แก้ไขครั้งล่าสุดเมื่อวันที่ 13 เมษายน 2568

Chapter 9
Mobile Transport Layer

ในบทที่แล้วเมื่อคอมพิวเตอร์ในอินเทอร์เน็ตเคลื่อนที่จะมีผลกระทบกับ network layer (IP) ในบทนี้จะศึกษาผลกระทบกับ transport layer (TCP) ขอเรียกสั้น ๆ ว่า TCP ตาม TCP/IP model, TCP สร้าง connection ขึ้นมาระหว่าง 2 applications (เสมือนว่ามี "สายสัญญาณ" เชื่อมอยู่ ทั้งที่จริงๆ แล้วเป็น packet-switched network) TCP มีหน้าที่

- รับประกัน in-order delivery หั่นข้อมูลขนาดใหญ่เป็น packet เล็ก ๆ แล้วส่งให้ IP layer ทำหน้าที่ route หาเส้นทางส่ง packet นี้ไปบน internet แต่ packet อาจจะไม่ได้วิ่งไปในทิศทางเดิมเสมอ แต่เปลี่ยนเส้นทางเนื่องจาก traffic ทำให้ packet ที่ส่งก่อนอาจจะไปถึงผู้รับทีหลัง packet ที่ส่งทีหลังไปถึงก่อนก็เป็นได้ ที่ TCP layer ของผู้รับมีหน้าเรียง packet ให้ถูกต้อง
- Reliable transmission ทำ retransmission และ congestion control (ลด transmission rate ลงเมื่อ packet loss เพิ่มขึ้น) เพราะถ้า packet loss แสดงว่าส่ง packet เร็วเกินไป router ส่งข้อมูลออกไม่ทันและไม่มีพื้นที่เก็บข้อมูลมากพอ ก็ต้องโยน packet ทิ้ง

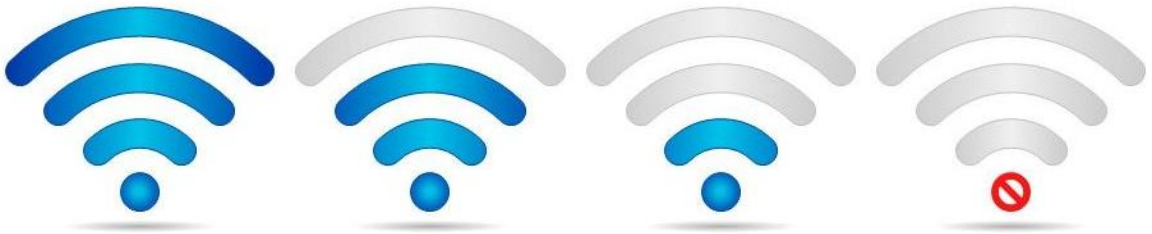
Congestion = ความแออัด

Traditional TCP: Congestion Control

- Congestion เกิดขึ้นบ่อย ๆ ในระบบ internet เมื่อมีปริมาณ packet จำนวนมาก router ส่ง (forward) packet ไม่ทัน และหน่วยความจำก็หมด ทำให้ router ต้อง drop packet ทิ้ง
- ผู้รับจะ acknowledge เฉพาะ packet ที่ได้รับ
- ผู้ส่งเห็นว่ามีบาง packet ไม่ได้รับ acknowledge กลับมา เดาว่าน่าจะเกิดจาก congestion ก็จะส่งใหม่โดยลด transmission rate ลง

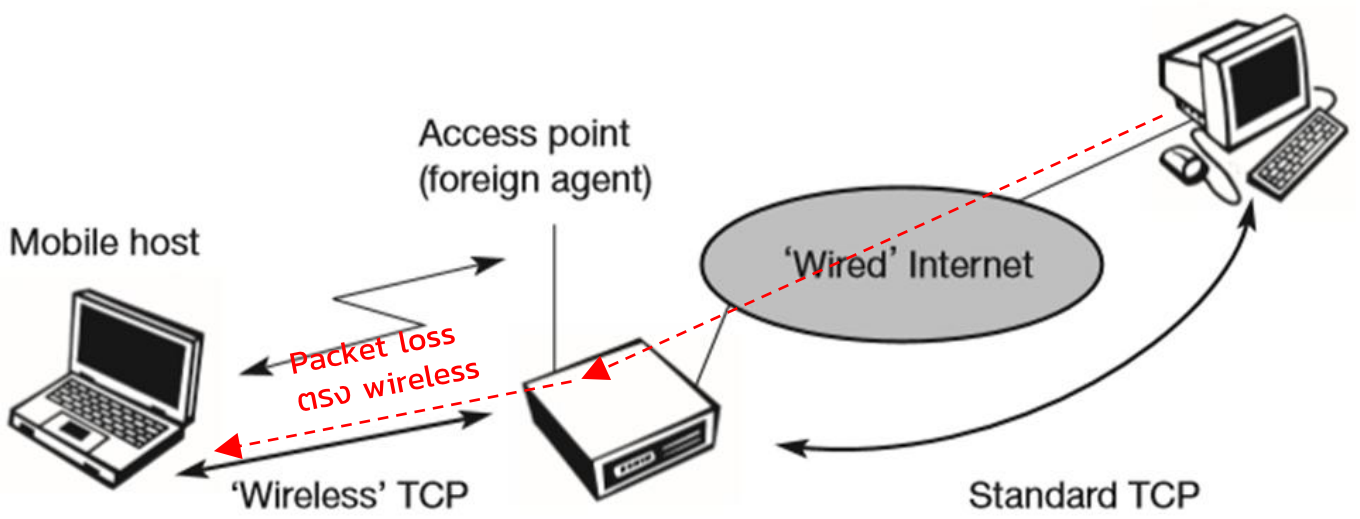
Traditional TCP: Slow Start

- TCP ป้องกันการเกิด congestion ด้วย slow start
- ผู้ส่งเริ่มต้นส่ง 1 packet ถ้าได้ acknowledge ก็จะ double เป็นส่งทีละ 2 packet และเพิ่มแบบ exponential ไปเรื่อย ๆ จาก 1, 2, 4, 8, 16, 32 จนถึง threshold ค่าหนึ่งจึงหยุด และเพิ่มทีละ 1 แทน
- เมื่อผู้ส่งรอ acknowledge จนหมดเวลา จะสรุปว่าเกิด congestion ผู้ส่งจะลดความเร็วในการส่งลงมาครึ่งนึง แล้วค่อย ๆ เพิ่มทีละ 1 เช่น 1, 2, 4, 8, 16, 32, 33, 34, 17, 18, 19, 20, 10, 11, 12, 13
- Wireless link ทำให้เกิด packet loss ได้ง่าย ทำให้ผู้ส่งไม่ได้รับ acknowledge และคิดว่าเกิด congestion ผู้ส่งจะลดความเร็วในการส่งข้อมูลลง (ทั้ง ๆ ที่ไม่ได้เกิด congestion) ทำให้ส่งข้อมูลได้ช้า

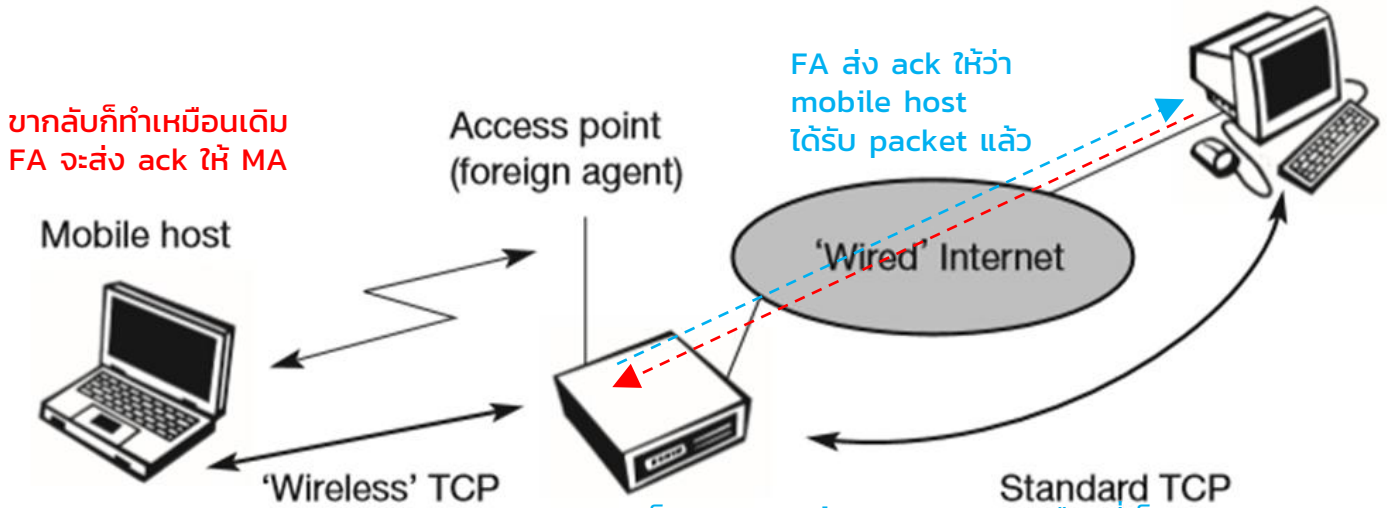


โอกาสที่จะเกิด packet loss ขึ้นอยู่กับคุณภาพสัญญาณ (signal strength)

Improvement 1: Indirect TCP (I-TCP)



ผู้ส่งไม่ได้รับ acknowledge
จึงลด transmission rate ลง
ถ้าเกิด packet loss บ่อย ๆ
ก็ทำให้สื่อสารกันไม่ได้ เพราะได้ data rate
ต่ำมาก

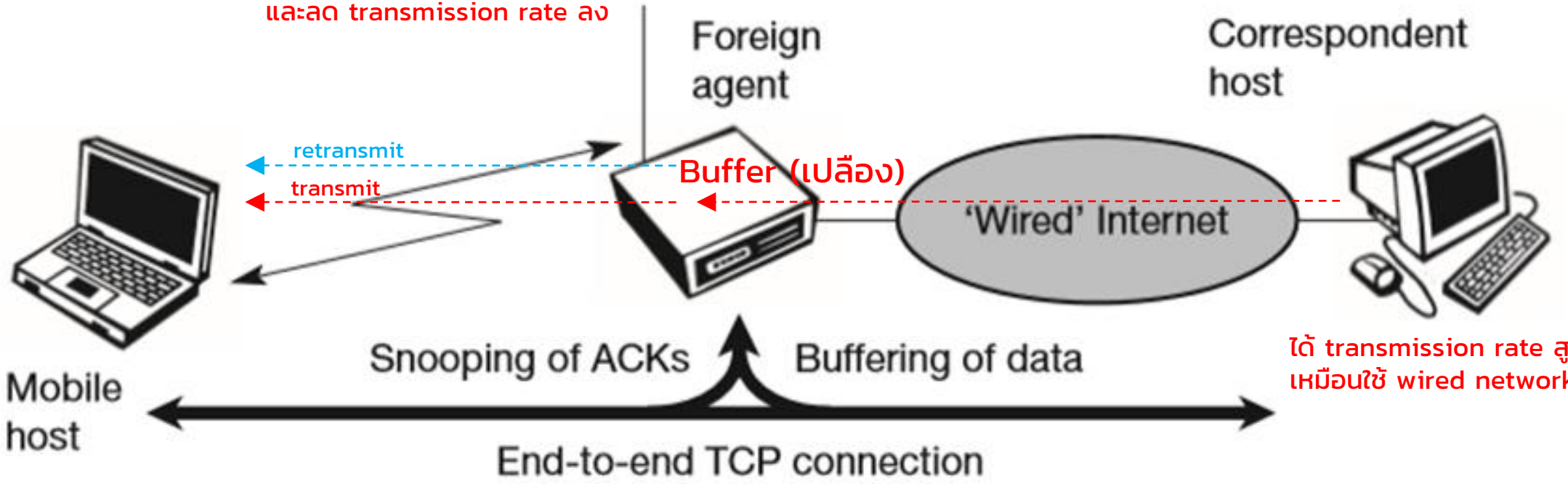


ได้ transmission rate สูงตามปกติ
เหมือนใช้ wired network

I-TCP ทำลาย
end-to-end semantic ของ TCP

Improvement 2: Snooping TCP

FA ดักฟัง ack จาก mobile host (snoop)
ถ้าไม่ได้ ack ภายในเวลาที่กำหนด
FA จะ retransmit packet
เพื่อ mobile host ส่ง ack ให้ corr. host
ก่อนที่จะถูก corr. host สรุปว่าเกิด congestion
และลด transmission rate ลง

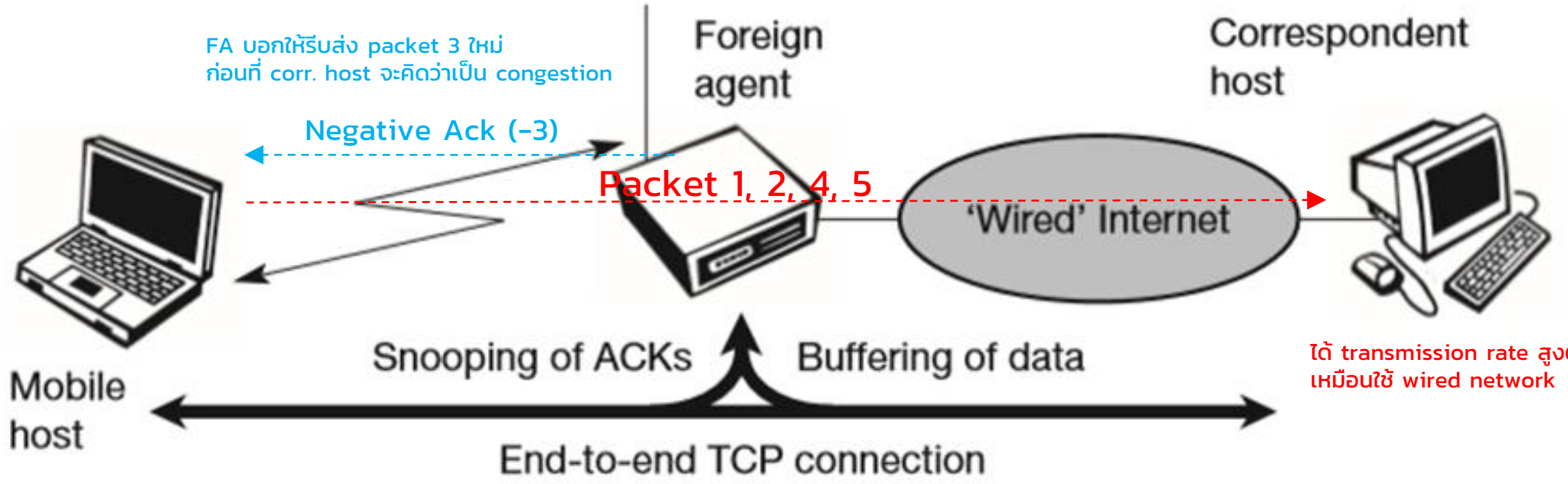


ได้ transmission rate สูงตามปกติ
เหมือนใช้ wired network

ถ้ามีการเข้ารหัสข้อมูล
จะ snoop ไม่ได้

Improvement 2: Snooping TCP (ขากลับ)

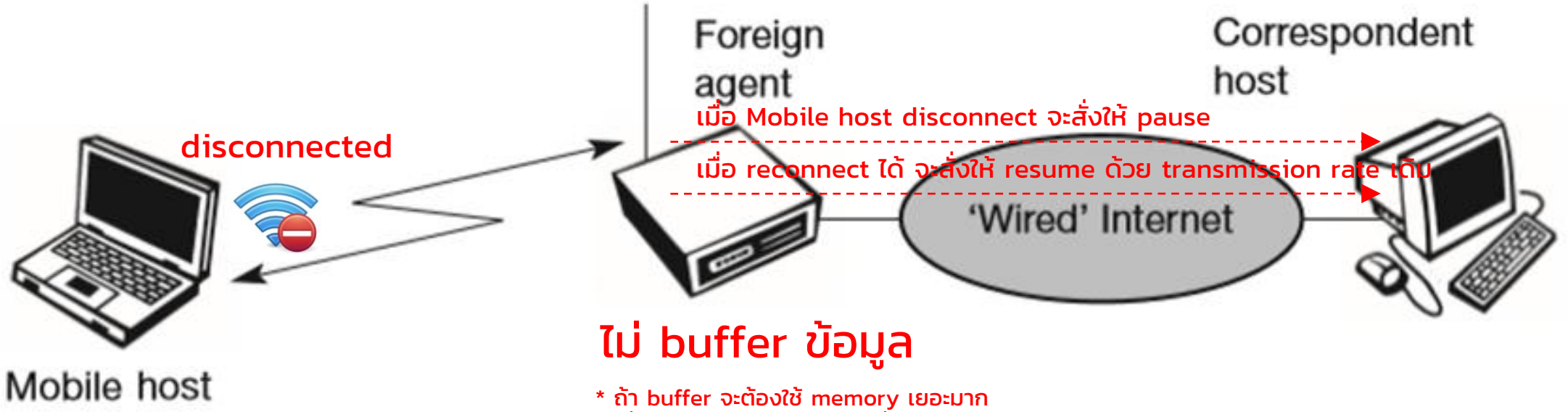
ต้องออกแบบ TCP (SW/HW) ใหม่
เนื่องจาก traditional TCP ไม่มีคำสั่ง negative ack



ได้ transmission rate สูงตามปกติ
เหมือนใช้ wired network

ถ้ามีการเข้ารหัสข้อมูล
จะ snoop ไม่ได้

Improvement 3: Mobile TCP

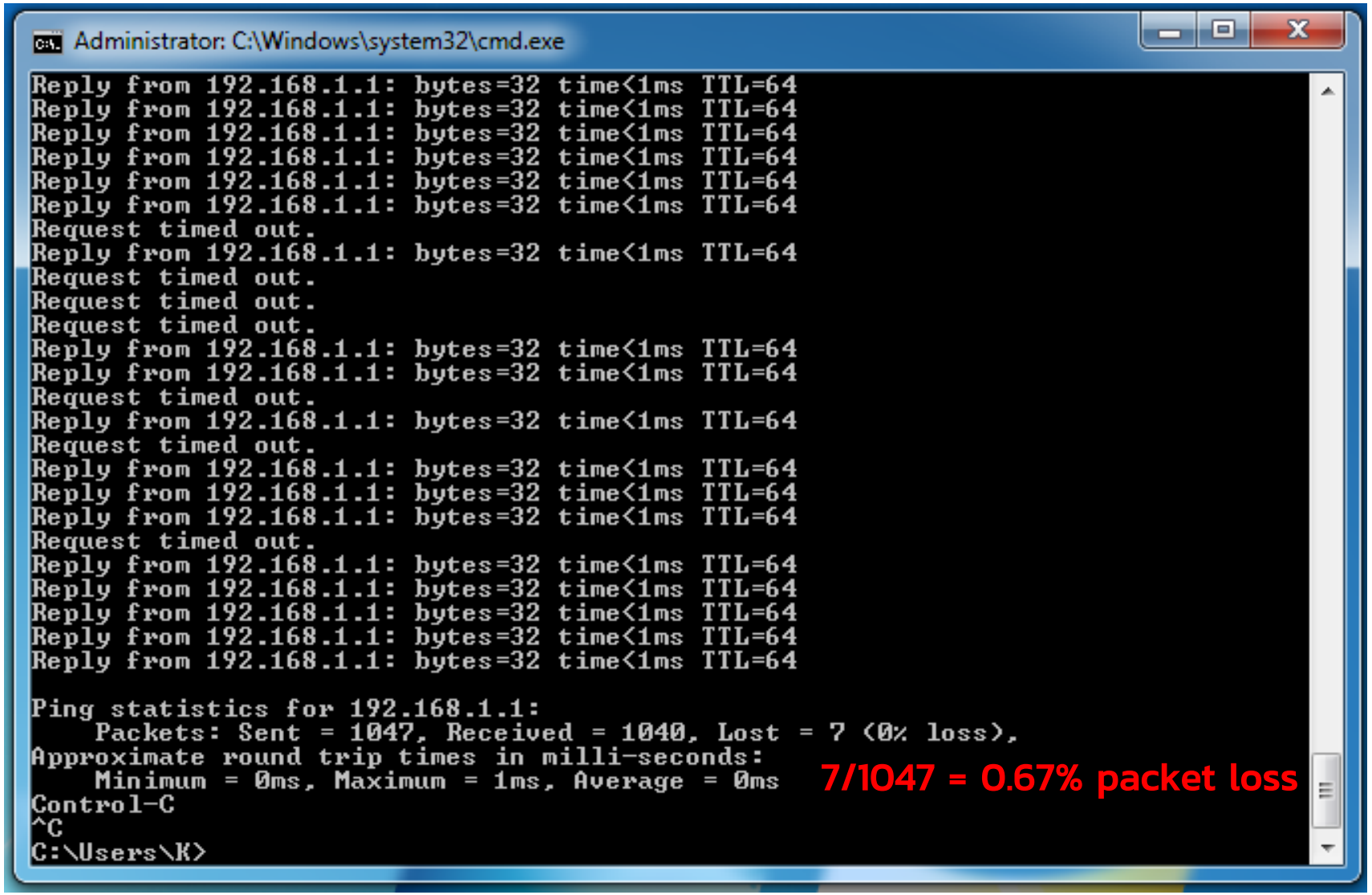


ไม่ buffer ข้อมูล

* ถ้า buffer จะต้องใช้ memory เยอะมาก
เมื่อ mobile host ขาดการเชื่อมต่อ
เป็นข้อเสียร้ายแรงของ I-TCP และ Snooping TCP ที่ buffer ข้อมูล
เมื่อ mobile host ขาดการเชื่อมต่อไปเป็นเวลานาน

ต้องออกแบบ TCP (SW/HW) ใหม่
เนื่องจาก traditional TCP ไม่มีคำสั่ง pause และ resume

ตัวอย่าง
ใช้คำสั่ง ping /n 100 www.google.com บน wireless LAN



สรุป

- wireless ปัจจุบันดีมากแล้ว แทบจะไม่เกิด packet loss
- เมื่อใช้งาน wireless ให้หลีกเลี่ยง packet loss โดยอยู่ใกล้ ๆ access point และไม่ให้มีสิ่งกีดขวาง/รบกวนสัญญาณ
- packet loss (%) เพียงเล็กน้อยจะทำให้ data rate ลงลดมาก

```
import React, { useState, useEffect } from "react"
import { useNavigate } from 'react-router-dom'
import { SideBar } from "./Sidebar"
import MaterialTable from "@material-table/core"
import axios from 'axios'
import { useCookies } from 'react-cookie'
import { toast } from "react-hot-toast";

function Main(props) {
  const [cookies, ] = useCookies(['token'])
  const [activities, setActivities] = useState([])
  let navigate = useNavigate()

  useEffect(() => {

    axios.get('http://localhost:5555/activities', { headers: { Authorization: 'Bearer ' + cookies.token }}).then((response) => {
      setActivities(response.data)
    }).catch((error) => {
      toast.error(error.response.status + " " + error.response.statusText)
    })
  }, []);
```

ติดตั้ง package
npm install @material-table/core

```
return (<div>
  <SideBar />
  <div style={{ marginLeft: 200, padding: 20 }}>
    <MaterialTable
      title="กิจกรรมของฉัน"
      columns={[
        { title: "กิจกรรม", field: "name" },
        { title: "วันเวลา", field: "when", type: "datetime" },
      ]}
      data={activities}
      editable={{ }}
      options={{ actionsColumnIndex: -1 }}
    />
  </div>
</div>)
```

ย้าย actions column ไปไว้ขวาสุด
ค่อย ๆ เติมใน editable

```
editable={{
        onRowAdd: async (newData) => {
          axios.post('http://localhost:5555/activities', newData, { headers: { Authorization: 'Bearer ' + cookies.token }}).then((response) => {
              setActivities((prev) => [...prev, newData])
              toast.success(response.status + " " + response.statusText)
          }).catch((error) => {
            toast.error(error.response.status + " " + error.response.statusText)
          })
        },
        onRowUpdate: async (newData, oldData) => {
          axios.put('http://localhost:5555/activities/' + oldData.id, { name: newData.name, when: newData.when }, { headers: { Authorization: 'Bearer ' + cookies.token }}).then((response) => {
            setActivities((prev) => prev.map((x) => x.id === oldData.id ? newData : x))
            toast.success(response.status + " " + response.statusText)
          }).catch((error) => {
            toast.error(error.response.status + " " + error.response.statusText)
          })
        },
        onRowDelete: async (oldData) => {
          axios.delete('http://localhost:5555/activities/' + oldData.id, { headers: { Authorization: 'Bearer ' + cookies.token }}).then((response) => {
            setActivities((prev) => prev.filter((x) => x.id !== oldData.id))
            toast.success(response.status + " " + response.statusText)
          }).catch((error) => {
            toast.error(error.response.status + " " + error.response.statusText)
          })
        }
      }}
```

ทำไมต้อง MVC ?

ต้องบอกก่อนเลยว่า MVC นั้นเป็น pattern การเขียนโค้ดรูปแบบหนึ่งที่ได้รับความนิยมมาก ซึ่งหากเริ่มเขียนซอฟแวร์แล้วยังมีฟังก์ชันน้อย หรือการทำงานน้อยก็อาจจะไม่เห็นภาพว่าทำไมต้องเขียนโค้ดให้เป็น pattern แต่ถ้าเขียนไปซักระยะหนึ่งมีฟังก์ชันเยอะขึ้น แล้วยิ่งถ้าไม่ได้เขียนนานแล้วกลับมาดูโค้ดที่ไม่มีการจัดรูปแบบ รับรองเลยว่าปวดหัวแน่นอน ซึ่งการเขียนโค้ดแบบเป็น pattern นี่แหละจะช่วยให้การอ่านโค้ดง่ายขึ้น และการจัดองค์ประกอบโค้ดที่แยกการทำงานชัดเจนที่เป็นจุดเด่น MVC จะทำให้เราเขียนโค้ดอย่างเป็นระบบและจัดการโค้ดได้ง่ายขึ้น

แนวคิดของ MVC เป็นยังไง ?

แนวคิดของ MVC นั้นจะใช้หลักการของ OOP ซึ่งแบ่งการทำงานหลักๆให้เป็นรูปแบบของ object โดยที่ MVC นั้นกำหนดชื่อ object มาให้เรียบร้อยแล้วตามชื่อเลยก็คือ model view controller ซึ่งมีกติกาคือการทำงานของทั้ง 3 object นี้จะแยกการทำงานอย่างชัดเจนห้ามก้าวก่ายงานกันเด็ดขาด

หน้าที่ และ ความสำคัญของแต่ละส่วน

Model

การทำงานของ model จะจัดการส่วนที่ข้อมูลทั้งหมดจะคอยเตรียมข้อมูลที่เหมาะสมไว้ และ model นั้นจะทำงานเมื่อ controller ร้องขอเท่านั้น

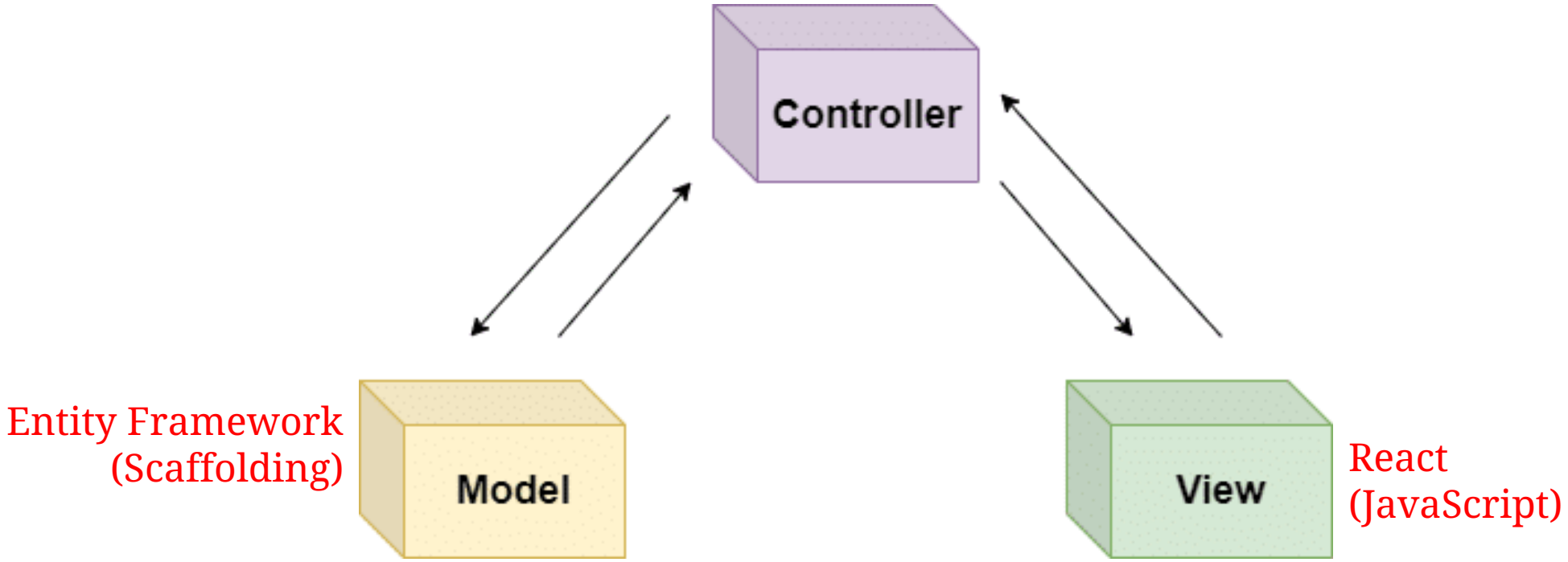
View

view นั้นจะจัดการส่วนของหน้าตาทั้งหมด หรือส่วนติดต่อกับผู้ใช้โดยตรง (user interface) โดย view นั้นจะรับคำสั่งการทำงานจาก controller และเป็นตัวกลางให้ผู้ใช้ติดต่อกับ controller อีกด้วย

Controller

controller เปรียบเสมือนกับมันสมองและศูนย์กลางการทำงานทั้งหมด จะเห็นว่าทุกส่วนนั้นจะติดต่อกับ controller ทั้งหมดรอคอยคำสั่งจาก controller นอกจากนี้ controller จะจัดการทำงานในส่วนที่เป็น logic ทั้งหมดในระบบ

Web API
(C#)



Entity Framework
(Scaffolding)

React
(JavaScript)

ASP.NET
MVC

เครื่องมืออื่น ๆ เช่น ASP.NET ก็เป็น MVC ได้
new project ครั้งเดียวได้ MVC ครบเลย
แบ่งงานให้โปรแกรมเมอร์หลาย ๆ คนช่วยกันทำไม่ได้