طبقة النقـــل

The Transport Layer

محتويات الفصل:

- مقدمة وخدمات طبقة النقل
- التجميع والتوزيع
- بروتوكول النقل اللاتوصيلي: UDP
- أساسيات النقل الموثوق للبيانات
- بروتوكول النقل التوصيلي: TCP
- مبادئ التحكم في الازدحام
- التحكم في الازدحام في بروتوكول TCP
- الخلاصة

نظراً لموقعها المتوسط بين طبقة التطبيقات وبقية طبقات الشبكة، تمثل طبقة النقل جزءاً أساسياً من البنية الطبقية للشبكة، حيث تلعب الدور الحاسم المتمثل في توفير خدمات الاتصال مباشرةً للتطبيقات التي يجري تشغيلها على المضيفات. تتلخص الفلسفة التعليمية التي سنتبعها في هذا الفصل في التناوب ما بين مناقشة مبادئ طبقة النقل ومناقشة طرق تطبيق تلك المبادئ في البروتوكولات الحالية، مع التركيز كالمعتاد على بروتوكولات الإنترنت، وبشكلٍ خاص بروتوكولي طبقة النقل: TCP وUDP.

سنبدأ بمناقشة العلاقة بين طبقة النقل وطبقة الشبكة، لتمهيد الطريق لفحص الوظيفة الهامة الأولى لطبقة النقل – ألا وهي تمديد خدمة طبقة الشبكة (للتوصيل بين نظامين طرفيين) إلى خدمة توصيل بين عمليتين في طبقة التطبيقات يجري تشغيلهما على النظامين الطرفيين. سنوضح هذه الوظيفة في تغطيتنا لبروتوكول وحدة بيانات المُستخدِم ((User Datagram Protocol (UDP) وهو بروتوكول نقل غير توصيلي على الإنترنت.

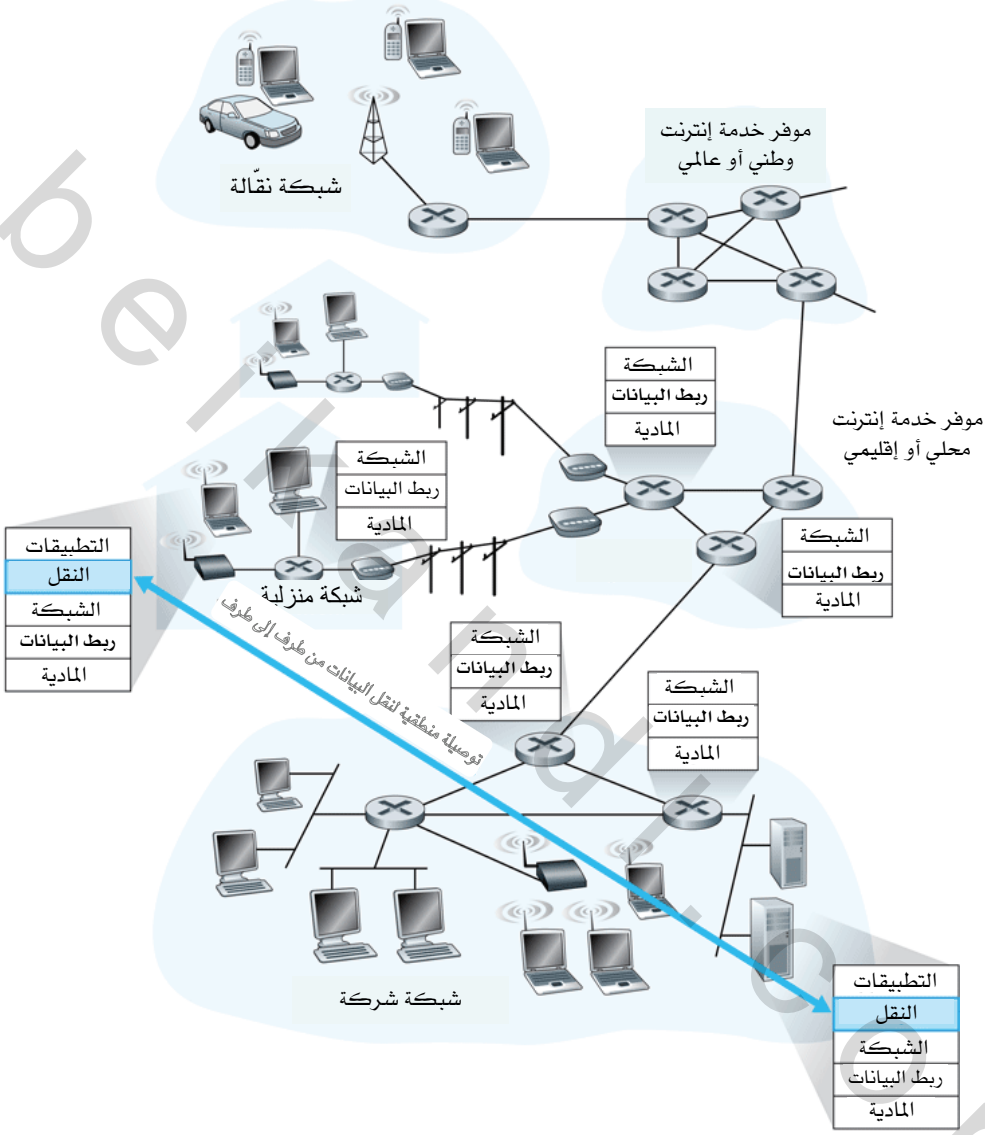
سنعود بعد ذلك إلى المبادئ ونواجه واحدةً من أهم المشاكل الأساسية في شبكات الحاسب: كيف يمكن لكيانين أن يتّصلا بشكلٍ موثوق عبر وسط قد يفقد أو يغيّر في البيانات؟ من خلال سلسلة سيناريوهات تزداد تعقيداً (وواقعية!) سنطور مجموعة من الأساليب التي تستخدمها بروتوكولات النقل لحل هذه المشكلة، وسنبين بعد ذلك كيف تتجسد هذه المبادئ في بروتوكول التحكم في الإرسال ((Transmission Control Protocol (TCP) وهو بروتوكول نقل توصيلي على الإنترنت.

بعد ذلك سننتقل إلى المشكلة الأساسية الثانية في ربط الشبكات، ألا وهي التحكم في معدلات الإرسال في كيانات طبقة النقل لكي تتفادى الازدحام أو تتعافى من حدوثه في الشبكة، وسنتناول أسباب ونتائج الازدحام ونستعرض تقنيات التحكم في الازدحام المستعملة بكثرة. وبعد فهم الأمور المتعلقة بالتحكم في الازدحام فهماً جيداً، سندرس الطريقة التي يتبعها بروتوكول TCP لتحقيق ذلك.

3-1 مقدمة وخدمات طبقة النقل

في الفصلين السابقين أشرنا إلى دور طبقة النقل والخدمات التي توفرها. دعنا نراجع بسرعة ما تعلّمناه عن طبقة النقل. يوفر بروتوكول طبقة النقل اتصالاً منطقياً (logical communication) بين عمليات التطبيقات التي يجري تشغيلها على مضيفات مختلفة. نعني بالاتصال المنطقي أنه من وجهة نظر التطبيق تبدو المضيفات التي تُشغّل تلك العمليات كما لو كانت موصلةً مباشرة؛ في حين أنه في الواقع يمكن أن تقع المضيفات مادياً على جانبين متقابلين من المعمورة موصلةً عبر العديد من الموجّهات وبواسطة تشكيلة كبيرة من أنواع الوصلات. تستخدم عمليات طبقة التطبيقات الاتصال المنطقي الذي توفره طبقة النقل لإرسال الرسائل بين بعضها البعض، دون الاهتمام بتفاصيل البنية التحتية المادية المستخدمة لحمل تلك الرسائل. يوضح الشكل 3-1 مفهوم الاتصال المنطقي.

يتضح من هذا الشكل أن بروتوكولات طبقة النقل موجودة في الأنظمة الطرفية وليس في موجّهات الشبكة. تقوم طبقة النقل على الطرف المُرسِل بتحويل الرسائل التي تستلمها من عملية التطبيق إلى رزم طبقة النقل، والتي تُعرف في مصطلحات الإنترنت بقطع طبقة النقل (segments). يُحتمَل أن يتم ذلك بتجزئة رسائل التطبيق إلى أجزاء أصغر وإضافة ترويسة طبقة النقل إلى كل جزء منها لتكوين قطعة طبقة النقل. تقوم طبقة النقل بعد ذلك بدفع القطعة إلى طبقة الشبكة في نظام طرف الإرسال، حيث يتم تغليف القطعة للحصول على رزمة طبقة الشبكة (وحدة بيانات (datagram)) وإرسالها إلى وجهتها. من المهم ملاحظة أن موجّهات الشبكة تعمل فقط على حقول طبقة الشبكة في وحدة البيانات؛ أي أن تلك الموجّهات لا تفحص حقول قطعة طبقة النقل المغلّفة ضمن وحدة البيانات. وفي جانب الاستلام تنتزع طبقة الشبكة قطعة طبقة النقل من وحدة بيانات طبقة الشبكة وتدفع بالقطعة لأعلى إلى طبقة النقل. تقوم طبقة النقل بعد ذلك بمعالجة القطعة المُستلَمة، وتجعل البيانات في تلك القطعة متاحةً لاستخدام التطبيق في جهة الاستلام.



الشكل 3-1 توفر طبقة النقل توصيلة منطقية وليست مادية لنقل البيانات بين عمليات التطبيقات على نظامين طرفيين.

قد يتوفر أكثر من بروتوكول لطبقة النقل للاستخدام من قِبَل تطبيقات الشبكة. على سبيل المثال للإنترنت بروتوكولان: TCP وUDP. يوفر كلٌّ من هذين البروتوكولين مجموعةً مختلفةً من خدمات طبقة النقل للتطبيق الذي يستخدمه.

1-1-3 العلاقة بين طبقة النقل وطبقة الشبكة

تذكّر أن طبقة النقل توجد مباشرة فوق طبقة الشبكة في رصة البروتوكولات. وبينما يوفر بروتوكول طبقة النقل اتصالاً منطقياً بين عمليات (processes) تجري على مضيفات مختلفة، فإن بروتوكول طبقة الشبكة يوفر اتصالاً منطقياً بين المضيفات (hosts)، وهذا الفرق دقيق ولكنه مهم. دعنا نشرح هذا الفرق بالاستعانة بمثال.

تصور أن هناك بيتين، واحد على الساحل الشرقي والآخر على الساحل الغربي للولايات المتحدة، وكل بيت يأوي 12 طفلاً، والأطفال الذين في الساحل الشرقي أبناء عم الأطفال الذين في الساحل الغربي، وكلٌّ منهم يحب الكتابة إلى كل أبناء عمومته – حيث يكتب كل طفل إلى كل ابن أو بنت منهم خطاباً كل أسبوع، وكل خطاب يُرسَل في ظرف مستقل عن طريق خدمة البريد العمومية. وبهذا يُرسِل كل بيت 144 رسالة إلى البيت الآخر كل أسبوع، (بوسع هؤلاء الأطفال توفير الكثير من المال إذا استخدموا البريد الإلكتروني!). في كلٍّ من البيتين يتولى طفل واحد مسؤولية جمع البريد وتوزيعه: آن (Ann) في بيت الساحل الغربي وبيل (Bill) في بيت الساحل الشرقي. في كل أسبوع تقوم آن بزيارة كل إخوتها وأخواتها لجمع البريد ثم تسلمه إلى ساعي البريد التابع لخدمة البريد العمومية، والذي يقوم بزيارة البيت يومياً. عندما تصل رسائل إلى بيت الساحل الغربي، تتحمل آن أيضاً مسؤولية توزيع البريد الواصل على إخوتها وأخواتها. وفي المقابل يضطلع بيل بمهمات مماثلة على الساحل الشرقي.

في هذا المثال تزوّد خدمة البريد اتصالاً منطقياً بين البيتين – حيث تنقل تلك الخدمة الرسائل من بيت إلى بيت آخر، وليس من شخص إلى شخص آخر. من ناحية أخرى توفر آن وبيل اتصالاً منطقياً بين أبناء العم – حيث يقومان باستلام البريد من

وإلى إخوتهما وأخواتهما وتوزيعه عليهم. لاحظ أنه من منظور أبناء العم يُعتبر آن وبيل بمثابة خدمة البريد، رغم أنهما في الواقع مجرد جزء فقط (الجزء الموجود على النظام الطرفي) من عملية نقل البريد من طرف إلى طرف. هذا المثال التوضيحي يناظر العلاقة بين طبقة النقل وطبقة الشبكة:

- رسائل التطبيقات = الخطابات في الظروف
- العمليات = أبناء العمومة في البيتين
- المضيفات (الأنظمة الطرفية) = البيتان
- بروتوكول طبقة النقل = آن وبيل
- بروتوكول طبقة الشبكة = الخدمة البريدية (بما في ذلك سُعاة البريد)

استمراراً مع هذا التناظر، لاحظ أن آن وبيل يقومان بكل عملهما كلٌّ في حدود بيته فقط؛ فليس لهما أي صلةٍ مثلاً بتصنيف البريد في أيٍّ من مراكز البريد أو بنقل البريد من مركز بريد إلى آخر. بنفس الطريقة فإن بروتوكولات طبقة النقل تكمن في الأنظمة الطرفية. في النظام الطرفي يقوم نظام بروتوكول النقل بنقل الرسائل من عمليات التطبيقات إلى حافة الشبكة (أي طبقة الشبكة) والعكس بالعكس، لكنه ليس له أي رأي في كيفية نقل الرسائل ضمن قلب الشبكة. في الحقيقة كما يوضح الشكل 3-1، لا تتصرّف الموجّهات الوسيطة بناءً على أي معلومات قد تكون طبقة النقل قد أضافتها إلى رسائل التطبيقات، كما أنها لا تتعرف على تلك المعلومات.

واستمراراً مع القصة السابقة، لنفترض الآن أنه عندما تسافر آن وبيل في إجازة، فإن اثنين آخرين من أبناء العم (مثلاً سوزان وهارفي) يحلان محلهما في توفير الخدمة الداخلية لجمع وتوزيع البريد للمجموعة في كل عائلة، لكن لسوء حظ العائلتين، قد لا يقومان بجمع وتوزيع البريد بالضبط بنفس طريقة آن وبيل لكونهما أصغر عمراً. فقد تقوم سوزان وهارفي بجمع وتوزيع البريد مراتٍ أقل، كما أنهما قد يفقدان الرسائل من حين لآخر. وهكذا، فإن ابني العم سوزان وهارفي لا يوفران نفس مجموعة الخدمات (أي نفس نموذج الخدمة) كآن وبيل. وبالمثل فإن شبكة

الحاسب قد يتوافر لديها عدة بروتوكولات للنقل، كلٌّ منها يوفر نموذج خدمة مختلف للتطبيقات التي تستخدمه.

واضح أن الخدمات التي يمكن لآن وبيل توفيرها محدودة بتلك التي يمكن للخدمة البريدية توفيرها. على سبيل المثال إذا كانت الخدمة البريدية لا تضمن حداً أقصى للمدة التي يستغرقها نقل البريد بين البيتين (ثلاثة أيام مثلاً) فلن يكون بوسع آن وبيل ضمان حدٍّ أقصى للتأخير في توصيل البريد بين أيٍّ من أبناء العمومة. وبالمثل فإن الخدمات التي يوفرها بروتوكول طبقة النقل غالباً ما تكون محدودة بنموذج الخدمة الذي يوفره بروتوكول طبقة الشبكة التحتي. إذا كان بروتوكول طبقة الشبكة لا يستطيع توفير ضمانات فيما يتعلق بالتأخير (delay) أو الحيّز الترددي (bandwidth) (ومن ثم معدّل أو سعة الإرسال) المتاح لقطع البيانات الخاصة بطبقة التطبيقات أثناء انتقالها بين المضيفات، فإن نظام طبقة النقل لن يستطيع توفير ضمانات عن التأخير أو الحيّز الترددي المتاح لرسائل التطبيقات التي يجري تبادلها بين العمليات.

ومع ذلك يمكن توفير بعض الخدمات من قِبَل بروتوكول النقل حتى عندما يكون بروتوكول الشبكة التحتي لا يوفر الخدمة المناظرة في طبقة الشبكة. على سبيل المثال، وكما سنرى في هذا الفصل، يمكن لبروتوكول النقل توفير خدمة موثوقة لنقل البيانات لتطبيقٍ ما حتى لو كان بروتوكول الشبكة التحتي غير موثوق، أي حتى لو كان بروتوكول الشبكة يفقد أو يُغير أو يكرر الرزم. كمثالٍ آخر (والذي سندرسه بتفصيل أكثر في الفصل الثامن أثناء تناولنا لأمن الشبكة)، يمكن لبروتوكول النقل استخدام التشفير لضمان عدم اطلاع الدخلاء على رسائل التطبيقات، حتى لو كانت طبقة الشبكة لا تستطيع ضمان سريّة قطع البيانات الخاصة بطبقة النقل.

2-1-3 فكرة عامة عن طبقة النقل في الإنترنت

تذكّر أن الإنترنت – وبشكلٍ عام شبكات TCP/IP – توفر بروتوكولين مختلفين في طبقة النقل لاستخدامات طبقة التطبيقات: أحدهما هو UDP

(بروتوكول وحدة بيانات المستخدم) والذي يوفر للتطبيق الذي يستخدمه خدمةً لاتوصيلية غير موثوقة، والآخر هو TCP (بروتوكول التحكم في الإرسال) والذي يوفر للتطبيق الذي يستخدمه خدمة نقل موثوق تعتمد على توصيلة. عند تصميم تطبيق للاستخدام على شبكة، يتعين على مطور التطبيق تحديد أيٍّ من هذين البروتوكولين سيستخدمه في طبقة النقل. وكما رأينا في الأجزاء 2-7 و2-8 يختار مطور التطبيقات ما بين UDP وTCP عند إنشاء المقابس.

لتبسيط المصطلحات ولتقليل التشويش والخلط (في كتاب تمهيدي كهذا عن شبكات الحاسب) سنشير إلى رزمة بيانات طبقة النقل كقطعة (segment) سواءً كان بروتوكول طبقة النقل TCP أو UDP. ومع ذلك فجدير بالذكر أن أدبيات الإنترنت (على سبيل المثال طلبات التعليقات RFCs) تسمّي رزمة طبقة النقل قطعة (segment) في حالة بروتوكول TCP فقط، بينما تستخدم التعبير "وحدة البيانات" (datagram) في حالة بروتوكول UDP. غير أن أدبيات الإنترنت نفسها تستخدم "وحدة البيانات" للتعبير أيضاً عن رزمة طبقة الشبكة! لكننا سنقصر استخدام التعبير "وحدة البيانات" (datagram) على رزمة بيانات طبقة الشبكة فقط.

قبل المضي قدماً في مقدمتنا القصيرة عن البروتوكولين TCP وUDP، من المفيد أن نذكر بضع كلمات عن طبقة الشبكة بالإنترنت (والتي سندرسها بالتفصيل في الفصل الرابع). يطلق على بروتوكول طبقة الشبكة بالإنترنت اسم بروتوكول الإنترنت (IP). يوفر هذا البروتوكول اتصالاً منطقياً بين المضيفات. نموذج الخدمة لبروتوكول IP هو نموذج "أفضل جهد" للتوصيل. بمعنى أن البروتوكول سيبذل "جهده الأقصى" لتوصيل قطع البيانات بين المضيفات المتصلة، ولكنه لا يقدم أي ضمانات بهذا الصدد. وبشكلٍ خاص فإنه لا يضمن وصول قطع البيانات، ولا يضمن توصيل القطع بالترتيب، ولا يضمن سلامة البيانات في تلك القطع من الأخطاء. لهذه الأسباب يقال: إن بروتوكول IP يوفر خدمة غير موثوقة (unreliable). من المفيد أيضاً أن نذكر هنا أن لكل مضيف عنواناً واحداً على الأقل من عناوين طبقة الشبكة، وهو ما يعرف بعنوان IP (سندرس عناوين IP بالتفصيل في الفصل الرابع).

بعد هذه اللمحة عن نموذج خدمة بروتوكول IP، دعنا نلخص الآن نماذج الخدمة التي يوفرها البروتوكولان UDP وTCP. إن المسؤولية الأساسية الأولى لهذين البروتوكولين هي تمديد خدمة التوصيل التي يوفرها بروتوكول IP بين نظامين طرفيين إلى خدمة توصيل بين عمليتين يجري تشغيلهما على النظامين الطرفيين. يُطلق على تمديد خدمة التوصيل من مضيف إلى مضيف إلى خدمة توصيل من عملية إلى عملية: التجميع (multiplexing) والتوزيع (demultiplexing) في طبقة النقل. سنتناول التجميع والتوزيع في طبقة النقل في الجزء التالي من هذا الفصل. يقوم كلٌّ من البروتوكولين UDP وTCP بتدقيق سلامة البيانات أيضاً عن طريق حقول خاصة باكتشاف الخطأ في الترويسة التي تُلحَق بقطعة البيانات.

تُعدّ خدمتا الحد الأدنى هاتان (نقل البيانات بين العمليات واكتشاف الخطأ) الخدمات الوحيدة التي يوفرها بروتوكول UDP. وبالتحديد يلاحظ أن بروتوكول UDP – مثله في ذلك مثل بروتوكول IP – يوفر خدمة غير موثوقة؛ فهو لا يضمن أن البيانات المُرسَلة من قِبَل عمليةٍ ما ستصل سليمة (أو ستصل على الإطلاق!) إلى العملية المقصودة على مضيف الوجهة. سنتناول بروتوكول UDP بالتفصيل في الجزء 3-3.

على الناحية الأخرى يوفر بروتوكول TCP عدة خدمات إضافية للتطبيقات. فهو يوفر – أولاً وقبل كل شيء – نقلاً موثوقاً للبيانات من خلال استعمال أساليب لضبط تدفق البيانات، والأرقام المتسلسلة للرزم، وأرقام إشعارات الاستلام، والموقّتات (وهي أساليب سنتناولها بالتفصيل في هذا الفصل). كما أنه يضمن توصيل البيانات المُرسَلة من عملية ما إلى العملية المقصودة على مضيف الوجهة صحيحة وغير مكرّرة وبنفس الترتيب. وهكذا يحوّل بروتوكول TCP خدمة IP غير الموثوقة بين الأنظمة الطرفية إلى خدمة موثوقة لنقل البيانات بين العمليات. كما يوفر بروتوكول TCP أيضاً تحكماً في الازدحام، وهذا الإجراء لا يعتبر خدمةً للتطبيق الذي يستخدم البروتوكول فقط بقدر ما هو خدمة للإنترنت ككل (أي خدمة للصالح العام). فبشكلٍ عام يمنع تحكم TCP في الازدحام أي توصيلة TCP من إغراق الوصلات والموجّهات بين المضيفات المتصلة بكمية مفرطة من حركة

مرور البيانات. ويسعى بروتوكول TCP لإعطاء كل توصيلة TCP تعبر وصلة مزدحمة نصيباً متساوياً من الحيّز الترددي الكلي المتاح للوصلة. يتم ذلك بتنظيم المعدلات التي تستخدمها جهات الإرسال التابعة لتوصيلات TCP في إرسال البيانات إلى الشبكة. ومن ناحية أخرى يُلاحَظ أن بروتوكول UDP لا ينظم حركة مرور بياناته، فالتطبيق الذي يستخدمه بوسعه استخدام أي معدل إرسال يرغبه وللمدة التي يرغبها.

أي بروتوكول يوفر نقلاً موثوقاً للبيانات وسيطرةً على الازدحام سيكون معقداً بالضرورة. سنحتاج إلى عدة أجزاء لتغطية مبادئ نقل البيانات الموثوق والتحكم في الازدحام، وأجزاء إضافية لتغطية بروتوكول TCP نفسه. سنتناول هذه المواضيع في الأجزاء من 3-4 إلى 3-8. وتتلخص طريقتنا في عرض هذه الموضوعات في هذا الفصل في التناوب مابين المبادئ الأساسية وتفاصيل البروتوكول نفسه. على سبيل المثال، سنناقش أولاً النقل الموثوق للبيانات بشكلٍ عام، ثم ننتقل لنوضح كيف يحقق بروتوكول TCP على وجه التحديد نقلاً موثوقاً للبيانات. وبنفس الطريقة سنناقش التحكم في الازدحام بشكلٍ عام أولاً ثم بعد ذلك نناقش كيف يتم التحكم في الازدحام في بروتوكول TCP. ولكن قبل التعرض لكل تلك المادة الجيدة، دعنا أولاً نلقي نظرة على خدمتي التجميع والتوزيع في طبقة النقل.

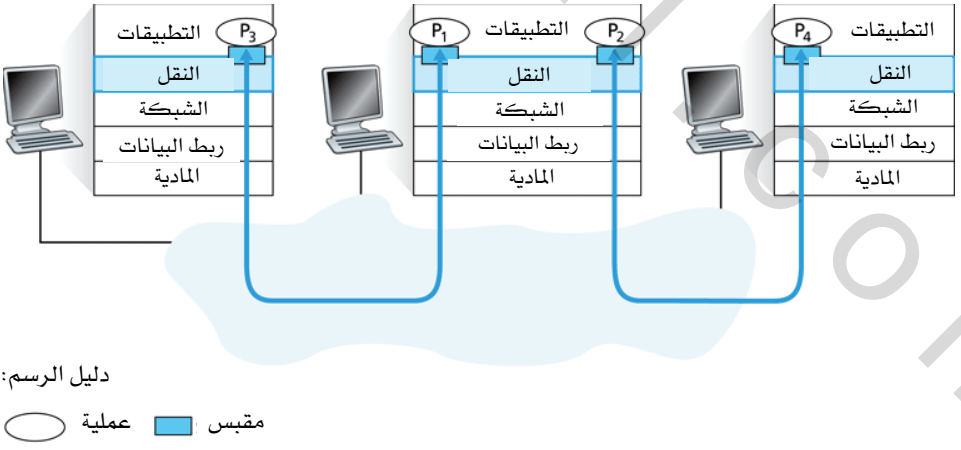
3-2 التجميع والتوزيع

نتناول في هذا الجزء خدمتي التجميع والتوزيع في طبقة النقل، أي تمديد خدمة التوصيل من مضيف إلى مضيف التي توفرها طبقة الشبكة إلى خدمة توصيل من عملية إلى عملية والتي توفرها طبقة النقل للتطبيقات التي يجري تشغيلها على المضيفات. وللحفاظ على التحديد في تناول الموضوع، سنناقش هذه الخدمة الأساسية لطبقة النقل ضمن سياق الإنترنت، إلا أننا نؤكد على أن التجميع والتوزيع خدمتان مطلوبتان لكل شبكات الحاسب.

في مضيف الوجهة: تتسلم طبقة النقل قطع البيانات من طبقة الشبكة التي تقع تحتها مباشرةً، وتتحمل مسؤولية توصيل البيانات الموجودة في تلك القطع إلى

عمليات التطبيقات الملائمة التي يجري تشغيلها في المضيف. لنلقِ نظرة على مثال. افترض أنك جالس أمام حاسبك تتجول بين صفحات الويب بينما تقوم في نفس الوقت بتشغيل عملية أخرى لتنزيل ملفات عبر بروتوكول FTP وتشغيل عمليتين أُخرَيَيْن للدخول على حاسب عن بعد عبر بروتوكول Telnet. لديك إذن أربع عمليات شغالة من تطبيقات الشبكة – عمليتا Telnet، وعملية FTP، وعملية HTTP. عندما تتسلم طبقة النقل في حاسبك البيانات من طبقة الشبكة تحتها، فإنها تحتاج لتوجيه تلك البيانات إلى إحدى تلك العمليات الأربع. دعنا الآن نرى كيف يتم ذلك.

لعلك تذكر من الأجزاء 2-7 و2-8 أن العملية (كجزء من تطبيق الشبكة) يمكن أن يكون لها مقبسٌ أو أكثر، أي بوابات تمر عبرها البيانات من الشبكة إلى العملية ومن العملية إلى الشبكة. وبالتالي كما يوضح الشكل 2-3 فإن طبقة النقل في مضيف الاستقبال لا تسلّم البيانات في الحقيقة مباشرة إلى العملية، ولكن بدلاً من ذلك تسلمها إلى مقبس متوسط. ونظراً لأنه في أي وقت يمكن أن يكون هناك أكثر من مقبس واحد في مضيف الاستقبال، فإن كل مقبس له مُعرّف (identifier) يميزه. تعتمد صيغة المُعرّف على ما إذا كان المقبس مقبس UDP أو TCP، كما سنرى بعد قليل.



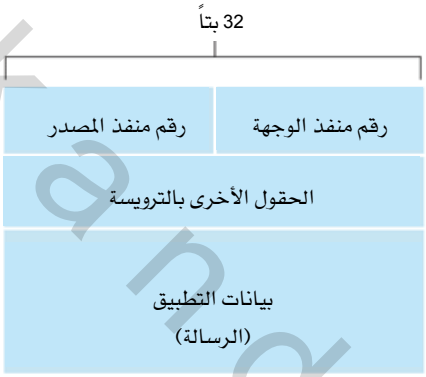
الشكل 2-3 عمليتا التجميع والتوزيع في طبقة النقل.

لنتأمل الآن كيف يقوم مضيف استقبال بتوجيه قطعة بيانات وصلت لطبقة النقل إلى المقبس الملائم. تتضمن قطعة بيانات طبقة النقل عدة حقول لهذا الغرض. في طرف الاستقبال تفحص طبقة النقل تلك الحقول لتحدد مقبس الاستلام ومن ثم توجّه القطعة إلى ذلك المقبس. يطلق على وظيفة توصيل البيانات الموجودة في قطعة طبقة النقل إلى المقبس الصحيح اسم "التوزيع" (demultiplexing). وبالمثل يطلق اسم "التجميع" (multiplexing) على وظيفة تجميع أجزاء البيانات في مضيف المصدر من المقابس المختلفة وتغليف كل جزء من البيانات بمعلومات الترويسة (التي ستستعمل لاحقاً في عملية التوزيع) لتكوين قطع بيانات طبقة النقل ثم دفع القطع إلى طبقة الشبكة. لاحظ أن طبقة النقل في المضيف المتوسّط في الشكل 3-2 يجب أن توزع قطع البيانات الواصلة من طبقة الشبكة تحتها لأي من العمليتين P1 أو P2 فوقها. يتم ذلك بتوجيه قطع البيانات الواصلة إلى مقبس العملية المناظرة. يجب على طبقة النقل في المضيف المتوسّط أيضاً تجميع البيانات الخارجة من تلك المقابس، وتكوين قطع بيانات طبقة النقل، ودفع تلك القطع لأسفل إلى طبقة الشبكة. ورغم أننا قدّمنا خدمتي التجميع والتوزيع في سياق بروتوكول النقل على الإنترنت، فإنه من المهم إدراك أنهما مطلوبتان طالما كان هناك بروتوكول واحد في طبقةٍ ما (طبقة النقل أو غيرها) يتم استخدامه من قِبَل عدة بروتوكولات في الطبقة الأعلى مباشرةً.

لتوضيح وظيفة التوزيع تذكّر مثال البيتين في الجزء السابق. يتم تمييز كل طفل أو طفلة من الأطفال باسمه أو اسمها. عندما يستلم بيل رزمة بريد من ساعي البريد، يقوم بعملية "توزيع" بملاحظة أسماء إخوته وأخواته المكتوبة على الرسائل الواصلة وبعد ذلك يسلّم كل رسالة إلى صاحبها. تقوم آن بعملية "تجميع" من خلال أخذها الرسائل من إخوتها وأخواتها وتعطي البريد المجمّع إلى ساعي البريد.

الآن وقد فهمنا الدور المناط بعمليتي التجميع والتوزيع في طبقة النقل، دعنا نرى كيف يتم ذلك في واقع الأمر في مضيف على الشبكة. من المناقشة أعلاه يتبين أن التجميع في طبقة النقل يتطلّب: (1) أن يكون لكل مقبس معرّف يميزه، و(2) أن تتضمن كل قطعة بيانات حقولاً خاصة تشير إلى المقبس الذي ستسلّم القطعة له. هذه الحقول الخاصة والمبينة في الشكل 3-3 هي حقل رقم منفذ المصدر وحقل رقم

منفذ الوجهة (لقطع بيانات بروتوكولات TCP وUDP حقول أخرى أيضاً، كما سنرى في الأجزاء اللاحقة من هذا الفصل). يُمثّل رقم المنفذ بعدد يتكون من 16 بتاً، أي يتراوح من 0 إلى 65535. تخصص أرقام المنافذ من 0 إلى 1023 لاستخدام بروتوكولات التطبيقات المشهورة مثل HTTP (الذي يستخدم رقم المنفذ 80) وFTP (الذي يستخدم رقم المنفذ 20 و21). توجد قائمة بأرقام تلك المنافذ في RFC 1700 ومحدّثة في RFC 3232. عندما نطوّر تطبيقاً جديداً (كأحد التطبيقات التي طُوّرت في الأجزاء 2-7 و2-8)، يجب أن نخصص رقم منفذ للتطبيق.



الشكل 3-3 حقول أرقام المنافذ على كلٍّ من المصدر والوجهة بقطعة بيانات طبقة النقل.

لعله من الواضح الآن كيف تقوم طبقة النقل بإنجاز خدمة التوزيع (demultiplexing). يمكن تخصيص رقم منفذ لكل مقبس في المضيف، وعندما تصل قطعة بيانات إلى المضيف تقوم طبقة النقل بفحص رقم منفذ الوجهة في القطعة وتوجّه القطعة إلى المقبس المناظر، ومن ثم تمر بيانات القطعة من خلال المقبس إلى العملية المناظرة. كما سنرى، تلك هي الطريقة التي يتبعها بروتوكول UDP بشكلٍ أساسي. ومع ذلك فسنرى أيضاً أن التجميع والتوزيع في بروتوكول TCP هو أدق وألطف من ذلك.

التجميع والتوزيع اللاتوصيلي

تذكّر من الجزء 2-8 أنه بوسع برنامج جافا يتم تشغيله على مضيف أن ينشىء مقبساً باستخدام السطر التالي:

```
DatagramSocket mySocket = new DatagramSocket();
```

عند إنشاء مقبس UDP بهذه الطريقة، تقوم طبقة النقل تلقائياً بتخصيص رقم منفذ للمقبس. وبالتحديد تخصص طبقة النقل رقم منفذ في المدى من 1024 إلى 65535 يكون غير مستخدم حالياً من قِبَل أي منفذ UDP آخر على المضيف. وكطريقة بديلة يمكن لبرنامج جافا إنشاء مقبس باستخدام السطر التالي:

```
DatagramSocket mySocket = new DatagramSocket(19157);
```

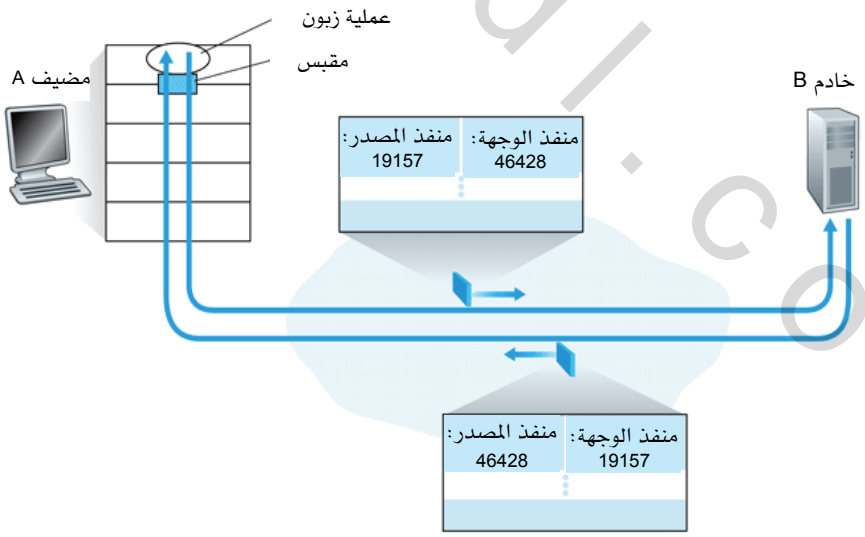
في هذه الحالة يخصص التطبيق رقم منفذ معين هو 19157 لمقبس الـ UDP. إذا كان مطور التطبيق الذي يكتب البرنامج ينجز جانب الخادم من بروتوكول مشهور، فعليه حينئذٍ تخصيص رقم المنفذ المناظر. عادةً ما يترك جانب الزبون من التطبيق لطبقة النقل القيام آليا (وبشفّافية) بتخصيص رقم المنفذ، بينما يقوم جانب الخادم من التطبيق بتخصيص رقم منفذ معين.

بعد تخصيص أرقام منافذ لمقابس UDP، يمكننا الآن وصف خدمتي التجميع والتوزيع بالضبط. افترض أن عملية في المضيف A تستخدم منفذ UDP رقم 19157 وتريد إرسال جزء من بيانات التطبيق إلى عملية تستخدم منفذ UDP رقم 46428 في المضيف B. تقوم طبقة النقل في المضيف A بتكوين قطعة طبقة نقل تتضمن بيانات التطبيق ورقم منفذ المصدر (19157) ورقم منفذ الوجهة (46428) وقيمتين أُخرَيَيْن (سنتناولهما لاحقاً، ولكنهما غير مهمّين للمناقشة الحالية). تقوم طبقة النقل بعد ذلك بدفع القطعة الناتجة إلى طبقة الشبكة. تغلف طبقة الشبكة القطعة في وحدة بيانات IP وتبذل أفضل جهد لتوصيل القطعة إلى مضيف الاستقبال. إذا وصلت القطعة إلى مضيف الاستقبال B، فإن طبقة النقل في المضيف تفحص رقم منفذ الوجهة في القطعة أي (46428) وتسلّم القطعة إلى المقبس المميّز بذلك الرقم.

لاحظ أن المضيف يمكن أن يكون عليه عدة عمليات تعمل في نفس الوقت، كل عملية لها مقبس UDP خاص بها ورقم منفذ مناظر. عند وصول قطع UDP من الشبكة يقوم المضيف B بتوجيه (توزيع) كل قطعة إلى المقبس المناظر بفحص رقم منفذ الوجهة الموجود على القطعة.

من المهم ملاحظة أن مقبس UDP يتم تمييزه بشكلٍ كامل بواسطة عنوان يضم عنوان IP للوجهة ورقم منفذ الوجهة، وكنتيجة لذلك إذا كان لقطعتي UDP عنوانا IP مختلفان للمصدر و/أو رقما منفذ مختلفان للمصدر، لكن لهما نفس عنوان IP ورقم المنفذ للوجهة، فإن القطعتين ستوجّهان إلى نفس عملية الوجهة النهائية عن طريق نفس مقبس تلك الوجهة.

قد تتساءل الآن وما فائدة رقم منفذ المصدر إذن؟ كما هو موضح في الشكل 3–4، في القطعة المتجهة من A إلى B يُستعمَل رقم منفذ المصدر كجزء من "عنوان العودة" – عندما يريد B إرسال قطعة إلى A، يأخذ منفذ الوجهة في القطعة المتجهة من B إلى A قيمته من قيمة منفذ المصدر في القطعة الواصلة من A إلى B (عنوان العودة الكامل هو عنوان IP للمضيف A ورقم منفذ المصدر).



الشكل 3-4 انعكاس أرقام منافذ المصدر والوجهة.

كمثال، تذكّر برنامج خادم UDP الذي تناولناه في الجزء 2-8. يستخدم الخادم في برنامج UDPServer.java طريقة لانتزاع رقم منفذ المصدر من القطعة التي يستلمها من الزبون، ثم يرسل قطعة جديدة إلى الزبون برقم منفذ المصدر المنتزع كرقم منفذ الوجهة في تلك القطعة الجديدة.

التجميع والتوزيع التوصيلي

لفهم وظيفة التوزيع (demultiplexing) في بروتوكول TCP يجدر بنا إلقاء نظرة فاحصة على مقابس TCP وطريقة إنشاء توصيلات TCP. هناك فرق دقيق بين مقبس UDP ومقبس TCP يكمن في أن مقبس TCP يتم تمييزه بعنوان رباعي: (عنوان IP للمصدر، ورقم منفذ المصدر، وعنوان IP للوجهة، ورقم منفذ الوجهة). وعليه فعندما تصل قطعة TCP من الشبكة إلى مضيف، يستعمل المضيف جميع تلك القيم الأربع لتوجيه (توزيع) القطعة إلى المقبس الملائم. بشكلٍ خاص وبالمقارنة مع UDP، فإنه عند وصول قطعتي TCP بعنواني IP مختلفين للمصدر أو رقمي منفذ مختلفين للمصدر سيوجّهان إلى مقبسين مختلفين (باستثناء قطعة TCP الأولى التي تحمل طلب إنشاء التوصيلة). لإلقاء مزيد من الضوء، دعنا نعيد النظر في مثال برمجة زبون/خادم TCP الذي استعرضناه في الجزء 2-7:

- لتطبيق الخادم في بروتوكول TCP "مقبس ترحيب" ينتظر من خلاله طلبات عمل التوصيلات من زبائن TCP على منفذ رقم 6789 (انظر الشكل 2-31)
- يقوم زبون TCP بتكوين قطعة إنشاء توصيلة (SYN) باستخدام السطر:

```
Socket clientSocket = new Socket ("serverHostName", 6789);
```

- طلب إنشاء توصيلة ماهو إلا قطعة TCP برقم منفذ وجهة = 6789 وفيها البت الخاص بإنشاء توصيلة في ترويسة TCP (سنتناولها في الجزء 3-5) له القيمة 1. تتضمن القطعة أيضاً رقم منفذ مصدر، والذي تم اختياره من قِبَل الزبون. يقوم السطر أعلاه أيضاً بإنشاء مقبس TCP لعملية الزبون حيث يمكن للبيانات أن تدخل وتغادر عملية الزبون من خلاله.

- عندما يستقبل نظام تشغيل الحاسب المضيف الذي يقوم بتشغيل عملية الخادم قطعة طلب إنشاء توصيلة والتي تحمل منفذ الوجهة رقم 6789، فإنه يقوم بتحديد عملية الخادم التي تنتظر قبول طلبات التوصيلات على منفذ رقم 6789. تقوم عملية الخادم بعد ذلك بإنشاء مقبس جديد:

```
Socket connectionSocket = welcomeSocket.accept();
```

- أيضاً تلاحظ طبقة النقل في الخادم القيم الأربع التالية في قطعة طلب التوصيلة: (1) رقم منفذ المصدر في القطعة، (2) عنوان IP لمضيف المصدر، (3) رقم منفذ الوجهة في القطعة، و(4) عنوان IP للمضيف الخاص بها. يتم تمييز مقبس التوصيلة الذي تم إنشاؤه حديثاً بتلك القيم الأربع؛ وكل القطع التي تصل بعد ذلك بنفس (منفذ المصدر، وعنوان IP للمصدر، ومنفذ الوجهة، وعنوان IP للوجهة) سيتم توزيعها إلى ذلك المقبس. الآن وقد تم تجهيز توصيلة TCP، يمكن للزبون والخادم أن يتبادلا إرسال البيانات بين بعضهما.

يمكن للمضيف الخادم دعم العديد من مقابس TCP في نفس الوقت، حيث يرتبط كل مقبس بعملية، ويُميّز كل مقبس بعنوانه الرباعي الخاص به. عندما تصل قطعة TCP إلى المضيف، تستعمل حقول العنوان الأربعة كلها (عنوان IP للمصدر، ومنفذ المصدر، وعنوان IP للوجهة، ومنفذ الوجهة) لتوجيه (توزيع) القطعة إلى المقبس الملائم.

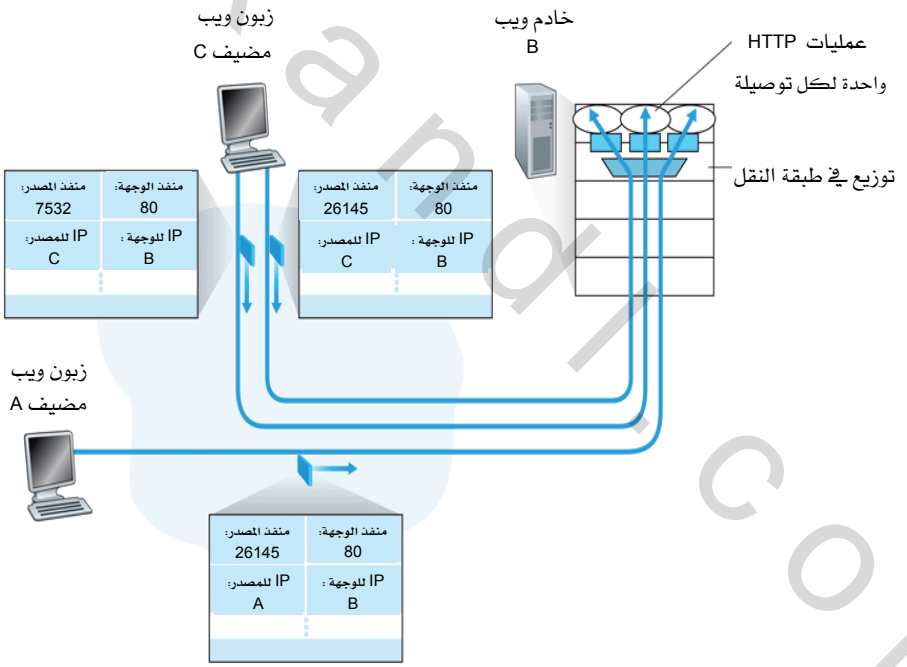
نبذة عن الأمن (Focus on Security)

مسح المنافذ:

رأينا أن عملية الخادم تنتظر بشغف ظهور منفذ مفتوح للاتصال من قِبَل زبون عن بعد. تُحجز بعض المنافذ للتطبيقات المشهورة (كخادمات الويب، وFTP، وDNS، وSMTP)، بينما تستخدم المنافذ الأخرى عادةً من قِبَل التطبيقات الأخرى المنتشرة (مثلاً: يُنصت خادم ميكروسوفت لقواعد البيانات SQL للطلبات على منفذ UDP رقم 1434). وهكذا إذا وجدنا منفذاً مفتوحاً على مضيف، يكون بوسعنا ربط ذلك المنفذ بتطبيق معين يجري تشغيله على المضيف. إن هذا مفيدٌ جداً لمدراء الأنظمة الذين غالباً ما يهتمون بمعرفة تطبيقات الشبكة المستخدمة على المضيفات في شبكاتهم. لكن المهاجمين – من منطلق "فحص المبنى قبل السطو عليه" – يريدون أيضاً معرفة المنافذ المفتوحة على المضيفات المستهدفة. فإذا وُجد مضيفٌ يشغّل تطبيقاً فيه نقطة ضعف أمنية معروفة، فإن المضيف يكون في ذلك الوقت جاهزاً للهجوم عليه (مثلاً يكون خادم SQL الذي يُنصت على منفذ 1434 عرضةً لفيض المخزن المؤقت buffer overflow، مما يسمح لمستخدم عن بعد بتشغيل أي برنامج اعتباطي على المضيف الضعيف، وهي نفس نقطة الضعف التي استغلتها دودة Slammer [CERT 2003-04]).

إن تحديد أي التطبيقات تُنصت على أي المنافذ في مضيف بعينه هو أمرٌ سهلٌ نسبياً، ففي الواقع هناك عدد من البرامج المتاحة للجميع على الإنترنت للقيام بذلك يطلق عليها اسم "ماسحات المنافذ" (port scanners)، ولعل أوسع تلك البرامج انتشاراً هو nmap، وهو متوفر مجاناً على الموقع http://insecure.org/nmap ومتضمن في معظم إصدارات نظام التشغيل لاينكس. يقوم برنامج nmap بمسح منافذ TCP بشكلٍ متسلسل بحثاً عن المنافذ التي تقبل توصيلات TCP. كما يقوم البرنامج بمسح منافذ UDP بشكلٍ متسلسل بحثاً عن المنافذ التي تستجيب لقطع UDP المُرسَلة. يعطي nmap في كلتا الحالتين قائمة تبين المنافذ المفتوحة والمغلقة وتلك التي تعذّر الوصول إليها. بوسع أي مضيف يقوم بتشغيل nmap محاولة مسح المنافذ على أي مضيف مستهدف في أي مكان على الإنترنت. سنزور برنامج nmap مرة أخرى في الجزء 3-5-6 عندما نناقش إدارة توصيلات بروتوكول TCP.

يبين الشكل 3-5 هذا الوضع، حيث يبدأ المضيف C جلستَي HTTP مع الخادم B، بينما يبدأ مضيف A جلسة HTTP واحدة مع الخادم B. لكلٍ من المضيفين A وC والخادم B عنوان IP الخاص به؛ لنفرض أن تلك العناوين هي A وC وB على التوالي. يخصص مضيف C رقمي منفذ مصدر مختلفين (26145 و7532) لتوصيلتي HTTP الخاصة به، ونظراً لأن المضيف A يختار أرقام منافذ المصدر بشكلٍ مستقل عن المضيف C، فربّما يخصّص رقم 26145 لمنفذ المصدر لتوصيلة HTTP الخاصة به، ولكن هذا لا يمثل مشكلة، فلا يزال بوسع الخادم B توزيع التوصيلتين بنفس رقم منفذ المصدر بشكلٍ صحيح، نظراً لأن التوصيلتين لهما عنوانا IP مختلفان للمصدر.



الشكل 3-5 زبونان يستخدمان نفس رقم منفذ الوجهة (80) للاتصال بنفس التطبيق على خادم ويب.

خادمات الويب وبروتوكول TCP

قبل أن ننتهي من هذه المناقشة، من المفيد ذكر بعض الملاحظات الإضافية حول خادمات الويب، والكيفية التي تستخدم بها أرقام المنافذ. خذ في الاعتبار مضيفاً يقوم بتشغيل خادم ويب Apache على منفذ 80. عندما يرسل الزبائن (مثلاً متصفّحات الويب) قطع البيانات إلى الخادم، سيكون لكل القطع الواصلة نفس منفذ الوجهة 80. على وجه الخصوص سيكون لكلٍّ من القطع الأولية الخاصة بإنشاء التوصيلة والقطع التي تحمل رسائل طلب HTTP نفس منفذ الوجهة 80. كما وضحنا أعلاه، يميّز الخادم القطع الواصلة من الزبائن المختلفة باستخدام عناوين IP للمصدر مع أرقام منافذ المصدر.

يبين الشكل 3-5 خادم ويب ينشىء عملية جديدة لكل توصيلة، ولكلٍ من هذه العمليات مقبس توصيلة خاص بها تستقبل من خلاله طلبات HTTP وترسل استجابات HTTP. ونذكر هنا أنه ليس هناك دائماً تناظر واحد لواحد بين كلٍّ من مقابس التوصيلات والعمليات، فغالباً ما تستخدم خادمات الويب الحديثة ذات مستوى الأداء العالي عملية واحدة فقط وتنشئ عملية فرعية بسيطة (thread) جديدة مع مقبس توصيلة جديد لكل طلب توصيلة جديد من الزبون. وإذا قمت بالإجابة على سؤال البرمجة الأول في الفصل الثاني، فإنك تكون قد طورت خادم ويب يؤدي ذلك بالضبط. قد يكون لمثل هذا الخادم في أي وقت من الأوقات العديد من مقابس التوصيلات المرتبطة بنفس العملية (بمعرّفات مختلفة).

إذا كان الزبون والخادم يستخدمان بروتوكول HTTP الدائم، فإنه طوال فترة التوصيلة الدائمة يتبادل الخادم والزبون رسائل HTTP عن طريق نفس مقبس الخادم. في حين إذا استخدم الخادم والزبون بروتوكول HTTP غير الدائم، فإن توصيلة TCP جديدة تُنشأ وتُغلق لكل طلب واستجابة، ويمكن أن يؤثر هذا الإنشاء والإغلاق المتكرّر للمقابس كثيراً على أداء خادم ويب مشغول (ورغم ذلك يوجد عدد من حيل نظام التشغيل التي يمكن استخدامها لتخفيف حدة تلك

المشكلة). وننصح القراء المهتمين بقضايا نظام التشغيل المتعلقة ببروتوكولات HTTP الدائمة وغير الدائمة بالاطلاع على [Nielsen 1997; Nahum 2002].

الآن وبعد أن انتهينا من مناقشة التجميع والتوزيع بطبقة النقل، ننتقل لمناقشة أحد بروتوكولات النقل على الإنترنت، ألا وهو UDP. وكما سنرى في الجزء التالي فإن بروتوكول UDP لا يضيف على بروتوكول طبقة الشبكة أكثر من خدمتي التجميع والتوزيع.

3-3 بروتوكول النقل اللاتوصيلي: UDP

سنلقي في هذا الجزء نظرةً فاحصةً على بروتوكول UDP، وكيف يعمل وماذا يعمل. وننصحك بمراجعة الجزء 2-1 الذي يتضمن نظرة عامة على نموذج خدمة UDP، والجزء 2-8 الذي يعالج برمجة المقابس باستخدام UDP.

لتحفيزك لمناقشتنا التالية حول UDP، افترض أنك مهتم بتصميم بروتوكول نقل بسيط يوفر الحد الأدنى من المتطلبات بلا رتوش. كيف ستبدأ في القيام بعملٍ كهذا؟ قد تفكر في البداية في استخدام بروتوكول نقل لا يضيف شيئاً سوى أخذ الرسائل من عملية التطبيق ودفعها مباشرة إلى طبقة الشبكة على جانب الإرسال؛ في حين على جانب الاستقبال، يرفع الرسائل التي تصله من طبقة الشبكة مباشرة إلى عملية التطبيق. غير أننا – كما تعلّمنا في الجزء السابق – يجب أن نفعل أكثر قليلاً من لا شيء!، فعلى أقل تقدير يجب أن توفر طبقة النقل خدمة تجميع وتوزيع لكي تنقل البيانات بين طبقة الشبكة والعملية الملائمة في طبقة التطبيقات.

يقوم بروتوكول النقل UDP والمعرّف في RFC 768 تقريباً بالحد الأدنى فقط المطلوب من بروتوكول نقل. باستثناء وظيفتي التجميع والتوزيع وبعض التدقيق الخفيف لاكتشاف الأخطاء، لا يضيف UDP شيئاً يُذكر إلى بروتوكول IP. في الحقيقة إذا اختار مطور التطبيقات البروتوكول UDP بدلاً من TCP، فإن التطبيق سيتعامل مباشرةً تقريباً مع بروتوكول IP. يأخذ UDP الرسائل من عملية التطبيق، ويضيف حقلين لرقم منفذ المصدر ورقم منفذ الوجهة، كما يضيف حقلين صغيرين

آخرين، ويدفع بالقطعة الناتجة إلى طبقة الشبكة. تغلّف طبقة الشبكة قطعة طبقة النقل للحصول على وحدة بيانات IP، ثم بعد ذلك تبذل الطبقة أفضل جهد لتوصيل القطعة إلى مضيف الاستقبال. إذا وصلت القطعة إلى مضيف الاستقبال، يستعمل UDP رقم منفذ الوجهة لتوصيل بيانات القطعة إلى عملية التطبيق الملائمة. لاحظ أنه مع بروتوكول UDP ليس هناك إجراءات مصافحة (handshaking) بين كيانات طبقتي النقل المُرسَلة والمُستقبِلة قبل إرسال قطعة. ولهذا السبب يعرف بروتوكول UDP بأنه بروتوكول غير توصيلي (connectionless).

يُعتبر بروتوكول DNS (نظام أسماء النطاقات) مثالاً لبروتوكولات طبقة التطبيقات التي تستخدم UDP عادةً. فعندما يريد DNS في مضيف عمل استفسار، فإنه يُنشىء رسالة استفسار DNS ويدفع بها إلى UDP. ويضيف UDP على جانب المضيف حقول الترويسة إلى الرسالة ويدفع بالقطعة الناتجة إلى طبقة الشبكة وذلك بدون أي إجراءات مصافحة مع كيان UDP الذي يجري تشغيله على النظام الطرفي في الوجهة. تغلّف طبقة الشبكة قطعة UDP لتكوّن وحدة بيانات طبقة الشبكة (datagram) وترسلها إلى خادم أسماء النطاقات. ينتظر DNS في المضيف المستفسر إجابةً على استفساره. إذا لم يتلق إجابة (ربما لأن الشبكة التحتية فقدت الاستفسار أو الإجابة)، فإما أن يحاول إرسال الاستفسار إلى خادم آخر للأسماء، أو يخبر التطبيق الذي يستخدمه بأنّه لا يستطيع الحصول على إجابة.

قد تتساءل الآن: وإذا كان الأمر كذلك فلماذا يختار مطور للتطبيقات تطوير تطبيقه على بروتوكول UDP بدلاً من بروتوكول TCP؟ أليس بروتوكول TCP مفضلاً على الدوام، حيث إنه يوفر خدمة موثوقة لنقل البيانات على العكس من UDP؟ الجواب لا، فالعديد من التطبيقات يلائمها بروتوكول UDP أكثر للأسباب التالية:

- التحكم الأدق من قبل التطبيق في البيانات التي تُرسَل ومتى تُرسَل تبعاً لبروتوكول UDP. فبمجرد أن تمرر عملية التطبيق البيانات إلى UDP، يقوم البروتوكول بوضع البيانات داخل قطعة UDP ودفعها على الفور إلى طبقة الشبكة. وعلى العكس من ذلك، يتضمن بروتوكول TCP آلية للتحكم

في الازدحام تخنق مُرسِل طبقة النقل عندما تصبح واحدة أو أكثر من الوصلات بين مضيفي المصدر والوجهة النهائية مزدحمة بشكلٍ مُفرط. من ناحية أخرى يواصل بروتوكول TCP إعادة إرسال قطعة البيانات المرة تلو الأخرى إلى أن يصله إشعار استلام (acknowledgment) من الوجهة بوصول القطعة، بغض النظر عن الوقت الذي يستغرقه التوصيل الموثوق للقطعة. إلا أن التطبيقات الفورية غالباً ما تتطلب ضمان معدل أدنى لإرسال البيانات، ولا تحبذ تأخير إرسال قطع البيانات كثيراً، حيث يمكنها أن تتحمل بعض الفقد في البيانات، لذا فإن نموذج الخدمة الذي يوفره بروتوكول TCP لا يلائم حاجة تلك التطبيقات بشكلٍ جيد كما سنبين لاحقاً، ويمكن لهذه التطبيقات استخدام بروتوكول UDP وتطوير أي وظائف إضافية مطلوبة علاوة على خدمة UDP كجزء من التطبيق.

- عدم الحاجة لإنشاء توصيلة: كما سنرى لاحقاً يستخدم بروتوكول TCP آلية ثلاثية للمصافحة قبل البدء في نقل البيانات، لكن على العكس من ذلك ينطلق بروتوكول UDP في نقل البيانات على الفور دون أي تمهيدات رسمية، ولذا فإن UDP لا يعاني من أي تأخير لإنشاء توصيلة. لعل هذا هو السبب الرئيس الذي بسببه يستخدم DNS بروتوكول UDP وليس TCP. إن DNS سيكون أبطأ بكثير إذا استخدم TCP. في المقابل يستخدم HTTP بروتوكول TCP بدلاً من UDP لأن الموثوقية شيء مهم فيما يتعلق بصفحات الويب التي تتضمن نصوصاً. لكن – كما استعرضنا سريعاً في الجزء 2-2 – يُعتبر التأخير بسبب إنشاء توصيلة في HTTP من عوامل التأخير الأساسية عند تنزيل صفحات الويب.
- عدم الاكتراث بحالة التوصيلة: يحتفظ بروتوكول TCP بـ"حالة التوصيلة" في الأنظمة الطرفية. تتضمن حالة التوصيلة هذه المخازن المؤقتة للإرسال والاستقبال، ومتغيرات التحكم في الازدحام، ومتغيرات الأرقام المتسلسلة، وأرقام إشعارات الاستلام. سنرى في الجزء 3–5 أن معلومات حالة التوصيلة هذه لازمة لتحقيق خدمة نقل موثوقة للبيانات مع التحكم في الازدحام في

بروتوكول TCP. في المقابل لا يحتفظ UDP بحالة توصيلة ولا يتتبع أياً من تلك المتغيرات، لذا فإن خادماً مخصصاً لتطبيق معين يمكنه عادةً دعم عدد أكبر بكثير من الزبائن عندما يستخدم التطبيق بروتوكول UDP بدلاً من TCP.

- تقليل الأعباء الإضافية بسبب ترويسة قطعة البيانات: لكل قطعة بيانات في بروتوكول TCP ترويسة تتكون من 20 بايتاً، في حين أن العبء الإضافي في حالة بروتوكول UDP يقتصر على 8 بايتات فقط.

يبين الشكل 3-6 تطبيقات الإنترنت المعروفة وبروتوكولات النقل التي تستعملها. كما نتوقّع فإن تطبيقات البريد الإلكتروني، والدخول على الحاسبات عن بعد، وتصفح الويب، ونقل الملفات، كلها تستخدم بروتوكول TCP – فكل تلك التطبيقات تحتاج لخدمة النقل الموثوق التي يوفرها TCP. ومع ذلك فالعديد من التطبيقات المهمة تستخدم UDP بدلاً من TCP. يُستخدم UDP على سبيل المثال لتحديث جداول التوجيه من قبل بروتوكول RIP (انظر الجزء 4-6-1)، وذلك نظراً لأن تحديثات RIP تتم بشكلٍ دوري (كلّ خمس دقائق عادةً)، وبالتالي فإن التحديثات التي تفقد في الطريق ستستبدل بتحديثات تالية، مما يجعل التحديثات المفقودة المنتهي تاريخها عديمة الفائدة. كما يُستخدم UDP أيضاً لتشغيل تطبيقات إدارة الشبكة من خلال بروتوكول SNMP (انظر الفصل التاسع). ويرجع سبب تفضيل UDP على TCP في هذه الحالة إلى أن تطبيقات إدارة الشبكة يجب تشغيلها عادةً والشبكة في حالة مجهَدة ومزدحمة، وهو بالضبط الوقت الذي يصعب فيه تحقيق نقل موثوق للبيانات. كما ذكرنا أعلاه، يستخدم UDP أيضاً من قِبَل بروتوكول DNS، وبذلك يتفادى الأخير تأخيرات إنشاء التوصيلات فيما لو استخدم TCP.

التطبيق	بروتوكول طبقة التطبيقات	بروتوكول طبقة النقل التحتي
البريد الإلكتروني	SMTP	TCP
للدخول على الحاسبات عن بعد	Telnet	TCP
الويب	HTTP	TCP
نقل الملفّات	FTP	TCP
نقل الملفّات عن بعد	NFS	عادةً UDP
عرض مواد الوسائط المتعددة	مملوكة	TCP أوUDP
هاتف الإنترنت	مملوكة	TCP أوUDP
هاتف الإنترنت	SNMP	عادةً UDP
بروتوكول التوجيه	RIP	عادةً UDP
تحويل أسماء النطاقات	DNS	عادةً UDP

الشكل 3-6 تطبيقات الإنترنت المشهورة وبروتوكولات طبقة النقل التحتية المستخدمة معها.

كما يظهر من الشكل 3-6، يُستخدَم كلٌّ من البروتوكولين UDP وTCP حالياً من قِبَل تطبيقات الوسائط المتعددة، كهاتف الإنترنت، والمؤتمرات الفورية عبر الفيديو، وتشغيل تسجيلات الصوت والفيديو المخزّنة. سنلقي نظرةً فاحصةً على هذه التطبيقات في الفصل السابع. فقط نذكر هنا أن كل تلك التطبيقات يمكن أن تتحمّل قدراً قليلاً من فقد الرزم، ولذا فإن النقل الموثوق للبيانات ليس ضرورياً جداً لنجاح التطبيق. علاوة على ذلك فإن التطبيقات الفورية، كهاتف الانترنت والمؤتمرات عبر الفيديو، تستجيب بشكلٍ سيئ جداً لوسائل التحكم في الازدحام التي يفرضها بروتوكول TCP. لهذه الأسباب قد يختار مطوّرو تطبيقات الوسائط المتعددة تشغيل تطبيقاتهم على UDP بدلاً من TCP. ومع ذلك، فإن TCP يُستخدم الآن على نحو متزايد لنقل مواد عرض الوسائط المتعددة. على سبيل المثال، وجد [Sripanidkulchai 2004] أن حوالى 75٪ من تطبيقات الفيديو للعرض الحي والعرض حسب الطلب تستخدم TCP. عندما تكون نسب فقد الرزم المسموح بها منخفضة، وعندما تحجب شبكات بعض الهيئات حركة مرور البيانات التي تستخدم

بروتوكول UDP لأسباب أمنية (انظر الفصل الثامن)، يصبح TCP بديلاً لا عوض عنه لنقل مادة عرض الوسائط المتعددة.

رغم أن استخدام تطبيقات الوسائط المتعددة لبروتوكول UDP منتشر اليوم، إلا أن الأمر لايزال محل خلاف. كما ذكرنا أعلاه لا يتضمن بروتوكول UDP تحكماً في الازدحام، ولكن التحكم في الازدحام مطلوب لمنع الشبكة من دخول حالة اختناق يتعذر فيها القيام بأي عمل مفيد. إذا استخدم كل شخص العرض المستمر لأفلام الفيديو بمعدل عالٍ لإرسال البيانات بدون أي تحكم في الازدحام، فسيغمر فيض كبير من الرزم موجّهات الشبكة بحيث يتمكن عدد قليل جداً من رزم UDP من قطع المسار من المصدر إلى الوجهة النهائية بنجاح. وعلاوةً على ذلك فإن نسب الفقد العالية للرزم بسبب عدم التحكم في معدلات الإرسال من قِبَل مُرسِلي UDP ستؤدي بمُرسِلي TCP (والذين، كما سنرى، يخفضون معدلات إرسالهم لمواجهة الازدحام) إلى تقليل معدلاتهم بشكلٍ كبير. وهكذا فإن عدم التحكم في الازدحام في UDP يمكن أن تؤدي إلى نسب فقد عالية بين مُرسِلي ومستقبلي UDP، بالإضافة إلى ازدحام جلسات TCP؛ وتلك مشكلة خطيرة فعلاً [Floyd 1999]. اقترح العديد من الباحثين آليات جديدة لإلزام كل مصادر البيانات، بما في ذلك مصادر UDP، بالقيام بتحكم يتواءم مع الازدحام [Mahdavi 1997; Floyd 2000; Kohler 2006; RFC 4340].

قبل مناقشة صيغة قطعة بيانات بروتوكول UDP، نذكر هنا أنه يمكن لتطبيقٍ ما تحقيق نقلٍ موثوقٍ للبيانات مع استعمال بروتوكول UDP، وذلك ببناء تلك الموثوقية في التطبيق نفسه (على سبيل المثال بإضافة إشعارات الاستلام وآليات إعادة الإرسال كالتي سندرسها في الجزء القادم)، غير أن هذه مهمّة ليست بالبسيطة، ويمكنها أن تشغل مطور التطبيقات بتنقيح وتصحيح برامجه لفترة طويلة. ومع ذلك فبناء الموثوقية مباشرةً في التطبيق يسمح للتطبيق بـ"الاحتفاظ بكعكته وأكلها أيضاً"، بمعنى أن عمليات التطبيقات يمكن أن تتصل بشكلٍ موثوق بدون أن تخضع للقيود على معدلات الإرسال التي تفرضها آليات التحكم في الازدحام ضمن بروتوكول TCP.

1-3-3 صيغة قطعة بيانات UDP

تم توصيف صيغة قطعة بيانات بروتوكول UDP في RFC 768 كما هو مبين في الشكل 3-7. تحتل بيانات التطبيق حقل البيانات (الحمل الآجر) في قطعة UDP. على سبيل المثال في حالة DNS يتضمن حقل البيانات إما رسالة استفسار أو رسالة رد. أما في تطبيق عرض صوتي فتملأ عينات من المادة الصوتية حقل البيانات. تتكون ترويسة UDP من أربعة حقول يتألف كلٌّ منها من بايتين. كما ذكرنا في القسم السابق تسمح أرقام المنافذ لمضيف الوجهة بتمرير بيانات التطبيق إلى العملية الملائمة التي يجري تنفيذها على النظام الطرفي للوجهة (أي لأداء وظيفة التوزيع demultiplexing). يستخدم مضيف الاستقبال حقل "المجموع التدقيقي" (checksum) لاكتشاف ما إذا كانت هناك أخطاء قد طرأت على القطعة أثناء انتقالها من مضيف المصدر. وفي الواقع يُحسب "المجموع التدقيقي" أيضاً على عدة حقول في ترويسة IP بالإضافة إلى قطعة UDP. لكنّنا سنهمل هذه التفاصيل لكي نتمكن من "رؤية الغابة من خلال الأشجار"، وسنناقش طريقة حساب المجموع التدقيقي في الجزء التالي، كما سنتناول المبادئ الأساسية لكشف الخطأ في الجزء 2-5. يحدد حقل الطول طول قطعة UDP الكلي بالبايتات بما في ذلك حقل الترويسة.

الشكل 3-7 صيغة قطعة بيانات UDP.

2-3-3 حقل المجموع التدقيقي بقطعة بيانات UDP

يُستخدم حقل المجموع التدقيقي بقطعة بيانات UDP لاكتشاف الأخطاء. بمعنى أنه يُستعمل لمعرفة ما إذا كانت البتات في القطعة قد طرأ عليها أي تغيير أثناء انتقالها من المصدر إلى الوجهة النهائية – على سبيل المثال بسبب الشوشرة في الوصلات أو عند التخزين المؤقت في الموجّهات. يقوم بروتوكول UDP في ناحية المُرسِل بحساب المكمّل للواحد (1's complement) لناتج جمع كل الكلمات (بطول 16 بتاً) الموجودة في القطعة، مع تدوير أي فيض (overflow) يحدث في الخانة الأخيرة وإضافته إلى الخانة الأولى. توضع النتيجة في حقل المجموع التدقيقي لقطعة UDP.

وفيما يلي مثال بسيط لتوضيح حساب المجموع التدقيقي، لكن يمكنك الإطلاع على تفاصيل طرق عالية الكفاءة لإجراء هذه العملية في RFC 1071 ومعلومات عن أدائها على بيانات حقيقية في [Stone 1998; Stone 2000]. افترض أن لدينا الكلمات الثلاث التالية بالقطعة (كل منها يتألف من 16 بتاً):

0110011001100000
0101010101010101
1000111100001100

حاصل جمع أول كلمتين من هذه الكلمات الثلاث هو:

0110011001100000
0101010101010101
1011101110110101

بإضافة الكلمة الثالثة نحصل على:

1011101110110101
1000111100001100
0100101011000001
+1
0100101011000010

لاحظ حدوث فيض في نهاية عملية الجمع الأخيرة، وقد تم تدويره وإضافته للخانة الأولى. نحصل على مكمل الواحد للناتج بتحويل كل 0 إلى 1 وكل 1 إلى 0. وهكذا فإن مكمل الواحد للمجموع 0100101011000010 هو 1011101010011110 1، والذي يوضع في حقل المجموع التدقيقي. في مضيف الوجهة يتم جمع كل الكلمات الأربع بالقطعة (في هذا المثال) بما في ذلك قيمة المجموع التدقيقي. إذا لم تحدث أي أخطاء في بتات القطعة، فمن الواضح أن المجموع سيكون 1111111111111111. أما إذا كان أحد البتات في الناتج له القيمة 0، فسندرك أن خطأ أو أكثر قد طرأ على القطعة أثناء رحلتها من المصدر إلى الوجهة النهائية.

قد تتساءل لماذا يتضمن بروتوكول UDP إمكانيات لاكتشاف الأخطاء أساساً، حيث إن الكثير من بروتوكولات طبقة ربط البيانات (بما في ذلك بروتوكول الإيثرنت الشهير) يوفر تلك الإمكانيات أيضاً. يكمن السبب في أنه ليس هناك ما يضمن أن كل الوصلات بين المصدر والوجهة تقوم بالكشف عن الأخطاء؛ بمعنى أن إحدى الوصلات قد تستخدم بروتوكول طبقة ربط بيانات لا يقوم بذلك. علاوة على ذلك فحتى إذا انتقلت القطع عبر الوصلة بشكلٍ صحيح، فمن المحتمل حدوث أخطاء في القطعة أثناء تخزينها في ذاكرة أحد الموجّهات. فإذا كان كلٌّ من موثوقية نقل البيانات على الوصلات واكتشاف أخطاء التخزين في ذاكرات الموجّهات غير مضمون، فلا غرابة إذن في أن يوفر بروتوكول UDP إمكانية اكتشاف الأخطاء في طبقة النقل – على أساس من طرف إلى طرف – إذا كان ذلك مطلوباً. يعتبر هذا مثالاً للمبدأ المشهور في تصميم الأنظمة "من طرف إلى طرف" [Saltzer 1984]، والذي يقول بأنه لمّا كان من الضروري القيام ببعض المهام على أساس من طرف إلى طرف (كاكتشاف الأخطاء في هذه الحالة) فإن "الوظائف المتعلقة التي تنفّذ في المستويات الأدنى قد تكون زائدة أو ذات قيمة ضئيلة بالمقارنة بكلفة أدائها في المستوى الأعلى".

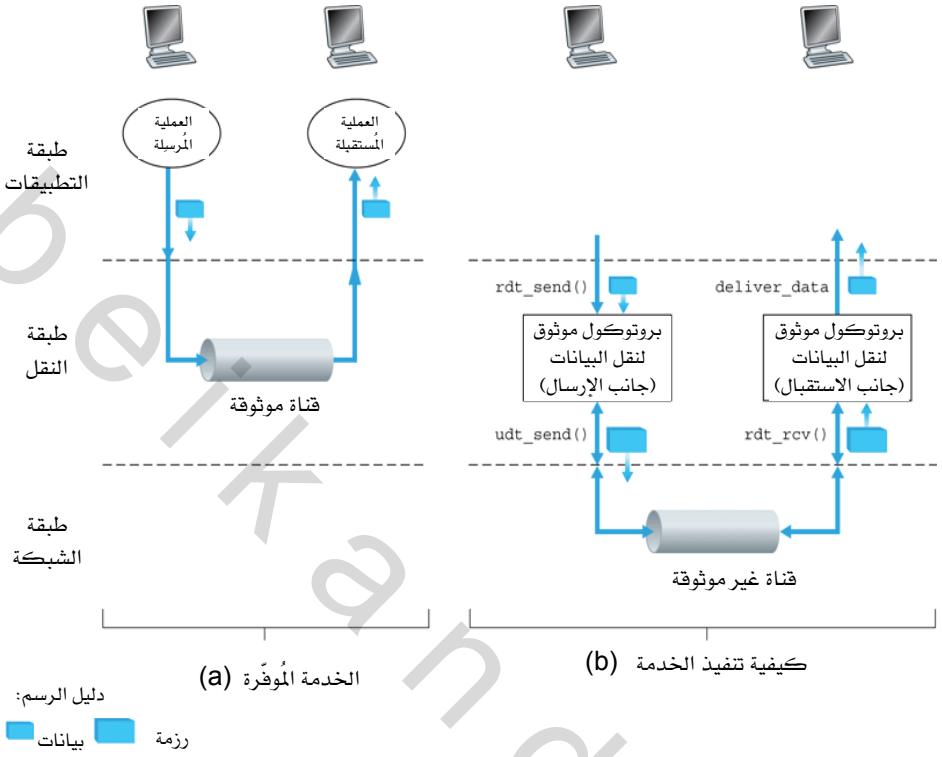
نظراً لأنه يفترض تشغيل IP فوق أي بروتوكول في الطبقة الثانية، فمن المفيد لطبقة النقل توفير إمكانيات اكتشاف الأخطاء كإجراء وقائي. ورغم أن بروتوكول UDP يوفر إمكانية التدقيق لاكتشاف الأخطاء، إلا أنه لا يتخذ أي إجراء لتصحيح أي خطأ يتم اكتشافه، فبعض تطبيقات UDP ببساطة تهمل قطع البيانات التي فيها خطأ، بينما تمرر التطبيقات الأخرى تلك القطع إلى التطبيق مع التنبيه لتلك الأخطاء.

وهكذا نصل إلى نهاية استعراضنا لبروتوكول UDP، وسنرى قريبا أن بروتوكول TCP يوفر لتطبيقاته نقلاً موثوقاً للبيانات بالإضافة إلى الخدمات الأخرى التي لا يوفرها بروتوكول UDP. من الطبيعي أيضاً أن يكون TCP أكثر تعقيداً عن UDP. ولكن قبل أن نتناول بروتوكول TCP بالتفصيل، سيكون من المفيد أن نرجع خطوةً للوراء لنناقش المبادئ الأساسية للنقل الموثوق للبيانات.

4-3 أساسيات النقل الموثوق للبيانات

في هذا الجزء سنتناول النقل الموثوق للبيانات في سياق عام. هذا أمر ملائم، حيث إن تحقيق نقل موثوق للبيانات أمرٌ مطلوب ليس فقط في طبقة النقل، ولكن أيضاً في طبقة ربط البيانات وطبقة التطبيقات، وعليه فإن المشكلة بهذا العموم لها أهميتها الأساسية في مجال الشبكات. في الواقع إذا طُلب من شخص سرد قائمة بأهم عشر مشاكل أساسية في مجال ربط الشبكات، فقد تكون تلك المشكلة مرشحة لشغل المركز الأول. في الجزء القادم سندرس بروتوكول التحكم في الإرسال TCP، وسنرى بشكلٍ خاص أن هذا البروتوكول يستثمر العديد من المبادئ التي نحن بصدد تناولها الآن.

يوضح الشكل 3-8 الإطار العام لدراستنا للنقل الموثوق للبيانات. يتمثل نموذج الخدمة التي يتم توفيرها للطبقات الأعلى في قناةٍ موثوقة يمكن نقل البيانات عبرها. في تلك القناة الموثوقة لا تتعرض بتات البيانات للتغير (0 يتحول إلى 1 أو العكس) ولا تفقد وتصل بنفس الترتيب الذي أُرسلت به. هذا بالضبط هو "نموذج الخدمة" الذي يوفره بروتوكول TCP لتطبيقات الإنترنت التي تستخدمه.



الشكل 3-8 النقل الموثوق للبيانات: (a) نموذج الخدمة، (b) كيفية تنفيذ الخدمة.

تتلخص مسؤولية بروتوكول النقل الموثوق للبيانات في تنفيذ هذا النموذج المذكور للخدمة. مما يجعل هذه المهمة صعبة أن الطبقة تحت بروتوكول النقل الموثوق للبيانات قد تكون غير موثوقة في واقع الأمر. فمثلاً بروتوكول TCP للنقل الموثوق للبيانات يعمل فوق بروتوكول IP في طبقة الشبكة والذي لا يوفر موثوقية للنقل من طرف إلى طرف. وبشكلٍ أكثر عموماً، قد تتألف الطبقة تحت النقطتين الطرفيتين اللتين تتصلان بشكلٍ موثوق من وصلة مادية واحدة فقط (كما في حالة بروتوكول وصلة البيانات)، أو من شبكة عالمية (كما في حالة بروتوكول طبقة النقل). وعلى أية حال يكفي من أجل المناقشة هنا أن نعتبر تلك الطبقة السفلى ببساطة كقناة غير موثوقة من نقطة لنقطة.

في هذا الجزء سنطوّر تدريجياً جوانب الإرسال والاستقبال لبروتوكول نقل موثوق للبيانات، مع الأخذ في الاعتبار نماذج معقدة أكثر فأكثر للقناة التحتية. يبين الشكل 3-8 (b) الواجهات (interfaces) الخاصة ببروتوكولنا للنقل الموثوق للبيانات. سيتم طلب جانب الإرسال في البروتوكول من أعلى عن طريق نداء لـ ()rdt_send والذي يقوم أيضاً بتمرير البيانات المطلوب توصيلها إلى الطبقة الأعلى في جانب الاستقبال. (ترمز rdt هنا لـ reliable data transfer protocol (أي بروتوكول النقل الموثوق للبيانات) بينما تشير send_ إلى أن النداء موجّه لجانب الإرسال من البروتوكول. إنّ الخطوة الأولى في تطوير أي بروتوكول تكمن في اختيار اسم جيد له!). على جانب الاستقبال سيتم النداء لـ ()rdt_rcv عندما تصل رزمة بيانات من جانب الاستلام في القناة. عندما يريد بروتوكول rdt توصيل البيانات إلى الطبقة الأعلى، سيقوم بذلك عن طريق نداء لـ ()deliver_data. سنستخدم هنا "رزمة" بدلاً من "قطعة" الخاصة بطبقة النقل. نظراً لأن النظرية التي نقوم بتطويرها في هذا الجزء تنطبق على شبكات الحاسب بصفة عامة وليس فقط على طبقة النقل في الإنترنت، فإن التعبير العام "رزمة" ربما يكون أكثر ملاءمة هنا.

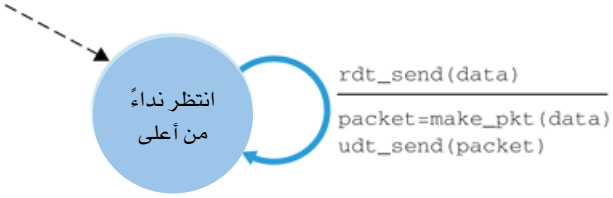
في هذا الجزء سنتناول حالة نقل البيانات في اتجاه واحد فقط: من جانب الإرسال إلى جانب الاستقبال. رغم أن حالة النقل الموثوق للبيانات في اتجاهين (أي بشكلٍ كامل الازدواج full-duplex) ليست أصعب كثيراً من حيث المفهوم إلا أنها أعقد بكثير في الشرح. ورغم أننا سنأخذ في الاعتبار نقل البيانات في اتجاه واحد فقط، إلا أنه من المهم ملاحظة أن جانبي الإرسال والاستقبال في بروتوكولنا سيحتاجان مع ذلك لإرسال رزم في كلا الاتجاهين كما يبين الشكل 3-8. سنرى بعد قليل أن جانبي الإرسال والاستقبال في البروتوكول rdt سيحتاجان إلى تبادل رزم التحكم ذهاباً وإياباً بالإضافة إلى تبادل الرزم التي تحتوي على بيانات مطلوب نقلها. يقوم كلا الجانبين من rdt بإرسال الرزم إلى الجانب الآخر بواسطة نداء لـ udt_send (حيث ترمز udt لنقل بيانات غير موثوق).

3-4-1 بناء بروتوكول للنقل الموثوق للبيانات

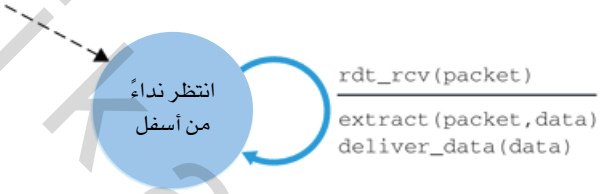
نتدرج الآن عبر سلسلة من البروتوكولات، كلٌّ منها أكثر تعقيداً من سابقه، لنصل في النهاية إلى بروتوكول سليمٍ خالٍ من العيوب للنقل الموثوق للبيانات.

نقل موثوق للبيانات عبر قناة موثوقة تماماً : rdt1.0

سنتناول أولاً الحالة الأسهل التي تكون فيها القناة التحتية موثوقة تماماً. في هذه الحالة يكون البروتوكول نفسه – والذي سنطلق عليه rdt1.0 – بديهياً. يبين الشكل 3-9 تعريفات آلة الأوضاع المحدودة (finite state machine (FSM)) لمُرسِل ومُستقبِل البروتوكول rdt1.0. تعرّف FSM في الشكل 3-9 (a) عملية المُرسِل، بينما تعرّف FSM في الشكل 3-9 (b) عملية المُستقبِل. من المهم ملاحظة أن هناك FSM مستقلة لكلٍ من المُرسِل والمُستقبِل. آلة FSM الخاصة بكلٍ من المُرسِل والمُستقبِل في الشكل 3-9 لها وضع واحد فقط. تشير الأسهم في وصف FSM إلى انتقال البروتوكول من وضع لآخر (نظراً لأن كل FSM في الشكل 3-9 لها وضع واحد فقط، فإن الانتقال يكون بالضرورة من ذلك الوضع إلى نفسه؛ سنرى مخططات حالات أكثر تعقيداً بعد قليل). يُبيّن الحدث الذي يسبّب الانتقال فوق الخط الذي يمثل الانتقال، كما تُبيّن الإجراءات التي تُتخذ عند حدوث الحدث تحت ذلك الخط. عندما لا يُتخذ أي إجراء عند وقوع حدث سنضع الرمز Λ تحت الخط، وعندما لا يقع حدث ولكن يتم اتخاذ إجراء سنضع الرمز Λ فوق الخط وذلك للدلالة بوضوح على عدم وجود إجراء أو حدث. يُبيّن وضع البداية لآلة FSM بسهم متقطع. ورغم أن كلاً من آلات FSM في الشكل 3-9 لها وضع واحد فقط، فإننا سنرى بعد قليل آلات FSM لها عدة أوضاع، لذا سيكون من المهم تمييز وضع البداية لكل آلة FSM.



(a) بروتوكول rdt1.0: جانب الإرسال



(b) بروتوكول rdt1.0: جانب الاستقبال

الشكل 3-9 بروتوكول rdt1.0 لقناة موثوقة تماماً.

يقبل جانب الإرسال من بروتوكول rdt ببساطة البيانات من الطبقة الأعلى عن طريق الحدث rdt_send(data)، ومن ثم يكوّن رزمة تتضمن البيانات (بواسطة الإجراء make_pkt(data)، وبعد ذلك يرسل الرزمة إلى القناة. عملياً ينشأ الحدث rdt_send(data) كنتيجة لنداء إجراء (على سبيل المثال rdt_send()) صادر عن تطبيق في الطبقة الأعلى.

على جانب الاستقبال، يستلم rdt رزمة من القناة التحتية عن طريق الحدث rdt_rcv(packet)، ثم يأخذ البيانات من الرزمة بواسطة الإجراء extract (packet, data)، ومن ثم تعبر البيانات إلى الطبقة الأعلى عن طريق الإجراء deliver_data (data). عملياً ينشأ الحدث rdt_rcv(packet) نتيجة لنداء إجراء (على سبيل المثال rdt_rcv()) من بروتوكول الطبقة السفلى.

في هذا البروتوكول البسيط ليس هناك اختلاف بين وحدة البيانات والرزمة. أيضاً يكون كل تدفق للرزم من المُرسِل إلى المُستقبِل؛ فعند استخدام قناة اتصال موثوقة تماماً لا توجد حاجة لجانب المُستقبِل لتزويد جانب المُرسِل بأي تعقيبات أو تغذية مرتدة، حيث لا يمكن حدوث خطأ! لاحظ أننا افترضنا أيضاً أن المُستقبِل يستطيع استلام البيانات بنفس السرعة التي يرسلها بها المُرسِل. وعليه فليست هناك حاجة للمُستقبِل لأن يطلب من المُرسِل التباطؤ!

نقل موثوق للبيانات عبر قناةٍ بأخطاء في البتات: rdt2.0

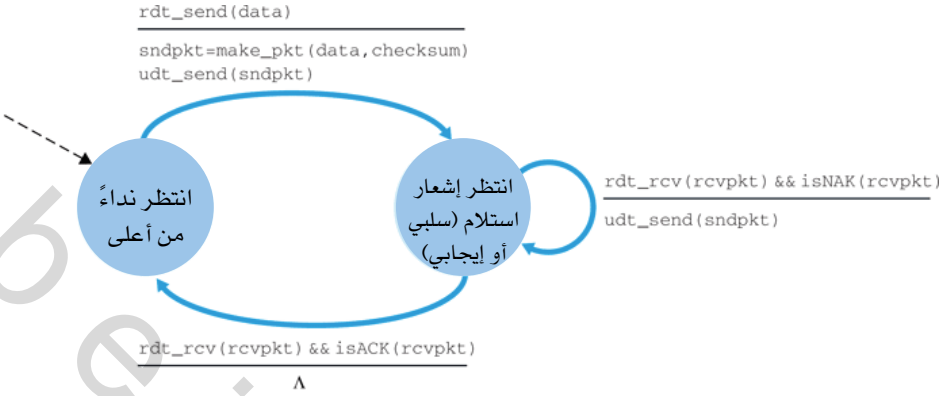
النموذج الأكثر واقعية للقناة التحتية هو ذلك الذي تتعرض فيه بتات الرزمة للخطأ. تحدث تلك الأخطاء في البتات عادةً في المكوّنات المادية للشبكة أثناء إرسال أو نقل الرزمة المُرسَلة، أو أثناء حفظ الرزمة في مخزن مؤقت. سنواصل الافتراض أيضاً بأن كل الرزم المُرسَلة يتم استلامها (رغم أن بِتاتها قد يعتريها الخطأ).

قبل تطوير بروتوكول للاتصال الموثوق عبر مثل تلك القناة، لنأخذ في الاعتبار كيف يمكن أن يتعامل البشر مع هذه الحالة. تأمل كيف تقوم أنت بإملاء رسالة طويلة على الهاتف. في سيناريو معتاد قد يقول آخذ الرسالة على الطرف الآخر من الخط "حسناً" بعد كل جملة يسمعها ويفهمها ويسجلها. إذا سمع آخذ الرسالة جملة مشوشة، فسوف يطلب منك تكرار الجملة المشوشة. يستخدم هذا البروتوكول لإملاء الرسالة الإشعار الإيجابي ("حسناً") والإشعار السلبي ("رجاءً كرّر تلك الجملة"). تسمح رسائل التحكم تلك للمُستقبِل بإعلام المُرسِل عما تم استلامه بشكلٍ صحيح، وما استُلم خطأ ومن ثم يحتاج إلى إعادة إرسال. في حالة شبكات الحاسب، يُطلق على بروتوكولات النقل الموثوق للبيانات التي تعتمد على مثل هذه الإعادة للإرسال بروتوكولات ARQ (طلب إعادة إرسال ذاتي Automatic Repeat Request).

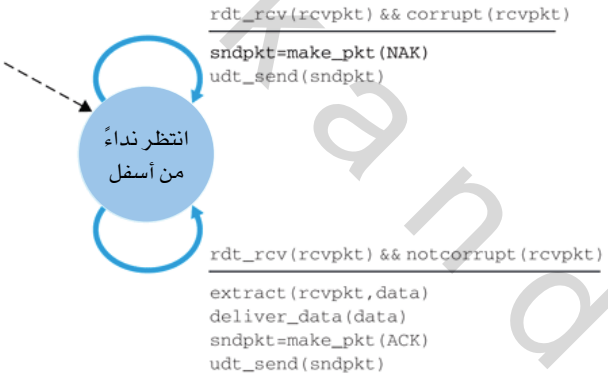
يحتاج الأمر إلى ثلاث إمكانيات إضافية لتتمكن بروتوكولات ARQ من التعامل مع أخطاء البتات:

- اكتشاف الخطأ: نحتاج أولاً إلى آلية تُمكّن المُستقبِل من اكتشاف ما إذا كانت هناك أخطاء قد حدثت في البتات المُرسَلة. تذكّر من الجزء السابق أن بروتوكول UDP يستخدم حقل المجموع التدقيقي (checksum) لهذا الغرض. سنتناول في الفصل الخامس أساليب اكتشاف الأخطاء وتصحيحها بتفصيل أكثر؛ تلك الأساليب التي تسمح للمُستقبِل باكتشاف – وربما إصلاح – أخطاء البتات في الرزمة الواصلة. نحتاج الآن لنعرف فقط أن هذه الأساليب تتطلب إرسال بتات إضافية من المُرسِل إلى المُستقبِل (زيادة على بتات البيانات الأصلية المطلوب نقلها)، والتي تُوضع في حقل المجموع التدقيقي برزمة بيانات rdt2.0.
- التغذية المرتدة من المُستقبِل: لمّا كان المُرسِل والمُستقبِل يعملان عادةً على نظامين طرفيين مختلفين قد يفصل بينهما آلاف الأميال فإن الطريقة الوحيدة لكي يعرف المُرسِل وجهة نظر المُستقبِل عن العالم (في حالتنا هذه، ما إذا كانت الرزمة المُرسَلة قد تم استلامها بشكلٍ صحيح) هي أن يقوم المُستقبِل بتزويد المُرسِل بتغذية مرتدة واضحة. تُعتبر إشعارات الاستلام الإيجابية (ACK) والسلبية (NAK) في سيناريو إملاء الرسالة أمثلة لتلك التغذية المرتدة. بالمثل سيرسل بروتوكولنا rdt2.0 رزم ACK ورزم NAK كإشعارات استلام إيجابية وسلبية، على الترتيب، إلى المُرسِل. من حيث المبدأ تحتاج تلك الرزم إلى بت واحد لنقل تلك المعلومة؛ فعلى سبيل المثال يمكن أن يمثل 0 الإشعار السلبي (NAK) و1 الإشعار الإيجابي (ACK).
- إعادة الإرسال: سيقوم المُرسِل بإعادة إرسال الرزمة التي تصل إلى المُستقبِل وفيها أخطاء في البتات.

يبين الشكل 3-10 تمثيلاً لآلات الحالات المحدودة (FSM) لبروتوكول rdt2.0 لنقل البيانات والذي يستخدم أسلوباً لاكتشاف الأخطاء، وإشعارات استلام إيجابية (ACK)، وإشعارات استلام سلبية (NAK).



(a) بروتوكول rdt2.0: جانب الإرسال



(b) بروتوكول rdt2.0: جانب الاستقبال

الشكل 3-10 بروتوكول rdt2.0 لقناة تسبب أخطاءً في البتات.

يتضمن جانب المُرسِل في rdt2.0 حالتين. في الحالة المبيّنة أقصى اليسار يكون بروتوكول المُرسِل في انتظار عبور البيانات إليه من الطبقة الأعلى. على إثر الحدث rdt_send(data) سيقوم المُرسِل بتكوين رزمة sndpkt تتضمن البيانات المطلوب إرسالها، مع المجموع التدقيقي للرزمة (على سبيل المثال كما ذكرنا في حالة قطعة بيانات UDP بالجزء 3-3-2)، وبعد ذلك يُرسل الرزمة بواسطة عملية udt_send(sndpkt). أما في الحالة المبيّنة أقصى اليمين فينتظر بروتوكول المُرسِل وصول رزمة إشعار استلام (ACK أو NAK) من المُستقبِل. إذا تم استلام إشعار ACK

(يدل الرمز rdt_rcv(rcvpkt) && isACK(rcvpkt) في الشكل 3-10 على هذه الحالة) يدرك المُرسِل أن آخر رزمة أرسلها قد تم استلامها بشكلٍ صحيح بواسطة المُستقبِل، وهكذا يعود البروتوكول إلى حالة انتظار وصول بيانات من الطبقة الأعلى. إذا تم استلام إشعار NAK، يعيد البروتوكول إرسال الرزمة الأخيرة وينتظر وصول إشعار استلام (ACK أو NAK) من قبل المُستقبِل ردّاً على رزمة البيانات التي أعيد إرسالها. من المهم ملاحظة أنه عندما يكون المُرسِل في حالة انتظار وصول إشعار استلام (wait-for-ACK-or-NAK) لن يكون بوسعه تلقّي بيانات جديدة من الطبقة الأعلى لإرسالها، بمعنى أنه يتعذر حصول الحدث rdt_send() والذي سيحدث فقط بعد أن يتسلم المُرسِل إشعار الاستلام ACK ويغادر تلك الحالة. أي أن المُرسِل لن يبعث ببيانات جديدة إلا إذا تأكد من أن المُستقبِل قد تسلّم الرزمة الحالية بشكلٍ صحيح. بسبب هذا السلوك، يُطلق على البروتوكولات من نوع rdt2.0 بروتوكولات التوقّف والانتظار (stop-and-wait).

لا يزال لجانب المُستقبِل في rdt2.0 وضع واحد. عند وصول الرزمة يجيب المُستقبِل بإشعار استلام ACK أو NAK، حسب كون الرزمة المستلمة صحيحة أو فيها أخطاء. يدل الرمز rdt_rcv(rcvpkt) && corrupt(rcvpkt) في الشكل 3-10 على هذا الحدث في حالة استلام رزمة بها أخطاء.

قد يبدو البروتوكول rdt2.0 صحيحاً ويعمل على ما يرام، ولكنه لسوء الحظ يعاني من عيبٍ قاتل! بالتحديد لم نأخذ في الاعتبار حتى الآن احتمال حدوث أخطاء في رزمة إشعار الاستلام ACK أو NAK، (قبل متابعة القراءة، عليك أن تفكّر في طريقة لحل تلك المشكلة). لسوء الحظ فإن هفوتنا البسيطة ليست غير مؤذية كما قد يبدو للوهلة الأولى! كحد أدنى سنحتاج لإضافة حقل المجموع التدقيقي إلى رزم إشعار الاستلام ACK/NAK لاكتشاف مثل تلك الأخطاء. غير أن السؤال الأكثر صعوبة هو كيف يمكن أن يتعافى البروتوكول من الأخطاء في رزم إشعارات الاستلام؟ تكمن الصعوبة هنا في أنه في حالة فساد إشعار استلام نتيجة تلك الأخطاء لن يكون لدى المُرسِل أي طريقة لمعرفة ما إذا كان المستلم قد تلقّى قطعة البيانات الأخيرة المُرسَلة بشكلٍ صحيح.

خذ في الاعتبار البدائل الثلاثة التالية للتعامل مع فساد إشعارالاستلام ACK أو NAK نتيجةً لأخطاء البتات:

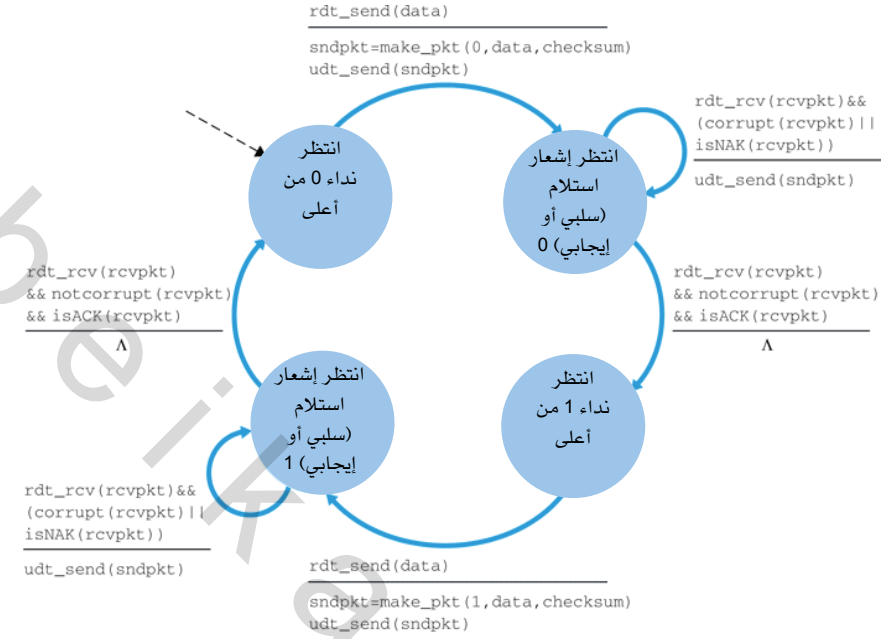
- البديل الأول: خذ في الاعتبار ماذا يمكن أن يحدث في المثال البشري الخاص بإملاء الرسائل. إذا لم يفهم المتكلم (مملي الرسالة) الإجابة "حسناً" أو "رجاءً كرّر" من آخذ الرسالة، فبوسع المتكلم أن يسأل: "ماذا قلت؟" (وبهذا نكون قد أضفنا نوعاً جديداً من الرزم من المُرسِل إلى المُستقبِل في بروتوكولنا). بعد ذلك يكرّر آخذ الرسالة إجابته. لكن ماذا لو أن "ماذا قلت؟" من المتكلم قد فسدت في الطريق هي الأخرى؟. في هذه الحالة لن يكون بوسع المُستقبِل تحديد ما إذا كانت الرزمة المشوهة التي وصلته جزءاً من الرسالة التي يجري إملاؤها أم طلباً لتكرار الإجابة الأخيرة. يُحتمل أن يرد هو الآخر عندئذ بـ "ماذا قلت؟" والتي بالطبع يمكن بدورها أن تفسد. واضح أننا بدأنا نسلك دروباً وعِراً.
- يتلخص البديل الثاني في إضافة حقل من المجموع التدقيقي يكون كافياً ليس فقط لاكتشاف الأخطاء بواسطة المُرسِل، بل أيضاً للتعافي منها. هذا يحل المشكلة التي نحن بصددها لقناة اتصال يمكن أن تفسد عليها الرزم ولكنها لا تُفقد تماماً.
- كبديل ثالث يمكن للمُرسِل ببساطة إعادة إرسال رزمة البيانات الحالية عندما يتلقى إشعار استلام ACK أو NAK فاسد. على أية حال قد تؤدي هذه الطريقة إلى تكرار الرزم المُرسَلة عبر قناة المُرسِل-المُستقبِل. تكمن المشكلة الأساسية للرزم المكرّرة في أن المُستقبِل لا يعرف ما إذا كان آخر إشعار استلام ACK أو NAK أرسله قد وصل إلى المُرسِل بشكلٍ صحيح أم لا، ومن ثم فإنه لا يستطيع الجزم بكون الرزمة الواصلة تضم بيانات جديدة أم أنها مجرد إعادة إرسال!

هناك حل بسيط لهذه المشكلة الجديدة (وهو حل مستخدم تقريباً في كل بروتوكولات نقل البيانات الحالية، بما في ذلك TCP). يكمن الحل في إضافة حقل جديد إلى رزمة البيانات وجعل المُرسِل يرقّم رزم بياناته بوضع رقم تسلسلي في ذلك

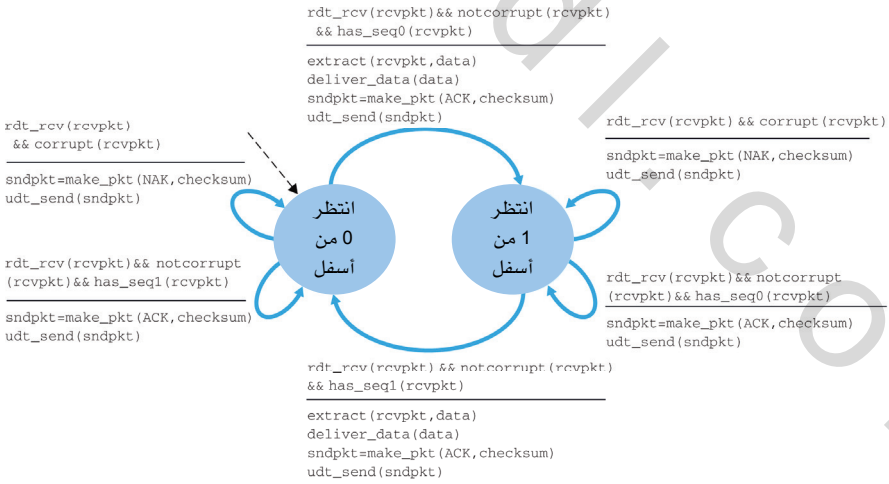
الحقل. عندئذ يحتاج المُستقبِل فقط لمراقبة هذا الرقم ليحدد ما إذا كانت الرزمة الواصلة جديدة أم إعادة إرسال. للحالة البسيطة لبروتوكول "توقّف وانتظر" الذي نحن بصدده، نحتاج لرقم تسلسلي حجمه بت واحدة فقط، حيث إن ذلك سيمكّن المُستقبِل من معرفة ما إذا كان المُرسِل يرسل للمرة الثانية الرزمة التي أرسلها سابقاً (الرقم التسلسلي للرزمة الجديدة هو نفسه للرزمة المستلمة مؤخراً) أو أنه يرسل رزمة جديدة (الرقم التسلسلي يتغيّر متحركاً "للأمام" حسب نظام حساب الباقي الثنائي (modulo-2 arithmetic). نظراً لأننا حالياً نفترض أن قناة الاتصال لا تفقد الرزم، فإن كلاً من إشعارات الاستلام نفسها ACK وNAK لا تحتاج لبيان الرقم المتسلسل للرزمة المتعلقة به. فالمُرسِل يعرف أن رزم ACK أو NAK (سواء كانت مشوشة أم لا) تم إرسالها من المُستقبِل كاستجابة لأحدث رزمة بيانات تم إرسالها.

يوضح الشكلان 3-11 و3-12 وصفاً لآلة FSM لبروتوكول rdt2.1 (إصدارنا المعدل لبروتوكول rdt2.0). لكلٍ من المُرسِل والمُستقبِل في هذا البروتوكول الآن ضعف عدد الأوضاع في الإصدار السابق. ذلك لأن حالة البروتوكول يجب أن تعكس الآن ما إذا كانت الرزمة التي ترسل حالياً (بواسطة المُرسِل) أو المتوقّع وصولها حالياً (عند المُستقبِل) لها الرقم التسلسلي 0 أو 1. لاحظ أن الإجراءات في تلك الأوضاع التي يتم فيها إرسال أو توقع رزمة رقم 0 هي صورة مطابقة لتلك التي يتم فيها إرسال أو توقع رزمة رقم 1 (الفرق الوحيد هو في التعامل مع الرقم المتسلسل).

يستخدم البروتوكول rdt2.1 كلاً من إشعارات الاستلام الإيجابية والسلبية من المُستقبِل إلى المُرسِل. عندما تصل رزمة برقم تسلسلي غير المتوقع، يُرسل المُستقبِل إشعار استلام إيجابي عن آخر رزمة تم استلامها. عندما تصل رزمة فيها أخطاء، يرسل المُستقبِل إشعار استلام سلبي. يمكننا الحصول على نفس تأثير الإشعار السلبي إذا أرسلنا بدلاً منه إشعار استلام إيجابي لآخر رزمة تم استلامها بشكلٍ صحيح. عندئذٍ سيعرف المُرسِل الذي يستلم إشعاري استلام إيجابيين لنفس الرزمة (أي يستلم إشعارات استلام مكرّرة) أن المُستقبِل لم يستلم بشكلٍ صحيح الرزمة التي أُرسلت بعد الرزمة التي وصل بخصوصها إشعاران إيجابيان.

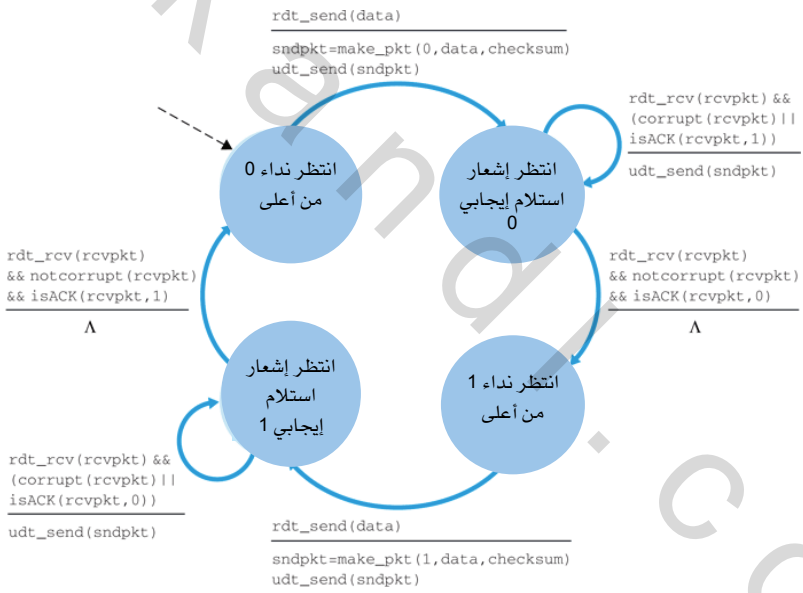


الشكل 3-11 مُرسل بروتوكول rdt2.1.

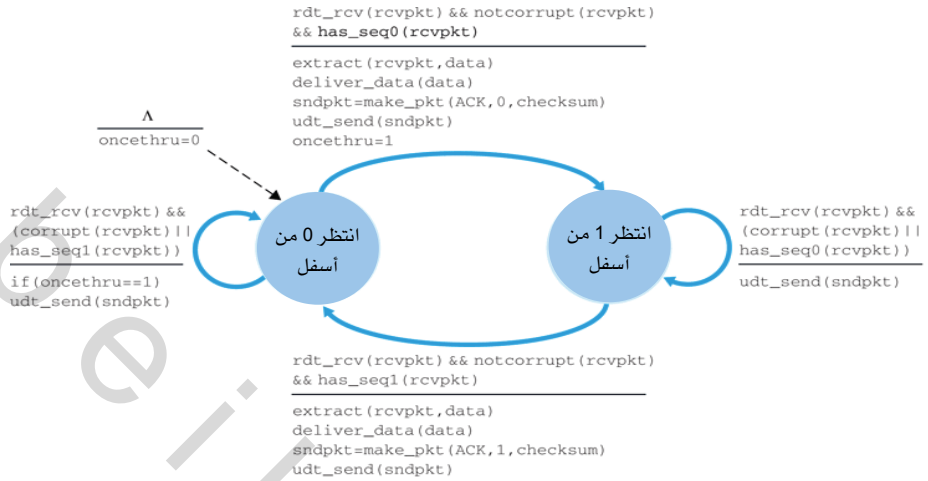


الشكل 3-12 مُستقبل بروتوكول rdt2.1.

يبين الشكلان 3-13 و3-14 بروتوكولنا الجديد rdt2.2 للنقل الموثوق للبيانات على قناة اتصال بأخطاء في بتات القطعة وبدون استخدام إشعار الاستلام السلبي. من التغييرات الدقيقة بين rdt2.1 وrdt2.2 أن المُستقبِل عليه الآن أن يضمّن الرقم التسلسلي للرزمة في رسالة إشعار الاستلام ACK (وذلك بتضمين ACK,0 أو ACK,1 كمعامل في make_pkt() في آلة FSM الخاصة بالمُستقبِل)، كما أن على المُرسِل الآن فحص الرقم التسلسلي للرزمة الجاري إشعار الطرف الآخر باستلامها بواسطة رزمة ACK الواصلة (وذلك بتضمين الرقم 0 أو 1 كمعامل في isACK() في آلة FSM الخاصة بالمُرسِل).



الشكل 3-13 مُرسِل بروتوكول rdt2.2.



الشكل 3-14 مُستقبِل بروتوكول rdt2.2 .

النقل الموثوق للبيانات عبر قناة تُفقد فيها الرزم وتتعرض لأخطاء بتات: rdt3.0

افترض الآن أنه بالإضافة إلى فساد قطع البيانات بسبب أخطاء في البتات، يمكن أيضاً أن تفقد قناة الاتصال التحتية الرزم، وهذا شائع في شبكات الحاسب اليوم (بما في ذلك الإنترنت). هناك الآن اعتباران إضافيان يجب أن يعالجهما البروتوكول: كيف يمكن اكتشاف فقد رزمة وما الإجراء الذي ينبغي اتخاذه إزاء ذلك. إن استعمال المجموع التدقيقي، ورقم الرزمة المتسلسل، وإشعار استلام الرزمة، وإعادة الإرسال – وهي الأساليب التي طوّرناها في بروتوكول rdt2.2 – ستمكننا من التعامل مع الاعتبار الأخير، أما معالجة الاعتبار الأول فتتطلب إضافة آلية جديدة للبروتوكول.

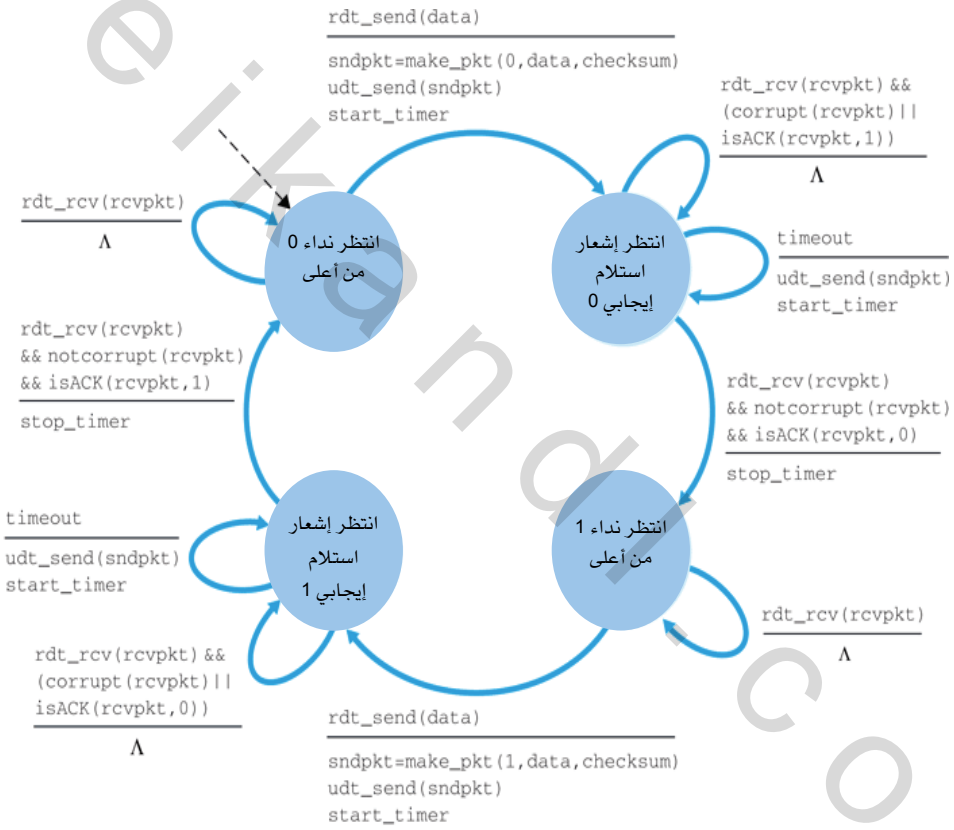
هناك العديد من الطرق للتعامل مع فقد الرزم (نستعرض المزيد منها في تمارين نهاية هذا الفصل). سنضع هنا عبء اكتشاف فقد الرزم والتعافي من ذلك على المُرسِل. افترض أن المُرسِل يرسل رزمة بيانات وأن تلك الرزمة أو إشعار الاستلام الخاص بها يفقد. في أي الحالتين لن تكون هناك إجابة قادمة من المُستقبِل إلى المُرسِل. إذا كان المُرسِل مستعداً للانتظار مدة طويلة تكفي للتأكد من أن رزمة قد فقدت، فبوسعه إعادة إرسال رزمة البيانات ببساطة بعد فترة الانتظار تلك. عليك أن تقنع نفسك بأن هذا البروتوكول يعمل في واقع الأمر!

ولكن إلى متى يجب على المُرسِل الانتظار ليتأكد من أن شيئاً ما قد فُقد؟ واضحٌ أنه على المُرسِل أن ينتظر على الأقل زمن التأخير المناظر لرحلة ذهاب وإياب الإشارة بين المُرسِل والمُستقبِل (والذي قد يتضمن زمن التخزين المؤقت في الموجّهات المتوسطة) بالإضافة إلى المدة اللازمة لمعالجة الرزمة لدى المُستقبِل. في العديد من الشبكات يصعب جداً تقدير هذا التأخير الأقصى لأسوأ الحالات، فضلاً عن تحديده بدقة. من ناحية أخرى وفي الوضع المثالي، على البروتوكول أن يتعافى من فقد الرزمة بأسرع ما يمكن، والانتظار لمدة تأخير مناظرة لأسوأ الحالات قد يعني انتظاراً طويلاً قبل أن تبدأ آليات التعافي من الخطأ في العمل. إن الطريقة المتبعة عملياً هي أن يختار المُرسِل بحذر قيمةً لوقت التأخير يكون بعدها فقد الرزمة محتمل الحدوث، حتى إن لم يكن الفقد مؤكداً. فإذا لم يصل إلى المُرسِل إشعار استلام ACK في غضون ذلك الوقت فإنه يعيد إرسال الرزمة على أي حال. لاحظ أنه إذا واجهت رزمة تأخيراً كبيراً جداً، فإن المُرسِل قد يعيد إرسال رزمة رغم عدم فقدها أو فقد إشعار استلامها، مما قد يؤدي إلى احتمال إرسال رزم مكرّرة من المُرسِل إلى المُستقبِل عبر قناة الاتصال. لحسن الحظ تتوافر لبروتوكول rdt2.2 إمكانية التعامل مع حالة الرزم المكرّرة (عن طريق الأرقام التسلسلية للرزم).

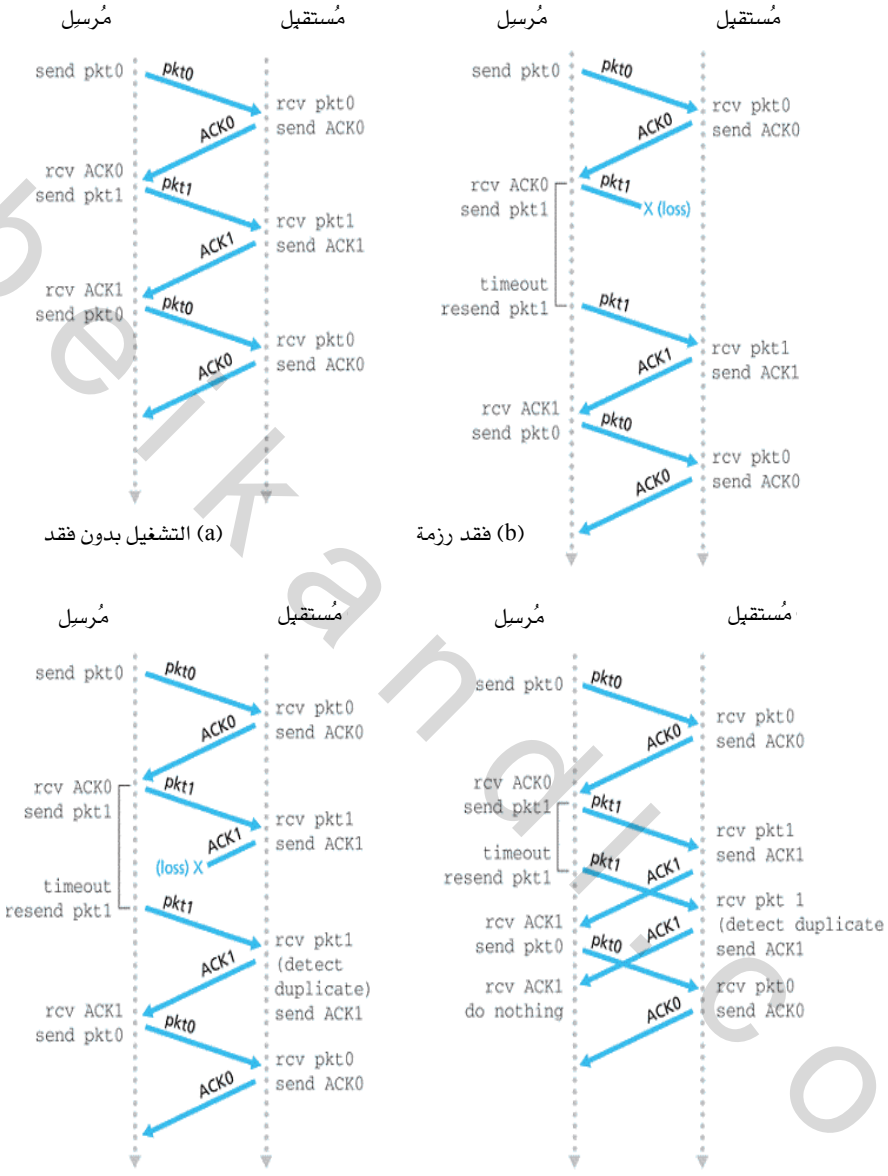
من وجهة نظر المُرسِل تعتبر إعادة الإرسال حلاً ناجعاً، فالمُرسِل لا يعرف ما إذا كانت رزمة بيانات قد فُقدت، أو أن إشعار الاستلام الخاص بها قد فُقد، أو أن الرزمة أو إشعار الاستلام ببساطة قد تأخرا في الطريق أكثر من اللازم. في كل تلك الحالات الإجراء المتخذ واحد: "أعِد الإرسال". يحتاج تنفيذ آلية إعادة الإرسال إلى موقّت بعَدٍّ تنازلي يقوم بمقاطعة (interrupt) المُرسِل بعد انقضاء فترة زمنية معينة يتم برمجتها مسبقاً. مما تقدم يتعين على المُرسِل أن يكون قادراً على: (1) بدأ الموقّت عند إرسال كل رزمة (سواء كانت رزمة جديدة أو رزمة معاداً إرسالها)، (2) الاستجابة لمقاطعة الموقّت (باتخاذ التدابير الملائمة)، و(3) إيقاف الموقّت.

يبين الشكل 3-15 آلة FSM للمُرسِل في بروتوكول rdt3.0 للنقل الموثوق للبيانات على قناة اتصال يمكن أن تُفسد أو تفقد الرزم. في تمارين نهاية الفصل، سيُطلب منك تصميم آلة FSM للمُستقبِل في بروتوكول rdt3.0. يوضح الشكل 3-16 كيف يعمل البروتوكول بدون رزم مفقودة أو متأخرة وكيف يعالج فقد رزم البيانات. في الشكل 3-16 يتقدّم الوقت للأمام من أعلى الشكل إلى أسفله. لاحظ أن وقت استلام رزمة يأتي بالضرورة بعد وقت إرسال الرزمة نتيجة لتأخيرات

الإرسال والانتقال. في الأشكال 3-16 (b)-(d) تشير الأقواس الجانبية إلى الأوقات التي يضبط عندها الموقّت والأوقات التي تنتهي فيها مدته لاحقاً. يتم استكشاف العديد من السمات الدقيقة لهذا البروتوكول في التمارين في نهاية هذا الفصل. نظراً لأن الأرقام المتسلسلة للرزم تتبادل ما بين 0 و1، فإن بروتوكول rdt3.0 يعرف أحيانا ببروتوكول البت المتناوبة (alternating bit protocol).



الشكل 3-15 مُرسِل بروتوكول rdt 3.0.



(a) التشغيل بدون فقد

(b) فقد رزمة

(c) فقد إشعار استلام

(d) انقضاء فترة الموقّت قبل وصول الإشعار

الشكل 3-16 طريقة عمل بروتوكول rdt 3.0 (بروتوكول البت المتناوبة).

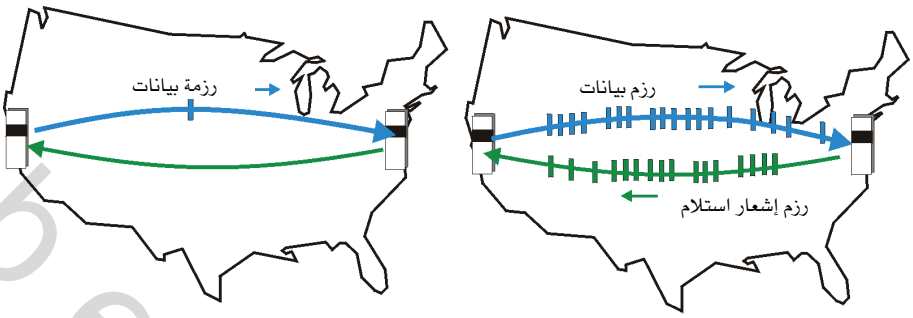
لقد جمّعنا الآن العناصر الرئيسة لبروتوكول نقل البيانات. يلعب كلٌّ من المجموع التدقيقي، والأرقام التسلسلية، والموقّتات، وإشعارات الاستلام الإيجابية والسلبية دوراً حاسماً وضرورياً في عمل البروتوكول. أخيراً أصبح لدينا الآن بروتوكولاً يصلح لنقل البيانات بشكلٍ موثوق!

3-4-2 بروتوكولات النقل الموثوق للبيانات بأسلوب خط الأنابيب

رغم أن بروتوكول rdt3.0 صحيحٌ إجرائياً، فإنه من غير المحتمل أن يكون أحدٌ سعيداً بأدائه، خصوصاً في شبكات اليوم السريعة. فأساس مشكلة أداء بروتوكول rdt3.0 أنه نظام توقّف وانتظار.

لإدراك تأثير التوقّف والانتظار على أداء البروتوكول، تصور حالة مثالية: مضيفين أحدهما يقع على الساحل الغربي للولايات المتحدة والآخر يقع على الساحل الشرقي، كما هو مبين في الشكل 3-17. يبلغ تأخير انتقال الضوء في رحلة الذهاب والإياب (*RTT*) بين هذين النظامين الطرفيين حوالي 30 ميللي ثانية. افترض أنهما موصلان عن طريق قناة اتصال لها معدل إرسال للبيانات (R) قدره 1 جيجابت/ثانية (= 10^9 بت/ثانية)، وأن حجم كل رزمة L = 1000 بايت بما في ذلك حقول البيانات والترويسة، وبالتالي يكون الوقت اللازم لإرسال تلك الرزمة على الوصلة هو:

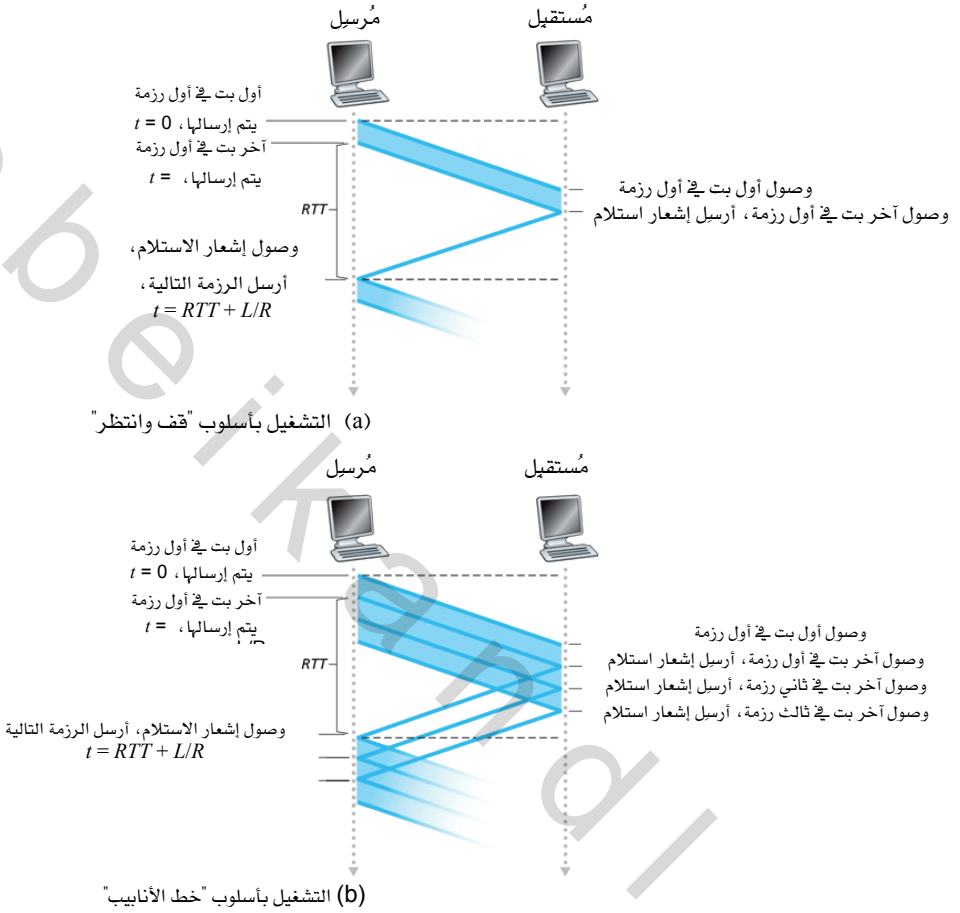
$$d_{trans} = \frac{L}{R} = \frac{8 \times 1000 \ bits / packet}{10^9 bits / \sec} = 8 \ \mu \sec$$



الشكل 3-17 بروتوكول "قف وانتظر" في مقابل بروتوكول "خط الأنابيب".

يبين الشكل 3-18 (a) أنه باستخدام بروتوكول من نوع "توقّف وانتظر"، إذا بدأ المُرسِل بإرسال رزمة عند زمن t = 0، فإنه عند t =R/L = 8 ميكروثانية، تدخل البت الأخيرة من الرزمة القناة في جانب المُرسِل. تقطع الرزمة رحلتها بعرض الولايات المتحدة في 15 ميللي ثانية، ومن ثم تصل البت الأخيرة من الرزمة إلى المُستقبِل عند $t = RTT/2 + L/R$ = 15.008 ميللي ثانية. افترض للتبسيط أن رزم إشعار الاستلام صغيرة جداً (بحيث يمكننا إهمال وقت إرسالها) وأنه بوسع المُستقبِل إرسال إشعار استلام بمجرد وصول البت الأخيرة من رزمة البيانات إليه، وبذلك يصل إشعار الاستلام إلى المُرسِل عند $t = RTT + L/R$ = 30.008 ميللي ثانية. في تلك اللحظة يمكن للمُرسِل البدء في بث الرزمة التالية. وهكذا فخلال 30.008 ميللي ثانية، تمكّن المُرسِل من بث البيانات لمدة 0.008 ميللي ثانية فقط. إذا عرّفنا استغلال (utilization) المُرسِل (أو القناة) على أنه كسر الوقت الذي يكون فيه المُرسِل مشغولاً ببث البيانات بالفعل إلى القناة، فإنه بدراسة الشكل 3-18 (a) يتضح أن نظام التوقّف والانتظار يعطي استغلالاً ضئيلاً للمُرسِل قيمته:

$$U_{sender} = \frac{L/R}{RTT + L/R} = \frac{0.008}{30.008} = 0.00027$$



الشكل 3-18 الإرسال ببروتوكول "قف وانتظر" و بروتوكول "خط الأنابيب".

أي أن المُرسِل يكون مشغولاً بالإرسال فقط بنسبة 2.7 جزء من كل عشرة آلاف جزء من الوقت! من منظور آخر، يكون المُرسِل قادراً على إرسال 1000 بايت فقط خلال 30.008 ميللي ثانية، أي بطاقة إنتاجية فعلية مقدارها 267 كيلو بت/ثانية فقط رغم توفر استخدام قناة سعة إرسالها 1 جيجابت/ثانية. تخيل موقف مدير الشبكة المستاء الذي دفع ثروة طائلة لتوفير وصلة بسعة إرسال 1 جيجابت/ثانية ولكنه لا يستطيع سوى تأمين طاقة إنتاجية لا تعدو 267 كيلوبت/ثانية فقط. هذا مثال رسومي يوضح كيف يمكن لبروتوكولات

الشبكة أن تُحِدّ من استغلال الإمكانيات التي توفرها البنية التحتية للشبكة. تذكّر أننا أهملنا أيضاً أوقات المعالجة لبروتوكولات الطبقات الأدنى في المُرسِل والمُستقبِل، بالإضافة إلى أوقات المعالجة وتأخيرات الانتظار في الصف التي يمكن أن تحدث في أي من الموجّهات المتوسّطة بين المُرسِل والمُستقبِل. لاحظ أن أخذ تلك التأثيرات بعين الاعتبار سيزيد من التأخير أكثر، وبالتالي سيزيد الأداء السيّئ سوءاً.

إن حل مشكلة الأداء هذه بسيط من حيث المبدأ: بدلاً من العمل بطريقة توقّف وانتظر، يُسمح للمُرسِل بإرسال عدة رزم بدون انتظار لإشعارات الاستلام، كما يبيّن الشكل 3-17 (b). يوضح الشكل 3-18 (b) أنه إذا ما سُمح للمُرسِل بإرسال ثلاثة رزم قبل الحاجة لانتظار إشعارات الاستلام، فإن نسبة الاستغلال عند المُرسِل تزداد إلى ثلاثة أضعاف تقريباً. لمّا كانت رزم البيانات العديدة المرصوصة على الوصلة في نفس الوقت في طريقها من المُرسِل إلى المُستقبِل يمكن تصورها كما لو كانت تملأ خط أنابيب، تعرف هذه الطريقة بأسلوب خط الأنابيب (pipeline)، ولهذا الأسلوب الانعكاسات التالية على بروتوكولات النقل الموثوق للبيانات:

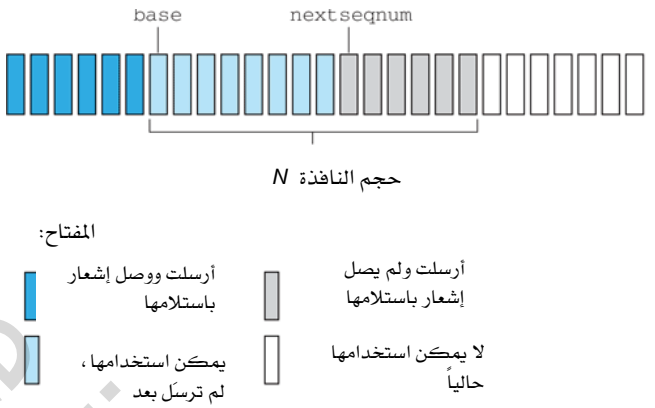
- ينبغي زيادة مدى الأرقام التسلسلية للرزم، نظراً لأنه يجب أن يكون لكل الرزم التي ما زالت في طريقها إلى المُستقبِل (باستثناء الرزم التي يعاد إرسالها) رقم تسلسلي فريد، كما يمكن أن تكون هناك عدة رزم لم يتم الإشعار عن استلامها.
- قد يحتاج الأمر إلى التخزين المؤقت لأكثر من رزمة لدى كلٍّ من جانبي المُرسِل والمُستقبِل من البروتوكول، لذا يتعين على المُرسِل كحد أدنى توفير حيّز تخزين مؤقت للرزم التي تم إرسالها ولم تصل إشعارات استلامها بعد. أحياناً يحتاج المُستقبِل أيضاً لتخزين رزم تم استلامها بشكلٍ صحيح، كما سنرى لاحقاً.
- يعتمد مدى الرقم التسلسلي المطلوب ومتطلبات التخزين المؤقت اللازمة على طريقة تعامل بروتوكول نقل البيانات مع الرزم المفقودة

والفاسدة والمتأخرة تأخيراً غير عادي. هناك طريقتان أساسيتان للتعافي من الخطأ في بروتوكول المعالجة بأسلوب خط الأنابيب: طريقة "ارجع N للوراء" ((Go-Back-N (GBN)) وطريقة "الإعادة الانتقائية" ((Selective Repeat (SR)).

3-4-3 بروتوكول "ارجع N للوراء" (GBN)

في بروتوكول "ارجع N للوراء" يُسمح للمُرسِل بإرسال عدة رزم (عند توفرها) بدون انتظار وصول إشعارات الاستلام، لكن الحد الأقصى المسموح به هو عدد N من تلك الرزم، سنتناول هنا بروتوكول GBN بشيء من التفصيل، لكن قبل مواصلة القراءة ننصحك بتشغيل برنامج جافا GBN applet الموجود في موقع الويب المصاحب لهذا الكتاب (وهو برنامج توضيحي رائع!).

يبين الشكل 3-19 مدى الأرقام التسلسلية للرزم من منظور المُرسِل في بروتوكول GBN. إذا عرّفنا base بأنه الرقم التسلسلي لأقدم رزمة أرسلت ولم يصل بعد إشعار باستلامها، و nextseqnum بأنه أصغر رقم تسلسلي لم يستخدم بعد (أي الرقم التسلسلي للرزمة التي سترسل في المرة القادمة)، عندئذٍ يمكن تحديد أربع فترات في مدى الأرقام التسلسلية: الأرقام في الفترة [0, base-1] وتناظر الرزم التي أرسلت ووصلت إشعارات باستلامها. والفترة [base, nextsegnum-1] وتناظر الرزم التي أرسلت ولكن لم تصل إشعارات باستلامها بعد، والفترة [nextsegnum, base +N-1] وتناظر الرزم التي يمكن أن ترسل فور وصول بيانات من الطبقة الأعلى، وأخيراً الفترة التي فيها الرقم التسلسلي ≥ base+N والتي لن يتسنى استخدام أرقامها للإرسال قبل وصول إشعار استلام لرزمة مُرسَلة وموجودة حالياً في خط الأنابيب (بالتحديد رزمة تحمل الرقم التسلسلي base).



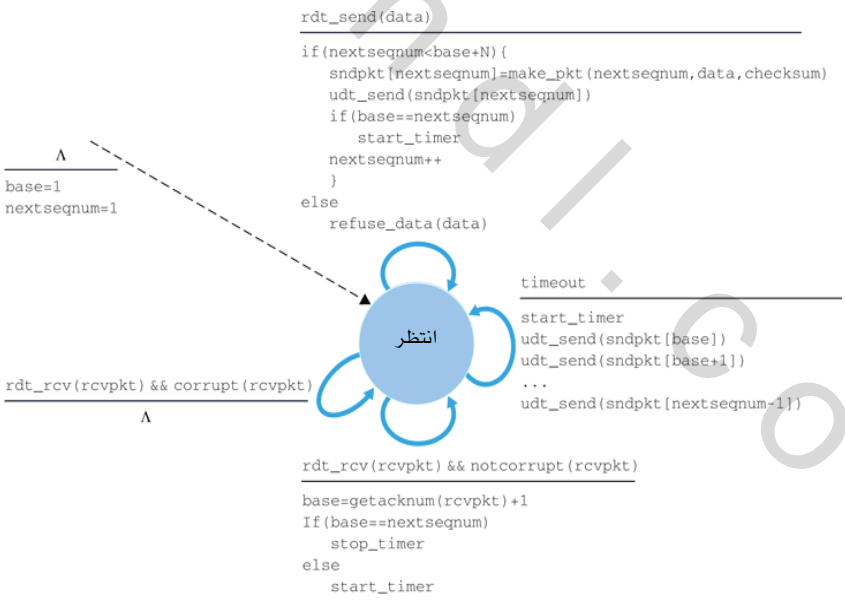
الشكل 3-19 الأرقام التسلسلية من منظور المُرسِل في بروتوكول العودة N للوراء (GBN).

من خلال الشكل 3-19 يمكن اعتبار مدى الأرقام التسلسلية المسموح به لإرسال الرزم مع عدم وصول إشعارات استلامها على أنه نافذة مقاسها N. وأثناء تشغيل البروتوكول تنزلق تلك النافذة للأمام على مدى الأرقام التسلسلية. لهذا السبب غالباً ما يطلق على N حجم النافذة وعلى البروتوكول نفسه بروتوكول النافذة المنزلقة (sliding window protocol). قد تتساءل لماذا نُحد من عدد الرزم المنتظرة بدون إشعار استلام بالقيمة N في المقام الأول؟ لماذا لا نسمح لعدد غير محدود من تلك الرزم؟ سنرى في الجزء 3-5 أن ضبط التدفق يمثل أحد الأسباب لوضع ذلك القيد على المُرسِل. سنرى سبباً آخر لذلك في الجزء 3-7 عندما ندرس السيطرة على الازدحام في بروتوكول TCP.

عملياً يُوضع الرقم التسلسلي للرزمة في حقل طوله ثابت في ترويسة الرزمة. إذا كانت k هي عدد البتات في حقل الرقم التسلسلي للرزمة فإن مدى الأرقام التسلسلية يكون $[0, 2^k-1]$. في وجود مدى محدود من تلك الأرقام التسلسلية، يتعين أن تُجرى كل الحسابات المتعلقة بتلك الأرقام التسلسلية على أساس modulo-2^k (بمعنى أن فضاء الأرقام التسلسلية يمكن تصوره كحلقة مقاسها 2^k، حيث يأتي الرقم 0 بعد الرقم 2^k-1). تذكّر أنه في البروتوكول rdt3.0 كانت الأرقام التسلسلية لها حقل يتكون من بت واحدة ومدى $[0، 1]$. من خلال عدة تمارين في نهاية هذا

الفصل ستكتشف النتائج المترتبة على المدى المحدود للأرقام التسلسلية. سنرى في الجزء 3-5 أنه في بروتوكول TCP يتألف حقل الرقم التسلسلي من 32 بتاً، حيث تعِدّ الأرقام التسلسلية في هذا البروتوكول البايتات المُرسَلة بدلاً من الرزم.

يصور الشكلان 3-20 و 3-21 وصفاً لآلة FSM موسعة لكلٍ من جانبي المُرسِل والمُستقبل لبروتوكول مبني على إشعارات الاستلام الإيجابية ACK وبدون إشعارات استلام سلبية NAK ويستخدم طريقة "ارجع N للوراء" (GBN). أطلقنا وصف "موسّعة" على آلة FSM تلك لأننا أضفنا متغيرات (تشبه المتغيرات المستخدمة في لغة البرمجة) لكلٍ من base و nextseqnum، بالإضافة إلى عمليات وإجراءات شرطية تتعلق بهذين المتغيرين. لاحظ أن مواصفات آلة FSM الموسعة الآن أخذت تشبه بعض الشيء مواصفات لغة البرمجة. يعطي [Bochman 1984] استعراضاً ممتازاً للامتدادات الإضافية لأساليب FSM وغيرها من الأساليب المبنية على لغات البرمجة والمستخدمة لتوصيف البروتوكولات.



الشكل 3-20 وصف لآلة الأوضاع المحدودة (FSM) الموسّعة للمُرسِل في بروتوكول العودة N للوراء (GBN).

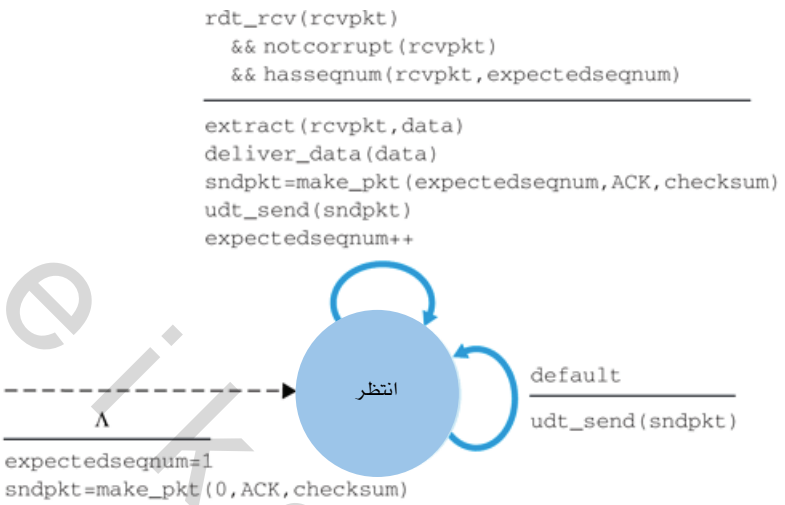
ينبغي على مُرسِل بروتوكول GBN الاستجابة لثلاثة أنواع من الأحداث:

- استدعاء من أعلى: عندما يحدث نداء لـ ()rdt_send من الطبقة الأعلى، يتأكد المُرسِل أولاً مما إذا كانت النافذة ممتلئة، بمعنى أن هناك N رزمة مُرسَلة لم تصل إشعارات استلامها. فإذا كانت النافذة ليست ممتلئة، يقوم المُرسِل بتكوين رزمة وإرسالها، ثم بتحديث المتغيرات بالشكل الملائم، وإذا كانت النافذة ممتلئة، يُعيد المُرسِل البيانات ببساطة إلى الطبقة الأعلى، في إشارة ضمنية لأن النافذة ممتلئة. يُفترض أن تعاود الطبقة الأعلى محاولة الإرسال من جديد لاحقاً. في تطبيق حقيقي للبروتوكول قد يقوم المُرسِل على الأرجح بتخزين تلك البيانات عنده بشكلٍ مؤقت (ولكنه لا يرسلها على الفور)، أو تكون لديه آلية للتزامن (على سبيل المثال سيمافور (semaphore) أو علَم (flag) تسمح للطبقة الأعلى باستدعاء rdt_send() فقط عندما تكون النافذة غير ممتلئة.
- تلقّي إشعار استلام: في بروتوكول من نوع GBN سيُأخذ إشعار الاستلام لرزمة تحمل الرقم التسلسلي n على أنه إشعار تراكمي بأن كل الرزم التي تحمل الأرقام التسلسلية حتى (وبما في ذلك) الرقم n قد تم استلامها بشكلٍ صحيح في المُستقبل. لنا عودة إلى هذا الوضع بعد قليل عندما نتناول جانب المُستقبِل من بروتوكول GBN.
- انقضاء فترة الموقّت: تذكّر أن اسم البروتوكول "ارجع N للوراء" مشتق من سلوك المُرسِل إزاء الرزم المفقودة أو التي تأخرت كثيراً. كما في نظام التوقّف والانتظار، يُستخدم هنا موقّت للتعامل من حالات فقد رزم البيانات أو رزم إشعارات الاستلام. إذا انقضت فترة الموقّت يقوم المُرسِل بإعادة إرسال كل الرزم التي أُرسلت سابقاً ولم تصل إشعارات باستلامها بعد. يستخدم مُرسِلنا في الشكل 3-20 موقّتاً واحداً فقط، والذي يمكن اعتباره موقّتاً لأقدم رزمة تم إرسالها ولكن لم يصل بعد إشعار استلامها. إذا لم تكن هناك أي رزم متبقية بدون وصول إشعار باستلامها، يتم إيقاف الموقّت قبل انقضاء فترته.

إنّ أعمال المُستقبِل في بروتوكول GBN هي أيضاً بسيطة. إذا وصلت رزمة برقم تسلسلي n وتم استلامها صحيحة وبترتيب سليم (بمعنى أن آخر بيانات تم

رفعها للطبقة الأعلى تم استخراجها من رزمة لها الرقم التسلسلي n - 1)، يرسل المُستقبِل إشعار استلام بخصوص الرزمة n ويقوم بتوصيل جزء البيانات من الرزمة إلى الطبقة الأعلى. في كل الحالات الأخرى يهمل المُستقبِل الرزمة الحالية ويعيد إرسال إشعار استلام لأحدث رزمة تم استلامها بالترتيب السليم. لاحظ أنه نظراً لأن الرزم يتم توصيلها الواحدة تلو الأخرى إلى الطبقة الأعلى، فإنه في حالة استلام الرزمة k بواسطة المُستقبِل وتوصيلها للطبقة الأعلى، فإن كل الرزم برقم تسلسلي أقل من k تكون قد سُلّمت أيضاً. وهكذا فإن الاستعمال التراكمي لإشعارات الاستلام يعتبر اختياراً طبيعياً في بروتوكول GBN.

في بروتوكول GBN يُهمل المُستقبِل الرزم التي تصل في غير ترتيبها السليم. ورغم أن إهمال رزم تصل صحيحة ولكن بترتيب خطأ قد يبدو تصرفاً غير حكيم وينطوي على شيء من التبذير، فإن هناك بعض التبريرات لذلك. تذكّر أنه على المُستقبِل توصيل البيانات للطبقة الأعلى بالترتيب. افترض الآن أن الحزمة n متوقّعة ولكن الرزمة $n + 1$ هي التي وصلت. نظراً لأن البيانات ينبغي توصيلها بالترتيب، فإنه يجب على المُستقبِل هنا أن يخزن الرزمة $n + 1$ لديه مؤقتاً على أن يسلّم تلك الرزمة إلى الطبقة الأعلى لاحقاً بعد أن يكون قد استلم الرزمة n وسلّمها للطبقة الأعلى. ومع ذلك فإذا فقدت الرزمة n، فإن كلاً من تلك الرزمة والرزمة التالية لها ($n + 1$) سيعاد إرسالهما في النهاية كنتيجة لقواعد بروتوكول GBN التي تحكم إعادة الإرسال. وعليه يمكن للمستلم ببساطة أن يهمل الرزمة $n + 1$. تكمن ميزة هذه الطريقة في تبسيط آليات التخزين المؤقت لدى المُستقبِل، حيث لا يحتاج المُستقبِل للتخزين المؤقت للرزم التي لا تأتي في ترتيبها السليم. وهكذا فبينما يتعين على المُرسِل الإبقاء على الحدود العليا والدنيا لنافذته وموضع nextseqnum ضمن حدود تلك النافذة، فإن المعلومة الوحيدة التي يجب على المُستقبِل الاحتفاظ بها هي الرقم التسلسلي للرزمة التالية حسب الترتيب السليم، تحفظ هذه القيمة في المتغير expectedseqnum، والموضحة في آلة FSM للمُستقبِل في الشكل 3-21. بالطبع فإن من عيوب إهمال رزمة صحيحة أن إعادة إرسال تلك الرزمة لاحقاً قد يتعرض للفقد أو لفساد البيانات، مما يتطلب بدوره المزيد من إعادة الإرسال.

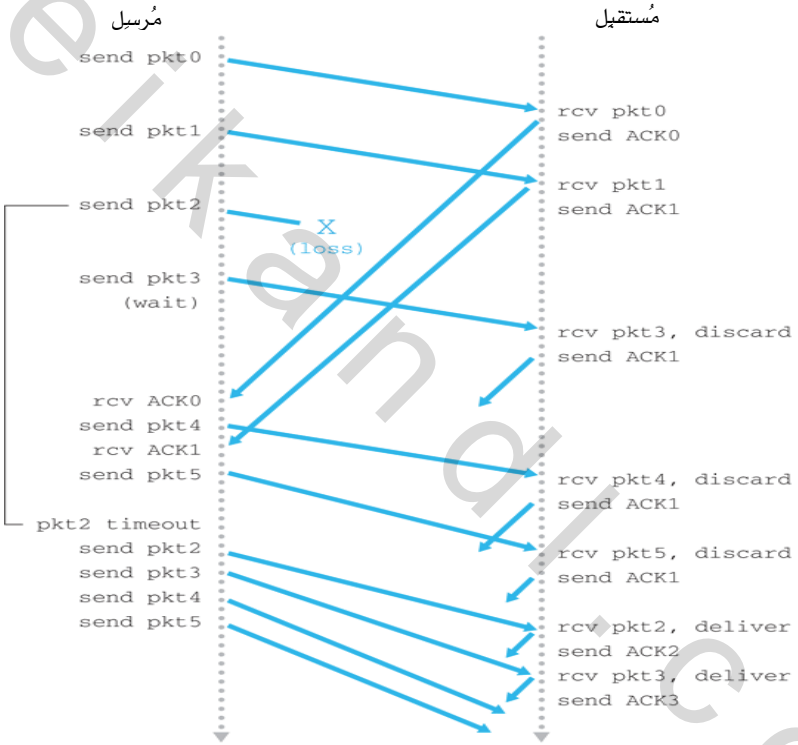


الشكل 3-21 وصف لآلة الأوضاع المحدودة (FSM) الموسّعة للمُستقبِل في بروتوكول العودة N للوراء (GBN).

يبين الشكل 3-22 عمل بروتوكول GBN عند استخدام نافذة حجمها أربع رزم. بسبب التقيد بسعة النافذة يرسل المُرسِل الرزم التي تحمل الأرقام التسلسلية من 0 إلى 3، ولكن عليه بعد ذلك أن ينتظر لحين وصول إشعارات استلام لواحد أو أكثر من تلك الرزم قبل المُضي قدُماً في إرسال المزيد من الرزم. وبوصول إشعارات الاستلام الواحد تلو الآخر (مثلاً ACK0 ثم ACK1)، تنزلق النافذة للأمام ويقوم المُرسِل ببث رزمٍ جديدة (رزمة 4 ثم رزمة 5 على التوالي). على جانب المُستقبِل تفقد الرزمة 2 ومن ثم تصل الرزم 3 و4 و5 بغير الترتيب السليم وبالتالي يتم إهمالها.

قبل أن ننهي مناقشتنا لبروتوكول GBN يجدر بنا ملاحظة أن تطبيقه في رصة البروتوكولات يتطلب في الغالب هيكلاً يشبه هيكل آلة FSM الموسّعة في الشكل 3-20، وغالباً ما سيأخذ هذا التطبيق أيضاً شكل إجراءات مختلفة لتنفيذ الأعمال المطلوب تنفيذها كاستجابة للأحداث المختلفة التي يمكن أن تقع. في مثل هذه البرمجة المبنية على الحدث (event-driven programming) تم استدعاء (تفعيل) الإجراءات المختلفة بواسطة إجراءات أخرى في رصّة البروتوكولات، أو كنتيجة

لحدوث مقاطعة (interrupt). بالنسبة للمُرسِل تتضمن هذه الأحداث ما يلي: (1) نداء من الكيان في الطبقة الأعلى لتشغيل rdt_send()، (2) مقاطعة من موقّت، (3) نداء من الطبقة الأسفل لتشغيل rdt_rcv() عند وصول رزمة. تتيح لك تمارين البرمجة في نهاية هذا الفصل الفرصة للتطبيق الحقيقي لتلك البرامج في محاكاة لمواقف واقعية لشبكات الحاسب.



الشكل 3-22 العمل ببروتوكول العودة *N* للوراء GBN.

نلاحظ هنا أن بروتوكول GBN يتضمن تقريباً كل الأساليب التي سوف نستعرضها عند دراستنا لمكوّنات بروتوكول TCP للنقل الموثوق للبيانات في الجزء 3-5. تتضمّن تلك الأساليب استعمال الأرقام التسلسلية، وإشعارات الاستلام

التراكمية، وحقل المجموع التدقيقي لاكتشاف أخطاء البيانات، وإعادة الإرسال بناءً على انقضاء فترة الموقّت.

4-4-3 بروتوكول "الإعادة الانتقائية" (SR)

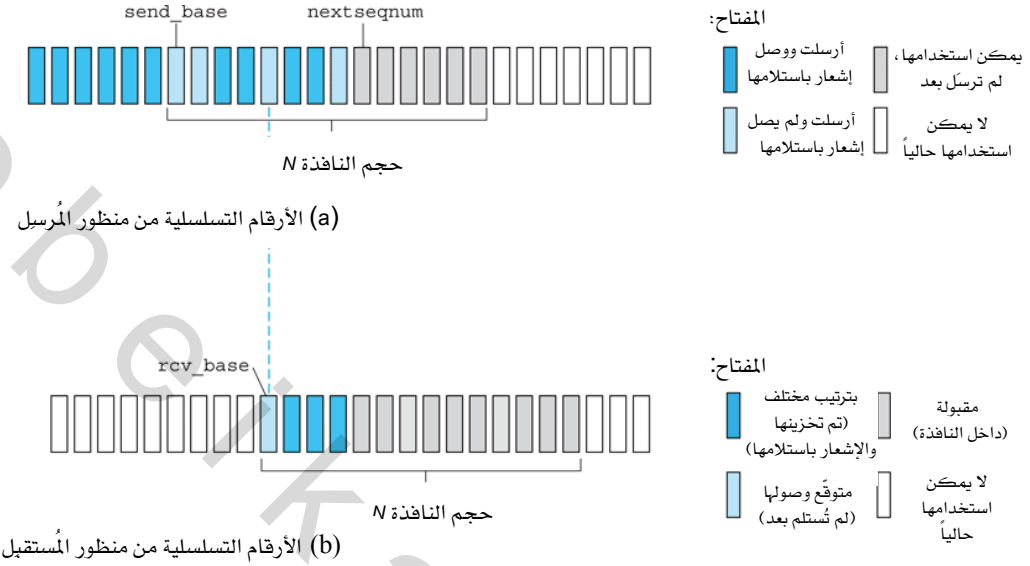
يوفر بروتوكول GBN للمُرسِل إمكانية "ملء خط الأنابيب" في الشكل 3-17 بالرزم، وبهذا يتفادى مشاكل انخفاض معدل استغلال قناة الاتصال التي أشرنا إليها أثناء تناولنا لبروتوكول التوقف والانتظار. ومع ذلك فهناك مواقف يعاني فيها بروتوكول GBN نفسه من مشاكل في الأداء، وبشكلٍ خاص عندما يكون كلٌّ من حجم النافذة وحاصل الضرب (الحيّز الترددي × التأخير) كبيرين، يمكن أن يكون العديد من الرزم في خط الأنابيب على الوصلة في نفس الوقت. أي خطأ في أي حزمة يمكن أن ينتج عنه قيام بروتوكول GBN بإعادة إرسال عدد كبير من الرزم – العديد منها بدون مبرر حيث إنها كانت قد وصلت صحيحة من قبل. مع زيادة احتمال حدوث أخطاء في البيانات أثناء انتقالها عبر القناة، يمكن أن يصبح خط الأنابيب مملوءاً بتلك الرزم المعاد إرسالها بدون داعٍ. في سيناريو إملاء الرسالة على الهاتف الذي تناولناه آنفا في الجزء 3-4-1، تخيل أنه في كل مرة يحدث فيها تشويش على كلمة واحدة نقوم بإعادة إملاء الكلمات الألف التالية لها (بافتراض أن حجم النافذة 1000 كلمة). لاشك أن عملية إملاء الرسالة ستتأخر كثيراً بسبب تكرار الكلمات التي يعاد إرسالها بلا داعٍ.

كما يبدو من الاسم، يتفادى بروتوكول "الإعادة الانتقائية" (SR) أي إعادة إرسال غير ضرورية من المُرسِل، حيث يُعاد فقط إرسال تلك الرزم التي يشتبه المُرسِل في حدوث مشكلة في وصولها لدى المُستقبِل (أي وصلت بخطأ في البيانات أو فُقدت). تتطلب إعادة الإرسال الفردية (عند الحاجة) تلك أن يقوم المُستقبِل بإشعار المُرسِل باستلام الرزم التي تصل صحيحة كلٌّ على حدة بغض النظر عن ترتيبها. مرة أخرى سنستخدم نافذة حجمها N لوضع حد أقصى لعدد الرزم المنتظرة في خط الأنابيب بدون إشعار استلام. غير أنه بخلاف بروتوكول GBN، يكون المُرسِل قد تلقّى إشعارات استلام لبعض تلك الرزم الموجودة في النافذة. يوضح الشكل 3-23

فضاء الأرقام التسلسلية من منظور كلٍّ من المُرسِل والمُستقبِل، كما يبين الشكل 3-24 تفاصيل الإجراءات المختلفة التي تُتخذ من قِبَل المُرسِل في بروتوكول SR.

يقوم المُستقبِل في بروتوكول SR بالإشعار باستلام كل رزمة تصله بشكلٍ صحيح سواء كانت بالترتيب السليم أم لا. يتم تخزين الرزم التي تصل بغير الترتيب السليم بشكلٍ مؤقت لدى المُستقبِل إلى أن تصل أي رزم مفقودة (أي الرزم بأرقام تسلسلية أقل)، وعندئذٍ يتم توصيل جملة الرزم بكاملها بالترتيب السليم إلى الطبقة الأعلى. يحدد الشكل 3-25 الإجراءات المختلفة التي يتخذها المُستقبِل في بروتوكول SR، بينما يصور الشكل 3-26 مثالاً لكيفية عمل بروتوكول SR في حالة وجود رزم مفقودة. لاحظ أنه في الشكل 3-26 يقوم المُستقبِل في البداية بتخزين الرزم {3، 4، 5} بشكلٍ مؤقت، ثم يسلّمها سويةً مع الرزمة 2 إلى الطبقة الأعلى عند استلام الرزمة 2 المتأخرة في النهاية.

من المهم ملاحظة أنه في الخطوة 2 من الشكل 3-25، يعيد المُستقبِل إرسال إشعار باستلام (بدلاً من مجرد إهمال) الرزم التي تم استلامها بأرقام تسلسلية معينة أقل من رقم بداية النافذة الحالية. عليك أن تقنع نفسك بأن إشعار الاستلام هذا مطلوب بالفعل. تبعاً لفضاء الأرقام التسلسلية المبين في الشكل 3-23 للمُرسِل والمُستقبِل، على سبيل المثال إذا لم يكن هناك إشعار استلام للرزمة send_base المنتقلة من المُرسِل إلى المُستقبِل، فإن المُرسِل في النهاية سيعيد إرسال الرزمة send_base، بالرغم من أنه واضح (لنا، ولكن ليس للمُرسِل!) أن المُستقبِل قد استلم تلك الرزمة بالفعل. إذا لم يرسل المُستقبِل إشعاراً باستلام تلك الرزمة، فإن نافذة المُرسِل لن تتقدم للأمام أبداً! يوضح هذا المثال سمة مهمة من سمات البروتوكول SR (والعديد من البروتوكولات الأخرى أيضاً). لن يتوافر للمُرسِل والمُستقبِل دائماً نفس وجهة النظر فيما يتعلق بما تم استلامه بشكلٍ صحيح من عدمه، وهذا يعني أنه في بروتوكول SR لن تتطابق نافذتا كلٍّ من المُرسِل والمُستقبِل على الدوام.



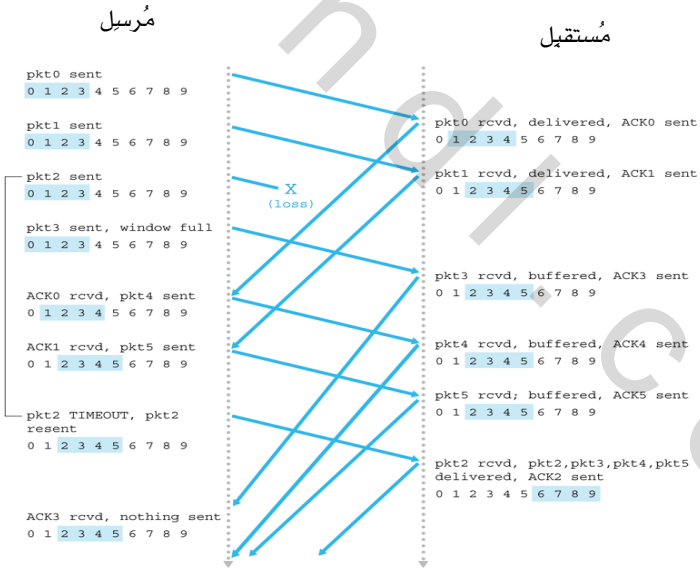
الشكل 3-23 الأرقام التسلسلية من منظور المُرسِل والمُستقبل في بروتوكول الإعادة الانتقائية (SR)

1. استلام البيانات من أعلى: عند استلام البيانات من أعلى، يحدد مُرسِل SR الرقم التسلسلي التالي المتوفر للرزمة. إذا كان الرقم التسلسلي ضمن نافذة المُرسِل، يتم وضع البيانات في رزمة وإرسالها؛ وإلا فإنها إما تخزن بشكلٍ مؤقت لدى المُرسِل أو تعاد إلى الطبقة الأعلى لتحاول إرسالها مرةً أخرى لاحقاً، كما هو الحال في بروتوكول GBN.
2. انقضاء فترة الموقّت: مرةً أخرى تُستخدم الموقّتات هنا أيضاً للحماية ضد فقد الرزم، غير أنه في هذه الحالة يتعين أن يكون لكل رزمة موقّتها المنطقي الخاص بها، حيث يتم إرسال رزمة واحدة فقط عند انقضاء فترة الموقّت. بالإمكان استخدام موقّت مادي واحد لمحاكاة استخدام عدة موقّتات منطقيّة [Varghese 1997].
3. تلقّي إشعار استلام: عندما يتلقى مُرسِل SR إشعار استلام، يؤشر المُرسِل على تلك الرزمة كرزمة تم استلامها بشرط أن تكون في حدود النافذة. إذا كان الرقم التسلسلي للرزمة يساوي رقم قاعدة النافذة send_base يتم تحريك قاعدة النافذة للأمام إلى الرزمة ذات أقل رقم تسلسلي ولم يصل إشعار استلامها بعد. وإذا تحركت النافذة وكانت هناك رزم لم تُرسل ولها أرقام تسلسلية تقع ضمن النافذة الحالية فإنه يتم إرسال تلك الرزم.

الشكل 3-24 الأحداث والإجراءات لدى المُرسِل في بروتوكول الإعادة الانتقائية (SR).

1. استلام رزمة برقم تسلسلي في المدى [rcv_base, rcv_base + N - 1] بشكلٍ صحيح: في هذه الحالة تقع الرزمة المستلَمة ضمن نافذة المُستقبِل، وترسل حزمة إشعار استلام انتقائية إلى المُرسِل، وإذا لم يكن قد تم استلام تلك الرزمة من قبل فإنه يتم تخزينها بشكلٍ مؤقت. وإذا كانت الرزمة لها رقم تسلسلي يساوي رقم قاعدة نافذة المُستقبِل (rcv_base في شكل 3-22)، فإن هذه الرزمة وأي رزم أخرى تم تخزينها مؤقتاً في السابق ولها أرقام تسلسلية متوالية (تبدأ بـ rcv_base) يتم توصيلها إلى الطبقة الأعلى. بعد ذلك يتم تحريك نافذة الاستقبال للأمام بعدد الرزم التي سُلّمت للطبقة الأعلى. كمثال خذ في الاعتبار شكل 3-26. عندما يتم استلام رزمة برقم تسلسلي يساوي rcv_base = 2 ، يمكن توصيل تلك الرزمة مع الرزم 3، و4، و5 للطبقة الأعلى.
2. استلام رزمة برقم تسلسلي في المد [rcv_base - N, rcv-base - 1] بشكلٍ صحيح: في هذه الحالة يجب إرسال إشعار استلام، بالرغم من أن هذه الرزمة تم الإشعار باستلامها من قبل.
3. كل الحالات الأخرى: أهمل الرزمة الواصلة.

الشكل 3-25 الأحداث والإجراءات لدى المُستقبِل في بروتوكول الإعادة الانتقائية (SR).

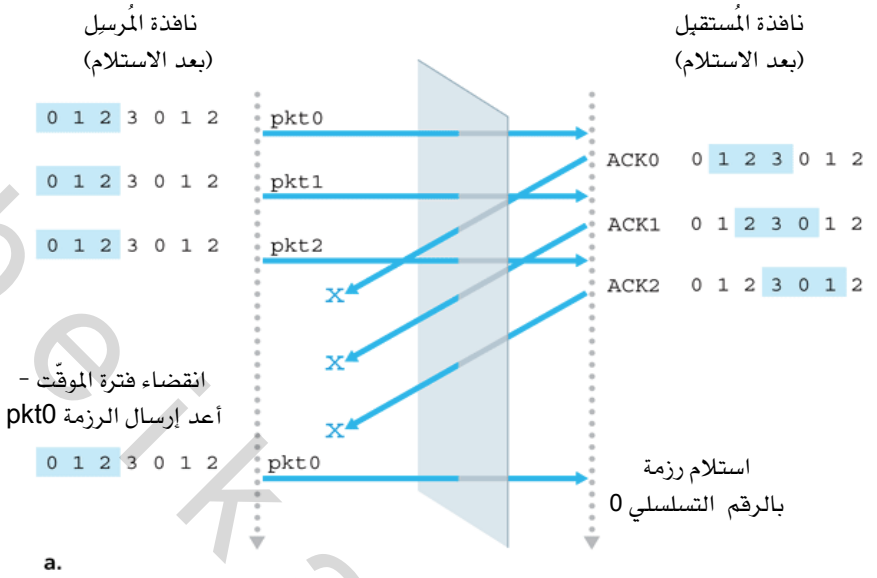


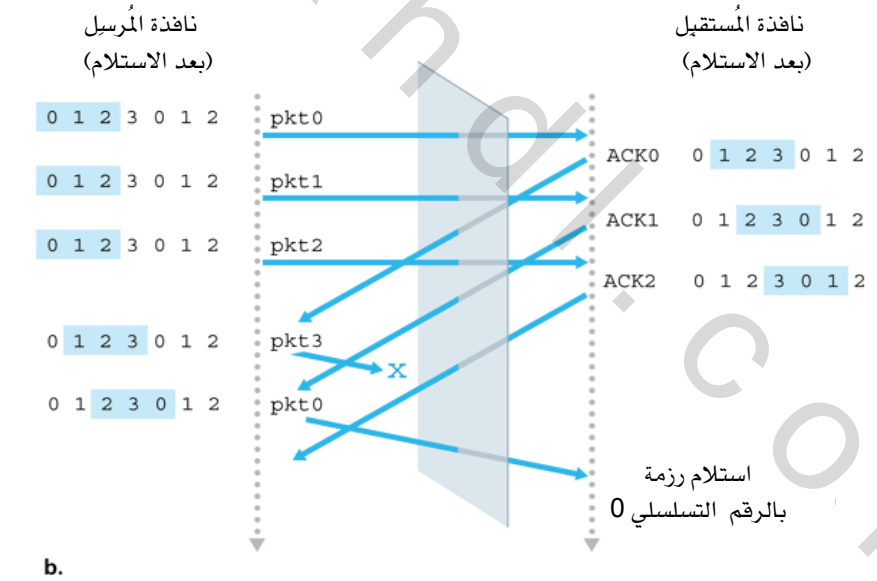
الشكل 3-26 العمل ببروتوكول الإعادة الانتقائية (SR).

يترتب على غياب التزامن بين نافذتي المُرسِل والمُستقبِل نتائج مهمة عندما تواجهنا حقيقة أن مدى الأعداد التسلسلية محدود. خذ بعين الاعتبار ما يمكن أن يحدث على سبيل المثال مع مدى محدود من أربعة أرقام تسلسلية هي {0، 1، 2، 3} وحجم نافذة $N = 3$. افترض أن الرزم من 0 إلى 2 تم إرسالها واستلامها بشكلٍ صحيح وكذلك إرسال إشعارات الاستلام الخاصة بها من قِبَل المُستقبِل. في هذه اللحظة تكون نافذة المُستقبِل تغطي الرزم الرابعة والخامسة والسادسة والتي لها الأرقام التسلسلية 3، 0، 1 على التوالي. لنعتبر السيناريوهين التاليين الآن: في السيناريو الأول والموضح في الشكل 3-27 (a)، تفقد إشعارات الاستلام الخاصة بالرزم الثلاث الأولى ومن ثم يضطر المُرسِل لإعادة إرسال تلك الرزم. وهكذا فإنه في هذه الحالة يتلقى المُستقبِل رزمة برقم تسلسلي 0 – والتي هي نسخة من الرزمة الأولى التي أُرسلت من قبل.

في السيناريو الثاني والموضح في الشكل 3-27 (b)، تصل إشعارات الاستلام للرزم الثلاثة الأولى جميعها بشكلٍ صحيح. وهكذا يدفع المُرسِل بنافذته إلى الأمام ويرسل الرزم الرابعة والخامسة والسادسة بالأرقام التسلسلية {3، 0، 1} على التوالي. تُفقد الرزمة التي تحمل الرقم التسلسلي 3، لكن تصل الرزمة بالرقم التسلسلي 0؛ وهي رزمة تحتوي على بيانات جديدة.

لنأخذ في الاعتبار الآن وجهة نظر المُستقبِل كما في الشكل 3-27، والذي يتضمن ستارة رمزية بين المُرسِل والمُستقبِل، لتذكيرنا بأن المُستقبِل لا يستطيع "رؤية" الإجراءات التي يتخذها المُرسِل. كل ما يلاحظه المُستقبِل هو سلسلة الرسائل التي يتسلمها من القناة ويرسلها إلى القناة. من وجهة نظر المُستقبِل السيناريوهان في الشكل 3-27 متماثلان، فليس لديه طريقة للتمييز ما بين إعادة إرسال الرزمة الأولى والإرسال الأصلي للرزمة الخامسة. واضح أن حجم النافذة الذي يقل بـ 1 عن عدد الأرقام التسلسلية الممكنة لن يعمل بشكلٍ صحيح. ولكن إلى أي حد يحتاج حجم النافذة لأن يكون صغيراً ؟ يُطلب منك في تمرين في نهاية هذا الفصل إثبات أنه في حالة بروتوكول SR يجب أن يكون حجم النافذة أقل من أو يساوي نصف عدد الأرقام التسلسلية الممكنة.





الشكل 3-27 معضلة المُستقبِل في بروتوكول الإعادة الانتقائية (SR): هل الرزمة الواصلة رزمة جديدة أم إعادة إرسال؟

ستجد في موقع الويب المصاحب لهذا الكتاب برنامج جافا صغير يحاكي تشغيل بروتوكول SR. حاول إجراء نفس التجارب التي قمت بها باستخدم بروتوكول GBN لترى ما إذا كانت النتائج تتوافق مع توقعاتك.

بهذا نكون قد أكملنا مناقشتنا لبروتوكولات النقل الموثوق للبيانات. لقد غطّينا كمية كبيرة من المادّة العلمية! واستعرضنا العديد من الآليات التي توفر – مجتمعةً – نقلاً موثوقاً للبيانات. يلخص الجدول 3-1 تلك الآليات. والآن وقد رأينا كل هذه الآليات وهي تعمل وأصبح بوسعنا رؤية "الصورة الكبيرة"، نحثك على مراجعة هذا الجزء ثانيةً لترى كيف تم إضافة تلك الآليات الواحدة تلو الأخرى لتغطي نماذج عملية بمستويات متزايدة من التعقيد لقناة الاتصال التي تربط المُرسِل بالمُستقبِل أو لتحسين أداء البروتوكولات.

دعنا ننهي استعراضنا لبروتوكولات النقل الموثوق للبيانات بتناول الفرضية المتبقية في نموذج قناة اتصالنا التحتية. تذكّر أننا افترضنا أن الرزم لا يمكن إعادة ترتيبها ثانيةً ضمن قناة الاتصال بين المُرسِل والمُستقبِل. تعتبر هذه فرضية معقولة عموماً عندما يكون المُرسِل مربوطاً بالمُستقبِل بواسطة وصلة مادية واحدة. ولكن عندما تكون "القناة" الموصّلة بين الاثنين شبكة، يمكن أن يحدث إعادة ترتيب للرزم. من مظاهر إعادة ترتيب الرزم أن نسخاً قديمة من رزمة برقم تسلسلي أو رقم إشعار استلام x يمكن أن تظهر، رغم أنه لا نافذة المُرسِل ولا نافذة المُستقبِل تتضمن x. مع إعادة ترتيب الرزم يمكن اعتبار قناة الاتصال ببساطة ومن حيث المبدأ كمكان يخزّن الرزم بشكلٍ مؤقت ثم يبعث بها بشكلٍ عفوي في أي لحظة في المُستقبل. نظراً لأن الأرقام التسلسلية قد تستعمل ثانيةً فإنه ينبغي توخي بعض الحذر في التعامل مع مثل تلك الرزم المزدوجة. تتلخص الطريقة المتبعة في الواقع العملي في تجنب استخدام رقم تسلسلي x مرة ثانية إلا بعد أن يتأكد المُرسِل من أن الرزم المُرسَلة سابقاً بالرقم التسلسلي x لم تعد موجودة في الشبكة. يتم ذلك بافتراض أن الرزمة لا تستطيع "العيش" في الشبكة لأطول من مدة زمنية قصوى ثابتة. تفترض امتدادات بروتوكول TCP للشبكات عمراً أقصى للرزمة قيمته حوالى 3 دقائق [RFC 1323]. يصف [Sunshine 1978] طريقة لاستعمال الأرقام التسلسلية تتفادى تماماً مشاكل إعادة الترتيب.

الآلية	الاستخدام، ملاحظات
المجموع التدقيقي	يُستخدم لاكتشاف أخطاء البتات التي تطرأ على الرزمة المُرسَلة أثناء انتقالها.
الموقّت	يُستخدم لإعادة إرسال رزمة إثر انقضاء فترة تأخير محددة، ربما بسبب فقد الرزمة (أو إشعار استلامها) على القناة. نظراً لأن انقضاء الفترة يمكن أن يحدث عندما تكون الرزمة قد تأخرت فقط دون أن تُفقد (انقضاء قبل الأوان)، أو عندما تكون الرزمة قد استُلمت من قِبَل المُستقبل ولكن إشعار استلامها المبعوث من المُستقبل إلى المُرسِل قد فقد، فإنه يمكن بهذا الأسلوب أن يتلقّى المُستقبل نسخاً مزدوجة من نفس الرزمة.
الأرقام التسلسلية للرزم	تُستخدم للترقيم التسلسلي لرزم البيانات التي تتدفّق من المُرسِل إلى المُستقبل. تسمح الفجوات التي قد تحدث في أرقام الرزم التي يتم استلامها للمُستقبل باكتشاف فقد الرزم. يسمح وصول الرزم بأرقام تسلسلية مكرّرة للمُستقبل باكتشاف النسخ المكرّرة من رزمة.
إشعارات الاستلام	يتم إرسالها من المُستقبل إلى المُرسِل لإخباره بأن رزمة أو مجموعة رزم قد تم استلامها بشكلٍ صحيح. غالباً ما تحمل الإشعارات الرقم التسلسلي للرزمة أو الرزم المراد الإشعار باستلامها. قد تكون تلك الإشعارات فردية أو تراكمية حسب البروتوكول المستخدم.
إشعارات الاستلام السلبية	يتم إرسالها من المُستقبل إلى المُرسِل لإخباره بأن رزمة لم يتم استلامها بشكلٍ صحيح. غالباً ما تحمل الإشعارات السلبية الرقم التسلسلي للرزمة المراد الإشعار بعدم استلامها بشكلٍ صحيح.
النافذة، خط الأنابيب	تستخدم للحَدّ من نشاط المُرسِل بحيث يرسل فقط رزماً بأرقام تسلسلية تقع ضمن مدى محدد (داخل النافذة). بالسماح بإرسال عدة رزم بدون انتظار إشعار باستلامها، يمكن تحسين معدل استغلال المُرسِل مقارنةً ببروتوكول التوقف والانتظار. سنرى بعد قليل أن مقاس النافذة يمكن ضبطه على أساس قدرة المُستقبل على استقبال الرسائل و تخزينها مؤقتاً عنده، أو على مستوى الازدحام في الشبكة، أو على كليهما.

الجدول 3-1 ملخص بآليات النقل الموثوق للبيانات واستخداماتها.

5-3 بروتوكول النقل التوصيلي: TCP

سنرى في هذا الجزء أنه لتوفير نقل موثوق للبيانات، يعتمد بروتوكول TCP على العديد من المبادئ الأساسية التي تناولناها في الجزء السابق، بما في ذلك

اكتشاف الأخطاء، وإعادة الإرسال، وإشعارات الاستلام التراكمية، والموقّتات، وحقول ترويسة الرزمة الخاصة بالأرقام التسلسلية للرزم وأرقام إشعارات الاستلام. تم تعريف بروتوكول TCP من خلال RFC 2581 وRFC 2018 وRFC 1323 و RFC 1122 وRFC 793.

تاريخ حالة (Case History)

فينتون سيرف، وروبرت كاهن، وبروتوكول TCP/IP

في أوائل السبعينيات بدأت شبكات تحويل الرزم في الانتشار، حيث كانت ARPAnet (سلف الإنترنت) مجرد واحدة من الشبكات الموجودة في ذلك الوقت. كان لكل شبكة من تلك الشبكات بروتوكولاتها الخاصة بها. تنبه الباحثان فينتوني سيرف وروبرت كاهن إلى أهمية ربط تلك الشبكات معاً، واخترعا بروتوكولاً للتوصيل بين الشبكات أطلقا عليه اسم TCP/IP (بروتوكول التحكم في الإرسال/بروتوكول الإنترنت). رغم أن سيرف وكاهن كانا في البداية يعتبران هذا البروتوكول كياناً واحداً، إلا أنه تم فصله فيما بعد إلى جزأين يعملان بشكلٍ مستقل هما TCP وIP. في مايو 1974 نشر سيرف وكاهن بحثاً عن بروتوكول TCP/IP في مجلة IEEE لتقنية الاتصالات [Cerf 1974].

لقد ابتُكر بروتوكول TCP/IP – والذي يُعتبر اليوم لُحمة الإنترنت وسداتها – قبل ظهور الحاسب الشخصي ومحطات العمل وانتشار الإيثرنت وتقنيات الشبكة المحلية الأخرى وقبل الويب ونقل الصوت المستمر والدردشة الإلكترونية. لقد أدرك سيرف وكاهن الحاجة إلى بروتوكول لربط الشبكات يوفر من ناحية دعماً واسعاً لتطبيقات لم تحدد بعد، ومن ناحية أخرى يسمح للمضيفات وبروتوكولات ربط البيانات الاختيارية بالعمل معاً على ما يرام.

في عام 2004 نال سيرف وكاهن جائزة Turing من منظمة ACM، والتي تعتبر بمثابة "جائزة نوبل في مجال الحاسبات" تقديراً لعملهما الرائد في تربيط الشبكات (internetworking)، بما في ذلك تصميم وإنجاز بروتوكولات الاتصال الأساسية للإنترنت TCP/IP، ولريادتهما الملهمة في مجال ربط الشبكات.

1-5-3 توصيلة بروتوكول TCP

يقال عن بروتوكول TCP أنه مبني على توصيلة، ذلك لأنه قبل أن تستطيع عملية تطبيق (application process) البدء في إرسال بيانات لعملية أخرى يتعين على العمليتين أولاً إجراء "مصافحة" بينهما. يتم خلال تلك المصافحة تبادل قطع بيانات تمهيدية لتحديد القيم الأولية للمتغيرات الخاصة بنقل البيانات بينهما. كجزء من إجراءات إنشاء التوصيلة في بروتوكول TCP، يقوم كلٌّ من الطرفين بوضع القيم الأوّلية للعديد من متغيرات الحالة (state variables) المتعلقة بتوصيلة البروتوكول (والتي سنتناول الكثير منها في هذا الجزء وفي الجزء 3-7).

جدير بالذكر أن "توصيلة" TCP التي نعنيها هنا ليست دائرة TDM أو FDM من طرف إلى طرف كما في شبكات تحويل الدوائر. كما أنها ليست أيضاً دائرة افتراضية (راجع الفصل الأول)، حيث توجد حالة التوصيلة بالكامل في النظامين الطرفيين. ولأن بروتوكول TCP يجري تشغيله فقط في الأنظمة الطرفية، وليس في الكيانات المتوسطة داخل الشبكة (كالموجّهات ومحوّلات طبقة ربط البيانات)، فإن كيانات الشبكة المتوسطة لا تحتفظ بحالة توصيلة TCP. تكون الموجّهات المتوسّطة في الواقع غافلة تماماً عن توصيلات بروتوكول TCP، فتلك الموجّهات ترى وحدات بيانات وليس توصيلات.

توفر توصيلة TCP خدمة اتصال كامل الازدواج في الاتجاهين (full-duplex)، ففي وجود توصيلة TCP بين العملية A على مضيف ما والعملية B على مضيف آخر، فإن بيانات طبقة التطبيقات يكون بوسعها التدفق من العملية A إلى العملية B في نفس الوقت الذي تتدفق فيه بيانات طبقة التطبيقات من عملية B إلى عملية A. كما أن توصيلة TCP هي دائماً من نقطة إلى نقطة (point-to-point)، بمعنى أنها تكون بين مُرسِل واحد ومُستقبِل واحد. أي أن الاتصال بعدة جهات في نفس الوقت (multicasting) (انظر الجزء 4-7) – حيث تتدفق البيانات من مُرسِل واحد إلى عدة مُستقبلين في عملية إرسال واحدة – غير ممكن في بروتوكول TCP.

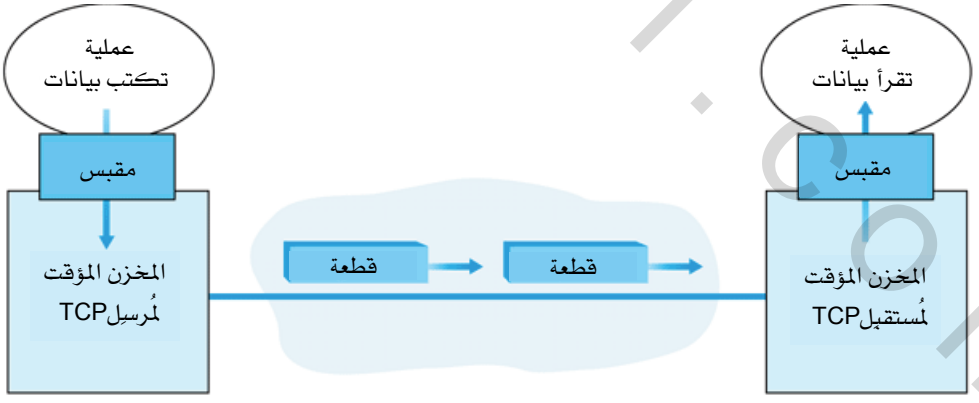
دعنا الآن نلقي نظرة على الكيفية التي يتم بها إنشاء توصيلة TCP. افترض أن عملية يجري تشغيلها على مضيفٍ ما وتود بدء توصيلة مع عملية أخرى على مضيفٍ آخر. تذكّر أن العملية التي تبدأ التوصيلة تُدعى عملية الزبون، بينما يطلق على العملية الأخرى عملية الخادم. في البداية تخبر عملية تطبيق الزبون طبقة النقل على الزبون بأنها تريد إنشاء توصيلة مع عملية الخادم. تذكّر من الجزء 2-7 أن برنامج زبون بلغة جافا يقوم بذلك بإصدار الأمر:

```
Socket clientSocket = new Socket("hostname", portNumber);
```

حيث hostname هو اسم الخادم وportNumber هو رقم المنفذ الذي يميّز العملية على الخادم. تمضي طبقة النقل على الزبون بعد ذلك في إنشاء توصيلة TCP مع بروتوكول TCP على الخادم. في نهاية هذا الجزء سنناقش بشيء من التفصيل إجراءات إنشاء توصيلة TCP. يكفي الآن معرفة أن الزبون يرسل قطعة بيانات TCP خاصة، فيرد الخادم بقطعة بيانات TCP خاصة ثانية، ويرد الزبون أخيراً بقطعة خاصة ثالثة. القطعتان الأوليان لا تحملان أي بيانات لطبقة التطبيقات بينما قد تحمل القطعة الثالثة بيانات لطبقة التطبيقات. نظراً لأنه يتم إرسال ثلاث قطع بيانات بين المضيفين، غالباً ما يطلق على هذا الإجراء لإنشاء توصيلة "إجراء المصافحة الثلاثية".

بمجرد إنشاء توصيلة TCP يصبح بوسع عمليتي التطبيق إرسال البيانات إلى بعضهما البعض. دعنا نأخذ في الاعتبار إرسال البيانات من عملية الزبون إلى عملية الخادم. تقوم عملية الزبون بتمرير سيل البيانات عبر المقبس (بوابة العملية)، كما سبق وصفه في الجزء 2-7. بمجرد مرور البيانات من خلال البوابة تصبح تحت تصرف بروتوكول TCP الذي يجري تشغيله على مضيف الزبون. كما هو موضح في الشكل 3-28، يوجّه TCP تلك البيانات إلى المخزن المؤقت للإرسال، وهو أحد المخازن المؤقتة التي تم إعدادها أثناء عملية المصافحة الثلاثية. من حين لآخر يلتقط نظام TCP كتلة بيانات من المخزن المؤقت لدى المُرسِل ليقوم ببثها. بشكلٍ مثير للانتباه، تعتبر مواصفات بروتوكول TCP [RFC-793] متساهلة جداً فيما يتعلق بتحديد متى يقوم TCP على المُرسِل بإرسال البيانات الموجودة في المخزن المؤقت

لديه، حيث تنص على أنه على البروتوكول أن "يرسل تلك البيانات على شكل قطع في الوقت الذي يراه مناسباً". الحد الأقصى من البيانات الذي يمكن تناوله ووضعه في قطعة يحكمه الحجم الأقصى للقطعة MSS، والذي يتم تحديده أولاً بتعيين الطول الأقصى لإطار طبقة ربط البيانات الذي يمكن لمضيف الإرسال المحلي بثه (وهو ما يسمى بوحدة الإرسال القصوى MTU، بعد ذلك تُحدّد MSS بحيث يمكن استيعاب قطعة بيانات TCP (بعد تغليفها في وحدة بيانات IP) في إطار واحد من إطارات طبقة ربط البيانات. القيم المشهورة للمتغير MTU هي 1,460 بايتاً، 536 بايتاً، و512 بايتاً. تم اقتراح بعض الطرق لتعيين قيمة MTU للمسار – أي إطار طبقة ربط البيانات الأكبر الذي يمكن أن يرسل على كل الوصلات التي ستُستخدم من المصدر إلى الوجهة [RFC 1191]، ومن ثم تحديد قيمة MSS على أساسه. لاحظ أن قيمة MSS هي أقصى كمية لبيانات طبقة التطبيقات في القطعة، وليست الحجم الأقصى لقطعة بيانات TCP والتي تشمل علاوةً على ذلك ترويسة القطعة وحقولها الأخرى. (قد تتسبب المصطلحات هنا في بعض الخلط، ولكن علينا أن نتعايش مع ذلك نظراً للتغلغل الشديد لتلك المصطلحات في أدبيات الشبكات والإنترنت).



الشكل 3-28 مخازن TCP المؤقتة للإرسال والاستقبال.

يضيف بروتوكول TCP لكل كتلة من بيانات الزبون ترويسة TCP الخاصة بها لتكوين قطعة بيانات TCP، ثم يدفع بتلك القطع لأسفل إلى طبقة الشبكة، حيث تغلّف كل قطعة على حدة لتكوين وحدات بيانات طبقة الشبكة. يتم إرسال وحدات البيانات تلك خلال الشبكة. عندما يتسلم بروتوكول TCP على مضيف الوجهة قطعة TCP، توضع البيانات المستخلصة من القطعة في مخزن الاستقبال المؤقت التابع لتوصيلة TCP، كما هو موضح في الشكل 3-28، حيث يقرؤها تطبيق الوجهة من ذلك المخزن. لكل جانب من جانبي الاتصال مخزن مؤقت خاص به لكلٍ من الإرسال والاستقبال. يمكنك تشغيل برنامج جافا الخاص بضبط التدفق وذلك بزيارة الموقع http://www.awl.com/kurose-rose والذي يشرح باستخدام الصور المتحركة عمل المخازن المؤقتة للإرسال والاستقبال.

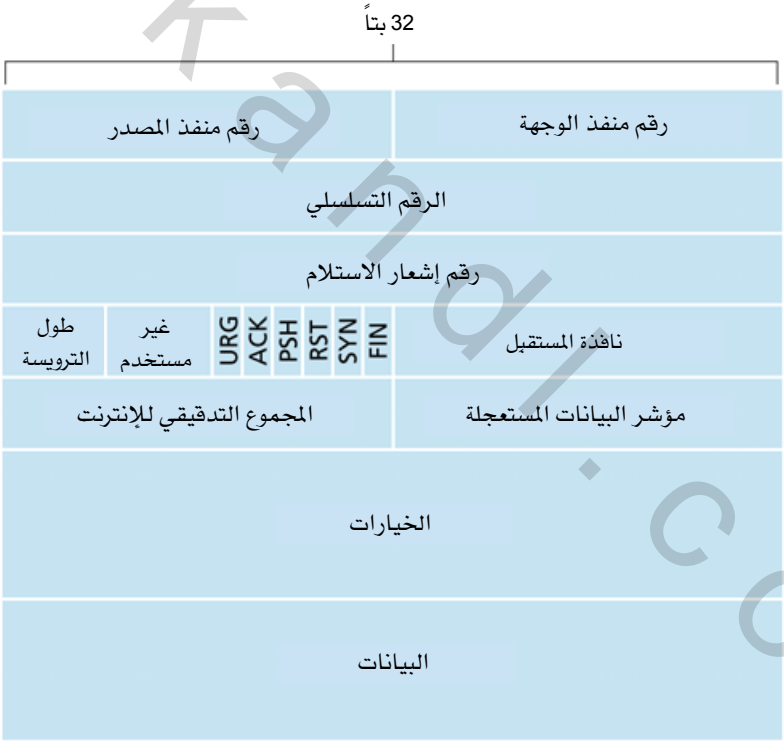
نرى من هذا العرض أن توصيلة TCP تشتمل على مخازن مؤقتة لبيانات المستخدم ومتغيرات ومقبس توصيل لعملية تطبيق على مضيف، بالإضافة إلى مجموعة أخرى من المخازن والمتغيرات ومقبس للعملية المناظرة على المضيف الآخر. كما ذكرنا سابقاً لا يتم تخصيص أي مخازن مؤقتة أو متغيرات تتعلق بتوصيلة TCP في الكيانات المتوسطة بالشبكة (كالموجّهات والمحوّلات والمكرّرات (repeaters) والموجودة بين المضيفين المتصلين.

3-5-2 صيغة قطعة بيانات TCP

بعد أن ألقينا نظرة سريعة على توصيلة TCP دعنا نفحص تركيب قطعة TCP. تتألف قطعة TCP من حقول ترويسة وحقل بيانات (الحمل الآجر) يضم حقل البيانات كتلة بيانات التطبيق المُرسَلة. كما ذكرنا أعلاه يكون الحجم الأقصى لحقل البيانات في القطعة محدوداً بالمتغير MSS. عندما يرسل بروتوكول TCP ملفاً كبيراً، كصورة تمثل جزءاً من صفحة ويب، يقوم البروتوكول عادةً بتجزيء الملف إلى كتل حجم كلٍّ منها MSS (باستثناء الكتلة الأخيرة التي غالباً ما تكون أقل من MSS). غير أن التطبيقات التفاعلية غالباً ما ترسل كتل بيانات أصغر حجماً من MSS. على سبيل المثال في تطبيقات الدخول على الحاسبات عن بعد، مثل Telnet،

غالباً ما يكون حقل البيانات في قطعة TCP بايتاً واحداً فقط. ونظراً لأن ترويسة قطعة TCP تتكون عادةً من 20 بايتاً (أطول من ترويسة قطعة UDP بـ 12 بايتاً) فإن قطع TCP التي يرسلها تطبيق Telnet قد يقتصر طولها على 21 بايتاً فقط.

يبين الشكل 3-29 صيغة قطعة بيانات بروتوكول TCP. كما هو الحال في بروتوكول UDP تتكون ترويسة القطعة من حقول لرقم منفذ المصدر ورقم منفذ الوجهة، والتي تستخدم في تجميع البيانات من تطبيقات الطبقة الأعلى وتوزيعها عليها. كما في UDP تتضمن الترويسة أيضاً حقل المجموع التدقيقي. وعلاوة على ذلك تتضمن ترويسة قطعة بيانات TCP الحقول الإضافية التالية:



الشكل 3-29 صيغة قطعة بيانات بروتوكول TCP.

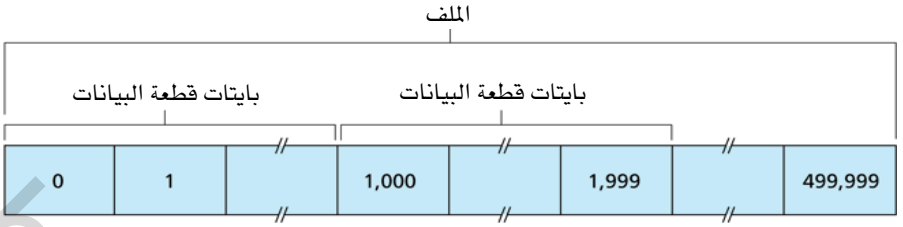
- حقل الرقم التسلسلي للقطعة بطول 32 بتاً وحقل رقم إشعارالاستلام بطول 32 بتاً، والتي يستخدمها مُرسِل ومُستقبل TCP لتوفير خدمة نقل موثوق للبيانات كما سيتم تفصيله لاحقاً.
- حقل نافذة المُستقبل بطول 16 بتاً والمستخدم في ضبط التدفق. وسنرى بعد قليل أنه يستخدم لبيان عدد البايتات التي يمكن للمُستقبل أن يقبلها.
- حقل طول الترويسة بطول 4 بتات ويحدد طول ترويسة قطعة بيانات TCP بالكلمات (words) (الكلمة = 32 بتاً). يمكن أن يكون طول ترويسة TCP متغيراً بسبب وجود حقل خيارات TCP (TCP options). غالباً ما يكون حقل الخيارات خالياً (أي غير مستخدم)، وفي هذه الحالة يكون طول ترويسة TCP هو الطول المعتاد أي 20 بايتاً).
- حقل الخيارات الاختياري، وهو حقل متغير الطول يُستخدم عندما يقوم المُرسِل والمُستقبل بالتفاوض بخصوص الحجم الأقصى للقطعة MSS أو كمعامل تكبير لمقاس النافذة للاستخدام مع الشبكات عالية السرعة. تم أيضاً تعريف خيار تضمين خاتم الوقت (time stamp). راجع RFC 854 وRFC 1323 للمزيد من التفاصيل.
- حقل الأعلام (أو المؤشرات) ويضم 6 بتات. تستخدم بت إشعار الاستلام (ACK) للإشارة إلى أن القيمة الموجودة في حقل رقم إشعار الاستلام هي قيمة حقيقية؛ أي أن القطعة تتضمن إشعاراً باستلام قطعة قد وصلت بنجاح. تُستخدم البتات RST وSYN وFIN لبدء وإنهاء توصيلة TCP كما سنبين في نهاية هذا الجزء. عند اختيار القيمة 1 للبت PSH، يكون على المُستقبل تمرير البيانات الواصلة إلى الطبقة الأعلى فوراً. وأخيراً تُستخدم البت URG للإشارة إلى أن هذه القطعة تتضمن بيانات قد علّمها كيان الطبقة الأعلى في جانب المُرسِل كبيانات "مستعجلة" (urgent). يشير حقل مؤشر البيانات المستعجلة (بطول 16 بتاً) إلى موقع البايت الأخيرة في تلك البيانات. يتعين على بروتوكول TCP إخبار كيان الطبقة الأعلى على جانب المُستقبل عند وجود بيانات مستعجلة مُرسَلة، وإرسال مؤشر يحدد نهاية تلك البيانات

المستعجلة في القطعة. (في الواقع العملي لا تُستخدم بتات الأعلام PSH وURG ولا حقل مؤشر البيانات المستعجلة، ومع ذلك فنحن نذكرها هنا لاستكمال الصورة).

الأرقام التسلسلية لقطع البيانات وأرقام إشعارات الاستلام

يُعدّ الرقم التسلسلي للقطعة ورقم إشعار الاستلام اثنين من أهم حقول ترويسة قطعة بيانات TCP، حيث يمثّلان جزءاً أساسياً من خدمة TCP للنقل الموثوق للبيانات. لكن قبل مناقشة كيفية استخدام هذين الحقلين في تحقيق نقل موثوق للبيانات، دعنا نوضّح أولاً ما الذي يضعه بروتوكول TCP في هذين الحقلين بالضبط.

ينظر بروتوكول TCP للبيانات على أنها سَيل غير مهيكل ولكنه مرتب من البايتات. يعكس استعمال البروتوكول للأرقام التسلسلية وجهة النظر هذه، حيث تمثل تلك الأرقام سلسلة البايتات المُرسَلة وليس سلسلة قطع البيانات المُرسَلة. وبالتالي فإن الرقم التسلسلي الذي تحمله قطعة بيانات هو رقم البايتات التسلسلي لأول بايت بيانات في القطعة. لنأخذ مثالاً. افترض أن عمليةً على المضيف A تريد إرسال سيلٍ من البيانات إلى عملية على المضيف B عبر توصيلة TCP. سيقوم بروتوكول TCP على المضيف A بترقيم كل بايت في سيل البيانات ضمنياً. افترض أن سيل البيانات يتألف من ملف يضم 500000 بايتاً، وأن الحجم الأقصى للقطعة MSS هو 1000 بايت، وأن البايت الأولى في سيل البيانات قد أعطيت الرقم 0. كما هو مبين في الشكل 3-30، يكوّن بروتوكول TCP 500 قطعة من سيل البيانات. وتُعطى القطعة الأولى الرقم التسلسلي 0، والقطعة الثانية الرقم التسلسلي 1000، والقطعة الثالثة الرقم التسلسلي 2000، وهكذا. يوضع كل رقم تسلسلي في حقل الرقم التسلسلي في ترويسة قطعة بيانات TCP المناظرة.



الشكل 3-30 تقسيم بيانات ملف إلى قطع بيانات TCP.

دعنا الآن نأخذ في الاعتبار أرقام إشعارات الاستلام. هذه أصعب بعض الشيء من الأرقام التسلسلية لقطع البيانات. تذكّر أن بروتوكول TCP يعتمد أسلوب الإرسال كامل الازدواج، ولذا فإنه في الوقت الذي يرسل المضيف A البيانات إلى المضيف B يمكنه أيضاً استلام البيانات من المضيف B (كجزء من توصيلة TCP نفسها). تحمل كل قطعة بيانات تصل من المضيف B رقماً تسلسلياً للبيانات التي تتدفق من B إلى A. رقم إشعار الاستلام الذي يضمّنه المضيف A في قطعة بياناته التي يرسلها هو الرقم التسلسلي لبايت البيانات التالية التي يتوقع وصولها من المضيف B. من المفيد النظر إلى بعض الأمثلة لفهم ماذا يجري هنا. لنفترض أن المضيف A قد تسلّم من B كل البايتات المرقمة من 0 إلى 535، وأن A على وشك إرسال قطعة بيانات إلى B. ينتظر المضيف A وصول بايت رقم 536 وكل البايتات التي تليها في سيل البيانات لدى المضيف B. وبالتالي يضع المضيف A الرقم 536 في حقل رقم إشعار الاستلام بقطعة البيانات التي يرسلها إلى B.

وكمثالٍ آخر افترض أن المضيف A قد تسلّم قطعة بيانات واحدة من المضيف B تحتوي على البايتات من 0 إلى 535 وقطعة أخرى تحتوي على البايتات من 900 إلى 1000. لسبب ما لم يتسلّم المضيف A بعد البايتات من 536 إلى 899. في هذا المثال لا يزال المضيف A ينتظر البايت 536 (وما بعدها) لكي يتمكن من إعادة تركيب سلسلة البيانات من A عند وصولها كاملةً لديه. وعليه فإن القطعة التالية من A التي سيرسلها إلى B ستحتوي على 536 في حقل رقم إشعار الاستلام. ونظراً لأن بروتوكول TCP يرسل إشعارات استلام للبايتات لغاية أول بايت مفتقَد ضمن سلسلة البيانات الجاري استقبالها، يقال: إن TCP يستخدم إشعارات استلام تراكمية.

يثير هذا المثال الأخير نقطة مهمة ولكنها دقيقة. افترض أن المضيف A استلم القطعة الثالثة (أي البايتات من 900 إلى 1000) قبل استلام القطعة الثانية (أي البايتات من 536 إلى 899)! ماذا يمكن للمضيف عمله عندما يتسلم قطع بيانات بغير ترتيبها السليم من خلال توصيلة TCP؟ بشكلٍ مثير للانتباه لا تفرض طلبات التعليقات (RFCs) الخاصة ببروتوكول TCP أي قواعد هنا وإنما تترك القرار لمبرمجي النسخة المعنية من البروتوكول. هناك خياران أساسيان: إما أن (1) يهمل المُستقبِل فوراً قطع البيانات الواصلة بغير الترتيب السليم (كما ناقشنا سابقاً، مما يسهم في تبسيط تصميم المُستقبِل) أو (2) يحتفظ المُستقبِل بقطع البيانات الواصلة بغير الترتيب السليم وينتظر وصول القطع المفتقدة لملء الفجوات في البيانات التي يتم استلامها. واضح أن الخيار الأخير أكثر كفاءة من ناحية استغلال الحيّز الترددي للشبكة، وهو الخيار المعمول به في الواقع.

لاحظ أننا في الشكل 3-30 افترضنا أن الرقم التسلسلي الأولي هو 0. في الواقع، يختار كلا الجانبين من توصيلة TCP ذلك الرقم الأول بشكلٍ عشوائي، وذلك لتقليل احتمال وجود قطعة ما في الشبكة من توصيلة سابقة تم إنهاؤها بالفعل بين نفس المضيفين واعتبارها كقطعة صحيحة في توصيلة جديدة بين نفس المضيفين (وتصادف أيضاً أنهما يستعملان نفس أرقام المنافذ المستخدمة في التوصيلة السابقة) [Sunshine 1978].

بروتوكول Telnet: مثال عن الأرقام التسلسلية وأرقام إشعارات الاستلام

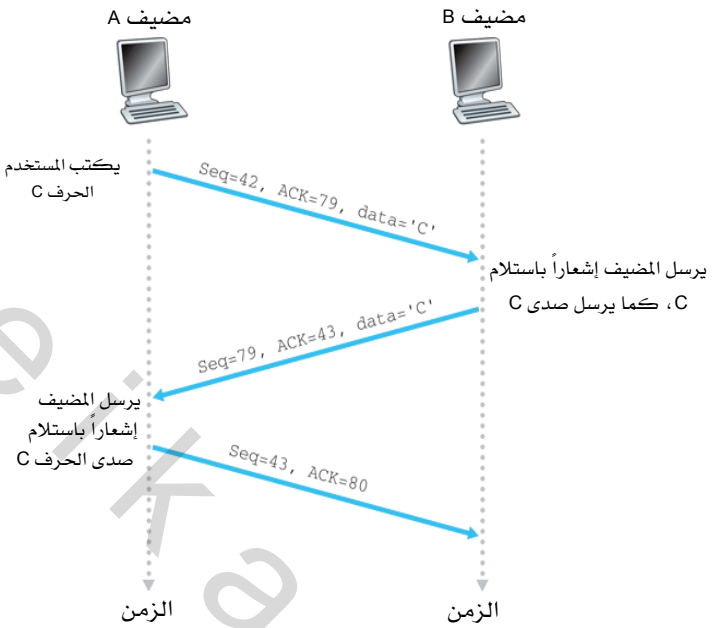
بروتوكول تيلنت (Telnet) هو بروتوكول شهير لطبقة التطبيقات، تم تعريفه بطلب التعليقات RFC 854، ويُستخدم للدخول على الحاسبات عن بعد. يعمل هذا البروتوكول فوق بروتوكول TCP، وهو مصمم للعمل بين أي زوج من المضيفات. خلافاً لمعظم تطبيقات نقل البيانات التي تناولناها في الفصل الثاني، فإن تيلنت يُعتبر تطبيقاً تفاعلياً. نناقش هنا مثال تيلنت لأنه سيوضح بشكلٍ جيد استخدام الأرقام التسلسلية وأرقام إشعارات الاستلام في بروتوكول TCP. جدير بالذكر أن العديد من المستخدمين يفضلون الآن استخدام بروتوكول ssh بدلاً من تيلنت، لأن البيانات

المُرسَلة في توصيلة تيلنت (بما في ذلك كلمات السر!) لا تشفّر، مما يجعل تيلنت عرضةً لهجمات التنصت (كما سنتناوله في الجزء 8-7).

افترض أن المضيف A يبدأ جلسة تيلنت مع المضيف B. ولأن المضيف A هو الذي بدأ الجلسة، فإننا نطلق عليه اسم الزبون، في حين نطلق على المضيف B الخادم. كل حرف يطبعه المستخدم (على الزبون) سيُرسَل إلى المضيف البعيد؛ والذي سيقوم بدوره بإرسال نسخة من كل حرف للعرض على شاشة مستخدم تيلنت. تستخدم طريقة "الصدى" هذه لضمان أن الحروف التي يراها مستخدم تيلنت على شاشته تكون قد تم استقبالها ومعالجتها بالفعل على المضيف البعيد. وعليه فإن كل حرف يكون قد عبر الشبكة مرتين في الفترة من ضغط المستخدِم المفتاح إلى لحظة عرضه على الشاشة أمامه.

افترض الآن أن المستخدم ضغط على حرف 'C' على لوحة المفاتيح، وبعدها أخذ يحتسى فنجاناً من القهوة. دعنا نفحص قطع بيانات TCP التي تُرسَل بين الزبون والخادم. كما هو موضح في الشكل 3-31، افترض أن الأرقام التسلسلية الأولى هي 42 و79 للزبون والخادم على التوالي. تذكّر أن الرقم التسلسلي لقطعة هو الرقم التسلسلي للبايت الأول في حقل البيانات بها. وعليه فستحمل القطعة الأولى المُرسَلة من الزبون الرقم التسلسلي 42؛ في حين يكون للقطعة الأولى المُرسَلة من الخادم الرقم التسلسلي 79. تذكّر أن رقم إشعار الاستلام هو الرقم التسلسلي لبايت البيانات التالي الذي ينتظره المضيف. بعد إنشاء وصلة TCP ولكن قبل إرسال أي بيانات، ينتظر الزبون البايت 79 والخادم البايت 42.

كما هو موضح في الشكل 3-31 يتم إرسال ثلاث قطع بيانات. تُرسَل القطعة الأولى من الزبون إلى الخادم وتتضمن في حقل البيانات بها بايت واحد يمثل الحرف المُرسَل 'C' حسب تكويد ASCII . تتضمن هذه القطعة أيضاً الرقم 42 في حقل الرقم التسلسلي بها، كما وصفنا آنفاً. لأن الزبون أيضاً لم يتسلم بعد أي بيانات من الخادم، فإن هذه القطعة الأولى ستحمل الرقم 79 في حقل رقم إشعار الاستلام.



الشكل 3- 31 الأرقام التسلسلية وأرقام إشعارات الاستلام أثناء عملية "تيلنت" بسيطة على بروتوكول TCP.

تُرسَل القطعة الثانية من الخادم إلى الزبون لتخدم غرضاً مزدوجاً. أولاً: تُزوّد القطعة الزبون بإشعار باستلام البيانات التي تسلمها الخادم. فبوضع 43 في حقل رقم إشعار الاستلام، يُخبر الخادم الزبون أنه تسلّم كل شيء بنجاح حتى البايت 42 وينتظر الآن البايتات 43 وما بعدها. أما الغرض الثاني لهذه القطعة فهو ترجيع صدى الحرف الواصل 'C' إلى الزبون، ولذا تتضمن القطعة الثانية تمثيل هذا الحرف في حقل البيانات بها. هذه القطعة الثانية لها الرقم التسلسلي 79، أي الرقم التسلسلي الأولي لتدفق البيانات من الخادم إلى الزبون في توصيلة TCP تلك، لأنها تحمل أول بايت بيانات يقوم الخادم بإرساله. لاحظ أن إشعار الاستلام للبيانات من الزبون إلى الخادم تحمله قطعة تحمل بيانات من الخادم إلى الزبون. يطلق على إشعار الاستلام هذا أنه "راكب على الظهر" (piggybacked) على قطعة البيانات من الخادم إلى الزبون.

تُرسَل القطعة الثالثة من الزبون إلى الخادم، وغرضها الوحيد هو إشعار الخادم باستلام البيانات التي أرسلها. (تذكّر أن القطعة الثانية تضمنت بيانات – الحرف 'C' من الخادم إلى الزبون). هذه القطعة لها حقل بيانات فارغ، (أي أن إشعار الاستلام لا "يركب على ظهر" أي بيانات من الزبون إلى الخادم). تحمل القطعة الرقم 80 في حقل رقم إشعار الاستلام لأن الزبون استلم البايتات المُرسَلة إليه حتى بايت 79، ومن ثم ينتظر الآن البايتات 80 وما بعدها. قد ترى من الغريب أن يكون لهذه القطعة رقماً تسلسلياً رغم أنها لا تحتوي على أي بيانات، ولكن نظراً لأن بروتوكول TCP له حقل للرقم التسلسلي، تحتاج القطعة لأن يكون لها رقما تسلسلياً على أي حال.

3-5-3 تقدير وقت رحلة الذهاب والعودة وفترة الموقّت

كما هو الحال في بروتوكول rdt الذي طوّرناه في الجزء 3-4، يستخدم بروتوكول TCP آليات انقضاء فترة الموقّت وإعادة الإرسال للتعافي من فقد قطع البيانات. وبالرغم من بساطة تلك الأساليب من حيث المبدأ، يظهر عدد من الأمور الدقيقة عندما نطبّق آلية انقضاء فترة الموقّت لإعادة الإرسال في بروتوكول حقيقي مثل TCP. لعل أكثر الأسئلة إلحاحًا يتعلق بطول فترات الموقّت المناسبة. واضح أن تلك الفترة يجب أن تكون أكبر من وقت رحلة الذهاب والإياب *RTT* على التوصيلة – أي الوقت من لحظة إرسال قطعة إلى حين وصول إشعار باستلامها– وإلا فقد يحدث إعادة إرسال بدون داعٍ. لكن إلى أي حد أكبر؟ بل كيف نقدر قيمة الوقت *RTT* بدايةً؟ هل نستخدم موقّتاً لكل قطعة لم يصل إشعار باستلامها؟ أسئلةٌ كثيرة! تستند مناقشتنا في هذا الجزء لبروتوكول TCP حسب [Jacobson 1988] وتوصيات فريق عمل هندسة الإنترنت (IETF) الحالية بخصوص إدارة موقّتات TCP [RFC2988].

تقدير وقت رحلة الذهاب والإياب

لنبدأ دراستنا لإدارة الموقّتات في بروتوكول TCP بالنظر في الطريقة التي يتبعها البروتوكول لتقدير وقت رحلة الذهاب والإياب (*RTT*) بين المُرسِل والمُستقبِل. يتم ذلك كالتّالي: يُعرّف وقت عينة *RTT* لقطعة بيانات، والذي نرمز له بـ SampleRTT ، بأنه الوقت المنقضي منذ إرسال القطعة (أي دفعها إلى بروتوكول IP بطبقة الشبكة) إلى وصول إشعار باستلامها. بدلاً من قياس SampleRTT لكل قطعة يتم إرسالها، تكتفي معظم تطبيقات TCP المستخدمة بقياس SampleRTT كعينة مرة واحدة كلما دعت الحاجة. بمعنى أنه عند كل نقطة زمنية، يجري تقدير SampleRTT لقطعة واحدة فقط من القطع التي أُرسِلت ولكن لم يتم الإشعار باستلامها بعد، مما يعطي قيمة جديدة لـ SampleRTT تقريباً مرة كل *RTT*. يلاحظ أيضاً أن TCP لا يقيس أبداً قيمةً SampleRTT لقطعة يعاد إرسالها؛ وإنما يقيسها فقط للقطع التي أرسلت مرة واحدة. (هناك تمرين في نهاية هذا الفصل يطلب منك تبرير ذلك).

واضحٌ أن قيمة SampleRTT ستتفاوت من قطعة إلى أخرى بسبب الازدحام في الموجّهات وتغيّر الأحمال على الأنظمة الطرفية، ولذلك يمكن أن تكون بعض قيم SampleRTT شاذة. لتقدير قيمة نمطية للوقت *RTT*، من الطبيعي حساب نوع من المتوسط لقيم SampleRTT المقاسة. يحتفظ TCP بقيمة متوسطة لقيم SampleRTT تُعرف بـ EstimatedRTT. عند الحصول على قيمة SampleRTT جديدة، يقوم TCP بتحديث تقديره للكمية EstimatedRTT حسب المعادلة التالية:

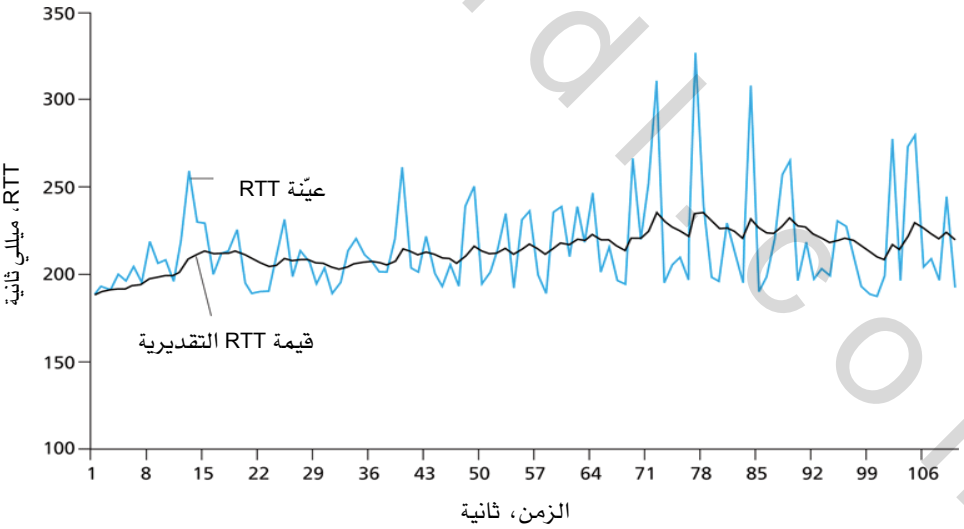
$$EstimatedRTT = (1 - \alpha) \times EstimatedRTT + \alpha \times SampleRTT$$

لاحظ أن تلك المعادلة مكتوبة على شكل تعليمة من تعليمات لغة البرمجة، حيث القيمة الجديدة لـ EstimatedRTT هي تركيبة موزونة من القيمة السابقة لـ EstimatedRTT والقيمة الجديدة لـ SampleRTT. القيمة الموصّى بها لـ α هي 0.125 [RFC 2988]، وفي هذه الحالة تصبح المعادلة:

$$EstimatedRTT = 0.875 \times EstimatedRTT + 0.125 \times SampleRTT$$

لاحظ أن EstimatedRTT هو معدل موزون لقيم SampleRTT. كما نوقش في أحد التمارين في نهاية هذا الفصل، هذا المتوسط الموزون يضع وزناً أكبر للعينات الأخيرة مقارنة بالعينات القديمة. وهذا الأمر طبيعي لأن العينات الأحدث تعكس بشكلٍ أدق وضع الازدحام الحالي في الشبكة. وفي علم الإحصاء يُطلق على مثل هذا المتوسط بالمتوسط المتحرك بأوزان أسّية (exponential weighted moving average (EWMA)). تظهر كلمة "أسّي" في الإسم لأن الوزن المعطى لعينة SampleRTT يضمحل بسرعة أُسّية مع توالي التجديدات. في التمارين سيطلب منك اشتقاق الحد الأسّي في EstimatedRTT.

يبين الشكل 3-32 قيم SampleRTT وقيم EstimatedRTT في حالة $\alpha = 0.125$ لتوصيلة TCP من المضيف gaia.cs.umass.edu (في مدينة Amherst بولاية Massachusetts)، إلى المضيف fantasia.eurecom.fr (في جنوب فرنسا). واضح أنه تم التخفيف من حدة الاختلافات الكبيرة في قيم SampleRTT من خلال حساب EstimatedRTT.



الشكل 3-32 عينات وقيم *RTT* التقديرية.

بعد الحصول على تقدير لقيمة متوسطة للوقت *RTT*، من المفيد أيضاً التوصل لمقياس لمدى التغير في *RTT*. يعرّف [RFC 2988] التفاوت في *RTT* (أي DevRTT) كتقدير لقيمة التغير الذي يُتوقع أن تنحرف به قيمة العينة SampleRTT عادةً عن قيمة المتوسط EstimatedRTT:

$$DevRTT = (1 - \beta) \times DevRTT + \beta \times | SampleRTT - EstimatedRTT |$$

لاحظ أنّ DevRTT هو متوسط متحرك بأوزان أسّية (EWMA) للفرق بين SampleRTT وEstimatedRTT. إذا كانت قيم SampleRTT تعاني من تفاوتات قليلة، فإن DevRTT سيكون صغيراً؛ وعلى العكس إذا كانت الاختلافات في قيم العينات كبيرة، فإن DevRTT سيكون كبيراً. قيمة β الموصى بها هي 0.25.

ضبط وإدارة فترة موقّت إعادة الإرسال (Timeout Interval)

إذا كانت لدينا قيمة لكلٍّ من EstimatedRTT وDevRTT، فما هي القيمة التي يجب أن نستخدمها لفترة الموقّت في TCP والتي بانقضائها يتم إعادة الإرسال؟ واضح أن تلك الفترة يجب أن تكون أكبر من أو تساوي EstimatedRTT، وإلا فإن عمليات إعادة إرسال ستتم بدون داعٍ. لكن فترة الموقّت لا ينبغي أيضاً أن تكون أكبر بكثير من EstimatedRTT، وإلا فعند فقد قطعة بيانات سيتأخر نظام TCP في إعادة إرسالها ومن ثم يتسبب في تأخيرات كبيرة في نقل البيانات. وعليه فمن المستحب اختيار فترة الموقّت بحيث تزيد قليلاً عن قيمة EstimatedRTT، ويجب أن تكون تلك الزيادة كبيرة عندما يكون هناك الكثير من التقلبات في قيم SampleRTT، وتكون قليلة عندما تكون تلك التقلبات صغيرة. معنى ذلك أن قيمة DevRTT يجب أن تلعب دوراً هنا. تُؤخذ كل هذه الاعتبارات في الحسبان في الطريقة التي يتبعها بروتوكول TCP لتحديد فترة موقّت إعادة الإرسال (TimeoutInterval):

$$TimeoutInterval = EstimatedRTT + 4 \times DevRTT$$

المبادئ في الواقع العملي (Principles in Practice)

يوفر بروتوكول TCP نقلاً موثوقاً للبيانات باستخدام إشعارات الاستلام الإيجابية والموقّتات تقريباً بنفس الطريقة التي درسناها في الجزء 3-4. يرسل TCP إشعاراً باستلام قطع البيانات التي يتسلمها بشكلٍ صحيح، ويعيد إرسال القطع متى غلب على ظنه أن تلك القطع أو إشعارات استلامها قد فُقدت أو وصلت فاسدة. تتضمن بعض إصدارات TCP أيضاً آلية ضمنية لإشعارات الاستلام السلبية (NAK). ففي آلية TCP السريعة لإعادة الإرسال، يؤخذ استلام ثلاثة إشعارات استلام إيجابية مكرّرة لقطعة ما على أنه إشعار استلام سلبي (NAK) ضمني بخصوص القطعة التالية، ومن ثم يؤدي إلى إعادة إرسال تلك القطعة قبل انقضاء فترة الموقّت. يستخدم بروتوكول TCP الأرقام التسلسلية للسماح للمُستقبل باكتشاف قطع البيانات المفقودة أو المكرّرة. تماماً كما في حالة البروتوكول rdt3.0 للنقل الموثوق للبيانات لن يتمكن بروتوكول TCP بنفسه من أن يحدد على وجه اليقين ما إذا كانت قطعة بيانات (أو إشعار استلامها) قد فُقدت أو فسدت أو تأخرت أكثر مما ينبغي. سيكون لبروتوكول TCP في المُرسِل نفس رد الفعل إزاء كل تلك الاحتمالات: إعادة إرسال القطعة موضع التساؤل.

يستخدم بروتوكول TCP أيضاً أسلوب خط الأنابيب (pipelining) للسماح للمُرسِل بأن يكون لديه عدة قطع بيانات منتظرة في وقت ما، تكون قد أُرسلت ولم تصل إشعارات باستلامها بعد. رأينا في وقت سابق أن هذا الأسلوب يمكن أن يحسّن كثيرا من الطاقة الإنتاجية للجلسة عندما يكون طول قطعة البيانات صغيراً مقارنةً بتأخير رحلة الذهاب والإياب. يتم تحديد العدد المعين من القطع المنتظرة بدون إشعار استلام لدى المُرسِل عن طريق آليات التحكم في الازدحام وضبط التدفق. سنتناول ضبط التدفق في بروتوكول TCP في نهاية هذا الجزء، أما التحكم في الازدحام في TCP فسنناقشه في الجزء 3-7. الآن علينا فقط وببساطة إدراك أن TCP يستخدم أسلوب خط الأنابيب لتحقيق ذلك.

3-5-4 النقل الموثوق للبيانات

تذكّر أن خدمة طبقة شبكة الإنترنت (خدمة IP) غير موثوقة، فهي لا تضمن توصيل وحدات بيانات IP، ولا تضمن توصيل تلك الوحدات بالترتيب السليم، ولا

تضمن سلامة البيانات المستلمة في تلك الوحدات. فمع خدمة IP، يمكن لوحدات البيانات أن تفيض في المخازن المؤقتة بالموجّهات فتُفقد ولا تصل إلى وجهتها أبداً، كما يمكن لتلك الوحدات أن تصل فاسدة بأخطاء في بتاتها (حيث يُقلب الـ 0 إلى 1 والعكس بالعكس). ولأن قطع طبقة النقل يتم نقلها عبر الشبكة بواسطة وحدات بيانات IP تلك، فإن قطع طبقة النقل يمكن أن تعاني من نفس المشاكل أيضاً.

ينشىء بروتوكول TCP خدمة نقل موثوق للبيانات فوق خدمة IP غير الموثوقة التي تعتمد طريقة الجهد الأفضل. تضمن خدمة TCP للنقل الموثوق للبيانات أن سيل البيانات الذي تقرؤه عمليةٌ ما من مخزن TCP المؤقت غير فاسدة، وبدون فجوات، وبدون تكرار، وبالترتيب السليم – أي أنها تتسلم بالضبط نفس سيل البايتات الذي أرسله النظام الطرفي على الجانب الآخر من توصيلة TCP . إذن فكيف يوفر TCP نقلاً موثوقاً للبيانات؟ يتضمن ذلك العديد من المبادئ التي درسناها في الجزء 3-4.

في تطويرنا السابق لأساليب النقل الموثوق للبيانات، كان من السهل من حيث المبدأ افتراض تخصيص موقّت على حِدة لكل قطعة بيانات تم إرسالها ولم يصل بعد إشعار باستلامها. رغم أن هذا رائع نظرياً إلا أن إدارة الموقّتات بهذه الطريقة ستشكل عبئاً كبيراً، ولذا فإن الإجراءات الموصى بها لإدارة موقّتات TCP [RFC 2988] تنص على استعمال موقّت واحد فقط لإعادة الإرسال، حتى لو كان هناك عدة قطع بيانات تم إرسالها ولم يصل إلى المُرسِل الإشعار باستلامها بعد. يتبع بروتوكول TCP الذي نصِفُه في هذا الجزء هذه التوصية باستعمال موقّت واحد.

سنناقش هنا كيف يوفر بروتوكول TCP نقلاً موثوقاً للبيانات في خطوتين تدريجيتين. نقدم أولاً وصفاً مبسطاً بدرجةٍ كبيرة لمُرسِل TCP يستخدم فقط أسلوب المؤقّت للتعافي من فقد قطع البيانات، ثمّ نقدم بعد ذلك وصفاً أكثر كمالاً لمُرسِل يستخدم إشعارات الاستلام المكرّرة بالإضافة إلى انقضاء فترة الموقّت. سنفترض في

المناقشة التالية أن البيانات تُرسَل فقط في اتجاه واحد، من المضيف A إلى المضيف B، وأن المضيف A يرسل ملفاً كبيراً.

يوضح الشكل 3-33 وصفاً مبسطاً للغاية لمُرسِل TCP، حيث نرى أن هناك ثلاثة أحداث رئيسة تتعلق بإرسال وإعادة إرسال البيانات في مُرسِل TCP: وصول البيانات من التطبيق في الطبقة الأعلى، وانقضاء فترة الموقّت، ووصول إشعار استلام. عند وقوع الحدث الرئيس الأول، يتسلم TCP البيانات من التطبيق ويغلّفها في قطعة، ثم يدفع بالقطعة إلى بروتوكول IP في طبقة الشبكة أسفله. لاحظ أن كل قطعة بيانات تتضمن رقماً تسلسلياً هو رقم بايت البيانات الأول في القطعة ضمن سيل البايتات الجاري إرساله، كما سبق وصفه في الجزء 3-5-2. لاحظ أنه إذا لم يكن الموقّت شغالاً لقطعة أخرى، فإن TCP يبدأ الموقّت عندما يدفع بالقطعة إلى IP (من المفيد اعتبار الموقّت مرتبطاً بأقدم قطعة لم يتم وصول إشعار باستلامها بعد). افترض أن فترة الانتهاء لهذا الموقّت هي TimeoutInterval، والتي يتم حسابها بمعلومية EstimatedRTT وDevRTT، كما تقدم وصفه في الجزء 3-5-3.

يتمثل الحدث الرئيس الثاني في انقضاء فترة الموقّت، والذي يستجيب له TCP بإعادة إرسال القطعة التي تسببت في انقضاء فترة الموقّت. بعد ذلك يقوم TCP ببدء تشغيل الموقّت من جديد لقطعة أخرى.

الحدث الرئيس الثالث الذي يجب على مُرسِل TCP التعامل معه هو وصول قطعة تحمل إشعار استلام ACK من المُستقبِل (بتحديد أكثر، قطعة تحتوي على قيمة حقيقية في حقل إشعار الاستلام). عند حدوث ذلك يقارن TCP قيمة إشعار الاستلام y بالمتغير SendBase لديه. لاحظ أن المتغير SendBase الذي يسهم في وصف حالة TCP هو الرقم التسلسلي لأقدم بايت لم يصل إشعار استلام له بعد. (وهكذا يكون SendBase - 1 هو الرقم التسلسلي للبايت الأخير الذي تم التأكد من أنه قد وصل بشكلٍ صحيح وبالترتيب السليم). كما سبق أن أشرنا، يستخدم TCP إشعارات استلام تراكمية، بحيث يُخبر إشعار الاستلام y باستلام كل البايتات التي أُرسلت قبل البايت رقم y. إذا كانت $y > SendBase$، فإن إشعار الاستلام

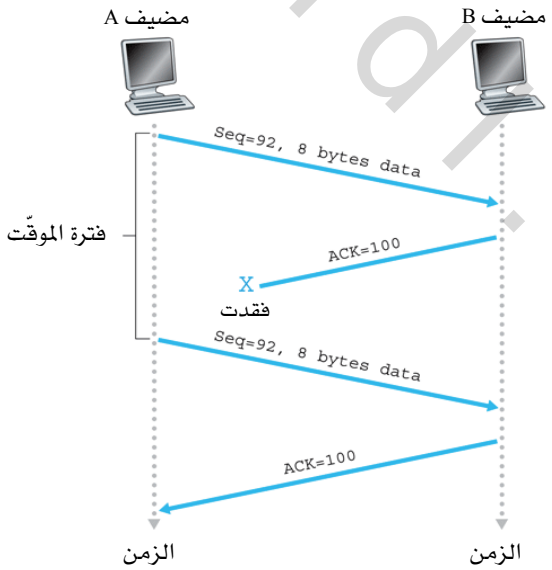
الواصل يُخبر باستلام قطعة أو أكثر من قطع لم يتم الإشعار باستلامها سابقاً. وعليه يقوم المُرسِل بتحديث المتغير SendBase لديه، كما أنه يبدأ تشغيل الموقّت من جديد إذا كانت هناك أي قطع متبقية لم يصل إشعار باستلامها.

```
/* Assume sender is not constrained by TCP flow or congestion control, that
data from above is less than MSS in size, and that data transfer is in one
direction only * /

NextSeqNum=InitialSeqNumber
SendBase=InitialSeqNumber

loop (forever) {
    switch (event)
    event: data received from application above
            create TCP segment with sequence number NextSeqNum
            if (timer currently not running)
                start timer
            pass segment to IP
            NextSeqNum=NextSeqNum+length (data)
            break;

    event: timer timeout
            retransmit not-yet-acknowledged segment with smallest
                sequence number
            start timer
            break;

    event: ACK received, with ACK field value of y
            if (y > SendBase) {
                SendBase=y
                if (there are currently any not-yet-acknowledged segments)
                    start timer
            }
            break,;

} /* end of loop forever * /
```

الشكل 3-33 مُرسِل مبسّط لبروتوكول TCP.

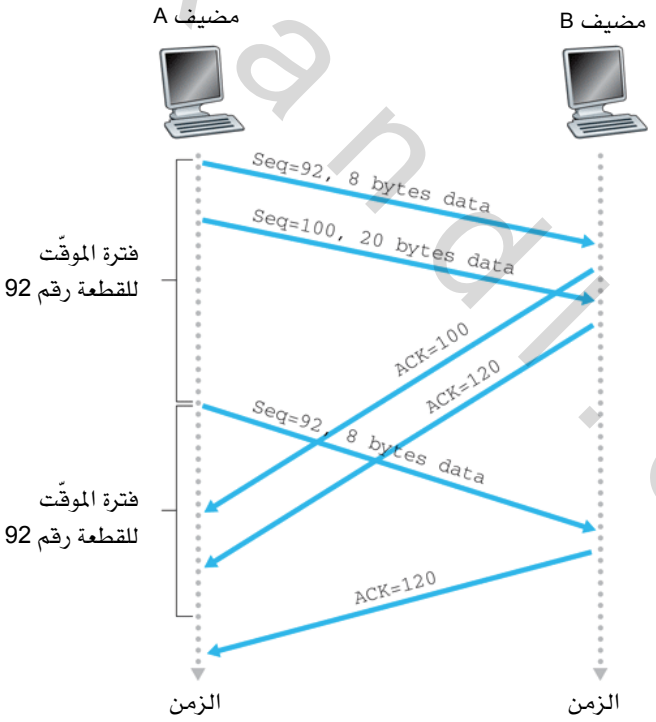
بضعة سيناريوهات هامة

انتهينا للتوّ من وصف مبسّط للغاية لأسلوب TCP في توفير نقل موثوق للبيانات. ولكن حتى هذا النمط المبسّط جداً من TCP غنيٌ بالكثير من الأمور الدقيقة والهامة. ولفهم جيد لكيفية عمل هذا البروتوكول دعنا الآن نتلمّس طريقنا خلال بضعة سيناريوهات مبسّطة. يبين الشكل 3-34 السيناريو الأول، حيث يرسل المضيف A قطعة واحدة للمضيف B. افترض أن هذه القطعة لها الرقم التسلسلي 92 وتتضمن 8 بايتات من البيانات. بعد إرسال تلك القطعة، ينتظر المضيف A قطعة من B تحمل إشعار استلام برقم 100. لنفرض أنه رغم استلام B للقطعة المُرسَلة من A، إلا أن إشعار الاستلام من B إلى A فُقد في الطريق. في هذه الحالة تنتهي فترة الموقّت، ويقوم المضيف A بإعادة إرسال نفس القطعة. بالطبع عندما يستقبل المضيف B القطعة المعاد إرسالها، سيلاحظ من الرقم التسلسلي للقطعة أنها تحمل بيانات سبق استلامها، ولذا يقوم TCP على المضيف B بإهمال بايتات البيانات في تلك القطعة.



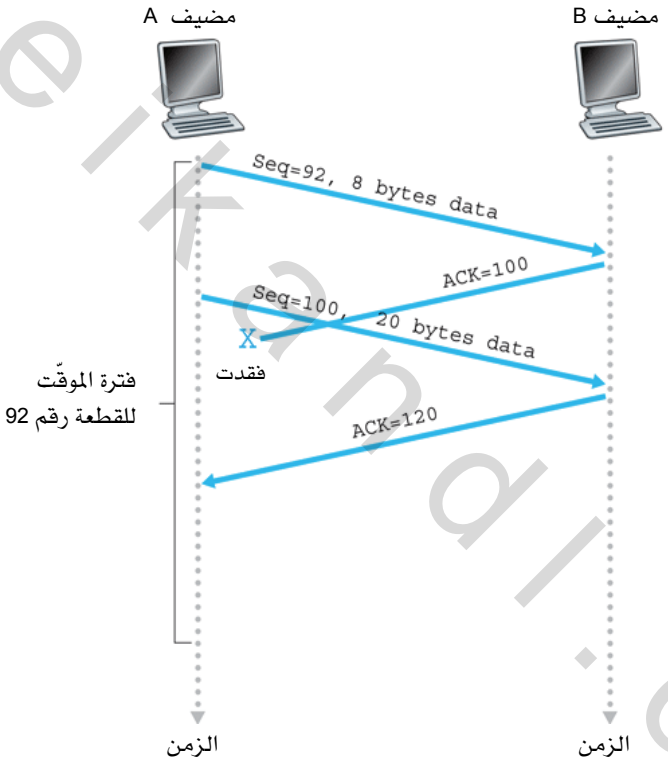
الشكل 3-34 إعادة الإرسال نتيجة فقد إشعار استلام.

في السيناريو الثاني والموضح في الشكل 3-35 يرسل المضيف A قطعتي بيانات الواحدة تلو الأخرى مباشرةً، الأولى لها الرقم التسلسلي 92 وتحمل 8 بايتات من البيانات، والثانية لها الرقم التسلسلي 100 وتحمل 20 بايت من البيانات. افترض أن كل قطعة وصلت سليمة إلى B، وأن B أرسل إشعار استلام منفصل لكل منهما، الإشعار الأول رقمه 100، والثاني رقمه 120. افترض الآن أن أياً من إشعاري الاستلام لم يصل إلى المضيف A قبل انقضاء فترة الموقّت. عند انقضاء فترة الموقّت يقوم المضيف A بإعادة إرسال القطعة الأولى برقم تسلسلي 92 ويبدأ تشغيل الموقّت من جديد. في حالة وصول إشعار استلام القطعة الثانية قبل انقضاء فترة الموقّت الجديدة، فإن القطعة الثانية لن يعاد إرسالها.



الشكل 3-35 لن يعاد إرسال القطعة 100.

في السيناريو الثالث والأخير والموضح في الشكل 3-36 افترض أن المضيف A أرسل القطعتين بالضبط كما في المثال الثاني، وأن إشعار الاستلام الخاص بالقطعة الأولى فقط فُقد في الشبكة، ولكن مباشرةً قبل انقضاء فترة الموقّت تسلّم المضيف A إشعار استلام رقمه 120. عندئذٍ يدرك المضيف A أن المضيف B قد تسلّم كل شيء لغاية البايت 119، ولذا فإن المضيف A لا يعيد إرسال أيٍّ من القطعتين.



الشكل 3-36 يؤدي إشعار الاستلام التراكمي إلى تجنب إعادة إرسال القطعة الأولى.

مضاعفة فترة الموقّت

سنناقش الآن بضعة تعديلات تتضمنها أكثر تطبيقات بروتوكول TCP المستخدمة حالياً. يتعلّق التعديل الأول بطول فترة الموقّت، فحينما تنقضي فترة الموقّت، يعيد TCP إرسال قطعة البيانات التي لم يصل إشعار باستلامها والتي لها

أصغر رقم تسلسلي كما ذكرنا سابقاً. ولكن في كل مرة يقوم البروتوكول بعملية إعادة إرسال، يضبط فترة الموقّت التالية بحيث تكون ضعف قيمتها السابقة، بدلاً من اشتقاقها باستخدام آخر قيم متاحة لـ EstimatedRTT وDevRTT (كما تقدم وصفه في الجزء 3-5-3). على سبيل المثال افترض أن فترة الموقّت TimeoutInterval المرتبطة بأقدم قطعة لم يتم الإشعار باستلامها هي 0.75 ثانية عندما انقضت فترة الموقّت للمرة الأولى. عندئذ سيعيد TCP إرسال تلك القطعة ثم يضبط فترة الموقّت الجديدة عند 1.5 ثانية. إذا انقضت فترة الموقّت مرة أخرى بعد 1.5 ثانية، فإن TCP سيعيد إرسال تلك القطعة من جديد، ثم يضبط فترة الموقّت هذه المرة عند 3.0 ثانية. وهكذا تنمو فترة الموقّت تصاعدياً بعد كل إعادة إرسال. ومع ذلك فحينما يتم بدأ تشغيل الموقّت بعد أيٍّ من الحدثين الآخرين (أي تسلّم بيانات من تطبيق في الطبقة الأعلى أو وصول إشعار استلام) فإن الفترة TimeoutInterval يتم اشتقاقها كالمعتاد من آخر قيم متوفرة لـ EstimatedRTT وDevRTT.

يمثل هذا التعديل شكلاً محدوداً من طرق التحكم في الازدحام. (سنتناول أشكالاً أشمل لوسائل التحكم في الازدحام في الجزء 3-7). غالباً ما تنقضي فترة الموقّت بسبب ازدحام الشبكة، أي بسبب وصول رزم كثيرة إلى واحد (أو أكثر) من صفوف الانتظار في موجّه أو أكثر على المسار بين المصدر والوجهة النهائية، مما يتسبّب في فقد بعض الرزم أو زيادة في تأخيرات الانتظار في الصف. في أوقات الازدحام إذا استمرت مصادر البيانات في إعادة إرسال الرزم المفقودة أو المتأخرة بإصرار، فإن حالة الازدحام قد تزداد سوءاً. بدلاً من ذلك يتصرّف TCP بطريقة أفضل حيث يقوم كل مُرسِل بإعادة الإرسال بعد فترات أطول فأطول. وسنرى عند دراستنا لأسلوب CSMA/CD في الفصل الخامس أن بروتوكول الإيثرنت يستخدم طريقة مماثلة.

الإعادة السريعة للإرسال

من مشاكل أسلوب إعادة الإرسال أن فترة الموقّت يمكن أن تكون طويلة نسبياً، فعند فقد قطعة بيانات، تؤدى فترة الموقّت الطويلة تلك إلى تأخير المُرسِل إرسال الرزمة المفقودة مرة ثانية، وبذلك يزداد التأخير الذي تعانيه الرزمة من طرف إلى طرف. لكن لحسن الحظ غالباً ما يكون بوسع المُرسِل اكتشاف فقد الرزمة قبل انقضاء فترة الموقّت بمدة طويلة وذلك بملاحظة إشعارات الاستلام المكرّرة. إشعار الاستلام المكرّر هو إشعار باستلام قطعة سبق أن تلقى المُرسِل ما يفيد استلامها. لإدراك طبيعة رد المُرسِل على إشعار استلام مكرّر، علينا التفكير في السبب الذي يجعل المُستقبِل يرسل إشعار استلام مكرّر بدايةً. يلخص الجدول 3-2 سياسة مُستقبِل TCP في توليد إشعارات الاستلام [RFC 1122, RFC 2581]. عندما يتلقى مُستقبِل TCP قطعةً لها رقم تسلسلي أكبر من الرقم التسلسلي التالي المتوقع بالترتيب السليم، فإنه يكتشف فجوة في سيل البيانات التي تصله من المُرسِل - أي يكتشف افتقاد (غياب) قطعة. يمكن أن تنتج تلك الفجوة من فقد أو تبديل ترتيب القطع داخل الشبكة. نظراً لأن بروتوكول TCP لا يستخدم أسلوب إشعارات الاستلام السلبية، فإن المُستقبِل لا يستطيع إرسال إشعار استلام سلبي صريح إلى المُرسِل، لكن عوضاً عن ذلك يقوم المُرسِل ببساطة بإعادة الإشعار باستلام آخر بايت بيانات استلمها صحيحة بالترتيب السليم (أي يرسل إشعار استلام مكرّر لها). (لاحظ أن الجدول 3-2 يتضمن الحالة التي لا يهمل فيها المُستقبِل القطع التي تصل بغير الترتيب السليم).

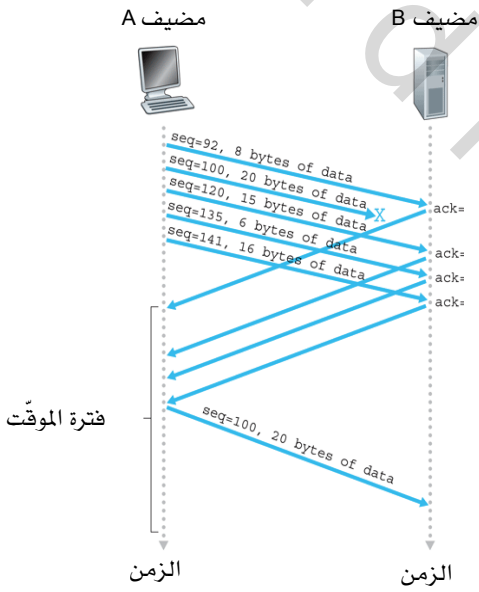
الحدث	الإجراء لدى مُستقبل TCP
وصلت قطعة بالترتيب السليم وبالرقم التسلسلي المتوقّع. كل البيانات لغاية القطعة بالرقم التسلسلي المتوقّع تم الإشعار باستلامها.	إشعار استلام متأخر. انتظر وصول قطعة أخرى بالترتيب السليم لمدة أقصاها 500 ميللي ثانية. إذا لم تصل القطعة التالية حسب الترتيب السليم خلال تلك الفترة، أرسل إشعار استلام.
وصلت قطعة بالترتيب السليم وبالرقم التسلسلي المتوقّع. توجد قطعة أخرى وصلت بالترتيب السليم تنتظر إرسال إشعار باستلامها.	أرسل إشعار استلام تراكمي واحد في الحال للإشعار باستلام كلا القطعتين الواصلتين بالترتيب السليم.
وصلت قطعة بترتيب غير سليم تحمل رقماً تسلسلياً أكبر من المتوقّع. اكتشاف فجوة.	أرسل حالاً إشعار استلام مكرّر يحمل الرقم التسلسلي للبايت التالية المتوقعة (والتي تمثل الطرف الأدنى للفجوة).
وصلت قطعة تملأ (جزئياً أو كلياً) الفجوة في البيانات التي يجري استقبالها.	أرسل حالاً إشعار استلام، إذا كانت القطعة تبدأ عند الطرف الأدنى للفجوة.

الجدول 3-2 توصيات إصدار إشعارات الاستلام في مُستقبل TCP [RFC1122; RFC2581].

نظراً لأن المُرسِل يرسل في أغلب الأحيان عدداً كبيراً من القطع، الواحدة تلو الأخرى، فإنه إذا فقدت قطعة واحدة، فمن المحتمل أن يؤدي ذلك إلى إرسال العديد من إشعارات الاستلام المكرّرة الواحد تلو الآخر. إذا استلم مُرسِل TCP ثلاثة إشعارات استلام مكرّرة لنفس قطعة البيانات، سيأخذ ذلك كإشارة إلى أن القطعة التالية للقطعة التي تم الإشعار باستلامها ثلاث مرات قد فقدت. (في التمارين سنناقش لماذا ينتظر المُرسِل تلقّي ثلاثة إشعارات استلام مكرّرة وليس فقط مجرد تلقي إشعار استلام مكرّر واحد). عند تلقّي ثلاثة إشعارات استلام مكرّرة يقوم مُرسِل TCP بإعادة سريعة للإرسال [RFC 2581]، حيث يعيد إرسال القطعة المفقودة قبل انقضاء فترة موقّت تلك القطعة. يُبيّن ذلك في الشكل 3-37 حيث تُفقد القطعة الثانية ثم يُعاد إرسالها قبل أن تنتهي فترة موقّتها. في هذه الحالة يمكن استبدال حدث وصول إشعار الاستلام في الشكل 3-33 بالأوامر التالية للحصول على بروتوكول TCP بإعادة سريعة للإرسال:

```
event: ACK received, with ACK field value of y
        if (y > SendBase) {
            SendBase=y
            if (there are currently any not-yet-acknowledged segments)
                start timer
        }
        else {
            /* a duplicate ACK for already ACKed segment */
            increment number of duplicate ACKs received for y
            if (number of duplicate ACKS received for y==3){
                /* TCP fast retransmit */
                resend segment with sequence number y
        }
        break;
```

لاحظنا في وقت سابق أن العديد من الأمور الدقيقة تنشأ عند تطبيق آلية انقضاء فترة موقّت إعادة الإرسال في بروتوكول فعلي مثل TCP. إن الإجراءات السابقة والتي نتجت كحصيلة لأكثر من 15 عاماً من الخبرة والتجربة مع موقّتات TCP، حريٌّ بها أن تقنعك بهذه الحقيقة!



الشكل 3-37 إعادة إرسال قطعة قبل انقضاء فترة الموقّت الخاص بها.

"الرجوع N للوراء" (GBN) أم "الإعادة الانتقائية" (SR)؟

دعنا نختم دراستنا لآليات بروتوكول TCP للتعافي من الأخطاء بطرح السؤال التالي: هل يستخدم بروتوكول TCP أسلوب "الرجوع N للوراء" (GBN) أم أسلوب "الإعادة الانتقائية" (SR)؟ تذكّر أن إشعارات الاستلام في TCP تراكمية، وأن القطع التي تُستلم بشكلٍ صحيح ولكن بغير الترتيب السليم لا يتم الإشعار باستلامها بشكلٍ فردي من قِبَل المُستقبِل. ولذا فكما هو موضح في الشكل 3-33 (انظر أيضاً الشكل 3-19)، يحتاج مُرسِل TCP فقط للاحتفاظ بأصغر رقم تسلسلي للبايت التي أُرسِلت ولم يتم بعد الإشعار باستلامها (SendBase) وكذلك الرقم التسلسلي للبايت التالية التي عليها الدور في الإرسال (NextSeqNum). بهذا المفهوم يشبه TCP كثيراً البروتوكولات بأسلوب "الرجوع N للوراء" (GBN)، غير أن هناك عدة اختلافات واضحة بين بروتوكول TCP وبروتوكولات GBN. العديد من تطبيقات TCP تقوم بالتخزين المؤقت لقطع البيانات التي تصل بشكلٍ صحيح ولكن بترتيب غير سليم [Stevens 1994]. تأمّل أيضاً ما يحدث عندما يقوم المُرسِل بإرسال القطع 1، 2، . . . ، N وكل القطع تصل صحيحة وبالترتيب السليم إلى المُستقبِل. افترض أيضاً أن إشعار الاستلام للرزمة n فُقد، حيث $n < N$ ، لكن كل إشعارات الاستلام الأخرى لباقي القطع وعددها (N - 1) تصل إلى المُرسِل قبل انقضاء فترة الموقّت. في هذا المثال يعيد بروتوكول GBN إرسال القطعة n وكلّ القطع التالية لها n + 1، n + 2، . . . ، N، وليس فقط القطعة n وحدها. أما بروتوكول TCP فيعيد إرسال قطعة واحدة على الأكثر وهي القطعة n. بل أكثر من ذلك، قد لا يعيد بروتوكول TCP إرسال حتى القطعة n في حالة وصول إشعار استلام القطعة n + 1 قبل انقضاء فترة الموقّت للقطعة n.

من التعديلات المقترحة لبروتوكول TCP تعديلٌ يُعرف بالإشعار الانتقائي بالاستلام [RFC 2018]. يسمح هذا التعديل لمُستقبِل TCP بالإشعار باستلام القطع التي تصل بغير الترتيب السليم بشكلٍ انتقائي بدلاً من الاكتفاء فقط بالإشعار باستلام آخر قطعة صحيحة بالترتيب السليم بشكلٍ تراكمي. عند دمج هذه

الطريقة مع أسلوب الإعادة الانتقائية للإرسال – مع تجنب إعادة إرسال القطع التي أُقرّت بشكلٍ انتقائي من قبل المُستقبِل – فإن بروتوكول TCP يبدو كثير الشبه ببروتوكول SR العام الذي سبق أن تناولناه. وهكذا فإن أفضل تصنيف لآلية التعافي من الأخطاء في بروتوكول TCP قد يكون اعتباره كهجين ما بين بروتوكولي GBN وSR.

5-5-3 ضبط التدفق (Flow Control)

تذكّر أن المضيفين على جانبي توصيلة TCP يخصصان حيّز تخزين مؤقت لاستقبال قطع البيانات الواصلة. عندما تتسلم توصيلة TCP بايتات صحيحة وبالترتيب السليم تضعها في مخزن الاستقبال المؤقت. تقوم عملية التطبيق المناظرة بقراءة البيانات من ذلك المخزن، ولكن ليس بالضرورة في نفس لحظة وصول البيانات. في الواقع قد يكون التطبيق المقصود مشغولاً بمهمّة أخرى وقد لا يحاول قراءة البيانات حتى بعد فترة طويلة من وصولها. إذا كان التطبيق بطيئاً نسبياً في قراءة البيانات، يمكن بسهولة أن يتسبب المُرسِل في فيضان المخزن المؤقت لدى المُستقبِل إذا قام بإرسال بيانات أكثر من اللازم بسرعة أكبر من اللازم.

يوفر بروتوكول TCP للتطبيقات التي تستخدمه خدمة لضبط التدفق لتجنب احتمال تسبب المُرسِل في فيضان المخزن المؤقت لدى المُستقبِل. وعليه فضبط التدفق هو خدمة لمواءمة السرعة – أي مواءمة السرعة التي يرسل بها المُرسِل البيانات للسرعة التي يقوم بها التطبيق المُستقبِل بقراءة تلك البيانات على الطرف الآخر من التوصيلة. كما ذكرنا سابقاً يمكن أيضاً أن يُحِد بروتوكول TCP من قدرة المُرسِل على الإرسال بسبب الازدحام في شبكة IP. يعرف هذا الشكل من التحكم في المُرسِل بالتحكم في الازدحام وهو موضوع آخر سنتناوله بالتفصيل في الجزأين 6-3 و7-3. ورغم أن الإجراءات التي تُتخذ للسيطرة على الازدحام تشبه تلك المتبعة في ضبط التدفق (أي الحد من سرعة إرسال المُرسِل للبيانات)، فإن تلك الإجراءات تُتخذ لأسباب مختلفة جداً في الحالتين. قد يستخدم الكثير من الباحثين المصطلحين بشكلٍ متبادل، غير أنه يجدر بالقارئ الواعي أن يميز بينهما. دعنا

نناقش الآن كيف يوفر TCP خدمة ضبط التدفق. لكي نرى الغاية من خلال الأشجار، سنفترض في كافة أنحاء هذا الجزء أن المُستقبِل في بروتوكول TCP المستخدم يقوم بإهمال قطع البيانات التي تصل بترتيب غير سليم.

يوفر بروتوكول TCP ضبط التدفق بجعل المُرسِل يحتفظ بمتغير يطلق عليه نافذة المُستقبِل. من حيث المبدأ يُستخدم هذا المتغير لإعطاء المُرسِل فكرة عن حيّز التخزين المؤقت المتاح لدى المُستقبِل. نظراً لأن الإرسال في بروتوكول TCP كامل الازدواج في الاتجاهين (Full-duplex)، يحتفظ المُرسِل في كل من جانبي التوصيلة بنافذة مُستقبِل مميزة. دعنا نتحرّى استخدام متغير نافذة المُستقبِل ضمن سياق إرسال الملفات. افترض أن المضيف A يرسل ملفاً كبيراً إلى المضيف B على توصيلة TCP. يخصص المضيف B حيّز تخزين مؤقت لتلك التوصيلة، سنرمز لحجمه بالرمز RcvBuffer. من حين لآخر تقوم عملية التطبيق في المضيف B بالقراءة من ذلك المخزن المؤقت. دعنا نعرّف المتغيرات التالية:

- LastByteRead: رقم آخِر بايت في سيل البيانات قامت عملية التطبيق على المضيف B بقراءته.
- LastByteRcvd: رقم آخِر بايت في سيل البيانات وصل من الشبكة وتم وضعه في المخزن المؤقت على المضيف B.

نظراً لأن TCP لا يسمح بفيضان المخزن المؤقت لدى المُستقبِل، ينبغي أن يكون:

$$LastByteRcvd - LastByteRead \leq RcvBuffer$$

يتم ضبط قيمة نافذة المُستقبِل بحيث تساوي سعة التخزين المتاحة في مخزن الاستقبال المؤقت:

$$RcvWindow = RcvBuffer - [LastByteRcvd - LastByteRead]$$

نظراً لأن حيّز التخزين المتاح يتغير مع الوقت، تتغير قيمة RcvWindow ديناميكياً كما هو موضح في الشكل 3-38.

الشكل 3-38 نافذة المُستقبِل (RcvWindow) والمخزن المؤقت لدى المستقبل (RcvBuffer)

كيف تُستخدم توصيلة TCP المتغير RcvWindow لتوفير خدمة ضبط التدفق؟ يخبر المضيف B المضيف A عن حيّز التخزين المتاح في مخزن الاستقبال لديه بوضع القيمة الحالية لـ RcvWindow في حقل نافذة المُستقبِل بكل قطعة بيانات يرسلها إلى المضيف A. لاحظ أنه لكي يتمكن B من ذلك، عليه أن يتابع التغيرات الحاصلة في عدّة متغيرات تتعلق بتوصيلة TCP. في البداية يضع المضيف B قيمة المتغير RcvWindow بحيث تساوي RcvBuffer.

يقوم المضيف A بدوره بمتابعة متغيرين هما LastByteSent وLastByteAcked واللذين يدل اسمهما بوضوح على وظيفتيهما. لاحظ أن حاصل طرح هذين المتغيرين، أي (LastByteAcked - LastByteSent)، يمثل كمية البيانات التي أرسلها A ولم يتلق من B إشعاراً باستلامها بعد. بالإبقاء على كمية البيانات التي لم يتم الإشعار باستلامها أقل من قيمة RcvWindow، يمكن للمضيف A أن يطمئن أنّه لن يتسبب في فيضان المخزن المؤقت في المضيف B. وعليه، يسعى المضيف A لضمان تحقق الشرط التالي طوال مدة التوصيلة:

$$LastByteSent - LastByteAcked \leq RcvWindow$$

يعاني هذا النظام من مشكلة فنية بسيطة. لاكتشاف هذه المشكلة افترض أن المخزن المؤقت على المضيف B ممتلئ، أي أن RcvWindow = 0. بعد إخطار المضيف B للمضيف A أن RcvWindow = 0، افترض أيضاً أن B ليس لديه ما يرسله إلى A. تصور ما يمكن أن يحدث الآن. بينما عملية التطبيق في B تفرّغ المخزن المؤقت، لا يرسل TCP قطع بيانات جديدة بقيم RcvWindow جديدة من B إلى A. في الواقع إن TCP على المضيف B يرسل قطعاً إلى A فقط إذا كان لديه بيانات يود إرسالها أو إشعار استلام يود إرساله. وبالتالي لن تتاح الفرصة أبداً للمضيف B لإخبار المضيف A أن انفراجاً قد حدث وأن هناك فراغاً متاحاً الآن للتخزين على المضيف B — لذا يظل المضيف A معاقاً ولا يمكنه إرسال المزيد من البيانات! لحل هذه المشكلة، تتطلّب مواصفات TCP من المضيف A مواصلة إرسال قطع بيانات تضم بايتاً واحدة عندما يكون حجم نافذة المُستقبل على المضيف B صفراً. سيقوم B بدوره بإرسال إشعارات استلام لتلك القطع. بمرور الوقت سيوجد مكان خالٍ في المخزن المؤقت على B وستحتوي إشعارات الاستلام على قيمة غير صفرية للمتغير RcvWindow. يتضمن الموقع المصاحب لهذا الكتاب على الإنترنت http://www.awl.com/kurose-ross برنامج جافا تفاعلي يصوّر عمل نافذة المُستقبل في بروتوكول TCP. بعد أن استعرضنا خدمة ضبط التدفق في بروتوكول TCP، نذكر سريعاً هنا أن بروتوكول UDP لا يوفر تلك الخدمة. لفهم تلك القضية خذ في الاعتبار إرسال سلسلة من قطع بيانات UDP من عملية على المضيف A إلى عملية على المضيف B. في تطبيق نمطي لبروتوكول UDP سيضع البروتوكول القطع في مخزن مؤقت محدود الحجم يسبق المقبس المناظر للعملية التي تستقبل البيانات (والذي يمثل البوابة إلى العملية). تقرأ العملية قطعة واحدة كاملة في كل مرة من ذلك المخزن المؤقت. فإذا كانت العملية لا تقرأ القطع من المخزن بالسرعة الكافية، سيفيض المخزن بالبيانات وتُفقد القطع.

6-5-3 إدارة توصيلة TCP

في هذا الجزء من الكتاب سنلقي نظرة أكثر تفحصاً على كيفية إنشاء وإنهاء توصيلات TCP. هذا الموضوع قد لا يبدو مثيراً بدرجة كبيرة، إلا أنه مهم - فإنشاء توصيلة TCP يمكن أن يضيف بشكلٍ ملحوظ إلى التأخيرات المحسوسة (على سبيل المثال، عند تصفح الويب). علاوة على ذلك فإن العديد من الهجمات الأكثر شيوعاً على الشبكات – بما في ذلك هجوم فيضان SYN ذائع الصيت – تستغل نقاط الضعف في إدارة توصيلات TCP. دعنا أولاً نلقي نظرة على كيفية إنشاء توصيلة TCP. افترض أن عملية يجري تشغيلها على مضيف ما (زبون) تريد بدء توصيلة مع عملية أخرى على مضيف آخر (خادم). تقوم عملية تطبيق الزبون أولاً بإخبار بروتوكول TCP على الزبون بأنها تريد إنشاء توصيلة مع عملية على الخادم. بعد ذلك يمضي بروتوكول TCP على الزبون في إنشاء توصيلة TCP مع بروتوكول TCP على الخادم بالطريقة التالية:

- خطوة 1: يرسل بروتوكول TCP في ناحية الزبون أولاً قطعة TCP خاصة إلى بروتوكول TCP على الخادم. لا تحتوي تلك القطعة على أية بيانات من طبقة التطبيقات، ولكن أحد بتات الأعلام في ترويسة القطعة (انظر الشكل 3-29)، وهي البت SYN، تكون لها القيمة 1. لهذا السبب تعرف هذه القطعة الخاصّة باسم قطعة SYN. بالإضافة لذلك يختار الزبون رقماً تسلسلياً عشوائياً (client_isn) ويضع هذا الرقم في حقل الرقم التسلسلي لقطعة SYN الأولي. يتم تغليف هذه القطعة لتكوين وحدة بيانات IP وترسل إلى الخادم. استحوذت عملية الاختيار العشوائي الجيد لرقم client_isn للزبون على اهتمام كبير، وذلك لتفادي بعض الهجمات الأمنية على الشبكة [CERT 2001-09].
- خطوة 2: بمجرد وصول وحدة بيانات IP التي تتضمن قطعة SYN إلى مضيف الخادم (على افتراض أنها ستصل!)، ينتزع الخادم قطعة SYN من وحدة البيانات، ويخصّص مخازن TCP المؤقتة ومتغيرات توصيلة TCP، ثم يرسل قطعة "منح توصيلة" إلى TCP الخادم (سنرى في الفصل الثامن أن

تخصيص تلك المخازن المؤقتة والمتغيرات للتوصيلة قبل إكمال الخطوة الثالثة من خطوات المصافحة الثلاثية تجعل بروتوكول TCP عرضةً لهجوم لحجب الخدمة يُعرف بفيضان SYN). لا تتضمن قطعة "منح توصيلة" هي الأخرى أي بيانات من طبقة التطبيقات، غير أنها تحتوي على ثلاث معلومات مهمة في ترويسة القطعة:

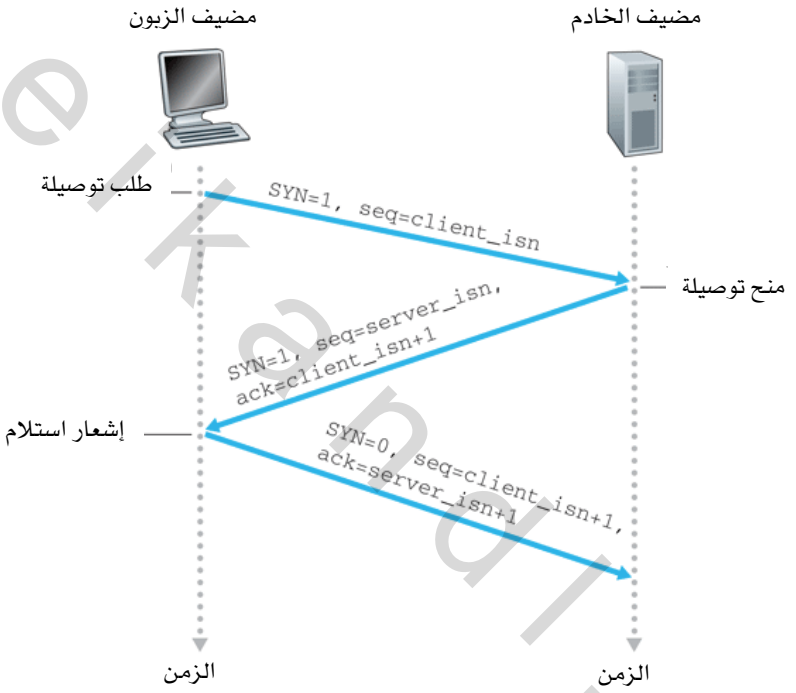
أولاً: البت SYN وقيمتها 1، ثانياً: حقل رقم إشعار الاستلام في ترويسة قطعة TCP وتكون له القيمة client_isn+1، وأخيراً: يختار الخادم رقمه التسلسلي الأول الخاص به server_isn ويضع قيمته في حقل الرقم التسلسلي بترويسة قطعة TCP.

في الواقع، لسان حال قطعة "منح توصيلة" هذه يقول: "استلمت رزمتك من نوع SYN لبدء توصيلة برقمك التسلسلي client_isn، وأوافق على إنشاء تلك التوصيلة، ورقمي التسلسلي الأولي هو server_isn". تعرف قطعة "منح توصيلة" باسم قطعة SYNACK.

- خطوة 3: عند استلام قطعة SYNACK يخصص الزبون أيضاً المخازن المؤقتة والمتغيرات الخاصة بالتوصيلة، ويرسل مضيف الزبون بعد ذلك إلى مضيف الخادم قطعة أخرى لإشعار الخادم بوصول قطعة "منح توصيلة" (يقوم الزبون بذلك بوضع القيمة server_isn+1 في حقل رقم إشعار الاستلام في ترويسة قطعة TCP). يكون للبت SYN القيمة صفر، حيث إن التوصيلة تم إنشاؤها. هذه المرحلة الثالثة من مراحل المصافحة الثلاثية يمكن أن تحمل بيانات من الزبون إلى الخادم في حقل payload للقطعة.

بمجرد الانتهاء من تلك الخطوات الثلاث، يمكن أن يُرسل مضيفا الخادم والزبون قطعاً تحتوي على بيانات إلى بعضهما البعض. في كل تلك القطع المُستقبلية تكون قيمة البت SYN صفراً. لاحظ أنه لإنشاء توصيلة، تبادل المضيفان ثلاثة رزم كما يبين الشكل 3-39، ولهذا السبب غالباً ما يطلق على هذا الإجراء لإنشاء التوصيلة اسم "المصافحة الثلاثية". يتم استكشاف عدة سمات لمصافحة TCP الثلاثية في التمارين في نهاية الفصل (مثلاً لماذا يحتاج الأمر إلى أرقام تسلسلية أولية؟

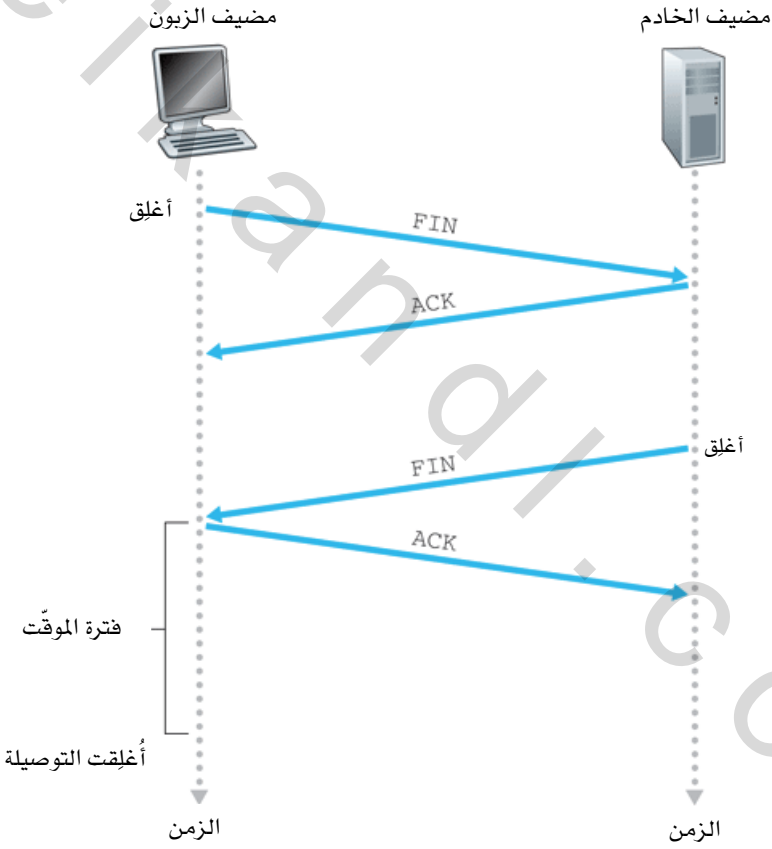
لماذا يتطلب الأمر مصافحة ثلاثية وليس فقط ثنائية؟). من الجدير بالملاحظة أن متسلق الصخور والمثبّت (الشخص الواقف على الأرض أسفل المتسلق والذي يضطلع بمهمة التعامل مع حبل أمان المتسلق) يستخدمان مصافحة ثلاثية تطابق مصافحة TCP لضمان أن كلا الجانبين جاهز قبل أن يبدأ المتسلق في الصعود.



الشكل 3-39 مصافحة بروتوكول TCP الثلاثية: تبادل القطع.

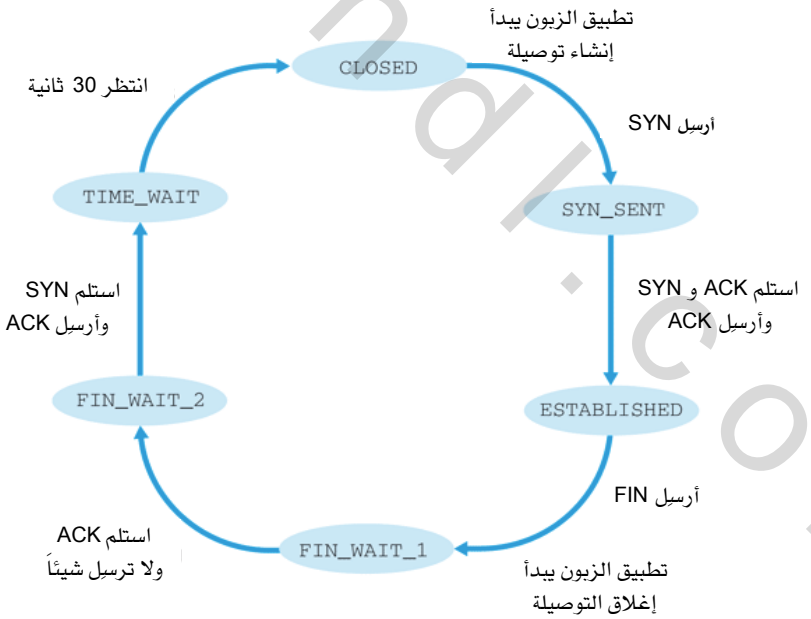
كل الأوقات الجميلة لا بد لها من نهاية، وهذا ينطبق أيضاً على توصيلات TCP. بوسع أيٍّ من العمليتين المشاركتين في توصيلة TCP إنهاء التوصيلة. وعند إنهاء توصيلة يتم إطلاق الموارد التي كانت تستخدمها تلك التوصيلة على كلٍّ من المضيفين (كالمخازن المؤقتة والمتغيرات) كي يتسنى استخدامها لتوصيلات أخرى. كمثال افترض أن الزبون قرّر إنهاء التوصيلة، كما هو مبين في الشكل 3-40. تُصدر عملية تطبيق الزبون أمراً بإنهاء التوصيلة. يؤدي ذلك إلى قيام بروتوكول

TCP على الزبون بإرسال قطعة TCP خاصة لها القيمة 1 للبت FIN في ترويسة القطعة إلى عملية الخادم (انظر الشكل 3-29). عندما يستلم الخادم هذه القطعة يرسل بدوره إلى الزبون إشعار استلام مقابل. بعد ذلك يرسل الخادم قطعة الإغلاق الخاصة به، والتي تكون قيمة البت FIN فيها 1 أيضاً. أخيراً يُرسل الزبون إشعاراً باستلام قطعة الإغلاق من الخادم. عند هذه النقطة، تكون كل موارد التوصيلة المنتهية على المضيفين قد تم إطلاقها وأصبحت متاحة لاستعمالات أخرى.



الشكل 3-40 إغلاق توصيلة TCP

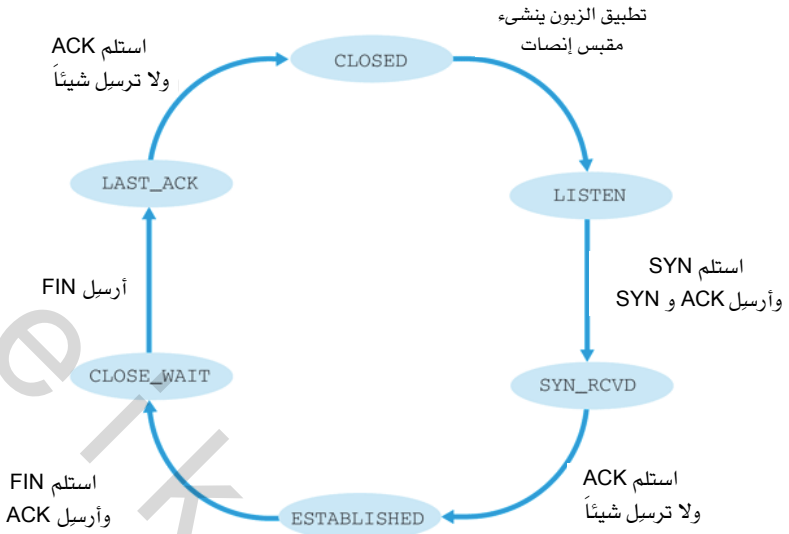
أثناء مدة توصيلة TCP يقوم بروتوكول TCP الذي يعمل على كلٍّ من المضيفين بالانتقال عبر عدة أوضاع مختلفة. يبين الشكل 3-41 تسلسلاً نمطياً للأوضاع التي يزورها بروتوكول TCP على الزبون. يبدأ زبون TCP في الوضع CLOSED (مغلق). يقوم التطبيق على جانب الزبون ببدء توصيلة TCP جديدة (بإنشاء كيان مقبس كما بيّنا في أمثلة جافا المذكورة في الفصل الثاني). يؤدي ذلك إلى قيام زبون TCP بإرسال قطعة SYN إلى خادم TCP. بعد إرسال القطعة SYN، يدخل زبون TCP وضع SYN_SENT (أُرسِلت قطعة SYN). ينتظر زبون TCP في ذلك الوضع وصول قطعة من خادم TCP تتضمّن إشعاراً باستلام قطعة الزبون السابقة وتحمل القيمة 1 للبت SYN. عند استلام تلك القطعة، يدخل زبون TCP وضع ESTABLISHED (التوصيلة شغّالة). في تلك الحالة يمكن لزبون TCP إرسال واستلام قطع TCP تتضمن بيانات (أي تولدها التطبيقات على مضيف الزبون).



الشكل 3-41 تسلسل نمطي لأوضاع TCP التي يزورها زبون TCP.

افترض أن تطبيق الزبون قرّر إنهاء التوصيلة – لاحظ أنه بوسع الخادم أيضاً أن يختار إنهاء التوصيلة – سيؤدي ذلك إلى إرسال زبون TCP قطعة TCP تحمل القيمة 1 للبت FIN، ومن ثم الدخول في وضع FIN_WAIT_1 منتظراً قطعة TCP من الخادم تتضمن إشعار استلام. عندما يتسلم الزبون تلك القطعة يدخل في وضع FIN_WAIT_2 منتظراً قطعة أخرى من الخادم تحمل القيمة 1 للبت FIN، وبعد استلام تلك القطعة يرسل زبون TCP إشعاراً باستلام قطعة الخادم ويدخل في وضع TIME_WAIT (الانتظار لفترة من الوقت) ليتيح الفرصة لبروتوكول TCP على الزبون لإرسال إشعار استلام نهائي في حالة فقد إشعار الاستلام السابق. ويعتمد وقت الانتظار الذي يقضيه الزبون في ذلك الوضع على إصدار TCP المستعمل، غير أن القيم المستخدمة عادةً هي 30 ثانية، ودقيقة، ودقيقتان. بعد الانتظار يتم إنهاء التوصيلة رسمياً وإطلاق كل الموارد التي كانت تستخدمها على جانب الزبون (بما في ذلك أرقام المنافذ).

يبين الشكل 3-42 تسلسلاً نمطياً للأوضاع التي يزورها بروتوكول TCP على جانب الخادم، على أساس أن الزبون يبدأ في إنهاء التوصيلة. الانتقالات بين الأوضاع المختلفة واضحة ولا تحتاج إلى شرح. وضّحنا في هذين المخططين للانتقال بين الأوضاع فقط الكيفية المعتادة لإنشاء وإنهاء توصيلة TCP، ولم نتعرّض لما يحدث في بعض السيناريوهات غير الطبيعية، على سبيل المثال عندما يرغب كلا الجانبين في إنشاء أو إنهاء توصيلة معاً في نفس الوقت. للمزيد من التفاصيل حول هذه النقطة وغيرها من القضايا المتقدّمة بخصوص بروتوكول TCP، ننصحك بالاطلاع على كتاب Stevens الشامل عن الموضوع [Stevens 1994].



الشكل 3-42 تسلسل نمطي لأوضاع TCP التي يزورها جانب الخادم من بروتوكول TCP.

افترضنا في مناقشتنا السابقة أن كلاً من الزبون والخادم مستعدان للاتصال، وبمعنى آخر أن الخادم يُنصت على المنفذ الذي يرسل إليه الزبون قطعة SYN. دعنا نتأمل ما يحدث عندما يستقبل مضيف قطعة TCP لها أرقام منافذ أو عنوان IP لا تتوافق مع أيٍّ من المقابس المستخدمة حالياً في المضيف. على سبيل المثال افترض أن مضيفاً يتسلم قطعة TCP من نوع SYN بمنفذ وجهة رقم 80، ولكن المضيف لا يقبل توصيلات على المنفذ 80 (أي أنه لا يقوم بتشغيل تطبيق خادم ويب على المنفذ 80). في تلك الحالة يرسل المضيف قطعة خاصّة Reset إلى المصدر تحمل القيمة 1 للبت RST (انظر الجزء 3-5-2)، ليخبر مضيف المصدر "ليس عندي مقبس لتلك القطعة، أرجو عدم إرسال القطعة مرة أخرى". أما عندما يتسلم مضيف قطعة بيانات UDP لا يتوافق منفذ الوجهة عليها مع أيٍّ من مقابس UDP الحالية على المضيف، يرسل المضيف وحدة بيانات ICMP خاصّة، كما سنبين في الفصل الرابع.

الآن وقد أصبحنا على دراية جيدة بإدارة توصيلات TCP، دعنا نزور مرة أخرى البرنامج nmap الخاص باستكشاف المنافذ لنرى عن قرب أكثر كيف

يعمل. لاستكشاف منفذ TCP بعينه، مثلاً منفذ 6789 على مضيف وجهة، يقوم برنامج nmap بإرسال قطعة SYN بمنفذ وجهة 6789 إلى ذلك المضيف. هناك ثلاث نتائج محتملة:

- يتسلم مضيف المصدر قطعة SYNACK من مضيف الوجهة، ولأن ذلك يعني وجود تطبيق يعمل على الوجهة بمنفذ TCP رقم 6789، فإن البرنامج nmap يعطي النتيجة "Open" (المنفذ مفتوح).
- يتسلم مضيف المصدر قطعة RST من مضيف الوجهة، وهذا يعني أن قطعة SYN قد وصلت لمضيف الوجهة إلا أنه ليس لديه تطبيق يعمل على منفذ TCP رقم 6789. بوسع المهاجم الآن على الأقل استنتاج أن القطع المُرسَلة إلى مضيف الوجهة بمنفذ 6789 لا يتم حجبها ببرنامج الحماية (firewall) على الطريق بين مضيفي المصدر والوجهة (سنناقش برامج الحماية في الفصل الثامن).
- لا يتسلم مضيف المصدر أي شيء، مما قد يعني أن القطعة SYN تم حجبها بأحد برامج الحماية على الطريق إلى مضيف الوجهة، ومن ثم لم تتمكن من الوصول إليه.

يعتبر برنامج nmap أداةً قويةً يمكن استخدامها في "فحص المبنى قبل السطو عليه"، ليس فقط لمنافذ TCP المفتوحة، ولكن أيضاً لمنافذ UDP المفتوحة، ولبرامج الحماية وإعداداتها، بل وحتى للإصدارات المستخدمة من التطبيقات وأنظمة التشغيل. يتم أغلب ذلك من خلال معالجة قطع TCP الخاصة بإدارة التوصيلات [Skoudis 2006]. إذا كنت الآن جالساً بالقرب من حاسب يعمل بنظام تشغيل لاينكس، فقد تريد تجربة البرنامج nmap بسرعة، فقط أدخل الأمر "nmap". أما إذا كنت تستخدم أحد أنظمة التشغيل الأخرى فبوسعك تنزيل البرنامج nmap من الموقع http://insecure.org/nmap.

بهذا نكون قد أنهينا مقدمتنا عن التحكم في الأخطاء وضبط التدفق في بروتوكول TCP. في الجزء 3-7 سنعود إلى بروتوكول TCP لنتناول بتفصيل أكثر كيفية التحكم في الازدحام. ولكن قبل ذلك دعنا نأخذ خطوة للوراء لكي نتفحص قضايا التحكم في الازدحام في سياق أوسع.

نبذة عن الأمن (Focus on Security)

هجوم فيضان SYN

رأينا من خلال مناقشتنا لمصافحة TCP الثلاثية أن الخادم يخصص ويضع القيم الأولية للمتغيرات والمخازن المؤقتة للتوصيلة كاستجابة لقطعة SYN التي يتسلمها. يقوم الخادم بعد ذلك بالرد بإرسال قطعة SYNACK، وينتظر وصول قطعة إشعار استلام ACK من الزبون، والتي تعتبر بمثابة الخطوة الثالثة والأخيرة في عملية المصافحة التي تسبق إنشاء توصيلة بالكامل. إذا لم يرسل الزبون إشعار الاستلام لإكمال الخطوة الثالثة من المصافحة الثلاثية فإنه في النهاية (غالباً بعد دقيقة أو أكثر) يقوم الخادم بإنهاء التوصيلة نصف المفتوحة، ويستردّ الموارد التي كان قد خصّصها لها.

يهيئ هذا الأسلوب لإدارة التوصيلة المسرح لهجوم شائع من نوع حجب الخدمة (DoS) يُعرف باسم فيضان SYN. في هذا الهجوم يرسل المهاجم عدداً كبيراً من قطع TCP SYN بدون إكمال الخطوة الثالثة من عملية المصافحة. يمكن مضاعفة أثر الهجوم بإرسال القطع SYN من مصادر متعدّدة، ومن ثم شن هجوم موزّع لحجب الخدمة (DDoS). بهذا الطوفان من قطع SYN يمكن بسرعة إنهاك موارد التوصيلات على الخادم حيث يتم تخصيصها لتوصيلات نصف مفتوحة (ولكنها لا تستخدم!). مع استنزاف مصادر الخادم بهذا الشكل يُحرَم زبائن شرعيون من الخدمة. ولقد كانت هجمات فيضان SYN تلك [CERT SYN 1996] بين هجمات حجب الخدمة الأولى التي تم توثيقها من قِبَل CERT [CERT 2007].

يمكن أن يكون هجوم فيضان SYN هجوماً مدمّراً فعلاً، لكن لحسن الحظ هناك دفاع فعّال ضده، يطلق عليه اسم كوكيز SYN (SYN cookies) والذي يُستخدم الآن في معظم أنظمة التشغيل الرئيسة [Skoudis 2006; Cisco SYN 2007; Bernstein 2007].

يعمل كوكيز SYN كالتّالي:

- عندما يتسلم الخادم قطعة SYN لا يعرف ما إذا كانت القطعة قادمة من مستخدم شرعي أو أنها جزء من هجوم فيضان SYN فإن الخادم لا ينشيء توصيلة TCP نصف مفتوحة لتلك القطعة، وإنما يقوم الخادم بتكوين رقم TCP تسلسلي أوّلي يكون دالة معقّدة (دالة تشفير الهاش hash function) في كلٍّ من عناوين IP للمصدر والوجهة وأرقام المنافذ لقطعة SYN، بالإضافة إلى رقم سري يعرفه الخادم فقط (ويستعمله لعدد كبير من التوصيلات). هذا الرقم التسلسلي الأوّلي والمحاك بعناية هو ما يسمّى بالكوكي. بعد

ذلك يرسل الخادم قطعة SYNACK بهذا الرقم التسلسلي الأوّلي الخاص. من المهم ملاحظة أن الخادم لا يتذكّر الكوكي ولا أي معلومات عن الأوضاع فيما يتعلق بقطعة SYN.

- إذا كان الزبون شرعياً، فإنه سيرد بقطعة إشعار استلام. عند استلام هذا الإشعار يقوم الخادم بالتحقق من أن الإشعار يقابل قطعة SYN التي أُرسلت في وقت سابق. كيف يقوم الخادم بذلك إذا كان لا يحتفظ في ذاكرته بأي شيء عن قطع SYN؟ كما قد تكون قد خمّنت، يتم ذلك باستخدام الكوكي. بالتحديد، إذا كان إشعار الاستلام شرعيا القيمة في حقل رقم إشعار الاستلام تزيد واحد عن الرقم التسلسلي لقطعة SYNACK التي أُرسِلت (انظر الشكل 3-39). يقوم الخادم باستخدام نفس الدالة مع نفس الحقول في قطعة إشعار الاستلام ونفس العدد السري. إذا كانت النتيجة تقل بواحد عن رقم إشعار الاستلام يستنتج الخادم أن إشعار الاستلام يناظر قطعة SYN سابقة ومن ثم فهو صحيح. يقوم الخادم بعد ذلك بإنشاء توصيلة مفتوحة بالكامل مع مقبس.
- من ناحية أخرى، إذا لم يرد الزبون بقطعة إشعار استلام، فإن قطعة SYN الأصلية لم تكن قد تسببت في أي ضرر عند الخادم، حيث إنه لم يقم بتخصيص أي موارد لها.

تتغلب طريقة كوكيز SYN بشكلٍ فعّال على خطر هجوم فيضان SYN. ومع ذلك هناك نوع من هجوم فيضان SYN يقوم فيه الزبون الماكر بإعادة قطعة إشعار استلام صحيحة لكل قطعة SYNACK يرسلها الخادم مما يؤدي إلى أن ينشىء الخادم توصيلات TCP مفتوحة بالكامل حتى في حالة استخدام نظام تشغيله لكوكيز SYN. كذلك إذا تم استخدام عشرات الآلاف من الزبائن – لكلٍ منهم عنوان مصدر IP مختلف – في الهجوم (أي هجوم موزّع لحجب الخدمة DDoS) يصبح من الصعب على الخادم التمييز بين المصادر الشرعية والخبيثة. وعليه فإن "هجوم المصافحة المكتملة" هذا يكون أكثر صعوبة في الدفاع ضده من هجوم فيضان SYN العادي.

6-3 مبادئ التحكم في الازدحام

في الفصول السابقة تناولنا كلاً من المبادئ العامة والآليات المحددة التي يستخدمها بروتوكول TCP لتوفير نقل موثوق للبيانات تحت ظروف فقد الرزم. وقد سبق أن ذكرنا أن فقد الرزم يرجع عادةً إلى فيض المخازن المؤقتة الموجودة في الموجّهات عند ازدحام الشبكة. ولذا فإن إعادة إرسال رزمة يعالج أحد أعراض ازدحام الشبكة وهو فقد قطعة بعينها من قطع طبقة النقل، ولكنه لا يعالج السبب الحقيقي لازدحام الشبكة – والذي يتلخص في محاولة الكثير من المصادر إرسال البيانات بمعدلات إرسال عالية جداً. لمعالجة سبب ازدحام الشبكة نحتاج إلى آليات للحد من قدرة الإرسال لدى المصادر لمواجهة ازدحام الشبكة.

في هذا الجزء سنتناول قضية التحكم في الازدحام في سياق عام لكي نفهم لماذا يُعتبر الازدحام شيئاً غير مرغوب فيه، وأثر هذا الازدحام على الأداء الذي توفره الشبكة لتطبيقات الطبقات الأعلى، وكذلك الطرق المختلفة التي يمكن اتباعها لتفادي حدوث ازدحام في الشبكة أو للتصرف إزاءه عند حدوثه. تعتبر هذه الدراسة العامة للسيطرة على الازدحام شيئاً مناسباً لأنها، كموضوع النقل الموثوق للبيانات، تحتل مرتبة متقدمة على قائمة أهم عشر مشاكل أساسية في مجال الشبكات. ثم بعد ذلك نختم هذا الجزء بمناقشة التحكم في الازدحام في خدمة معدّل البتات المتاح ((Available Bit Rate (ABR المستخدمة في شبكات نمط النقل غير المتزامن (ATM). أما الجزء التالي فيتضمن دراسة مفصلة لخوارزميات بروتوكول TCP للسيطرة على الازدحام.

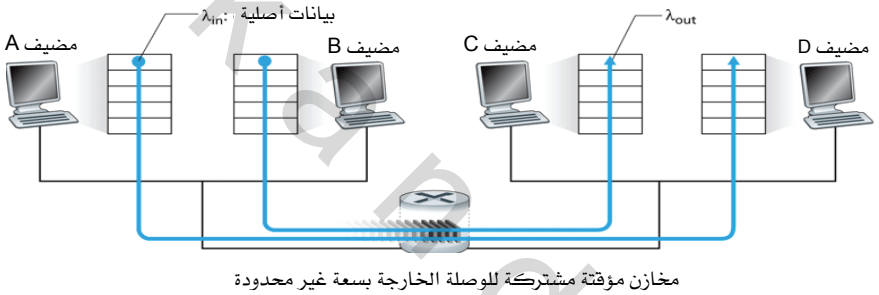
1-6-3 أسباب ازدحام الشبكة ومضاره

دعنا نبدأ دراستنا العامة للسيطرة على الازدحام بتناول ثلاثة سيناريوهات يحدث فيها ازدحام في الشبكة وتزداد تعقيداً بشكلٍ تدريجي. في كل حالة سنرى لماذا حدث الازدحام أساساً وماهو حجم الخسارة الناجمة عنه (من حيث الاستخدام غير الأمثل لموارد الشبكة والمستوى السيئ من الخدمة التي توفرها الشبكة

للأنظمة الطرفية). لن نركّز الآن على كيفية تفادي حدوث ازدحام أو كيفية التعامل معه عند حدوثه، ولكن سيكون التركيز على القضية الأسهل المتعلقة بفهم ما يحدث عندما تزيد المضيفات من معدلات إرسالها ومن ثم تزدحم الشبكة.

السيناريو الأول: مُرسِلان وموجّه بحيّز لانهائي للتخزين المؤقت

نبدأ بتناول ما قد يُعتبر أسهل سيناريو ازدحام ممكن: مضيفان A وB لكلٍ منهما توصيلة تشترك في وصلة واحدة بين المصدر والوجهة، كما هو موضح في الشكل 3-34.

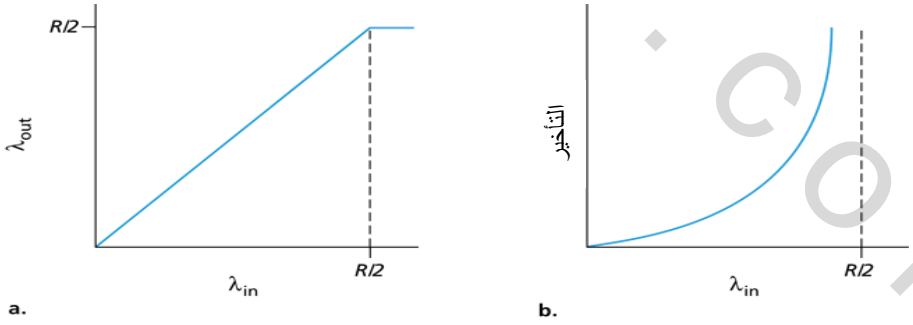


الشكل 3-43 سيناريو الازدحام رقم 1: توصيلتان تشتركان في وصلة واحدة بمخازن مؤقتة ذات سعة غير محدودة.

لنفترض أن التطبيق في المضيف A يرسل البيانات إلى التوصيلة (على سبيل المثال يدفع بالبيانات إلى بروتوكول طبقة النقل عبر المقبس) بمعدل إرسال متوسط مقداره λ_{in} بايت/ثانية. هذه البيانات كلها أصلية بمعنى أن كل وحدة بيانات ترسل إلى المقبس مرة واحدة فقط. افترض أن بروتوكول نقل البيانات التحتي بسيط، فالبيانات تغلّف ثم ترسل بدون أي آليات للتعافي من أخطاء البيانات (بإعادة الإرسال على سبيل المثال) أو لضبط التدفق أو للسيطرة على الازدحام. بإهمال الأعباء الإضافية الناجمة عن إضافة معلومات الترويسة على مستوى طبقة النقل والطبقات الأدنى، يكون معدل البيانات التي يرسلها المضيف A إلى الموجّه في هذا السيناريو

الأول λ_{in} بايت/ثانية. يعمل المضيف B بطريقة مماثلة، ولنفترض للتبسيط أنه أيضاً يرسل البيانات بمعدل λ_{in} بايت/ثانية. تمر الرزم من المضيفين A وB عبر الموجّه وعلى وصلة خارجة مشتركة لها سعة إرسال R بايت/ثانية. يحتوي الموجّه على مخزن مؤقت للرزم التي يتلقاها يُستخدم عندما يتجاوز معدل وصول الرزم سعة الإرسال الخاصة بالوصلة الخارجة. سنفترض أيضاً في هذا السيناريو الأول أن الموجّه لديه كمية لانهائية من حيّز التخزين.

تبيّن الرسوم البيانية في الشكل 3-44 أداء توصيلة المضيف A تحت ظروف هذا السيناريو الأول. يوضح الرسم البياني الأيسر الطاقة الإنتاجية لكل توصيلة (عدد البايتات التي تصل إلى المُستقبِل كل ثانية) كدالة في معدل الإرسال على التوصيلة. لمعدلات الإرسال من 0 إلى $R/2$، تساوي الطاقة الإنتاجية عند المُستقبِل معدل الإرسال عند المُرسِل – أي أن كل شيء يرسله المُرسِل يتم استلامه لدى المُستقبِل بعد فترة تأخير محدودة. ولكن عندما يتجاوز معدل الإرسال القيمة $R/2$، تتوقف الطاقة الإنتاجية عندالقيمة $R/2$. ينشأ هذا الحدّ الأعلى للطاقة الإنتاجية بسبب اشتراك توصيلتي المضيفين في سعة إرسال الوصلة، أي أن الوصلة ببساطة لا تستطيع توصيل البايتات إلى المُستقبِل بمعدل يتجاوز $R/2$. فمهما كان معدل الإرسال من كلٍّ من المضيفين A وB، فلن يحظى أي منهما أبداً بطاقة إنتاجية تتجاوز $R/2$.

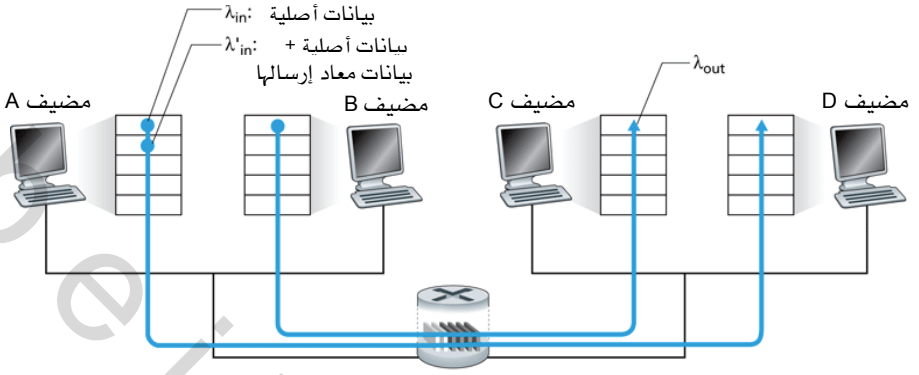


الشكل 3-44 سيناريو الازدحام رقم 1: الطاقة الإنتاجية والتأخير كدالة في معدل إرسال البيانات من الخادم.

قد يبدو تحقيق طاقة إنتاجية تعادل $R/2$ لكل توصيلة شيئاً جيداً في الواقع، لأن الوصلة تُستغَل بالكامل في توصيل الرزم إلى وجهاتها. غير أن الرسم البياني الأيمن في الشكل 3-44 يبين العواقب الوخيمة للتشغيل قرب سعة الإرسال القصوى لوصلة، فعند اقتراب معدل الإرسال على كلتا التوصيلتين من $R/2$ (بالزيادة من ناحية اليسار)، يزداد متوسط التأخير أكثر فأكثر. وعندما يتجاوز معدل الإرسال $R/2$، يصبح العدد المتوسط للرزم المنتظرة في الصف الخاص بالوصلة الخارجة بالموجّه غير محدود، ومن ثم يكون متوسط وقت التأخير بين المصدر والوجهة لانهائياً (على افتراض أن التوصيلات تبقى تعمل على ما يرام عند معدلات الإرسال هذه لفترات لانهائية وأنه تتوافر كمية لانهائية من حيّز التخزين المؤقت). وهكذا فبينما يكون التشغيل عند طاقة إنتاجية كلية قريبة من R أمراً مثالياً من منظور الطاقة الإنتاجية، إلا أنه يكون غير مرغوب فيه من حيث وقت التأخير المتزايد. حتى في هذا السيناريو المثالي للغاية، وجدنا عيباً واضحاً لازدحام الشبكة – تأخيرات انتظار كبيرة تعاني منها الرزم عندما تقترب معدلات وصول بيانات الرزم من سعة الإرسال للوصلة.

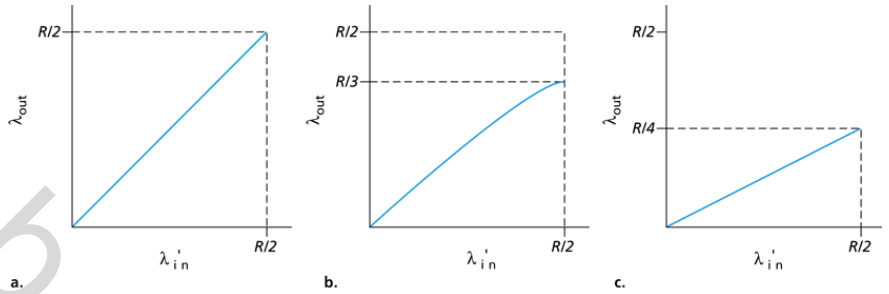
السيناريو الثاني: مُرسِلان وموجّه بحيّز محدود للتخزين المؤقت

دعنا الآن نُدخِل بعض التعديلات على السيناريو الأول من ناحيتين (انظر الشكل 3-45). أولاً: افترض أن حيّز التخزين المؤقت محدود. كنتيجة لهذا الافتراض الواقعي ستُفقَد الرزم التي تصل إلى الموجّه فتجد حيّز التخزين ممتلئاً عن آخره. ثانياً: افترض أن كل توصيلة توفر نقلاً موثوقاً للبيانات، فإذا فقدت في الموجّه رزمة تتضمن قطعة من قطع طبقة النقل، فإن المُرسِل سيعيد إرسالها في النهاية. ولأن الرزم يمكن أن يعاد إرسالها الآن، علينا أن نكون أكثر حذراً في استخدامنا لمصطلح معدل الإرسال. وبتحديد أكثر دعنا نرمز مرة أخرى للمعدل الذي يرسل به التطبيق بياناته الأصلية إلى المقبس بالرمز λ_{in} بايت/ثانية. أما المعدل الذي ترسل به طبقة النقل البيانات (قطع تحتوي بيانات أصلية وقطع معاد إرسالها) إلى الشبكة فسنرمز له بالرمز λ'_{in} بايت/ثانية. أحياناً يطلق على λ'_{in} الحمل المقدم إلى الشبكة (offered load).



الشكل 3-45 سيناريو الازدحام رقم 2: مضيفان (بإعادة إرسال) وموجّه بمخازن مؤقتة ذات سعة محدودة.

سيعتمد الأداء الذي يمكن تحقيقه تحت ظروف السيناريو الثاني بشكلٍ قوي الآن على كيفية القيام بإعادة الإرسال. أولاً خذ في الاعتبار الحالة غير الواقعية التي يتمكن فيها المضيف A بشكلٍ أو بآخر (بطريقة سحرية!) أن يحدد ما إذا كان هناك مكان خالٍ أم لا في المخزن المؤقت على الموجّه، ومن ثم يرسل رزمة فقط إذا توفر مكان للتخزين هناك. في هذه الحالة لا يحدث فقد للرزم ومن ثم لا يكون هناك داعٍ لإعادة الإرسال، وعليه يكون λ_{in} مساوياً لـ λ'_{in}، وتكون الطاقة الإنتاجية للتوصيلة λ_{in}. يبين الشكل 3-46 (a) هذه الحالة. من منظور الطاقة الإنتاجية يُعدّ الأداء فيها مثالياً، فكل شيء يتم إرساله يتم استلامه. لاحظ أن متوسط معدل الإرسال من المضيف لا يمكن أن يتجاوز $R/2$ في هذا السيناريو حيث افترضنا أن فقد الرزم لا يمكن أن يحدث أبداً.



الشكل 3-46 سيناريو الازدحام رقم 2: الأداء في حالة استخدام مخازن مؤقتة ذات سعة محدودة.

لنأخذ بعين الاعتبار بعد ذلك الحالة الأكثر واقعية بعض الشيء والتي يعيد فيها المُرسِل الإرسال فقط عندما يعرف بالتأكيد أن رزمة قد فُقدت (مرةً أخرى هذه الفرضية مسرفة في الخيال بعض الشيء، ولكن من الممكن أن يضبط المضيف فترة الموقّت عند قيمة كبيرة بما فيه الكفاية بحيث يتأكد عملياً من أن الرزمة التي لم يصل إشعار باستلامها تكون فعلاً قد فُقدت). في هذه الحالة يمكن أن يبدو الأداء كما هو مبين في الشكل 3-46 (b). لإدراك ما يحدث هنا تأمل الحالة التي يكون فيها الحمل المقدم للشبكة λ'_{in} (معدل إرسال البيانات الأصلية بالإضافة إلى البيانات المعاد إرسالها) يساوي $R/2$. حسب الشكل 3-46 (b)، عند هذه القيمة للحمل المقدم للشبكة، يكون معدل توصيل البيانات إلى تطبيق المُستقبِل $R/3$. وعليه، فإنه من كل $0.5R$ بايت/ثانية أُرسلت، هناك 0.333R بايت/ثانية (في المتوسط) بيانات أصلية و$0.166R$ بايت/ثانية (في المتوسط) بيانات يعاد إرسالها. نرى الآن كلفة أخرى لاستخدام شبكة مزدحمة - على المُرسِل أن يعيد الإرسال لكي يعوّض فقد الرزم بسبب فيض المخزن المؤقت بالموجّه.

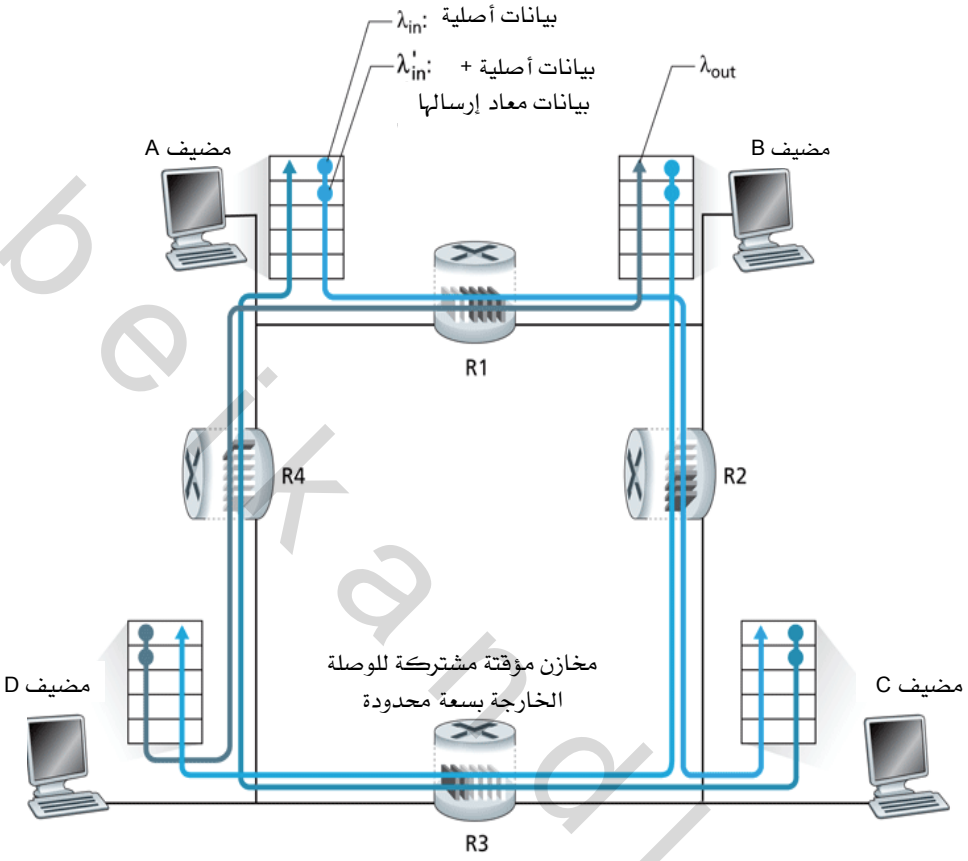
دعنا أخيراً نأخذ بعين الاعتبار الحالة التي قد تنقضي فيها فترة الموقّت عند المُرسِل قبل الأوان بحيث يعيد إرسال رزمة تأخّرت في صف الانتظار على الموجّه ولكنها لم تفقد بعد. في هذه الحالة قد تصل كلٌّ من رزمة البيانات الأصلية ورزمة إعادة الإرسال إلى المُستقبِل. وبالطبع فإن المُستقبِل يحتاج لنسخة واحدة فقط من تلك الرزمة وسيهمل رزمة إعادة الإرسال، وبذلك يعتبر الجهد الذي بذله الموجّه في إعادة

إرسال نسخة من الرزمة الأصلية جهداً مهدراً، لأن المُستقبل سيكون قد تسلم النسخة الأصلية من تلك الرزمة. وبدلاً من ذلك كان يمكن للموجّه استغلال سعة إرسال الوصلة بشكلٍ أفضل لإرسال رزمة مختلفة. هنا إذن نلاحظ كلفة أخرى لاستخدام شبكة مزدحمة - عمليات إعادة الإرسال بدون داعٍ من قِبَل المُرسِل بسبب التأخيرات الكبيرة، والتي قد تتسبب في جعل الموجّه يستخدم سعة الإرسال لوصلته لتوجيه نسخ غير مطلوبة من الرزم. يبين الشكل 3-46 (c) الطاقة الإنتاجية مقابل الحمل المقدّم للشبكة على افتراض أن كل رزمة ترسل (في المتوسط) مرتين بواسطة الموجّه. ونظراً لأن كل رزمة ترسل مرتين، فإن الطاقة الإنتاجية سيكون لها قيمة تقاربية (asymptotic value) مقدارها $R/4$ عندما يقترب الحمل المقدم للشبكة من $R/2$.

السيناريو الثالث: أربعة مُرسِلين، وموجّهين بحيّز محدود للتخزين المؤقت، ومسارات بعدة وصلات

في هذا السيناريو الأخير يقوم أربعة مضيفات بإرسال الرزم، كلٌّ على مسارات متطابقة تضم وصلتين، كما هو موضح في الشكل 3-47. نفترض مرة أخرى أن كل مضيف يستخدم آلية انقضاء فترة موقّت إعادة الإرسال لتوفير خدمة نقل موثوق للبيانات، وأن كل المضيفات لها نفس قيمة معدل إرسال البيانات الأصلية λ_{in}، وأن كل وصلات الموجّهات لها سعة إرسال R بايت/ثانية.

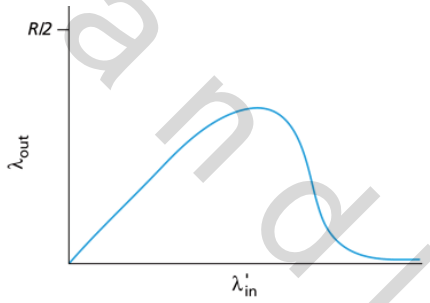
لنأخذ في الاعتبار التوصيلة من المضيف A إلى المضيف C مروراً بالموجّهين R1 R2. تشترك التوصيلة A↔C في الموجّه R1 مع التوصيلة D↔B وفي الموجّه R2 مع التوصيلة B↔D. للقيم الصغيرة جداً لـ λ_{in} يكون فيض المخازن المؤقتة نادراً (كما في سيناريوهات الازدحام الأول والثانى)، وتكون الطاقة الإنتاجية مساوية تقريباً للحمل المقدم للشبكة. للقيم الأكبر قليلاً لـ λ_{in} تكون الطاقة الإنتاجية المناظرة أكبر أيضاً، حيث يتم إرسال بيانات أصلية أكثر إلى الشبكة وتوصيلها إلى الوجهة، ويبقى الفيض نادراً. وهكذا فإنه لقيم λ_{in} الصغيرة، تؤدي الزيادة في λ_{in} إلى زيادة في λ_{out}.



الشكل 3-47 سيناريو الازدحام رقم 3: أربعة مضيفات مُرسِلة، وموجّهات بمخازن مؤقتة ذات سعة محدودة، ومسارات متعددة الوصلات.

بعد أن تناولنا حالة حركة مرور البيانات المنخفضة جداً، دعنا ندرس الآن الحالة التي تكون فيها λ_{in} (وبالتالي λ'_{in}) كبيرة جداً. خذ في الاعتبار الموجّه R2. إن حركة مرور بيانات A↔C الواصلة إلى الموجّه R2 (والتي تصل إليه بعد توجيهها من R1) يمكن أن يكون لها معدل وصول عند R2 كحد أقصى R، أي سعة الإرسال على الوصلة من R1 إلى R2، بغض النظر عن قيمة λ_{in}. إذا كانت λ'_{in} كبيرة جداً لكل التوصيلات (بما في ذلك التوصيلة B↔D)، فإن معدل وصول حركة المرور على B↔D عند R2 يمكن أن يكون أكبر بكثير من معدل وصول حركة المرور

على A↔C. ونظراً لأنه على حركتي المرور A↔C وB↔D أن تتنافسا على السعة المحدودة لحيّز التخزين المؤقت في الموجّه R2، فإن كمية حركة المرور A↔C التي تعبر R2 بنجاح (أي دون أن تُفقد هناك بسبب فيض المخزن المؤقت) تصبح أقل فأقل عندما يصبح الحمل المقدّم للشبكة من B↔D أكبر فأكبر. في النهاية عندما يقترب الحمل المقدم للشبكة من اللانهاية، فإن الفراغ في المخزن المؤقت في R2 سيُملأ فوراً برزمة من B↔D، وبذا تضمحل الطاقة الإنتاجية للتوصيلة A↔C عند الموجّه R2 لتصل إلى صفر. وهذا يعني بدوره أن الطاقة الإنتاجية للتوصيلة A↔C من طرف إلى طرف تصل إلى صفر في النهاية في حالة الازدحام الشديد. تؤدي تلك الاعتبارات إلى العلاقة المبينة في الشكل 3-48 والتي توضح الموازنة بين الحمل المقدم للشبكة في مقابل الطاقة الإنتاجية.



الشكل 3-48 سيناريو الازدحام رقم 3: الأداء في حالة موجّهات بمخازن مؤقتة ذات سعة محدودة ومسارات متعددة الوصلات.

يتضح السبب في النقص النهائي في الطاقة الإنتاجية عند زيادة الحمل المقدم للشبكة عندما نأخذ بعين الاعتبار كمية الشغل المهدر الذي تبذله الشبكة. في سيناريو حركة المرور العالية الذي تناولناه من قبل، كلما سقطت رزمة في موجّه الوصلة الثانية، فإن الشغل الذي بذله موجّه الوصلة الأولى في توجيه الرزمة إلى موجّه الوصلة الثانية يكون قد ضاع هباءً. كان من الممكن أن تكون الشبكة على نفس الحالة من الجودة (أو بتعبير أدق على نفس الحالة من السوء!) إذا كان الموجّه الأول ببساطة قد أهمل تلك الرزمة وبقي خاملاً. وبتحديد أكثر فإن قدرة

الإرسال التي استُخدمت في المسار الأول لإرسال تلك الرزمة إلى الموجّه الثاني كان يمكن استغلالها بشكلٍ أكثر فائدة بكثير لإرسال رزمة مختلفة (على سبيل المثال عند اختياره لرزمة يرسلها، قد يكون من الأفضل للموجّه إعطاء الأولوية لرزم قد قطعت شوطاً في رحلتها وعبرت عدة موجّهات في الطريق إلى وجهتها النهائية). ومن هنا يتضح لنا كلفة أخرى لإسقاط الرزم بسبب الازدحام – فعند إسقاط رزمة على مسارٍ ما، فإن سعة الإرسال التي استعملت في كل الوصلات السابقة لإيصال تلك الرزمة إلى تلك النقطة من الشبكة التي أُسقطت منها الرزمة تكون قد أُهدرت للأسف.

3-6-2 طرق التحكم في الازدحام

سنتناول في الجزء 3-7 بتفصيلٍ أكثر الطرق التي يتبعها بروتوكول TCP للتحكم في الازدحام. سنتعرف هنا فقط على أسلوبين عامّين للتحكم في الازدحام يُستخدمان في الواقع العملي، ونناقش بِنَى معمارية وبروتوكولات معينة لشبكات تتجسد فيها تلك الأساليب.

وعلى المستوى الأوسع يمكننا التمييز بين أساليب التحكم في الازدحام تبعاً لما إذا كانت طبقة الشبكة تقدم أي مساعدة واضحة لطبقة النقل لأغراض التحكم في الازدحام:

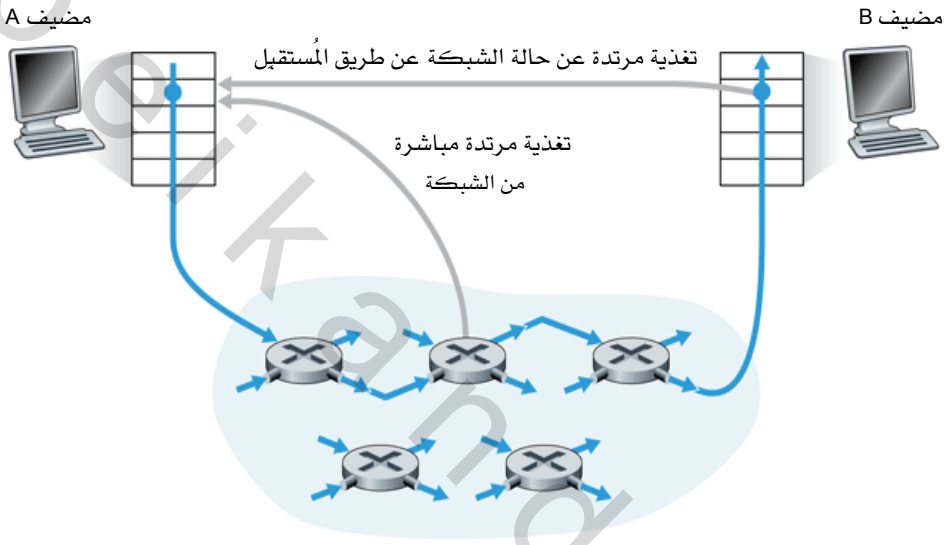
- أسلوب التحكم في الازدحام من طرف إلى طرف: في أساليب التحكم في الازدحام من طرف إلى طرف، لا تقدم طبقة الشبكة أي دعم محدد لطبقة النقل لأغراض التحكم في الازدحام، حتى اكتشاف وجود الازدحام في الشبكة يقع على عاتق الأنظمة الطرفية بناءً على ملاحظتها فقط لسلوك الشبكة (على سبيل المثال ملاحظة فقد الرزم والتأخير). سنرى في الجزء 3-7 أن بروتوكول TCP يجب أن يعتمد أسلوب من طرف إلى طرف هذا في التحكم في الازدحام، حيث لا يوفر بروتوكول طبقة الشبكة IP أي تغذية مرتدة (feedback) للأنظمة الطرفية بخصوص ازدحام الشبكة. إن فقد قطع بيانات TCP (والتي يدل عليها انقضاء فترة الموقّت أو وصول ثلاث إشعارات

استلام مكرّرة) ستؤخذ كمؤشر على ازدحام الشبكة، وبناءً عليه سيخفّض بروتوكول TCP من مقاس نافذته وفقاً لذلك. سنرى أيضاً اقتراحاً أحدث للتحكم في الازدحام في بروتوكول TCP يستخدم القيم المتزايدة لتأخير رحلة الذهاب والإياب كمؤشر على زيادة ازدحام الشبكة.

- أسلوب التحكم في الازدحام بمساعدة من الشبكة: في أسلوب التحكم في الازدحام بمساعدة من الشبكة، تقوم مكوّنات طبقة الشبكة (أي الموجّهات) بتقديم تغذية مرتدة صريحة إلى المُرسِل بخصوص حالة الازدحام في الشبكة. قد تكون تلك التغذية المرتدة بسيطة، مثلاً بت واحدة تشير إلى الازدحام في وصلة، وقد اتبع هذا الأسلوب في الشبكات الأولى كشبكات IBM SNA [Schwartz 1982] وDECnet [Jain 1989; Ramakrishnan 1990]، كما اقتُرح نفس الأسلوب مؤخراً لشبكات TCP/IP [Floyd TCP 1994; RFC 3168]، ويُستخدم أيضاً في شبكات ATM ضمن أسلوب ABR (معدل البتات المتاح) للتحكم في الازدحام والذي سنناقشه لاحقاً. هناك طرق أكثر تعقيداً لتوفير المزيد من تلك التغذية المرتدة، فمثلاً يسمح أحد أنواع بروتوكول ABR للتحكم في الازدحام على شبكات ATM – والذي سندرسه بعد قليل – للموجّه على الشبكة بإخبار المُرسِل صراحةً بمعدّل الإرسال الذي يمكن للموجّه دعمه على وصلة خارجة. وكذلك يوفر بروتوكول XCP (أي بروتوكول التحكم الصريح في الازدحام eXplicit Congestion control) (Protocol) تغذيةً مرتدة يقوم الموجّه بحسابها وإرسالها إلى كل مصدر ضمن ترويسة الرزمة، لتخبر المصدر بالكيفية التي يمكنه بها أن يزيد أو ينقص من معدل إرساله [Katabi 2002].

في أسلوب التحكم في الازدحام بمساعدة من الشبكة، غالباً ما تُرسَل المعلومات المتعلقة بالازدحام على شكل تغذية مرتدة إلى المُرسِل بإحدى طريقتين، كما هو موضحٌ في الشكل 3-49. قد تُرسَل التغذية المرتدة مباشرةً من موجّه على الشبكة إلى المُرسِل. عادةً ما يأخذ هذا الإخطار شكل رزمة خنق (choke packet) (لتقول على لسان الموجّه "أنا مزدحم!"). أما الشكل الثاني من الإخطارات فيحدث

عندما يقوم الموجّه بتعليم أو تحديث حقل في رزمة تعبره أثناء تدفّقها من المُرسِل إلى المُستقبِل للدلالة على حالة الازدحام. عند استلام حزمة معلّمة من موجّه يقوم المُستقبِل بإشعار المُرسِل بالازدحام. لاحظ أن هذا الشكل الأخير من الإخطارات يأخذ على الأقل وقتاً كاملاً لرحلة ذهاب وإياب.



الشكل 3-49 مساران للتغذية المرتدة بمعلومات عن ازدحام الشبكة.

3-6-3 مثال لأسلوب التحكم في الازدحام بمساعدة من الشبكة: أسلوب معدّل البتات المتاح (ABR) للسيطرة على الازدحام في شبكات ATM

سنختم هذا الجزء بدراسة حالة لخوارزمية للتحكم في الازدحام في بروتوكول ABR (معدّل البتات المتاح) المستخدم على شبكات نمط النقل غير المتزامن ((Asynchronous Transfer Mode (ATM) والذي يعتمد على أسلوب التحكم في الازدحام بمساعدة من الشبكة. نؤكد أن هدفنا هنا ليس وصف السمات المعمارية لشبكات ATM بالتفصيل، ولكن فقط توضيح بروتوكولٍ يسلك مسلكاً مختلفاً بدرجة كبيرة عن بروتوكول TCP على الإنترنت في أسلوب

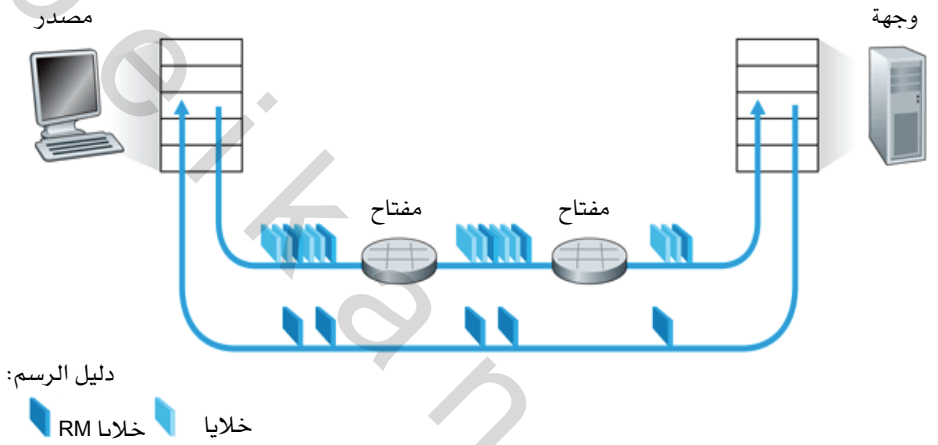
التحكم في الازدحام. سوف نقتصر فيما يلي على استعراض عدد من السمات القليلة لعمارة شبكات ATM والتي نحتاجها لفهم أسلوب ABR للتحكم في الازدحام.

من حيث المبدأ، تتبع شبكات ATM أسلوب الدائرة الافتراضية (VC) لتحويل رزم البيانات. تذكّر من مناقشتنا في الفصل الأول أن هذا يعني أن كل محوّل على المسار من المصدر إلى الوجهة يحتفظ لديه بحالة الدائرة الافتراضية من المصدر إلى الوجهة. هذه المعلومات عن حالة كل دائرة افتراضية تمكّن المحوّل من تتبع سلوك مصادر البيانات كلٍّ على حدة (مثلاً تتبع المعدلات المتوسطة للإرسال لديها) واتخاذ إجراءات معينة على المصدر للتحكم في الازدحام (كإخطار المُرسِل صراحةً بضرورة تخفيض معدل إرساله عندما يصبح المحوّل مزدحماً). إن الاحتفاظ بمعلومات الحالة لكل دائرة افتراضية على محوّلات الشبكة بهذا الشكل يجعل التحكم في الازدحام بمساعدة الشبكة مناسباً في شبكات ATM.

تم تصميم بروتوكول ABR لتوفير خدمة مرنة لنقل البيانات بطريقة تشبه بروتوكول TCP، فعندما تكون الشبكة غير مزدحمة، ينبغي أن تكون خدمة ABR قادرة على استغلال الحيّز الترددي المتاح. وعندما تكون الشبكة مزدحمة، ينبغي أن تُحِد الخدمة من معدل إرسالها إلى معدل إرسال أدنى محدّد مسبقاً. يمكنك الاطلاع على دراسة تدريبية عن طريقة التحكم في الازدحام وإدارة حركة مرور البيانات ببروتوكول ABR في [Jain 1996].

يبين الشكل 3-50 إطار التحكم في الازدحام في بروتوكول ABR. سنستخدم في مناقشتنا التالية مصطلحات ATM (على سبيل المثال سنستخدم "محوّل" بدلاً من "موجّه"، و"خلية" بدلاً من "رزمة"). في خدمة ABR، ترسل خلايا البيانات من مصدر إلى وجهة عبر سلسلة من المحوّلات بينهما، تتخلل خلايا البيانات تلك خلايا إدارة موارد الشبكة ((Resource Management (RM)، والتي يمكن استخدامها لنقل المعلومات المتعلقة بالازدحام بين المضيفات والمحوّلات. عندما تصل خلية RM إلى وجهة، يتم تدويرها وإعادتها إلى المُرسِل (ربما بعد تعديل محتوياتها

بواسطة الوجهة). يمكن أيضاً لمحوّل إنشاء خلية RM جديدة بنفسه وإرسالها مباشرةً إلى المصدر. وعليه يمكن استخدام خلايا RM لتوفير تغذية مرتدة من الشبكة مباشرةً إلى المُرسِل أو بطريقة غير مباشرة عن طريق المُستقبِل، كما هو مبين في الشكل 3-50.



الشكل 3-50 آلية التحكم في الازدحام مع خدمة ABR على شبكات ATM.

يعتمد بروتوكول ABR في التحكم في الازدحام على أسلوب مبني على معدل إرسال البتات، بمعنى أن المُرسِل يحسب بشكلٍ محدد أقصى معدل إرسال يمكن أن يستخدمه وينظّم نفسه وفقا لذلك. يوفر بروتوكول ABR ثلاث آليات لنقل المعلومات المتعلقة بالازدحام من المحوّلات إلى المُستقبِل:

- البت EFCI: تحتوي كل خلية بيانات على بت للبيان الصريح للازدحام الأمامي (أي في اتجاه الوجهة) (Explicit Forward Congestion Indication ((EFCI). يمكن لمحوّل في شبكة مزدحمة وضع البت EFCI بالقيمة 1 لإخطار مضيف الوجهة بوجود ازدحام. يجب على مضيف الوجهة فحص البت EFCI على كل خلايا البيانات التي يتسلمها. عند وصول خلية RM إلى المضيف، إذا كانت البت EFCI في آخر خلية بيانات تم استلامها

تساوى 1، فإن مضيف الوجهة يضع 1 في بت CI (إشارة الازدحام Congestion Indication) ضمن خلية RM ويُرسلها في الاتجاه الآخر إلى المُرسِل. وهكذا فباستخدام البت EFCI في خلايا البيانات والبت CI في خلايا RM يمكن إخطار المُرسِل بحالة الازدحام عند محوّل بالشبكة.

- زوج البتات CI وNI: كما ذكرنا سابقاً تتخلل خلايا البيانات خلايا RM. إن نسبة خلايا RM الموزّعة وسط خلايا البيانات هي متغير قابل للضبط، يبدأ بقيمة أصلية مقدرها خلية RM واحدة لكل 32 خلية بيانات. تتضمن خلايا RM تلك زوجاً من البتات يمكن تغييرهما بواسطة محوّل على شبكة مزدحمة هما البت CI (إشارة الازدحام Congestion Indication) والبت NI (عدم الزيادة No Increase). بتحديد أكثر يمكن للمحوّل وضع القيمة 1 للبت NI في خلية RM تعبره في حالة ازدحام معتدل للشبكة، كما يمكنه وضع القيمة 1 للبت CI عند ظروف الازدحام الشديد. عندما يتسلم مضيف الوجهة خلية RM يقوم بإعادة تلك الخلية إلى المُرسِل بنفس قيم زوج البتات CI وNI التي استلمها (إلا أن البت CI يمكن أن تتغير إلى 1 في مضيف الوجهة كنتيجة لآلية البت EFCI المذكورة أعلاه).
- معدل الإرسال المحدد (Explicit Rate (ER)): تتضمن كل خلية RM أيضاً حقل معدل الإرسال المحدد (ER) والذي يتألف من بايتين. يمكن لمحوّل مزدحم تقليل قيمة الحقل ER في خلية RM تعبره. وبهذه الطريقة فإن حقل ER سيتم ضبطه بحيث يمثل الحد الأدنى لمعدل الإرسال الذي يمكن دعمه على كل المحوّلات الموجودة على المسار من المصدر إلى الوجهة.

يقوم مضيف المصدر في بروتوكول ATM بضبط معدل الإرسال الذي يمكنه استخدامه لبث الخلايا بناءً على قيم زوج البتات CI وNI والحقل ER في خلية RM الراجعة إليه من مضيف الوجهة. إن القواعد التي تحكم عملية ضبط معدل الإرسال

تلك معقدة ومجهِدة نوعًا ما، وعلى القارئ المهتم بالمزيد من التفاصيل مراجعة [Jain 1996].

7-3 التحكم في الازدحام في بروتوكول TCP

نعود في هذا الجزء إلى دراستنا لبروتوكول TCP. كما عرفنا في الجزء 3-5، يوفر TCP خدمة نقل موثوقة للبيانات بين عمليتين يتم تنفيذهما على مضيفين مختلفين. من المكونات الرئيسة الأخرى لبروتوكول TCP آلية التحكم في الازدحام. كما بينّا في الجزء السابق يجب أن يستخدم TCP أسلوب التحكم في الازدحام من طرف إلى طرف بدلاً من التحكم بمساعدة من الشبكة، نظراً لأن طبقة الشبكة (IP) لا توفر أي تغذية مرتدة إلى الأنظمة الطرفية بخصوص ازدحام الشبكة.

يتلخص الأسلوب الذي يتبعه بروتوكول TCP في جعل كل مُرسِل يُحِد من معدل إرساله للبيانات كدالة في الازدحام المحسوس للشبكة. فإذا أحس مُرسِل TCP أن الازدحام قليل على المسار بينه وبين الوجهة، فإن المُرسِل يزيد من معدل إرساله للبيانات. أما إذا شعر المُرسِل بأن هناك ازدحاماً على المسار، فإنه يخفض من معدل إرساله. غير أن هذا الأسلوب يطرح ثلاثة أسئلة. أولاً: كيف يمكن لمُرسِل TCP الحد من المعدل الذي يرسل به البيانات إلى التوصيلة؟ ثانياً: كيف يدرك المُرسِل أن هناك ازدحاماً في الشبكة على المسار بينه وبين الوجهة؟ وثالثاً: ما هي الخوارزمية التي يجب على المُرسِل استعمالها لتغيير معدل إرساله كدالة في الازدحام المحسوس في الشبكة من طرف إلى طرف؟ سنتناول هذه القضايا الثلاث في سياق الإصدار رينو Reno لبروتوكول TCP والذي يُستخدم في معظم أنظمة التشغيل الحديثة [Padhye 2001]. وللحفاظ على الطابع العملي للمناقشة سنفترض أن مُرسِل TCP يقوم بإرسال ملف كبير.

لنتناول أولاً كيف يُحد مُرسِل TCP من معدل إرساله للبيانات على التوصيلة. رأينا في الجزء 3-5 أن كلاً من جانبي توصيلة TCP يتكون من مخزن مؤقت

للاستقبال، ومخزن مؤقت للإرسال، وعدة متغيرات (LastByteRead،RcvWindow ، وهكذا). تتطلب آلية TCP للتحكم في الازدحام أن يتابع كل جانب من جانبي التوصيلة قيمة متغير إضافي يطلق عليه نافذة الازدحام (congestion window) والذي يرمز له بالرمز CongWin. يفرض المتغير CongWin قيداً على المعدل الذي يمكن لمُرسِل TCP استخدامه لإرسال البيانات إلى الشبكة. بالتحديد لا يُسمح لكمية البيانات التي أُرسلت ولم يتم الإشعار باستلامها عند مُرسِل أن تتجاوز القيمة الأصغر لكلٍّ من CongWin وRcvWindow، أي:

$$\text{LastByteSent} - \text{LastByteAcked} \le \min\{\text{CongWin}, \text{RcvWindow}\}$$

لكي نركّز على التحكم في الازدحام (وليس ضبط التدفق)، دعنا نفترض من الآن فصاعداً أن مُستقبِل TCP له مخزن استلام مؤقت كبير جداً بحيث يمكن إهمال تأثير التقييد الناجم عن نافذة الاستقبال – بمعنى أن كمية البيانات التي أرسلت ولم يصل إشعار باستلامها بعد عند المُرسِل تُحدّها فقط قيمة المتغير CongWin.

يُحِد القيد أعلاه من كمية البيانات التي يمكن إرسالها بدون انتظار وصول إشعار باستلامها عند المُرسِل، وبالتالي فإنه يُحد بشكلٍ غير مباشر من معدل إرسال البيانات من المُرسِل. لتوضيح ذلك خذ في الاعتبار توصيلة يكون فيها فقد وتأخير الرزم ضئيلين بحيث يمكن إهمالهما. وعليه فإنه تقريباً في بداية كل *RTT* يسمح هذا القيد للمُرسِل بإرسال CongWin بايت من البيانات إلى التوصيلة، وفي نهاية فترة *RTT* يتسلم المُرسِل إشعارات بوصول البيانات. وهكذا فإن معدل إرسال البيانات يساوي تقريباً Congwin/*RTT* بايت/ثانية. باختيار قيمة CongWin يمكن للمُرسِل ضبط المعدل الذي يرسل به البيانات على وصلته.

لنرَ الآن كيف يُدرك مُرسِل TCP أن هناك ازدحاماً على المسار بينه وبين الوجهة. دعنا نعرّف "حدث الفقد" في مُرسِل TCP بحدوث انقضاء فترة موقّت أو تسلم ثلاثة إشعارات استلام مكرّرة من المُستقبِل (تذكّر من مناقشتنا في الجزء 3-4-5 لحدث انقضاء فترة الموقّت في الشكل 3-33 والتعديل اللاحق لتضمين الإعادة

السريعة للإرسال عند تلقي ثلاثة إشعارات استلام مكرّرة). عند حدوث ازدحام شديد، يفيض واحد (أو أكثر) من المخازن المؤقتة على الموجّهات على المسار، مما يؤدي إلى سقوط وحدة بيانات (تتضمن قطعة TCP)، وبالتالي ينشأ عن سقوط وحدة البيانات تلك "حدث فقد" لدى المُرسِل – إمّا على شكل انقضاء فترة موقّت أو تلقي ثلاثة إشعارات استلام مكرّرة – والذي يعتبره المُرسِل إشارة لوجود ازدحام على المسار من المُرسِل إلى المُستقبِل.

بعد أن تناولنا كيفية اكتشاف وجود الازدحام، دعنا الآن نرى الحالة الأكثر تفاؤلاً عندما تكون الشبكة لا تعاني من الازدحام، أي عندما لا يكون هناك أحداث فقد. في هذه الحالة سيتلقى مُرسِل TCP إشعارات الاستلام التي ينتظرها لكل القطع التي تم إرسالها. وكما سنرى يأخذ مُرسِل TCP وصول تلك الإشعارات كدليل على أن الأمورعلى ما يرام – أي أن كل القطع التي يتم إرسالها عبر الشبكة تُسلّم بنجاح إلى الوجهة، ومن ثم يزيد قيمة نافذة الازدحام عنده (وبالتالي معدل الإرسال الذي يستخدمه). لاحظ أنه إذا وصلت إشعارات الاستلام بمعدل بطيء (مثلاً بسبب تأخير كبير عبر مسار الشبكة من طرف إلى طرف أو إذا كان ذلك المسار يتضمن وصلة ذات حيّز تردد صغير (أي سعة إرسال منخفضة)، فإن الزيادة في نافذة الازدحام ستكون بمعدل بطيء نسبياً. وبالعكس إذا وصلت إشعارات الاستلام بمعدّل سريع، فسيتم زيادة نافذة الازدحام بسرعة أكبر. نظراً لأن بروتوكول TCP يستخدم إشعارات الاستلام لتحقيق زيادة في نافذة الازدحام (كساعة توقيت)، يُقال عن بروتوكول TCP إنه ذاتي التوقيت.

بوسعنا الآن بحث تفاصيل الخوارزمية التي يستخدمها مُرسِل TCP لضبط معدل إرساله كدالة في الازدحام المحسوس على الشبكة. تلك هي خوارزمية TCP الشهيرة للتحكم في الازدحام. تتألف الخوارزمية من ثلاثة مكونات رئيسة: (1) زيادة خطية ونقصان أسّي، (2) البداية البطيئة، و(3) ردّ الفعل لأحداث انقضاء فترة الموقّت.

الزيادة الخطية والنقصان الأسّي

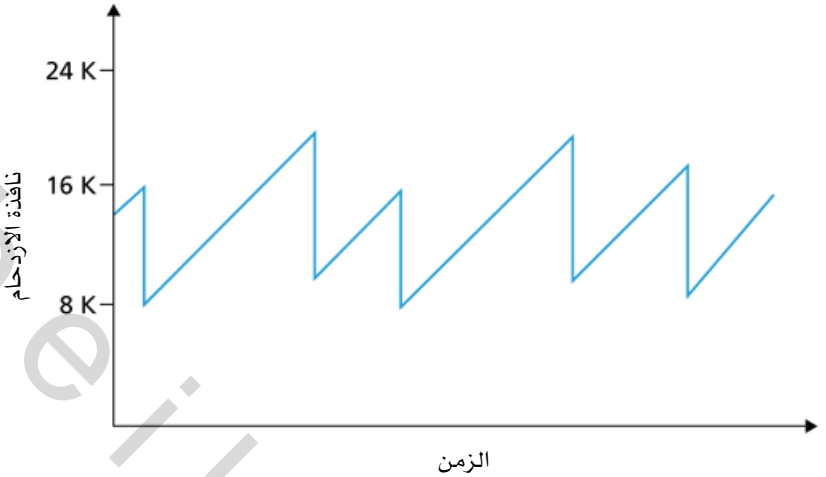
تكمن الفكرة الأساسية وراء التحكم في الازدحام ببروتوكول TCP في تخفيض المُرسِل معدل إرساله (بتصغير نافذة الازدحام CongWin لديه) عند حدوث فقد في الرزم. نظراً لأن توصيلات TCP الأخرى التي تعبر نفس الموجّهات المزدحمة يُحتمل أن تعاني هي الأخرى من "أحداث فقد"، فمن المحتمل أن تقوم هي الأخرى بتخفيض معدلات إرسالها بخفض قيم متغيرات CongWin لديها. وعليه تكون المحصلة النهائية قيام المصادر التي تستخدم المسارات المزدحمة بتخفيض المعدلات التي تستخدمها لإرسال البيانات عبر الشبكة، والذي يُتوقع أن يؤدي بدوره إلى تخفيف حدة الازدحام في المسارات المزدحمة. ولكن ما مقدار التقليل اللازم في نافذة الازدحام على المُرسِل على إثر حدث فقد؟ يستخدم بروتوكول TCP ما يعرف بأسلوب "النقصان الأسّي"، حيث يقلل قيمة CongWin الحالية إلى النصف بعد كل حدث فقد. وبالتالي فإذا كانت قيمة CongWin الحالية على مُرسِل TCP تساوي 20 كيلوبايت وتم اكتشاف حدث فقد، يتم تقليل قيمة CongWin إلى 10 كيلوبايت. إذا وقع حدث فقد آخر، تخفّض CongWin أكثر إلى 5 كيلوبايت. وهكذا يستمر تخفيض قيمة CongWin بحيث لا يُسمح لها بأن تقل عن الحجم الأقصى للقطعة MSS (هذا وصف كلي للصورة الكبيرة لكيفية تغيير نافذة الازدحام بعد حدث فقد. في الواقع فإن الأمر أكثر تعقيداً من ذلك بعض الشيء، كما سنرى قريباً).

بعد أن عرفنا كيف يُخفّض مُرسِل TCP معدل إرساله إزاء اكتشاف ازدحام في الشبكة، من الطبيعي أن نتناول في الخطوة التالية كيف يقوم مُرسِل TCP بزيادة معدل إرساله عندما يدرك أن الشبكة غير مزدحمة، أي عندما يتلقى المُرسِل إشعارات الاستلام المتعلقة بكل القطع التي أُرسلت. إن السبب الجوهري لزيادة معدل الإرسال هو أنه عند عدم وجود ازدحام محسوس يكون هناك احتمال توفر حيّز تردد غير مستعمل يمكن استغلاله لصالح توصيلة TCP. في مثل هذه الظروف يزيد مُرسِل TCP من نافذة الازدحام لديه ببطء، متقصّياً بحذر الحيّز الترددي الإضافي المتوفر على المسار من طرف إلى طرف. يقوم مُرسِل TCP بذلك بزيادة قيمة

نافذة الازدحام رويداً رويداً في كل مرة يتلقى فيها إشعار استلام بهدف زيادتها بـ MSS واحدة في كل *RTT* [RFC 2581]، ويمكن تحقيق ذلك بعدة طرق.

من الطرق المستخدمة بكثرة أن يقوم المُرسِل بزيادة CongWin بمقدار (MSS×MSS/CongWin) بايت في كل مرة يصله فيها إشعار استلام جديد. على سبيل المثال إذا كانت قيمة *MSS* تساوي 1,460 بايتاً وقيمة CongWin تساوي 14,600 بايت، يتم إرسال 10 قطع خلال وقت رحلة الذهاب والعودة *RTT*، ويؤدي وصول كل إشعار استلام (بافتراض استخدام إشعار استلام لكل قطعة) إلى زيادة نافذة الازدحام بمقدار *MSS*/10، وعليه فبعد وصول إشعارات الاستلام للقطع العشر التي أُرسلت خلال *RTT*، تكون نافذة الازدحام قد زادت بمقدار *MSS*، وهو المطلوب.

الخلاصة: إنّ مُرسِل TCP يزيد من معدل إرساله بطريقة الإضافة عندما يدرك أن المسار من طرف إلى طرف خالٍ من الازدحام، ويُنقص المعدل بطريقة أسّية عندما يكتشف (عن طريق حدث فقد للرزم) أن المسار مزدحم. لهذا السبب غالباً ما يعرف أسلوب التحكم في الازدحام في بروتوكول TCP بأنه خوارزمية من نوع الزيادة الخطية والنقصان الأسّي ((Additive Increase Multiplicative Decrease (AIMD). تُعرف مرحلة الزيادة الخطية في عملية التحكم في الازدحام ببروتوكول TCP بمرحلة تجنب الازدحام. تمر قيمة CongWin بدورة متكرّرة تبدأ فيها بالزيادة الخطية ثم تهبط فجأة إلى نصف قيمتها الحالية (عند وقوع حدث فقد)، ومن ثم تأخذ التغيرات في تلك القيمة شكل سن المنشار في توصيلات TCP طويلة الأمد، كما هو مبين في الشكل 3-51.

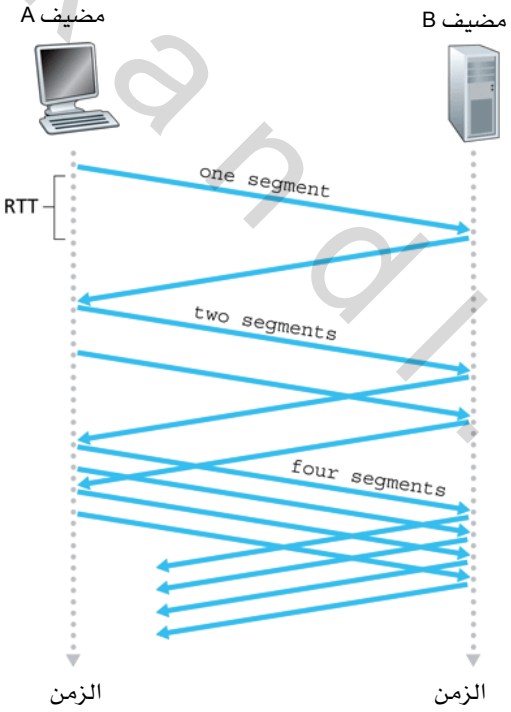


الشكل 3-51 التحكم في الازدحام بزيادة خطية ونقص أسّي.

البداية البطيئة (Slow Start)

عندما تبدأ توصيلة TCP في العمل تأخذ CongWin قيمة مبدئية تكون عادةً مساوية لـ MSS واحدة [RFC 3390]، ومن ثم يكون معدل الإرسال الأوّلي MSS/RTT تقريباً. كمثال إذا كانت MSS = 500 بايت وRTT = 200 ميللي ثانية، فإن معدل الإرسال الناتج يكون 20 كيلوبت/ثانية. نظراً لأن حيّز التردد المتوفر للوصلة قد يكون أكثر بكثير من *MSS/RTT* يكون من المؤسف زيادة معدل الإرسال فقط بشكلٍ خطي والانتظار لوقت طويل أكثر مما ينبغي حتى يرتفع معدل الإرسال إلى مستوى معقول. وعليه فبدلاً من زيادة المعدل بشكلٍ خطي أثناء تلك المرحلة الأولية، يقوم مُرسِل TCP بزيادة المعدل تصاعدياً (بشكلٍ أسّي) بمضاعفة قيمة CongWin كل *RTT*. يواصل مُرسِل TCP زيادة معدل الإرسال بطريقة أسّية حتى يقع حدث فقد، عندئذٍ تُنصّف قيمة CongWin وبعد ذلك تنمو بشكلٍ خطي، كما سبق وصفه من قبل. وهكذا فأثناء تلك المرحلة الأولية التي تدعى البداية البطيئة ((Slow Start (SS))، يبدأ مُرسِل TCP الإرسال بمعدل بطيء (ومن هنا كانت التسمية البداية البطيئة) ولكنه يزيد معدل الإرسال بطريقة أسّية، يحدث ذلك بزيادة المُرسِل قيمة CongWin في كل مرة بـ MSS واحدة عن

كل قطعة تم إرسالها وتلقّى المُرسِل إشعاراً باستلامها. كما هو مبين في الشكل 3-52 يرسل TCP القطعة الأولى إلى الشبكة وينتظر إشعار استلام. إذا وصل إشعار باستلام تلك القطعة قبل وقوع حدث فقد، يزيد مُرسِل TCP نافذة الازدحام بـ MSS واحدة وبالتالي يبعث بقطعتين بأقصى حجم ممكن. إذا وصل إشعار استلام لهذين القطعتين قبل وقوع حدث فقد، يزيد المُرسِل نافذة الازدحام بـ MSS لكل واحدة من هاتين القطعتين اللتين تم الإشعار باستلامهما، ومن ثم تصبح نافذة الازدحام 4×MSS ويقوم المُرسِل ببث أربع قطع بأقصى حجم ممكن. يستمر هذا الإجراء طالما استمرت إشعارات الاستلام في الوصول، إلى أن يقع حدث فقد في نهاية الأمر. وهكذا تتضاعف عملياً قيمة CongWin كل *RTT* أثناء مرحلة البداية البطيئة.



الشكل 3-52 البداية البطيئة لبروتوكول TCP

ردّ الفعل لأحداث انقضاء فترة الموقّت

تتلخص الصورة التي رسمناها حتى الآن لنافذة الازدحام ببروتوكول TCP في النمو الأسي بدءاً من MSS واحدة (أثناء مرحلة البداية البطيئة) إلى حين وقوع حدث فقد، حيث يبدأ عندئذٍ نمط سن المنشار. وعلى الرغم من أن هذه الصورة قريبة من وصف الواقع بدقة، إلا أننا سنكون مقصرين إذا لم نذكر أن أسلوب التحكم في الازدحام ببروتوكول TCP يستجيب في الواقع لحدث الفقد الذي يُكتشف بانقضاء فترة موقّت بشكلٍ مختلف عن حدث الفقد الذي يُكتشف بوصول ثلاثة إشعارات استلام مكرّرة. فبعد وصول ثلاثة إشعارات استلام مكرّرة، يتصرّف بروتوكول TCP بالطريقة التي وصفناها آنفاً، حيث تنصّف نافذة الازدحام وبعد ذلك تزيد قيمتها بشكلٍ خطي. أما بعد وقوع حدث انقضاء فترة موقّت، فإن مُرسِل TCP يدخل مرحلة البداية البطيئة، أي يضبط قيمة نافذة الازدحام عند MSS واحدة وبعد ذلك تنمو قيمة النافذة بطريقة أسّية. تواصل قيمة النافذة نموها التصاعدي حتى تصل CongWin إلى نصف قيمتها قبل وقوع حدث انقضاء فترة الموقّت مباشرة. عند تلك النقطة، تبدأ CongWin في النمو بشكلٍ خطي، تماماً كما هو الحال بعد تلقي ثلاثة إشعارات استلام مكرّرة.

يدير بروتوكول TCP هذه الديناميكية المعقدة بالاحتفاظ بمتغير يسمى العتبة (threshold) يحدد قيمة نافذة الازدحام التي تنتهي عندها مرحلة البداية البطيئة وتبدأ مرحلة تجنب الازدحام. في البداية يأخذ متغير العتبة قيمة كبيرة (65 كيلوبايت عملياً [Stevens 1994]) بحيث لا يكون له تأثير أولي. وعند وقوع حدث فقد، توضع قيمة العتبة مساويةً لنصف القيمة الحالية لـ CongWin. على سبيل المثال، إذا كانت قيمة نافذة الازدحام 20 كيلوبايت مباشرةً قبل وقوع حدث فقد، فإن قيمة العتبة تصبح 10 كيلوبايت، وتبقى هذه القيمة لحين وقوع حدث الفقد التالي.

بعد أن وصفنا متغير العتبة، بوسعنا الآن أن نصف بالضبط كيف يتصرّف المتغير CongWin بعد حدث انقضاء فترة موقّت. كما أشرنا من قبل يدخل مُرسِل

TCP مرحلة البداية البطيئة بعد حدث انقضاء فترة موقّت. أثناء مرحلة البداية البطيئة يتم زيادة قيمة CongWin بطريقة أسّية بسرعة حتى تصل CongWin إلى العتبة، عند ذلك يدخل TCP مرحلة تجنب الازدحام، وخلالها تزداد قيمة CongWin بشكلٍ خطي كما وصفنا في وقت سابق.

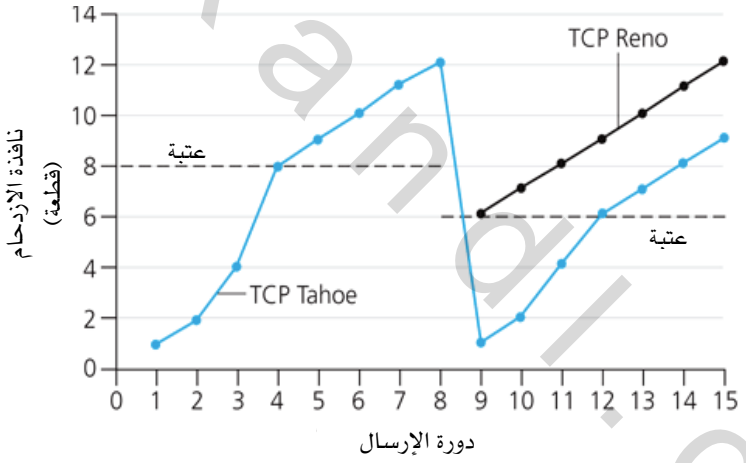
يلخص الجدول 3-3 مناقشتنا لخوارزمية التحكم في الازدحام ببروتوكول TCP. عند هذه النقطة من الطبيعي أن نتساءل لماذا تتصرّف الخوارزمية بشكلٍ مختلف بعد حدث انقضاء فترة موقّت عنه بعد تلقي ثلاثة إشعارات استلام مكرّرة. بالتحديد، ما الذي يجعل مُرسِل TCP يتصرّف بتحفظ أكثر إثر وقوع حدث انقضاء فترة موقّت، فيضع قيمة نافذة الازدحام عند MSS واحدة، بينما عند تلقّي ثلاثة إشعارات استلام مكرّرة يكتفي فقط بتقليل نافذة الازدحام إلى النصف؟ من الغريب أن إصداراً مبكراً من بروتوكول TCP، يعرف ببروتوكول TCP Tahoe، يُحِد من نافذة الازدحام عند MSS واحدة بشكلٍ غير مشروط ويدخل مرحلة البداية البطيئة بعد أيٍّ من نوعي حدث الفقد. أما الإصدار الأحدث TCP Reno فيلغي مرحلة البداية البطيئة بعد تلقي ثلاثة إشعارات استلام مكرّرة. إنّ الفلسفة وراء إلغاء البداية البطيئة في تلك الحالة الأخيرة أنه بالرغم من أن رزمة قد فقدت، إلا أن وصول ثلاثة إشعارات استلام مكرّرة يدل على أن المُستقبِل قد استلم بعض القطع (بالتحديد ثلاث قطع إضافية بعد القطعة المفقودة). وهكذا فبخلاف حالة انقضاء فترة الموقّت، تبدو الشبكة قادرة على توصيل بعض القطع على الأقل، حتى إذا كانت هناك قطع أخرى تُفقد بسبب الازدحام. هذا الإلغاء لمرحلة البداية البطيئة بعد ثلاثة إشعارات استلام مكرّرة يعرف بالتعافي السريع (fast recovery) من أثر الازدحام.

الحالة	الحدث	إجراء التحكم في الازدحام في مُرسِل TCP	ملاحظات
بداية بطيئة (SS)	وصول إشعار استلام بيانات لم يتم الإشعار باستلامها من قبل	CongWin = CongWin + MSS, If (CongWin > Threshold) set state to "Congestion Avoidance" انتقل إلى حالة "تجنب الازدحام"	ينتج عنه مضاعفة CongWin كل *RTT*
تجنب الازدحام (CA)	وصول إشعار استلام بيانات لم يتم الإشعار باستلامها من قبل	CongWin=CongWin+ MSS × (MSS/CongWin)	زيادة خطيّة، ينتج عنها زيادة CongWin بـ MSS واحدة كل *RTT*.
SS أو CA	اكتشاف حدث فقد بإشعارات استلام ثلاثية مكرّرة	Threshold=CongWin/2 CongWin=Threshold انتقل إلى حالة "تجنب الازدحام"	تعافٍ سريع، تفعيل التنقيص الضربي. لن تقل CongWin عن MSS واحدة
SS أو CA	انقضاء فترة الموقّت	Threshold=CongWin/2 CongWin =1 MSS انتقل إلى حالة "البداية البطيئة"	دخول مرحلة البداية البطيئة
SS أو CA	إشعارات استلام مكرّرة	قم بزيادة عدد إشعارات الاستلام المكرّرة للقطعة الجاري الإشعار باستلامها	تبقى كلٌّ من CongWin وThreshold بدون تغيير.

الجدول 3-3 التحكم في الازدحام لدى مُرسِل TCP، على افتراض أن قيمة CongWin الأولية تساوي MSS، والقيمة الأولية للعتبة (threshold) كبيرة (مثلاً 65 كيلوبايت [Stevens 1994])، وأن مُرسِل TCP يبدأ في حالة البداية البطيئة. الحالة المبينة بالجدول هي حالة مُرسِل TCP قبل حصول الحدث مباشرةً. راجع [RFC 2581] لمزيد من التفاصيل.

يبين الشكل 3-53 التغير في قيمة نافذة الازدحام لكلٍ من إصداري Reno وTahoe من بروتوكول TCP، في هذا الشكل كانت القيمة الأولية للعتبة 8×MSS لدورات الإرسال الثماني الأولى يتّخذ كلٌّ من Reno وTahoe إجراءات مماثلة. تزداد قيمة نافذة الازدحام بطريقة أسّية بسرعة أثناء مرحلة البداية البطيئة حتى تلتقي

بالعتبة في الدورة الرابعة للإرسال. بعد ذلك تزداد قيمة نافذة الازدحام بشكلٍ خطي إلى أن يحدث إشعار استلام ثلاثي مكرّر مباشرةً بعد دورة الإرسال رقم 8. لاحظ أن قيمة نافذة الازدحام كانت 12×MSS عند وقوع حدث الفقد هذا. عندئذ تضبط قيمة العتبة عند نصف CongWin أي تصبح 6×MSS. تبعاً لإصدار TCP Reno، تأخذ نافذة الازدحام القيمة 6×MSS وبعد ذلك تنمو بشكلٍ خطي. تبعاً لإصدار TCP Tahoe تأخذ نافذة الازدحام القيمة MSS واحدة وبعد ذلك تنمو بشكلٍ أسّي حتى تصل إلى العتبة. هذه الخوارزمية للتحكم في الازدحام من تأليف Jacobson [Jacobson 1988]، يوجد وصف لعدد من التعديلات على خوارزمية Jacobson الأصلية في [Stevens 1994] وفي [RFC 2561].



الشكل 3-53 نموّ نافذة الازدحام لبروتوكول TCP تبعاً للخوارزميتين Tahoe و Reno.

كما ذكرنا آنفاً تستخدم أكثر تطبيقات بروتوكول TCP الحالية خوارزمية Reno. تم اقتراح العديد من نوعيات خوارزمية Reno [RFC 3782; RFC 2018]. تسعى خوارزمية Vegas المقترحة [Brakmo 1995; Ahn 1995] لتفادي الازدحام مع الحفاظ على طاقة إنتاجية عالية. الفكرة الأساسية وراء خوارزمية Vegas هي: (1) اكتشاف الازدحام في الموجّهات بين المصدر والوجهة النهائية قبل حدوث فقد

للرزم، و(2) تخفيض معدل الإرسال بشكلٍ خطي عند اكتشاف هذا الفقد الوشيك للرزم. يتم توقع الفقد الوشيك للرزم بملاحظة وقت رحلة الذهاب والإياب *RTT*، فكلما كان *RTT* للرزم أطول كان الازدحام في الموجّهات أكبر.

وصف ماكروسكوبي (تقريبي) للطاقة الإنتاجية لبروتوكول TCP

بالنظر إلى نمط سن المنشار الذي يميز سلوك بروتوكول TCP في التحكم في الازدحام، من الطبيعي أن نفكر في حساب الطاقة الإنتاجية المتوسطة (أي معدل الإرسال المتوسط) لتوصيلة TCP طويلة الأمد. في هذا التحليل سنهمل مراحل البداية البطيئة التي تحدث بعد أحداث انقضاء فترة الموقّت. (عادةً ما تكون هذه المراحل قصيرة جداً، حيث تنمو قيمة نافذة الازدحام لدى المُرسِل بطريقة أسّية ويتم تجاوز تلك المرحلة بسرعة). أثناء فترة رحلة ذهاب وإياب معينة، يعتمد المعدل الذي يرسل به بروتوكول TCP البيانات على نافذة الازدحام وقيمة *RTT* الحالية. عندما تكون نافذة الازدحام *w* بايتات ووقت رحلة الذهاب والإياب الحالي *RTT* ثانية، يكون معدل الإرسال تقريباً *w/RTT*. يقوم TCP بعد ذلك باستكشاف وجود حيز تردد إضافي، وذلك بزيادة *w* بـ MSS واحدة كل فترة *RTT* إلى أن يقع حدث فقد. لنرمز لقيمة النافذة عند وقوع حدث الفقد بالرمز *W*. على افتراض أن كلاً من *RTT* و*W* ثابتان تقريباً طوال مدة التوصيلة، فإن معدل إرسال TCP يتراوح من *W*/(2×*RTT*) إلى *W/RTT*.

تؤدي هذه الفرضيات إلى نموذج ماكروسكوبي (تقريبي) مبسط للغاية لسلوك بروتوكول TCP في الحالة الاستقرار (steady state). تُسقط الشبكة رزمة للتوصيلة عندما يزداد معدل الإرسال إلى *W/RTT*، وعندها يقلل المُرسِل معدل إرساله إلى النصف ثم يعاود زيادته بمقدار *MSS/RTT* كل فترة *RTT* إلى أن يصل من جديد إلى *W/RTT*. تكرّر هذه العملية نفسها مراراً وتكراراً. نظراً لأن طاقة TCP الإنتاجية (أي معدل الإرسال) تزيد بشكلٍ خطي بين هاتين القيمتين الطرفيتين، فإننا نحصل على:

$$\text{معدل الإرسال المتوسط للتوصيلة} = \frac{0.75\,W}{RTT}$$

باستعمال هذا النموذج المثالي جداً لديناميكية بروتوكول TCP في حالة الاستقرار (steady state)، يمكننا أن نشتق تعبيراً شائقاً يربط معدل فقد الرزم على التوصيلة بحيز التردد المتاح لها [Mahdavi 1997]، وسيتم تناول ذلك من خلال التمارين. هناك نموذج أكثر تطوّراً تم التوصل إليه بشكلٍ تجريبي ليوافق قياسات البيانات [Padhye 2000].

مُستقبِل بروتوكول TCP

من المهم إدراك أن أساليب التحكم في الازدحام في بروتوكول TCP قد تطورت على مر السنين، وهي تواصل تطورها حالياً. يمكن الاطلاع على ملخص للتحكم في الازدحام في TCP ابتداءً من أواخر التسعينيات في [RFC 2581]. لاستعراض التطوّرات الأخرى هذا المجال راجع [Floyd 2001]. واضحٌ أن ما كان مناسباً للإنترنت عندما كانت توصيلات TCP تنقل في الغالب حركة مرور بيانات SMTP وFTP وTelnet لن تكون بالضرورة مناسبة لإنترنت اليوم التي يغلب عليها حركة مرور بيانات HTTP أو لإنترنت المُستقبل بخدمات لم نحلم بها بعد.

لتوضيح الحاجة المستمرة لتطوير بروتوكول TCP، دعنا نأخذ بعين الاعتبار وصلات TCP السريعة المطلوبة لتطبيقات الحوسبة الشبكية (grid computing) [Foster 2002]. خذ على سبيل المثال توصيلة TCP بقطع بيانات مقاسها 1,500 بايت ووقت رحلة الذهاب والإياب قدره 100 ميللي ثانية. افترض أننا نريد إرسال البيانات عبر هذه التوصيلة بمعدل 10 جيجابت/ثانية. تبعاً لـ [RFC 3649] نلاحظ أنه باستخدام معادلة الطاقة الإنتاجية المذكورة آنفاً، لتحقيق طاقة إنتاجية قدرها 10 جيجابت/ثانية ينبغي استخدام نافذة ازدحام مقاسها 83,333 قطعة في المتوسط. تلك كمية كبيرة من القطع، مما يجعلنا قلقين بعض الشيء بخصوص ما يمكن أن يحدث إذا فقدت إحدى تلك القطع الـ 83,333 أثناء انتقالها. ماذا يحدث في حالة الفقد؟ بمعنى آخر، ما النسبة التي يمكن أن تفقد من القطع المُرسَلة ومع ذلك تسمح لخوارزمية TCP للتحكم في الازدحام والملخّصة في الجدول 3-3 بإنجاز معدل الإرسال 10 جيجابت/ثانية المطلوب؟ في تمارين هذا الفصل سيتم توجيهك عبر

خطوات اشتقاق معادلة لحساب الطاقة الإنتاجية لتوصيلة TCP بدلالة نسبة الفقد L، وقت رحلة الذهاب والإياب RTT، والحجم الأقصى للقطعة MSS:

$$\text{معدل الإرسال المتوسط للتوصيلة} = \frac{1.22\,MSS}{RTT\sqrt{L}}$$

باستعمال هذه المعادلة يمكننا أن نرى أنه لكي نحقق طاقة إنتاجية مقدارها 10 جيجابت/ثانية، يمكن أن تتحمل خوارزمية التحكم في الازدحام احتمال فقد قطعة بحد أقصى 2×10^{-10} (أو مايعادل حدث فقد واحد لكل 5,000,000,000 قطعة ترسل؛ وهي نسبة فقد منخفضة جداً). أدت هذه الملاحظة بعدد من الباحثين للبحث عن إصدارات جديدة من بروتوكول TCP مصممة خصّيصاً لمثل هذه البيئات للنقل السريع للبيانات، يمكنك مراجعة [Jin 2004; RFC 3649; Kelly 2003] لمناقشة تلك الجهود.

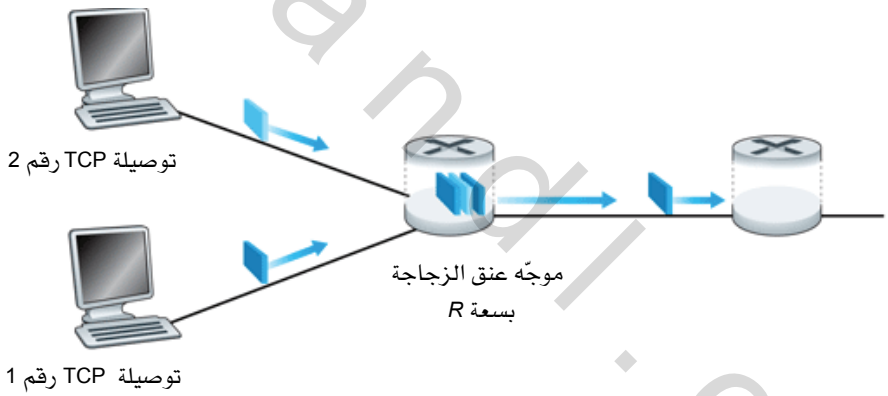
1-7-3 عدالة توزيع سعة الإرسال

خذ في الاعتبار عدد K من توصيلات TCP لكلٍّ منها مسار من طرف إلى طرف، ولكنها تمر جميعاً عبر وصلة عنق الزجاجة بمعدل إرسال قدره R بت/ثانية (نعني بوصلة عنق الزجاجة أن كل الوصلات الأخرى على مسار كل توصيلة ليست مزدحمة وتتوافر لديها سعة إرسال كافية مقارنةً بسعة الإرسال لوصلة عنق الزجاجة). افترض أن كل توصيلة تنقل ملفاً كبيراً وأنه لا توجد حركة بيانات UDP تمر عبر وصلة عنق الزجاجة. يقال: إن آلية التحكم في الازدحام عادلة إذا كان معدل الإرسال المتوسط لكل توصيلة يساوي تقريباً R/K، أي أن كل توصيلة تحصل على نصيب متساوٍ من حيّز التردد للوصلة.

السؤال هو هل خوارزمية AIMD ببروتوكول TCP عادلة، علماً بأن توصيلات TCP المختلفة يمكن أن تبدأ في أوقات مختلفة ومن ثم يمكن أن تكون لها نوافذ ازدحام بقيم مختلفة في نقطة معينة من الزمن؟ يتضمن [Chiu 1989] تفسيراً بديهياً رائعاً يوضح كيف يتقارب أسلوب التحكم في الازدحام ببروتوكول TCP بمرور

الوقت ليوفر حصة متساوية من حيّز التردد لوصلة عنق الزجاجة لكلٍ من توصيلات TCP المتنافسة.

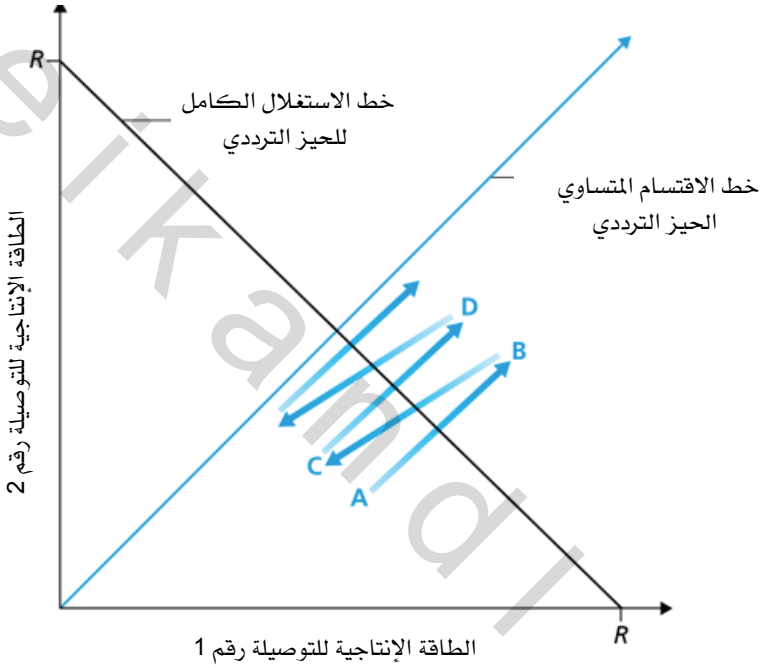
لنأخذ في الاعتبار الحالة البسيطة لتوصيلتي TCP تشتركان في وصلة واحدة لها معدل إرسال مقداره R كما هو مبيّن في الشكل 3-54. افترض أن كلتا التوصيلتين لهما نفس القيم للمتغيرات MSS وRTT (بحيث إنه إذا كان لهما نفس حجم نافذة الازدحام، فسيكون لهما نفس الطاقة الإنتاجية)، وأن لديهما كمية كبيرة من البيانات المطلوب إرسالها، وأنه لا توجد توصيلات TCP أخرى أو وحدات بيانات UDP تعبر تلك الوصلة المشتركة. افترض أيضاً أننا سنهمل مرحلة البداية البطيئة لبروتوكول TCP وأن كل توصيلات TCP تعمل في نمط تجنب الازدحام (AIMD) في جميع الأوقات.



الشكل 3-54 توصيلتا TCP مشتركتان في وصلة عنق زجاجة واحدة.

يوضح المخطط البياني بالشكل 3-55 الطاقة الإنتاجية التي تحصّلها كلٌّ من توصيلتي TCP. لتحقيق مشاركة عادلة بين التوصيلتين في الحيّز الترددي للوصلة المشتركة، ينبغي أن تقع الطاقة الإنتاجية المتحققة على خط ينطلق من نقطة الأصل بميل 45 درجة. مثالياً يجب أن يساوي مجموع الطاقتين الإنتاجيتين للتوصيلتين سعة إرسال الوصلة المشتركة R. (بالتأكيد يُعتبر حصول كل توصيلة على حصة

متساوية تساوي الصفر من سعة الإرسال للوصلة يُعتبر أمراً غير مرغوب فيه!). وعليه يجب أن يكون الهدف هو أن تقع الطاقة الإنتاجية المنجزة في مكان ما بالقرب من تقاطع خط المشاركة المتساوية في الحيّز الترددي (بميل 45 درجة) وخط الاستغلال الكامل للحيّز الترددي في الشكل 3-55.



الشكل 3-55 الطاقة الإنتاجية المتحققة لتوصيلتي TCP رقم 1 و2.

افترض أن مقاسات نوافذ TCP في نقطة معينة من الزمن كانت بحيث تمثّل الطاقة الإنتاجية لكلٍ من التوصيلتين 1 و2 بالنقطة A في الشكل 3-55. نظراً لأن سعة إرسال الوصلة المستخدمة بالتوصيلتين معاً أقل من *R* عند هذه النقطة، فلن يحدث فقد للرزم وستقوم كلٌّ من التوصيلتين بزيادة مقاس نافذتها بمعدل *MSS* واحدة لكل فترة *RTT* حسب خوارزمية TCP لتجنب الازدحام. وهكذا تسير الطاقة الإنتاجية المشتركة للتوصيلتين على خط بميل 45 درجة (زيادة متساوية لكلتا التوصيلتين) بدءاً من النقطة A. في النهاية ستتجاوز سعة إرسال الوصلة المستخدمة

بالتوصيلتين معاً القيمة R، ويبدأ وقوع أحداث فقد للرزم. افترض أن التوصيلتين 1 و2 تعانيان من فقد رزم عندما تكون الطاقة الإنتاجية لهما ممثلة بالنقطة B. عندئذ ستقرر التوصيلتان 1 و2 إنقاص نوافذهما بمقدار النصف، ومن ثم تصبح الطاقات الإنتاجية الناتجة ممثلة بالنقطة C في منتصف المسافة على الخط الواصل من B إلى نقطة الأصل. نظراً لأن سعة إرسال الوصلة المستخدمة بالتوصيلتين معاً أقل من R عند النقطة C، تزيد التوصيلتان من طاقتيهما الإنتاجية مرة أخرى على طول خط بميل 45 درجة يبدأ من C. في النهاية سيحدث فقد للرزم من جديد، على سبيل المثال عند النقطة D، وعندئذٍ تُنقص التوصيلتان نوافذهما ثانيةً بمقدار النصف. وهكذا عليك إقناع نفسك بأن معدل الإرسال المتحقق للتوصيلتين سيتأرجح في النهاية على طول خط معدل الإرسال المتساوي. كما ينبغي أن تقنع نفسك أيضاً بأن التوصيلتين ستصلان تقاربياً لهذا السلوك بغض النظر عن موقعهما في البداية في الفضاء ثنائي الأبعاد! رغم أن هذا السيناريو يعتمد على عددٍ من الفرضيات المثالية، إلا أنه يعطينا إحساساً بديهياً للسبب وراء توفير بروتوكول TCP مشاركة متساوية في حيّز التردد للوصلة المشتركة بين التوصيلات التي تستخدمها.

في السيناريو المثالي الذي نحن بصدده افترضنا أن توصيلات TCP فقط هي التي تعبر وصلة عنق الزجاجة، وأن تلك التوصيلات لها نفس قيمة RTT، وأن هناك توصيلة TCP واحدة مرتبطة بكل زوج من مضيفات المصدر والوجهة النهائية. في الواقع العملي لا تتحقق تلك الشروط عادةً، ومن ثم يمكن أن تحصل تطبيقات الزبون/الخادم على أنصبة غير متساوية أبداً من الحيّز الترددي للوصلة المشتركة. بشكلٍ خاص تم إثبات أنه عند اشتراك عدة توصيلات في وصلة عنق الزجاجة فإن الجلسات التي لها قيم RTT أصغر تتمكن من الاستحواذ على الحيّز الترددي على تلك الوصلة بسرعة أكبر عند توفره (أي تقوم بفتح نوافذ الازدحام الخاصة بها بشكلٍ أسرع) ومن ثم تتمتع بطاقة إنتاجية أعلى من تلك التوصيلات التي لها قيم RTT أكبر [Lakshman 1997].

بروتوكول UDP وعدالة توزيع سعة الإرسال

رأينا الآن كيف أن التحكم في الازدحام في بروتوكول TCP ينظم معدل الإرسال من التطبيقات عن طريق آلية نافذة الازدحام. لا يستخدم العديد من تطبيقات الوسائط المتعددة، كهاتف الإنترنت والمؤتمرات عبر الفيديو بروتوكول TCP في أغلب الأحيان لهذا السبب بعينه – فهي لا تقبل بخنق معدل الإرسال لها حتى إذا كانت الشبكة مزدحمة جداً. بدلاً من ذلك تفضّل تلك التطبيقات التشغيل على بروتوكول UDP، والذي لا يتضمن سيطرة مبيّتة على الازدحام. عند تشغيلها على UDP يمكن للتطبيقات ضخ بيانات الصوت والفيديو إلى الشبكة بمعدل إرسال ثابت وتفقد الرزم من حين لآخر، وهي تفضل ذلك على تقليل معدلات إرسالها إلى مستويات "عادلة" في أوقات الازدحام مع عدم فقد أي رزم. من منظور بروتوكول TCP، لا تعتبر تطبيقات الوسائط المتعددة التي يتم تشغيلها على UDP منصفة – فهي لا تتعاون مع التوصيلات الأخرى ولا تتحكم في معدلات إرسالها بشكلٍ ملائم. نظراً لأن التحكم في الازدحام في بروتوكول TCP يقلل من معدل الإرسال لمواجهة الازدحام المتزايد (فقد الرزم)، بينما لا تحتاج مصادر UDP لذلك، فمن المحتمل أن تزاحم مصادر UDP حركة مرور TCP وتضيّق المجال عليها. وعليه فمن مجالات البحث المهمة اليوم تطوير آليات للتحكم في الازدحام بالإنترنت تمنع حركة مرور UDP من جعل طاقة الإنترنت الإنتاجية تتدهور لتصل إلى توقّف كامل [Floyd 1999; Floyd 2000; Kohler 2006].

عدالة توزيع سعة الإرسال وتوصيلات TCP المتوازية

لكن حتى لو أمكننا إجبار حركة مرور UDP على التصرّف بإنصاف، فإن مشكلة عدالة التوزيع لن تكون قد حُلّت تماماً. يكمن السبب في ذلك في أنه ليس هناك ما يمنع التطبيقات المبنية على بروتوكول TCP من استخدام عدة توصيلات على التوازي. على سبيل المثال غالباً ما تستخدم متصفحات الويب عدة توصيلات TCP متوازية لنقل بيانات عدة كائنات ضمن صفحة الويب في نفس الوقت (العدد الدقيق للتوصيلات المتعدّدة هو متغير قابل للضبط في أكثر المتصفّحات). عندما

يستخدم تطبيقٌ ما عدة توصيلات متوازية فإنه يستحوذ على جزء أكبر من حيّز التردد لوصلة مزدحمة. كمثال خذ في الاعتبار وصلة بسعة إرسال R تدعم تسعة تطبيقات زبون/خادم مستمرة، بحيث يستخدم كل تطبيق توصيلة TCP واحدة. إذا ظهر تطبيق جديد يستخدم توصيلة TCP واحدة كذلك، فإن كل تطبيق سيستخدم تقريباً نفس قيمة معدل الإرسال والتي مقدارها $R/10$. أما إذا كان ذلك التطبيق الجديد يستخدم بدلاً من ذلك 11 توصيلة TCP متوازية، فإن التطبيق الجديد سيستحوذ على نصيب غير عادل يزيد على $R/2$. نظراً لأن حركة مرور الويب واسعة الانتشار جداً على الإنترنت، فإن توصيلات TCP المتعدّدة على التوازي تعد أمراً شائعاً.

8-3 الخلاصة

بدأنا هذا الفصل بدراسة الخدمات التي يوفرها بروتوكول طبقة النقل لتطبيقات الشبكة. فمن ناحية قد يكون بروتوكول طبقة النقل بسيطاً للغاية بحيث يوفر خدمة بلا رتوش للتطبيقات التي تستخدمه، فيقوم فقط بوظيفة التجميع والتوزيع للعمليات التي تتصل فيما بينها. ومثال ذلك بروتوكول الإنترنت UDP. وعلى الطرف الآخر يمكن أن يوفر بروتوكول طبقة النقل تشكيلة من الضمانات للتطبيقات، كالتوصيل الموثوق للبيانات، وضمانات التأخير، وضمانات الحيّز الترددي. ومع ذلك فغالباً ما يُحِد نموذج الخدمة لبروتوكول طبقة الشبكة التحتي من الخدمات التي يمكن أن يوفرها بروتوكول طبقة النقل. فمثلاً إذا كان بروتوكول طبقة الشبكة لا يستطيع توفير ضمانات الحيّز الترددي أو التأخير لقطع طبقة النقل، فإن بروتوكول طبقة النقل لن يستطيع توفير ضمانات الحيّز الترددي أو التأخير لقطع البيانات التي تُنقل بين العمليات.

عرفنا في الجزء 3-4 أن بروتوكول طبقة النقل يمكن أن يوفر نقلاً موثوقاً للبيانات حتى لو كانت طبقة الشبكة التحتية غير موثوقة. ورأينا أن توفير نقلٍ موثوقٍ للبيانات يتضمن العديد من النقاط الدقيقة، لكن هذه المهمة يمكن أن

تنجز من خلال الجمع بعناية ما بين إشعارات الاستلام، والموقّتات، وإعادة الإرسال، والأرقام التسلسلية.

رغم أننا غطّينا النقل الموثوق للبيانات في هذا الفصل، يجب ألا يغيب عن بالنا أن النقل الموثوق للبيانات يمكن أن توفره بروتوكولات طبقة ربط البيانات أو طبقة الشبكة أو طبقة النقل أو حتى طبقة التطبيقات. يمكن لأيٍ من الطبقات العليا الأربعة في رصة البروتوكولات أن تستخدم إشعارات الاستلام، والموقّتات، وإعادة الإرسال، والأرقام التسلسلية، لتوفر نقلاً موثوقاً للبيانات للطبقة التي تعلوها. في الحقيقة وعلى مر السنين صمّم علماء ومهندسو الحاسب وطوروا بشكلٍ مستقل بروتوكولات للنقل الموثوق للبيانات لطبقة ربط البيانات، وطبقة الشبكة، وطبقة النقل، وطبقة التطبيقات (رغم أن العديد من تلك البروتوكولات اختفت بهدوء من الساحة).

في الجزء 3-5 ألقينا نظرة فاحصة على بروتوكول TCP، بروتوكول الإنترنت التوصيلي للنقل الموثوق للبيانات. عرفنا أن بروتوكول TCP نظام معقّد، يتضمّن إدارة التوصيلات، وضبط التدفق، وتقدير وقت رحلة الذهاب والإياب، بالإضافة إلى النقل الموثوق للبيانات. ومع ذلك يُعتبر بروتوكول TCP في الواقع أكثر تعقيداً من الوصف الذي قدمناه. لقد أغفلنا عمداً مناقشة تشكيلة من الترقيعات والإصلاحات والتحسينات التي شاع استخدامها على نطاق واسع في إصدارات بروتوكول TCP المختلفة. ومع ذلك فإن كل هذا التعقيد يتم حجبه عن التطبيق الذي يتم تشغيله على الشبكة. إذا أراد زبون على مضيف إرسال البيانات بشكلٍ موثوق إلى خادم على مضيف آخر، فإنه يفتح ببساطة مقبس TCP على الخادم ويضخ البيانات عبر ذلك المقبس. إن تطبيق الزبون/الخادم محظوظ لأنه لا يحمل هم أيٍ من التعقيد الذي ينطوي عليه بروتوكول TCP.

استعرضنا في الجزء 3-6 موضوع التحكم في الازدحام من منظور واسع، وفي الجزء 3-7 شرحنا كيف يطبّق بروتوكول TCP التحكم في الازدحام، وعرفنا أن التحكم في الازدحام ضروري لصحة الشبكة. بدون تحكم في الازدحام يمكن

بسهولة أن تصبح الشبكة مقفولة، بحيث يتم نقل القليل من البيانات أو لا يتم نقل أي بيانات على الإطلاق من طرف إلى طرف. وفي الجزء 3-7 عرفنا أيضاً أن بروتوكول TCP يطبّق آلية تحكم في الازدحام من طرف إلى طرف، حيث يتم من خلالها زيادة معدل الإرسال بشكلٍ خطي عند توقع كون مسار الإرسال خالياً من الازدحام، وتقليل معدل الإرسال بشكلٍ أسّي عند حدوث فقد في الرزم. كما تسعى تلك الآلية أيضاً لإعطاء كل توصيلة TCP تمر عبر وصلة مزدحمة حصةً متساويةً من حيّز التردد للوصلة. وتناولنا أيضاً بشيءٍ من التفصيل تأثير إنشاء توصيلات TCP والبداية البطيئة على التأخير، فلاحظنا أنه في العديد من السيناريوهات المهمة، يسهم إنشاء التوصيلة ومرحلة البداية البطيئة بشكلٍ ملحوظ في التأخير من طرف إلى طرف. نؤكّد مرة أخرى هنا على أنه بينما تطوّرت أساليب التحكم في الازدحام على مر السنين، فإنها تبقى مجالاً حيوياً للبحث ومن المحتمل أن يتواصل ذلك التطور خلال السنوات القادمة.

تركزت مناقشتنا للبروتوكولات المحددة لنقل البيانات على الإنترنت في هذا الفصل على بروتوكولي UDP وTCP – "خيول العمل" في طبقة النقل على الإنترنت. ومع ذلك فقد اتضح من خلال عقدين من التجربة مع هذين البروتوكولين الظروف التي لا يصلح فيها أيٌّ منهما بشكلٍ مثالي. ولذا فقد انكب الباحثون على تطوير بروتوكولات إضافية لطبقة النقل، أصبح عددٌ منها الآن معايير مقترحة لدى فريق عمل هندسة الإنترنت (IETF) نذكر منها ما يلي:

- بروتوكول التحكم في الازدحام لوحدة البيانات (DCCP) [RFC 4340]، والذي يوفر خدمةً للنقل غير الموثوق أساسها رسالة البيانات وبأعباء إضافية أقل. تشبه تلك الخدمة خدمة UDP ولكن بتحكم في الازدحام يختاره التطبيق ومتوائم مع بروتوكول TCP. إذا احتاج تطبيق ما نقلاً موثوقاً أو شبه موثوق للبيانات، يتم تحقيق ذلك ضمن التطبيق نفسه، ربما باستخدام الآليات التي درسناها في الجزء 3-4. يُتوقع استخدام بروتوكول DCCP بواسطة تطبيقات مثل العرض المستمر لمواد الوسائط المتعددة (انظر الفصل

السابع) التي يمكنها الاستفادة من الموازنة ما بين الوصول في الوقت المناسب والموثوقية في توصيل البيانات، ولكنها بحاجة أيضاً للتجاوب مع العواقب الوخيمة لازدحام الشبكة.

- بروتوكول النقل بالتحكم في مسارات البيانات (SCTP) [RFC 2960; RFC 3286]، وهو بروتوكول للنقل الموثوق أساسه رسالة البيانات ويسمح بتجميع أكثر من مسارٍ بيانات على مستوى التطبيقات عبر توصيلة SCTP واحدة (أسلوب يعرف باسم المسارات المتعددة). من وجهة نظر الموثوقية تعالَج المسارات المختلفة ضمن التوصيلة بشكلٍ منفصل كي لا يؤثر فقد الرزم في مسارٍ على توصيل البيانات في المسارات الأخرى. يسمح بروتوكول SCTP أيضاً بنقل البيانات على طريقين خارجين عندما يكون المضيف موصلاً بشبكتين أو أكثر، وبتوصيل اختياري للبيانات التي تصل بغير الترتيب السليم، بالإضافة إلى عدد من السمات الأخرى. يستخدم بروتوكول SCTP تقريباً نفس خوارزميات ضبط التدفق والتحكم في الازدحام التي يستخدمها بروتوكول TCP.
- بروتوكول التحكم في معدل البيانات المتوائم مع TCP (TFRC)، وهو بروتوكول للسيطرة على الازدحام وليس بروتوكولاً كاملاً لطبقة النقل [TFRC 2448]. يحدّد البروتوكول آليةً للسيطرة على الازدحام يمكن أن تستخدم في بروتوكول نقل آخر مثل DCCP (في الحقيقة فإن بروتوكول TFRC هو أحد البروتوكولين اللذين يمكن اختيارهما بواسطة التطبيق والمتوفرين في DCCP). إن الهدف من TFRC هو تنعيم نمط سن المنشار لسلوك التحكم في الازدحام في بروتوكول TCP (انظر الشكل 3-51)، وفي الوقت نفسه الحفاظ على معدل إرسال على المدى البعيد يقارب إلى حدٍ معقول معدل إرسال TCP. بمعدل إرسال أكثر سلاسة مما عليه الحال في TCP، يكون بروتوكول TFRC ملائماً لتطبيقات الوسائط المتعددة مثل

هاتف الإنترنت والنقل المستمر لمواد الوسائط المتعددة، حيث يكون من المهم استخدام معدل إرسال سَلِس. وجدير بالذكر أن TFRC هو بروتوكول مبني على معادلة، حيث تُستخدم النسبة المقاسة لفقد الرزم كمتغير في المعادلة [Padhye 2000] التي تقوم بتقدير الطاقة الإنتاجية لبروتوكول TCP إذا ما كانت جلسة TCP تعاني من فقد الرزم بتلك النسبة. يؤخذ هذا المعدل عندئذٍ كمعدل الإرسال المستهدف لبروتوكول TFRC.

ستُخبر الأيام عمّا إذا كانت البروتوكولات DCCP أو SCTP أو TFRC سترى انتشاراً على نطاق واسع. رغم أن تلك البروتوكولات توفر إمكانيات أفضل بشكلٍ واضح مقارنةً ببروتوكولي TCP وUDP، فإن هذين الأخيرين من ناحية أخرى قد أثبتا أنهما "جيدان بما فيه الكفاية" على مرّ السنين. هل سيفوز "الأفضل" على "الجيد بما فيه الكفاية"؟ سوف يعتمد ذلك على تركيبة معقدة من الاعتبارات التقنية والاجتماعية والتجارية.

ذكرنا في الفصل الأول أن شبكة الحاسب يمكن تقسيمها إلى "حافة الشبكة" و" قلب الشبكة". تغطي حافة الشبكة كل شيء يحدث في الأنظمة الطرفية. الآن وبعد أن غطينا طبقة التطبيقات وطبقة النقل نكون قد انتهينا من مناقشتنا لحافة الشبكة بالكامل. لقد حان الوقت الآن لاستكشاف قلب الشبكة! ستبدأ هذه الرحلة في الفصل القادم حيث سندرس طبقة الشبكة، وتستمر إلى الفصل الخامس حيث سندرس طبقة ربط البيانات.

أسئلة وتمارين وتدريبات الفصل الثالث

❖ أسـئلة مراجعة

- **الأجزاء 3-1 إلى 3-3**

1. افترض أن طبقة الشبكة توفر الخدمة التالية: تقبل طبقة الشبكة من المضيف المصدر قطعة بيانات بطول أقصى قدره 1200 بايت وعنوان وجهة من طبقة النقل. بعد ذلك تضمن طبقة الشبكة تسليم القطعة إلى طبقة النقل على مضيف الوجهة. افترض أن مضيف الوجهة يمكن أن يدعم العديد من عمليات تطبيقات الشبكة والتي يتم تنفيذها عليه في نفس الوقت.
   a. صمم أبسط بروتوكول ممكن لطبقة النقل والذي يقوم بنقل بيانات التطبيق إلى العملية على مضيف الوجهة. افترض أن نظام التشغيل على مضيف الوجهة قد خصّص رقم منفذ من 4 بايتات لكل عملية تطبيقات يجري تنفيذها على المضيف.
   b. قم بتعديل هذا البروتوكول بحيث يوفر "عنوان عودة" إلى عملية الوجهة.
   c. في بروتوكولاتك أعلاه، هل على طبقة التطبيقات أن تعمل شيئاً في قلب شبكة الحاسب؟
2. تخيّل كوكباً ينتمي كل شخص عليه لعائلة من 6 أفراد، وكل عائلة تعيش في بيتها الخاص بها، وكل بيت له عنوانه الفريد الخاص به. افترض أن هذا الكوكب تتوافر به خدمة بريدية لنقل البريد من منزل المصدر إلى منزل الوجهة. تتطلب خدمة البريد أن: (1) يكون الخطاب في ظرف، و(2) يكون عنوان منزل الوجهة (ولاشيء غيره) مكتوباً بوضوح على الظرف. افترض أن كل عائلة انتدبت أحد أفرادها لجمع وتوزيع البريد لأفراد العائلة الآخرين. لا تبيّن الخطابات بالضرورة أي دلالة على الشخص الموجّه له الخطاب.
   a. باستخدام الحل للتمرين 1 أعلاه، صف بروتوكولاً يمكن لمندوبي العائلات استخدامه لتسليم البريد من فرد مرسل في عائلة إلى فرد مستلم في عائلة.
   b. في بروتوكولك أعلاه، هل يحدث أبداً أن تضطر خدمة البريد لفتح ظرف وفض خطاب لكي تتمكن من تقديم خدمتها؟

3. خذ في الاعتبار توصيلة TCP بين المضيف A والمضيف B. افترض أن قطع TCP لها رقم منفذ المصدر x ورقم منفذ الوجهة y. ما رقم منفذ المصدر ومنفذ الوجهة للقطع التي تنتقل من المضيف B إلى المضيف A؟
4. اذكر لماذا قد يختار مطور تطبيق تشغيل تطبيقه على بروتوكول UDP بدلاً من بروتوكول TCP.
5. لماذا ترسل حركة بيانات الصوت والفيديو عادةً على بروتوكول TCP بدلاً من بروتوكول UDP في إنترنت اليوم؟ (ملاحظة: الإجابة التي نتوقعها ليس لها أي علاقة بآلية بروتوكول TCP للسيطرة على الازدحام).
6. هل يمكن لتطبيقٍ ما الاستمتاع بخدمة نقل موثوقة للبيانات رغم أن التطبيق يعمل بالفعل على بروتوكول UDP؟ إذا كان الأمر كذلك، فكيف؟
7. افترض أن عملية على المضيف ج لها مقبس UDP برقم المنفذ 6789. افترض أن كلاً من المضيفين A وB يرسل قطعة UDP إلى المضيف C برقم منفذ الوجهة يساوي 6789. هل سيتم توجيه كلتا القطعتين إلى نفس المقبس على المضيف C؟ إذا كان الأمر كذلك، كيف ستستطيع العملية على المضيف ج أن تعرف أن القطعتين نشأتا على مضيفين مختلفين؟
8. افترض أن خادم ويب يعمل على المضيف C على منفذ 80. افترض أن هذا المضيف يستخدم توصيلات دائمة، ويتلقى حالياً طلبات من مضيفين مختلفين، A وB. هل يتم إرسال كل الطلبات عبر نفس المقبس على المضيف C؟ إذا كانت الطلبات تمر عبر مقبسين مختلفين، هل كلا المقبسين لهما المنفذ 80؟ ناقش واشرح.

- **الجزء 3-4**

9. في بروتوكولات rdt التي طوّرناها، لماذا احتجنا إلى استخدام الأرقام التسلسلية؟
10. في بروتوكولات rdt التي طوّرناها، لماذا احتجنا إلى استخدام موقّتات؟
11. افترض أن زمن تأخير رحلة الذهاب والإياب بين المرسِل والمستقبِل ثابت ومعروف للمرسل. هل سيكون من الضروري في بروتوكول rdt 3.0 استخدام موقّت، على افتراض أن الرزم يمكن أن تضيع؟ اشرح.
12. قم بزيارة برنامج جافا التفاعلي الخاص ببروتوكول العودة N للوراء على موقع الويب المصاحب لهذا الكتاب.

a. اجعل المصدر يرسل خمس رزم، ثم أوقف الرسوم التوضيحية المتحركة قبل وصول أي من تلك الرزم الخمس إلى الوجهة. بعد ذلك قم بإفناء أول رزمة ثم استأنف تشغيل الرسوم التوضيحية المتحركة. صف ما يحدث.

b. كرّر التجربة، ولكن الآن مع السماح بتدفق الرزمة الأولى إلى الوجهة وإفناء أول اشعار استلام. صف ما يحدث.

c. وأخيراً، حاول إرسال ست رزم. ماذا يحدث؟

13. كرّر تمرين 12، ولكن مع برنامج جافا التفاعلي الخاص ببروتوكول "الإعادة الانتقائية". ما هو الاختلاف بين بروتوكولي "ارجع N للوراء" و "الإعادة الانتقائية"؟

- **الجزء 3-5**

14. صح أم خطأ؟:

a. يقوم المضيف A بإرسال ملف كبير إلى المضيف B على توصيلة TCP. افترض أن المضيف B ليس لديه بيانات يريد إرسالها إلى المضيف A. لن يرسل المضيف B إشعارات استلام إلى المضيف A لأنه (أي المضيف B) لا يكون بوسعه تركيب إشعارات الاستلام على ظهر البيانات.

b. لا يتغير حجم نافذة الاستقبال RcvWindow في بروتوكول TCP أبداً طوال فترة التوصيلة.

c. افترض أن المضيف A يرسل إلى المضيف B ملفاً كبيراً على توصيلة TCP. لا يمكن أن يتجاوز عدد البايتات التي يرسلها A ولم يتم الإشعار باستلامها حجم مخزن الاستقبال المؤقت على B.

d. افترض أن المضيف A يرسل إلى المضيف B ملفاً كبيراً على توصيلة TCP. إذا كان الرقم التسلسلي لقطعة بيانات على هذه التوصيلة هو m، فإن الرقم التسلسلي لقطعة البيانات التالية هو بالضرورة $m+1$.

e. تتضمن قطعة بيانات TCP حقلاً في ترويستها يحتوي على حجم نافذة الاستقبال RcvWindow.

f. افترض أن قيمة آخر عينة SampleRTT على توصيلة TCP هي 1 ثانية. ستكون القيمة الحالية لفترة انقضاء الموقت TimeoutInterval بالضرورة أكبر من ثانية.

g. افترض أن المضيف A يرسل إلى المضيف B على توصيلة TCP قطعة بيانات تحمل 4 بايتات من البيانات ولها الرقم التسلسلي 38. في هذه القطعة يكون رقم إشعار الاستلام هو بالضرورة 42.

15. افترض أن المضيف A يرسل إلى المضيف B على توصيلة TCP قطعتي TCP الواحدة تلو الأخرى مباشرة. تحمل القطعة الأولى الرقم التسلسلي 90 بينما تحمل الثانية الرقم التسلسلي 110.
   a. كم بايت من البيانات تحمل القطعة الأولى؟
   b. افترض أن القطعة الأولى تُفقد في الطريق ولكن القطعة الثانية تصل إلى المضيف B. ماذا سيكون رقم إشعار الاستلام الذي يرسله المضيف B إلى المضيف A

16. خذ في الاعتبار مثال بروتوكول الوصول للحاسبات عن بعد (Telnet) الذي أوردناه في الجزء 3-5. بعد بضع ثوان من إدخال المستخدم للحرف 'C' يقوم بإدخال الحرف 'R'. بعد إدخال الحرف 'R'، كم عدد قطع البيانات التي يتم إرسالها؟ وما هي محتويات حقلي الرقم التسلسلي ورقم إشعار الاستلام على كل قطعة؟

- **الجزء 3-7**

17. افترض أن توصيلتي TCP موجودتان على وصلة عنق زجاجة لها معدل إرسال *R* بت/ثانية. لدى كلٍّ من التوصيلتين ملف ضخم مطلوب إرساله (في نفس الاتجاه على وصلة عنق الزجاجة). يبدأ إرسال الملفين في نفس الوقت. ما هو معدل الإرسال الذي يود بروتوكول TCP أن يعطيه لكلٍ من التوصيلتين؟

18. صح أم خطأ؟: خذ في الاعتبار السيطرة على الازدحام في بروتوكول TCP. عند انقضاء فترة الموقّت عند المرسل، يتم ضبط قيمة العتبة (Threshold) عند نصف قيمتها السابقة.

❖ تدريبات

1. افترض أن الزبون A يبدأ جلسة Telnet مع الخادم S، وفي الوقت نفسه تقريباً يُنشىء الزبون B جلسة Telnet مع نفس الخادم S. أوجد أرقام منافذ ممكنة للمصدر والوجهة لكلٍّ من:
   a. قطع البيانات المرسلة من A إلى S.
   b. قطع البيانات المرسلة من B إلى S.

c. قطع البيانات المرسلة من S إلى A.
d. قطع البيانات المرسلة من S إلى B.
e. إذا كان A وB مضيفين مستقلين، هل يمكن أن يكون رقم منفذ المصدر في القطع من A إلى S هو نفسه من B إلى S؟
f. ماذا لو كانا نفس المضيف؟

2. خذ في الاعتبار الشكل 3-5. ما قيمة منافذ المصدر والوجهة في قطع البيانات التي تتدفق من الخادم إلى عمليات الزبائن؟ ما هي عناوين IP في وحدات بيانات طبقة الشبكة التي تحمل قطع طبقة النقل؟

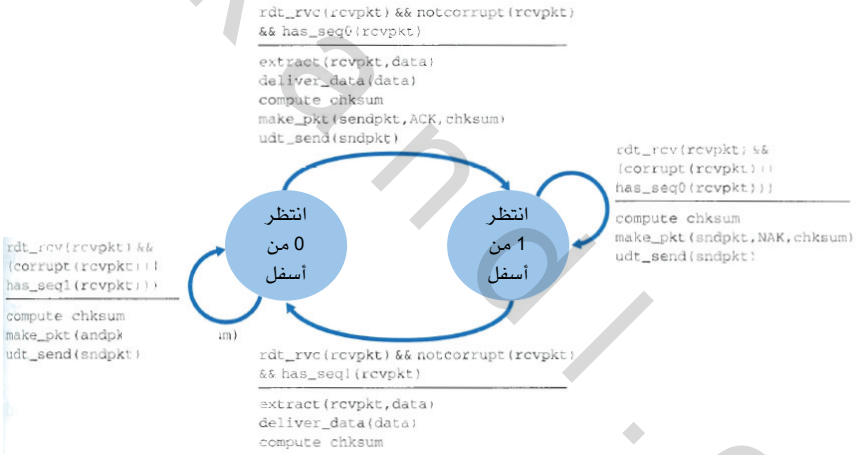
3. يستخدم كلٌّ من بروتوكولي برنامج UDP وTCP مكمّل الواحد لنظام المجموع التدقيقي. لنفترض أن لديك البايتات الثلاثة التالية: 01010101، 01110000، 01001100. ما هو مكمّل الواحد لمجموع تلك البايتات؟ (لاحظ أنه رغم أن UDP وTCP يستخدمان في الواقع كلمات (words) تتكون كلٌّ منها من 16 بتاً لحساب المجموع التدقيقي، فالمطلوب منك في هذا التمرين الحصول على المجموع لبايتات يضم كلٌّ منها 8 بتات فقط. وضّح كل الخطوات. لماذا يأخذ UDP مكمّل الواحد للمجموع، أي لماذا لا يأخذ المجموع نفسه وحسب؟ في نظام مكمّل الواحد، كيف يمكن للمستقبل اكتشاف الأخطاء؟ هل من الممكن مرور خطأ في بت واحد دون أن يُكتشف؟ ماذا عن خطأ في بتين؟

4. a. افترض أن لديك البايتين التاليين: 00110100 و01101001. ما هو مكمّل الواحد لمجموع هذين البايتين؟
b. افترض أن لديك البايتين التاليين: 11110101 و00101001. ما هو مكمّل الواحد لمجموع هذين البايتين؟
c. للبايتين في الجزء (a)، اعط مثالا لبت واحد لو انقلب (من 0 إلى 1 أو من 1 إلى 0) لما تغير مكمّل الواحد لمجموع البايتين.

5. افترض أن مستقبل UDP يحسب مجموع الإنترنت التدقيقي لقطعة UDP التي يستقبلها ويجد أنه مطابق للقيمة الموجودة في حقل المجموع التدقيقي في القطعة. هل يمكن للمستقبل أن يكون متأكداً تماماً بدون أدنى شك أنه لا توجد بتات خطأ في القطعة. اشرح إجابتك.

6. خذ في الاعتبار دوافعنا لتصحيح بروتوكول rdt2.1. وضّح أن المستقبل، والذي يظهر في الشكل على الصفحة التالية، عندما يعمل مع المرسل المبين في الشكل 3-11، فإنه يمكن أن يؤدي بالمرسل والمستقبل للدخول في حالة جمود مستمر، حيث ينتظر كلٌّ منهما حدثاً لن يقع أبداً.

7. في بروتوكول rdt3.0 ، لا تحمل إشعارات الاستلام التي تتدفق من المستقبل إلى المرسل أرقاماً تسلسلية (رغم أنها تحمل الرقم التسلسلي للرزم التي تُشعر باستلامها). لماذا لا تحتاج إشعارات الاستلام التي نرسلها إلى أرقامٍ تسلسلية؟
8. ارسم آلة الأوضاع المحدودة (FSM) على جانب المستقبل من بروتوكول rdt3.0.
9. اعط تعقباً لتسلسل الأحداث لتشغيل بروتوكول rdt3.0 عندما تكون كلٌّ من رزم البيانات ورزم إشعار الاستلام عرضةً للأخطاء. ينبغي أن تشبه إجابتك تلك المستخدمة في الشكل 3-16.
10. خذ في الاعتبار قناة يمكن أن تفقد رزماً ولكن لها تأخير له حد أقصى معروف. قم بتعديل بروتوكول rdt2.1 ليشمل انقضاء وقت الموقّت وإعادة الإرسال على المرسل. وضّح كيف يمكن لبروتوكولك الاتصال بشكلٍ صحيح عبر تلك القناة.



11. يتجاهل جانب المرسل من بروتوكول rdt3.0 ببساطة (أي لا يتخذ أي إجراء بشأن) كل الرزم التي يتم استلامها وبها خطأ أو فيها خطأ في حقل رقم الإشعار ack-num في رزمة إشعار بالاستلام. افترض أنه في مثل هذه الظروف، يقوم rdt3.0 ببساطة بإعادة إرسال رزمة البيانات الحالية. فهل يظل البروتوكول يعمل؟ (ملاحظة: خذ في الاعتبار ما يمكن أن يحدث إذا لم يكن هناك سوى أخطاء في البتات فقط؛ ولا يوجد أي فقد في الرزم، ولكن يمكن حدوث انتهاء لفترة الموقت قبل الأوان. خذ في الاعتبار عدد المرات التي ستُرسل فيها الرزمة رقم n في نهاية الأمر عندما تقارب n من اللانهاية.

12. خذ في الاعتبار بروتوكول rdt3.0. وضح برسم بياني أنه إذا كان بوسع توصيلة الشبكة بين المرسِل والمستقبِل إعادة ترتيب الرسائل (أي إذا كان يمكنها تبديل ترتيب رسالتين من الرسائل التي تنتقل على الوسط مابين بين المرسِل والمستقبِل)، فإن بروتوكول البت المتناوب لن يعمل بشكلٍ صحيح (تأكد من تحديد كيف أنه لن يعمل بشكلٍ صحيح). في الرسم ينبغي أن يكون المرسِل على اليسار والمستقبِل على اليمين، ومحور الوقت يمتد من أعلى الصفحة إلى أسفلها، مع بيان تبادل البيانات (D) وإشعارات الاستلام (A). تأكد من توضيح الرقم التسلسلي المرتبط بكل رزمة بيانات أو إشعار استلام.
13. خذ في الاعتبار بروتوكولاً للنقل الموثوق للبيانات يستخدم فقط إشعارات استلام سلبية. افترض أن المرسِل يرسل بيانات فقط على فترات متباعدة. هل يكون استخدام بروتوكول يرسل إشعارات استلام سلبية أفضل من استخدام بروتوكول يرسل إشعارات استلام إيجابية؟ لماذا؟ افترض الآن أن المرسِل لديه الكثير من البيانات لإرسالها وأن التوصيلة من طرف إلى طرف تعاني من فقد قليل في البيانات. في تلك الحالة الثانية، هل يكون استخدام بروتوكول يرسل إشعارات استلام سلبية أفضل من استخدام بروتوكول يرسل إشعارات استلام إيجابية؟ لماذا؟
14. خذ في الاعتبار مثالاً لاتصال يمر عبر بلد كبير كالمبيّن في الشكل 3-17. ماذا ينبغي أن يكون حجم النافذة ليصبح مدى استغلال الشبكة أكثر من 90٪.
15. في بروتوكول الإعادة الانتقائية (SR) العام الذي درسناه في الجزء 3-4-4، يرسل المرسل الرسالة بمجرد توفرها (إذا كانت ضمن النافذة) وبدون انتظار وصول إشعار استلام. لنفترض الآن أننا نريد بروتوكول إعادة انتقائية يرسل الرسائل اثنتين في كل مرة. بمعنى أن المرسل يرسل زوجاً من الرسائل، وسوف يرسل الزوج التالي من الرسائل فقط عندما يعلم أن كلتا الرسالتين المرسلتين ضمن الزوج الأول قد وصلتا بشكلٍ صحيح. لنفترض أن القناة يمكن أن تفقد الرسائل ولكنها لا تفسدها ولا تعيد ترتيبها. قم بتصميم بروتوكول للتحكم في الخطأ أثناء النقل الموثوق للرسائل باتجاه واحد. بيّن آلة أوضاع محدودة (FSM) تعطي وصفاً للمرسل والمستقبِل. صِف صيغة الرزم المُرسَلة بين المرسل والمستقبِل، والعكس بالعكس. إذا استخدمت أياً من الإجراءات غير تلك المذكورة في الجزء 3-4 (على سبيل المثال udt_send() و start_timer()، و rdt_rcv()، وهلم جرّا)، فقم بتوضيح وظيفة كل إجراء. اعط مثالاً (جدولاً زمنياً لتعاقب الأحداث في المرسل و المستقبِل) يبيّن كيف يمكن لبروتوكولك التعافي من فقد رزمة.
16. خذ في الاعتبار سيناريو يريد فيه المضيف A إرسال رزم في نفس الوقت إلى كلٍّ من المضيف B والمضيف C. يتصل المضيف A بكلٍ من المضيف B وC عن طريق قناة إذاعة

– كل رزمة تُرسل من A تُنقل إلى كلٍّ من B وC على القناة. افترض أن قناة الإذاعة التي تربط مابين A وB وC يمكن أن تفقد أو تُفسد الرزم بشكلٍ مستقل (ومن ثم يمكن مثلاً لرزمة مرسلة من A أن تصل سليمة إلى B ولكن ليس إلى C). صمّم بروتوكولاً للتحكم في الخطأ من نوع "قف وانتظر" للنقل الموثوق للرزم من A إلى B وC بحيث لا يحصل A على بيانات جديدة من الطبقة الأعلى قبل التأكد من أن كلاً من B وC قد استلم الرزمة الحالية صحيحة. اعط وصفاً لآلة الأوضاع المحدودة (FSM) على كلٍّ من A وC. (ملاحظة: يجب أن تكون آلة الأوضاع المحدودة (FSM) على المضيف B هي نفسها على المضيف C تقريباً). قم أيضاً بإعطاء وصف لصيغ الرزم المستخدمة.

17. خذ في الاعتبار سيناريو يريد فيه كلٌّ من المضيف A و B إرسال رسائل الى C. المضيف A يرتبط بالمضيف C بقناة يمكن أن تُفقد وتُفسد (ولكن ليس إعادة ترتيب) الرسائل. يتصل المضيفان B وC بقناة أخرى (مستقلة عن القناة التي تربط A بـ C) لها نفس المواصفات. يجب أن تتناوب طبقة النقل الموجودة على C في تسليم الرسائل من A إلى B إلى الطبقة الأعلى (أي أنها ينبغي أن تسلّم أولاً البيانات من الرزمة من A، ثم البيانات من الرزمة من B، وهكذا). صمّم بروتوكولاً للتحكم في الخطأ من نوع قف وانتظر للنقل الموثوق للرزم من A إلى B إلى C بتسليم متبادل للبيانات عند C كما هو موضّح أعلاه. اعط وصفاً لآلة الأوضاع المحدودة (FSM) على كلٍّ من A و C. (ملاحظة: يجب أن تكون آلة الأوضاع المحدودة (FSM) على المضيف ب هي نفسها على المضيف A تقريباً). قم أيضاً بإعطاء وصف لصيغ الرزم المستخدمة.

18. خذ في الاعتبار بروتوكول "ارجع N للوراء" (GBN) فيه حجم نافذة المستقبل 3، ومدى الأرقام التسلسلية 1024. افترض أنه عند الوقت t يتوقع المستقبل استلام الرزمة ذات الرقم التسلسلي k كالرزمة التالية بالترتيب السليم. افترض أن الوسط لا يعيد ترتيب الرسائل. أجب على الأسئلة التالية:

a. ما هي مجموعات الأرقام التسلسلية الممكن وجودها داخل نافذة المرسل عند الوقت t؟ برّر إجابتك.

b. ما هي كل القيم الممكنة في حقل ACK في كل الرسائل الممكنة التي تنتقل حالياً عائدةً إلى المرسل عند الوقت t؟ برّر إجابتك.

19. افترض أن لدينا كيانان A وB على شبكة. يوجد لدى B مصدر بيانات سترسلها إلى A وفقاً للترتيبات التالية. عندما يتلقى A طلباً من الطبقة الأعلى لإحضار رسالة البيانات التالية (D) من B، يتعين على A إرسال رسالة طلب (R) إلى B على القناة من A إلى B. فقط عندما يتلقى B رسالة R يقوم بإرسال رسالة البيانات (D) إلى A على القناة من B إلى A. يجب على A تسليم نسخة واحدة فقط من كل رسالة بيانات (D) إلى الطبقة

الأعلى. يمكن أن تضيع الرسائل R (ولكن لا تفسد) على القناة من A إلى B؛ بمجرد إرسال الرسائل D فإنها تصل صحيحة دائماً. التأخير عبر كلا القناتين غير معروف ومتغير. صمّم (اعط وصفاً لآلة الأوضاع المحدودة (FSM)) بروتوكولاً يتضمن الآليات المناسبة للتعويض عن الفقد الذي تتعرض له البيانات على القناة من A إلى B ويقوم بتمرير الرسالة إلى الطبقة الأعلى في الكيان A كما هو موضّح أعلاه. استخدم فقط تلك الآليات الضرورية جداً.

20. خذ في الاعتبار بروتوكول العودة N للوراء (GBN) وبروتوكول الإعادة الانتقائية (SR). افترض أن مدى الأرقام التسلسلية هو k. ما هي أكبر نافذة مرسل مسموح بها من شأنها تجنّب حدوث مشاكل كالمبينة في الشكل 3-27 لكلٍ من هذين البروتوكولين؟

21. أجب بصح أو خطأ على كلٍ من الأسئلة التالية وبرر إجابتك بإيجاز:

a. مع بروتوكول الإعادة الانتقائية (SR)، يمكن أن يتلقّي المرسل إشعار استلام (ACK) لرزمة تقع خارج نافذته الحالية.

b. مع بروتوكول العودة N للوراء (GBN)، يمكن أن يتلقّي المرسل إشعار استلام (ACK) لرزمة تقع خارج نافذته الحالية.

c. بروتوكول البت المتناوب هو نفسه بروتوكول الإعادة الانتقائية (SR) بنافذة إرسال ونافذة استقبال حجم كلٍ منهما 1.

d. بروتوكول البت المتناوب هو نفسه بروتوكول العودة N للوراء (GBN) بنافذة إرسال ونافذة استقبال حجم كلٍ منهما 1.

22. ذكرنا أن التطبيق يمكنه اختيار UDP كبروتوكول نقل لأن UDP يوفر تحكماً أدق (مقارنةً ببروتوكول TCP) للتطبيق في أي البيانات يتم إرسالها في قطعة البيانات ومتى يتم ذلك.

a. لماذا يتمتع التطبيق بتحكم أكبر في أي بيانات يتم إرسالها في قطعة البيانات؟

b. لماذا يتمتع التطبيق بتحكم أكبر في وقت إرسال قطعة البيانات؟

23. خذ في الاعتبار نقل ملف طوله L بايت في وجود أخطاء من المضيف A إلى المضيف B. افترض أن الحجم الأقصى للقطعة (MSS) هو 1460 بايت.

a. ما هي القيمة القصوى للطول L بحيث لا تُستنفد أرقام TCP التسلسلية؟ تذكّر أن حجم حقل الرقم التسلسلي في بروتوكول TCP هو 4 بايتات.

b. لقيمة L التي حصلت عليها في (a) أعلاه، احسب الوقت اللازم لإرسال الملف. افترض إضافة ترويسة نقل وشبكة وربط بيانات بطول كلي قدره 66 بايتاً إلى كل قطعة بيانات قبل إرسال الرزمة الناتجة على وصلة بمعدل إرسال قدره 10

ميجابت/ثانية. اهمل التحكم في التدفق والسيطرة على الازدحام، بحيث يقوم المضيف A بضخ قطع البيانات الواحدة تلو الأخرى مباشرةً وبشكلٍ مستمر.

24. يتصل المضيفان A وB عبر توصيلة TCP، وقد استلم المضيف B بالفعل كل البايتات التي أرسلها A حتى البايت 248. افترض أن المضيف A يرسل بعد ذلك قطعتين متعاقبتين إلى المضيف B. تتضمن القطعة الأولى والثانية 40 و60 بايتاً من البيانات، على الترتيب. الرقم التسلسلي للقطعة الأولى هو 249، ورقم منفذ المصدر هو 503، ورقم منفذ الوجهة هو 80. يقوم المضيف B بإرسال إشعار استلام كلما استلم قطعة بيانات من المضيف A.
   a. في القطعة الثانية المرسلة من A إلى B، ما هو الرقم التسلسلي، ورقم منفذ المصدر، ورقم منفذ الوجهة؟
   b. إذا وصلت القطعة الأولى قبل القطعة الثانية، فما هو رقم إشعار الاستلام، ورقم منفذ المصدر، ورقم منفذ الوجهة في إشعار استلام القطعة التي تصل أولاً؟
   c. إذا وصلت القطعة الثانية قبل القطعة الأولى، فما هو رقم إشعار الاستلام في إشعار استلام القطعة التي تصل أولاً؟
   d. افترض أن القطعتين اللتين بعث بهما A وصلتا بنفس الترتيب إلى B. افترض أن إشعار الاستلام الأول فُقد في الطريق والثاني وصل بعد أول انقضاء لفترة الموقّت. ارسم مخططاً زمنياً يبيّن قطع البيانات تلك وكل القطع وإشعارات الاستلام الأخرى التي يتم إرسالها (افترض أنه لا يوجد أي فقد إضافي في الرزم). لكل قطعة بيانات في الشكل الذي سترسمه، بيّن الرقم التسلسلي وعدد بايتات البيانات؛ ولكل إشعار استلام تضيفه بيّن رقم إشعار الاستلام.

25. المضيفان A وB مرتبطان مباشرةً عن طريق وصلة بسرعة إرسال قدرها 200 ميجابت/ثانية. توجد توصيلة TCP بين المضيفين، ويقوم المضيف A بإرسال ملف بأخطاء إلى المضيف B عبر تلك التوصيلة. يمكن للمضيف A إرسال بيانات التطبيقات على الوصلة بسرعة 100 ميجابايت/ثانية ولكن المضيف B يمكنه قراءة مخزن الاستقبال المؤقت على TCP لديه بمعدّل أقصاه 50 ميجابت/ثانية. صف تأثير تحكم TCP في التدفق.

26. تم مناقشة كوكيز SYN في الجزء 3-5-6.
   a. لماذا يكون من الضروري لخادم استخدام رقم تسلسلي أولي خاص في SYNACK؟
   b. افترض أن المهاجم يعرف أن المضيف المستهدَف يستخدم كوكيز SYN. هل يمكن للمهاجم إنشاء توصيلات نصف مفتوحة أو مفتوحة تماما ببساطة عن طريق إرسال رزمة إشعار استلام ACK إلى المضيف المستهدَف؟ برر إجابتك.

27. خذ في الاعتبار إجراء TCP المستخدم لتقدير زمن رحلة الذهاب والإياب (*RTT*). افترض أن $\alpha = 0.1$ واعتبر أن $SampleRTT_1$ هي آخر (أحدث) عينة *RTT*، وأن $SampleRTT_2$ هي عينة *RTT* قبل الأخيرة، وهلم جرا...
    a. لتوصيلة TCP بعينها، افترض أنه تم إعادة أربعة إشعارات استلام بعينات *RTT* التالية: $SampleRTT_1$ و $SampleRTT_2$ و $SampleRTT_3$ و $SampleRTT_4$. عبّر عن القيمة المقدّرة لـ *RTT* (أي EstimatedRTT) بدلالة قيم عينات *RTT* الأربع.
    b. قم بتعميم النتيجة أعلاه لعدد *n* من عينات *RTT*.
    c. في المعادلة في الجزء (b)، دع *n* تؤول إلى ما لانهاية. علّق على سبب تسمية هذا الإجراء بالمتوسط المتحرّك الأسي.
28. في الجزء 3-5-3 ناقشنا تقدير *RTT* في بروتوكول TCP. لماذا في اعتقادك يتجنب بروتوكول TCP قياس SampleRTT لقطع البيانات المعاد إرسالها؟
29. ما هي العلاقة بين المتغير SendBase في الجزء 3-5-4 والمتغير LastByteRcvd في الجزء 3-5-5 ؟
30. ما هي العلاقة بين المتغير LastByteRcvd في الجزء 3-5-5 والمتغير *y* في الجزء 3-5-4؟
31. في الجزء 3-5-4 رأينا أن بروتوكول TCP ينتظر حتى يتلقّى ثلاثة إشعارات استلام (ACKs) مكرّرة قبل القيام بعملية إعادة إرسال سريعة (Fast Retransmit). لماذا في رأيك اختار مصممو بروتوكول TCP عدم القيام بإعادة إرسال سريعة بمجرد استلام أول إشعار استلام مكرر لقطعة بيانات؟
32. خذ في الاعتبار الشكل 3-46 (b). إذا زادت λ'_{in} عن $R/2$، هل يمكن أن تزيد λ_{out} عن $R/3$؟ اشرح. الآن خذ في الاعتبار الشكل 3-46 (c). إذا زادت λ'_{in} عن $R/2$، هل يمكن أن تزيد λ_{out} عن $R/4$ على افتراض أن الرزمة ستُمرّر مرتين في المتوسط من الموجّه إلى المستقبل؟ اشرح.
33. خذ في الاعتبار المخطط البياني التالي الذي يبيّن تغير حجم نافذة TCP كدالة في الوقت. افترض أن بروتوكول TCP رينو هو البروتوكول الذي يتعرّض للسلوك المبين أعلاه، أجب على الأسئلة التالية. وفي جميع الحالات، ينبغي توفير مناقشة قصيرة لتبرير إجابتك.
    a. حدّد الفترات من الوقت التي تكون فيها بداية TCP البطيئة شغّالة.
    b. حدّد الفترات من الوقت التي تكون فيها آلية TCP لتجنب الازدحام شغّالة.
    c. بعد جولة الإرسال رقم 16، هل يتم اكتشاف فقد قطعة عن طريق ثلاثة إشعارات استلام مكررة أم بانقضاء فترة الموقّت؟

d. بعد جولة الإرسال رقم 22، هل يتم اكتشاف فقد قطعة عن طريق ثلاثة إشعارات استلام مكررة أم بانقضاء فترة الموقّت؟.

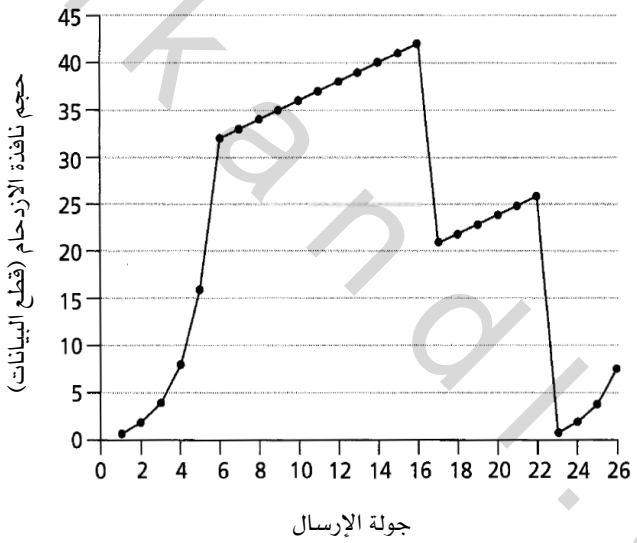
e. ما القيمة الأولية للعتبة (threshold) عند أول جولة إرسال؟

f. ما قيمة العتبة عند جولة الإرسال رقم 18؟

g. ما قيمة العتبة عند جولة الإرسال رقم 24؟

h. في أي جولة إرسال يتم إرسال قطعة البيانات رقم 70؟

i. بافتراض أنه يتم اكتشاف فقد رزمة بعد جولة الإرسال رقم 26 عن طريق تلقّي 3 إشعارات استلام مكرّرة. ماذا ستكون قيم كلٍّ من حجم نافذة الازدحام والعتبة؟



34. راجع الشكل 3-55، والذي يوضح تقارب خوارزمية AIMD لبروتوكول TCP. افترض أنه بدلاً من التناقص الضربي، يقوم TCP بتقليل حجم النافذة بكمية ثابتة. هل تؤول خوازمية AIMD الناتجة في هذه الحالة إلى خوارزمية حصص متساوية؟. قم بتبرير إجابتك باستخدام مخطط بياني يماثل الشكل 3-55.

35. ناقشنا في الجزء 3-5-4 مضاعفة فترة انقضاء الموقّت بعد حدوث انقضاء للفترة. تُعتبر تلك الآلية شكلاً من أشكال السيطرة على الازدحام. لماذا يحتاج بروتوكول TCP الى آلية مبنية على مفهوم النافذة للسيطرة على الازدحام (كالتي درسناها في الجزء 3-7) بالإضافة إلى آلية مضاعفة فترة انقضاء الموقّت تلك؟

36. يقوم المضيف A بإرسال ملف ضخم بأخطاء عبر توصيلة TCP إلى المضيف ب. عبر تلك التوصيلة لا يوجد أي فقد للرزم، والموقتات لا تنقضي فتراتها أبداً. ارمز لمعدل الإرسال على الوصلة التي تربط المضيف A بالإنترنت بالرمز R بت/ثانية. افترض أن العملية في المضيف A بوسعها إرسال البيانات إلى مقبس TCP الخاص بها بمعدل S بت/ثانية، حيث $S = 10R$. افترض أيضاً أن مخزن الاستقبال المؤقت لبروتوكول TCP من الضخامة بحيث يمكن أن يسع الملف بأكمله، بينما يسع مخزن الإرسال المؤقت 1٪ من الملف. ما الذي يحول دون تمكّن العملية في المضيف A من تمرير البيانات باستمرار إلى مقبس TCP بمعدّل S بت/ثانية: تحكم TCP في التدفق؟ سيطرة TCP على الازدحام؟ أم شيء آخر؟ أجب بالتفصيل.

37. خذ في الاعتبار عملية إرسال ملف كبير من مضيف إلى آخر على توصيلة TCP لا يتم عليها أي فقد.

a. افترض أن TCP يستخدم خوارزمية AIMD للسيطرة على الازدحام دون بداية بطيئة. بافتراض أن CongWin تزداد بـ MSS واحدة في كل مرة تصل فيها دفعة إشعارات استلام (ACKs) وأن أوقات رحلة الذهاب والعودة (RTT) ثابتة تقريباً، كم نحتاج من الوقت لكي تزداد CongWin من 1MSS إلى 6MSS (على افتراض عدم حدوث فقد)؟

b. ما هو المتوسط العام (بدلالة MSS وRTT) لتلك التوصيلة حتى الوقت $= 5RTT$؟

38. تذكّر الوصف الماكروسكوبي للطاقة الإنتاجية لبروتوكول TCP. في الفترة التي تغيّر فيها معدّل الإرسال على التوصيلة من $W/(2RTT)$ إلى W/RTT، فُقِدت رزمة واحدة (في آخر تلك الفترة تماماً).

c. بيّن أن معدل فقد الرزم (الكسر الذي يمثل الرزم المفقودة) هو:

$$L = \text{Loss rate} = \frac{1}{\frac{3}{8}W^2 + \frac{3}{4}W}$$

d. استخدم النتيجة أعلاه لتبيّن أنه إذا كان معدل الفقد على توصيلة هو L، فإن معدل الإرسال عليها يكون تقريباً:

$$\approx \frac{1.22 \text{ MSS}}{RTT\sqrt{L}}$$

39. في مناقشتنا لسمات بروتوكول TCP في الجزء 3-7، لاحظنا أنه لتحقيق طاقة إنتاجية قدرها 10 جيجا بت/ثانية يمكن للبروتوكول السماح باحتمال فقدان قطع بيانات قدره 2×10^{-10} (أي ما يعادل فقد قطعة واحدة من كل 5000000000 قطعة). بيّن اشتقاق

الاحتمال 2×10^{-10} (واحد لكل 5000000000) لقيم *RTT* وMSS المعطاة في الجزء 3-7. إذا كان على TCP توفير توصيلة بطاقة إنتاجية قدرها 100 جيجابايت/ ثانية، فما هو احتمال الفقد الأقصى المسموح به في القطع؟

40. في مناقشتنا للسيطرة على الازدحام في بروتوكول TCP في الجزء 3-7، افترضنا ضمناً أن مرسل TCP كان دائماً لديه بيانات يريد إرسالها. خذ في الاعتبار الآن الحالة التي يرسل فيها المرسِل كمية كبيرة من البيانات وبعد ذلك يتوقف عن الإرسال عند الوقت t_1 (لأنه لم يبق لديه بيانات لإرسالها). يبقى المرسِل عاطلاً لمدة طويلة نسبياً من الزمن ثم يستأنف بعدها إرسال المزيد من البيانات من جديد عند الوقت t_2. ما هي مزايا وعيوب استخدام TCP لقيم CongWin وThreshold المستخدمة عند t_1 عند استئناف إرسال البيانات من جديد عند t_2؟ ما هو البديل الذي تقترحه؟ ولماذا؟

41. في هذا التمرين نبحث ما إذا كان أياً من بروتوكولي UDP أو TCP يوفر قدراً من توثيق النقطة الطرفية.
   a. خذ في الاعتبار خادماً يتلقى طلباً ضمن رزمة UDP ويستجيب لهذا الطلب ضمن رزمة UDP (مثلاً على النحو المتّبع في خادم بنظام أسماء النطاقات DNS). إذا قام زبون له عنوان IP قيمته X بانتحال العنوان Y، فإلى أي عنوان سيرسل الخادم رده على الطلب؟
   b. افترض الآن أننا نستخدم TCP وليس UDP، هل سيكون بوسع الزبون خداع الخادم لجعله يرسل ردّه إلى العنوان Y ببساطة بمجرد انتحال عنوان IP الخاص به؟
   c. افترض أن خادماً يتلقى SYN مع عنوان مصدر IP قيمته Y، وبعد الاستجابة بـ SYNACK يستلم ويرسل إشعار استلام ACK على عنوان مصدر IP قيمته Y باستخدام الرقم الصحيح لإشعار الاستلام. بافتراض أن الخادم يختار الرقم التسلسلي الأوليّ بشكلٍ عشوائي وأنه لا يوجد هجوم من نوع "رجل في الوسط"، هل يمكن للخادم أن يكون على يقين من أن الزبون هو في الواقع Y (وليس زبوناً غيره له عنوان مختلف قام بانتحال العنوان Y)؟

42. في هذا التمرين سنأخذ في الاعتبار التأخير الذي ينشأ عن مرحلة البداية البطيئة في بروتوكول TCP. خذ في الاعتبار زبوناً وخادم ويب موصّلان عن طريق وصلة واحدة لها معدل إرسال *R* بت/ثانية. افترض أن الزبون يريد الحصول على كائن حجمه بالضبط يساوي $15S$، حيث *S* هو الحد الأقصى لحجم قطعة البيانات (MSS). افترض أن زمن رحلة الذهاب والإياب بين الزبون والخادم هو *RTT* (وافترض أنه ثابت). بإهمال ترويسات

البروتوكولات، احسب الوقت اللازم لكي يحصل الخادم على الكائن المطلوب (بما في ذلك وقت إنشاء توصيلة TCP) في كلٍّ من الحالات التالية:

a. $4\,S/R > S/R + RTT > 2S/R$
b. $8\,S/R > S/R + RTT > 4S/R$
c. $S/R > RTT$

❖ أسئلة للمناقشة

1. ما المقصود باختطاف توصيلة TCP ؟ كيف يمكن القيام بذلك؟
2. في الجزء 3-7 لاحظنا أن تطبيق زبون – خادم يمكنه بطريقة "غير عادلة" إنشاء العديد من توصيلات متوازية في وقت واحد. ما الذي يمكن عمله لجعل الإنترنت عادلة حقا؟
3. اقرأ البحوث المنشورة لمعرفة ما المقصود بالتعبير "متوائم مع TCP (TCP Friendly). أكتب وصفاً من صفحة واحدة للتواؤم مع TCP.
4. في نهاية الجزء 3-7-1 ناقشنا حقيقة أنه بوسع التطبيق فتح عدة توصيلات TCP للحصول على طاقة إنتاجية أعلى (أو بالمكافئ معدلات أعلى لنقل البيانات). ماذا يحدث لو أن كل التطبيقات حاولت تحسين أدائها عن طريق استخدام وصلات متعددة؟ ما هي بعض الصعوبات التي تكتنف محاولة عنصر من عناصر الشبكة تحديد ما إذا كان تطبيقٌ ما يستخدم وصلات TCP متعددة؟
5. بالإضافة إلى مسح منافذ TCP و UDP، ما هي الوظائف الأخرى لـ nmap؟ اجمع آثار رزم باستخدام إيثيريل (أو أي أداة أخرى لالتقاط الرزم) لعمليات تبادل رزم nmap. استخدم تلك الآثار لشرح كيف تعمل بعض السمات المتقدمة.
6. اقرأ البحوث المنشورة فيما يتعلق ببروتوكول النقل بالتحكم في مسارات البيانات (SCTP) [RFC 2960; RFC 3286]. ما هي التطبيقات التي يتصور مصممو SCTP أن يُستخدم فيها البروتوكول الجديد؟ ما هي السمات التي أضيفت إلى بروتوكول SCTP لتلبية احتياجات تلك التطبيقات؟

❖ أسئلة برمجة: تنفيذ بروتوكول للنقل الموثوق للبيانات

في تمرين مختبر البرمجة هذا، ستقوم بكتابة كود الإرسال والاستقبال على مستوى النقل لتنفيذ بروتوكول بسيط للنقل الموثوق للبيانات. هناك إصداران من هذا المختبر، إصدار خاص ببروتوكول البت المتناوب والآخر ببروتوكول العودة N إلى الوراء GBN. هذا المختبر سيكون ممتعاً – كما أن البروتوكول الذي ستطوره في نهاية المختبر لن يختلف كثيرا عما هو مطلوب في العالم الحقيقي.

نظراً لأنه قد لا يتوافر لديك أجهزة قائمة بذاتها (بنظام تشغيل يمكنك تعديله)، فإن الكود الذي ستطوّره ينبغي تشغيله من خلال بيئة محاكاة من البرامجيات والعتاد. ومع ذلك، فإن واجهة البرمجة التي يتم توفيرها لبرامجك – أي أجزاء الكود التي سوف تستدعي برامجك من أعلى ومن أسفل – تعتبر قريبة جداً مما يحدث في بيئة يونيكس حقيقية. (وبالفعل، فإن واجهات البرمجيات البينية الموصوفة في تدريب البرمجة هذا هي أكثر واقعية بكثير من المرسل والمستقبل بدورة تنفيذ لانهائية (Infinite Loop) والتي تستخدمها العديد من الكتب الدراسية). يتم أيضاً محاكاة بدء وإيقاف الموقّتات، كما أن المقاطعة بسبب الموقّت سوف تقوم بتفعيل برامجك الخاصة بالتعامل مع الموقتات.

يوجد تدريب المختبر كاملاً، بالإضافة إلى الكود التي ستحتاج لتجميعها مع الكود الخاصة بك، على الموقع الخاص بهذا على الإنترنت: http://www.aw1.com/kurose-ross.

❖ مختبر إيثيريل: استكشاف بروتوكول TCP

في هذا المختبر سوف تستخدم متصفح الشبكة لديك للوصول إلى ملف من خادم ويب. كما هو الحال في مختبرات إيثيريل السابقة، ستستخدم إيثيريل لالتقاط الرزم الواصلة إلى جهاز الحاسب الخاص بك. ولكن خلافاً للمختبرات السابقة، سيكون بوسعك أيضاً تنزيل أثر رزم يمكن قراءتها بواسطة برنامج إيثيريل من خادم الويب الذي قمت بتنزيل ذلك الملف منه. في أثر الرزم ذلك من الخادم، ستجد الرزم التي تم إنشاؤها على الخادم نتيجة وصولك إلى خادم الويب. ستقوم بتحليل آثار الرزم على كلٍ من جانبي الزبون والخادم لاستكشاف جوانب من عمل بروتوكول TCP. على وجه الخصوص، ستقوم بتقييم أداء توصيلة TCP بين حاسبك وخادم الويب. سوف تقوم بتتبع أداء نافذة TCP، واستقراء المعلومات عن فقدان الرزم، وإعادة الإرسال، و التحكم في التدفق، والسيطرة على الازدحام، وتقدير زمن رحلة الذهاب والإياب.

كما هو الحال مع جميع مختبرات إيثيريل، يوجد وصفٌ كاملٌ لهذا المختبر على الموقع الخاص بهذا الكتاب على الإنترنت: http://www.aw1.com/kurose-ross.

❖ مختبر إيثيريل: استكشاف بروتوكول UDP

في هذا المختبر القصير ستقوم بعمليات اقتناص وتحليل الرزم لتطبيقاتك المفضلة التي تستخدم بروتوكول UDP (كنظام أسماء النطاقات DNS و تطبيقات الوسائط المتعددة مثل Skype). كما عرفنا في الجزء 3-3، فإن UDP بروتوكول نقل بسيط بلا رتوش. في هذا المختبر سوف تدرس الحقول المختلفة لترويسة قطع بيانات UDP بالإضافة إلى حساب المجموع التدقيقي.

كما هو الحال مع جميع مختبرات إيثيريل، يوجد وصفٌ كاملٌ لهذا المختبر على الموقع الخاص بهذا الكتاب على الإنترنت: http://www.aw1.com/kurose-ross.