Chapter 3 Transport Layer

3.1 transport-layer service (transport ทำไรบ้าง)

- ทำหน้าที่ในการสื่อสารทางลอจิค (logical communication) ระหว่าง host
- transport layer จะรันใน end system คอยทำงานดังนี้
  - ฝั่งผู้ส่ง แตก application message ออกเป็นชิ้นๆ เรียกว่า segment แล้วเอาไปส่งให้ network layer
  - ฝั่งผู้รับ รวม segment ที่รับมาให้กลับไปเป็น message แล้วส่งขึ้นไปยัง application layer
- Transport VS Network layer
  - Network จะรับผิดชอบในการส่งข้อมูลจาก host ไปยัง host
  - Transport จะรับผิดชอบในการส่งข้อมูลไปให้แต่ละ process (program)
- Internet transport layer protocols
  - ส่งติดแน่นอน, เรียงลำดับการส่ง -> TCP
  - ส่งติดมั้ง, ไม่เรียงด้วย -> UDP

3.2 multiplexing and demultiplexing

- multiplexing (ทำตอนส่ง)
  - รับข้อมูลจาก socket ต่างๆ
  - เพิ่ม transport header เข้าไปยังข้อมูล
- demultiplexing (ทำตอนรับ)
  - อ่านข้อมูลจาก header เพื่อส่งไปยัง socket ที่ถูกต้อง

3.3 connectionless transport: UDP

3.4 principles of reliable data transfer

- reliable data transfer protocol = rdt
- rdt 1.0
  - sender
    - ได้ข้อมูลมาจาก application ก็ส่งเลย
  - receiver
    - รอรับข้อมูล
  - * ข้อมูลต้องไม่พังและไม่หายถึงจะใช้ได้
- rdt 2.0
  - * เพิ่ม ACK, NACK
  - sender
    - ได้ข้อมูลมาก็ส่ง
    - ส่งเสร็จรอคำตอบจาก receiver
      - ถ้าเป็น ACK ส่งข้อมูลชุดต่อไป
      - ถ้าเป็น NACK ส่งข้อมูลชุดเดิมใหม่
  - receiver
    - ได้รับข้อมูลแล้วเอา checksum ที่ส่งมาด้วยเช็คข้อมูล
      - ถ้าข้อมูลไม่พัง ส่ง ACK กลับ
      - ถ้าข้อมูลพัง ส่ง NACK กลับ
  - * แล้วถ้า ACK/NACK พังล่ะ? (sender จะไม่รู้ว่า ตกลงกูส่งได้ป่ะว่ะ, receiver ก็งงๆ กูตอบมึงไปแล้วไง
- rdt 2.1
  - * เพิ่ม sequence number (สลับ 0, 1 ไปเรื่อยๆ (0, 1, 0, 1, …))
  - sender
    - * เริ่มต้นด้วยจะส่ง seq=0
    - ได้ข้อมูลมาแล้วส่งไปพร้อมกับ seq
    - รอคำตอบจาก receiver
      - ถ้าคำตอบเป็น NACK -> ส่งใหม่
      - ถ้าคำตอบพัง -> ส่งใหม่
      - ถ้าคำตอบเป็น ACK -> เปลี่ยน seq แล้วส่งข้อมูลต่อไป
  - receiver
    - * เริ่มต้นด้วยรอ seq=0
    - ได้รับข้อมูลมาแล้วเช็ค

- ข้อมูลพัง -> ส่ง NACK
- ข้อมูลไม่พัง แต่ seq ไม่ตรงกับที่รอ -> ส่ง ACK
- ข้อมูลไม่พัง seq ตรงกับที่รอ -> ส่ง ACK แล้วเปลี่ยน seq

* seq สลับ 0, 1 เพื่อ?
* sender state ต้องจำว่าจะส่ง 0 หรือ 1
* receiver state ต้องจำว่าจะรอ 0 หรือ 1

- rdt 2.2
  * ใช้แค่ ACK แต่มี seq ติดไปด้วย, ยังใช้ seq 0, 1 เหมือนเดิม
  - sender
    - ส่งข้อมูลไปพร้อม seq
    - รอคำตอบ
      - คำตอบพัง -> ส่งใหม่
      - ACK seq ไม่ตรงกับที่ส่งไป -> ส่งใหม่
      - ACK seq ตรงกับที่ส่งไป -> เปลี่ยน seq แล้วส่งข้อมูลต่อไป
  - receiver
    - ได้รับข้อมูลแล้วเช็ค
      - ข้อมูลไม่พัง seq ตรงกับที่รอ -> ส่ง ACK พร้อม seq (เก็บไว้ในตัวแปร sndpkt)
      - seq ไม่ตรงที่รอ -> ส่ง sndpkt (ACK เก่า สมมติถ้าตอนนี้รอ 0 sndpkt จะเป็น ACK1)
      - ข้อมูลพัง -> ส่ง sndpkt (ACK เก่า)
  * rdt 2.2 รองรับกรณีที่คำตอบพังแล้ว (จริงๆก็รองรับตั้งแต่ 2.1 แล้ว แต่ปรับให้ไม่ต้องใช้ NACK) แต่ยังไม่ครอบคลุมกรณีที่ ข้อมูลตายกลางทาง (packet loss)
- rdt 3.0
  * เหมือน rdt 2.2 แต่ sender จะมี time-out สำหรับรอคำตอบแต่ละครั้ง
  - sender
    - ส่งแล้วข้อมูลแล้วเริ่มจับเวลา
    - รอคำตอบจาก receiver
      - คำตอบพัง -> ~~ส่งใหม่~~ ไม่ต้องทำไร
      - ACK seq ไม่ตรงกับที่ส่งไป -> ~~ส่งใหม่~~ ไม่ต้องทำไร
      - ACK seq ตรงกับที่ส่งไป -> หยุดเวลา, เปลี่ยน seq แล้วส่งข้อมูล
    - หมดเวลา -> ส่งใหม่, เริ่มจับเวลา
  - receiver เหมือน rdt 2.2
- pipelined protocols
  - rdt 3.0 เป็น protocol แบบ stop-and-wait คือจะส่งทีละ packet แล้วรอคำตอบกลับมาก่อนถึงจะส่งต่อไป
  - pipelined protocols มีแนวคิดว่า “ทำไมกูต้องส่งทีละ packet วะ สายกูเส้นเบอเร่อ เก็บไว้นอนหรอ” ก็เลยจะส่งทีละหลายๆ packet แล้วค่อยรอคำตอบ
  - แนวคิดของ pipelined สามารถนำไปปฏิบัติได้ 2 วิธีคือ Go-Back-N และ Selective Repeat
- Go-Back-N
  - วิถีของ Go-Back-N ได้กล่าวไว้ว่า
    - จะมี packet ที่ยังไม่ได้ ACK อยู่ในท่อได้ครั้งล่ะไม่เกิน N packet
    - receiver จะตอบกลับแค่ cumulative ACK
    - sender มี timer จับเวลาแค่ packet ที่เก่าที่สุด
      - ถ้า timer ตัวนี้หมดเวลา จะส่ง packet ที่ยังไม่ได้คำตอบใหม่ทั้งหมด (go back ไงล่ะ)
    - ใช้ง่าย ไม่เปลือง resource ไปสร้าง timer ด้วยนะเทอว์ (แต่เปลือง bandwidth นะอิอิ)
  - การทำงานของ Go-Back-N
    * มีระบบ window เพิ่มระบุว่ามี packet ไหนบ้างที่กำลังส่งอยู่และยังไม่ได้ ACK
    - sender
      - มี window-size ขนาด N (N จากนิยามข้างบนเหละ)
      - window จะประกอบด้วย
        base ระบุ seq แรกของ window
        nextseq ระบุ seq ต่อไปสำหรับเพิ่มข้อมูลที่จะส่ง

N ระบุขนาดของ window

- เมื่อเริ่ม base=1, nextseq=1
- เมื่อข้อมูลเข้ามาใหม่จาก application
    - ถ้า nextseq >= base+N -> packet เต็ม window ยังไม่รับข้อมูลเพิ่ม
    - สร้าง packet เก็บไว้ที่ตำแหน่ง nextseq แล้วส่งข้อมูลไป
    *ถ้าbase==nextseq(แสดงว่าข้อมูลเป็น packet แรกใน window)->start time
- เมื่อ timeout
    - ส่งข้อมูลทั้ง window ใหม่หมด
- ถ้า ack ที่ตอบมากลับมาพัง -> ช่างมัน
- ถ้า ack ที่ตอบกลับมาไม่พัง
    - base = ack+1 (เลื่อน base ขึ้นจาก ack ที่ส่งมา)
    - ถ้า base = nextseq (ข้อมูลหมด window) -> stop time
    - ถ้า base != nextseq (ยังมีข้อมูลอยู่ใน window) -> start time

- receiver
    - มี expectnum เพื่อเก็บไว้ว่ารอ seq หมายเลขไรอยู่
    - ถ้า packet ที่ส่งมา seq==expectnum ส่ง ACKกลับ, expect++
    - ถ้า packet ที่ส่งมา seq!=expectnum ส่ง ACKกลับ (จะได้ค่า expect ล่าสุด)

3.5 connection-oriented transport: TCP
3.6 principles of congestion control
3.7 TCP congestion control