Network Simulator - Project Guidelines*

Linqi Guo (lguo@caltech.edu) Fengyu Zhou (f.zhou@caltech.edu)

October 19, 2018

1 Introduction

During the CS/EE/IDS 143 course, you will learn about the mechanics of communication networks as well as some of the theoretical and practical considerations related to analyzing their performance. As part of this process, you will write a program that can take a description of an abstract network and simulate its operation. This task will serve two purposes: to create a tool with which theoretical analyses of networks can be validated, and to help you gain a greater understanding of some of the common components and mechanics underlying today's networked world.

This is a group project. For some groups, this will be your first such substantial project. An ancillary goal, then, is to gain some experience working on a large project in a group. *This project has historically been very difficult even for experienced groups. Start early!*

2 Specifications

Your group is tasked with creating a computer program that fits the descriptor of "abstract network simulator". From a high-level overview, this program should:

- Take as input a description of a network in a format of your choice.
- Run a simulation of the described network for a user-specified duration.
- Record data from user-specified simulation variables at regular intervals.
- Output graphs after each run showing the progression of the specified variables over time.

We'll be leaving a lot of the details to your discretion, as long as the overall goals are met.

2.1 What's in a network anyway?

The abstract networks we're dealing with have a small set of components: hosts, routers, flows, and links. Hosts represent individual endpoint computers, like desktop computers or servers, while routers represent the network equipment that sits between hosts. Links represent the communication lines that connect hosts and routers together, while flows represent active connections.

Every host and router has a network address. This address uniquely identifies each node on the network. Flows will have source and destination addresses; packets generated by each flow will have the same destination address, to ensure they are routed correctly.

*This document is revised from previous year's project guideline by Ruijia Sun and Zilong Chen.

Links connect hosts and routers, and carry packets from one end to the other. Every link has a specified capacity. You may assume that every host and router can process an infinite amount of incoming data instantaneously, but outgoing data must sit on a link buffer until the link is free. Link buffers are first-in, first-out. Packets that try to enter a full buffer will be dropped. For the purpose of this project, all links are full-duplex (data can flow in both directions simultaneously).

Hosts will have at most one link connected; routers may have an arbitrary number. Routers will implement a dynamic routing protocol that uses link cost as a distance metric and route packets along a shortest path according to this metric. Each link will have a static cost, based on some intrinsic property of the link (e.g. its 'length'). Additionally, the link should also be able to take a dynamic cost dependent on link congestion (this will be used in one testcase, and only in that specific testcase). This dynamic routing protocol must be decentralized, and thus will use message passing to communicate among routers. This message passing must send packets along the link (and thus coexist with the rest of the simulation); 'telepathy' among nodes is not permitted, and all of the constraints that come with the fact that networks are distributed systems must be followed.

Flows have a source and destination address, and generate packets at a rate controlled by the congestion control algorithm defined for that flow. You should implement at least two different congestion control algorithms, e.g. TCP Reno and FAST-TCP, and be able to choose independently between them for each flow. Flows may send a continuous stream of data, or may send a finite user-specifiable amount of data; they may also start immediately or after some user-specifiable delay.

2.2 What are we measuring?

Ideally, you should be able to measure any simulation quantity you want; doing so will help you with the debugging process. However, there are a few metrics we are really interested in:

- Per-link buffer occupancy, packet loss, and flow rate.
- Per-flow send/receive rate and packet round-trip delay.
- Per-host send/receive rate.

For each of these, you should be able to produce both a time trace and an overall average. Writing the code to log these metrics and plot them at the end will help a lot with debugging.

2.3 How big should things be?

You may assume that flow-generated data packets have a fixed size of 1024 bytes, including any packet headers or trailers. You may assume that acknowledgement packets have a fixed size of 64 bytes.

Link buffer size, capacity, and propagation delay must be user-specifiable, as must flow data size and start time.

Any packet size we haven't mentioned is up to you, but must be large enough to contain the information the packet is intended to convey.

3 Project Timeline

This project is calibrated to take around eight weeks overall to complete. There are multiple milestones that your group will have to meet.

In addition to the milestones listed below, you will have a weekly meeting with either of the project TAs. These check-ins will serve as a means for the TA to gauge your progress and address any issues that may be occurring in any part of the project. They are also an opportunity (among many!) for you to ask questions or raise concerns that you may have.

3.1 Phase 1: Architecture Design (Weeks 4-6)

During these two weeks, you should meet with your group at least once to map out the overall architecture of your project. This task will probably take more than one meeting to complete, so plan accordingly. Your objective is to develop a greater understanding of the project requirements, clarify any preliminary questions, and develop a preliminary system design and plan for implementation.

You will be required to produce a short ($\sim$10-15 min) presentation explaining your group's approach to the task at hand, division of labor, and roadmap to completion. Observe that you may need to make revisions to your architecture as the project progresses, but you should map out as much as possible to make the implementation as smooth as possible.

Doing a good job in the design phase will make your subsequent work much easier and much more straightforward. While it is not required to have started writing code at this point, it would be good idea.

3.2 Phase 2: Preliminary Implementation (Weeks 7-8)

Bootstrapping a project is hard work, and we don't expect a fully functional implementation in a span of just a couple weeks. That said, you will be expected to have made at least some progress towards a full implementation.

You will be required to produce a short demonstration showing the progress your group has made so far. At a minimum, you should have implemented the following features:

- Working hosts and links.
- Basic router functionality with a statically generated routing table.
- Basic flow functionality with working packet sending and receiving.
- Enough statistical measuring functionality, input capability, and output capability to be able to demonstrate that these functions work.

This demonstration will be presented during your regular meeting with Project TAs during the eighth week (penalty may apply if no one from your group shows up during this week's meeting!). Test Case 0 should be working by this time, and Test Case 1 at least partially working.

3.3 Phase 3: Full Implementation (Weeks 8-10)

During this final phase of the project, you'll be extending your preliminary implementation, adding functionality and fixing any issues that arise. In particular, during this time you'll be implementing congestion control, dynamic routing, and enough I/O capability to run the test cases provided with this project.

In addition to the in-software simulation of the provided test cases, you will also (where applicable) be examining some of the test cases analytically, using techniques you will learn during the term. This process will help you validate the correctness of both your implementation and your analysis.

By the end of this stretch, you should have a fully spec-compliant implementation of your project and should have used it to examine the provided test cases. You are also required to produce a final presentation

that demonstrates your product and your simulation results. This presentation will be given during the final week of classes.

Additionally, you will need to produce a final technical report. This report should summarize the work your group has done. Include a description of your product, an analysis (both analytical and simulation-based) of the provided test cases, the labor division, and some commentary on the project process. This report will be due during finals week (exact date to be determined), along with your source code.

4 Grading Guidelines (50 pt.)

This project will compose 50% of your grade in the course, with the other 50% allocated to course homework. As such, we hope that you will take it seriously and exercise due diligence in its completion.

4.1 Weekly Project Meetings (5 pt.)

Regular meetings with the Project TAs are good ways to have your questions answered and ensure your group is on good track. It is not necessary to have the whole group to be present, yet we encourage you to at least designate someone from your group to meet with the TAs. Up to 5 points may be earned by being punctual and active during the meetings.

4.2 Presentation Deliverables (20 pt.)

It is important that you meet all project deadlines on time, and that your presentations are clear, coherent, and articulate. Up to 10 points may be earned per project presentation (including the design phase presentation and the final presentation), based on presentation quality and satisfactory completion of the objectives for each milestone.

4.3 Code Correctness (18 pt.)

A simulation is of little use if it cannot make accurate predictions about the phenomena of interest. Your simulator should be able, with appropriate input, to produce reasonable results for each different test case. Where an analytical solution is possible, your quantitative analytical solution should be correct, and your simulation results should agree with the analytical results within a reasonable margin of error (ideally less than 5%). The project TAs will announce new test cases without giving out sample time traces in the end of the term, and your group will need to include the plots for these test cases in your final report. Up to 18 points may be earned for a submission that meets this requirement.

4.4 Code Documentation and Style (7 pt.)

An unclean codebase is notoriously difficult to work with and to understand, as is a poorly documented one. Code should follow a consistent, legible format; all functions should receive accurate, descriptive headers; lines of code that are not self-evident to a relative novice should receive accurate, descriptive comments. Up to 7 points may be earned for a codebase that meets this requirement to our satisfaction. Note that we may deduct points beyond normal limits for especially gross violations of this requirement.

4.5 Individual Adjustments (± 10 pt.)

We expect groups to balance workload fairly among their members, and we expect students to put forth their best possible effort towards this project. If we determine that substantial inequity in labor distribution exists, we will make adjustments from the base group grade for individual members as we deem necessary.

5 Optional Objectives

Trust us when we say the main project is difficult enough! Members of the TA team have completed this project in the past, and can attest to this observation. These objectives are provided solely to provoke thought, and we don't expect any group to implement these. That said, if you find the main project unexpectedly easy, here are some additional things you might try, listed in no particular order. These objectives vary widely in difficulty, so choose only objectives you think your group can handle. While successful completion of any of these objectives will certainly cause the TA team to look favorably upon your group, we can't guarantee any sort of extra credit for any of these - but why let that stop you?

- We mention TCP Reno and FAST-TCP in this document, and should also cover TCP Vegas in class. In practice, these algorithms are not widely used currently: FAST-TCP remains proprietary, while TCP Reno and TCP Vegas show generally undesirable behavior. Microsoft writes its own congestion control algorithm, while modern Linux kernels default to an algorithm called CUBIC TCP. Try looking up and implementing the CUBIC TCP algorithm, and compare it to the ones we've mentioned in class: what kind of results do you get?

- The input objective could easily be satisfied by parsing a text file, XML file, or other structured input file. Editing such files would become cumbersome, especially with large test networks. Try creating a graphical tool that lets you edit these files interactively. You should be able to manipulate all parameters we mention in the spec - adding and removing objects, and specifying quantities to measure.

- The output objective can be satisfied by producing some structured output text that can be parsed by a graphing tool, like a MATLAB script or gnuplot. Try adding a module to your simulator that outputs the graphs directly instead. For extra instant awesome, have your simulator graph results as they are being generated and display them on the screen.

- We don't expect you to implement an entire realistic RFC-compliant TCP stack; doing so would be well beyond the scope of your project, and would pull time away from more pedagogically valuable objectives. But you could...

- All of the test cases we give you are mostly static: besides flows and packets, no objects are added or deleted during the simulation. But in the real world, hardware experiences problems all the time: routers break, hosts go offline, links experience decay and must be replaced. And in the real world, additional hardware gets added to a network all the time: new hosts are provisioned, new routers are placed, the network expands. Add the capability to add or delete objects dynamically during the simulation. For extra props, make this capability fully interactive.

- We'll have mentioned autonomous systems (AS) in class. For the purposes of this project, we assume all nodes are in the same AS. Drop this assumption. You'll need to implement a border gateway protocol, assign AS designations to nodes somehow, and build a test case big enough to show that your system works.