# A small network simulator for learning routing fundamentals

Miguel Bazdresch
Electrical, Computer & Telecom Engineering Technology Dept.
Rochester Institute of Technology
Rochester, NY

***Abstract*—In this Innovative Practice Full Paper, we present a novel network simulator designed specifically for beginner students (at undergraduate or graduate level) learning the fundamental principles of data communications at the data link and network layers. The simulator can also be used to strengthen programming skills. Simulation is one of the main tools in computer network engineering, both in education and in industry. One of its benefits in education is that it allows every student the freedom to explore, implement and analyze different networks, without requiring the use of physical laboratory infrastructure. This hands-on approach usually complements and supports a lecture course that covers networking theory. Many available network simulators, however, are oriented towards research and industry applications and their complexity can make them difficult to use for beginner students. Furthermore, they do not readily expose the layer-2 and layer-3 algorithms that perform some of the fundamental network functions, and it can be difficult for students to study and modify them. In contrast, the proposed network simulator is tailored to teaching fundamental networking algorithms and network analysis. Some interesting features of the simulator are its very small size and its capability to simulate data flow on arbitrary graphs. The simulator is designed so that students can easily modify it to add to its functionality; it is also fully documented and released under an open-source license.**

## I. Introduction

Computer networking is a required topic in many electrical, computer and software engineering, computer science, and information technology programs, at both undergraduate and graduate levels. In the past few decades, networking technology has evolved significantly, and data communications have become part of everyday life in many societies. With this progress, network protocols and equipment have also become significantly more complex. As a result, teaching this topic has become more challenging; in order to succeed in industrial or academic positions in networking, students need to understand more and more complex concepts.

Engineering educators have reported several different teaching approaches and methodologies to address this complexity and to help students not only cope with it, but to acquire meaningful and lasting knowledge. Active learning and problem-based approaches that involve laboratory work are widely recognized as effective engineering education techniques [1]. The effectiveness of these methods has been found to extend to networking. For example, in [2] graduate students are immersed in realistic network design scenarios; in [3], collaborative learning is used to improve understanding of the Transmission Control Protocol (TCP). The effectiveness of problem-based learning applied to Internet addressing is reported in [4]. Laboratory activities appropriate for an early course covering the physical, data link and networking layers are presented in [5], while [6] proposes that students implement a data link layer in the context of an otherwise operational network. Further proposed laboratory exercises are presented in [7]. An intriguing approach is presented in [8], where students spend one month reproducing results found in the networking literature. Interestingly, some of these techniques have been successfully extended to high-school classes [9].

However, practical and laboratory activities in networking present several challenges related to infrastructure. Setting up a network is time consuming, and the equipment occupies significant amounts of space. When a problem occurs, it can be difficult to trace the cause (which could be bad cabling, software bugs, or equipment malfunction, among others). Creating realistic usage scenarios and measuring the network's performance can be challenging. In programs with a large number of students, it may be difficult or impossible to offer equipment access to all of them. As a result, simulation, emulation and virtualization have become important tools in teaching networking. An additional attraction of these tools is that they are widely used in industry and research, because they allow evaluating a particular network design or technology before costly and uncertain deployment.

At the more complex end of the spectrum, a virtualized network consists of actual virtual machines, usually running the Linux operating system along with its full network software stack [10]. The benefit is that current virtualization techniques allow the creation and interconnection of tens to hundreds of network nodes on one physical computer [11]. Several educational tools have been designed around virtual networks [12], [13].

A subset of virtual networks, emulated networks attempt to mimic more closely the actual behavior of routers and other network nodes, while reproducing realistic traffic and other usage scenarios. An emulated network can be connected to an actual network. As an example, Mahimahi [14] can record real HTTP traffic, and then replay it in an emulated network, allowing evaluation of Web multiplexing protocols.

Less complex than virtualization, network simulation consists of replacing an actual physical network with a computer program that simulates the network [15]. In a simulator,

978-1-5386-1174-6/18/$31.00 ©2018 IEEE

traffic and nodes are replaced by simple mathematical models. Simulation has been shown to engage students and enhance their understanding of computer networks [16]. Examples include the NS simulators [17], [18], and Packet Tracer [19].

It is well established that complementing a lecture course on networking with experiments involving virtualization, emulation and simulation result in enhanced learning; furthermore, these tools give every student the freedom to explore, implement and analyze different networks, without requiring the use of physical laboratory infrastructure. Many of the available tools, however, are oriented towards research and industry use, and their complexity can make them difficult to use for beginner students. Another consideration is that most of the tools available are focused on the implementation of complex networks and advanced algorithms; they are not designed for exposing the fundamental networking concepts occurring in the lower layers.

In this sense, it can be argued that there is a missing simulation tool, one that is oriented towards the beginner student, who is not yet an expert programmer or network designer; a tool that exposes the fundamental networking algorithms in a general fashion, instead of focusing on advanced applications based on established protocols. In this paper, we present a new network simulator, called TinyNS, that is tailored to teaching the fundamental networking algorithms to students. Some of its main features are:

1) It is written in just 130 lines of simple Python code, so that even beginner students can study and understand the simulator in its entirety.
2) It simulates data flow on arbitrary graphs.
3) It is designed so that students can easily modify it and add functionality to it, such as different traffic patterns, packet types, queuing priorities or routing algorithms.
4) It is fully documented and released as open-source software[1].

In this sense, the simulator follows the tradition of old networking textbooks, such as [20], rather than more moderns ones, such as [21]. However, the proposed simulator should be of interest to any educator interested in a simple simulation tool to teach certain fundamental networking concepts in a generic way (that is, without reference to specific protocols or standards). To be sure, students should quickly move on to more powerful simulators/emulators; however, a first network simulation experience with TinyNS can be a solid foundation that students can build on.

This paper is organized as follows. In Section II, we present a description of the learning goals that can be supported with TinyNS. Section III presents a detailed description of the simulator implementation. This is followed by several networking problems that have been assigned to our students to solve using the proposed simulator. Section IV presents examples of the results that our students have been able to obtain and how these demonstrate that we have made progress towards our stated learning goals. We end the paper by presenting our conclusions and future plans.

[1]Code and documentation for TinyNS, released under the MIT open-source license, can be found in https://www.github.com/mbaz/TinyNS.

## II. Learning goals

The TinyNS simulator is intended to support the following student learning objectives:

1) Understand, and experiment with, the basic concepts and algorithms behind routing in data networks, including switch ports, switch buffers, and dynamic, heuristic construction of routing tables.
2) Measure network performance using some classic indicators, such as packet delay, congestion, and throughput, and how these are affected by network topology, traffic intensity, and switch buffer capacity.
3) Collect and analyze the relevant data, drawing appropriate conclusions, and communicating them effectively in writing.
4) Understand the key concepts that underlie a network simulator and its internals, to become more proficient when using more advanced tools in later courses.
5) Strengthen Python programming skills, by modifying an existing program in order to augment its capabilities toward a specific goal.

We consider it to be an important element of this approach that it is not tied to specific technologies or protocols. We believe that students that learn the fundamentals in a generic way might be better prepared to apply and adapt them to the myriad technological variations they will face in their careers.

Feisel [1] identifies 13 fundamental objectives of engineering instructional laboratories. Using a network simulator such as TinyNS contributes to the following objectives:

- *Modeling,* by providing a first experience in theoretical models of network elements.
- *Experimenting,* by having the student figure out how to generate and collect the required indicators.
- *Data analysis,* by drawing conclusions about network performance from the large amount of indicators collected.
- *Learning from failure,* since the task presented to them is not trivial and several attempts will likely be necessary.
- *Communication,* by reporting their analysis, results and conclusions in an effective manner.

## III. Simulator implementation

The objective of the implementation is to provide a minimal framework for node interconnection, packet flow and routing, in as few lines of clear, documented Python code as feasible. The programming language Python was selected because of its popularity in both academia and industry, and because of its high expressiveness. To keep the simulator small, anything that is not strictly required is not included; the purpose is to allow students to focus on the networking fundamentals, instead of spending their time learning to use a complex tool. However, the simulator must be extensible enough to allow students to implement, test and evaluate their own additions

and modifications in a variety of topologies and under different traffic conditions.

The main sacrifice that was made in terms of simulation power was the ability to do a discrete-event simulation. However, we have found that, even with this limitation, meaningful simulations can be set up and run. The core ideas behind TinyNS are:

1) There are two kinds of nodes: edge nodes and switches. Edge nodes generate and consume packets; switches only forward packets.
2) Edge nodes have only one port; switches can have any number of ports.
3) Nodes are connected by connecting their ports. Ports are bidirectional.
4) The simulation consists of a repetition of these three steps:
   a) Edge nodes are allowed to transmit a packet.
   b) Switches forward one packet stored in their memory, according to their routing directives.
   c) Edge nodes are allowed to receive one packet.

A minimal packet is implemented as follows:

```
class Packet:
    def __init__(self, s_addr, d_addr, payload):
        self.s_addr = s_addr
        self.d_addr = d_addr
        self.payload = payload
```

where the arguments are the source and destination addresses and a payload. Students typically add other fields, such as a time-to-live counter, a message type indicator, or a sequence number.

An edge node is implemented as follows:

```
class EdgeNode:
    def __init__(self, address, d_addr_set, tx_prob):
        self.address = address
        self.d_addr_set = d_addr_set
        self.tx_prob = tx_prob
        self.memory = []
```

The edge node has an address, a set of addresses that can be selected as destination when a packet is created, and a probability that a packet will be generated and transmitted during a simulation iteration. The memory acts as an infinite buffer to store arriving packets. Examples of capabilities that students could add to this node are an initial TTL value, event logging (such as number of received packets), a sequence number counter, a finite memory, and internal state.

Edge nodes have also certain auxiliary functions: `transmit(self, time)` returns a new packet with the transmit probability specified by the node, and `receive(self, time)` takes a user-defined action (such as logging) if there is a packet in the node's buffer and removes the packet. The argument `time` is provided by the controller, explained below.

Finally, switches are implemented with this code:

```
class SwitchNode:
    def __init__(self, ID, ports, rt):
        self.ID = ID
        self.ports = ports
        self.rt = rt
        self.memory = []
```

The arguments are the switch's unique ID, a list of its ports, and a routing table, specified as a dictionary. The switch also has an infinite buffer. Packets are forwarded with function `forward(self, time)`, defined as follows:

```
def forward(self, time):
    if len(self.memory) > 0:
        pkt = self.memory.pop()
        outport = None
        for i in self.rt:
            if i == pkt.d_addr:
                outport = self.rt[i]
        return pkt, outport
    else:
        return None, None
```

If there is a packet in the buffer, then the packet is removed from the buffer and its destination address is obtained. The routing table is consulted to find the port the packet should be forwarded to. If a packet is forwarded, then a tuple `(packet, outputport)` is returned to the controller. If the buffer is empty, then a tuple `(None, None)` is returned.

The controller is the code that handles all the nodes and packets. The simulation executes what is called a "round robin scheduler", which calls each node in order and asks it to perform a function. The key function is called `run`. It first lets edge nodes transmit; then, it calls the switches to forward one packet; finally, it lets the edge nodes receive. This is executed for the desired number of iterations. This function uses a node list `nlist` and a connection list `clist` to know which nodes to call, and in which node's memory to write the packets as they flow through the network:

```
def run(nlist, clist, iterations):
  for i in range(iterations):
    # Edge nodes transmit
    for n in nlist.nlist:
      if n.isedge():
        pkt = n.transmit(i)
        if pkt != None:
          nextnode = findnextnode(clist, n, '1')
          nextnode.memory.insert(0,pkt)
    # Switch nodes forward
    for n in nlist.nlist:
      if n.isswitch():
        pkt, port = n.forward(i)
        if pkt != None:
          nextnode = findnextnode(clist, n, port)
          nextnode.memory.insert(0,pkt)
    # Edge nodes receive
    for n in nlist.nlist:
      if n.isedge():
        n.receive(i)
  print("Done!")
```

Several auxiliary functions can be seen in the code, and these do the obvious thing: `n.isedge()` and `n.isswitch()` return `true` if `n` is an edge or a switch, respectively; `findnextnode(cl,n,p)` returns the node in the connection list `cl` connected to node `n` through port `p`.

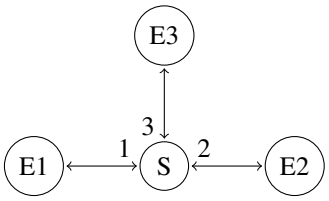
A network is created by instantiating nodes and interconnecting them. A list of nodes is defined by:

```
class NodeList:
    def __init__(self):
        self.nodelist = []

    def append_node(self, node):
        self.nodelist.append(node)
```

while a connection list is defined by:

```
class ConnectionList:
  def __init__(self):
    self.connlist = []

  def connect(self, node1, port1, node2, port2):
    self.connlist.append((node1, port1, node2, port2))
```

As an example, let us simulate the following network:




We want each node to transmit a different, fixed message; each node can transmit to any other node; the transmit probability is 0.1; and we wish to run the simulation for ten iterations. We follow these steps:

1) Create the node list and connection list.
```
nl = NodeList()
cl = ConnectionList()
```
2) Create the edge nodes and append them to the node list.
```
e1 = EdgeNode('E1',['E2','E3'],0.1,'Hello')
e2 = EdgeNode('E2',['E1','E3'],0.1,'Bye')
e3 = EdgeNode('E3',['E1','E2'],0.1,'Me too!')
nl.append_node(e1)
nl.append_node(e2)
nl.append_node(e3)
```
3) Create the switch and append it to the node list.
```
s = SwitchNode('S',['1','2','3'], \
    dict([('E1','1'),('E2','2'),('E3','3')]))
nl.append_node(s)
```
4) Connect the nodes:
```
cl.connect(e1,'1',s,'1')
cl.connect(e2,'1',s,'2')
cl.connect(e3,'1',s,'3')
```
5) Run the simulation for the desired number of steps:
```
run(nl,cl,10)
```

Note the following limitations to the proposed approach:

1) Each network node processes at most one packet per iteration.
2) All packets are the same size, in the sense that they take the same time to arrive to their destination.

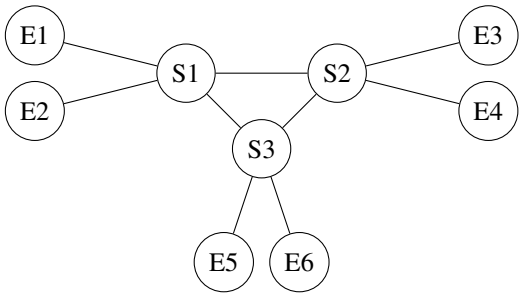
However, as will be seen in the following sections, this simple approach still enables the simulation of some interesting and illustrative networking scenarios.

## IV. Example assignments

In this section, we provide a detailed description of one student assignment, and shorter descriptions of other illustrative assignments. These have been used in a graduate course on telecommunication network fundamentals.

### A. Detailed assignment description

Students are asked to implement the following network topology:




Their task is to modify the switches to implement the following simple routing table heuristic algorithm [22]: At first, set the routing tables of each switch at random. Every time a switch receives a packet, it notes the source address and the port it is coming from, and modifies the routing table so that packets whose destination is that address are forwarded through that port. After sufficient traffic has flowed through the network, the routing tables of all three nodes should be optimal.

In order to focus on the routing table construction problem, traffic is assumed random (so that each node generates packets for all other nodes), and of low enough intensity not to cause bottlenecks.

To solve this problem, students must, first, understand the routing table algorithm and use it to determine what the correct routing tables should be. Then, using their Python programming skills, they must modify the forward function of the `SwitchNode` class to implement the algorithm. In turn, this requires providing the switch with information about the port a given packet was received through. There are several ways to implement this.

While the changes, measured in lines of code, are not great, this exercise allows students to:

1) Deepen their knowledge about routing tables and how they are constructed dynamically.
2) Practice their verification skills: they must be able to prove their implementation is correct.
3) Practice their programming skills.

A more complex addition to this assignment consists in modifying the network after it has been running for some time (for example, edge node e5 is disconnected from s3 and connected to s2), and verifying that, after a period of erroneous packet delivery, the switches are able to adjust their routing tables without any human intervention.

### B. Further assignment suggestions

There is a large variety of possible simulation exercises. We present a list of those that we have used in our courses:

- Measure average packet delay under different traffic loads. It can be particularly interesting to observe the effect of a single large-traffic node on the delay of unrelated low-traffic nodes. Try to correlate the concept of congestion with the traffic patterns observed in simulation.

- Measure the proportion of dropped packets in a network, for a given switch buffer size. This requires converting the infinite switch buffer memory to a finite memory.
- Measure the performance of flood routing, under different traffic loads. A more complex version of this exercise requires switches to drop packets that they have already seen, in order to avoid saturating the network.
- Implement a stop-and-wait protocol, and measure throughput under different traffic conditions. This requires implementing a mechanism for classifying packets into data and acknowledgment packets. The edge nodes must also be modified not to transmit new packets until an acknowledgment has been received. If combined with a non-zero probability of packet loss, the need for sequence numbers in packets will arise naturally.
- Add error detection through a frame-check sequence in the packet. A possible implementation would be that a switch has a certain probability of introducing an error in a packet when it is forwarded. This can be more easily simulated by just adding a packet header field that indicates whether a data error is present or not.
- Change the probability distribution of packet generation (for instance, using exponential and Rayleigh distributions) and observe the effect on network performance indicators. Research what distribution better models reality.

Of course, each of these exercises can be specified under any of a large number of network topologies (ring, bus, mesh, star, etc.) Also, most of them require supplementing the simulator's capabilities by adding new packet header fields, adding switching intelligence, and adding logging and reporting facilities.

More advanced assignments may include modifying the switches to add queueing priorities and QoS, to add packet fragmentation and adaptation layers, and to randomize packet transmission time.

To complement the practical exercises, students are asked to prepare a report where they explain the theoretical fundamentals of the scenarios they will simulate, and provide an analysis of their results. Students are also required to submit their code.

## V. Student results

Our experience using the TinyNS simulator in a graduate course indicates that a majority of students are able to complete their assignments successfully. However, the simulator is sufficiently complex that those students without a strong programming background struggle, and sometimes are unable to obtain results and complete their assignments, even after spending significant amounts of time. Our suggestion is that students taking a course that adopts the proposed approach either be required to have at least intermediate programming skills, or to take a programming course concurrently.

In the following two subsections, we present two detailed assignments that have been used in our courses. We present the results and analyses that students were able to obtain and produce using TinyNS. One assignment is on average packet delay, and the other on dynamic routing table construction.

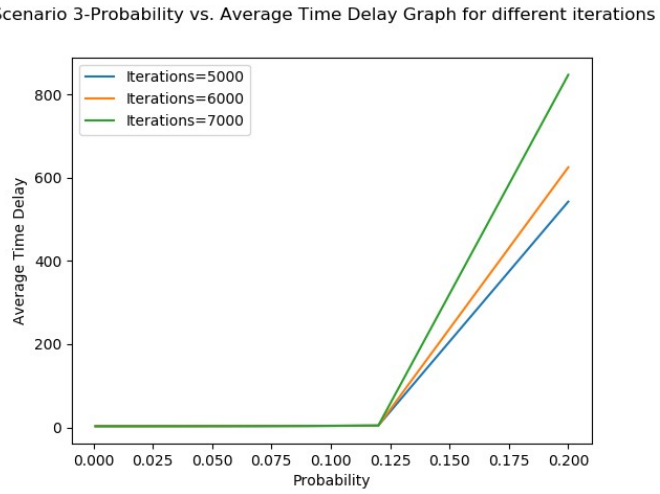
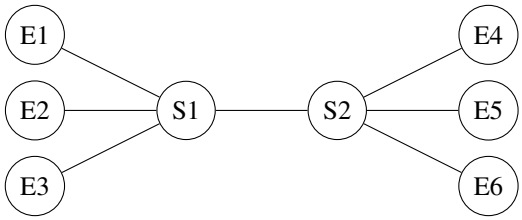



Fig. 1. Student-obtained plot of the average time delay for different traffic conditions, measured at different times. Each of the six edge nodes has the specified probability of transmitting a packet in a given simulator iteration.

### A. Student results on average packet delay

In this assignment, students were given a network topology where two switches are used to interconnect six edge nodes, as shown in the figure below.




The student's task was to measure the average time a packet took to reach its destination, under different traffic conditions. Network traffic is increased by increasing the probability that a given node transmits a packet in a given simulator iteration. Since the switches have limited transmission rate (in fact, only one packet per iteration), the expected result is that the nodes will see minimum delay for low transmission probability, and that the delay will start increasing as the traffic increases.

The primary educational objective of this assignment is for students to grasp, experimentally, the relationship between packet delay, network traffic, and switch buffer size. Students need to think about the indicators they need to measure, and how to process the data they gathered. They also confront their initial hypothesis on network behavior, and what they observe in practice.

Students obtained the plot shown in Fig. (1). While the behavior does correspond to expectations, there were a few surprises. For example, the sudden (instead of gradual) onset of congestion, and the exponential increase in delay. Also unexpected for many students is that, for fixed probability, the delay increases over time; in other words, the memory usage of the switches increases without bound.

TABLE I
ROUTING TABLE OF SWITCH S1 AT THE START OF SIMULATION.

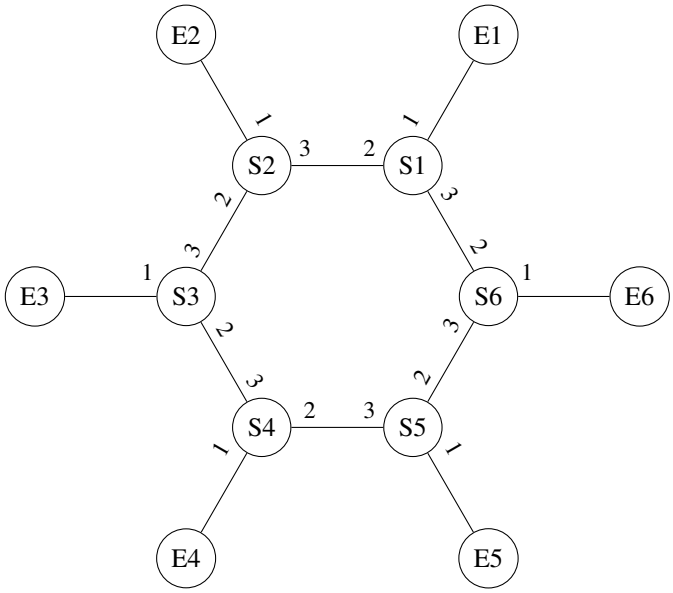
| Destination | Port |
|---|---|
| E1 | 1 |
| E2 | 3 |
| E3 | 3 |
| E4 | 3 |
| E5 | 3 |
| E6 | 3 |

TABLE II
ROUTING TABLE OF SWITCH S1 AFTER SIMULATION ENDS.

| Destination | Port |
|---|---|
| E1 | 1 |
| E2 | 2 |
| E3 | 2 |
| E4 | 3 |
| E5 | 3 |
| E6 | 3 |

### *B. Student results on routing table construction*

In this exercise, students are asked to implement the routing table construction heuristic detailed in Section IV, specifically for the ring topology shown below. Switch ports are always labeled '1' towards the edge node, '2' in the counter-clockwise direction, and '3' in the clockwise direction.




Initially, each switch had a routing table configured such that packets would always flow "clockwise". To illustrate, the initial routing table of switch S1 is shown in Table (I). These routing tables are correct in the sense that all packets will eventually arrive at their destination. However, they are suboptimal; for example, a packet from E1 to E2 would traverse six switches (E1 $\rightarrow$ S1 $\rightarrow$ S6 $\rightarrow$ S5 $\rightarrow$ S4 $\rightarrow$ S3 $\rightarrow$ S2 $\rightarrow$ E2) instead of only two (E1 $\rightarrow$ S1 $\rightarrow$ S2 $\rightarrow$ E2).

The learning objective of this exercise is for students to implement and verify the functionality of one of the most fundamental networking algorithms, and to realize that switches can find optimal routes even if they are unaware of the network topology. Students also need to come up with a way to gather and analyze the data they need, and to configure their network traffic in such a way that all switches observe the packets they need in order to update their tables appropriately.

Ultimately, students had to determine if the simple routing table construction heuristic is capable of finding the optimal routes. Students ran their simulation for a few hundred iterations, after which the routing tables were examined. To illustrate a part of the results, the final routing table for switch S1 is presented in Table (II). From this evidence, it is possible to verify that all routes are chosen to minimize the number of hops.

## VI. CONCLUSIONS

In this paper, we have presented a small and simple open-source computer network simulator, as well as the learning objectives that it can support. This simulator is suitable to be used by students who are beginners in both networking and programming. It focuses on supporting learning of some of the basic ideas and concepts behind networking, as opposed to most of the available tools, which are designed for advanced use cases, present a high-level view of the network while hiding some of the internal mechanisms, and rely on specific technologies and protocols. We described the simulator's architecture, and presented some possible student exercises. We also showcased some interesting student results.

We believe that there is a need for simple, technology-agnostic tools for teaching and learning networking. TinyNS is a first step, but there are many other opportunities and areas for improvement. We are working on a hardware-based networking system, that can be considered a real-world version of TinyNS. Using low-cost and small system-on-chip hardware development boards, we believe it is possible to create a small, simple network that extends TinyNS's capabilities from the network layer to the data link and physical layers, while also providing the tactile, real-world experience that is missing in simulation exercises.

## REFERENCES


[1] L. D. Feisel and A. J. Rosa, "The Role of the Laboratory in Undergraduate Engineering Education," *J. of Engineering Education*, vol. 94, no. 1, pp. 121–130, Jan. 2005.

[2] N. Linge and D. Parsons, "Problem-based learning as an effective tool for teaching computer network design," *IEEE Trans. on Education*, vol. 49, no. 1, pp. 5–10, Feb. 2006.

[3] D. Hernandez, J. I. Asensio, and Y. A. Dimitriadis, "Collaborative Learning Strategies and Scenario-based Activities for Understanding Network Protocols," in *Proc. of the 36th Annual Frontiers in Education Conference*, Oct. 2006, pp. 19–24.

[4] A. Jevremovic, G. Shimic, M. Veinovic, and N. Ristic, "IP Addressing: Problem-Based Learning Approach on Computer Networks," *IEEE Trans. on Learning Technologies*, vol. 10, no. 3, pp. 367–378, Jul. 2017.

[5] D. R. Surma, "Lab exercises and learning activities for courses in computer networks," in *Proc. of the 33rd Annual Frontiers in Education*, vol. 1, Nov. 2003, pp. T2C–21–5 Vol.1.

[6] M. W. El-Kharashi, G. Darling, B. Marykuca, and G. C. Shoja, "Understanding and implementing computer network protocols through a lab project," *IEEE Trans. on Education*, vol. 45, no. 3, pp. 276–284, Aug. 2002.

[7] N. I. Sarkar, "Teaching computer networking fundamentals using practical laboratory exercises," *IEEE Trans. on Education*, vol. 49, no. 2, pp. 285–291, May 2006.

[8] L. Yan and N. McKeown, "Learning Networking by Reproducing Research Results," *SIGCOMM Comput. Commun. Rev.*, vol. 47, no. 2, pp. 19–26, May 2017.

[9] E. Shifroni and D. Ginat, "Simulation Game for Teaching Communications Protocols," in *Proc. of the 28th SIGCSE Technical Symposium on Computer Science Education*, ser. SIGCSE '97. ACM, 1997, pp. 184–188.

[10] A. Gaspar, S. Langevin, W. Armitage, and M. Rideout, "Enabling new pedagogies in operating systems and networking courses with state of the art open source kernel and virtualization technologies," *J. of Computing Sciences in Colleges*, vol. 23, no. 5, pp. 189–198, May 2008.

[11] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: Rapid prototyping for software-defined networks," in *Proc. of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*. ACM, 2010, pp. 19:1–19:6.

[12] A. Ruiz-Martinez, F. Pereniguez-Garcia, R. Marin-Lopez, P. M. Ruiz-Martinez, and A. F. Skarmeta-Gomez, "Teaching Advanced Concepts in Computer Networks: VNUML-UM Virtualization Tool," *IEEE Trans. on Learning Technologies*, vol. 6, no. 1, pp. 85–96, Jan. 2013.

[13] M. Wannous and H. Nakano, "NVLab, a Networking Virtual Web-Based Laboratory that Implements Virtualization and Virtual Network Computing Technologies," *IEEE Trans. on Learning Technologies*, vol. 3, no. 2, pp. 129–138, Apr. 2010.

[14] R. Netravali, A. Sivaraman, S. Das, A. Goyal, K. Winstein, J. Mickens, and H. Balakrishnan, "Mahimahi: Accurate record-and-replay for HTTP," in *USENIX Annual Technical Conference*, 2015, pp. 417–429.

[15] N. Al-Holou, K. K. Booth, and E. Yaprak, "Using computer network simulation tools as supplements to computer network curriculum," in *Proc. of the 30th Annual Frontiers in Education Conference*, vol. 2, 2000, pp. S2C/13–S2C/16 vol.2.

[16] C. Goldstein, S. Leisten, K. Stark, and A. Tickle, "Using a network simulation tool to engage students in active learning enhances their understanding of complex data communications concepts," in *Proc. of the 7th Australasian Conference on Computing Education*. Australian Computer Society, 2005, pp. 223–228.

[17] The ns-2 network simulator. The Information Sciences Institute at the University of Southern Carolina. [Online]. Available: https://www.isi.edu/nsnam/ns/

[18] The ns-3 network simulator. The NS-3 Project. [Online]. Available: https://www.nsnam.org/

[19] (2010) Cisco packet tracer data sheet. Cisco Networking Academy. [Online]. Available: https://www.cisco.com/c/dam/en_us/training-events/netacad/course_catalog/docs/Cisco_PacketTracer_DS.pdf

[20] L. Kleinrock, *Communication Nets; Stochastic Message Flow and Delay*. New York: McGraw-Hill, 1964.

[21] J. Kurose and K. Ross, *Computer Networking: A Top-Down Approach*, 7th ed. Essex, England: Pearson, 2016.

[22] P. Baran, "On Distributed Communications Networks," *IEEE Trans. on Communications Systems*, vol. 12, no. 1, pp. 1–9, Mar. 1964.