Integrity through Non-Fungible Assessments in Cloud-Based Technology Courses

Aspen Olmsted
Fisher College
Department of Computer Science, Boston, MA 02116
email: aolmsted@fisher.edu

***Abstract*— The US and world economies need more trained technical workers. These workers' demand has driven prominent private universities to create large, reduced-cost programs for graduate students. Unfortunately, less than twenty-five percent of the population has an undergraduate degree, and most do not have the pre-requisite knowledge to enter these graduate-level programs. In this paper, we look at developing an undergraduate technology program through cloud-based automatically graded labs and assessments that can guarantee the integrity and availability required to scale these programs to meet the demand for workers with these skills. We develop techniques to increase lab participation and integrity through a concept we call non-fungible labs. We also formulate testing assessments that allow each student to have a different version of the test. We provide preliminary evidence that these assessments have, in fact, increased engagement and integrity in our online sections of courses in our undergraduate Massive Open Online Courses computer science programs.**

Keywords-E-Learning; Cloud Computing; Cybersecurity; Auto-Graders

I. INTRODUCTION

Demand for Cybersecurity workers alone is estimated to increase three hundred and fifty percent between 2013 and 2021 [1]. In response, large universities have created online cybersecurity graduate programs with reduced tuition to attract adult learners. New York University (NYU) established a Cyber Fellows scholarship program that provides a 75% scholarship to all US eligible workers [2]. Georgia Institute of Technology (Georgia Tech) has created an online MS degree earned at the cost of fewer than ten thousand dollars [3]. Both of these programs are designed to scale to thousands of students. Fisher College [4] is a small minority-serving private liberal arts college located in downtown Boston, MA. At Fisher College, we have designed an undergraduate program designed to serve our students online with scalability and integrity.

Like many sciences, computer science devotes a great deal of the students' time to learning to hands-on lab activities. These labs include core application, programming, database courses, and upper-level information technology, computer science, and cybersecurity courses. In programming courses, these labs have the students write application code in the language of the course. In database courses, the students are often submitting SQL queries in response to question prompts. In information technology and cybersecurity courses, the labs are often steps taken on real systems to configure systems or eliminate vulnerabilities.

In face-to-face classes, instructors often use reverse classrooms to have hands-on time with the instructor or a Teaching Assistant (TA). When they get stuck, they can get started again quickly without a long duration between submissions. The students watch lectures, read and take quizzes at home and work on the labs to facilitate the just-in-time assistance. We show that students who learn with uninterrupted time do better in the completion of the labs.

There is often a long time between a question and submission for students in online classes and the response or feedback that allows them to continue learning in the lab. Online auto-graded systems help ensure that the student will immediately get feedback, but the student may have to wait for online office hours or a response to a forum post to continue with work. There is also a problem of ensuring integrity that the student submitting the result is the student who did the lab activities.

In this paper, we describe a technique we use in developing auto-graders that allow the student to receive feedback quicker while improving the integrity that the submitter is the author of the lab. The feedback comes in the form of auto-grader unit test results and allows for peer discussions around the assignments. The number and quality of peer discussions increased because, in some cases, each student has a unique derivate of the lab they the students are completing. We call these derivate labs non-fungible because the solutions to each lab are not mutually interchangeable. So, instead of stopping student peer communication about lab solutions, we can encourage student sharing. Students naturally want to discuss the problems when they run into issues. With fungible assessments, we discourage this. With non-fungible assessments, peer-to-peer student sharing has increased the students' understanding of the lab that cannot exist with fungible assessments.

The organization of the paper is as follows. Section II describes the related work and the limitations of current methods. In Section III, we describe the elements in the secBIML programming language. Section IV explains the auto-graders we developed for our database courses. Section V describes how we developed our auto-graders for programming courses. Section VI investigates the way we build auto-graders for upper-level computer science courses. In Section VII, we drill into the auto-graders in our cyber-security upper-level courses. Section VIII looks at our research questions and preliminary empirical data. In Section IX, we discuss early data in our work with non-fungible assessments and granularity. We conclude in Section X and discuss future work.

II. RELATED WORK

Jeffrey Ulman [2] developed an E-learning system with derivate questions. The system was called Gradiance Online

Accelerated Learning (GOAL). GOAL provided quizzes and labs for several core computer science topics, including operating systems, database design, compiler design, and computer science theory. Each course was linked to a textbook with several quizzes per chapter and sometimes a few labs. The examinations were composed of questions with separate pools of correct and incorrect answers. When students take an exam, they are presented with a multiple-choice quiz where one correct answer and several wrong answers are displayed for the student to choose the right answer. The system's standard configuration was four correct answers and eight incorrect answers—this configuration yield two hundred and twenty-four non-fungible questions per each original item in the quiz. We have used GOAL over the years in database courses and found the non-fungible quiz questions allowed them to discuss the exam without giving away the answer. The non-fungible versions of the assessment will also enable an instructor to answer a single version of an assessment as an example in an online lecture. Unfortunately, GOAL only proved derivatives in the quizzes and not in the labs. GOAL's labs were auto graded, giving students immediate feedback, but since all students worked on the same labs at home, it was hard to stop answer sharing. Our work here supplements the job done in GOAL by providing both automated grading of labs and non-fungible questions per student. It was easier for GOAL to have derivative quizzes than it was for developing the derivative labs. The GOAL system was designed to run learner self-paced entirely. Labs often lead to many students' questions, so our belief is the GOAL system did not want to tackle this challenge. Our implementation assumes some form of interaction with the instructors, TAs, or peers.

McGraw Hill [3] produces a commercial E-Learning product called SIMnet. SIMnet ambition is to teach students the skills required in the utilization of the Microsoft Office suite. SIMnet provides auto-graded labs that grade the students' submissions of database, spreadsheet, presentation, and word processing software. Students can learn the skills through online lessons that present the tasks in both reading and video format. SIMnet does protect the integrity of each student's work by inserting a unique signature into the starter file that the students download. If a student tries to upload a file with a different student's signature, the system catches the integrity violation. Either the upload is rejected, or the instructor is notified, depending on the lab configuration. Unfortunately, the labs that the students perform are not differentiated between students, so nothing stops one student from copy the work in the other students' files. Our work here improves the integrity of the students' submission by deriving a different problem per student so they cannot just copy the other student's work.

Gradescope [4] sells a commercial E-Learning product that allows instructors to scan student paper-based assignments. The grading of the paper-based assignments can then be automated through the E-Learning system. The scanning feature has driven many mathematics and science departments in universities to adopt the system. A relatively unknown function of the system is the auto-graded programming framework. Gradescope designed a system

TABLE I. SAMPLE VALUE TEST.

Field	*Value*
Name	*Assignment 1*
Value 1	*Product_code*
Value 2	*Prodcut_name*
Value 3	*List_price*
Order	*List_price*
Derivative	*Random Row*

that allows a student to upload a file for an assignment. The system then spins up a Docker [5] Linux session that is configured for the task. Test cases are developed in the auto-grader configuration with specific grading weights assigned for each test. We utilize this auto-grading environment for our non-fungible SQL, Python, and C# based labs.

III. DATABASE AUTO-GRADER

The auto-grader we developed for the database courses creates a docker environment with a MySQL database running on a Linux environment. The students upload their query with a specific name: query.sql. The auto-grader then reads the metadata about the assignment to determine the

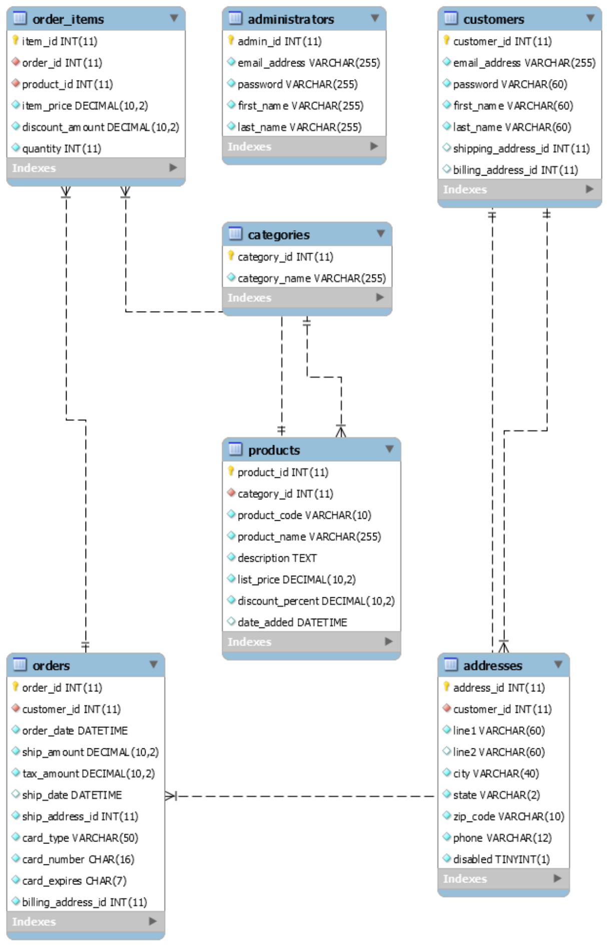

Figure 1. Student Lab ER Diagram

TABLE II. SAMPLE EXISTENCE TEST.

Field	*Value*
Name	*Assignment 2*
Test 1	*Show index from @RandomTable where key_name=@login_orders_ix*
Test 1 Type	*Exists*
Test 2	*Show index from @RandomTable where key_name=@login_orders_ix and column_name = '@RandomColumn*
Test 2 Type	*Exists*

assignment name and performs between three and five tests. Each test is weighted at two points each.

The test is actually stored in the MySQL database that is installed in the Docker session. The database connected has both the test information for the auto-graders and the same data provided to the students for their lab. There are two types of unit tests

- Value tests
- Existence tests

For the value tests, each assignment has a row in the value_unit_tests table. TABLE I shows a sample assignment row for a query that returns a specific product record. The primary key for the value_unit_test table is the name of the assignment. The value_unit_test also contains three value tests, along with an order test. The last value in the value_unit_test is the non-fungible method. Currently, supported non-fungible methods are:

- Random Row – In this non-fungible method, the student's login is converted to a unique number between 1 and the maximum assignment number. The unique number comes from an order the student id comes in the roster. The system will read the specific record in the order by value from the database to prompt the student to return that record
- Random Range – This is similar to Random Row but asks the students to return a tuple between a start and end value. The start value is the same as the random row value, and the end value is the fifth value after that record

Figure 1 shows the Entity-Relationship (ER) diagram for the student lab database. An assignment description website was built to display the question values that match the student's auto-grader tests logged in. The auto-grader will score the student on ten possible points distribute across five tests:

1. Did the query execute
2. Did the values match for the 1st value?
3. Did the values match for the 2nd value?
4. Did the values match for the 3rd value?
5. Did the order match

Existence tests are similar to value tests, except they are used to grade queries that mutate the database, such as insert, update and delete statements, and queries that create views, functions, stored procedures, and triggers. In the case of existence tests, the auto-grader will score the student on six possible points distribute across five tests:

1. Did the query execute
2. Did test 1 pass
3. Did test 2 pass

TABLE II shows an example of an entry for the existence unit test table. The case is from an assignment where the student needs to write a query to create a new index. The test types are either exists or not exists. The test will either pass or fail if there is a value returned from the query. For an exist unit test type, data should be returned for success. For not exists unit test type, no data should be returned for a successful test. Instead of a derivative based on a specific record as we used in the value unit tests, replacement variables are used to change the queries. TABLE III shows the available replacement variables. The variables allow names based on the user logged in, tables and columns to be different for each student, and literal string and numbers to randomized.

IV. PROGRAMMING AUTO-GRADER

We had previously developed a set of auto-graded foundational programming assignments in Python, Java, Visual Basic, and CSharp for students in a first programming class. Unfortunately, many of these assignments did not lend themselves to derivates that required different solutions per student. In our first attempt, we randomized the test cases to ensure students were not hard coding the output to match the input tests. To illustrate the challenge, we will itemize the labs below:

- Labs to Practice Programming Expressions:
  - Hello World – In this assignment, the student just outputs the words – Hello World.
  - Coin Counter – In this assignment, the student would be sent input variables for the number of quarters, dimes, nickels, and pennies and would

TABLE III. REPLACEMENT VARIABLES.

Variable	*Meaning*
@login	*The user code for the logged in user*
@RandomTable	*A random table from the students sample database. This variable can be suffixed with a number between 1 and 9*
@RandomColumn	*A random column from the random table selected in the variable above. This variable can be suffixed with a number between 1 and 9*
@RandomWord	*A random word from the dictionary*
@RandomInt	*A random possitive integer*

output the total in dollars and cents.
    - Coin Converter – In this assignment, the students would be given dollars and cents, and they would output the minimum number of coins by denomination.
    - BMI Metric – In this assignment, the student would send weight in kilograms and height in meters, and they would output the BMI.
    - BMI Imperial - In this assignment, the student would be sent input of weight in pounds and height in inches, and they would output the BMI. The students would need to convert the imperial measurements to metric before calculating the BMI.
    - BMI Metric with Status – This assignment is a modification of the earlier work and adds a decision branch to display Underweight, Normal, Overweight, or Obese. The students have not learned decision branching yet, so the expectation is they will use modular division for this problem.
- Labs to Practice Programming Iteration & Decision Branching:
    - Cash Register – This assignment allows multiple inputs of item prices along with a club discount card and tax rate. The student outputs the base price, price after discount, and total price.
    - Call Cost – This assignment provides the students with a rate table based on the day of week and time of day. Input is sent with the day, time, and duration of the call, and the students outputs the total cost for the request.
    - Even Numbers – This assignment has the student output a certain number of event numbers based on the number input.
    - Fibonacci - This assignment has the student produce the first n Fibonacci numbers. The number n is sent as input to the program.
- Labs to Practice Programming String Operations:
    - String Splitter – This assignment tests the student's ability to divide up an odd length input string into middle character, string up to the middle character, starting after the middle character
    - Character Type – This assignment has the student read a character of input and classify it into a lower-case letter, upper case letter, digit, or non-alphanumeric character.
- Labs to Practice Programming Functions:
    - Leap Year Function – This assignment has the student write a function that takes a parameter and return true if the year is a leap year
    - First Word Function – This assignment sends a sentence as a parameter to a function the student writes, and the student returns the first word of the sentence.
    - Remaining Word Function – This assignment sends a sentence as a parameter to a function the student writes, and the student returns the remaining words after the first word of the sentence.
- Labs to Practice Programming Lists:
    - Max in List Function – This assignment sends a list of integers as a parameter to a function the student writes, and the process should return the largest integer in the list.
    - Max Absolute in List Function – This assignment sends a list of integers as a parameter to a function the student writes. The function should return the maximum absolute value of each integer in the list.
    - Average in List Function – This assignment sends a list of integers as a parameter to a function the student writes, and the function should return the average of all the integers in the list.

A. *Non-Fungible Programming Labs*

We modified the above labs that allow students to practice programming expressions to receive a derivative version. Each of these labs initially provided the student with a formula or included an inherent method. For example, the Metric BMI lab provided students with the procedure to calculate BMI by taking the weight in kilograms and dividing by the height in meters squared. The currency-based labs used an inherent method for converting the value of each coin. For example, a nickel is worth five pennies in the US currency. We modified the labs to use different currencies or measurement systems for each student, so the calculations and currencies utilized different constants and exponents in the calculations. For example, one student would calculate the BMI using the formula of three times weight divided by two times height raised to the fourth power.

V. IT COURSE AUTO-GRADERS

This section will drill into different categories of lab auto-graders we have developed for Information Technology courses. Information Technology courses often use many tools in the labs to allow the students to understand the concepts from the lectures.

A. *Helpdesk Course Auto-graders*

In a helpdesk course, students learn technical problem-solving skills so they can solve end-user IT problems. We developed labs deployed through Docker sessions with questions and recipe-type instructions for the students to solve technical issues. We utilize the Linux Bash history file to auto-grade the student's work to ensure they execute all the recipe commands. Each student has a different user id shown in the bash prompt stored in the history file that ensured we had derivatives for each student. If a student tries to submit a file with a different prompt from the submitting student, the grader detects and rejects it.

B. *Networking Admin Course Auto-graders*

Like the helpdesk course, the networking admin course teaches the student the core competency around network tools. We developed labs utilizing Wireshark Packet Capture (PCAP) files. The students perform a network scan and then answer questions about their scan in a Google Form. They upload their PCAP file to the grader, and the grader compares the data to the form values utilizing Scapy [9]. Scapy allows the grader to parse the PCAP file and ensure that the student's form submission matches the data in their scanned file. We provide no two students submit duplicate PCAP files in two ways; the first is to ensure the timestamp is recent (within one hour of submission). We stored a CRC code for previous submissions and rejected secondary submissions that match.

VI. Computer Science Course Auto-graders

This section will drill into different categories of lab auto-graders we have developed for upper-level Computer Science courses. The upper-level computer science courses often include theory and high-level algorithms and protocols. The students need to apply these algorithms and protocols in programming labs to reinforce the ideas from the lectures.

A. *Operating Systems Auto-graders*

In an operating system course, students learn how operating systems manage limited hardware resources so that many application programs can run simultaneously. We developed auto-graders that allowed the students to explore the data structures and algorithms used to manage physical memory, virtual memory, hard disks, and the central processing unit (CPU).

B. *Networking Programming Course Auto-graders*

Students learn about the Open Systems Interconnection model (OSI) model and Transmission Control Protocol/Internet Protocol (TCP/IP) layers in a networking course. The students write programs in Python that utilize TCP/IP services that talk to a cloud application.

VII. Cybersecurity Science Course Auto-graders

This section will drill into different categories of lab auto-graders we have developed for Cybersecurity courses.

A. *Information Security Auto-graders*

Students learn about threat modeling, security policy models, access control policies, and reference monitors in an information security course. We developed a set of auto-graded reference monitor labs. The student implements a reference monitor in each lab that implements different access control policies and security policies.

B. *Secure Programming Auto-graders*

In a secure programming course, students learn how to develop code free of vulnerabilities. The perspective in a secure programming course comes from the concept that the code is a white box. The students have full visibility of the source code as they perform labs to secure the code. We developed labs where students are provided code with vulnerabilities. Docker auto-graders are provided that exploit the vulnerabilities. Students need to improve the code and submit a version without the original vulnerability to receive credit.

C. *Penetration Testing Auto-graders*

In a penetration course, students think about security from a different perspective. The perspective in a penetration testing course comes from the concept that the code is a black box. The students do not have visibility into the source code they are trying to penetrate in the labs. We developed labs where students are provided a signature for a code library with vulnerabilities. Docker auto-graders are equipped to execute the students' code and determine if they found a weakness.

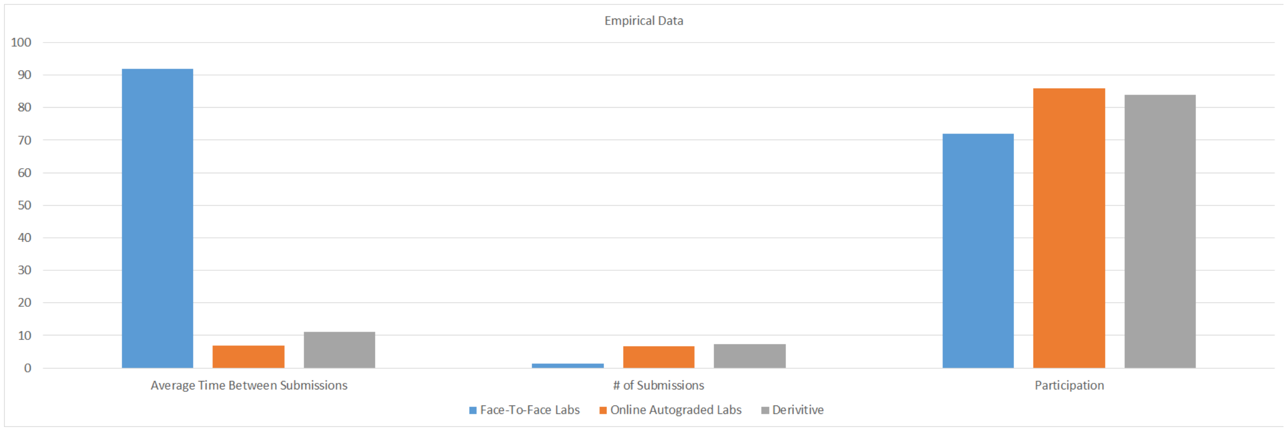

Figure 2. Empirical Data

VIII. EMPIRICAL DATA

In this section, we examine the data we gathered from three sections of a database course. There were three questions we wanted to answer about our use of auto-graders in the cybersecurity curriculum:

- Do the auto-graders help online student progress quicker through a lab
- Do the auto-graders help increase participation at the undergraduate level in labs
- Do the non-fungible assessments help students by facilitating peer discussion?

We choose the database course because every lab had a non-fungible version so that each student was working on a unique problem in the lab. The original face-to-face section used manual graded lab submissions without non-fungible derivatives, one online section used auto-graded fungible labs, and one section used non-fungible labs. The students in the face-to-face section had a reverse classroom where they worked individually on labs during class time, and the instructor would answer questions as they ran into problems. In the non-fungible labs section, a discussion forum was provided for students to complete the lab.

Figure 2 shows a summary of the data we used to answer the questions. The average time between submissions was reduced significantly for the two sections that utilized auto-graders. The number of submissions was increased for the two sections that used auto-graders. Lastly, the participation rate was raised for the two sections that used auto-graders.

The three research statements' answer was a strong yes to the first two and a weaker yes to the third question. The auto-graders helped online student progress quicker through the lab by shortening the time between submissions. The increase in submissions with the auto-graded assessments shows an increase in participation. In our small study, the auto-graders helped increase involvement at the undergraduate level in both versions of the labs. Lastly, we believe the non-fungible assessment helped students by facilitating peer discussion. The participation rate was a little lower for the derivative version of the labs. Still, we felt it was close enough to the non-derivate lab to show progress in learning since students were performing unique work, and the increased student communication help to facilitate that progress.

IX. GRANULARITY OF ASSIGNMENT AND PARTICIPATION

In this section, we examine the participation rates in the self-paced online courses. Our goal is to increase student participation in technology courses while increasing the integrity of the assessments. We offered three core technology courses through the edX [10] platform on programming, networking, and operating systems. TABLE IV shows the enrollment and completion data from the first year. The number of auditors is the number of learners who signed up for the free version of the course in the table. The free version offered the recorded lectures, readings, and discussion forums. The verified users pay a small fee for access to the assessments, course certificate, and undergraduate credits. Based on our first year of data in three foundations courses, less than half the students who took the initiative to sign up as a verified learned completed enough of the assessments to earn the course certificate and the undergraduate credits. The courses are still available for the students to complete the work, but we do not expect students who lost motivation to return.

TABLE IV. ONE YEAR STUDENT PARTICIPATION RATES

Course	*Auditors*	*Verified*	*Completed*
Programming	*39,657*	*698*	*241*
Networking	*17,671*	*743*	*273*
OS	*16,945*	*721*	*264*

In a recent experiment, we offered programming courses on the Coursera [11]. These offerings were targeted at an audience with less technical experience. In our development, we wanted to create non-fungible programming assignments and opted for smaller grained tasks than we offered in our early work. For example, if we assess a lesson on iteration, the student is given a job that has them write a loop until a condition is seen in each student's different code. These offerings are just a few months old, but preliminary data shows a much higher completion rate by the learners in these assigned.

X. CONCLUSIONS AND FUTURE WORK

Our research demonstrates that the use of our E-learning auto-graded assignments improves participation in the technology course assessments. We also show that using our technique of creating non-fungible versions of the lab for each student can increase communication between students and improve learning. Our future work will continue to develop finer-grained versions of assessments that allow for labs in the advanced courses that randomize the unit test data and provide for non-fungible assessments per student. We will also gather more empirical evidence in the future to show how the auto-graders improve the learning experience for online E-Learners.

REFERENCES

[1] S. Morgan, "Cybersecurity Talent Crunch To Create 3.5 Million Unfilled Jobs Globally By 2021," Cybercrime Magazine, 24 October 2019. [Online]. Available: https://cybersecurityventures.com/jobs/. [Accessed 8 April 2020].

[2] NYU Tandon, "NYU Cyber Fellows," 2020. [Online]. Available: https://engineering.nyu.edu/academics/programs/cybersecurity-ms-online/nyu-cyber-fellows. [Accessed 8 April 2020].

[3] Georgia Institute of Technology, "Online Master of Science in Cybersecurity," 2019. [Online]. Available: https://info.pe.gatech.edu/oms-cybersecurity/?utm_source=cpc-

google&utm_medium=paid&utm_campaign=omsc-search-converge-top5&gclid=CjwKCAjw7LX0BRBiEiwA__gNw2xT3-grxlfiyfYGdGDGIpZZSkf6_bIttgoipp830ue3le5MByUu0hoCGtoQAvD_BwE. [Accessed 8 April 2020].

[4] Fisher College, "Find the world at Fisher," 2020. [Online]. Available: www.fisher.edu. [Accessed 8 April 2020].

[5] J. D. Ullman, "Gradiance online accelerated learning," in *Proceedings of the 28th Australasian Conference on Computer Science*, Newcastle, Australia, 2005.

[6] McGraw Hill, "SIMnet," 2020. [Online]. Available: https://www.mheducation.com/highered/simnet.html. [Accessed 6 April 2020].

[7] Gradescope, "Grade All Coursework in Half the Time," 2019. [Online]. Available: https://www.gradescope.com/. [Accessed 6 April 2020].

[8] Docker Inc, "Debug your app, not your environment," 2020. [Online]. Available: https://www.docker.com/. [Accessed 6 April 2020].

[9] Philippe Biondi and the Scapy community, 2021. [Online]. Available: https://scapy.net/. [Accessed 3 February 2021].

[10] edX Inc., 2021. [Online]. Available: www.edx.org.

[11] Corsera Inc., 2021. [Online]. Available: https://www.coursera.org/. [Accessed 15 Febraury 2021].