

# A Failsafe Architecture for Mesh Testbeds with Real Users

Harald Schiöberg  
Technische Universität Berlin  
Deutsche Telekom  
Laboratories  
harald@net.t-labs.tu-berlin.de

Ruben Merz  
Technische Universität Berlin  
Deutsche Telekom  
Laboratories  
ruben.merz@telekom.de

Cigdem Sengul  
Technische Universität Berlin  
Deutsche Telekom  
Laboratories  
cigdem.sengul@telekom.de

## ABSTRACT

We build a research testbed at our campus with real users, namely all the members of the university, to bridge the gap between synthetic mesh testbeds and productive mesh deployments. This requires combining the flexibility of research-only testbeds with the high reliability of production networks. This paper shows how we bring these two contradicting design goals together, given the degree of flexibility we demand calls for the ability to exchange the implementation of possibly all layers of the network stack. Given the experimental nature of the environment, we expect frequent software failures. However, we argue our failsafe architecture can still limit the impact of such failures to a satisfactory level of user experience.

## Categories and Subject Descriptors

C.2 [COMPUTER-COMMUNICATION NETWORKS]:  
Network Architecture and Design—*Wireless communication*

## General Terms

Algorithms, Design, Reliability

## 1. INTRODUCTION

Contemporary wireless research is hampered by the lack of credible validation of results in realistic environments. Currently, evaluation of algorithms usually takes place in either simulators or small-scale testbeds with no real traffic. Yet it is shown in [1] that the outcome of experiments in simulated or emulated environments can vastly differ from the behaviour under realistic conditions. We believe that this can only be remedied with a realistic testbed providing Internet access to real users to verify experiments. We are currently deploying such a testbed, the Berlin Open Wireless Lab (BOWL)[2], to meet these demands in the area of our university campus.

One challenge for such a testbed is the contradicting demands of the end users and the researchers: End users want stable Internet connectivity without interruptions, while researchers want to perform experiments, causing frequent interruptions. To bring users and researchers together, we have specified the following requirements for the BOWL network:

- **Reliability** Each node must provide end user connectivity to the Internet. Interruptions must not be longer than one minute in the worst case. For the time being, throughput and delay are allowed to vary. Some bounds can be defined after the testbed is fully deployed and baseline measurement results are available.
- **Transparency** As it is infeasible to maintain specific software for each possible client and ask the users to install it on their system, access to the Internet must be transparent to the client.
- **Flexibility** BOWL may not put any limitation on the implementation of experiments. Yet we do encourage the use of our environment for implementing experiments.

These requirements are reflected on the testbed design and we define a set of minimal functionalities that experimental systems need to provide, so that the entire system in the end still meets the requirements. Definition of these functionalities must be precise, such that the system behaviour can be tested before deployment and constantly monitored during the experiment. Hence a *node failure* is defined by the monitoring component detecting an erroneous behaviour. We furthermore define a *rescue mode* for the whole system, to which we can reliably revert any amount of failing nodes within the one minute time constraint and in which connectivity is restored throughout the whole system.

The remainder of this paper is structured as follows: Section 2 gives an overview over related works, Section 3 briefly outlines the hardware of our testbed. Section 4 describes the relevant pieces of the software environment. Section 5 describes the network architecture and its different states of operation. Section 6 concludes the paper.

## 2. RELATED WORK

Architectures designed for mesh network deployments fall under two categories: Research testbeds or production networks. We define a research testbed as a testbed where actual networking protocols are tested, and these can be easily changed for different experiments. Prominent examples are the Orbit testbed [3, 4, 5] and Emulab [6].

Production networks in our definition are networks, that are used for their capability to actually transfer data. This can be for the pure purpose of providing Internet coverage [7, 8] or to measure the network and user behaviour [9].

Notably, to the best of our knowledge, there exists no protocol level research testbed, that directly serves real user traffic. Recognising the importance of more realistic workloads, extensions to the research testbeds exist [10] to provide real Internet traffic as workload. Yet none of them

allows easy protocol level experiments in a realistic network deployment with real users.

### 3. PHYSICAL TESTBED DESIGN

The degree of reliability we demand cannot be achieved by software alone. It needs support by the mesh node itself, as well as the entire network deployment. The most important part is obviously the mesh node. First of all it needs to support a reliable recovery mechanism in case of catastrophic software failure, such as a kernel freeze. Fortunately, most contemporary embedded platforms feature a hardware watchdog, that is used for this purpose. It is basically a dedicated circuit, that will reboot the hardware after a given time, unless triggered regularly by the software. Second, we need support for an alternative operating system, since after such an emergency reboot the original system must be considered flawed, and probably needs manual intervention before being started again. To prevent interference of the actual experiment with client access to the network, the mesh nodes have dedicated wireless interfaces for access and for the mesh network. Finally we need to be able to observe faulty behaviour from the outside. For instance, crash dumps yield valuable information for debugging. We also need the ability to interact with the system if only some subsystems have failed. We opted to equip each mesh node with an auxiliary board, that connects to the main mesh board. The auxiliary board has significantly lower hardware specifications, but connects to the serial terminal of the mesh board to allow recording crashes and to supply an emergency management console. Yet the main function of the auxiliary boards is to serve as passive wireless measurement drones during the experiments.

The second component that needs to be designed for reliability is the supporting infrastructure. For this purpose, each node is connected over an Ethernet connection to our central servers. This might at the first glance contradict the concept of a mesh network, yet these wires need not be used for Internet access. On the contrary, due to the wires, each node can be made an Internet gateway on demand by pure software reconfiguration. In fact, a reliable connection is not only required for proper monitoring and measurements, it is needed to take over the user traffic instantaneously in case a multihop experiment fails, to prevent noticeable impact on the end users.

Finally, to avoid introducing too many faults on the live network, a proper stage testing environment is needed. We opted for a three tier design: Tier one, called *Smoketest*, is a testbench consisting of a set of nodes, with every possible interface being remotely accessible. It is used for software development and verification. The second tier is the Indoor testbed, a classical “a dozen of nodes in an office” deployment, but featuring an identical hardware configuration as the third tier, which is the live Outdoor network.

## 4. SOFTWARE

As the software on the nodes plays a key role in providing reliability, this section describes its key features. The software ensuring reliability that runs on the nodes is described in Section 4.1, the relevant parts of the supporting software are described in Section 4.2.

### 4.1 Node software

The crucial part of the software design is to support a stable *rescue mode* and a very flexible *live mode*. This is implemented by installing a *rescue system* and possibly multiple *guest systems*, each of them a fully self-contained Linux system. We use our own version of OpenWRT [11], called the Open Wireless Experimentation and Evaluation Distribution (OpenWEED).

On larger machines, one would possibly run the guest systems in a virtualized environment, yet given the constraints of embedded hardware this is not an option.

The rescue system is installed on the internal flash memory of the nodes, and is started by the boot loader. It is stripped to the bare minimum, providing only functionality to install and launch guest systems and simple access point functionality as explained in Section 5. Finally, the rescue system initializes the hardware watchdog. It ensures that in case anything goes wrong, the node reboots even if a launched guest system fails to boot.

Multiple partitions of an externally attached flash memory store the guest systems. Guest systems are launched using *kexec*, that allows to load another kernel into the main memory and boot it, completely replacing the entire running system.

The guest system also contains a watchdog daemon, that is used to monitor the system. It periodically checks network connectivity and the presence of important other daemons. In case such a check fails (or the watchdog itself crashes), the hardware watchdog is no longer triggered, forcing a reboot of the system.

### 4.2 Staging environment

The precise rules for minimal system behaviour allow for testing the behaviour of the guest system before it is allowed in the live network. The *Smoketest* test bench is used for this purpose. First of all, external behaviour in live case is tested. This means, that network connectivity for external clients must be granted even if the client repeatedly associates to different nodes, and management access to the node must be possible. Furthermore the system must show proper reaction to erroneous conditions: Client connectivity must be reestablished if the mesh interfaces lose connectivity. Furthermore the node must reboot if wired connectivity is lost, important daemons crash, or the kernel freezes.

All of these test are relatively simple, and can be performed automatically, even if the network protocol stacks that are used in the experiment, are unknown to the test suite. Obviously this can not prevent abusive behaviour of the researchers, for instance replacing the watchdog daemon on live nodes for a more permissive one to prevent early reboots. But, since full access to the hardware is required, there is no way of preventing intentional policy violations. Yet we think, that most unintentional errors can be confined to a certain level, and hence, prolonged interruptions of node operation can be prevented.

## 5. NETWORK ARCHITECTURE

As we argued in Section 1, network access needs to work transparently for the users no matter what kind of experiment is run in the testbed. Yet the setup must be supported by a very simple rescue mode of the nodes. The simplest setup of a node is to act as an 802.11 access point, bridging

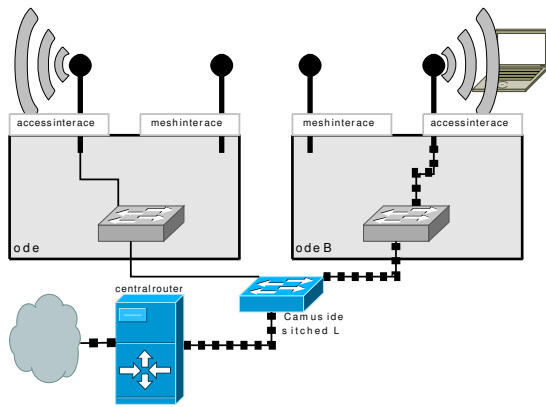


Figure 1: Mesh architecture – rescue mode

all traffic directly to the wire. This obviously also satisfies the transparency requirement.

Figure 1 depicts the simple network configuration in rescue mode: The nodes act as basic access points, with no extra functionality. In this configuration, data from the Internet (dotted path) is flowing from our central router through a VLAN in the switched campus network to the nodes. The nodes bridge it to the wireless access interface.

A reboot of some or all of the nodes into rescue mode should not affect the behaviour of any external components, such as routers or servers. Especially specific configurations of the external components to either mode must not be needed. This is achieved by requiring that also the live mode exposes the same behaviour to external components and emulates a transparent layer 2 switched network.

The network configuration of the live system is shown in Figure 2. The data path from the Internet stays unchanged. In this example, to allow for a multihop environment, Node A is configured as the Internet gateway, and data should only flow to this node from the Internet. This is achieved by the gateway node A answering ARP requests for clients using this gateway with its own wired interface MAC address. Hence, the uplink router sends all data to the mesh gateway. All nodes answer all ARP requests on their access interface with the MAC address of the access interface. This gets traffic from both directions to the right edge node.

Figure 2 also shows part of an experiment, where Location Information Base (LIB) is used, to lookup the node where to client is attached. It then encapsulates the packet in an extra IP header (IPIP), destined to the attachment point. The packet is then routed using the mesh Forwarding Information Bases (FIB) of potentially multiple nodes until it arrives at the attachment point. Here, it is decapsulated (IPIP) and sent to the access interface. The return path is analogous.

This ARP proxy setup comes reasonably close to a completely layer 2 transparent network: Only the ARP caches of external hosts need to expire to reestablish connectivity after a node changes status. Typical timeouts on current operating systems are in the order of 30 seconds, well within reasonable bounds.

## 6. CONCLUSION

With minimal software requirements, we can provide a

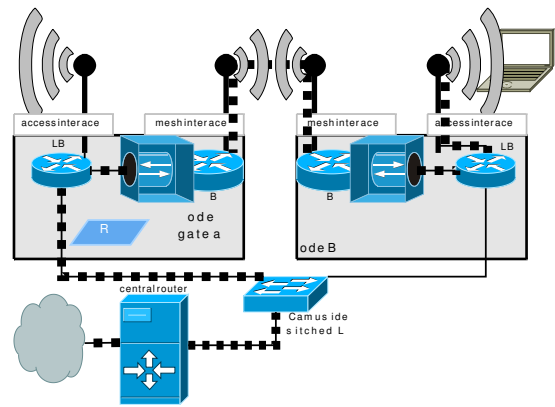


Figure 2: Mesh architecture – live mode

robust system, which allows us to build a testbed, combining the flexibility of a purely research-oriented testbed with the reliability of a production network. We also believe that this system will allow us to provide the testbed to external researchers on a time sharing basis, as it provides quick changeover of configurations and guarantees automatic recovery under all conditions.

## 7. REFERENCES

- [1] J. Eriksson, S. Agarwal, P. Bahl, and J. Padhye, "Feasibility study of mesh networks for all-wireless offices," in *The International Conference on Mobile Systems, Applications, and Services (MobiSys)*, June 2006, pp. 69–82.
- [2] TU-Berlin / T-Labs, "BOWL," <http://bowl.net.t-labs.tu-berlin.de/>.
- [3] "Orbit," <http://www.orbit-lab.org/>.
- [4] D. Raychaudhuri, M. Ott, and I. Secker, "ORBIT radio grid testbed for evaluation of next-generation wireless network protocols," *Tridentcom 2005*, Feb. 2005.
- [5] D. Raychaudhuri, I. Seskar, M. Ott, S. Ganu, K. Ramachandran, H. Kremono, R. Siracusa, H. Liu, and M. Singh, "Overview of the orbit radio grid testbed for evaluation of next-generation wireless network protocols," *IEEE Wireless Communications and Networking Conference, 2005*, vol. 3, pp. 1664–1669 Vol. 3, March 2005.
- [6] U. o. U. School of Computing, "Emulab - total network testbed," [www.emulab.net](http://www.emulab.net).
- [7] Meraki Inc., <http://www.meraki.com/>.
- [8] "Freifunk," <http://www.freifunk.net/>.
- [9] J. Camp, E. Knightly, and W. Reed, "Developing and deploying multihop wireless networks for low-income communities," in *Digital Communities*, 2005.
- [10] R. Ricci, J. Duerig, P. Sanaga, D. Gebhardt, M. Hibler, K. Atkinson, J. Zhang, S. Kasera, and J. Lepreau, "The flexlab approach to realistic evaluation of networked systems," in *Symposium on Networked Systems Design and Implementation (NSDI)*, Apr. 2007.
- [11] "OpenWRT," <http://www.openwrt.org/>.