OFCProbe: A Platform-Independent Tool for OpenFlow Controller Analysis

Michael Jarschel, Christopher Metter, Thomas Zinner, Steffen Gebert, Phuoc Tran-Gia
Institute of Computer Science University of Würzburg, Germany.
Email: {michael.jarschel,christopher.metter,zinner,steffen.gebert,trangia}@informatik.uni-wuerzburg.de

***Abstract*—Controller performance and behavior are key to the operation of an SDN network. Therefore, choosing the right controller implementation and corresponding set of applications is essential. In order to facilitate this decision we previously introduced a tool for controller performance analysis called ”OFCBenchmark”. In this paper, we present ”OFCProbe” a platform-independent and extended re-design of our original approach. We describe the new architecture and explain the implemented features. Finally, we provide some sample results to illustrate the kind of investigations that can be performed using the tool.**

***Keywords*-OpenFlow Controller, SDN, Performance Analysis**

I. Introduction

The key component in the Software Defined Networking architecture is the controller or ”networking operating system”. The controller provides a platform for the operation of diverse network control and management applications. However, little is known about the stability and performance of current controller applications, which is a requirement for a smooth operation of the network.

In case of OpenFlow, the currently most popular realization of SDN, the controller is not specified by the standard. Its performance depends on the specific implementation. As a consequence, some controllers are more suitable for certain tasks than others. Choosing the right controller for a task requires a thorough analysis of the available candidates in terms of system behavior and performance.

In this paper, we present the extended platform-independent and flexible OpenFlow controller performance analyzer ”OFCProbe” as a follow up to our previous work with ”OFCBenchmark”. The new tool features a scalable and modular architecture that allows a deep granular analysis of a controller's behavior and characteristics. It allows the emulation of virtual switches that each provide sophisticated statistics about different aspects of the controller's performance. The virtual switches are arrangeable into topologies to emulate different scenarios and traffic patterns. This way a detailed insight and deep analysis of possible bottlenecks concerning the controller performance or unexpected behavior is possible. Key features of the re-implementation are a more flexible, simulation-style packet generation system as well as Java Selector-based connection handling. In order to highlight the tool's features, we perform some experiments for the Nox and Floodlight controllers in different scenarios.

The remainder of this paper is structured as follows. In Section II, we discuss related work in terms of OpenFlow controller performance. The architecture and features of OFCProbe are then introduced in Section III. We show and discuss the results of our example experiments in Section IV before drawing our conclusions in Section V.

II. Related Work

This section summarizes related work on benchmarking and troubleshooting SDN control software. First we describe related work in this area. Afterward, we briefly present our previous benchmarking tool and highlight its drawbacks.

A. Benchmarking and Troubleshooting Tools

The SDN Troubleshooting Simulator (STS), presented in [1] and [2] by Scott et al., addresses the problem of troubleshooting SDN control software that arises with the development of SDN platforms as network managment services. Current troubleshooting techniques are quite simple, they base on log inspection in the hope of finding relevant information. STS automatically identifies a minimal sequence of inputs to reproduce a given bug.

In the paper, ”'On Controller Performance in Software-Defined Networks [3] Tootoonchian et al. investigate the influence of the controllers performance on the overall network performance. They studied the NOX, NOX-MT (*m*aximum *t*hroughput), Beacon, and Maestro controllers each running with a L2 switching application. L2 switching has been chosen by the authors as it provides a lower-bound for look-up. Additionally, only basic switching had been implemented in all the tested controllers. The results provided in this paper show that existing controllers perform better than predicted in the referenced literature. However, they also state that understanding the overall SDN performance remains an open research problem.

Cbench [4] was the first available controller benchmark and emulates a given number of virtual OpenFlow switches to measure different aspects of a controllers performance, specifically mean latency and throughput. As Cbench's functions are very limited and controllers have been further developed since the release of this paper, the results shown in the paper only give basic information on controllers performance. To be able to make more profound statements on a controllers behavior a more advanced tool is needed. Preferable features that still were missing in Cbench are, e.g., to be able to distribute the program on multiple cores or machines and to provide statistics for each virtual switch individually.

B. Previous Work - OFCBenchmark

In [5], we introduced ”OFCBenchmark - a flexible OpenFlow-Controller Benchmark”. The benchmark creates a certain amount of virtual switches which in turn generate independently configured messages and produce their own statistics. Scalability, detailed performance statistics, and modularity were the top design goals. With the OFCBenchmark distribution enabled, the benchmark can be spread over multiple hosts with each host running its own client. The virtual switch, the key component of the tool, holds a simplified flow table, to be able to respond to controller requests, statistics counters and the connections to the controller.

While the tool yielded interesting results regarding the treatment of individual switches by the tested controllers (cf. [5]), non-ideal design and implementation decisions became apparent

C++ in combination with the Boost library for thread-handling was chosen for the implementation as the focus was on packet generation performance. This decision led to a platform-dependent tool that was not as easily extendable as originally conceived. Additionally, in light of the results yielded by the original tool, the focus of interest shifted towards the controller behavior rather than its raw throughput, which is also covered quite well by CBench. Therefore, we chose to design a new tool.

III. OFCProbe

In this section, we introduce our new tool for OpenFlow controller analysis ”OFCProbe”.

We begin by explaining our revised design goals that guided our development process.

A. Design Goals

Based on the lessons learned from the implementation OFCBench, we defined the following design goals:

Platform-Independency: In working environments one is often limited to certain operating systems and or hardware components. To bypass these requirements, we need a software that is executable on the most common system architectures.

Scalability: Scalability describes the software's ability to be run in a coordinated way on multiple CPU cores, CPUs, and hosts. This guarantees that the tool is not limited by a single core or the available memory. The tool should be multi-threading enabled and its threading overhead should be reduced in comparison to OFCBench.

Modularity: Modularity is the key to be able to easily adapt to new OpenFlow controller versions, new scenarios and/or new types of measurement values. The separation of the program logic and the controller communication is also a central goal to simplify the adaptation of the tool itself to possible new communication protocol versions.

Performance Analysis: We want to investigate OpenFlow controllers and their performance by sending generated messages to the controller and recording the controller's responses. Performance has different meanings in this context, it is defined by throughput and latency in a best-effort situation, but it also means recording the controller's behavior in different scenarios.

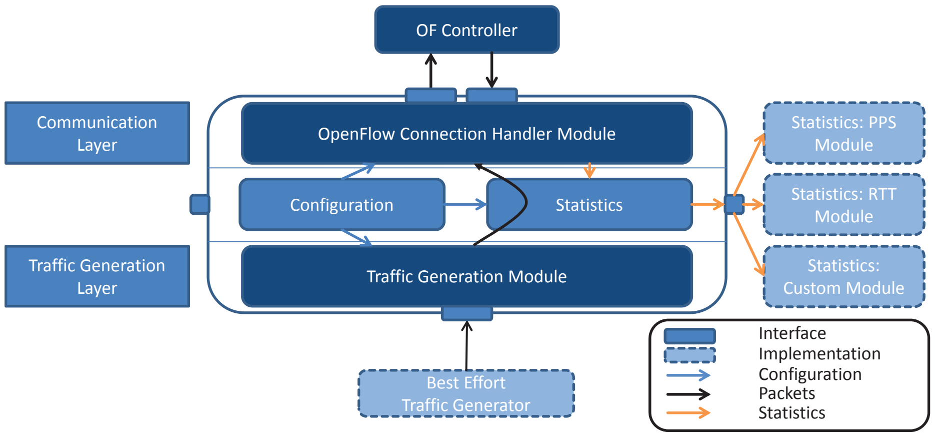
Detailed Statistics: Detailed statistics include a set of features that permits the investigation of the OpenFlow controllers behavior, e.g., whether it is treating multiple switches differently or changes its behavior over time.

B. Architecture of OFCProbe

Due to the fact that both OpenFlow and SDN are still evolving and changing, we decided to build this tool modular with interfaces between the modules. A schematic is depicted in Figure III-B. The ”OpenFlow Connection Handler Module” is responsible for the connection to the controller. The generation of messages is handled by the ”Traffic Generation Module”. Finally, there is a module for the storage of measurement values as well as a central management module, which handles the inter-module control messages. To realize platform independency, we chose JAVA in combination with the OpenFlowJ library as programming language. The implementation of the schematic first loads a configuration file and provides it to all other modules. After that the OpenFlow connection is established and after a successful OpenFlow handshake the traffic generation begins. Generated messages and their responses are traversing the configured statistics modules (e.g., round trip time, packets per second) and are recorded in a data format that enables further analysis.

Modularity enables users and developers to easily adapt the experimental setup to their needs. This could mean an update to an updated OpenFlow protocol version, but also a reconfiguration or the implementation of a new statistics module. A detailed description of each module follows.

1) The OpenFlow Connection Handler Module: This module is key to of our OFCProbe tool. It is in charge of establishing and holding the connection to the OpenFlow controller. Java's NIO Selector [6] is used for connection handling. With a Selector it is possible to handle multiple channels in one thread thus reducing the multithreading overhead. The module also contains the flow table implementation for the virtual switches. Apart from connection handling, the main task of this module is the acceptance of messages from the ”Traffic Generation Module” and their subsequent encapsulation into OpenFlow messages, which are then sent to the controller. The replies from the controller are also handled here and transmitted to the connected statistics modules for further analysis.



2) The Traffic Generation Module: The Traffic Generation Module is an event-driven scheduler/queue processor running within its own thread. The module has one queue in which all future events are stored. Each event consists of an event time, an event type and an associated virtual switch. When the event time is reached, the event is taken from the queue and based on its type, different actions are executed in the virtual switch.

A virtual switch goes through a life-cycle of events that is depicted in Figure 1. The first event is the OFSwitch_Connect_Event, which specifies when the virtual switch should start its connection establishment with controller. Subsequently, an OFSwitch_ConCheck_Event for this virtual switch is automatically queued for the near future to check for a successful connection. Once this is done, the first Packet_In_Event is scheduled for time $t = 0$. When a Packet_In_Event is executed, the filling level of the corresponding virtual switch's packet queue is checked. If it is below a configured threshold, the queue is filled up to that threshold with new packet payloads. Then the next Packet_In_Event is scheduled in a pre-defined amount of milliseconds. This continues until the Generation_End_Event is reached, which terminates the experiment. The values for the inter-arrival time of Packet_In_Events and the fill threshold are provided by the central configuration file or through a per-switch configuration. In the current version of OFCProbe, the payloads generated are TCPSyn-packets either with static or randomized address values. For each TCPSyn-packet three headers have to be generated: the Ethernet and IP headers, with each a destination and source address, and the TCP header with a destination and source port. OFCProbe uses a pre-generated master TCPSyn packet, that contains a correct TCPSyn packet with all fields except the address fields set. The remaining fields then are filled with generated addresses taken from the corresponding MAC-, IP- or TCP-generator.

3) The Statistics Module: Each outgoing OF Packet-In message and incoming OF Packet-Out message is forwarded by the connection handler module to every connected statistics module. This process is shown in Figure 2. With these messages and the corresponding arrival times, it possible to measure, e.g., the throughput, the latency, or the inter-arrival time of subsequent messages to the controller. Currently, following statistics modules are available: PPS (Packets Per Second), RTT (Round Trip Time), IAT (Inter-Arrival Times), and a CPU and RAM utilization monitor for the controller. The CPU and RAM monitors are implemented using SMTP.

C. Additional Features of OFCProbe

In this section we describe the additional emulation features that OFCProbe possesses.

1) Packet Capture Playback: Apart from the generated TCPSyn packets there is also another option to create the payloads for the OF Packet-In messages. To emulate a real network environment, it is also possible to specify a PCAP file containing a packet trace for each virtual switch individually. PCAP is a capture file format for network traffic that is used, e.g., by Wireshark [7]. These files are parsed by OFCProbe and each time a new payload is required, the next packet from the PCAP file is taken and encapsulated into an OpenFlow message. Figure 3 illustrates this feature. Ideally, the PCAP file should only include the first packets of arriving flows.

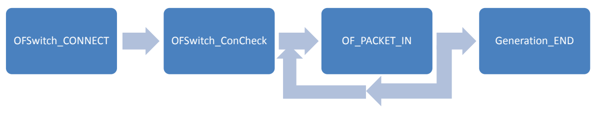


Figure 1. OFCProbe Event Chain

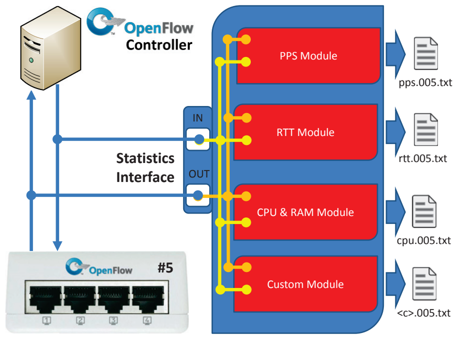


Figure 2. OFCProbe Statistics Module

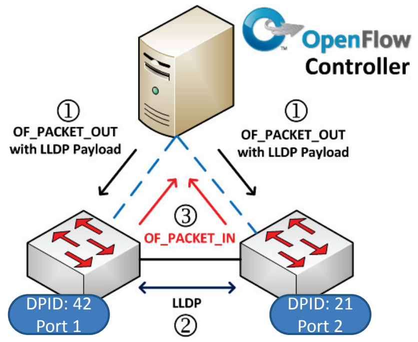


Figure 4. Topology Discovery via LLDP

2) Topology Emulation: Another feature of OFCProbe is the emulation of a network topology. Controllers like Floodlight or NOX are able to detect the underlying topology between the connected OpenFlow switches.

The protocol that enables devices to detect these links is called Link-Layer Discovery Protocol (LLDP). The discovery process works as depicted in Figure 4. The OpenFlow controller sends an OF Packet-Out message containing an LLDP payload (1) with the information that it was 'inserted' into the topology at a certain switch and the action to flood it to all ports of the OpenFlow switch. The OpenFlow switch recognizes the flood action in the message and floods the LLDP payload out of all its ports (2). A connected OpenFlow switch detects a new flow and sends this packet encapsulated into an OF Packet-In message to the controller (3). The controller then determines that the payload is an LLDP packet and at which switch this specific packet was inserted. This way the controller learns that there is a connection between the original switch and the one it received the packet from.

To enable OFCProbe for topology emulation the virtual switches have to detect the LLDP payloads in the OF Packet-Out messages from the controller and then flood them to connected virtual switches where they are encapsulated into OF Packet-In messages and sent back to the controller. The emulated topology can be defined in an separate topology initialization file.

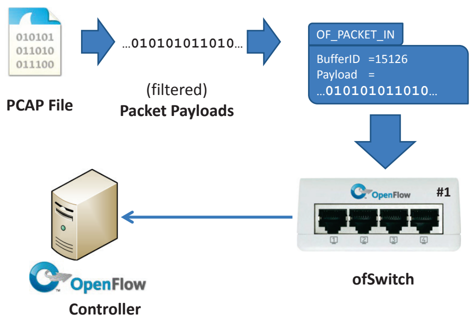


Figure 3. OFCProbe Packet Capture Playback

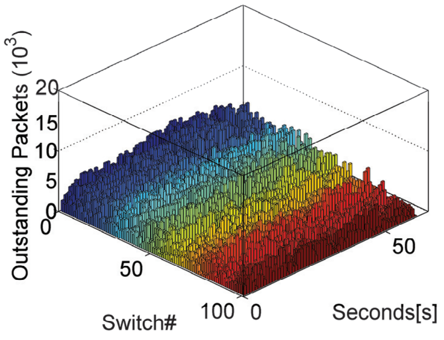
3) ARPing Feature: As an enhancement to the topology emulation, we implemented what we call the "ARPing feature". This feature enables the user to emulate devices connected to the free virtual switch ports of a topology. For every free port in the emulated topology a device with a MAC- and an IP-address is generated. After the successful topology emulation as described in the last section, each end device "sends" an ARP-request to every other end device in this topology. This is realized by sending an OF Packet-In message containing the ARP-request to the controller. The controller's reaction is an OF Packet-Out message with the action to flood this packet out of all virtual switch ports except the ingress port. The ARP-request is then queued in the next virtual switches that are connected to the original virtual switch in the emulated topology. There, the payload is again encapsulated in an OF Packet-In message and sent to the controller. This procedure continues until the ARP-request has reached the virtual switch the corresponding end device is connected to. The virtual switch recognizes the target device as one of its connected devices and then generates an ARP-reply in form of an OF Packet-In message to the controller. Now, the controller has learned to which virtual switch the target device is connected to.

IV. Example Measurement Evaluation of Different SDN Controllers using OFCProbe

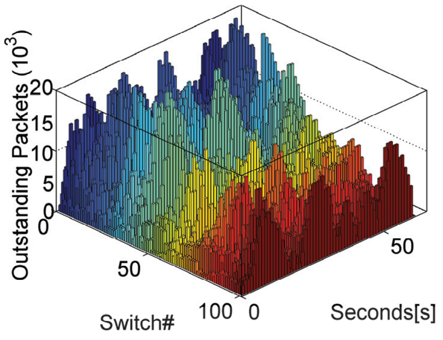
In this section, we present the results of some sample experiments conducted with the OFCProbe tool.

A. Best-Effort Tests

As stated before, OFCProbe is capable of an analysis per virtual switch. This allows a granular inspection of each switch at each time during the analysis. Figure 5(a) displays the outstanding packets, i.e., OF Packet-In messages that



(a) Floodlight



(b) NOX

Figure 5. Outstanding Packets for Different Controllers

have not been answered yet, of the Floodlight controller for a run with 100 emulated switches. The x-axis shows the number of the virtual switch, the y-axis the seconds passed since measurement start and the z-axis the number of outstanding packets in units of 10k packets. The controller's behavior appears to be quite fair as no switch has at any time a significantly higher amount of outstanding packets than the others with a maximum of outstanding packet values lower than 10k packets.

The outstanding packets for the NOX controller are shown in Figure 5(b). The arrangement of the axes is the same as for Figure 5(a). Here certain inequalities between the switches and for each switch over the time can be observed. There are ”waves” of outstanding packets in both x- and y-axis direction. This might be caused by the controller's implementation. The controller iterates over a list of connected switches, processing only one switch at a time. And while switch n is processed, the other connected switches have to wait for replies from the controller. Furthermore, the numbers of outstanding packets is notably higher than the numbers for Floodlight, having peaks with up to 30k outstanding packets.

B. *Topology Emulation and ARPing*

In this section, we investigate the controllers' behavior in a ”fat-tree” topology [8] setup as it can be found in current data centers. The particular fat-tree topology in the experiment has 20 virtual switches with 4 ports each. As shown in Figure 6, at the lowest level the virtual switches are connected to the emulated host devices with their two free ports. Each of these emulated devices generates TCP_SYN packets for flows to all other host at every packet-generation event. Therefore, 15x2 packets are generated per virtual switch per event. The inter-send time for switches 07, 11, 15 and 19 is set to 15 ms, switches 08, 12, 16 and 20 have a inter-send time of 40 ms.

Figure 7 illustrates the outstanding packets for the ingress switches of the Floodlight controller. The x-axis shows the passed time since measurement start, the y-axis the outstanding packets in multiples of 10k packets, respectively. The switches with the inter-send time of 15 ms (type #1) between their packet generation events have a drastically higher number of outstanding packets than the switches with the inter-send time of 40 ms (type #2), which have almost no outstanding packets except for one.

A general characteristic is observable for virtual switches with notable outstanding packet counts. Within 10 seconds their count rises above the other switches, continuously climbing and reaching a mean value of 4k at 30 seconds from the start. At around the 36 seconds, one switch settles at 6k outstanding packets, while the remaining switches constantly increase up to 15k packets.

As can be seen the confidence intervals are quite large for switches with a high number of outstanding packets. Therefore, it is interesting to look at the differences among them. Figure 8 shows a 3D representation of the outstanding packets of all switches of a single run. Here, again, the drastically higher outstanding packet numbers for the switches #07, #11 and #19 can be observed as they reach 20k, 10k and 8k outstanding packets. Switch #15 is an exemption.

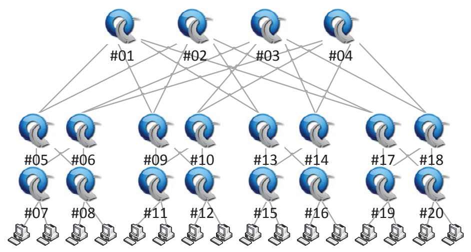


Figure 6. Fat-Tree Topology

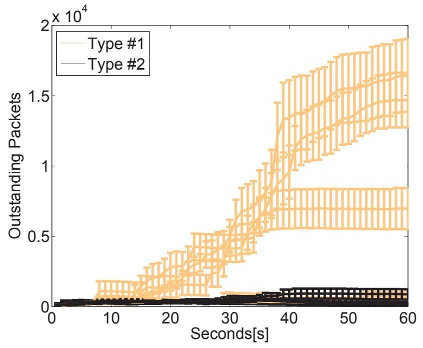


Figure 7. Outstanding Packets for Floodlight

The difference between the two sets of inter-send times is significant and observable. The other non-ingress switches all have next to no outstanding packets.

To summarize, Floodlight's switch handling appears to be fair in terms of performance as only switches with a high packet load have to wait for controller processing. All requests from the other switches are handled immediately.

V. Conclusion

With OFCProbe, we introduced a platform-independent tool to perform measurements on the behavior, characteristics and bottle-necks of current OpenFlow controllers. The architecture of OFCProbe is platform-independent, scalable, and modular to allow a granular analysis of the controllers behaviors and characteristics. OFCProbe emulates virtual OpenFlow-enabled switches that each provide granular statistics about different aspects of the controllers' behavior. The virtual switches are arrangeable into topologies to emulate realistic scenarios and recorded PCAP files can be used to test the controllers reaction to real requests. With each run, a detailed insight into the OpenFlow controller and its implementation is provided.

To demonstrate the features of OFCProbe, we conducted different experiments for NOX and Floodlight. The gained insights demonstrate the heterogeneous behavior of the investigated controllers. The results show an unequal load balancing for the Nox controller, where a couple of switches are congested while others are idle. The Floodlight controller seems to distribute its processing capabilities more fairly among the switches. Even in uneven setups, like the Fat-Tree topology, it remains fair towards each connected switch.

There are still various questions open for future research. Thanks to its modularity, OFCProbe offers many possibilities for future analysis. This can be the investigation of new scenarios and OpenFlow protocol versions, a verification module to validate the correct operation of a controller, or new statistic modules. The tool and source code are available for download [9].

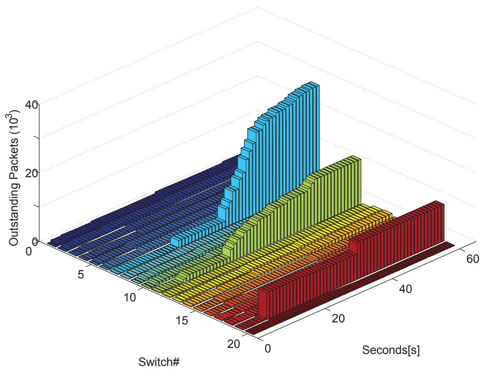


Figure 8. Outstanding Packets for Floodlight

References

[1] C. Scott, A. Wundsam, S. Whitlock, A. Or, E. Huang, K. Zarifis, and S. Shenker, "Automatic troubleshooting for sdn control software," *online*, 2013.

[2] C. Scott, A. Wundsam, S. Whitlock, A. Or, E. Huang, K. Zarifis, and S. Shenker, "How did we get into this mess? isolating fault-inducing inputs to sdn control software," *online*, 2013.

[3] A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado, and R. Sherwood, "On controller performance in software-defined networks," in *USENIX Workshop on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services (Hot-ICE)*, 2012. [Online; accessed 11-June-2013].

[4] R. Sherwood and K.-K. Yap, "Cbench Controller Benchmarker." http://www.openflowswitch.org/wk/index.php/Oflops, 2011. last accessed on 2011.12.19.

[5] M. Jarschel, F. Lehrieder, Z. Magyari, and R. Pries, "A Flexible OpenFlow-Controller Benchmark," in *European Workshop on Software Defined Networks*, (Darmstadt, Germany), Oct. 2012.

[6] Oracle, "Java nio selector." http://docs.oracle.com/javase/7/docs/api/java/nio/channels/Selector.html, 2013. [Online; accessed 30-August-2013].

[7] Wireshark Foundation, "Wireshark - go deep." http://www.wireshark.org/, 2013. [Online; accessed 04-July-2013].

[8] Wikipedia, "Fat tree — wikipedia, the free encyclopedia." http://en.wikipedia.org/w/index.php?title=Fat_tree&oldid=556326188, 2013. [Online; accessed 12-August-2013].

[9] "Ofcprobe: Openflow controller analysis tool." http://www3.informatik.uni-wuerzburg.de/research/ngn/ofcprobe/ofcprobe.shtml, 2014.