HYDRA: Virtualized Distributed Testbed for DTN Simulations

Johannes Morgenroth Sebastian Schildt Lars Wolf

IBR, Technische Universität Braunschweig
Mühlenpfordstraße 23
Braunschweig, Germany
morgenroth|schildt|wolf@ibr.cs.tu-bs.de

ABSTRACT

We present and evaluate HYDRA, a virtualized testbed for realistic large-scale network simulations. While classic simulation tools only provide approximations of the protocol stack, HYDRA virtualizes nodes running a complete Linux system. Mobility models and connection management integrated into HYDRA allow for the simulation of various wireless networking scenarios. Our distributed virtualization approach achieves excellent scalability and the automated node setup makes it easy to deploy large setups with hundreds of nodes. Hardware-in-the-loop simulations are possible, using HYDRA to augment a testbed of real devices. The ability to boot a HYDRA node completely from an USB flash drive enables the user to convert temporarily unused computer resources into a testbed without the need for any complex setup.

Categories and Subject Descriptors

C.2.2 [**Computer-Communication Networks**]: Network Protocols Protocol Verification; C.4 [**Performance of Systems**]: Measurement techniques; C.2.4 [**Computer-Communication Networks**]: Distributed Systems

General Terms

Experimentation, Performance

1. INTRODUCTION

During the development of new networking protocols, extensive simulation is used to evaluate the approach chosen. Also, before deploying a large wireless network, usually a testbed, that is a small scale setup of the components used, is installed in an controlled environment in order to anticipate potential problems before deployment. Even the setup and configuration of such a small scale version of a realistic deployment is a time consuming and error-prone task. Large scale environments for testing protocols and implementations are even more challenging to setup. In practice often those setups are replaced by large scale simulations which do not run production software but only conceptual approximations specifically crafted to run on top of a simulation engine. This has the significant drawback, that apart from simplified assumptions about the environment, the protocol implementations in the simulator are also often not identical to the real world implementation. This can severely limit the validity of simulation results when trying to make predictions about real-world performance.

In this paper we present HYDRA, a distributed testbed that runs virtualized nodes with a full operating system and protocol stack. HYDRA provides a highly customizable automated node setup and a very extensible plugin architecture. We use HYDRA as simulation framework in our research projects in the area of delay tolerant networks, however HYDRA is not specific to this field and can be used for any kind of generic network simulation.

In OPTraCom[1] we study how DTNs can be used for local public transport applications and distributed pollution measurement within the city area. In order to evaluate new concepts and thoroughly test developed software before deploying them on embedded hardware in buses and streetcars, we evaluate and stress test the software using HYDRA.

The NTH project IT-ecosystems[2] deals with the management and controllability of complex systems of systems. It facilitates a human centric view of complex distributed technical systems, where humans are not only beneficiaries of offered services but also act as agents in a DTN network transferring data between different otherwise disconnected systems. To be able to evaluate the developed communication paradigms reasonably, simulations of sufficiently large scenarios with tens to hundreds of mobile agents are necessary. HYDRA allows testing of the real software under controlled conditions in a large testbed.

The remainder of this paper is organized as follows: In section 2 we will give an overview of related work. In section 3 we present our software architecture. The mobility and connectivity modules included with HYDRA are detailed in section 4. We then give the details of our experimental setup in section 5 followed by a performance and scalability evaluation in section 6. Finally we give our conclusion and propose directions of future work.

WiNTECH'10, September 20, 2010, Chicago, Illinois, USA.

[1] http:/www.optracom.de/
[2] http://www.it-ecosystems.org/

2. RELATED WORK

Network simulators are the tool of choice for developing novel network protocols and evaluating them in different wireless networking scenarios. Common variants, among others, are ns-2, GloMoSim, OMNet++ and JiST/SWANS. Ns-2[2] uses a dual-language approach, where the simulator is written in C and simulations are set up using OTcl, a object oriented variant of the TCL scripting language. Ns-2 became the most widely used simulator in the MANET community. GloMoSim[27] is a simulator build upon the C-based Parsec language specifically designed for parallel simulations. Development of the open GloMoSim version ceased in 2002. The technology was continued in form of the commercial Qualnet simulator. OMNet++[25] is a modular discrete event simulation framework written in C++. OMNet++ is free for non-commercial use and is mostly used for network simulation. JiST[4] is a Java-based discrete event simulation engine that augments the Java execution model with simulation time semantics through the use of tagging interfaces and bytecode rewriting. SWANS[4] is an AdHoc network simulator build upon JiST.

While the tools presented so far aim to be generic network simulators, there are also more specialized simulation environments. A premier choice for DTN simulations is the Opportunistic Networking Environment (ONE) simulator[15]. The ONE is a Java based simulator offering a broad set of DTN simulation capabilities. It supports different mobility models and can import traces from external sources for evaluation of routing protocols and applications. The common DTN protocols are already implemented in this simulator and an interface for external modules is available.

The advantage of pure simulation tools is, that they are able to simulate large setups that would not be practical to test in reality at acceptable speeds. Also, programming against a simulator is usually easier, since one does not have to deal with the complexities of hardware, an underlying operating system or a full blown network communication stack. The disadvantage is, that the protocol layers supplied by the simulators are only an (often rather rough) approximation of the protocols simulated. Quite often, for reasons of speed and simplicity, only the semantics, but not the wire format of a given protocol is simulated. Depending on the simulation requirements and the simulator capabilities, lower layers, such as MAC or radio propagation models, might not be simulated at all.

A problem with those protocol approximations is, that it is not quite clear to what extent simulation results can be transferred to a real world implementation. For instance, Cavin et. al. showed in [7] that even for simple scenarios simulations outputs differ significantly between several established MANET simulators. Similar results are reported in [11].

To alleviate this problem network emulators can be used. The difference between emulation and simulation is, that in an emulation at least parts of the system are real. For example, OppBSD[14] is a full FreeBSD IP stack that can be used as module in the OMNet++ simulator. JiST/Mobnet[18] is based on JiST and extends and replaces SWANs to allow for using JiST/Mobnet as network emulation interacting with real testbeds. This is achieved by patching real network interfaces through Mobnet and by supporting the correct on-wire frame formats in the JiST emulation.

More accurate simulations can be achieved by including more production code down to the operating system in the the emulation. For wireless sensor networks there are even emulators which go down to the hardware level. The emulator for TinyOS is TOSSIM[19]. A TOSSIM simulation includes the whole TinyOS operating system and runs the application unmodified on top of it. It even emulates hardware resources such as AD converters found in TinyOS compatible hardware. The radio model can operate on the bit level, allowing the correct simulation of link biterror rates. The equivalent for Contiki[10], another sensor network OS, is Cooja[22]. Cooja goes even further: Instead of just running the operating system and emulating a few abstracted hardware components, Cooja has the ability to emulate a MSP430 CPU, which is a commonly used microcontroller for Contiki based sensor nodes.

Emulab[26] is a software framework for the operation of large scale networking testbeds. It combines physical nodes with advanced network emulation capabilities. In contrast to our approach Emulab needs a fixed infrastructure: It requires dedicated machines which run the Emulab software distribution and manageable switches from a range of supported devices. Experiments are set up automatically from an ns-2 compatible topology specification. Recently Emulab aquired experimental support for operating system virtualization by using FreeBSD Jails[13]. Because FreeBSD Jails do not offer hardware virtualization, it is not possible to use different operating systems using this technique, but as long as the experiment can be run on FreeBSD this allows Emulab to multiplex simulation hosts.

Pure testbeds are installations of real hard- and software for evaluation purposes. They offer the greatest degree of realism at the cost of flexibility. DieselNet [28] is a testbed for delay tolerant networking with 40 buses. Each of them is equipped with a small computer and a harddrive for persistant storage. Additionally a so-called "Throwbox" device is used for fast deployment of additional stationary routers. In [8] a DTN demonstrator was presented which controls the WiFi interface of several nodes. The goal was to evaluate the performance of the IBR-DTN software stack operating in a small scale heterogeneous environment. However, the setup itself is not trivial and does not scale well if many nodes communicate with the same frequency in a shared area.

Sometimes the lines between emulation and testbeds are blurred. In general, pure network simulations are the most flexible tools but their results have to be analyzed carefully with regard to their validity and transferability to the real world, while emulation techniques achieve better accuracy [21]. A physical testbed is the most accurate approximation of real world applications.

Hydra employs virtualization techniques in order to multiplex a single machine to emulate a number of virtual nodes. This approach causes some overhead. In [20] Macdonnel and Lu showed that overhead can be as low as 6 % for CPU intensive jobs and 9.7 % for jobs dominated by I/O for x86 based virtualization solutions. The evaluation in [9] reports an overhead of significantly less than 20 % for most CPU bound applications running under virtualization. The paper also reports only a small influence on host performance, when VMs are idle.

In contrast to most other simulation environments Hydra runs unmodified applications on top of linux. Thus it can be used to asses the performance of a concrete implementation of a network protocol rather than simulating only an ab-

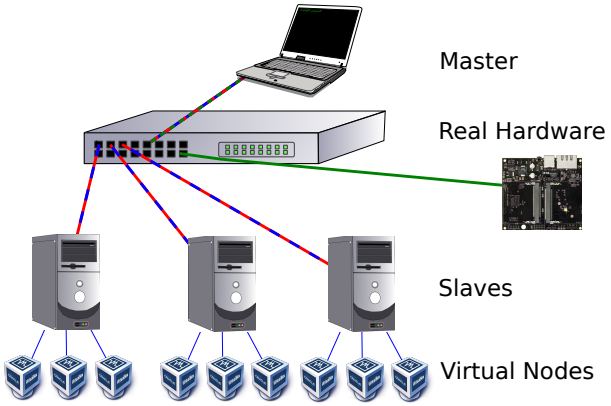


Figure 1: HYDRA standard hardware setup

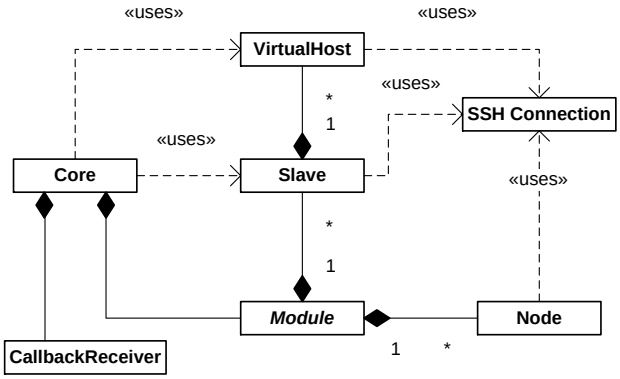


Figure 2: UML diagramm of HYDRA's classes

stract reimplemention. This makes HYDRA a valuable tool for the development of networking software. In contrast to netlab, which also runs real software and stacks, HYDRA is much more lightweight and can be setup in matter of minutes instead of dedicating a complex infrastructure to it.

3. HYDRA ARCHITECTURE

3.1 Hardware setup

Figure 1 shows a typical hardware setup for HYDRA simulations. We differentiate between three types of nodes. The master runs the emulation logic. It controls and monitors the other nodes and triggers events like connection up and down. A slave is a host for virtual nodes. A virtual bridged network interface connects the virtual nodes to the hardware switch. Note, that real hardware can be transparently added to the HYDRA setup allowing for hardware-in-the-loop verification. If desired the network used for controlling the slave and virtual nodes can be separated from the network used by the applications running on the Virtual Nodes using measures such as VLANs.

On behalf of the master, the slave is preparing and configuring the image files and virtual machine configuration for its own virtual nodes. We created a bootable Ubunutu based USB flash drive which included preconfigured software components needed for the slave. This enables a fast adaptation of unused hardware for temporary large scale runs. Virtual nodes are run by the VirtualBox virtualization software [23] and use a standard OpenWRT [1] image for x86 architecture. OpenWRT is a Linux distribution specifically designed for wireless networking applications. It has very low CPU and memory requirements, so we are able to run many nodes on a single standard PC, while still being able to use virtually all Linux software. Additionally, OpenWRT is used on several embedded networking hardware platforms and thus provides an excellent platform for developing and testing of software for embedded wireless devices.

3.2 Software architecture

The software of this emulation setup is written in Python and has a plugin-like architecture to serve different needs. Figure 2 shows the main classes of the implementation and the relations between them. A simulation starts with the `Core` which creates controllable objects for slaves and virtual hosts. Both classes uses the Secure Shell Protocol (SSH) wrapped into the `SSHConnection` class to execute actions on the corresponding systems. We have chosen this way of controlling, because SSH is ubiquitous and very robust. The depicted class `Module` is representative for any plugin module used in HYDRA. One task of such a module is to provide a list of node configurations (as `Node` classes) which are used to create virtual nodes during the setup. If the generic setup of all virtual nodes is done and the `CallbackReceiver` has recognized callback events of all nodes, this module takes over control of the emulation process and can control connections between nodes and the behavior of the nodes.

Two plugin modules are available so far. The default module generates a given number of unique virtual nodes and distributes them to the slave instances. Additional, a choosable mobility pattern provides a virtual movement with virtual meetings between the nodes. The virtual movement is simulated by static connections, playing a predefined trace or a random walk mobility model for infinite emulation (see section 4). The second module realizes a special setup for DTNs. It generates several nodes out of a predefined logfile of a modified ONE Simulator [15] and replays the connectivity history. In addition to connection up and down events, the generation of bundles is interpreted and executed on the virtual nodes. With this module an evaluation of routing algorithms is possible and comparable to the results of the ONE Simulator.

To manage the whole setup, a bunch of utility modules provide access to virtual machine and node control functions. With them it is possible to prepare and customize the given software image for running as a distinct virtual node on a slave. Figure 3 shows the general flow of the HYDRA software. The very first action of the simulator is to define the environment for the run by reading setup-specific configuration files and setup a control connection to each configured slave which is used to send commands and upload files. In the next step, a prototype image is generated out of a image file specified in the setup configuration and custom preparation scripts. Normally the creation of the prototype image involves the installation of additional software needed on the virtual nodes and configuring everything that is not node-dependant. Once the upload of the prototype image to the slaves is done, the second preparation of the images is started. This includes making a node unique by setting the hostname and the network configuration parameters and embedding a callback mechanism which connects the mas-

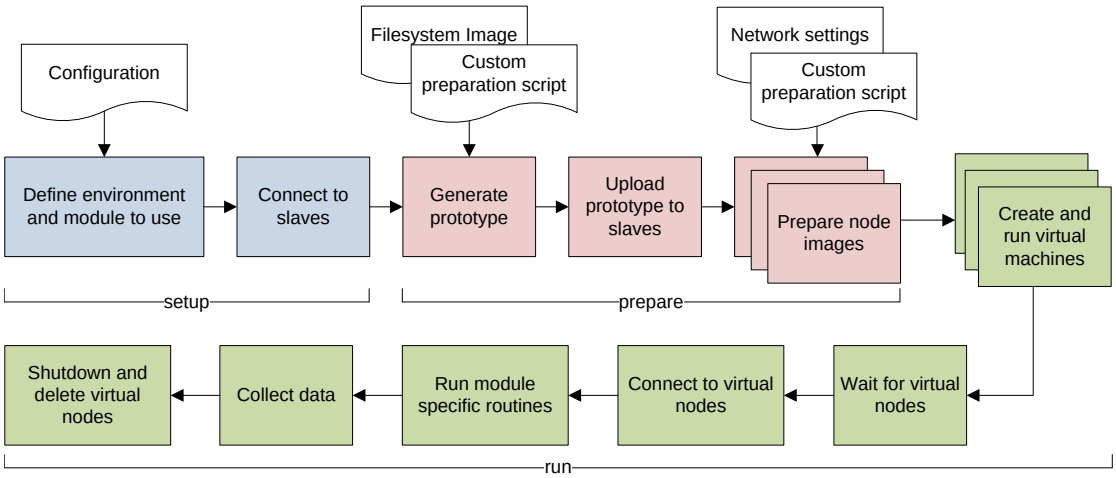


Figure 3: HYDRA **general flowchart with three stages: setup, prepare, and run**

ter after the virtual node booted up successfully. To shorten this step, we do this on all slaves in parallel.

Right after the preparation stage, the run stage of the emulation is started. This creates and configures the virtual machines in VirtualBox and starts each virtual node on the assigned slave. As in the preparation step before we do all this in parallel. After bootup each virtual node executes its callback script embedded in the init process. This script sends a ready signal to the master, which is waiting until the signals of all virtual nodes are received. Finally, a control connection is established to each virtual node. When this process is finished, the run routine of the selected module is executed and takes control over the whole emulation. This module has complete control over all virtual instances and can trigger any simulation specific events on the virtual nodes. By default an movement module is started which emulates connection up and down events between different nodes. After the simulation is completed or interupted, the cleanup phase begins. This includes the collection of generated data on the nodes and finally the shutdown of the virtual nodes on all slaves and the deletion of the virtual machine configurations.

For easy monitoring and debugging HYDRA provides two mechanisms. To have a central place to monitor and log the activity of all nodes, each node is automatically configured to report their syslog messages to a remote syslog daemon on the master. Additionally, each virtual node is accessible through a virtual remote desktop protocol console (VRDP). This enables debugging of wrong configurations or defect images during the startup phase and enables live manipulation of a single virtual node.

4. MOBILITY AND CONNECTIVITY

In order to simulate wireless networking scenarios it is essential to have some sort of topology control. HYDRA contains functions to explicitly allow or forbid packets to travel between an ordered pair of virtual nodes. This is realized by the use of iptable rules on the virtual nodes themselves. While this is putting part of the simulation semantics *inside* the virtual nodes, the advantage is scalability: Setting the iptable rules through SSH consumes almost no CPU time on the master and the processing overhead for executing the iptable rules is distributed evenly through all slave nodes. A centralized approach, where all virtual nodes are tunnelled to a central point managing the network topology, would increase latencies and the load on the node managing the topology.

HYDRA's topology control primitives are used by mobility or logfile playback modules. HYDRA includes a bounded Random Walk as standard mobility model. Area, range, speed, time between changes of direction and resolution of the model can be configured. The velocity of nodes is distributed uniformly between a minimum and maximum speed. However, instead of reimplementing every standard mobility model in HYDRA, the preferred way is to playback movement or connectivity information from specialized tools, such as SUMO[17] or realworld traces like those found in the CRAWDAD archives[16]. For this HYDRA includes two basic topology controllers, which can be easily extended to support arbitrary formats: The Connectivity topology controller does not care for node positions or ranges, instead it expects events describing at which timestamp a connection between two nodes goes up or down. We implemented a module that uses the Connectivity controller to playback traces that where obtained by running a simulation in the ONE. The other controller is the generic mobility controller: It expects events describing the position of nodes at certain timestamps. This information is used to derive the topology based on radio range.

The playback modules have the ability to compress or stretch time by a constant factor. The nodes in HYDRA always run in real-time, that is, there is no concept such as "simulation time". This means for simple scenarios the virtual nodes might spend a lot of time idling between interactions. Speeding up mobility traces allows to get qualitatively the same results in a shorter time. Similarly, when the virtual nodes are overloaded, dilating time lengthens contact times and gives the nodes more time for processing. It should be noted, that without further measures speeding up the simulation by a factor of 2 means effectively halving the bandwidth and vice versa.

To get a more fine grained control over link parameters

netem [12] could be used. In cooperation with Linux' iptables and QoS facilities *netem* is able to limit the bandwidth of connections and introduce packet loss and delaying. We already included support for netem into the OpenWRT images we use, however as of yet there is no framework in HYDRA to centralize the control and configuration of *netem*.

5. EXPERIMENTAL SETUP

For all experiments the master node was a standard PC with an Intel Pentium 4 CPU running at 3.20 GHz, 2 GiB RAM and a 160 GB SATA harddisk. The CPU uses hyperthreading but does not support Intel VT virtualization extensions. Ubuntu 9.10 (kernel version 2.6.31) was used as operating system. For experiment 1, one equivalent machine was used as slave node. For experiment 3, two equivalent machines where used as slaves, with the master acting as a third slave in addition to managing the whole simulation. For experiment 2 we used a pool of 10 computers as slaves. Those machines were equipped wit a dual-core Intel Pentium 4 D CPU running at 3.2 GHz with 2 GiB of RAM. VT-x virtualization extension are supported by this CPU. The slave nodes where booted with HYDRA's customized Linux distribution directly from USB flash drives as explained in section 3.1. The internal harddisk has not been used.

The principal network setup for all experiments was equal and is depicted in figure 1. All slave nodes had Suns VirtualBox, version 3.1.6 installed. All virtual nodes where configured to a separated 192.168.56.0/24 subnet. The virtual nodes used a standard image of OpenWRT 8.09 (Kamikaze) and where provided with 32 MiB RAM and one virtual NIC. The NIC was bridged to an ethernet interface on the host machine which was connected to the other slaves and the master through an unmanaged ethernet switch.

Experiment 1: Slave scalability

In order to show the principal scalability of our approach, we used a simple testcase to determine the responsiveness of virtual nodes. We used one slave node and gradually increased the number of virtual nodes hosted by the slave. The virtual nodes where idling to make sure we only measure overhead from our setup. On the master, all virtual nodes are monitored by SmokePing[3], which probed all virtual nodes and the slave with 20 56 Byte ICMP Echo packets every 60 seconds. As a baseline we included one physical node into our simulation, which is also monitored by the Master, to make sure that any changes in responsiveness where not caused by load on the Master. The physical node chosen was an Ubiquiti RouterStation Pro board, which is the premier development platform for IBR-DTN[8]. The RouterStation Pro includes a 680 MHz MIPS 24k core and 128 MiB of RAM. The integrated Ethernet port was used to connect to the test network.

Experiment 2: Mobility enabled DTN simulation

To test HYDRA's capability to run a big simulation with custom software we setup an emulation run with the IBR-DTN software. The goal is to observe the behavior of each single node in a realistically sized DTN setup. The DTN stack software itself reports connections and bundle deliveries to the syslog. We ran 100 virtual nodes with Random Walk mobility. The area was 2000 m × 2000 m, the speed of nodes was uniformly distributed between 0.75 m/s and 1.25 m/s. Radio range was set to 100 m, and nodes changed direction every 180 s. The mobility model was simulated with a resolution of 5 s. All DTN daemons continuously logged their current neighbors, and a data packet has been injected into the network, measuring the time until it reaches its destination.

[3]http://oss.oetiker.ch/smokeping/

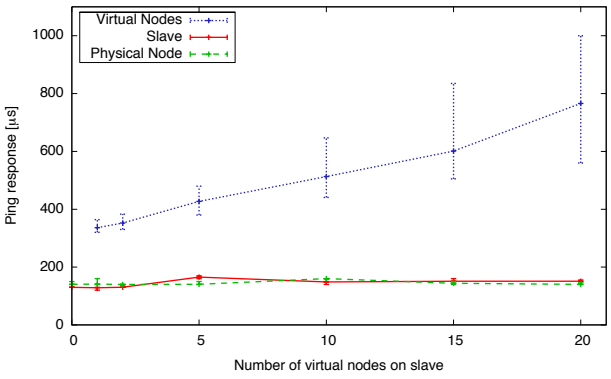


Figure 4: Ping response times

Experiment 3: Repeatability of simulations

Being a testbed, conceptually experiments are not completely repeatable in HYDRA. In contrast to a pure simulator, where starting the simulation with the same random seed yields exactly the same results, with HYDRA running instances of non-realtime operating systems under virtualization on top of another non-realtime OS can lead to different characteristics on each run. However, on a macroscopic level results for the same simulation setup should be the same at least qualitatively and preferably have a high degree of quantitative consistence as well. To evaluate the stability of HYDRA's simulation results we performed the following experiment: We simulated 18 nodes in an area of 1200 m × 300 m using a connectivity trace generated by the random walk mobility module. Radio range was set to 100 m, node velocity was uniformly distributed between 0.75 m and 1.25 m. Nodes changed direction every 180 s. The resolution for the mobility model was set to 2 s. After the simulation has been running for 300 s, a packet of size 1 MiB is injected. The payload was injected at node 11 which was at that time at coordinates 141:256. The destination was node 7, which was at injection time at coordinates 1135:131. The time until the packet arrives at the destination was measured and the route the packet travelled was logged. We compared the results to a similar setup running within the ONE. This configuration was run on a single slave running all 18 nodes, two slaves with 9 virtual nodes per slave and 3 slaves with 6 virtual nodes per slave. Additionally, for each number of slaves, we did two runs where we speed up movement of nodes by a factor of 2 and 4 as explained in section 4.

6. EVALUATION

Experiment 1: Slave scalability

The ping setup shows the scalability of our approach for a single slave node. As the virtual nodes where idle during this measurement, we are basically measuring overhead for

virtualizing the nodes and the network. Figure 4 shows the ping response times for the physical node, the slave hosting the virtual machines and the virtual nodes themselves. The figure shows average roundtrip time (RTT) as well as maximum and minimum RTT. It can be seen that the RTT for the physical and slave Node stay around 180 μs for any number of virtual nodes with negligible jitter. This shows that the master is not overloaded with controlling and monitoring the virtual nodes and that the slave hosting the virtual nodes is not overloaded either. Ping response times for virtual nodes increases from 340 μs with 1 virtual node to 770 μs with 20 virtual nodes. Also note that the jitter increases significantly from 44 μs with 1 node to 440 μs with 20 nodes. On average on the slave each additional VM takes 56.3 MiB RAM and consumes around 4 % CPU when idling, leading to 1126 MiB RAM usage and around 80 % CPU usage (200 % is the maximum for the hyperthreading single-core CPU) for a run with 20 Virtual Nodes.

While, of course, the amount of nodes which can be used on a slave is greatly dependent on the amount of resources used by the application running on the virtual nodes this measurement shows that from virtualization overhead alone it is feasible to host 10 to 20 VMs on a single slave. Also, because in its standard configuration OpenWRT does not need much RAM, the number of virtual nodes seems to be more CPU bound than memory bound. Depending on the application the jitter has to be taken into account, e.g. when testing time synchronization schemes, however in absolute terms the response times and jitter are still very low compared to times that are to be expected in internet applications which are in the area of 10ths of ms.

Experiment 2: Mobility enabled DTN simulation

We injected a packet into the DTN network to test the mobility model and DTN operation. The packet started at node 0 with node 1 as destination. A simple epidemic routing [24] scheme was used. The situation after injection is depicted in figure 5(a). The figure shows the radio range for node 0. In this setup all nodes have an equal radio range. After 20 min the packet reached the target node. The shortest path from target to destination leads through nodes $28 \rightarrow 4 \rightarrow 33$. Figure 5(b) shows the topology at the time the packet is received by the target. Again we plotted the radio range of node 0 as reference. Purple nodes are already in possession of the test packet.

This experiment shows that HYDRA is capable of simulating meaningful experiments, of such a size that would be very difficult to handle when using a completely manual approach. The required effort for setting up this experiment was small: Setting the correct Random Walk parameters and the installation of the IBR DTN software is achieved through standard HYDRA configuration files. The slaves where booted using our live USB Linux, without any further configuration. The only manual task was giving the master a list with the IP addresses of all slaves.

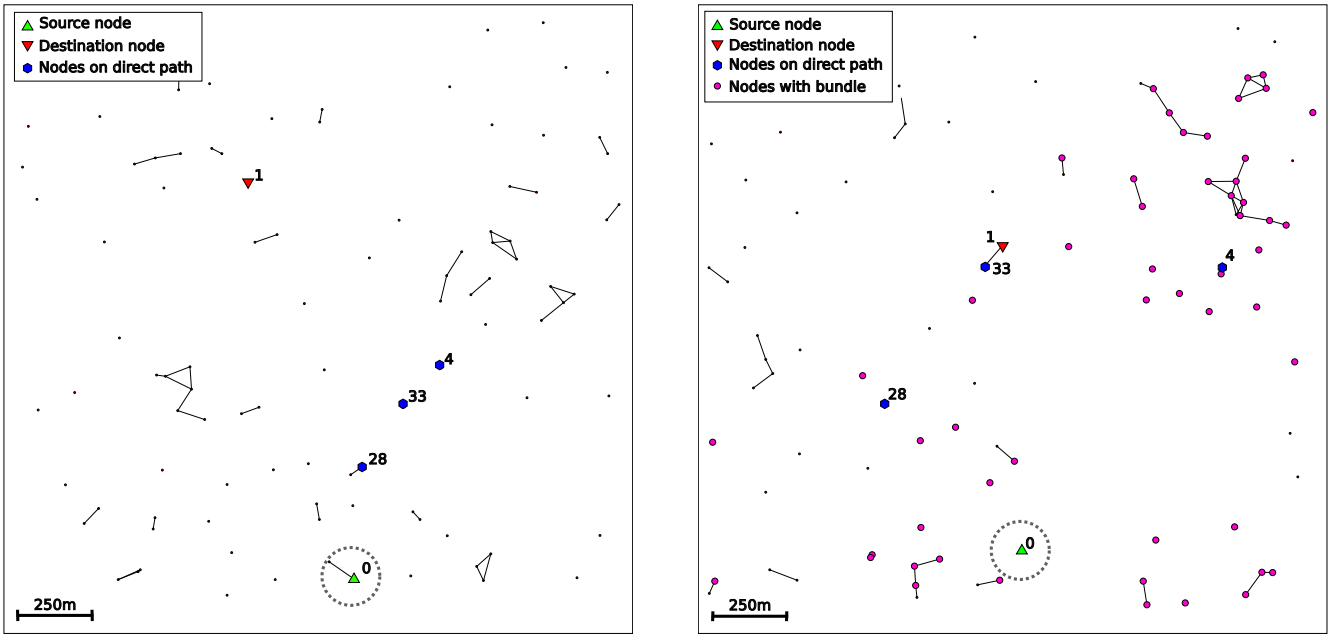
Experiment 3: Repeatability of simulations

The results of the robustness test are given in table 1. In the table n is the number of slaves used, f is the factor by which the mobility model has been speed up, t is the time in seconds until the packet arrives at the destination and $t_n = f \cdot t$ is the normalized time to make runs comparable. The last column shows the route the packet has taken. The reference run simulated using The ONE took 18 min and 53 sec of simulation time until the packet arrived at its destination, the route taken by the simulation was $11 \rightarrow 3 \rightarrow 5 \rightarrow 16 \rightarrow 9 \rightarrow 2 \rightarrow 1 \rightarrow 7$. It can be seen that with regard to the route taken the results from all runs distributed on 2 or 3 slaves were correct. Additionally the run on a single slave with normal speed was correct compared to the reference simulation from the ONE. However the packet delivery times reported in the table are between 17 min:20 s and 19 min:28 s. This is not an indication of unstable simulation results, but rather due to our measurements method: When logging the network we used information from the syslogs of the virtual nodes. Unfortunately, after being started and synchronized to the host clocks, the virtual machine clocks can drift apart. This is a common problem under virtualization [3]: The virtual clocks can loose ticks when the host machine is busy. In HYDRA this happens when many VMs are busy, especially when all virtual nodes are booting simultaneously. To get better results a NTP client could be installed in each virtual node or the VirtualBox software suite for Linux guests could be used. We observed differences up to 3 min between different virtual nodes at the end of a simulation. With regard to this, all normalized times for the correct runs reported in table 1 can be considered the same within our measurement accuracy.

The runs with increased speed on a single slave fail to use the correct route, even though they get the packet delivered eventually. This is an indication that the slave was overloaded. Increasing the playback speed of the connectivity log also means that contact times are shorter. The virtual nodes are not able to perform all needed transfers in the shortened time. We also found that in these cases the results are less stable, e.g. the route chosen changes on subsequent runs using the same setup.

The following effects explain simulation instabilities: Imagine a very short contact time of 2 s. This is the shortest possible contact time in single speed when the mobility model has been rendered with 2 s resolution. For the higher speed simulations short contact times are even more probable as all contact times are halved or quartered. The IBR DTN implementation has a neighbour discovery mechanism which is based on a broadcast sent every 1 s by each node. So for a 2 s contact window two nodes will see each other at anytime between 0 and 1 after contact, leaving 1 to 2 seconds transfer time, which is a difference of 100% between those extreme values. So even when slaves are not overloaded, it is possible that HYDRA runs will differ between different runs. This will not happen with a pure simulator, because if both nodes see each other 1.23 s after contact, this will happen at exact the same point in simulation time in every run (assuming the simulators random number generator is seeded with the same number). When a slave is overloaded, such as is the case for the $\times 2$ and $\times 4$ runs on a single slave, this problem can be amplified by the fact that the virtual nodes are loosing ticks, which means that in reality the discovery interval (and all other times periods the virtual node keeps track of) can get longer than intended: While the virtual node thinks it has waited only 1 s, a longer timespan has passed in reality, e.g. on the host which manages the connectivity between nodes.

In light of these measurements, we conclude that in general the results of a HYDRA run are not affected by the number of slaves used, as long as no slave is overloaded. In re-



(a) Topology after packet injection

(b) Test packet distribution after successful delivery

Figure 5: Experiment 3 topology

n	f	t [*mm:ss*]	t_n [*mm:ss*]	route
1	1	17:53	17:53	$11 \to 3 \to 5 \to 16 \to 9 \to 2 \to 1 \to 7$
1	2	16:34	33:05	$11 \to 3 \to 5 \to 15 \to 7$
1	4	05:30	22:00	$11 \to 14 \to 4 \to 12 \to 3 \to 5 \to 7$
2	1	17:20	17:20	$11 \to 3 \to 5 \to 16 \to 9 \to 2 \to 1 \to 7$
2	2	09:04	18:05	$11 \to 3 \to 5 \to 16 \to 9 \to 2 \to 1 \to 7$
2	4	04:46	19:04	$11 \to 3 \to 5 \to 16 \to 9 \to 2 \to 1 \to 7$
3	1	19:10	19:10	$11 \to 3 \to 5 \to 16 \to 9 \to 2 \to 1 \to 7$
3	2	08:41	17:22	$11 \to 3 \to 5 \to 16 \to 9 \to 2 \to 1 \to 7$
3	4	04:52	19:28	$11 \to 3 \to 5 \to 16 \to 9 \to 2 \to 1 \to 7$

Table 1: Experiment 3, packet delivery times

ality of course it can be bit tricky to find the point where the simulation has glitches due to overloaded slaves. Thus, as with normal simulators we recommend repeating simulations several times and carefully compare and examine the results and try to explain any differences.

7. CONCLUSIONS

We have presented Hydra, a virtualized testbed supporting large scale setups using virtual nodes running a complete Linux operating system and protocol stack. The process of setting up testcases for verifying network software implementations is simple and by providing a bootable live distribution for Hydra slaves it is easy to convert temporarily unused machines into a testbed. By running real operating systems and application software, the ouptut of Hydra simulations can be considered more realistic than results from purely synthetic simulators. Even though by design Hydra can not be completely deterministic, analysis has shown that Hydra's results are robust and repeatable on a macroscopic level even when using different setups. The source code of the Hydra system will be made available under an open source license.

There are still opportunities for optimizing and extending Hydra: The current setup uses full hardware virtualization. Network I/O performance could be improved by using the VirtualBox "virtio" network card which needs a para-virtualized virtio network driver for the guest. In future versions of Hydra we plan to add support for *netem* [12] (see section 4). While the proposed usage of *netem* is distributed across all virtual nodes, a more strict and synchronized control over link parameters is possible using a centralized approach. By routing all connections through a central node, network emulators such as *nistnet*[6] could be used. This would provide a more accurate link characteristic simulation at the cost of scalability. For applications and protocols based on geographic data, a virtual GPS can be used to generate simulated positions for each virtual node. With this mechanism GPS based routing algorithms could be simulated in Hydra. Instead of virtualization, a hardware emulator such as QEMU[5] could be integrated into Hydra. This is the only way to emulate software running on special nodes like sensors or embedded hardware at the cost of reduced performance.

8. ACKNOWLEDGEMENTS

This work has been partially supported by EFRE (European fund for regional development) project OPTraCom (W2-800-28895) and by the NTH School for IT Ecosystems. NTH (Niedersaechsische Technische Hochschule) is a joint university consisting of TU Braunschweig, TU Clausthal, and Leibniz Universitaet Hannover.

9. REFERENCES

[1] OpenWRT. `http://www.openwrt.org/`.

[2] The Network Simulator - ns-2. `http://www.isi.edu/nsnam/ns/`.

[3] Timekeeping in VMware Virtual Machines. Technical report, VMware, Inc., 2008.

[4] R. Barr, Z. Haas, and R. van Renesse. JiST: An efficient approach to simulation using virtual machines. *Software: Practice and Experience*, 35(6), 2005.

[5] F. Bellard. QEMU, a fast and portable dynamic translator. In *ATEC '05: Proceedings of the annual conference on USENIX Annual Technical Conference*, pages 41–41, Berkeley, CA, USA, 2005. USENIX Association.

[6] M. Carson and D. Santay. NIST Net: A Linux-based network emulation tool. *SIGCOMM Computer Communication Review*, 33(3), Jul 2003.

[7] D. Cavin, Y. Sasson, and A. Schiper. On the accuracy of MANET simulators. *POMC '02: Proceedings of the second ACM international workshop on Principles of mobile computing*, Oct 2002.

[8] M. Doering, S. Lahde, J. Morgenroth, and L. Wolf. IBR-DTN: An efficient implementation for embedded systems. In *CHANTS '08: Proceedings of the third ACM workshop on Challenged networks*, pages 117–120, New York, NY, USA, 2008. ACM.

[9] P. Domingues, F. Araujo, and L. Silva. Evaluating the performance and intrusiveness of virtual machines for desktop grid computing. *IEEE International Symposium Parallel & Distributed Processing, 2009*, pages 1–8, 2009.

[10] Dunkels. Contiki - a lightweight and flexible operating system for tiny networked sensors. *Local Computer Networks, 2004. 29th Annual IEEE International Conference on*, pages 455 – 462, 2004.

[11] F. Haq. Abstract Simulation vs. Emulation: Evaluating Mobile Ad Hoc Network Routing Protocols. In *Porceedings IWWAN 2005*, 2005.

[12] S. Hemminger. Network Emulation with NetEm. In *Linux Conf Au*, April 2005.

[13] M. Hibler, R. Ricci, L. Stoller, J. Duerig, S. Guruprasad, T. Stack, K. Webb, and J. Lepreau. Large-scale virtualization in the emulab network testbed. *USENIX Annual Technical Conference, Boston, MA*, 2008.

[14] Institute of Telemtics - University of Karlsruhe. OppBSD - A FreeBSD Network Stack integrated into OMNeT++. `https://projekte.tm.uka.de/trac/OppBSD/`.

[15] A. Keränen, J. Ott, and T. Kärkkäinen. The ONE simulator for DTN protocol evaluation. In *Simutools '09: Proceedings of the 2nd International Conference on Simulation Tools and Techniques*, pages 1–10, ICST, Brussels, Belgium, Belgium, 2009. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[16] D. Kotz and T. Henderson. CRAWDAD: A Community Resource for Archiving Wireless Data at Dartmouth. *IEEE Pervasive Computing*, 4:12–14, 2005.

[17] D. Krajzewicz, M. Bonert, and P. Wagner. The open source traffic simulation package SUMO. *RoboCup 2006 Infrastructure Simulation Competition*, 2006.

[18] T. Krop, M. Bredel, M. Hollick, and R. Steinmetz. JiST/MobNet: combined simulation, emulation, and real-world testbed for ad hoc networks. *WinTECH '07: Proceedings of the second ACM international workshop on Wireless network testbeds, experimental evaluation and characterization*, Sep 2007.

[19] P. Levis, N. Lee, M. Welsh, and D. Culler. TOSSIM: accurate and scalable simulation of entire TinyOS applications. *SenSys '03: Proceedings of the 1st international conference on Embedded networked sensor systems*, Nov 2003.

[20] C. Macdonell and P. Lu. Pragmatics of virtual machines for high-performance computing: A quantitative study of basic overheads. *Proc. of the 2007 High Performance Computing and Simulation Conf*, 2007.

[21] E. Nordström, P. Gunningberg, C. Rohner, and O. Wibling. Evaluating wireless multi-hop networks using a combination of simulation, emulation, and real world experiments. *MobiEval '07: Proceedings of the 1st international workshop on System evaluation for mobile platforms*, Jun 2007.

[22] F. Osterlind, A. Dunkels, J. Eriksson, N. Finne, and T. Voigt. Cross-Level Sensor Network Simulation with COOJA. *Local Computer Networks, Proceedings 2006 31st IEEE Conference on*, pages 641 – 648, 2006.

[23] Sun Microsystems. VirtualBox. `http://www.virtualbox.org/`.

[24] A. Vahdat and D. Becker. Epidemic Routing for Partially Connected Ad Hoc Networks, 2000.

[25] A. Varga and R. Hornig. An overview of the OMNeT++ simulation environment. pages 1–10, 2008.

[26] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar. An integrated experimental environment for distributed systems and networks. *SIGOPS Operating Systems Review*, 36(SI), Dec 2002.

[27] X. Zeng, R. Bagrodia, and M. Gerla. GloMoSim: a library for parallel simulation of large-scale wireless networks. *PADS '98: Proceedings of the twelfth workshop on Parallel and distributed simulation*, Jul 1998.

[28] X. Zhang, J. Kurose, B. N. Levine, D. Towsley, and H. Zhang. Study of a Bus-Based Disruption Tolerant Network: Mobility Modeling and Impact on Routing. In *Proc. ACM Intl. Conf. on Mobile Computing and Networking (Mobicom)*, pages 195–206, September 2007.