Design of a Configurable Wireless Network Testbed with Live Traffic

Ruben Merz, Harald Schiöberg, and Cigdem Sengul

Deutsche Telekom Laboratories, An-Institut der Technischen Universität Berlin
{ruben.merz,cigdem.sengul}@telekom.de, harald@net.t-labs.tu-berlin.de

Abstract. Testbeds have become standard tools for performance evaluation in wireless networks as they allow for evaluation with realistic wireless propagation and interference conditions, hardware constraints and timing requirements. However, in most testbeds, traffic and user mobility is generated by either using synthetic load generation tools, or replaying traffic traces. Evaluations using live traffic generated by real users, possibly moving around the network, are typically not possible. This is the main motivation of our research: building a wireless research testbed that enables experimentation with live traffic. In this paper, we present the design and architecture of a campus-wide wireless network testbed towards this goal. Our testbed enables both transparent Internet access and seamless mobility to the network users, and supports full network reconfigurations in the presence of live traffic. The reliability of user traffic is guaranteed by failure avoidance mechanisms that are invoked whenever a disruption occurs in the network.

Keywords: Wireless testbeds, live traffic, online configurations.

1 Introduction

Wireless networking research has gained prominence over the years due to its potential for realizing "anytime, anywhere networking". However, the transition from research to production level deployment has been somewhat slow as many wireless networking ideas do not typically move out of simulation environments. Therefore, despite the efforts for more realism and accuracy in wireless simulation [1,2,3], evaluation with testbeds needs to become more prevalent to understand the impact of realistic wireless propagation and interference conditions, hardware constraints and timing requirements. However, evaluation with testbeds is extremely tedious: research ideas are throttled by high implementation barrier, cumbersome and time-consuming experiment set-up and debugging, and validation challenges. These difficulties preclude testing in real systems and limit a broader adoption of testbed-based experimentation and evaluation.

To address these difficulties, wireless testbeds such as Orbit [4,5] and Emulab [6] allow for automating network and topology configurations and provide better experiment management and repeatability. Although these works ease deployment (e.g., by providing remote access to nodes and operating system distributions, tools to automatically script measurements), the implementation of an idea still remains significantly difficult. For instance, even the reconfiguration of network protocols and parameters need to go through invasive kernel changes. Therefore, current testbeds need to facilitate both implementation and deployment. Furthermore, current wireless testbeds do not typically serve live user traffic. In addition to allowing realistic input for evaluations, serving real users also urges researchers to build *complete systems.*

To the best of our knowledge, no testbed exists that provides both real-user traffic and easy online reconfigurations for wireless networking experimentation. We believe this limits full evaluation of research proposals; i.e., answering how a system behaves under different conditions, and finding out the underlying reasons for such behavior and the advantages of a system compared to others. To this end, a research testbed should allow for reconfiguring the network and running experiments with live traffic, characterizing the current network, traffic and mobility conditions, correlating system behavior to these conditions, and reproducing past conditions and system behavior. Bringing all these together presents a much challenging agenda. However, we assert that it is also much required to be able to fully evaluate systems. In this paper, we present a first step - the design, architecture and implementation of a testbed, the Berlin Open Wireless Lab (BOWL) [1], which reliably supports live traffic in a research network.

The main challenges that we address in this work stem from the conflicts between the expectations of network users and researchers. Real users expect reliable network access and privacy. On the contrary, a research network is an unstable environment with frequent outages due to experimental software and reconfigurations. These two different viewpoints create the following trade-offs in a research environment:

– **Reliable Internet access for users vs. network programmability for researchers:** Researchers require network programmability to run controlled experiments. In turn, any disruption to the user traffic in the presence of experiment reconfigurations or an outage caused by experiment failures should be avoided. Essentially, failure avoidance mechanisms should automatically push the network to a safe state and gracefully recover affected nodes.
– **Privacy for users vs. providing a rich set of measurements for researchers:** Since user traffic is not controlled, all the parameters of the underlying layers, e.g., TCP, routing and medium access control (MAC) layers should be captured for any meaningful measurement study. Furthermore, application level information might be necessary to reproduce an experiment. However, the measurement and tracing of system information should be performed without jeopardizing user privacy.

[1] http://bowl.net.t-labs.tu-berlin.de/

The main contribution of our research is providing two essential capabilities that address these trade-offs:

– Automated and online reconfigurations: it includes turning on/off functionality of a specific protocol and changing protocols at different network layers respecting inter-layer dependencies.
– Hosting real user traffic: this requires avoiding disruption to the user traffic during reconfigurations. Failure avoidance mechanisms are triggered to automatically push the network to a safe state during reconfigurations. The network comes out of this state once a reconfiguration is complete.

To the best of our knowledge, this is the first wireless testbed of its kind.

The rest of the paper is outlined as follows. In Section 2, we present the implementation of our outdoor wireless testbed, both its network and node design, as well as the software components. In Section 3, we present an example configuration of our network: the mesh network configuration. Section 4 discusses the related work and we conclude in Section 5 with a summary and future work.

2 The Berlin Open Wireless Lab Network

To create our testbed, we deployed 46 nodes outdoor, on the rooftop of several buildings in the TU-Berlin campus, covering roughly 40 hectares. The maximum link distance is one kilometer. In addition, we deployed 12 nodes indoor on two floors of the Deutsche Telekom Laboratories. In the remainder of this section, we walk through our network and node design, software architecture and the main components for reconfigurations with live traffic.

2.1 Network Architecture and Node Design

Our network architecture and the node design are shaped mainly by our design objectives: to be able to perform remote management as well as failure avoidance and recovery in the presence of disruptions to the network (e.g., due to failures or launching new experiments), and to provide a rich set of configuration options. To achieve high *topology configurability*, our nodes are deployed densely in the campus, allowing switching off a significant fraction without losing connectivity. This enables effective evaluation of power control and interference issues in dense networks. Additionally, our nodes have several wireless interfaces. Also, the antenna poles are extensible and can easily host additional hardware.

Our mesh nodes are built around an Avila Gateworks GW2348-4 motherboard [7]. Each node has an Intel XScale 533 MHz CPU (ARM architecture), 64 Mbyte of RAM and four wireless interfaces - Wistron CM9 with an Atheros 5213 chipset. One wireless interface is always operated in the ISM band at 2.4 GHz, running IEEE 802.11g, where either a 2.4 GHz 12 dBi omnidirectional antenna or a 17 dBi sector antenna is attached. This interface provides an access point (AP) functionality for users to access the network and the Internet. The three other wireless interfaces are operated in the UNII 5 GHz band, where a 5 GHz 12 dBi omnidirectional antenna is attached.

For most of the rooftop nodes, physical access is severely limited. Therefore, each node is remotely accessible through a wired, 100 Mbit/s Ethernet, connection. This Ethernet connection plays a critical role for remote management, failure avoidance and reconfigurations. In practice, the outdoor nodes and the indoor nodes are on two separate VLANs that connect to a central router. Two other VLANs are also attached to this central router: one connects to the Internet through the TU-Berlin network and the other one is used for management. For failure avoidance, additionally, a hardware watchdog daemon automatically triggers a reboot in case of malfunction or connectivity loss. Finally, a majority of the nodes contain a second independent unit for managing the router in case of a malfunction or as a passive measurement and monitoring device. This second unit is built around an Asus WL-500GP router. A custom-built circuit interfaces the two boards and allow us to remotely power on and off each board from each other. The nodes are powered by Power over Ethernet (PoE). Hence, for the rooftop nodes, only one cable per node is necessary. Both units run a customized version of the OpenWrt operating system, a GNU/Linux distribution for embedded devices [8].

2.2 Software Architecture

Building on its hardware and network architecture, BOWL brings together several software components to achieve a reconfigurable live network:

- **Online reconfigurations** provide network, topology, parameter and protocol reconfigurations.
- **Failure avoidance mechanisms** switch nodes to a fail-safe state to maintain reliability during disruptions.
- **Remote management and monitoring** remotely and automatically manages nodes as well as checks their status.
- **Measurements and tracing** provide user, system and network information.
- **Visualization and control interface** provides both flow and connectivity based views of the network in real-time.

In the rest of the section, we explain these components in further detail.

Online Reconfigurations

To run experiments in a wireless network, researchers need the capability to perform automated and online reconfigurations. To this end, the BOWL testbed can be run in different *network configurations* for each experiment. We currently support three basic configurations: an infrastructure network (the default configuration), a mesh network or a mixture of the two.

In each configuration, the 2.4 GHz wireless interface is used as the "client interface" to provide an AP functionality for Internet access. However, it is up to the experimenter to use the live traffic or to direct it straight to the Internet over the wired interface. In the infrastructure configuration, the client interface is transparent and bridged to the central router. Authentication and access control are handled on this central router, as well as DHCP. We provide transparent and seamless mobility and roaming between the APs. This is achieved by mainly running the "client interfaces" in promiscuous mode.

Additionally, the BOWL testbed supports configurations for changing the network topology, exploring the parameter space of a given protocol, or comparing different protocols under similar network conditions. To modify the *topology*, we support connecting additional nodes to or disconnecting nodes from the network remotely. The level of network connectivity can also be modified by changing the per-node transmission power levels. Going a step further, we also support modifications to *protocol parameters*. For instance, we can change a MAC layer functionality: turn on and off RTS/CTS in IEEE 802.11. We also enable *entire protocol* switches at a given layer of the network stack. As a proof of concept, we currently support routing protocol switches in our mesh configuration. These examples are explained in more detail in Section 3.

For efficient execution, all these reconfigurations rely on the "remote management and monitoring" component, which will be described later in this section. Furthermore, during all these reconfigurations, user traffic is expected to be disrupted. How to protect user traffic from such disruptions is discussed next.

Failure Avoidance Mechanisms

The core of our software design consists of providing one stable and safe default configuration, or "rescue" mode to fall back to during disruptions and a very flexible "live" mode to facilitate research. This is implemented by installing a so-called "rescue system" and possibly multiple so-called "guest systems", each of which are fully self-contained Linux systems. To this end, we extended OpenWrt for our purposes. The rescue system is installed on the internal flash memory of all the nodes and is started by the boot loader. Guest systems are installed on additional flash memory storage.

The rescue mode boots the rescue system, runs the default infrastructure configuration (i.e., the user traffic is bridged to the central router over the wired interface) and provides the functionality to install and launch guest systems. The live mode boots a guest system and runs a configuration corresponding to an experiment. Both modes must always implement the AP functionality.

In addition to the rescue and live modes, BOWL supports a "transient" mode, where the data flows by default through the Ethernet interface and can be replicated over the wireless network for testing (i.e., the duplicate traffic is discarded at the exit point of the network). Like the AP functionality, the transient mode must be implemented by any guest system.

Failure avoidance makes nodes to fall back to either rescue or transient modes. It can be triggered on-demand whenever we expect a major disruption or automatically. If the watchdog triggers a reboot, a node reboots in the rescue system. The expected delay is in terms of a few seconds. The management of these different states are handled by the "remote management and monitoring" component, which is explained next.

For the implementation of the online reconfiguration and failure avoidance mechanisms, we make extensive use of the Click modular router [9,10].

Remote Management and Monitoring

The management architecture was designed using a centralized approach that contains two parts: (1) **A central node manager** keeps various pieces of node state and information in a database and marshals commands and information to the nodes. (2) **A node controller** runs on each node and executes commands initiated by the central node manager, as well as collects state and information, which, in turn, are sent back to the central node manager.

All software is written in Ruby [11], as it is readily available on several platforms which cuts down development time in case we upgrade or change our hardware. The second reason is the availability of Distributed Ruby (DRb), a remote method invocation (RMI) implementation. The database uses Postgresql[12].

The central node manager is a process that uses a database back end to maintain information about each node controller. Communication is marshaled to the nodes using DRb, which allows for both state and code distribution. In addition, the central node manager implements an event-based callback framework. Finally, the node manager supports exporting data to other components (e.g., a visualization component) out of the database. To support privacy, sensitive data is mangled: for example, the MAC addresses are anonymized.

Each node runs an instance of the node controller to communicate with the central node manager. The node controller implements two main concepts: so-called adaptors and an event framework. An adaptor maintains a particular functionality on the node (e.g., DHCP), control various daemons and relay information to the node controller itself. Each adaptor runs in its own thread within the controller process. In the default configuration, the default adaptor is the association adaptor, which maintains information about the associated clients. The central node manager learns about new clients through this adaptor.

The event framework allows the controller to react to changes in the environment and the state of its adaptors. The typical communication flow between a node controller and central node manager is as follows: (1) Changes in an adaptor may result in an event. (2) This is relayed by the node controller to the central node manager. (3) Next, this may lead to the execution of callbacks. (4) The callbacks, in turn, may be dispatched back to the node in question.

Measurement and Tracing

Measurements typically consist of data points, collected by the nodes and transferred into the measurement database for further processing (e.g, for visualization). Measurement and tracing processes currently run separately from the node controllers. However, like any other program, these processes are also generally configured and started through the node controllers. To collect measurements from the nodes to a central location, we extensively use the Orbit measurement framework and library (OML), which comprises a measurement library and server developed in the context of the Orbit testbed [13]. We currently support two ways of collecting data. Connectivity measurements collect signal strength information by monitoring IEEE 802.11 beacons or other traffic, and traffic measurements collect data regarding the existing flows in the network.

Visualization and Control Interface. We implemented a live testbed map, which displays the current situation in the network (e.g., existing *real user* connections, current node configurations and protocol parameters). Using this map, changes like new nodes or disruptions can be easily displayed. The data to the network visualization component is acquired from two different interfaces: (1) the database of central node manager, which maintains different node states and (2) the measurement component, which, for instance, provides connectivity strength among different nodes and flow information.

The network is also remotely configurable through either a command line interface or a "Control" frame embedded in the map, which can interface to the node manager. In the next section, we give examples of currently possible reconfigurations and implemented controls based on the mesh network configuration.

3 A Configuration Example: The Mesh Experiment

In this section, we show how the BOWL testbed can be configured as a wireless mesh network [14]. In the current implementation, any node can act as an AP and one node acts as the gateway. Figure 1 depicts a snapshot of the network in this configuration on 27 October, 2009. In the mesh configuration, the mesh network is transparent to the clients and is seen as a regular layer 2 network. The mesh interfaces use a different IP address range than the clients and the central router. In the mesh, IP packets are encapsulated using IP in IP tunneling and sent towards the mesh gateway. They are decapsulated at the gateway before being delivered to the central router. The exterior IP addresses of the tunnel belong to the mesh IP address range. The AP nodes use ARP spoofing to answer ARP requests from the clients to obtain the hardware address of the first hop. Also, the gateway uses the same technique to answer ARP requests from the first central router for the hardware addresses of the clients.

This tunnel setup also enables seamless mobility. At an AP node, a so-called location table indicates which node is the current gateway of the network[2]. The destination IP address of the tunnel is then set to the mesh IP address of the gateway. At the gateway, the location table indicates to which AP node is a client currently attached. Hence, the destination IP address of the tunnel is set to the mesh IP address of this AP node. These location tables need to be maintained only on nodes that function as an AP or a gateway. The location tables are currently updated centrally by using the Ethernet interface.

In this mesh configuration, we support several challenging reconfiguration scenarios, e.g., a routing protocol switch from OLSR to DSR and vice versa. In DSR, a route is discovered only when a new flow is initiated. Therefore, additional steps need to be taken to first find routes for flows that exists in the network during reconfiguration. In this case, the transient mode is a perfect fit as it redirects the client traffic through the wired interface but also duplicates it on the active wireless interfaces. When switching between DSR and OLSR, the transient mode is activated right before starting the routing protocol switch. This

[2] This setup is straightforward to extend to multiple gateways.

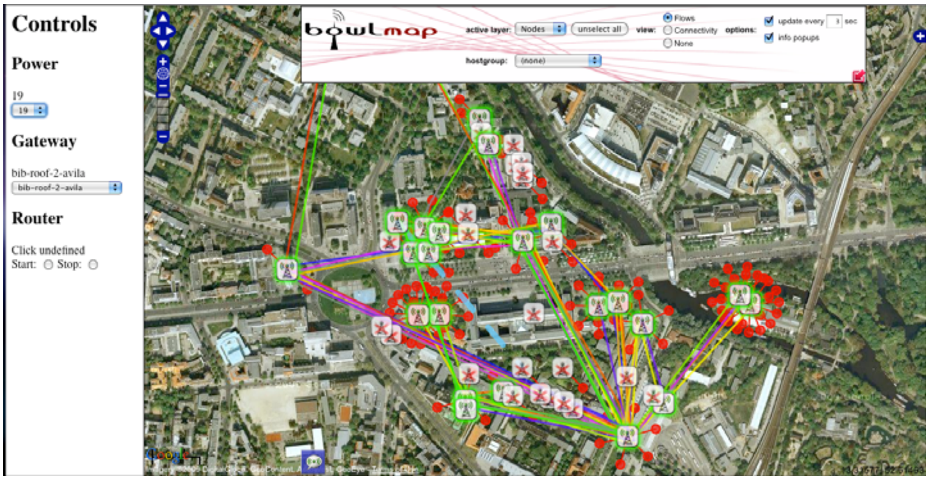

Fig. 1. Snapshot of the network in the mesh network configuration. The green nodes are online APs; the red nodes are unreachable as they are newly deployed, or not configured. The red dots are the clients attached to the access points. The map depicts the flow view, where each flow is is shown with a different color line. The control frame on the left enables transmit power control, gateway change, and both wired-only and wireless-only communication through "start and stop mesh routing".

duplicated traffic bootstraps DSR and fills route caches on the nodes. Similarly, transient mode is used when the gateway functionality is assigned to another node, or when the transmission power is modified. In both cases, the connectivity to the gateway might be impaired and hence, requires client traffic to be safely delivered over the wired interface until connectivity to the gateway is restored.

4 Related Work

The existing work aiming at evaluating wireless research can be roughly categorized between simulators, emulation testbeds and testbed deployments. Simulators are typical tools of choice [3] in wireless research due to their relative ease-of-use. However, a recent survey also shows that simulations may have several pitfalls due to faulty practices, *but* more importantly, due to abstracting details, especially at the physical layer.

Some of the limitations facing wireless scientific research and some suggestions to overcome these are presented in [15]. The closest to our work in essence are PlanetLab in the wired world and TFA [16], the Netbed/Emulab [6] and Orbit [4,17,13,18,5] in the wireless world. PlanetLab [19] also tries to bring together researchers that develop and test services, and clients that want to use these services. The TFA network [16] covers residential users with the goal of providing reliable connectivity to low-income households, whereas our goal is to enable evaluation and comparison of real systems in an open experimental environment. Netbed/Emulab provides a test-bed management framework with a Web-based front end used to create and manage experiments. Note that Emulab is an indoor network, which allows testing protocols with artificial traffic. Despite these differences, Emulab provides a very mature service interface for distributing code, controlling applications and gathering log files, which can be extended to match our requirements.

To compensate for the lack of realistic network conditions in Emulab, FlexLab [20] proposes to loosely couple Emulab with PlanetLab. This can only capture limited aspects of user behavior, namely traffic characteristics but not mobility. Another effort is presented in [21]. However, the goal is limited to re-introducing "laboratory notebooks" to the networking community and automating re-running recorded experiments or their variations.

The Orbit testbed [4,5] is an indoor two-dimensional grid of 400 IEEE 802.11 radios. Nodes can dynamically form specified topologies with reproducible wireless channel models. However, Orbit does not support real users. Similarly, the Hydra testbed [22] is a purely research testbed with no support for real users. To the best of our knowledge, no experimental research environment exists, which can meet all three design goals (1) supporting real users, (2) enabling to build real systems and (3) facilitating real system evaluation.

5 Conclusion and Future Work

This works presents the Berlin Open Wireless Lab (BOWL) network, a configurable testbed with support for live traffic. The current network deployment comprises 46 multi-radio IEEE 802.11 nodes deployed outdoor on the rooftops of the TU-Berlin campus. The live traffic is generated by TU-Berlin students and staff using the network for Internet access. Thanks to its unique software architecture, the network allows researchers to reliably and remotely reconfigure the network and run experiments with live traffic.

Currently the following reconfiguration scenarios that can be remotely activated in the BOWL network include a mesh network configuration with a single gateway, gateway activation and gateway location update, and switching protocols and updating protocol parameters. This work is a first step towards an experimental research environment with real user traffic. For future work, we plan to extend the software architecture to better support the collection of measurements, validation of configurations and experiment results.

Acknowledgments

We would like to thank Anja Feldmann for her continuous support to the BOWL project, Thomas Hühn who made the deployment of this network a reality, Mustafa Al-Bado for the implementation of node installation validation tools, Benjamin Vahl and Julius Schulz-Zander for their work on the live map, and our system administrator, Thorsten Fischer.

References

1. The ns-3 network simulator, `http://www.nsnam.org`
2. Kotz, D., Newport, C., Gray, R.S., Liu, J., Yuan, Y., Elliott, C.: Experimental evaluation of wireless simulation assumptions. In: MSWiM 2004: 7th ACM Int. Symp. on Modeling, Analysis and Simulation of Wireless and Mobile Systems, pp. 78–82. ACM, New York (2004)
3. Kurkowski, S., Camp, T., Colagrosso, M.: MANET simulation studies: The incredibles. Mob. Comp. and Comm. Rev (MC2R) 9, 50–61 (2005)
4. Orbit, `http://www.orbit-lab.org/`
5. Raychaudhuri, D., Seskar, I., Ott, M., Ganu, S., Ramachandran, K., Kremo, H., Siracusa, R., Liu, H., Singh, M.: Overview of the orbit radio grid testbed for evaluation of next-generation wireless network protocols. In: IEEE WCNC 2005 (2005)
6. School of Computing, University of Utah: Emulab - total network testbed, `http://www.emulab.net/`
7. Avila network platform gw 2348-4, `http://www.gateworks.com/products/avila/gw2348-4.php` (retrieved November 2009)
8. Openwrt, `http://openwrt.org/`
9. Kohler, E., Morris, R., Chen, B., Jannotti, J., Kaashoek, M.F.: The click modular router. ACM Trans. Comput. Syst. 18, 263–297 (2000)
10. The click modular router, `http://read.cs.ucla.edu/click/`
11. The ruby programming language, `http://www.ruby-lang.org/`
12. Postgresql open source object-relational database system, `http://www.postgresql.org/`
13. Singh, M., Ott, M., Seskar, I., Kamat, P.: Orbit measurements framework and library (oml): Motivations, design, implementation, and features. In: TRIDENTCOM, pp. 146–152 (2005)
14. Akyildiz, I.F., Wang, X., Wang, W.: Wireless mesh networks: a survey. Comput. Netw. ISDN Syst. 47, 445–487 (2005)
15. White, B., Lepreau, J., Guruprasad, S.: Lowering the barrier to wireless and mobile experimentation. ACM SIGCOMM CCR 33, 47–52 (2003)
16. Camp, J., Knightly, E., Reed, W.: Developing and deploying multihop wireless networks for low-income communities. In: Digital Communities (2005)
17. Ott, M., Seskar, I., Siraccusa, R., Singh, M.: Orbit testbed software architecture: Supporting experiments as a service. In: TRIDENTCOM, pp. 136–145 (2005)
18. Ganu, S., Kremo, H., Howard, R., Seskar, I.: Addressing repeatability in wireless experiments using orbit testbed. In: TRIDENTCOM, pp. 153–160 (2005)
19. Peterson, L., Anderson, T., Culler, D., Roscoe, T.: A blueprint for introducing disruptive technology into the internet. In: Hotnets (2002)
20. Ricci, R., Duerig, J., Sanaga, P., Gebhardt, D., Hibler, M., Atkinson, K., Zhang, J., Kasera, S., Lepreau, J.: The flexlab approach to realistic evaluation of networked systems. In: NSDI (2007)
21. Eide, E., Stroller, L., Lepreau, J.: An experimentation for replayable networking research. In: NSDI (2007)
22. Mandke, K., Choi, S.H., Kim, G., Grant, R., Daniels, R.C., Kim, W., Heath, R.W., Nettles, S.M.: Early results on hydra: A flexible mac/phy multihop testbed. In: IEEE VTC-Spring, pp. 1896–1900 (2007)