# MP0: Event Logging

## Due: 11:59 p.m., Wednesday, Feb 8

# Notes

- This MP is largely intended to (re)familiarize yourself with the process of writing networked applications. Future MPs will be more challenging.
- You can do this (and all other MPs) in a group of 1 or 2 students. If you choose to work by yourself, the expected amount of work is the same, so we do encourage finding a partner.
- You are allowed to use any programming language for this MP; however, the TAs will only help with the four supported languages: C/C++, Go, Java, and Python.
- Your implementation will be tested on the CS425 VMs. It is your responsibility to make sure that the code runs on these VMs. Note that the VMs do not have persistent storage, so do not store the only copy of your code on the VMs! The typical practice is to use `git` to manage your code.

# Overview

A key tool in distributed systems is the ability to collect a log of events that have occurred across multiple processes. This allows you to obtain a picture of what is happening in your system and aids debugging and diagnosing crashes or other incorrect behavior. In this MP, you will implement a simple logging system that will send events from several *nodes* to a centralized *log server* that will collect them all in one place. You may find this functionality useful in future MPs.

## Event Generator

Events can be generated using this Python script: `generator.py` – do not edit this script (we will test your code using the original script provided here). For this assignment, events are represented as random strings of characters. Each event is preceded by the generation timestamp, expressed as a fractional number of seconds since 1970. Here is a sample of the output of `generator.py`:

```
% python3 generator.py 0.1
1610688413.782391 ce783874ba65a148930de32704cd4c809d22a98359f7aed2c2085bc1bd10f096
1610688418.2844002 b6b9592d531331512fd4f74b1e055434b2d8126e772dc30fb9b8c65298696517
1610688428.992117 4e51685633af8aacd4bcd2cfcee16bbbc2514be43faa20743f2d2cc4de853162
1610688432.144099 5828e97bf79bef141f2c243ab1203fd119a16a35d6354039c12289841bc33608
1610688433.771072 deaf6bc7b28c868fec560e40cffaeddaf757b677eab62b51e8bec87955ca3274
1610688449.1301062 ca6e5225e2ea02c1174701dd0320954fbfffb51dbcd9d15717e11d7e40556efb
1610688455.484428 ed4b1eb8a7bd980a1f0da41f5d6513e919e2bf201ba9ec9f9c05201bd777af94
1610688455.813278 3b014179e1cc1d2cc9cf553441492ad4f054634d2f0f0b66d0185c60fc4355da
1610688463.543133 8110f0cc37404a10989bfe14ae83224a73e642bb676ded625b08ed7d3e439706
[...]
```

The argument to `generator.py` is the *rate* at which events occur. Note that this is the average rate, specified in hertz (i.e., events / second), so 0.1 Hz = 1 event every 10 seconds on average. The events are exponentially distributed and so you could receive a burst of events spaced closer together.

## Centralized Logger

Your centralized logger should start by listening on a port, specified on a command line, and allow nodes to connect to it and start sending it events. It should then print out the events, along with the name of the node sending the events, to `stdout`. (If you want to include diagnostic messages, make sure those are sent to `stderr`).

```
% ./logger 1234
1610688413.743385 - node1 connected
1610688413.782391 node1 ce783874ba65a148930de32704cd4c809d22a98359f7aed2c2085bc1bd10f096
1610688418.2844002 node1 b6b9592d531331512fd4f74b1e055434b2d8126e772dc30fb9b8c65298696517
1610688426.373611 - node2 connected
1610688426.4092941 node2 b33cc5ccb2b360c95bc429e3fcd60bb003ce52d9345df033a4345bde49f5da2c
1610688428.992117 node1 4e51685633af8aacd4bcd2cfcee16bbbc2514be43faa20743f2d2cc4de853162
1610688432.144099 node1 5828e97bf79bef141f2c243ab1203fd119a16a35d6354039c12289841bc33608
1610688433.771072 node1 deaf6bc7b28c868fec560e40cffaeddaf757b677eab62b51e8bec87955ca3274
1610688447.583555 - node3 connected
1610688447.6169448 node3 4185ac49e15903e3d1c5a40458b3a3b3b7546626ef1bc41c454a00c6f038ed2f
1610688449.1301062 node1 ca6e5225e2ea02c1174701dd0320954fbfffb51dbcd9d15717e11d7e40556efb
1610688450.077894 node2 42ef6faea4a0dcd02daf88052e319b462dbc7770117ad583d22e779de977a5eb
1610688452.211595 - node2 disconnected
1610688454.279077 node3 e4283896d7389e362e0c9716e7b5e13b6e6e5e2b4ddff3dd3d3eb0c35d6bb8b5
1610688455.484428 node1 ed4b1eb8a7bd980a1f0da41f5d6513e919e2bf201ba9ec9f9c05201bd777af94
[...]
```

The first field is the time of the event; the second field is the name of the node that generated the event. The remainder of the line is the event itself. Connection events are specified by using `-` as the node name, as shown above. You do not need to implement an explicit failure detector; it is sufficient to create a TCP connection from the nodes to the logger and have the logger report when it closes. More specifically, the logger is `connected` to a node upon accepting the TCP connection created by the node. It is `disconnected` from the node when the TCP connection breaks and reading from the connection returns an error.

## Node

Your node should receive events from the standard input (as sent by the generator) and send them to the centralized logger. You will run your node as follows:

```
% python3 -u generator.py 0.1 | ./node node1 10.0.0.1 1234
```

The first argument is the name of the node. The second and third arguments are the address and port of the centralized logging server. This should be the address of your VM running the centralized server (e.g., VM0) and the port.

## Graphs

Once you see that your system is working, you will also want to evaluate its performance. To do this, you will need to generate graphs. In this assignment we want to track two metrics:

- Delay from the time the event is generated to the time it shows up in the centralized logger
- The amount of bandwidth used by the centralized logger

You will want to create an auxiliary log maintained by the centralized logger to track these two metrics. For the delay, you can just use the difference between the current time when you are about to print the event and the timestamp of the event itself. For measuring the bandwidth, you will need to track the length of all the messages received by the logger.

You should produce graphs of these two metrics over time (i.e. the metric on y-axis and the corresponding time on the x-axis). For the bandwidth, you should track the average bandwidth across each second of the experiment, i.e. count the total number of bytes received by the logger each second and plot it in units of bits per second (either Kbps or Mbps). For the delay, for each second you should plot the minimum, maximum, median, and 90th percentile delay at each second. If the logger receives no events during any one-second interval, you may plot the delay to be zero for that second. Make sure your graphs and axes are well labeled, with units.

## Evaluation scenarios

You will need to generate graphs to evaluate your system in two scenarios:

1. 3 nodes, 2 Hz each, running for 100 seconds
2. 8 nodes, 5 Hz each, running for 100 seconds

Your logger and each of the nodes will need to be running on a separate VM.

## Submission instructions

Your code needs to be in a git repository hosted by the Engineering GitLab. After creating your repository, please add the following NetIDs as members with *Reporter* access:

- gmk6
- jw22
- eashang2
- sm106
- radhikam

You will also need to submit a report through Gradescope. In your report, you must include the URL of your repository and the full revision number (git commit id) that you want us to grade, as shown by `git rev-parse HEAD`. Make sure that revision is `push`ed to GitLab before you submit.

In addition to this, your report should include:

- The names and NetIDs of the group members
- The cluster number you are working on
- Instructions for building and running your code. Please include a `Makefile` if you're using a compiled language! If there are any libraries or packages that need to be installed, please list those, too. There will be no demos. We will run your code ourselves and check that it works as expected. It is important that you provide sufficiently clear instructions on how to run your code; if we are unable to run it, we will not be able to score you on its functionality.
- A description of how you are measuring the delay and bandwidth
- Graphs of the evaluation as described above

# High-Level Rubric

- Report: 15 Points
  - Clear instructions on how to build and run your code, 5 points
  - Graphs, 10 points - Correctly computed metrics
    - Clear, readable graphs with labeled axes and units
- Functionality testing: 15 points
  - Small scenario: 3 nodes, 2 Hz each, 7 points
  - Large scenario: 8 nodes, 5 Hz each, 8 points

The source code of the website has been borrowed from Nikita Borisov