسال] والتأخير هي العودة إلى الأساسيات. يُحدد معدل التدفق الفعال من البداية إلى النهاية end-to-end عبر الشبكة من خلال العلاقة البسيطة:

Throughput = TransferSize / TransferTime

حيث لا يشمل فقط زمن الإنتشار أحادي الاتجاه المحدد سابقًا، بل يشمل أيضًا أي وقت إضافي مُستغرق في طلب أو إعداد نقل [المعلومات]. عمومًا، نُمثل هذه العلاقة على أنها

TransferTime = RTT + (1/Bandwidth) x TransferSize

نستخدم هذه العملية الحسابية لحساب زمن النقل transfer time لرسالة الطلب request message المرسلة عبر الشبكة والبيانات التي يعاد إرسالها. على سبيل المثال، لنفترض أن مستخدمًا يريد جلب ملف بقياس/بحجم [TransferSize] 1 MB عبر شبكة بسرعة إرسال 1 Gbps، وتأخير RTT 100 ms . يشمل ذلك كلاً من زمن الإرسال لـ (1 / 1 Gbps × 1 MB = 8 ms) 1 MB وتأخير RTT البالغ 100 ms، ليصبح إجمالي زمن النقل 108 ms. هذا يعني أن معدل النقل الفعلي سيكون:

1 MB / 108 ms = 74.1 Mbps

وليس 1 Gbps. من الواضح أن نقل كميات أكبر من البيانات سيساعد على تحسين معدل التدفق throughput، حيث سيؤدي حجم النقل الكبير للغاية في الحد الأقصى إلى اقتراب معدل التدفق الفعلي من سرعة الإرسال [عرض االحزمة الترددية] bandwidth للشبكة. من ناحية أخرى، فإن الاضطرار إلى تحمل أكثر من 1 RTT - على سبيل المثال، لإعادة إرسال الحزم المفقودة سيؤثر سلبًا على معدل التدفق الفعلي لأي نقل ذي حجم محدود، وسيكون ذلك ملحوظًا بشكل خاص في عمليات النقل الصغيرة.

احتياجات أداء برامج التطبيقات Applications Performance Needs

تناولت المناقشة في هذا القسم الأداء من منظور الشبكة network-centric view؛ أي أننا تحدثنا من حيث ما يدعمه رابط link أو قناة channel معينة. وكان الافتراض غير المعلن هو أن احتياجات برامج التطبيقات بسيطة، فهي تريد أقصى سرعة إرسال/تدفق [عرض الحزمة الترددية] توفره الشبكة. وينطبق هذا بالتأكيد على مثال برنامج المكتبة الرقمية المذكور سابقًا، والذي يسترجع retrieve صورة [مخزنة في المكتبة] بحجم 250 MB؛ فكلما زادت سرعة الإرسال المتاحة، كلما زادت سرعة البرنامج في إعادة الصورة إلى المستخدم.

ومع ذلك، تستطيع بعض التطبيقات تحديد حد أقصى لسرعة التدفق الذي تحتاجه. تطبيقات الفيديو مثال رئيسي على ذلك. لنفترض أن أحدهم يريد بث فيديو بحجم ربع شاشة تلفزيون عادية؛ أي بدقة 352 × 240 بكسل pixel. إذا كان كل بكسل مُمَثَّلاً بـ 24 بتًا من المعلومات، كما هو الحال في الألوان ذات 24 بتًا، فسيكون حجم كل إطار هو

$$(352 \times 240 \times 24)/8 = 247.5 \text{ KB}$$

إذا احتاج تطبيق [الفيديو] إلى دعم معدل إطارات يبلغ 30 إطارًا في الثانية، فقد يطلب معدل نقل بيانات يبلغ 75 Mbps. لا يُهمّ هذا التطبيق قدرة الشبكة على توفير سرعة إرسال أعلى، نظرًا لمحدودية كمية البيانات التي يتعين إرسالها خلال فترة زمنية محددة.

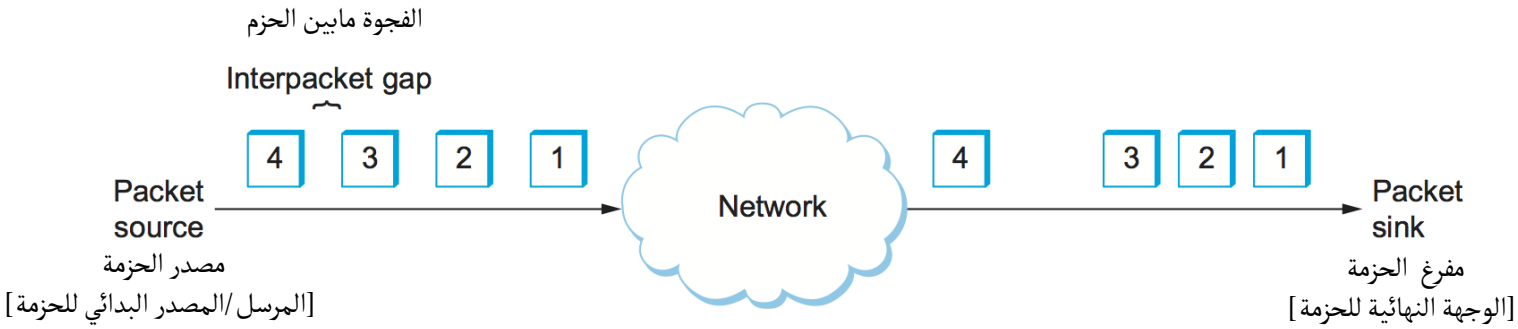
للأسف، ليس الأمر بهذه البساطة كما يوحي هذا المثال. نظرًا لأن الفرق بين أي إطارين متجاورين في بث الفيديو غالبًا ما يكون ضئيلًا، يُمكن ضغط الفيديو video compression بإرسال الفرق بينهما فقط. كما يُمكن ضغط كل إطار لأن العين البشرية لا تُدرك جميع تفاصيل الصورة بسهولة.لذلك لا يتدفق الفيديو المضغوط بمعدل ثابت، بل يتغير مع مرور الوقت وفقًا لعوامل مثل كمية الحركة والتفاصيل في الصورة وخوارزمية الضغط compression algorithm المُستخدمة. وبالتالي، يُمكن تحديد متوسط متطلبات معدل سرعة الإرسال، ولكن قد يكون المعدل اللحظي instantaneous rate أعلى أو أقل.

المسألة الأساسية هي الفترة الزمنية time interval التي يُحسب خلالها المتوسط. لنفترض أن تطبيق الفيديو هذا يُمكن ضغطه إلى الحد الذي يحتاج فيه إلى 2 Mbps فقط في المتوسط. إذا أرسل [تطبيق الفيديو] 1 Mbps، ثم 3 Mbps التالية، فسيكون **متوسط** معدل الإرسال 2 Mbps خلال فترة الثانيتين؛ ومع ذلك، لن يكون هذا كافيًا في قناة مُصممة لدعم ما لا يزيد عن 2 Mbps. من الواضح أن مجرد معرفة **متوسط** احتياجات التطبيق من معدل التدفق/الإرسال [عرض الحزمة الترددية] لن يكون كافيًا دائمًا.

بشكل عام، يُمكن تحديد حد أقصى لحجم الدفعة burst التي يُحتمل أن يُرسلها تطبيق كهذا. يُمكن وصف الدفعة بمعدل ذروة peak rate يُحافظ عليه لفترة زمنية مُحددة. أو يُمكن وصفها بعدد البايتات التي يُمكن إرسالها بمعدل الذروة قبل العودة إلى المعدل المتوسط أو معدل أقل. إذا كان معدل الذروة هذا أعلى من سعة القناة channel capacity المُتاحة، فيجب تخزين البيانات الزائدة مؤقتًا في مكان ما، لإرسالها لاحقًا. تُمكّن معرفة حجم الدفعة المُحتمل إرسالها مُصمم الشبكة من تخصيص سعة تخزين مؤقت كافية لاستيعاب الدفعة.

كما هو الحال مع احتياجات التطبيقات لعرض عرض الحزمة الترددية، قد تكون متطلبات تأخير التطبيق أكثر تعقيدًا من مجرد "أقل تأخير ممكن". في حالة التأخير، لا يهم أحيانًا ما إذا كان زمن الوصول أحادي الاتجاه للشبكة 100 ms أو 500 ms، بقدر ما يهم مدى اختلاف زمن الوصول من حزمة إلى أخرى. يُطلق على هذا الاختلاف في زمن الوصول اسم "التذبذب" jitter.

لنفترض أن المصدر يرسل حزمةً واحدةً كل 33 ms، كما هو الحال في تطبيق فيديو يرسل إطاراتٍ 30 مرةً في الثانية. إذا وصلت الحزم إلى وجهتها بفارق 33 ms بالضبط، فيمكننا استنتاج أن التأخير الذي واجهته كل حزمة في الشبكة كان متماثلًا تمامًا. أما إذا كانت المسافة بين وصول الحزم إلى وجهتها - والتي تُسمى أحيانًا الفجوة بين الحزم - متغيرة، فلا بد أن التأخير الذي واجهته سلسلة الحزم كان متغيرًا أيضًا، ويُقال إن الشبكة قد أدخلت تذبذبًا في تدفق الحزم، كما هو موضح في الشكل 5. لا يحدث هذا التباين عادةً في رابط مادي واحد، ولكنه قد يحدث عندما تواجه الحزم تأخيرات انتظار مختلفة في شبكة تبديل حزم متعددة القفزات multihop [بين عدة عقد]. يتوافق هذا التأخير في الانتظار مع مُكوّن التأخير المُعرف سابقًا في هذا القسم، والذي يتغير مع الوقت.



الشكل 5: تذبذب في التأخير delay jitter ناتج عن الشبكة

لفهم أهمية التذبذب [في التأخير]، لنفترض أن الحزم المرسلة عبر الشبكة تحتوي على إطارات فيديو video frames، ولعرض هذه الإطارات على الشاشة، يحتاج جهاز الاستقبال إلى استقبال إطار جديد كل 33 ms. إذا وصل إطار مبكرًا، فيمكن لجهاز الاستقبال حفظه ببساطة حتى يحين وقت عرضه. أما إذا وصل الإطار متأخرًا، فلن يحصل جهاز الاستقبال على الإطار الذي يحتاجه في الوقت المناسب لتحديث الشاشة، وستتأثر جودة الفيديو ولن يكون [عرض الفيديو] سلسًا. تجدر الإشارة إلى أنه ليس من الضروري إزالة االتذبذب، بل معرفة مدى سوئه. ويرجع ذلك إلى أنه إذا كان جهاز الاستقبال يعرف الحدين الأعلى والأدنى للتأخير الذي يمكن أن تواجهه الحزمة، فيمكنه تأخير وقت بدء تشغيل الفيديو playback time (أي عرض الإطار الأول) لفترة كافية لضمان وجود إطار لعرضه عند الحاجة إليه في المستقبل. يؤجل جهاز الاستقبال الإطار، مما يُخفف االتذبذب بفعالية، عن طريق تخزينه في ذاكرة تخزين مؤقت.

1.6: المنظور الأوسع Broader Perspective

سرعة تطور سمات أنظمة الشبكات

يُقدّم هذا الفصل بعض الجهات المعنية بشبكات الحاسوب - مصممو الشبكات، ومطورو التطبيقات، والمستخدمون، ومشغلو الشبكات - للمساعدة في تحديد المتطلبات التقنية technical requirements التي تغير شكل كيفية تصميم الشبكات وبنائها. يفترض هذا أن جميع قرارات التصميم تقنية بحتة، ولكن هذا ليس صحيحًا عادةً. فهناك عوامل أخرى عديدة، من قوى السوق، إلى السياسات الحكومية، وصولًا إلى الاعتبارات الأخلاقية، تُؤثّر أيضًا على كيفية تصميم الشبكات وبنائها.

ومن بين هذه العوامل، فإن ساحة السوق هو الأكثر تأثيرا، ويتوافق مع التفاعل بين مشغلي الشبكات (على سبيل المثال، Comcast ، AT&T ، China Mobile ، NTT ، DT ، Verizon)، وبائعي معدات الشبكات network equipment vendors (على سبيل المثال، Juniper ، Cisco ، NEC ، Huawei ، Nokia ، Ericsson)، ومقدمي التطبيقات والخدمات (على سبيل المثال، Microsoft ، Amazon ، Google ، Facebook ، Spotify ، Netflix ، Apple)، وبالطبع المشتركين والزبائن (أي الأفراد، ولكن أيضا الشركات والمؤسسات الكبيرة). لا تكون الخطوط الفاصلة بين هذه الجهات واضحة دائمًا، حيث تلعب العديد من الشركات أدوارًا متعددة. أبرز مثال على ذلك شركات الخدمات السحابية cloud computing providers الكبرى، التي (أ) تبني معدات شبكاتها الخاصة باستخدام مكونات تجارية، (ب) تنشر وتُشغّل شبكاتها الخاصة، و(ج) تُقدّم خدمات وتطبيقات للمستخدم النهائي end-user عبر شبكاتها.

عند مراعاة هذه العوامل الأخرى في عملية التصميم التقني، تُدرك وجود افتراضين ضمنيين في النسخة التقليدية من القصة، وهما بحاجة إلى إعادة تقييم. أولهما أن تصميم الشبكة نشاط لمرة واحدة.أي "أنشئها مرة واحدة واستخدمها للأبد" (باستثناء ترقيات الأجهزة ليتمكن المستخدمون من الاستفادة من أحدث تحسينات الأداء). ثانيهما أن مهمة بناء الشبكة منفصلة إلى حد كبير عن مهمة تشغيلها. كلا هذين الافتراضين غير صحيح تمامًا.

من الواضح أن تصميم الشبكة يتطور، وقد وثّقنا هذه التغييرات مع كل طبعة جديدة من الكتاب على مر السنين. كان القيام بذلك على مدى زمني مُقاس بالسنوات كافيًا تاريخيًا، لكن أي شخص حمّل واستخدم أحدث تطبيقات الهواتف الذكية يدرك مدى البطء الشديد لأي شيء مُقاس بالسنوات وفقًا لمعايير اليوم. يجب أن يكون التصميم للتطور جزءًا من عملية اتخاذ القرار.

فيما يتعلق بالنقطة الثانية، غالبًا ما تكون الشركات التي تبني الشبكات هي نفسها التي تُشغّلها. تُعرف هذه الشركات مجتمعةً باسم مُشغّلي الشبكات، وتشمل الشركات المذكورة أعلاه. ولكن إذا نظرنا مجددًا إلى السحابة بحثًا عن الإلهام، فسنرى أن مبدأ التطوير والتشغيل develop-and-operate لا يقتصر على مستوى الشركة فحسب، بل يشمل أيضًا كيفية تنظيم شركات السحابة الأسرع نموًا لفرقها الهندسية: استنادًا إلى نموذج DevOps. (إذا لم تكن على دراية بـ DevOps، فننصحك بقراءة كتاب "هندسة موثوقية الموقع Site Reliability Engineering": كيف تُشغّل Google "أنظمة الإنتاج" لمعرفة كيفية تطبيقه).

ما يعنيه كل هذا هو أن شبكات الحاسوب تمر الآن بمرحلة تحول كبير، حيث يسعى مشغلو الشبكات إلى تسريع وتيرة الابتكار (ما يُعرف أحيانًا بسرعة الميزات feature velocity) مع الاستمرار في تقديم خدمة موثوقة (مع الحفاظ على الاستقرار). ويحققون ذلك بشكل متزايد من خلال تبني أفضل ممارسات best practices مزودي الخدمات السحابية، والتي يمكن تلخيصها في محورين رئيسيين: (1) الاستفادة من الأجهزة التقليدية ونقل جميع المعلومات الذكية إلى البرمجيات، و(2) اعتماد عمليات هندسية مرنة تكسر الحواجز بين التطوير والعمليات.

يُطلق على هذا التحول أحيانًا اسم "التحويل السحابي" أو "البرمجة البرمجية" "softwarization" or "cloudification" للشبكة، ولكن يُعرف أيضًا باسم الشبكات المُعرّفة برمجيًا Software Defined Networks (SDN). أيًا كان الاسم، فإن هذا النهج الجديد يُحدث نقلة نوعية، ليس فقط من حيث كيفية معالجة التحديات التقنية الأساسية المتمثلة في التأطير والتوجيه والتجزئة/إعادة التجميع fragmentation/reassembly وجدولة الحزم packet scheduling والتحكم في الازدحام والأمان congestion and security control، وما إلى ذلك، بل أيضًا من حيث سرعة تطور الشبكة لدعم الميزات الجديدة.

هذا التحول بالغ الأهمية، لذا سنتناوله مجددًا في قسم "المنظور الأوسع" في نهاية كل فصل. وكما ستتناول هذه المناقشات، فإن ما يحدث في قطاع الشبكات يتعلق جزئيًا بالتكنولوجيا، ولكنه يتعلق أيضًا بالعديد من العوامل غير التقنية الأخرى، وهذا شهادة على مدى رسوخ الإنترنت في حياتنا.

المنظور الأوسع

لمواصلة القراءة عن تحويل الإنترنت إلى إنترنت سحابي، راجع [كتاب] السباق إلى الحافة Race to the Edge.

لمعرفة المزيد عن DevOps، نوصي بما يلي:

- *Site Reliability Engineering: How Google Runs Production Systems, 2016.*