Introducing Hands-On Experience to a Massive Open Online Course on openHPI

Christian Willems, Johannes Jasper, and Christoph Meinel
Internet Technologies and Systems Group
Hasso Plattner Institute for IT Systems Engineering
Potsdam, Germany
Email: {christian.willems, meinel}@hpi.uni-potsdam.de,
johannes.jasper@student.hpi.uni-potsdam.de

***Abstract— Massive* Open Online Courses (MOOCs) have become the trending topic in e-learning. Many institutions started to offer courses, either on commercial platforms like Coursera and Udacity or using own platform software. While many courses share the concept of lecture videos combined with automatically assessable assignments, and discussion forums, only few courses provide hands-on experience. The design of practical exercises poses a great challenge to a teaching team and gets even more challenging if these assignments should be gradable. In the course *Internetworking with TCP/IP* on the German MOOC platform *openHPI,* the teaching team conducted an experiment with three practical tasks that were implemented as assessed bonus exercises. The exercise design was limited by the constraint that the platform software could not be adapted for these exercises and that there could be no central training environment to perform these assignments. This paper describes the experiment setup, the challenges and pitfalls and evaluates the result based on statistical data and a survey taken by the course participants.**

Keywords—MOOC, hands-on experience, online assessment, TCP/IP, e-mail, DNS.

I. INTRODUCTION

Starting from 2011, a new concept of Massive Open Online Courses (actually xMOOCs, see [2]) emerged from open online courses at Stanford University, which, like a traditional university lecture, offers a well-defined body of knowledge. In general, MOOCs draw on three types of resources for the dissemination of this knowledge to a massive audience: (1) video lectures, mostly segmented into small pieces, and presented in an engaging and entertaining manner; (2) interactive quizzes that allow immediate exercise of the learning content; and (3) communication tools efficiently managed by the learning community, that allow to highlight, discuss and solve relevant questions.

openHPI[1] is a platform for MOOCs of this type, hosted at the Hasso Plattner Institute (HPI) in Potsdam, Germany. The first two courses on this platform have targeted two very distinct audiences: while the first course, *In-Memory Data Management,* was offered in English and dealt with an advanced topic in database technology, the second course, *Internetworking mit TCP/IP* was targeting a German-speaking non-specialist audience and offered an introduction to networking technology. Both courses have met with substantial interest from the respective target audience: 13,126 learners registered for the *In-memory* course, from which 4,068 actively participated and 2,137 received the graded certificate of successful completion. The *Internetworking* course had 9,891 registered learners, with 2,726 active participants, and 1,635 successful completions with graded certificates.

When reviewing the first course, we could observe a demand for hands-on experience. The teaching team of the *In-Memory* course invited the participants to take part in a short survey. About 440 students answered to a free text question on feature requests: *"Which features did you miss? Would you propose any additional community features? Please consider the overall website, as well as the videos, quizzes and the forum."* A remarkable number of users asked for practical examples and exercises, which is outlined with the following example statements:

> *"Some practical, hands-on exercises on SanssouciDB would be very helpful [...]"*
>
> *"More practical examples e. g. run SQL and see [...] how the statement was processed."*
>
> *"It would be awesome to have practical assignments."*

The teaching team that supervised the second course took up this suggestion and implemented three bonus exercises that could only be solved practically. These exercises still build on the technical capabilities of the openHPI platform; no major modifications were made to the software.

The paper at hand describes the setup for this experiment as well as the teaching team's experience. In the following sections, we describe the capabilities and limitations of openHPI assessment tools and motivate the need for hands-on exercises from an academic point of view. Section II describes the experiment setup in detail. This covers the design of the practical assignments as well as the learning objectives. Concretely, we had exercises on *Packet Inspection with Wireshark, DNS Resolution,* and *E-mail Transport with SMTP*. Furthermore, we highlight limitations and possible refinements of the chosen approach. Section III presents some numbers on the commitment of the course participants concerning these hands-on exercises, enriched with results from an evaluation survey taken by more than 1,000 course participants of the

[1] accessible at https://openhpi.de/

Internetworking course. Section IV summarizes the results and gives an outlook on future activities.

A. Tests and Assessment in openHPI Courses

The current (and first) version of the openHPI platform software is a massively modified fork of the open-source learning management system *Canvas*[2]. While we customized many parts of the platform, removed lots of functionality and added own components, the original quiz environment remained nearly untouched.

The sophisticated environment allows for easy creation of automatically verifiable quizzes that we use for self-tests, homework assignments and the final examination. Each of the lecture videos, which build the central part of the course contents, is associated with a self-test covering its topic. These tests should allow the learners to verify their understanding and to monitor their learning progress. Thus, self-tests can be taken as often as necessary; the results do not count in for the overall course score. One homework assignment is issued each course week and has to be solved until the beginning of the following week. These homework assignments have a strict time limit (e.g. one hour) and can only be submitted once. The user frontend of the platform's quiz environment implements the enforcement of these limitations. Homework assignments come with points that count for the overall score. Precisely, on openHPI the cumulative homework assignments for 6 course weeks are awarded with the same amount of points as the final examination. Course participants must gain at least 50% of the overall maximum score to qualify for an *openHPI certificate.*

Due to the large number of participants in our courses, all assignments must be assessable in an automatic manner. This constraint limits the possible question types to *multiple choice/multiple answer, true-false, pop-up/drop-down, fill-in the blanks, matching,* and *ordering* (plus variants). These types of questions allow for a very limited degree of freedom concerning the answers. In contrast, questions with answers in essay style provide way more flexibility but can only be graded manually. Other MOOC platforms provide facilities for peer reviewing (e.g. Coursera, see [4]), where students revise each other's essay submissions in a double-blind review process. Due to the high number of inactive enrolled students, the difficulty to formulate objective criteria for consistent review quality, and other issues [5], openHPI currently abandons this option.

B. Hands-On Learning and E-Assessment

The implementation of systems that allow the assessment of practical exercises can be a great challenge for course and platform designers. There are numerous solutions for courses and online laboratories in the domain of programming (e.g. *Codeacademy*[3]*, CodingBat*[4]*,* or several *Coursera* offerings on programming[5]), and databases (e.g. Stanford's online course *Introduction to Databases,* known as *DB-Class*[6]*).*

The programming classes usually provide the possibility to code in a browser environment, where an interpreter or compiler runs on the server and provides feedback to the user. The feedback can also be enhanced with hints or explanations that help the students to understand what went wrong. Assessment can be done by

- automatic inspection of the submitted code fragments,
- evaluation of the program output when being compiled and/or executed,
- running the submitted code against unit tests,

or a combination of several of these methods. Database classes can follow a similar design for the practical training environment but would replace interpreters and compilers by an SQL command prompt, e.g. of a lightweight SQLite database. Running these training environments in a cloud-driven infrastructure allows easy adoption of resources for MOOC-scale classes.

These kinds of assessable hands-on exercises are limited to specific domains of computer science. More general approaches for laboratory environments from the past years usually build on virtualized computer labs that provide remote access to virtual machines running on a central server (respectively the cloud) or are distributed on removable media. These labs can provide pre-configured machines with operating systems and installed software or even computer networks. Examples for this kind of training environments are summarized in [1]. Nevertheless, the majority of the research projects on virtual lab environments for education do not cover the question of automatic assessment of practical exercises. The authors of [3] for example implemented a service that allows the students to evaluate their practical work by means of a scripted test procedure: if the assigned task was about configuring a firewall to restrict the access on certain ports, the test script would run a port scan against the student's lab VM and then parse the port scanner's output for open and closed ports. In [7], the authors propose a more generic approach for the automatic assessment of hands-on exercise assignments: the lab management system asserts a student- and task-specific pre-condition that is configured inside a training machine before the student can get access. During the exercise, the student can reveal a "secret" that is affected by the pre-condition and thus proof the successful completion of the practical task by submitting the unique secret value to a quiz environment.

For the practical exercises in openHPI, we follow a similar approach, but do not implement a centralized training lab – openHPI students had to perform the exercises on their own computers.

II. Introducing Hands-On Exercises in openHPI

The general idea of hands-on exercises for the *Internetworking* course on openHPI was motivated by the teaching team's experience with on-campus courses and seminars. The instructional elements of the course (lectures, slides, textbook excerpts) give an abstract view on the technical aspects of e.g. network protocols. These can easily be transferred to the real world just by inspecting the messages that are sent over a

[2] see https://github.com/instructure/canvas-lms
[3] see http://www.codeacademy.com/
[4] see http://codingbat.com/
[5] see e.g. „An Introduction to Interactive Programming in Python", available at https://www.coursera.org/course/interactivepython
[6] see http://class2go.stanford.edu/db/

network. This applies for all layers of the TCP/IP reference networking stack: when knowing about the protocols and their headers it is straightforward to actually "read" realistic Ethernet packets or ARP messages on the low level, as well as high level protocols like SMTP (for mail transfer) or Domain Name resolution messages.

The teaching team designed three practical exercises, which were about using tools and interpreting real life Internet messages. The exercise design included a quiz for each assignment that could only be solved after the practical tasks had been performed. Due to the experimental character of these exercises, they were treated as voluntary *bonus exercises.* The points for these exercises (one third of the points for a regular homework assignment) did actually count for the overall course score. However, they were not needed to reach 100% of the overall score, giving the participants the opportunity to make up for missing points from regular homework assignments. In the following, we describe each of the three bonus exercises in detail.

A. Exercise 1: Wireshark and the IP Identification Field

In this exercise the students were supposed to find out how the identification field of IP packets is determined on their system.

The IP *identification field* (also called *IP-ID)* is a number stored in every datagram using the Internet protocol (IP). Its purpose is to uniquely identify packets in a transmission. This is necessary when packets need to be fragmented due to size limitations of the underlying LAN technology. In the process of reassembling the original packet, all its fragments are accumulated. The IP-ID is used to identify the fragments belonging to the same original IP packet. The number therefore needs to be unique to distinguish fragments of different packets. This is specified in the standard RFC 791 [8]:

> *"The originating protocol module of an internet datagram sets the identification field to a value that must be unique for that source-destination pair and protocol for the time the datagram will be active in the internet system."*

It does not, however, specify how this uniqueness is to be achieved.

The number in the identification field can either be picked randomly or generated by incrementing the IP-ID of the last sent packet. This implementation is integrated into the operating system. The correct answer for this question therefore varies, and depends on the system used by the students. The operating systems we took into consideration for this task were Microsoft Windows, Apple OS X, Linux and openBSD. While Windows and Linux increment the IP- ID of outgoing packets, Mac OS X and BSD-Unix variants pick the numbers randomly. In this specific task we made use of the inhomogeneity of the students' systems to emphasize one of the learning objectives, which was to demonstrate that standards can be object to interpretation, and that there can be a discrepancy between different implementations of the same specification.

The students were advised to use the packet sniffer *Wireshark*[7]. A packet sniffer (also called network analyzer) is a tool capable of capturing data packets that are exchanged on the local network and of examining their internal structure. The introduction of this tool gave the students the opportunity to gain first hand insight into the workings of their local network. Another learning objective of this task was to show that the structures and processes introduced theoretically in the lecture are well applicable to the Internet. A tutorial showing the required range of functionality of Wireshark and its basic use was published beforehand in form of a screencast. We offered tutorials and individual assistance in installing and configuring the application on the students' computers. Furthermore, the students helped each other with technical problems using openHPI's discussion forum. The structure of IP datagrams including the mechanisms of fragmentation and reassembly as well as the IP-ID header field was explained in the lecture.

In order to solve the task, we expected the participants to observe packets sent by their computers and create traffic if necessary. The students had to apply filter rules in order to reduce the displayed packets to outgoing IP datagrams of one connection. Thereupon, they were able to examine the header information of several IP datagrams and look for patterns in the change of the identification field.

For each combination of the four considered operating systems and the two possible implementation strategies we offered a multiple-choice answer. The actual question was *How is the IP identification field value of subsequent packets chosen on your system?* The answer options for each operating system were *incremental* and *random.*

B. Exercise 2: Understanding DNS Resolution

In this exercise the students were supposed to retrieve information about the Domain Name System from a real live example.

The Domain Name System (DNS) is a service that provides a naming scheme for online resources [9]. It enables users to work with meaningful *domain names* instead of technical qualifiers, i.e. IP addresses. The Domain Name System has a hierarchic structure, in which every level specifies the target host or service in more detail. The DNS provides a format for storing associations of domain names and IP addresses in so called *resource records.* Servers that are the official owner of a domain name are called *authoritative* name servers for their respective resource record. Other non-authoritative name servers can temporarily cache this resource record in order to increase the overall performance of the Domain Name System. The resolution of a domain such as www.google.com involves multiple name servers. Each name server either has the necessary resource record available in its memory and returns it, or it refers to another name server that is lower in the hierarchy. If neither applies the requested domain name does not exist.

At this point one has to differentiate between two types of name resolution, namely *iterative* and *recursive* resolution. When receiving a query to which it does not have a matching

[7] see http://www.wireshark.org/

resource record, a name server working iteratively returns a reference to another name server. The user then has to repeat the query to this new name server. A name server working recursively, in contrast, queries the new name server itself and only returns the result to the user. Therefore, in a recursive resolution the user is not able to keep track of the resolution.

Even though the result of a DNS query – the IP address of the requested resource – is usually constant over time, the process of resolving a domain name is highly flexible and depends on numerous factors. The specification of the DNS offers a variety of flags [9], each inducing slightly different behavior. Furthermore, a wide range of tools exists for the various operating systems, each with a different set of features. Most importantly, the state of the name servers, especially their caches, is not reproducible. Therefore, an exercise in which the participants created a DNS query themselves would have led to incomparable results. Instead, we provided the students with a file including a stream of packets captured with the sniffer Wireshark, as introduced in exercise 1 (see II-A). The file could be opened in Wireshark in order to restore the recorded stream.

The captured packets were created during the resolution of the domain name `www.google.com`. The query was sent from within the network of the Hasso Plattner Institute. It had two noteworthy characteristics. On the one hand, the resolver was instructed to work iteratively, to ensure that all intermediate steps in the process of the resolution were visible to the students. On the other hand, the query's DNS header had the *Authoritative Answer* (AA) flag set, which instructs the resolver to ignore any non-authoritative answers. This prevents name servers other than the ones from Google (such as servers within the Hasso Plattner Institute) to provide the requested IP address.

The captured packets showed that the local resolver initially required the IP address of a root name server and queried the name server of the Hasso Plattner Institute. Such a root name server (`a.root-servers.net`) then was queried for the IP address of `www.google.com`. It returned a reference to an authoritative name server for the .com top-level domain such as `a.gtld-servers.net`. After being queried for `www.google.com`, this server returned a reference to a Google name server (`ns1.google.com`). This name server is authoritative for the domain `google.com` and provided the user with the required resource record.

The specific questions to the students were the following:

What domain name is resolved in this example? By inspecting the numerous DNS queries in the recorded stream, the students were able to see that the Domain Name System was used to resolve the domain name `www.google.com`. All other queries were intermediate steps.

Is the resolution performed iteratively, recursively or both? As explained above, a recursive resolution is invisible to the user as it is performed by the name servers themselves. The fact that every intermediate step of the resolution is visible within the stream of captured packets indicates that the resolution was performed iteratively.

In which order are the following name servers involved with the resolution of the domain name?

- *Google's name servers*
- *Verisign's name servers*
- *the name servers within the Hasso Plattner Institute*
- *a DNS root name server*

As described before, the resolver fist asks the local name server for a DNS root server. In private networks the local name server assigned by the Internet service provider. In larger networks though, there are often private name servers, as is the case with the Hasso Plattner Institute. In a next step, the DNS root servers are queried. These refer to name servers authoritative for the required top-level domain. In this example Verisign[8] runs the servers holding resource records for .com domains. Those name servers are not authoritative for the domain either and refer to Google's name servers. At last, the name servers run by Google are authoritative for the queried domain name www.google.de and return the required IP address.

The major learning objective in this task was to understand the internal workings of the Domain Name System. The students had the opportunity to examine a DNS query, which is usually performed in the background by the operating system. The use of a packet sniffer enabled them to experience the technical implementation of the theoretic principles learned in the lecture.

C. Exercise C: Inspecting E-Mail Headers

In this exercise the students were supposed to read and interpret the source code of an e-mail. The e-mail was sent from within the network of the Hasso Plattner Institute to every person enrolled in this course.

The first part of this exercise addresses the transfer of messages on the Internet.

The transfer of electronic mail is specified in the *Simple Mail Transfer Protocol* (SMTP) [10]. To read and write messages, the user interacts with a *Mail User Agent* (MUA), usually incorporated in an e-mail application. In order to send mail, the MUA hands it over to the user's *Mail Transfer Agent* (MTA). Using the Domain Name System, the MTA searches for another MTA closer to the receiver and forwards the message via SMTP. Note that during the transfer of an e-mail, it may pass several MTAs before reaching its final destination. The endpoint is the receivers *Mail Delivery Agent* (MDA) where the e-mail can be downloaded. Concerning the process of forwarding, RFC 5321 specifies the following [10]:

> *"When forwarding a message into or out of the Internet environment, a gateway MUST prepend a Received: line, but it MUST NOT alter in any way a Received: line that is already in the header section."*

Every MTA forwarding the message leaves a line in the e-mail header, indicating its identity. Thus, a message's route from

[8] http://www.verisign.com

sender to receiver can be retraced by inspecting the individual Received: lines in its source file.

Provided with the source code of an e-mail, the students were asked how many MTAs the message passed within the network of the Hasso Plattner Institute. Additionally, they were supposed to find out the name of the MTA that accepted the message from the MUA.

As explained above, the header-field of an e-mail gives insight into the path it took. The relevant lines are depicted in Listing 1. In order to reconstruct the route of a message one has to read the `Received:` lines from bottom to top. Each MTA identifies itself with the keyword by and its host name or its IP address. Listing 1 shows that the first MTA that accepted the message called itself `webuniVM82`. To find out how many MTAs the message passed within the network of the Hasso Plattner Institute one has to count the `Received:` lines, that refer to the HPI's domain name. As the assignment states, the message was sent from within the HPI, `webuniVM82` is therefore known to be part of the network. The MTAs identified in lines 9, 12 and 15 of Listing 1 each have the HPI-specific domain name `.hpi.uni-potsdam.de`. Thus, the message passed 4 MTAs within the HPI.

```
1  Received:   by 10.58.189.9
2      Fri, 30 Nov 2012 8:16:21 -0800 (PST)
3  Received:   by 10.204.136.207
4      Fri, 30 Nov 2012 8:16:20 -0800 (PST)
5  Received:   from mail3.hpi.uni-potsdam.de
6      by mx.google.com
7      Fri, 30 Nov 2012 08:16:20 -0800 (PST)
8  Received:   from owa2.hpi.uni-potsdam.de
9      by mail3.hpi.uni-potsdam.de
10     Fri, 30 Nov 2012 17:16:19 +0100 (CET)
11 Received:   from webuni-piwik.hpi.uni-potsdam.de
12     by owa2.hpi.uni-potsdam.de
13     Fri, 30 Nov 2012 17:16:18 +0100
14 Received:   from [10.210.0.199]
15     by webuni-piwik.hpi.uni-potsdam.de
16     Fri, 30 Nov 2012 17:16:18 +0100
17 Received:   from openhpi
18     by webuniVM82
```

Listing 1. Excerpt of an e-mail header-field. The Received: lines show the route the message took

Note that the first part of each message's route lays within the same network. Therefore, even though the messages take different routes to each user and therefore a different set of MTAs is involved in the transfer, the critical part of the messages' route is constant for each participant. Thus the task was automatically assessable despite the inhomogeneous routes.

The second part of this exercise covers the e-mail's content, rather that its transfer.

SMTP, as defined in RFC 5321 [10], was designed to supports the 7-Bit ASCII character set. This does not, however, allow language specific characters or non-text attachments, as they are very common nowadays. The Multipurpose Internet Mail Extensions (MIME) introduces a means to overcome this restriction [11]. It defines new header fields describing the structure and the content of the message. The *Content-Type* field describes the media type of the message's body. With the type `multipart/mixed`, MIME allows e-mails to be split in several parts, separated by a custom boundary. Each part of the message declares its own content, also using the Content-Type header field. By default this is `text/plain`, indicating a text-message. It can, however, also indicate novel types such as `image/jpeg`. This field enables the user agent to adjust to the submitted content and display it correctly. Another important MIME header field is the *Content-Transfer-Encoding*. This indicates how the submitted content is encoded to match the specifications of the transfer protocol (7-Bit ASCII as is the case with SMTP).

```
1 MIME-Version: 1.0
2 Content-Type: multipart/mixed;
3   Boundary="mimepart_50b8dbd12afa"
4 --mimepart_50b8dbd12afa
5 Content-Type: text/html; charset="utf-8"
6 Content-Transfer-Encoding: Quoted-printable
7
8 <p>Dear Students,</p> [...]
9 --mimepart_50b8dbd12afa--
```

Listing 2. MIME header fields of an exemplary e-mail (excerpt)

In this part of this exercise the students were supposed to identify the Content-Type and Content-Transfer-Encoding of the received e-mail. For this purpose, they had to open the e-mail's source code and find the lines defining the MIME header fields. They read as depicted in Listing 2. The actual content of this message is the text beginning in line 8, the Content-Type of this part is `text/html`, as can be seen in line 5. The following line declares it's Content-Transfer-Encoding as `Quoted-printable`.

The major learning objectives for this task was to understand the internal workings of electronic mail, one of the oldest Internet applications. The exercise demonstrated how Internet standards are adjusted to changing demands and technical innovation. Furthermore, the students got the opportunity to read an e-mail's source code which is usually hidden by the e-mail application.

D. Challenges and Pitfalls with the Hands-On Exercises

As explained in the previous sections, the exercises were supposed to be solved on the students' personal computers. This created a challenging inhomogeneity amongst the involved machines, operating systems and network infrastructures.

Many support requests and forum discussions focused on the first hands-on exercise (as described in chapter II-A), in which the students had to examine local network traffic using the packet sniffer Wireshark. Most problems emerged during the installation of Wireshark. The provided tutorial offered hints on where to find installers for all supported operating systems. In some cases however, further instructions were necessary. As the tool accesses the network controller, administrative privileges had to be granted in order to install and run the packet sniffer. Users with little experience in computer administration or users that installed Wireshark on machines at their workplace received further assistance – which mainly came from the community, where experienced users helped out with hints and troubleshooting.

Another challenge was posed by the various network infrastructures. Several participants, for example, reported problems when connecting to the Internet with a mobile UMTS adapter. Other users (especially in enterprise settings) had problems due to restrictive firewall settings.

Apart from technical issues, some users raised legal concerns, especially related to the intrusion into other peoples' privacy. Besides an admonition to only use the tool in their own private networks and with consent of all other participants, we gave advice on how to protect personal communications from eavesdropping.

In the process of validating the expected answers for the first hands-on assignment, we found that the effects observed with Wireshark strongly depend on how network traffic is generated. Linux for example, assigns an IP-ID of 0 if the `Don't Fragment (DF)` flag is set in the IP packet header. Students who generated network traffic without paying attention to the DF flag might be lead to wrong conclusions. Further, we expected Linux to increment the IP-ID by 1 in each new packet. It turned out though, that this is only the case for packets of the same connection (identified by a pair of hosts). The initial start value for each connection is picked randomly. As this circumstance was not specified in the assignment, we accepted both possible answers from Linux users.

Further problems emerged in the third hands-on exercise. As described in section II-C, the students were supposed to read and interpret the header field of an e-mail. Several students, however, reported problems with getting access to the unaltered source code of their e-mails. This applied to users of some browser-based mail applications, but also local e-mail-clients posed challenges. The popular Microsoft Outlook, for example, reduces the header fields of incoming e-mails to those entries crucial for displaying e-mails, thus making it impossible to retrace them, unless a certain registry value is not explicitly set. Even if this value is set, it only applies to newly received e-mails but not to those, which had already been fetched[9]. As a reaction, we published an unaltered copy of the mail on our course's site.

The baseline of problems that arose during the experiment is that inhomogeneous training environments make it close to impossible to identify all possible results of a practical task and all potential issues for single users during the exercise design. The active community in the online course helped to handle many of these issues, while the flexibility and goodwill of the teaching team made it possible to still provide a satisfying hands-on experience to the learners.

III. Evaluation of the Hands-On Exercises

As already mentioned in section II, the hands-on exercises were completely voluntary – students could reach a score of 100% without working on the practical tasks. Furthermore, these exercises were "priced" with relatively few points per handling time compared to the regular homework assignments. While a homework assignment gained 15 points for 20-40 minutes of work, the practical tasks gave 5 points for 1-2 hours of work, where especially bonus exercise 1 was quite elaborate and could easily consume 3 or 4 hours for computer novices. Nevertheless, the participation rate in the practical tasks was considerably high, as shown in table I. The relative number of participants is in relation to the number of submitted homework assignments in the course week where the respective practical exercise was due: exercise 1 took place in week 3 with 1,928 homework submissions, exercises 2 and 3 was in week 6 with 1,797 homework submissions.

TABLE I. Student participation in practical exercises

Exercise	Participants (absolute)	Participants (relative)	Average Score
#1	1,534	79.6%	4.26 / 5
#2	1,516	84.4%	2.51 / 5
#3	1,444	80.4%	3.26 / 5

We take the relatively low average score for exercise 2 in conjunction with the low exit rate from exercise 2 to exercise 3 as indicator that students were not scared off by the difficulty of exercise 2, but see the importance of the hands-on assignments. The impression given by the participation ratio – that prac- tical tasks are of major importance for the openHPI students – is also backed up by results from a survey we conducted among our participants after the course finished. The survey was taken by 1,046 students and dealt with various topics on platform functionality as well as on course design.

When we asked *"How useful do you consider the practical bonus exercises for your learning outcomes?",* approximately 4 out of 5 students answered with either "very useful" or "rather useful" (see table II).

TABLE II. Usefulness of practical exercises for learning outcome

Options	Answers (relative)
very useful	43.6%
rather useful	35.2%
rather not useful	18.9%
not useful at all	2.2%

We also asked for different kinds of learning material offered in the course (e.g. lecture videos, tutorial videos, self-tests, homework assignments, practical assignments, etc.): *"Of which kind of learning material would you like to see more (or less) items in the course content?"* While 58.1% of the answers indicated the wish for more practical exercises, only 2.4% of the students[10] said they wanted less of this kind, which is the highest approval rate for more content among all kinds of offered learning material (see table III).

[9] several articles on the web describe this issue with Microsoft Outlook and offer solutions, e.g. http://superuser.com/questions/390806/ how- can- i-view- the- entire- source- code- of- an- email- in- outlook- 2010

[10] The question had three answer options: besides "more" and "less" the participants could also choose "neutral".

TABLE III. PARTICIPANTS' DEMAND FOR MORE ITEMS OF DIFFERENT LEARNING MATERIAL TYPES

Material type	Answered "more" (relative)
Lecture videos	17.6%
Tutorial videos	51.5%
Discussion	4.8%
Readings	29.9%
Self-tests	41.0%
Homework assignments	17.0%
Practical assignments	58.1%

Furthermore, there is a considerably high number of free-text comments that explicitly expressed the wish for more practical exercises when we asked *"Which additional types of learning material or communication channels would you like to see in future courses?"* 70 survey participants left comments on practical exercises, while the next most named content type got 24 comments. Many participants also expressed their wish for more hands-on experience in the open *"I like, i wish"* question that concluded the survey.

IV. CONCLUSIONS AND OUTLOOK

This case study shows that even graded hands-on assignments for massive open online courses can be provided without the need for major adoptions to the learning platform and without the provision of a resource intensive centralized training environment infrastructure. However, these assignments inherently come with a number of limitations:

- The *assignment design* is difficult, not every instructive practical task can be fit into the challenge-response scheme (as introduced in [7]).
- The *heterogeneous training environments* (students' local computers) cause a wide range of hardly predictable problem sources. Troubleshooting on individual users' computers can be very tricky and time-consuming.
- *Customization of the exercises* per student is hardly possible (at least not in a generic way). Every student basically gets the same assignment, which is problematic in terms of cheating. This applies particularly for a social learning platform, where sharing information between users belongs to the basic concepts.

Nonetheless we managed to use the heterogeneity of the training environments to introduce at least some rudimentary customization (especially for exercise 1). Despite these limitations, the conducted experimental ap- proach can be considered successful. The high participation rates, the feedback in the discussion forum, and the interpre- tation of the survey results show a broad consent concerning the practical assignments that were introduced to the course "Internetworking with TCP/IP". Furthermore, the high demand for more exercises of this kind highlights the importance of hands-on experience for the individual learning experience and outcome. The design and implementation of suitable general-purpose computer laboratory environments for massive open online courses poses important challenges for future research and development. Some requirements for these "massive open online labs" can be derived from the experiment presented in this paper:

- The lab should provide a homogenous, but flexible environment for hands-on training sessions as well as for practical assignments.
- The lab should provide functionality that allow for automatic evaluation of a conducted practical task.
- Lab instances for individual users should be customizable. Customization should be as generic as possible and must be carried out automatically to scale for a massive number of students.
- The lab must scale for a huge number of users, possibly through the exploitation of cloud resources.

Besides the research on such a general computer lab environment, openHPI will also investigate in the integration of centralized lab environments for special purposes, i.e. server- side SQL shells for classes on database technology or coding environments for programming classes. The ability to offer classes with a high share of practical tasks and assignments will have a crucial impact on the success of competing MOOC platforms.

REFERENCES

[1] A. Gaspar, S. Langevin, and W. D. Armitage. "Virtualization technologies in the undergraduate IT curriculum", in *IT Professional,* Vol. 9(4), IEEE Computer Society, 2007, pp. 10–17.

[2] P. Hyman. "In the Year of Disruptive Education" in *Communications of the ACM,* 55(12), p. 20–22, ACM Press, 2012.

[3] J. Keller, R. Naues. "A Collaborative Virtual Computer Security Lab", in *Proc. 2nd IEEE International Conference on e-Sciece and Grid Comput- ing (e-Science '06),* IEEE Computer Society, Amsterdam, Netherlands, 2006.

[4] A. Ng, D. Koller. "The Online Revolution: Education at Scale" in *2012 conference of the American Association for the Advancement of Artificial Intelligence (AAAI-12),* Toronto, Canada, 2012. (Unpublished).

[5] G. Ulm (2012, October 18). "A Critical View on Coursera's Peer Review Process" [Online]. Available: http://gregorulm.com/ a- critical-view- on- courseras- peer- review- process

[6] K. Webb, M. Hibler, R. Ricci, A. Clements, J. Lepreau. "Implementing the Emulab-PlanetLab Portal: Experience and Lessons Learned" in *Workshop on Real, Large Distributed Systems (WORLDS),* San Francisco, USA, 2004.

[7] C. Willems, C. Meinel. "Online Assessment for Hands-on Cyber Security Training in a Virtual Lab" in *Proceedings of the Global Engineering Education Conference (IEEE EDUCON) 2012,* Marrakech, Morocco. IEEE Press, 2012.

[8] J. Postel. "Internet Protocol", IETF RFC 791, September 1981. Available: http://tools.ietf.org/html/rfc791.

[9] P. Mockapetris. "Domain Names – Implementation and Specification", IETF RFC 1035, November 1987. Available: http://tools.ietf.org/html/rfc1035.

[10] J. Klensin. "Simple Mail Transfer Protocol", IETF RFC 5321, October 2008. Available: http://tools.ietf.org/html/rfc5321

[11] N. Freed, N. Borenstein. "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", November 1996. Available: http://tools.ietf.org/html/rfc2045