CS3251 Fall 2025
Programming Assignment 2

Handed Out: 31st Oct 2025   **Due:** 21st Nov 2025

1 Introduction

In this assignment, you will implement your own congestion control algorithm. Specifically, you will implement the congestion avoidance component of TCP Vegas. The goal of the assignment is to help you understand how congestion control algorithms are implemented and tuned. Thus, we do not expect you to write a lot of code as this implementation builds upon the existing UDT4 codebase[1].

TCP Vegas is a delay-based CC algorithm that attempts to estimate the congestion in the network based on sample RTTs and uses this to adjust the Congestion Window (CWND) before a loss event happens. Start by reviewing the Vegas paper, mainly the Abstract and Section 3.2. You will implement only Section 3.2.

This implementation, as evident in the provided template, utilizes UDT's Configurable Congestion Control (CCC) class.

We expect you to implement a fully functioning algorithm that keeps queues short and achieves high utilization. Your grade will be based on the performance of your implementation.

2 Instructions

2.1 Implementing the Congestion Control Algorithm

You will implement your algorithm in the `Vegas` class in `src/cc.h`, which inherits from the `CCC` class. We have provided a template implementation in the src/cc.h file. Read over the comments in the file and complete your implementation in the marked sections.

- You must implement the Vegas algorithm in the onACK() function of the Vegas class. The code structure is shown in Figure 1.

- The CCC class already has member variables that are populated by UDT to measure the RTT ($CCC::m\_iRTT$). You can set CWND simply by assigning a value to $Vegas::m\_dCWndSize$.

- In the provided skeleton, we already initialized some variable for you. Please do not change our initialization. Further, the skeleton code ensures that CWND never drops below 2.

- You will need to declare variables for you implementation of Vegas (alpha, beta, etc.) in the protected section of the Vegas class and initialize them in the marked section of the Vegas::init() function.

- Please don't change the initial values we set for predefined variables.

---

[1] https://udt.sourceforge.io/doc.html

Figure 1: Vegas Class Structure

Your `onACK()` function handles an acknowledgment packet when it is received. Implement your Vegas logic inside this function and update the **m_dCWndSize** variable to change the congestion window size. The **src/cc.h** file provides detailed documentation about the implementation you must complete, which can be summarized into the following steps:

- Calculate the expected throughput. This is the ratio between CWND and baseRTT, where CWND is the size of the current congestion window.

- Calculate the actual throughput (aka actual sending rate). This is calculated per RTT, i.e, every time an ACK is received. The details of the calculations are at the end of Section 3.2 of the TCP Vegas paper.

- Calculate the difference between the expected throughput and the actual throughput. Then, adjust the window size **m_dCWndSize** accordingly. In particular, if the difference is smaller than **alpha**, increase CWND by a small linear factor, `LinearIncreaseFactor * CWND`. If it is greater than **beta**, reduce it by the same amount. Otherwise, do not change the window.

You implementation will be evaluated based on its ability to achieve good throughput while keeping the RTT short and stable. We will use the standard deviation of the RTT over the period of a short experiment to test this stability. A low standard deviation for the RTT reflect a relatively stable queue length.

You are not required to implement slow start. Thus, our evaluation of your algorithm provides a few seconds for your algorithm to slowly probe the available bandwidth in the network. Also, we

don't require you implementation to react to loss. Thus, all our tests will include a bottleneck with very large (virtually infinite) buffer.

In our experience, the amount of code needed to implement the logic above is less than 30 lines of code, including variable definitions and initialization. We expect this part to be the easier component of the assignment. The main task is identifying appropriate `Alpha`, `Beta`, and `LinearIncreaseFactor` values based on window size. Please use the autograder to experiment with different values until you find the correct configuration that passes our test cases.

2.2   Building and Running Experiments

Use the Gradescope autograder to check if your implementation meets the project requirements. Pay attention to how the window size changes in the "Client Process Output" on Gradescope. If it stays the same or changes too quickly, you might need to tweak the values of `Alpha` and `Beta`. The `LinearIncreaseFactor` controls how fast the window size adjusts, so you can adjust it to get the right standard deviation.

If you prefer, we have provided a Dev Container environment and the associated Makefiles for compiling and debugging your implementation locally. While it is arguably more convenient, you do not need a fully functional local development environment to complete this assignment editing the "cc.h" file and testing your implementation with the Autograder should be sufficient in most cases. The bulk of this assignment should be tuning the Vegas parameters instead of implementation.

1. Install Visual Studio Code from `code.visualstudio.com`

2. Install Docker or Podman for your system. See the instructions on `docs.docker.com`

3. Install the "Remote Development" extension pack. If you are using Podman, follow the instructions here to configure the remote container extension.

4. Decompress the project archive and open the folder in Visual Studio Code. If you prefer Podman over Docker, read and modify the ".devcontainer/devcontainer.json" in the project folder accordingly.

5. VSCode should prompt you to re-open the project in a container. If not, use the keyboard shortcut Ctrl + Shift + P (Cmd + Shift + P on macOS) to open the command palette. Type and click "Dev Containers: Rebuild and Reopen in Container".

6. VSCode should set up the development environment as specified in the provided ".devcontainer/devcontainer.json" file. You may need to wait a few minutes for VSCode to install the required extensions inside the container.

7. Refer to the MakeFiles to compile your code. Highlevel steps are to first run *"make"* in the root directory and to run the appserver and appclient in the tests directory as follows: *./appserver 9000 Vegas* and *./appclient 127.0.0.1 9000 Vegas*, respectively.

3   Deliverable

Please submit your `src/cc.h` file to Gradescope. No other deliverables are necessary.

4   Helpful Resources

- `https://www.cs.emory.edu/~cheung/Courses/558/Syllabus/02-transport/vegas.html`
- `https://udt.sourceforge.io/udt4/doc/ccc.htm`
- `https://www.geeksforgeeks.org/basic-concept-of-tcp-vegas/`

5 Submission Guidelines and Rubric

5.1 Guidelines

- Upload only `cc.h` in the src/ folder to Gradescope.

- We do not limit the number of your submissions. But do keep in mind that the autograder will likely be congested closer to the deadline.

- The autograder sometimes may be nondeterministic for the same submission. Please run the submission until you get a consistent score.

- We will run your submission one final time after the deadline and use that grade. So, ensure that your configuration consistently achieves the target performance.

5.2 Rubric

We will create three different network conditions to test your algorithm. In the rubric below, STD DEV refers to the standard deviation of the Round-Trip Time (RTT) over the duration of the test. Each setting has a different Bandwidth Delay Product, to ensure that no static configuration of the window works for all of them. In particular, we will use the following scenarios:

- one-way delay of 200.00 ms, and bandwidth rate of 1Mbps

- one-way delay of 200.00 ms, and bandwidth rate of 10Mbps

- one-way delay of 300.00 ms, and bandwidth rate of 10Mbps

Your submission will be graded as follows (total 100 points):

- Compilation: 4 points

- **Test Case 1 (32 points)**:

  - STD DEV < 5ms: 8 points
    * Throughput > 0.2Mbps: 8 points (if STD DEV < 5ms)
    * Throughput > 0.4Mbps: 16 points (if STD DEV < 5ms)
  - Note that you will lose all points of the test case if STD DEV ≥ 5ms

- **Test Case 2 (32 points)**:

  - STD DEV < 10ms: 8 points
    * Throughput > 2Mbps: 8 points (if STD DEV < 10ms)
    * Throughput > 3Mbps: 16 points (if STD DEV < 10ms)
  - Note that you will lose all points of the test case if STD DEV ≥ 10ms

- **Test Case 3 (32 points)**:

  - STD DEV < 30ms: 8 points
    * Throughput > 2Mbps: 8 points (if STD DEV < 30ms)
    * Throughput > 3Mbps: 16 points (if STD DEV < 30ms)
  - Note that you will lose all points of the test case if STD DEV ≥ 30ms

6 Logistics

- Please **DO NOT** wait to the last minute to start this assignment! Starting only 1-2 days before the deadline will almost guarantee that you do not have enough time to finish. The Gradescope autograder experiences heavy load near the deadline, which significantly increases queue times and may affect the performance characteristics of your submissions. Start early to allow adequate time for testing and parameter tuning.

- Try to make use of existing threads set up by TAs to post questions, it helps keep things organized.

- Your question may already be answered: try to go through existing questions before posting new ones.