Improving a Computer Networks Course Using the Partov Simulation Engine

Behnam Momeni and Mehdi Kharrazi

***Abstract*—Computer networks courses are hard to teach as there are many details in the protocols and techniques involved that are difficult to grasp. Employing programming assignments as part of the course helps students to obtain a better understanding and gain further insight into the theoretical lectures. In this paper, the Partov simulation engine and experience using this engine in a computer networks course are discussed. Since 2009, various programming assignments based on the Partov system have been set to help students in their learning process. Student feedback has been very good; this has been quantified in two surveys in which a majority of students expressed their satisfaction with this approach.**

***Index Terms*—Computer networks, network simulator, Partov, teaching tools.**

I. Introduction

TEACHING an engineering course is a hard task, especially if the only tools available are presentation slides and the teacher's eloquence. The problem is that although students listen and understand the underlying theory, they rarely grasp its importance and how it applies to real-world scenarios. This problem becomes immensely important in computer networks courses. For example, the Spanning Tree Protocol (STP) [1] is, on the surface, a straightforward protocol that creates a spanning tree among a set of devices at the link layer. The algorithm is so simple that, given a small network layout, the student can execute it on paper and draw the spanning tree. However, when deployed in a real network environment, such protocols can become quite complex, as instead of their being a single entity with a global view of the network (as in the case of the student), each node must execute the algorithm with only local knowledge and with no global view of the network. In this example, this change of perspective between the global and the local view is immensely important to help students grasp the fact that in computer networks, a number of elements interact to execute an algorithm, without any central coordinated mechanism.

There are a wide range of approaches to mitigating this problem and helping students obtain a deeper appreciation of networking concepts. At one extreme, students are provided with actual network devices (i.e., routers, switches, etc.) and are assigned tasks that would require them to employ and experiment with these devices in various network layouts. In such cases, each student or group of students must have dedicated devices, which requires hardware-laden laboratories. Such laboratories [2], [3] are costly, bias the educational process toward a specific vendor, require the students' physical presence in the lab, and limit their understanding of the underlying protocols. While this would be an ideal scenario for students of a network administration course, it does not provide students with the required in-depth knowledge of networking concepts. For example, students may learn how to deploy STP between switches in a network at the lab, perhaps biased by the hardware vendor interface, but their knowledge of the STP operation will remain abstract and only at an algorithmic level.

At the other extreme, students use a simulator software to simulate different scenarios [4]–[6], which, unlike the hardware-based approach, requires no dedicated network device and allows students to work on the assignments outside of class time by executing the simulator on their personal computers. However, this approach does not allow students to go beyond the predicted scenarios/features of the simulator software, limiting the device experience they gain to that programmed into the simulated world. For example, user traffic is generated based on a model whose traffic resembles real user traffic passing through other virtual nodes (i.e., routers, switches, etc.) and which allows various simulation models (i.e., packet loss, RTT variations, etc.) to be applied to the traffic. In short, there is a lack of realness inherent in this approach. Nevertheless, simulators are quite useful and facilitate analysis of protocols, and they are very useful for a network researcher to verify the validity of proposed ideas before going forward with their implementation.

A number of packages are available in this category, like *packet tracer* [7], which is released and maintained by Cisco Systems, *KivaNS* [8], which is a free and open-source simulator providing both an application programming interface and a graphical interface, and *Network Simulator (NS)* [9], which allows students to design topologies and execute arbitrary simulations. *OPNET* [10] and *OMNeT++* [11] are two other well-known simulation frameworks that are somewhat similar to each other in functionality and with which various types of simulation models (such as network queues) can be employed.

A hybrid approach offers a middle road between the two extremes discussed above. In the hardware-based solution, the main limitation was the extensive dependency on the physical network devices. This limitation does not exist in the case of simulation, which also allows students to conduct experiments anywhere and at anytime, thus avoiding the expense and limited access time of laboratories. On the other hand, simulation limits the students to the simulated world and prevents them

Manuscript received February 23, 2011; revised May 18, 2011 and November 02, 2011; accepted December 12, 2011.

The authors are with the Department of Computer Engineering, Sharif University of Technology, Tehran 11155-9567, Iran (e-mail: b_momeni@ce.sharif.edu; kharrazi@sharif.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TE.2012.2183597

from having any interaction with real-world network traffic and the events that affect it. In a hybrid approach, however, simulators are connected to a physical network topology, allowing students to design, implement, and execute their code on a virtual node (or nodes), while allowing this node (or nodes) to exist and interact in a real network topology. For example, students could implement a router on a virtual node and then observe how their router interacts with real Internet traffic directed toward it.

This paper discusses the use of *Portable And Reliable Tool fOr Virtualization* (Partov) as a new hybrid educational tool. Partov provides a set of simulation models and a language for building simulation topologies. It also supports the distributed deployment of simulation codes with which students can implement/execute their simulation codes from their own computers while employing the central Partov server to coordinate the distributed codes with the centralized simulation models.

In Section II, an overview of Partov and its architecture is provided. Section III covers the various programming assignments handed out in the Partov-based computer networks class. Section IV discusses related works with a comparison to Partov. Section V discusses the effectiveness of the Partov-based programming assignments in the learning process, and Section VI draws conclusions.

II. Partov Architecture

This section discusses the general architecture of the Partov system; the reader is referred to [12] for a more detailed discussion of the system. In a high-level view, the Partov system consists of two main components, the *Network Simulation Server* (NSS) and the *Client Framework* (CF). The NSS is the central component of the Partov system that contains both the Partov kernel for creating virtual topologies and performing the simulation and the plugin infrastructure for extending the kernel functionality (explained in Section II-C). The NSS is responsible for all interactions with outside network via the *libpcap* [13] library and will cooperate with the CF in a client/server architecture.

The NSS in turn consists of three components: the *Simulation Server*, the *Virtualization Engine*, and the *Plugin Infrastructure*. These components and their relationship are shown in Fig. 1. The *Simulation Server* component is responsible for connecting the centralized virtualization framework to the distributed frameworks provided by the CF (see Section II-A). Whenever a CF instance requests a connection to the Partov server, the *Simulation Server* component will authenticate it via a username/password, and then instantiate a new topology map or locate a previously instantiated topology map instance and assign it to the CF.

A. CF

This component is the key enabler to making Partov suitable for easy educational use, allowing students' programs—executed on their own personal computers—to be connected to the Partov central server and participate in the planned simulations. For more information on how this framework can be used, the reader is referred to the CF user manual located at [14].

The CF will be able to send/receive packets in place of the connected virtual node. Whenever a packet is received by the connected virtual node, it will be inspected by the simulation

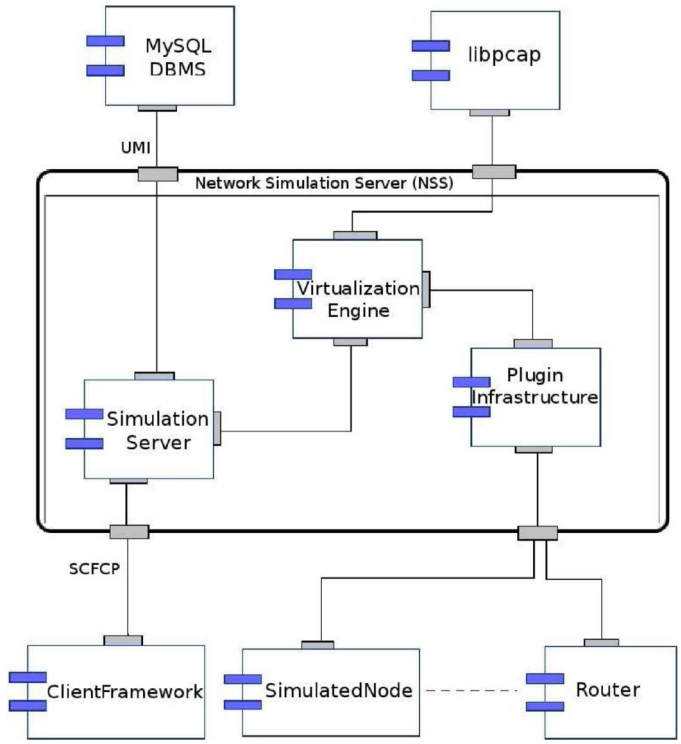

Fig. 1. Partov component diagram.

server that decides whether the packet must be forwarded to another component within the NSS itself or to the CF and in turn to the student's program. For example, the student could have implemented a bridge, router, or network address translation (NAT) functionality. This logic can inspect packets and decide whether it should ignore them, send replies, or send another set of required packets to the network. This allows students to implement algorithms that will interact with a real network environment.

B. Partov Topology Language

The *Partov Topology Language* (PTL) is an XML-based language that allows declaration of the Partov topologies. Each topology in PTL is defined in a separate file with a `.map` suffix and consists of three sections for defining *nodes*, *lists* (used for resource allocation to the nodes), and *links*.

Additionally, the PTL can include an optional *Finite State Machine* (FSM) section, allowing the topology to change dynamically. A simplified PTL file, shown in Fig. 2, will be used to describe each of these four sections. This sample PTL is obtained from an assignment on IP mobility (see Section III).

1) Link: In the `<links>` tag, each virtual link of the described topology could be defined by a `<link>` tag. For example, the first link in Fig. 2 has the name `lk-n1` and is referenced by the second link, namely `link1-d`, via its `connected-to-link` parameter. A wireless channel can be modeled as a link by connecting multiple nodes to a link. Furthermore, connecting links to other links can be used to model the collision between wireless channels.

Each wireless channel covers an area, namely the *propagation-area*. This is the area that the wireless signal can reach without a large attenuation. Each wireless receiver listens in an

Listing 1 Sample PTL File

```
<map version="2.2" name="mipoe" count="2">
 <links>
  <link name="lk-n1" loss="10%" latency="35ms" log="true" />
  <link name="link1-d" connected-to-link="lk-n1"
   direction="inout" bandwidth="50Kbps" /> .....
 </links>
 <fsm initial-state="s1">
  <state name="s1">
   <transitions><transition after="10s" probability="0.4" target="s2" />
   </transitions>
   <activity><do command="link-down" arg0="link1-d" /></activity>
  </state> .....
 </fsm>
 <nodes>
  <plugin name="node1" plugin-identifier="SimulatedNode">
   <interfaces><ethernet-interface mac-address="$node2-mac#"
     ip-address="192.168.1.18" netmask="255.255.255.252"
     connected-to-link="lk-n1" max-buffer-size="1MB" /> .....
   </interfaces>
   <parameters>
    <param name="accept-packets"><value>unicast-ip-ver4-only</value>
    </param> .....
   </parameters>
  </plugin> .....
 </nodes>
 <lists>
  <list name="node1-mac">
   <element>00:24:8C:01:79:01</element>
   <element>00:24:8C:01:79:07</element>
  </list> .....
 </lists>
</map>
```

Fig. 2. Sample PTL file.

area, namely the *receiving-area*. This is the area across which the wireless signal must reach in order to be detected by the receiver. Both the *propagation-area* and the *receiving-area* can be modeled using the *link* concept. For example, the `node1` node (from Fig. 2) that is listening on `lk-n1` link can receive packets sent over the `link1-d`, meaning that the `link1-d` *propagation-area* collides with the `lk-n1` *receiving-area*.

The links are used for connecting elements of the topology and can be configured to provide different bandwidth, latency, or loss probability rates. Additionally, traffic traversing each link can be logged in the pcap [13] format. For example, `lk-n1` is configured to log the delivered packets using `log="true"` parameter. These pcap logs can be used for further analysis by tools like tcpdump [13], Wireshark [15], etc.

2) FSM: PTL can define a *Finite State Machine* that can be used to create a dynamic topology structure. Examples of this would be modifying the routing table entries of simulated routers by adding a link between two separate routing areas at runtime (useful for checking implementation of routers in a class assignment), changing the network bridge topology by creating or removing loops between bridges at runtime, or making nodes movable within the topology like the `node1` in Fig. 2.

3) Node: In the `<nodes>` tag, each virtual node of the described topology can be defined by its tag that provides information about the node, such as its interfaces, buffer size, or IP addresses, as for the `node1` node (in Fig. 2), which is a `SimulatedNode` plugin. It also uses the `accept-packets` parameter under the `parameters` tag for identifying the fact that only valid IPv4 packets unicasted to it should be delivered to the CF for processing. These parametric definitions in PTL allow any plugin to be easily configurable via the standard PTL notation.

4) List: Lists are used for allocating resources to maps. Each topology can have multiple maps (i.e., instances), so they can be used by multiple concurrent users. Hence, each map requires its own resources, such as IP addresses. To address this allocation requirement, each map has an assignment index. This index can be used to distinguish between maps of a single topology. Then, each map can use its appropriate values from the lists. Thus, the first element in a list will be given to the first instantiated map (first arriving user), and the second element to the second instance (second arriving user), and so on. For example in the first map, node `node1` will have `00:24:8C:01:79:01` as its MAC address, and in the second map, it will have `00:24:8C:01:79:07` as its MAC address.

C. Plugin Infrastructure

This component provides an infrastructure for implementing plugins. Each plugin can use all of the utilities provided by the Partov for processing packets and can be configured via parametric definitions of the PTL. Currently, four plugins have been developed, and further plugins are being developed with the help of the student community.

- *Simulated Node* is an essential plugin through which packets are forwarded to the CF (i.e., student program), and packets returned from CF are injected back into the virtual environment.
- *Router* is an IPv4 router that accepts static routing tables. It is used to make topologies scalable (by creating separated network broadcast domains) and extensible (via hierarchical network topologies).
- *InternetGatewayNode* is a special router for connecting virtual maps to the real network that could be used for seamless integration of virtual maps and the Internet.
- *GeneralTCPReverseProxy* offers a means of emulating real outside servers within the virtual network by forwarding all Transmission Control Protocol (TCP) packets and their responses to/from real servers running in the network.

D. Virtualization Engine

This component serves the Partov's kernel. From each PTL file, the description of a topology is extracted, and from each topology, many instances—namely *map*—could be instantiated for different users (students). Therefore, multiple users can work simultaneously using the same topology, although a unique map will be assigned to each student. Also, users can create a common map serving multiple users, which could be beneficial for group collaboration on an assignment. There is a limitation that each student can only instantiate one map from each different topology at the same time, which is necessary to prevent *denial-of-service* (DoS) [16] attacks on the server.

III. Deployment and Course Assignments

Partov has been employed as a teaching tool for the Computer Networks course taught at the Department of Computer Engineering, Sharif University of Technology (SUT), Tehran, Iran, since 2009 [17]–[19]. Before reviewing the assignments handed out in class since the initial deployment of Partov, a typical assignment on node mobility, *Mobile IP over Ethernet* (MIPoE), is discussed in detail. The purpose of the *Mobile IP* protocol is to let nodes move through different

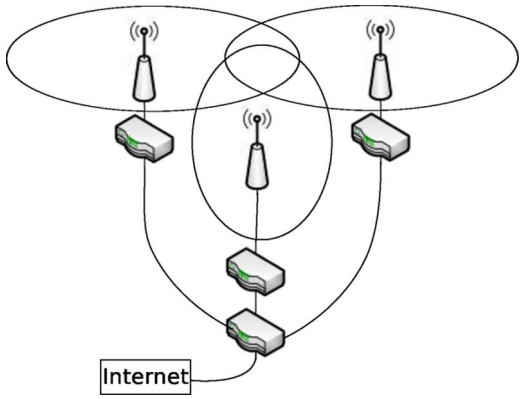

Fig. 3. MIPoE assignment topology.

networks while preserving their IP addresses, and hence their connections to those addresses. Students have to implement wireless access points (APs) so that wireless mobile nodes can keep their IP addresses while moving from one AP area to another, and implement the mobile nodes so they can request IP addresses from APs and speak with them while joining/leaving each wireless area.

In the assignment, each student is given a virtual topology consisting of three LANs as shown in Fig. 3. The PTL file shown in Fig. 2 is a summarized version of the MIPoE topology description. Each topology contains two nodes (like `node1` from Fig. 2) and three APs. Each AP has a coverage area and can receive packets from nodes in its coverage area. Also, packets transmitted by the AP reach all the nodes in its coverage area. For example, `node1`, which is connected to `lk-n1`, can send/receive packets to the first AP node through the `link1-d` link (which is similarly connected to the first AP node). Of course, the coverage area of the various APs can overlap. Packets that are sent in such overlap areas will be received by all of the related APs.

Using the FSM, it is possible to change a node's location (disconnecting it from a link and connecting it to another link) like the `s1` state, which shuts down `link1-d` and so pushes `node1` out of the coverage area of the first AP node. On the other hand, each AP is connected to the Gateway node, and so to the Internet via its second network interface.

Students are asked to prepare two programs using CF. The node program should start by asking for an IP address from an AP. The node will then move toward other APs, in the course of which it may exit one AP coverage area and enter another AP coverage area. It may also be in an area of overlap between multiple APs. The node program is expected to communicate with the AP program accordingly to find its current position. The AP program is expected to detect the node position and redirect packets from the home network to the visiting network via IP tunneling. Students are also asked to design their own protocol for receiving IP addresses in node programs, as well as a protocol for node–AP communication so the APs can keep track of nodes visiting the network at any one time.

The AP programs are expected to establish an overlay network between themselves to exchange information on the mobile nodes and establish IP tunnels to route node packets from the home to the visiting network. In terms of workload, it is possible for a student to complete this assignment in a total of about 15–20 h. Hence, a two-week deadline is suggested for this assignment, considering the workload from other courses taken by the student in the same semester. More importantly, after completing this assignment, the student is familiar with overlay and with the design of networking protocols based on concepts such as IP tunneling and IP mobility. The complete assignment statement can be found at [14].

In addition to the MIPoE assignment, a number of other assignments have been developed and handed out to students; a brief description of these follows.

- *Bandwidth Throttler:* In this assignment, which provides good insight into the third and fourth network layers, students are asked to implement a bandwidth throttler that operates at layer 4 and throttles TCP and UDP flows. With this assignment, students learn how to detect different TCP/UDP flows by inspecting IP, TCP, and UDP headers, how to determine a flow's rates, and how to limit a flow rate.
- *Simplified FTP Over UDP:* Here, students are asked to implement a simplified file transfer protocol using UDP transport layer. This assignment helps students learn concepts like end-to-end reliability, packet reordering, packet corruption, and packet loss/timeout. Use of TFTP is avoided in this assignment, as TFTP includes details and requirements unrelated to the networking concepts that are the focus of this assignment.
- *DHCP and ARP:* In this assignment, students implement both the server and client components of the *Dynamic Host Configuration Protocol* (DHCP) [20], and the *Address Resolution Protocol* (ARP), which focuses the data-link, network, and application layers. This teaches students address resolution via ARP, acquiring IP and other network information (like gateway address) via DHCP, and more on network layering concepts.
- *Simplified STP:* Here, students are first asked to implement a *learning bridge* to connect nodes in the topology. Afterwards, they are asked to upgrade their *learning bridges* and implement a simplified version of the STP. This teaches students how network nodes can participate in a distributed algorithm through local decisions and interactions.
- *Network Address Translation:* This assignment goes a bit further than the requirements of an NAT. It requires students to implement a client, based on the UDP protocol, and query for a file location from a location server, which they also have to implement, at the edge of a server farm located behind a NAT server. Seeing the query, the location server should set up proper NAT rules, so that the client can connect to the proper server and download the requested file. This teaches students about public and private IP addressing, UDP and TCP ports, and how NAT and static NAT mechanisms work in practice.
- *Transmission Control Protocol:* As this is one of the most important protocols in today's computer networks, students are asked to implement a minimal TCP and use it to download a file from a socket-based Web server. The correctness of the implementation and compatibility with

an actual TCP stack is verified by having the student only implement one side of the TCP connection and using a complete and standard implementation at the other end (i.e., socket-based Web server). This assignment teaches students about the TCP internals including TCP window management, flow control, ISN, sequence and acknowledgment numbers, and TCP flags.

- *Content Distribution Network (CDN):* CDN [21] is a network that allows fast response to large number of requests for content. Students are assigned a topology consisting of several geographically distributed server farms and are asked to create an overlay network over those server farms, redirecting requests to appropriate server farms based on their loads and paths latency. After this assignment, students should understand overlay networking and load balancing and have the ability to design a communication protocol for special purposes like between CDN server farms.
- *DHT and the Chord Protocol: Distributed Hash Table* (DHT) allows information to be distributed and stored among N peers. In this assignment, the student's program will be deployed on N distinct virtual nodes. These nodes must establish a DHT, store information about locations of some file hosting servers, and reply to location queries. The DHT should be resistant against random nodes leaving by storing information appropriately on other nodes. It should also use newly joined nodes and efficiently distribute information among them. With this assignment, students learn about distributed protocols, the DHT, and the Chord protocol.

IV. Related Work

To the best of the authors' knowledge, only two other systems within the hybrid category are comparable to Partov. *Emulab* [22] was initially started by the University of Utah and the Flux Research Group. Its primary installation [23] consists of more than 500 computers as nodes that are connected together via 12 switches and routers and coordinated via several other servers. Over this hardware deployment, the Emulab software establishes the required emulated links, sets their requirements like bandwidth, and creates various emulated topologies. Topologies are created based on the various assignment requirements. A large number of university courses [24] are currently using this infrastructure. Similarly, the Emulab infrastructure is currently deployed in a dozen other sites [25] across the world.

A more closely related work, which was also motivational in the design and development of Partov, is the Stanford University *Virtual Network System* (VNS) [26] project. The VNS server is connected to a number of physical servers running Web, SSH, and FTP services. These services are then accessible via simulated VNS topologies. Students can receive real Internet traffic originating from various services (i.e., web, ssh, etc.) for processing in the simulated topologies. Students can implement programming assignments using client base code, which in turn executes in the VNS infrastructure and processes packets received from real Internet traffic by virtual nodes in the virtual topology.

TABLE I
Hybrid Solutions: Their Associated Assignments and Network ISO OSI Layers Covered

No.	Assignment	Solution			Network layer			
		Partov	VNS	Emulab	2	3	4	7
1	Bandwidth Throttler	✓	✓	✓	×	✓	✓	×
2	FTP over UDP	✓	✓	✓	×	✓	✓	×
3	DHCP& ARP	✓	✓	✓	✓	✓	✓	✓
4	Simplified STP	✓	×	×	✓	×	×	×
5	NAT	✓	✓	✓	×	✓	✓	×
6	TCP	✓	×	×	×	×	✓	×
7	MIPoE	✓	×	×	✓	✓	×	✓
8	CDN	✓	×	×	×	✓	✓	✓
9	DHT	✓	✓	✓	×	×	×	✓

Table I compares Partov, VNS, and Emulab in terms of their support for the assignments used in the class. Although all three tools have the features required to be used with assignments 1–3, 5, and 9, assignments 4 and 6–8 are only supported by Partov. For example, the *Simplified STP* assignment requires the network topology to be changed at runtime. It would be possible to execute these assignments over the sort of fixed topologies provided by Emulab and/or VNS, but these tools would not directly support the checking of features such as the rearrangement of a bridges' tree (by adding or removing a link). Similarly, the *TCP* assignment can only be properly evaluated when there is the ability to simulate variable link loss and delay in order to evaluate the congestion management techniques implemented by the student. Furthermore, in the *MIPoE* assignment, neither the VNS and Emulab will work, as the assignment requires the nodes to be moved between various wireless areas at runtime.

Equally, the CDN assignment would have not been possible with VNS and Emulab since this requires online Internet connectivity, distributed processing for simulating multiple content distribution areas, and fine-grained control over the simulated network properties for simulating different load situations. This would both require the simulation tools to be connected to the Internet, processing real traffic and the dynamic nature for controlling the simulated network properties like RTT and loss (for different load situations).

Emulab has a far greater hardware requirement that easily differentiates it from Partov. The main difference between VNS and Partov could be described as the dynamic nature of Partov. The VNS-like Emulab only supports static and predefined topologies, while Partov allows the dynamic configuration of virtual topologies.

V. Discussion

The Partov system [14] is currently deployed at the Department of Computer Engineering, Sharif University of Technology, and was employed in the course Computer Networks for Fall 2009 [17], Spring 2010 [18], Fall 2010 [19], Spring 2011 [27], and currently in Fall 2011 [28]. Before Partov, VNS was employed in Spring 2009 [29] and at the

beginning of the Fall 2009 [17] class. Assignments were designed to reinforce the concepts taught in the class and covered various network layers in the ISO OSI model [30] as indicated in Table I. Furthermore, and in order to avoid handing out the same assignment every semester, different assignment scenarios are being developed while the concepts being reinforced remain the same.

Student feedback has been highly encouraging. Students state that the assignments helped them gain deeper understanding of the networking concepts taught in class, even though programming assignments are generally harder and more time-consuming than theoretical questions answered with pen and paper. This deeper understanding of the concepts becomes evident when discussing issues with students who have completed the assignments.

In order to obtain a more quantitative student opinion about Partov, and also more generally about using programming assignments in the class, an e-mail-based survey was taken by students who took the class between Fall 2009 and Spring 2011. Overall, 50 students participated in this survey, answering five main questions.

1) How helpful were programming assignments in better learning/understanding networking protocols/concepts?
2) How helpful was Partov in better learning/understanding networking protocols/concepts in class?
3) How easy was the programming [how much did the provided Partov client framework (CF) help]?
4) Were Partov or non-Partov assignments (i.e., Socket programming assignments) more helpful in learning/understanding the protocols/concepts in class?
5) Have you used any other network learning/simulation tools (i.e., Packet tracer, NS-2, VNS, OPNET, OMNeT++, KivaNS, etc.)?

Responses to question 4 show that 74% of students think Partov assignments were more helpful in learning protocols and concepts than the Socket programming assignments handed out in the course. In addition, responses to question 5 show that 81.25% of students had experience with other network learning tools, to which Partov could be compared.

Students answered the first three questions by selecting one of the five distinct levels, namely Extremely (100%), Very (75%), Somewhat (50%), Not Very (25%), and Not at All (0%). A similar survey was taken at the end of Fall 2010 [19], where 30 students participated and only the first three questions were asked. The average responses to the first three questions for both surveys are plotted in Fig. 4.

Survey results support the student feedback received since 2009. By answering the first question, students indicate a baseline for their opinion of the usefulness of programming assignments, which is about 76.4% on average over the two surveys. Then, by answering exactly the same question with just one difference, replacing *programming assignments* with *Partov*, they indicate how much of this usefulness is related to use of the Partov, which is about 67.4%. This assesses how this special Partov environment affected their thinking about programming assignments. The third question assesses the students' view of the quality and ease of use of the CF.

Also investigated was the correlation between grades for the Partov programming assignments and the final exam.

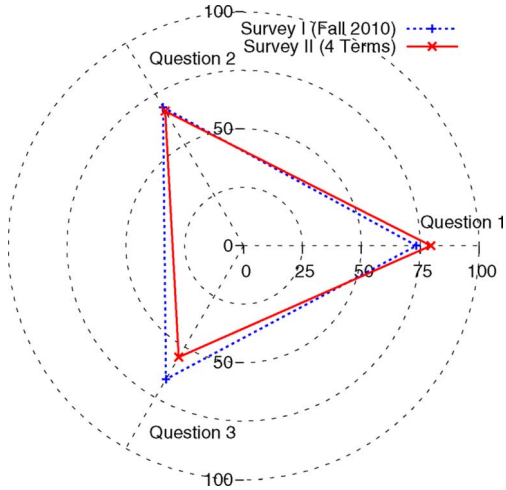

Fig. 4. Result of two surveys on the Partov system.

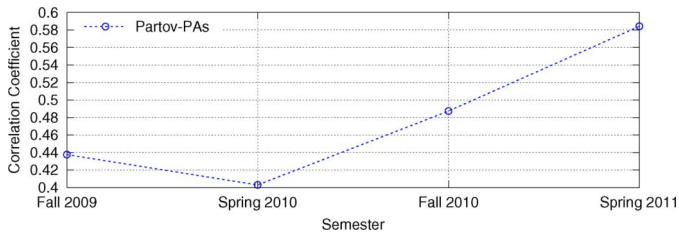

Fig. 5. Average correlation coefficient of grades of Partov programming assignments with the final exam grades.

The correlation coefficient of the final exam grade and the Partov assignment grades for each student was calculated; the averaged correlation coefficient for each semester is plotted in Fig. 5. This figure indicates the following: 1) performance in the final exams was effected by performance in the Partov programming assignments; and 2) this correlation increased as newer versions of Partov system were released, which indicates a continuous improvement in the Partov system.

The correlation between the Partov assignments and final exam grades is not very high. The Partov assignment grades were plotted against the final exam grades, as in Fig. 6, for all students who had taken the course during the past four semesters, from Fall 2009 to Spring 2011. The dotted line in the figure indicates the average final exam grade, and clearly students who did well in the assignments also did well in the final exam. However, there are a few exceptions, as seen with a few students who did well in assignments but not very well in the final exam, which can always happen due to some unpredicted personal issue. Alternatively, there were a few students who did averagely well in assignments but well in the exam, which again is understandable as there are always a few students who do not receive full marks for the assignments but nevertheless grasp the underlying concepts.

More importantly, Fig. 6 indicates that students with lower grades in Partov assignments obtain a more distributed range of grades in the final exam, while students with higher grades in the Partov assignments have a narrower and higher range of grades in the final exam. Therefore, although the assignment grades and final exam grades are not completely correlated, it is

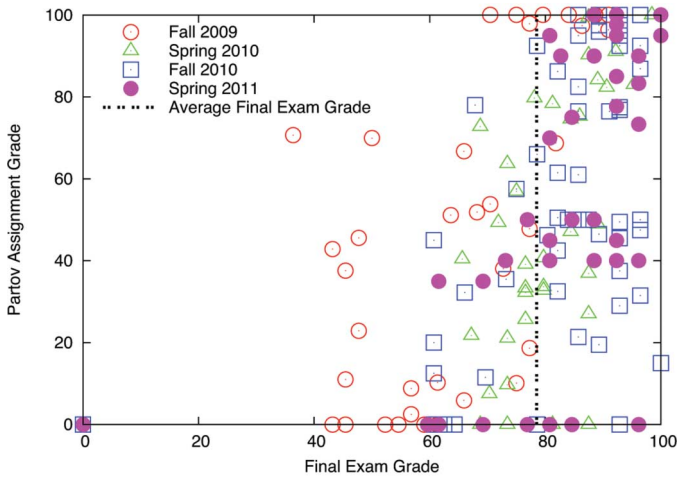

Fig. 6. Final exam grades versus Partov assignment grades during four semesters.

evident from the figure that assignment grades clearly affect the distribution of the final exam grades.

Nevertheless, it could be argued that any tool such as Emulab, VNS, or Partov can improve student learning, but what makes Partov different from previously proposed systems is the fact that it allows more flexibility with respect to VNS by simulating dynamic environments and is more affordable than Emulab, which requires extensive hardware deployment. These two parameters are of importance for any instructor wanting to improve a course by employing such tools.

Specifically, Partov's ability to simulate dynamic environments, similar to that of the Internet, is an important feature that allows a wider coverage of the concepts covered in class. For example, the ability to model a variety of network links with full control over properties like loss and delay is helpful in designing and evaluating of the assignments and is specially helpful when evaluating routing protocols implemented by students. Other assignments that could be based on Partov and are being developed include, but are not limited to, TCP congestion window controlling algorithms, fast retransmission algorithms, P2P file sharing, distributed hash tables, routing protocols, BGP, and so on. For more up-to-date information on the project and developed assignments, the reader is referred to the Partov project Web page [14].

VI. Conclusion

This paper has presented the experience of employing the Partov simulation engine as a teaching tool in the Computer Networks course. The architecture of Partov allows students to easily implement real network devices and observe how they interact with real Internet traffic. During this period, a number of programming assignments were designed to help students acquire a deeper understanding of the networking concepts. Assignments covered various concepts, from the data-link layer up to the application layer. Furthermore, student feedback has been favorable and was quantified with the help of student surveys. Partov is currently being expanded to include an online judge for the automated grading of assignments, and work on a graphical interface to help in generating PTL files given a network topology is in progress.

Acknowledgment

The authors would like to thank the Computer Networks course teaching assistants, including A. Abniki, S. Dorri, H. Eslami, A. Fattaholmanan, D. Jalali, S. Khajouie, K. Mirhosseini, F. Moghaddam, A. Nikravesh, S. Pooya, A. Shaikhha, and M. Zolghadr, for their help and useful suggestions on the development of the system and programming assignments.

References

[1] R. Perlman, "An algorithm for distributed computation of a spanningtree in an extended lan," *Comput. Commun. Rev.*, vol. 15, no. 4, pp. 44–53, 1985.
[2] B. Smith, "CMPE 151, Network Administration, Spring 2010" University of California, Santa Cruz, CA, Dec. 5, 2011 [Online]. Available: http://www.soe.ucsc.edu/classes/cmpe151/Spring10/
[3] Y. Liu, L. Zhang, and F. Jiao, "Teaching computer networking experiment in the realistic network laboratory," in *Proc. CiSE*, Dec. 2009, pp. 1–4.
[4] J. Theunis, B. V. D. Broeck, P. Leys, J. Potemans, E. V. Lil, A. V. D. Capelle, W. Lans, and V. Streaming, "Opnet in advanced networking education," opnetwork02 johan.pdf, Dec. 5, 2011 [Online]. Available: http://www.esat.kuleuven.be/telemic/networking/
[5] N. Al-Holou, K. Booth, and E. Yaprak, "Using computer network simulation tools as supplements to computer network curriculum," in *Proc. 30th Annu. FIE 2000*, 2000, vol. 2, pp. S2C/13–S2C/16.
[6] X. Yu, "The construction and application of simulation teaching system for computer network curricula," in *Proc. 1st IEEE ISITAE*, Nov. 2007, pp. 524–527.
[7] Cisco Systems, "Cisco packet tracer," Dec. 5, 2011 [Online]. Available: http://www.cisco.com/web/learning/netacad/course_catalog/PacketTracer.html
[8] F. Candelas Herias and P. Gil Vazquez, "Practical experiments with KivaNS: A virtual laboratory for simulating IP routing in computer networks subjects," in *Proc. m-ICTE*, 2009, vol. 3, pp. 1414–1418.
[9] "The network simulator—ns-2," Dec. 5, 2011 [Online]. Available: http://www.isi.edu/nsnam/ns/
[10] R. Kaparti, "OPNET IT Guru: A tool for networking education" MSCIT Practium Paper, Regis University, Denver, CO, Aug. 14, 2011 [Online]. Available: http://www.opnet.com/university_program/teaching_with_opnet/textbooks_and_materials/materials/ITGAE_Tool_Ntwrk_Ed.pdf
[11] A. Varga, "Using the OMNet++ discrete event simulation system in education," *IEEE Trans. Educ.*, vol. 42, no. 4, p. 372, Nov. 1999.
[12] B. Momeni, "Hybrid virtual network system," B.S. thesis, Sharif University of Technology, Tehran, Iran, 2010, portable And Reliable Tool fOr Virtualization.
[13] "Tcpdump/libpcap public repository," Dec. 5, 2011 [Online]. Available: http://www.tcpdump.org/
[14] M. Kharrazi, "Partov project," Dec. 5, 2011 [Online]. Available: http://sharif.edu/~kharrazi/partov/
[15] Wireshark, "Wireshark," Dec. 5, 2011 [Online]. Available: http://www.wireshark.org/
[16] S. de Vries, "Application level denial of service (DoS) attacks," 2004 [Online]. Available: http://research.corsaire.com/whitepapers/040405-application-level-dos-attacks.pdf
[17] M. Kharrazi, "CE 40-443: Computer networks," Sharif University of Technology, Tehran, Iran, Dec. 5, 2011 [Online]. Available: http://sharif.edu/~kharrazi/courses/40443-881/
[18] M. Kharrazi, "CE 40-443: Computer networks," Sharif University of Technology, Tehran, Iran, Dec. 5, 2011 [Online]. Available: http://sharif.edu/~kharrazi/courses/40443-882/
[19] M. Kharrazi, "CE 40-443: Computer networks," Sharif University of Technology, Tehran, Iran, Dec. 5, 2011 [Online]. Available: http://sharif.edu/~kharrazi/courses/40443-891/
[20] R. Droms and T. Lemon, *The DHCP Handbook (Networking)*, 2nd ed. Indianapolis, IN: Sams, Nov. 2002.
[21] G. Peng, "Cdn: Content distribution network," State University of New York at Stony Brook, Stony Brook, NY, Tech. Rep., 2003.

[22] Emulab, "Emulab—Network emulation testbed home," Dec. 5, 2011 [Online]. Available: http://www.emulab.net/
[23] Trac, "Hardware overview, "Emulab Classic"," Dec. 5, 2011 [Online]. Available: http://users.emulab.net/trac/emulab/wiki/UtahHardware
[24] Emulab, "Projects that have actively used emulab.net," Dec. 5, 2011 [Online]. Available: http://www.emulab.net/projectlist.php3
[25] Trac, "Other Emulab testbeds," Dec. 5, 2011 [Online]. Available: http://users.emulab.net/trac/emulab/wiki/OtherEmulabs
[26] M. Casado and N. McKeown, "The virtual network system," *SIGCSE Bull.*, vol. 37, no. 1, pp. 76–80, 2005.
[27] M. Kharrazi, "CE 40-443: Computer networks," Sharif University of Technology, Tehran, Iran, Dec. 5, 2011 [Online]. Available: http://sharif.edu/~kharrazi/courses/40443-892/
[28] M. Kharrazi, "CE 40-443: Computer networks," Sharif University of Technology, Tehran, Iran, Dec. 5, 2011 [Online]. Available: http://sharif.edu/~kharrazi/courses/40443-901/
[29] M. Kharrazi, "CE 40-443: Computer networks," Sharif University of Technology, Tehran, Iran, Dec. 5, 2011 [Online]. Available: http://sharif.edu/~kharrazi/courses/40443-872/
[30] H. Zimmermann, "OSI reference model-the ISO model of architecture for open systems interconnection," *IEEE Trans. Commun.*, vol. COM-28, no. 4, pp. 425–432, Apr. 1980.

Behnam Momeni received the B.Sc. degree in information technology (with first rank) from Sharif University of Technology, Tehran, Iran, in 2010, and was accepted as a talented student in the M.Sc. computer engineering program at Sharif University of Technology.

He is currently conducting research on computer and network security. His research interests include network security, multimedia systems, peer-to-peer networks, and software design patterns and methodologies.

Mehdi Kharrazi received the B.E. degree in electrical engineering from the City College of New York. New York, in 1999, and the M.S. and Ph.D. degrees in electrical engineering from Polytechnic University, Brooklyn, NY, in 2002 and 2006, respectively.

He is currently an Assistant Professor with the Department of Computer Engineering, Sharif University of Technology, Tehran, Iran. His current research interests include network and multimedia security.