CSC0056 Data Communication

Week 4 — Data Communication Systems Evaluation

Instructor: Chao Wang

Networked Cyber-Physical Systems Laboratory
Department of Computer Science and Information Engineering
National Taiwan Normal University

Fall 2024

NATIONAL TAIWAN NORMAL UNIVERSITY

Outline

Note that this table of contents has hyper links embedded. Click on it to go to the corresponding part.

README first

This semester, I am experimenting with different styles of asynchronous lectures. Last week, we used video recordings and notes; this week, we will in addition use no-video slides (i.e., the one you're reading right now). The intention here is to use a series of conversation-style text to bring the reader through each topic. One reason of doing so is that, instead of having you hear what I spoke, it may be more helpful to have you read what I brought :) For the parts that demand demonstrations, I will use video.

Feel free to let me know what you think. That will help me in adjusting the format of presentation that will benefit the most.

Thanks,

Chao

Introduction to systems performance evaluation

This part includes slides only; no video.

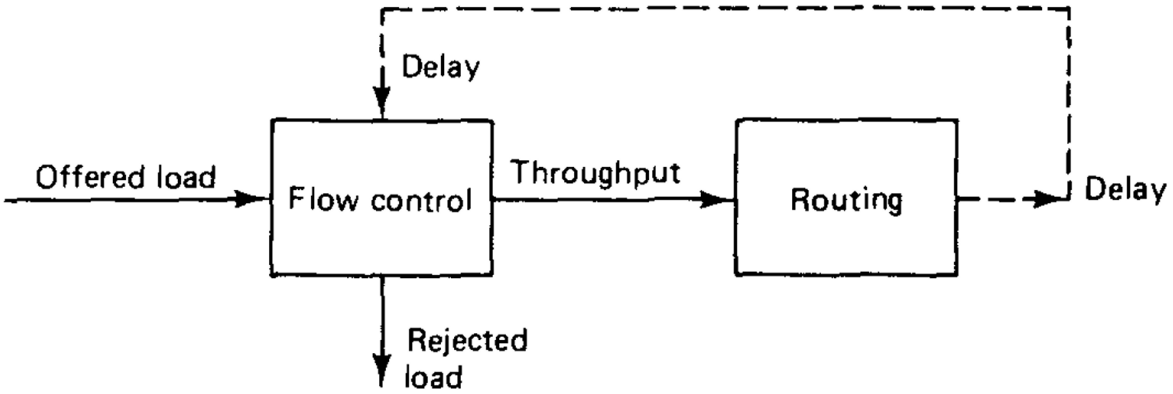
The figures used in this section are from the following textbook, Chapter 5:

Bertsekas, Dimitri and Gallager, Robert. Data networks (2nd edition). Prentice Hall, 1992. ISBN 0132009161.
(link to the author's book page)

Delay and throughput, a review

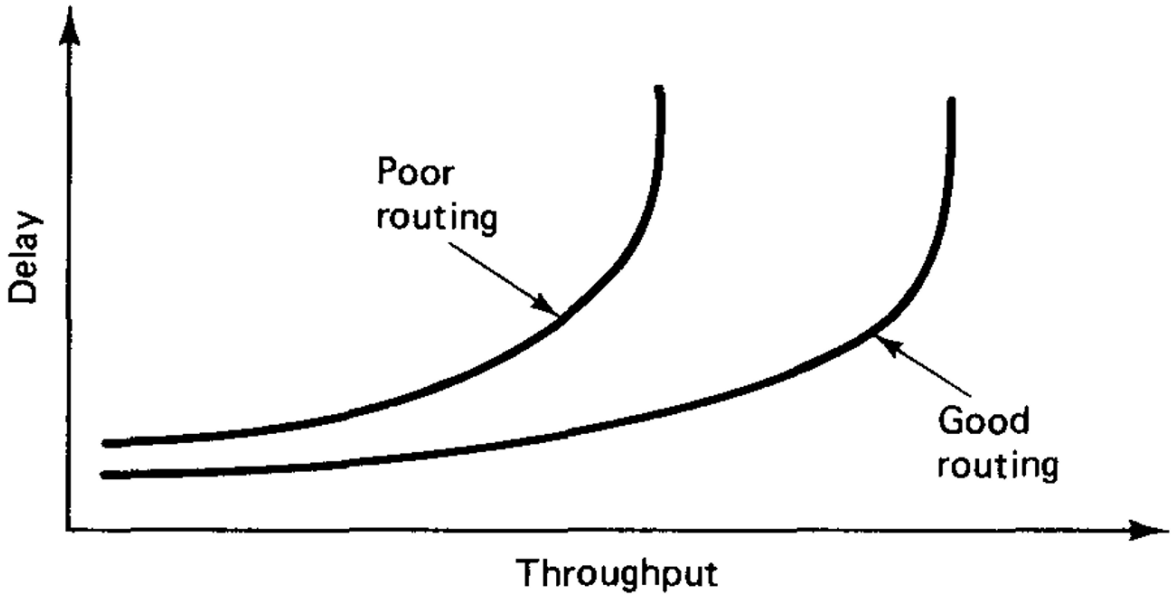
Back to the days when we studied computer network fundamentals, we have learned the idea of routing and flow control, where routing decides the delivery path of a packet from the sender to the receiver, and flow control decides the permission for a packet from the sender to *enter* the network. A consequence of the use of different routing protocols is different delay value (i.e., end-to-end latency) in packet delivery, and often, the delay value is used as a criterion that drives flow control. And the use of flow control will lead to different throughput (i.e., data rate). The relation of these four red keywords can be nicely summarized in the figure shown in the next page:

Relation between routing, flow control, delay, and throughput



Now think about it: how would delay and throughput relate to each other? If we try to plot the inter-dependency, using x-axis for throughput and y-axis for delay, how would the curve be? A typical answer is shown in the next page:

Delay and throughput



Try to make sense of these curves yourself first. In particular, try to answer the following two questions:

1. Look at one curve first. Why is it that as the throughput increases, the delay will not increase much at first, but will increase rapidly at last?
2. Now look at both curves. Why do we say one is good routing and the other is poor routing?

The answer is in the next page. Think about it yourself first.

Delay and throughput (cont.)

Answer:

1. In the beginning, flow control may permit most of the out-going traffic, because there is very light load in the network, and the flow control module observed this by monitoring the delay value. But as the load in the network increases, data buffering at various *points* (e.g., routers) in the network may cause packet drops, which in turn could bring up a chain effect of data re-transmissions and would result in rapid growth in the delay.
2. A poor routing may, for example, lead multiple packets to the same sub-path in the network, which would make worse data buffering and thus would cause an increase in delay sooner.

Delay and throughput in broker-based data communication

So we see the general relation between delay and throughput. Now, these performance metrics, delay and throughput, are also important in broker-based data communication, because in many network applications we look for timeliness (i.e., low latency) and we want good utilization of network resources (i.e., good throughput).

In the context of broker-based data communication, the analysis of delay and throughput is a bit different. Think about what the difference could be, before you turn the page.

Delay and throughput in broker-based data communication (cont.)

Here is my analysis: conceptually, we may think of a broker also as a *point* (in the same sense as what we used two pages before), within which messages may be buffered, waiting for delivery. Because the broker may be busy in finding the matched subscribers to send messages, new message arrivals will need to wait. Now here are some differences:

1. Typically, messages in the broker will not be dropped, unlike a router or a switch. But it may need to wait for some much longer time, depending on the design of the broker. So that affects the delay.
2. The throughput of the broker depends on the number of message subscribers. Suppose the incoming data rate of a broker is one message per second. The out-going data rate could be K messages per second, if there are K subscribers of the message.

Measuring latency and throughput in broker-based data communication

A good broker design for low latency and high throughput is an interesting research topic. But for now, we will first learn how to measure latency and throughput. In the following and in future homework assignments, we will use Mosquitto as an example for empirical measurement.

The learning objectives here are

1. Gain some hands-on experience in empirical measurement of latency performance.
2. Learn the concepts of measuring throughput.
3. Beware of some pitfalls in the above two endeavors.
4. Know how to plot the results of our measurement using Python.

Now, let's go for each of these objectives!

Measuring latency of Mosquitto

Watch the video I recorded; some notes were embedded in the video; see our Moodle page.

Measuring throughput (of Mosquitto)

For this section, instead, we do slides only; no video.

Theoretical throughput vs. empirical throughput

Previously, we have mentioned that, suppose the incoming data rate of a broker is one message per second, the out-going data rate could be K messages per second, if there are K subscribers of the message. Know, suppose the message size is $1KB$ and that $K = 10$, then we would expect that the in-coming data rate is about $8Kbps$ (i.e., $8K$ bits per second) and the out-going throughput is about $80Kbps$.

But in reality, the out-going throughput could be way lower than $80Kbps$. Try to come up with at least one reason for it, before you move on to the next page.

Theory vs. practice

In reality, both the broker design (and its implementation) and the operating system workload could hamper the throughput performance of the broker. Think about this, if our broker somewhat runs v—e—r—y slowly, than it might not be about to catch up with the in-coming rate of message arrivals—it is inundated. On the other hand, if our system is so busy that the CPU utilization reaches almost 100%, then depending on the process scheduling policy the broker process may be preempted by some other processes. Even if the CPU% is not that high, if the priority-based process scheduling is engaged, the broker process could still be preempted and blocked for a while. All these could reduce the actual throughput.

In general, most of today's system has become complicated enough that there could be many different factors that will make the actual system performance deviate from the theoretical one. Therefore, we will need to learn how to do empirical measurements.

Empirical measurement

To make empirical measurement of throughput, the example utility program that we will use here is `iftop`. It should be available on most UNIX-based operating systems (for example, run '`sudo apt install iftop`' on Ubuntu). In general, `iftop` helps display network usage of your machine's network *i*nter*f*ace, like what `top` does for displaying your CPU usage.

Now, have your `iftop` ready on your machine, and let's work on some hands-on examples (next slide).

Example measurement, by running `iftop` (1)

Here is the result I got when running 'sudo apt update' to update the package information. You should try to run it yourself, too:

```
                 19.1Mb          38.1Mb          57.2Mb          76.3Mb          95.4Mb
└──────────────────┴───────────────┴───────────────┴───────────────┴───────────────┘
172.28.157.2                     => ubuntu-mirror-1.ps5.canonical.com   185Kb  44.0Kb  11.0Kb
                                 <=                                     24.7Mb  5.16Mb  1.29Mb
172.28.157.2                     => ubuntu-mirror-3.ps5.canonical.com   156Kb  50.9Kb  12.7Kb
                                 <=                                     19.5Mb  4.82Mb  1.21Mb
172.28.157.2                     => 151.101.131.42                      832b   1.30Kb   333b
                                 <=                                     41.7Kb  18.5Kb  4.62Kb
172.28.159.255                   => Chao-Zenbook-2.mshome.net              0b      0b      0b
                                 <=                                        0b    139b     70b
─────────────────────────────────────────────────────────────────────────────────────────────
TX:             cum:   120KB   peak:   342Kb                 rates:    342Kb  96.2Kb  24.0Kb
RX:                    12.5MB          44.3Mb                           44.3Mb  10.0Mb  2.50Mb
TOTAL:                 12.6MB          44.6Mb                           44.6Mb  10.1Mb  2.53Mb
```

```
Get:8 https://packages.adoptium.net/artifactory/deb jammy/main amd64 Packages [9135 B]
Get:9 http://security.ubuntu.com/ubuntu jammy-security/main Translation-en [298 kB]
Get:10 http://security.ubuntu.com/ubuntu jammy-security/main amd64 c-n-f Metadata [13.3 kB]
Get:11 http://security.ubuntu.com/ubuntu jammy-security/restricted amd64 Packages [2439 kB]
Get:12 http://security.ubuntu.com/ubuntu jammy-security/restricted Translation-en [420 kB]
Get:13 http://archive.ubuntu.com/ubuntu jammy-updates/main Translation-en [357 kB]
Get:14 http://archive.ubuntu.com/ubuntu jammy-updates/main amd64 c-n-f Metadata [17.8 kB]
Get:15 http://archive.ubuntu.com/ubuntu jammy-updates/restricted amd64 Packages [2504 kB]
Get:16 http://security.ubuntu.com/ubuntu jammy-security/universe amd64 Packages [906 kB]
Get:17 http://security.ubuntu.com/ubuntu jammy-security/universe Translation-en [177 kB]
Get:18 http://security.ubuntu.com/ubuntu jammy-security/universe amd64 c-n-f Metadata [19.3 kB]
Get:19 http://archive.ubuntu.com/ubuntu jammy-updates/restricted Translation-en [432 kB]
Get:20 http://archive.ubuntu.com/ubuntu jammy-updates/universe amd64 Packages [1126 kB]
97% [20 Packages 975 kB/1126 kB 87%]                                    2063 kB/s 0s
```

Example measurement, by running `iftop` (2)

And here is the result I got when running 'sudo apt upgrade' to upgrade some installed packages in my system. Run it yourself, too:

```
                 19.1Mb          38.1Mb          57.2Mb          76.3Mb          95.4Mb
172.28.157.2                     => ubuntu-mirror-2.ps5.canonical.com   522Kb   283Kb  72.0Kb
                                 <=                                    52.3Mb  22.6Mb  5.74Mb
172.28.159.255                   => Chao-Zenbook-2.mshome.net             0b      0b      0b
                                 <=                                       0b    194b     83b
172.28.157.2                     => ubuntu-mirror-3.ps5.canonical.com     0b      0b     20b
                                 <=                                       0b      0b     12b

TX:             cum:   482KB   peak:   522Kb                rates:    522Kb   283Kb  72.0Kb
RX:                    41.4MB          52.3Mb                          52.3Mb  22.6Mb  5.74Mb
TOTAL:                 41.9MB          52.8Mb                          52.8Mb  22.9Mb  5.81Mb
```

```
Get:15 http://archive.ubuntu.com/ubuntu jammy-updates/main amd64 apparmor amd64 3.0.4-2ubuntu2.4 [598 kB]
Get:16 http://archive.ubuntu.com/ubuntu jammy-updates/main amd64 libpcap0.8 amd64 1.10.1-4ubuntu1.22.04.1 [145 kB]
Get:17 http://archive.ubuntu.com/ubuntu jammy-updates/main amd64 python3-update-manager all 1:22.04.21 [39.1 kB]
Get:18 http://archive.ubuntu.com/ubuntu jammy-updates/main amd64 update-manager-core all 1:22.04.21 [11.5 kB]
Get:19 http://archive.ubuntu.com/ubuntu jammy-updates/main amd64 ubuntu-standard amd64 1.481.4 [2948 B]
Get:20 http://archive.ubuntu.com/ubuntu jammy-updates/main amd64 curl amd64 7.81.0-1ubuntu1.18 [194 kB]
Get:21 http://archive.ubuntu.com/ubuntu jammy-updates/main amd64 libcurl4 amd64 7.81.0-1ubuntu1.18 [289 kB]
Get:22 http://archive.ubuntu.com/ubuntu jammy/main amd64 libeatmydata1 amd64 130-2build1 [7498 B]
Get:23 http://archive.ubuntu.com/ubuntu jammy/main amd64 eatmydata all 130-2build1 [5584 B]
Get:24 http://archive.ubuntu.com/ubuntu jammy-updates/universe amd64 emacs-el all 1:27.1+1-3ubuntu5.2 [16.1 MB]
Get:25 http://archive.ubuntu.com/ubuntu jammy-updates/universe amd64 emacs-nox amd64 1:27.1+1-3ubuntu5.2 [3556 kB]
Get:26 http://archive.ubuntu.com/ubuntu jammy-updates/universe amd64 emacs-bin-common amd64 1:27.1+1-3ubuntu5.2 [133 kB]
Get:27 http://archive.ubuntu.com/ubuntu jammy-updates/universe amd64 emacs-common all 1:27.1+1-3ubuntu5.2 [14.6 MB]
24% [27 emacs-common 5987 B/14.6 MB 0%]                                          2722 kB/s 41s
```

Interpreting the measurement results

Try to also run `iftop` along with some network-intensive applications (e.g., `wget` or `curl`) and watch how the data rate booms. Now, run '`man iftop`' to read the detail documentation and learn how to interpret the display.

In my previous cases, it shows that my system was getting package information from at least two mirror servers (ubuntu-mirror-1 and ubuntu-mirror-3), and that my system was downloading package upgrades from the other server mirror (ubuntu-mirror-2)—a behavior possibly due to some server-side load balancing policy. In the case of upgrading packages, the average download rate in the past ten seconds was about 22.6*Mbps*. Finally, we can also observe a typical asymmetry in these client-server applications: the client (my laptop) made some requests, generating small up-link traffic, and then the server (Ubuntu mirror site) responded with the requested data, in large down-link traffic.

I hope this simple exercise make the concept of throughput more vivid :)

Where is the mentioned throughput measurement for Mosquitto?

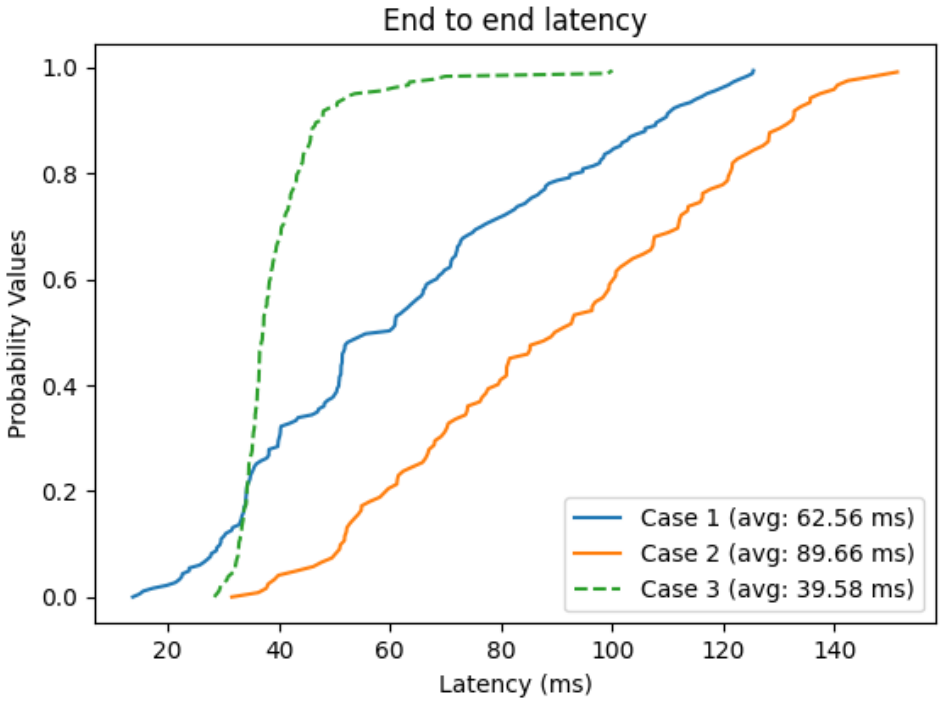
Having had some hands-on experience in throughput measurement, you might wonder: so what may we do for Mosquitto? No hurry. We will do that in future homework assignments.

Motivation to queueing theory

Having said that sometimes the actual system performance may deviate from the theoretical one, we by no means indicate that theory is useless. In contrast, theoretical analysis of system performance can be very useful. For one thing, it may help us predict the boundary values of our target performance metric (e.g., the maximum possible throughput, or a lower bound on latency); for another, it may give us some sense of the average performance of the system; for yet another, it may help us do quick analysis *before* we actually finish building the system. Besides simple theoretical analysis like those 1–vs.–K things that we've talked about, there actually exists a sophisticated framework of analytical tools—collectively called queueing theory—the surface of which we will scratch in the next couple weeks :)

A crash course for plotting experimental results

Suppose now we have made some empirical measurements. The next step is to visualize the results and try to make sense of them. For example, the following figure shows the CDFs (cumulative density functions) of some latency performance of a system, from one of my graduate students in his research project:



A crash course for plotting experimental results (cont.)

Now we are going to learn how to do data visualization like that, and a bit more. We will use Python and the Jupyter notebook. I've recorded a short video demo; see our Moodle page for both the video clip and the Jupyter notebook.

No worries if you've never used Python before—if you know C programming already, you will be able to learn general Python programming quickly.

In many cases, once we learned how to plot data in some specific style, the next time when we need to plot something else using the same style, we may simply change a bit of the existing plotting script to suit the new need. So, what you will learn here can serve as a template for your own future use.