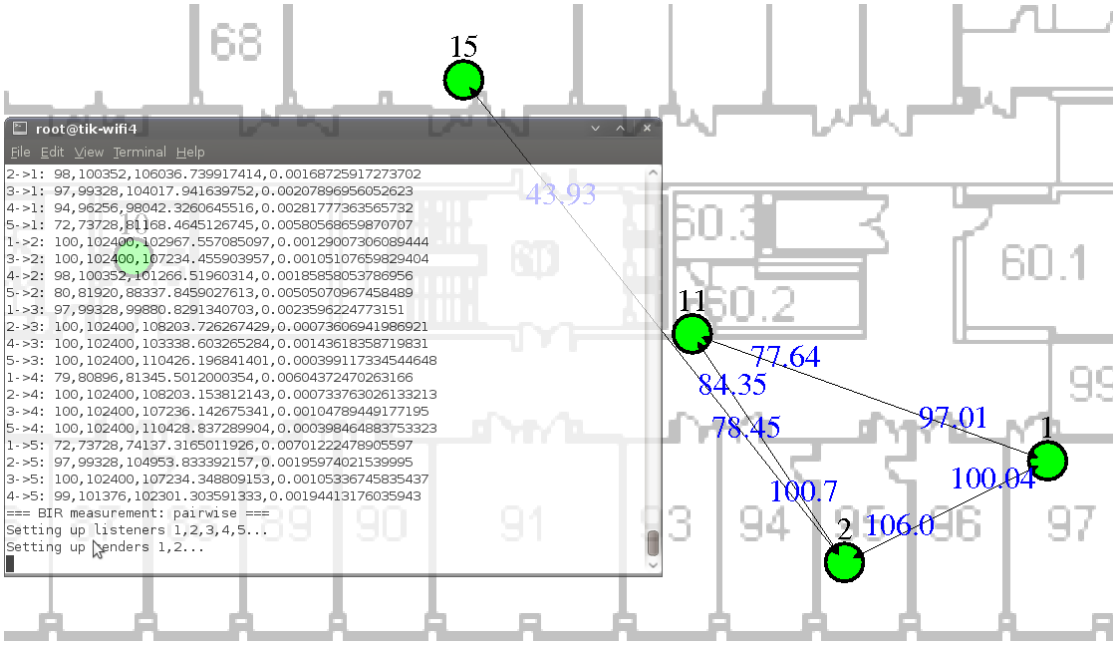
Semester Thesis
June 2009

Development of software tools to facilitate management and experimentation with the TIK wireless testbed



Michael von Känel
Advisor: Merkouris Karaliopoulos

Table of Contents

Abstract

The goal of this project has been to implement a set of user-friendly tools to facilitate the usage, monitoring and management of the TikNet, a static multi-hop wireless network located at the G floor of the ETZ building. The testbed is used both for research and teaching in a real case environment.

First existing testbed software solutions are checked and goals for the new software defined. Since no available software matches precisely the case of the TikNet, a custom made solution had to be devised and implemented.

During this semester project software tools were developed to manage and monitor the testbed. It supports testers in the easiest but still most powerful way to simplify their work on the TikNet. The software module *nodecontrol* is coded in the Ruby programming language and executed mainly from the command line. It provides functionality ranging from simple monitoring tasks to wireless interface management and experimentation. Additional abilities include execution of predefined broadcast measurements as well as simplifications for generic measurement aggregation.

A framework to visualize asymmetric wireless ad-hoc network links was developed and customized for the TikNet. It is coded in Ruby as well and produces vectorized xfig images, which are convertible to any commonly used image format. In order to ease the usage of *nodecontrol*, a module for Webmin was developed. Webmin is a web-based interface for Unix system administration. The module allows users to execute the most basic operations of *nodecontrol* directly through a web browser.

With help of these new tools, experimentation on the testbed is highly facilitated as shown in a demonstration exercise in section 3.1. Since the scope of this work is wide, there are various ways for improvements and extensions to the current software and the testbed in general. Some of them are stated in the last chapter.

1. Project specification

1.1. Introduction and Motivation

Over the past few years wireless technologies such as 802.11 were deployed everywhere around us. Beside the infrastructure mode, several new technologies have been developed or are still under research. Special attention is given to ad-hoc networks in order to improve the performance of wireless networks and develop new applications based on this technology. For example in static multi-hop wireless ad-hoc networks, several nodes cooperate by forwarding each others packets. The great advantage is the easy deployment and the self organization of the nodes. Another branch of wireless research goes into direction of mobile ad-hoc networks (MANET) which often have to deal with sparse network connectivity.

However most of todays wireless research is based on simulations. Simulations have the clear advantage, that they are easy to design. Running a new experiment comes with less effort than in the real world. However, due to the complex nature of wireless signal propagation, it has been shown that the results of even the most common simulators, like the ns-2, differ significantly form real-world experiments[1].

Therefore a functional indoor static wireless testbed was installed at ETH. Every ETH member can run arbitrary real-world experiments on these nodes without the limitations of a simulator. One work, for example, evaluated several routing metrics on the testbed[3] whereas another used it to obtain measurement to assess signal strength-based positioning algorithms [4]. Furthermore the testbed can be used by students for short exercises, where they could obtain hands-on experience with real wireless network equipment. On the other hand, and despite the advantages a wireless testbed can offer, it is a lot more complicated to setup a new experiment compared to a simulator. Therefore powerful tools are needed in order to control, manage and monitor the testbed.

There exist several real-world testbeds, each using its own software heavily based on the specific purpose of the network. To state a few, Emulab[5] is one of the biggest sets of testbeds. It was first developed by the Flux Froub, part of the School of Computing at the University of Utah. The software is open source and nowadays also used on several sites around the world. Then the ORBIT[6] radio grid emulator offers an indoor wireless network testbed. It comes along with a whole experiment control framework coded in the Ruby[7] programming language. It has also been shown that most of these software solutions are too specific to be ported to another network[8]. More information about software solutions of other testbeds can be found in section Error: Reference source not found

Considering the TikNet in the beginning of this work, only simple ssh based connection to each node (or in very simple batch mode) was possible. Setting up new wireless interfaces mostly let crash the machine requiring a manual hard reset. Since the size of the testbed was growing, the administrative effort was also growing, making it time-consuming to set up even simple experiments Nearly none of the functionality the testbed offered was documented. Because it was difficult to configure nodes, experimenters did not shutdown the wireless interfaces properly,which lead to interference with other experiments. This had also negative consequences for the acceptance of the testbed by the rest of the local community.

1.2. The TikNet

The TIK wireless testbed (TikNet) consists of 21 nodes[1] distributed throughout the G floor of the ETZ building of the ETH (see Illustration 1). Every node is equipped with a D-Link wireless card which supports 802.11a/b/g based on the Atheros chip-set therefore running with the Madwifi drivers (madwifi-ng) on a Debian system.

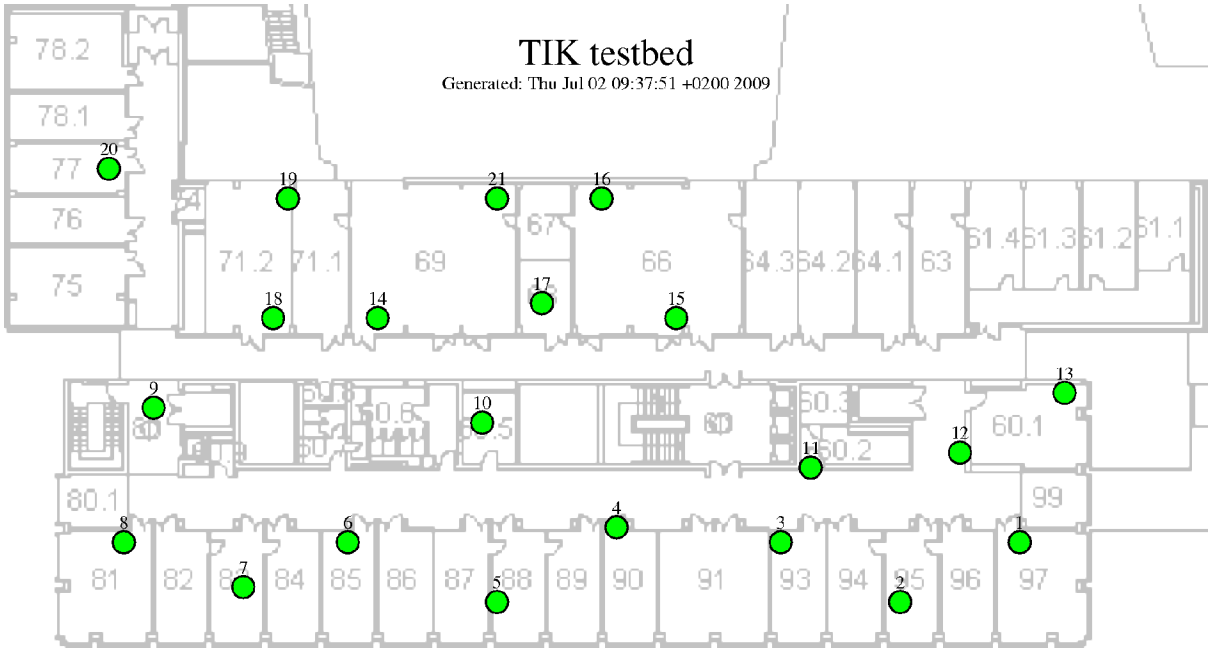


Illustration 1: Snapshot of nodes in the TikNet.

All nodes are accessible over the wire (e.g. using ssh) from a central node (tik-wifi4), which acts as an entry server in this case. Configuration and monitoring is all done over the wire. Only experiments use the wireless interface.

1.3. Requirements

The goal of this project is to implement a set of user-friendly tools to facilitate the usage, monitoring and management of the TIK wireless testbed.

In order to have a guideline several objectives or flexible targets where specified:

1.3.1. Management and monitoring

It should be possible to do basic operations on each node or a set of nodes. These includes testing for connectivity, rebooting of machines, silencing the wireless devices, observing its current wireless settings and running services. A special focus lies on creating new wireless interfaces which should support the full functionality of the Madwifi driver.

In order to improve user friendliness and facilitate the work for those who are not familiar with a Linux shell, a GUI is needed. The usage and the code should be well documented to reduce difficulties to continue this project in another work.

1 Due to maintenance work not all 21 nodes can be used in an experiment (Stand: June 09).

1.3.2. Visualization possibilities

In order to get a quick overview over the testbed it should be possible to visualize several link parameters such as throughput or jitter.

1.3.3. Experimentation control

In order to execute arbitrary experiments faster, the possibility to remotely execute scripts on multiple nodes is needed. Also some basic experiments like link and broadcast measurements need to be available. This includes the automatic setup of the experiment, along with trace collection and some simple result visualization.

Additional tools for more powerful measurements should be tested and installed.

1.3.4. Security

Determine guidelines or access policies for work on the testbed.

1.3.5. Maintenance

In parallel maintenance work is required which includes upgrading of current operating systems, help setting up new nodes and installing and upgrading software tools for experimentation with the testbed.

2. Development of software tools

Probably the easiest solution is to adapt some wireless testbed tools from other testbeds to be used by TikNet. However the freely available Emulab[5] software is designed to run on on a wired network, which can be controlled using the same script language as the ns-2 simulator does. This simplifies the step needed to try out a simulation in the real world. It offers an internal database for trace aggregation and a web interface to control and manage it. Besides several public but wired testbeds, only the original Emulab, located at the University of Utha, provides a wireless testbed. This one matches size and form of the TikNet. However the source for the wireless network was never disclosed. So the Emulab could only be used to aggregate the data. The whole process of setting up new nodes is done in the wireless Emulab by using low level commands or simple scripts, which is not more than the TikNet offered in the beginning. Since setting up this Emulab environment is difficult and time consuming and the only advantage would be a more efficient data aggregation, we decided to not use the Emulab at all.

Roofnet[9] is another experimental 802.11 mesh network currently under development at MIT. It consists of several nodes located at roofs. The custom made madwifi driver and other used software is open source. However the Roofnet is not actually a testbed. It mainly provides a static network using a routing protocol called SrcRR. This does not match our testbed designed to run arbitrary experiments and therefore cannot be used.

ORBIT[6] offers a testbed setup where every node of the network can reach any other node. An experimenter can use the predefined Debian based OS and describe its experiments in the Ruby programming language or run arbitrary code as root for a scheduled time slot. Similarly to Emulab, ORBIT also offers a network which can be seen as a compromise between simulation and real world setup. This is different from a real-world case the TikNet offers. Furthermore it was not possible to get the source of the control software.

There are many more testbeds available. But none of the available software solutions could easily be adapted to the TikNet. It has already been shown that it is difficult to reuse software tools for wireless testbeds.[8] Therefore several possibilities for a new software design where tested.

First the possibility to aggregate the data with the Simple Network Management Protocol (SNMP) was checked. SNMP is a standard for network management. The basic idea is, that a managed device running an agent reports meta-data, organized in a tree architecture (MIB), back to the management system. System variables such as “free memory” or “number of running processes” are measured typically. There are many SMNP solutions available, mostly focusing on monitoring web servers. However finding a MIB which supports 802.11 on Atheros cards turned out to be an unsolvable task. Instead of spending a lot of time in developing our own MIB, it was decided to go into direction of the ORBIT testbed and to start from scratch.

Ruby[7] was the language of choice. To describe it in a few words: It is a dynamic, reflective object-oriented programming language. It has Perl and Smalltalk like features and offers the possible to develop powerful code in few lines of code. This keeps the code understandable and simple to reuse. Because also Ruby itself is simple to learn and understand, it is also favorable to be used in a subsequent work.

2.1. Nodecontrol

In this chapter the general concept of the software responsible for managing and monitoring is described. The code of *nodecontrol* is modularly organized. The sequence of actions during a generic task execution is the following: Fist the appropriate code is generated. This can be given by user input, but most of the time it will be a predefined piece of external code. The action is then passed to the instance responsible for executing, along with a list of affected nodes. Wherever possible the action will be forked and executed in parallel on the remote machines. This speeds up many of the tasks drastically. Finally the processes report back to the parent process, which will take care of correct ordering of the collected data and maybe further analyzes the data. This process can be seen in Illustration 2.

Illustration 2: General concept of task execution.

This general concept is also depicted in the construction of the command line argument:

```
nodecontrol action [arguments] affectedNodes
```

Every node has its id starting from 1 upwards (e. g. tik-wifi4). The affected nodes can be described as a comma separated list of node-Id's, a hyphen separated range or a specific keyword. This allows fast specification of target nodes.

Currently several actions are implemented, some of which are presented here. The rest can be found in the documentation or the program internal help menu (see Appendix).

The most basic action is *test*. It checks whether a given node allows an incoming ssh connection without password, which is required for most of the actions. As in most cases the output can be a simple console output or a picture. The picture output is further described in chapter 2.2. There are more self explaining actions to *reboot* and *ping* nodes.

The more complex action *set*, is used to create a new wireless interface. On Linux systems generally exists the programs “ifconfig” and “iwconfig” which are generic tools to control network and wireless related settings. Theoretically they have the power to control every detail needed to setup and modify an interface. However in practice they are often unable to do so, sometimes with drastic effects such as a complete freezing of a system requiring a hard reset. This was a big problem before this work was done. Most problems are caused by the Madwifi[10] driver which does not fully support these tools and offers a set of its own tools for specific tasks.

Nodecontrol chooses the right tools for the desired configuration and makes it as easy as using the generic tools. Without loosing any of the scripts power a testbed user can simply specify all differences from default configuration and therefore does not need to deal with the complicated setup done in the background. As an example the following command which is used to set nodes 1 to 4 into ad-hoc mode:

```
nodecontrol set mode adhoc 1..4
```

The above command is executed on every of the four nodes with slightly modified commands. First the old virtual wireless interface (VAP) is removed due to stability reasons. Then the MAC is changed in order to re identify a node quickly according to its MAC. Then a new VAP is created according to the specified mode. As for changing the MAC, a separated mode is needed to operate the card in 802.11b. And finally the well known commands iwconfig (for hardware related changes) and ifconfig (for network related configuration) take action:

```
wlanconfig ath1 destroy; macchanger --mac=00:19:5B:00:00:%0id
wifi0 && wlanconfig ath1 create wlandev wifi0 wlanmode adhoc
&& iwpriv ath1 mode 11b && iwconfig ath1 essid tik-grid nick
tik-wifi%id channel 1 rate 11M txpower fixed  && ifconfig -v
ath1 inet 10.0.0.%id netmask 255.255.255.0 up
```

On the opposite side also the need to quickly destroy a set of interfaces exists since they also might interfere with other experiments. This can be done by completely disabling the wlan card (*silent*) or by creating a new interface which does not send beacons at all (*nosbeacon*). It therefore is completely silent if there are no packages to be sent, but it is still part of the network.

Another action worth mentioning is called *tudp*. The program called Tudp is a custom measurement tool written by Yang Su at the computer science department of ETHZ. It is used to perform mainly MAC broadcast measurements. Why this tool is used, is explained in chapter 2.4. Without arguments a set of nodes broadcasts and another set listens. But many other patterns such as pairwise or parallel broadcasting are available. Besides the experimenting part the collected data is analyzed and basic statistics such as packet loss, throughput or jitter is displayed to the user. Even outputting the gathered statistics in form of a picture is possible as explained in the following chapter. If a experimenter wants to calculate more advanced stuff, the raw data output files can be used which are automatically collected at the entry node.

There are even more advanced pattern such as *bir*, which calculated the Broadcast Interference Ratio[16] (BIR). In order to calculate the BIR a series of tudp experiments is done starting by broadcasting from every node and listening at the others. Then each pair of nodes starts broadcasting. The results are then used to calculate the BIR of each pair of links.

Even if a needed experiment is not supported by *tudp*, an experimenter most possibly wants to collect packages at a set of nodes. So after setting up an experiment, the *tshark* action might be useful to collect data at every desired node and sending this back to the root node.

There are lots of other powerful functions for experimenters. The reader is here referred to the script internal help menu, the appendix with the user documentation about *nodecontrol* or the TikNet wiki[17].

```
nodecontrol help [action]
```

2.2. Visualization framework

For visualization, primarily one software tool called ViTAN[11] was tested. The ViTAN project offers a tool written in Perl and has a well documented technical report.[12] An input file describes positions of nodes and link qualities out of which an Xfig file is generated. Xfig[13] is an open source vector graphics editor which runs on most UNIX-compatible platforms. It is also possible to export to non-vectorized image formats. However the tool had several problems in displaying special graphs such as the zero-graph, synchronization issues which generated finitely large files and link qualities must be integers from 0 to 255, just to name a few.

We decided to develop one's own visualization framework because it seemed to be easier than fixing all limitations of ViTAN. Fortunately lots of code could be reused. As ViTAN does, the framework can output files in the native Xfig format[14] and convert them to Portable Network Grapgics (png).

A network graph is first a Ruby object. Step-by-step nodes can be added in different states (such as “green” meaning node is fine). Then unidirectional links between these nodes can be added in several formats. For example to display all throughputs, one can enter them as bits/s and let them convert automatically depending on the input size to Mb/s.

2.3. Webmin wrapper

Webmin[15] is a web-based interface for system administration for Unix. Webmin comes per default with various modules to configure many operating system internals, such as user access, services, disk quotas and many more. Webmin is open source and largely based on Perl, running its own web-server. It is installed on all nodes to manage simple tasks, mostly for users not familiar with a UNIX shell. The service can be accessed on TCP port 10000 (per default), using a regular web browser.

In order to improve user friendliness a Webmin module was developed in Perl which acts as a wrapper for *nodecontrol* described in chapter 2.1. If *nodecontrol* runs on a given node under a given user, it also runs via this wrapper. In the current setup of the testbed one should run it on the entry node as root.

The idea was to create a web interface for the most basic commands of *nodecontrol* such as testing if a set of nodes are up, setting a new interface and doing some predefined broadcast measurements for a quick snapshot over the current state of the testbed. The module is capable of using the output pictures created by using the visualization framework and displaying them to the user right in the web browser.

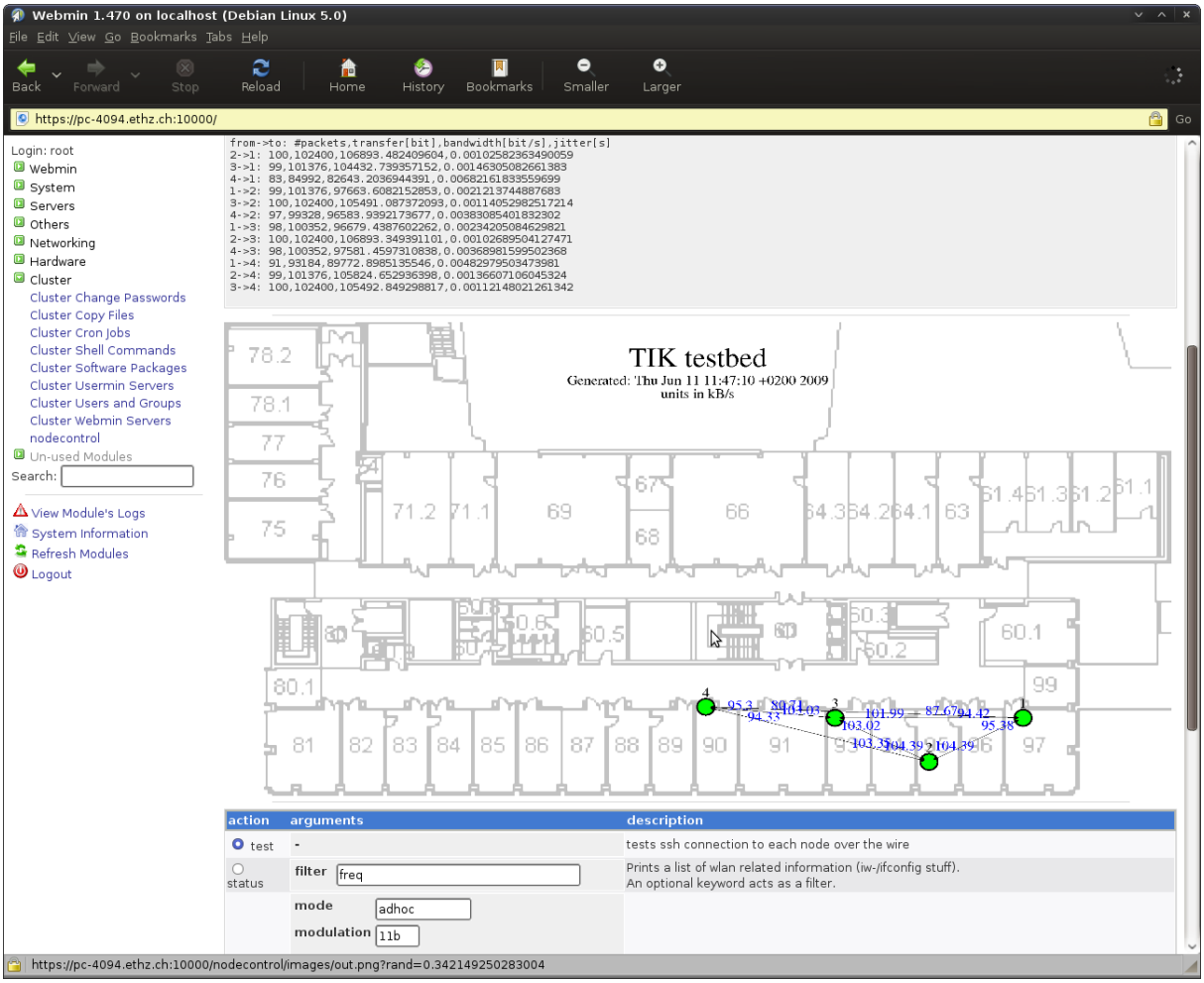


Illustration 3: The Webmin module for nodecontrol in action.

2.4. Maintenance and challenges

The wiki[17] of the TikNet not only contains a set of hints and scheduling about this work but should mainly be a central place of information for every user of the testbed. Hence all necessary documentation about several software tools can be found there. We also provide some info in the Appendix. Specific maintenance information for example how to add a new node to the TikNet was written during this work. Concerning maintenance, some small example problems encountered are presented in the following paragraphs.

Because there are lots of different wireless networks at ETH operating in 11b/g there is lots of noise around the 2.4GHz band. To minimize interference for the TikNet and also not to disturb other experiments in 11b/g too much, it was decided to operate the testbed in 11a per default. Sadly because 11a uses the 5GHz band it has problems propagating through the thick walls. In fact the range is reduced drastically such that no connected graph was possible any more. If an experimenter wants to have a sparse graph this is an option, but for most cases 11a can not be used.

Another problem encountered was related to link measurements. A standard tools to measure

link quality in IP based networks is *iperf*[18]. Besides TCP it also supports UDP data streams. In order to measure all links of the TikNet, some capability to do broadcast measurements was needed. Iperf does not directly support broadcasting, but has the ability to send multicast packages for which we went first. Sadly iperf's multicast is IP based, therefore it sends the package only once to the card, but the card sends one package to each member of the multicast group. Therefore this does not improve number of packages sent. Other well known tools like *mgen* have similar problems. Additionally iperf had serious stability problems while performing UDP experiments.

A solution was found with *tudp* which is a custom measurement tool written by Yang Su at the computer science department of ETHZ for implementing MAC broadcast transmissions. Since it is only a simple noise generator, it does not offer any data analysis. Simple statistics are offered by the *nodecontrol* or, to completely work around this, one can monitor all packages with an additional program like *tshark* and use standard analysis tools.

For experimenting, one likes to have as few unknown variables as possible. Per default all cards use power and rate adaption which makes experimenting less predictable. Disabling power control could be done easily but rate adaption is hard compiled into the Madwifi driver. The current version of Madwifi (v0.9.4) uses the sample algorithm[2]. It turned out to be infeasible to completely disable rate adaption, but the statistics about what rates are used for a specific MAC can be found as following:

```
cat /proc/net/madwifi/ath?/ratestats_* | grep "MAC" -A 5
```

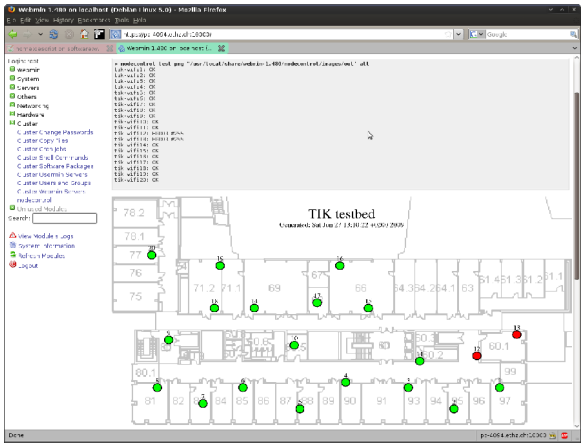
2 SampleRate chooses the bit-rate it predicts will provide the most throughput based on estimates of the expected per-packet transmission time each bit-rate. SampleRate periodically sends packets at bit-rates other than the current one to estimate when another bit-rate will provide better performance. [19]

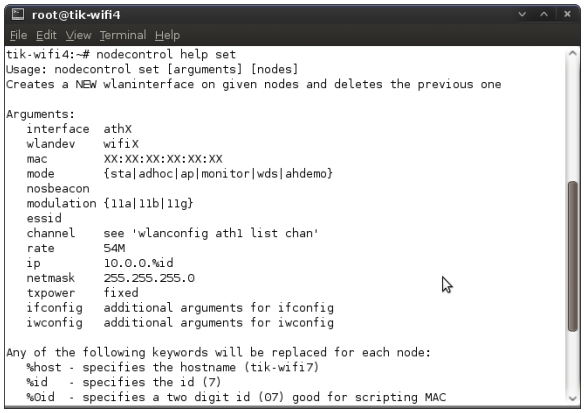
3. Demonstration and discussion

3.1. Example problem

To illustrate the work on the testbed with the new functionality, we will solve a Lab exercise of the ATCN course which basically contains the following steps.

1. setup a new ad-hoc network on a subset of nodes
2. verify the configuration and test the connectivity
3. paint the connectivity graph
4. calculate the broadcast interference ratio (BIR) of each link combination

To get a quick overview of the nodes we will be using, we go to the Webmin page and run a *test* on all nodes. We immediately realize that nodes 11 and 12 are down (mentioned in the command output and painted in red rather than green) and the rest of the nodes is ready to receive actions. We will take the subset of nodes 1 up to 5 for this exercise.	

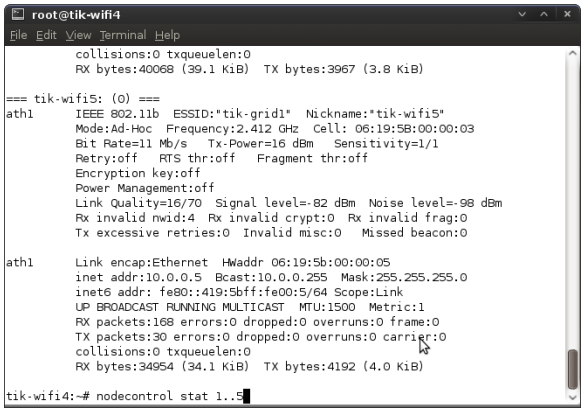
Even though we could do the following steps also within Webmin we will go for the console. First we need to login to the entry server.
`ssh root@tik-wifi4`
 	
In order to keep the node clean we change to our user directory.
`cd /home/testuser`
 	
Then we setup a new grid. Sadly we forgot how to setup a new grid network so we checkout the help.
`nodecontrol help set`
 	
For every action there is a help chapter, so this will not be mentioned during this exercise anymore.
	

We create new interfaces on our set of nodes with a non default ESSID. The script takes care of the whole rest of the settings.

```
nodecontrol set essid tik-grid1 1..5
```

To verify the functionality we also check the settings on all nodes quickly.

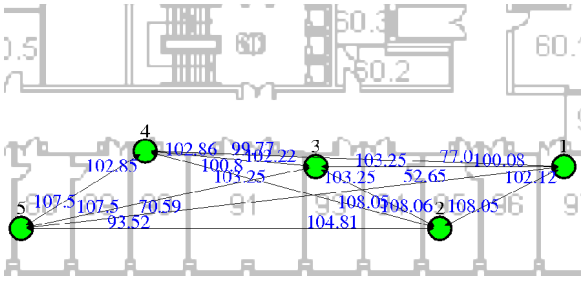
```
nodecontrol stat 1..5
```



We now do some throughput measurements using tudp and draw this as a picture.

```
nodecontrol tudp png ex1 1..5
```

A predefined broadcast measurement will be performed and the picture is stored to ex1.png. In the current case we get a fully connected graph with high loss rates for nodes far apart.

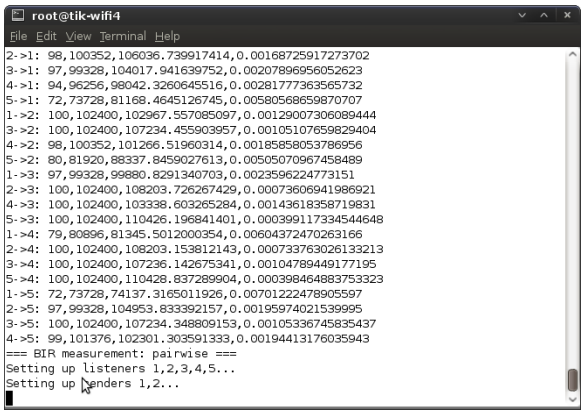


The last step includes to calculate the BIR which can be easily done.

```
nodecontrol tudp mode bir log ex1 1..5
```

This will do all measurements automatically: First let all nodes broadcast sequentially and then each possible pair of nodes simultaneously to calculate the BIR of each link combination.

Optionally we output the raw data of the experiments to log files with the prefix “ex1”. For example in the case of pairwise broadcast on nodes 1 and 3 this created the file “ex1_bcast1,3_node2.log” which contains the packets received at node 2.



Now the experiments are finished. In order to silent the cards to minimize interference to other experiments, we can kill the wireless interface

```
nodecontrol silent 1..5
```

...or simply disable beaconing but letting the network unchanged.

```
nodecontrol set essid tik-grid1 nosbeacon 1..5
```

3.2. Self-assessment of project goal achievement

The rationale behind certain decisions taken during development is already described in section 2. about software development. Therefore the discussion in this paragraph concentrates on how and in what amount the goals are reached.

A great step was done in terms of facilitating the management of the testbed. With help of the developed tools it is easy to get a quick overview of the status of the testbed nodes. For example, creating new interfaces works around all difficulties and lets the user execute powerful commands with very simple actions. For users not so familiar with a shell, a web GUI, integrated in Webmin as a module, was created. This module only provides basic functionality to work with *nodecontrol*. If the future shows that most of the users only want to use the web-interface, one needs to improve the plugin as stated in section 4.

In order to visualize information about the testbed, a new framework was developed optimized for showing ad-hoc network links. This is very simple to use on the predefined static TikNet nodes but still can be used in any network setup. Sure a whole semester thesis could be written about how to visualize the TikNet in an optimal way and there is plenty of room for improvements (see future work).

Besides management, our work has enabled simple experimentation using the testbed. Most time was spent into broadcast measurements. It is difficult to provide user friendliness without giving away too much power such that a simple measurement can be performed without much knowledge; nevertheless, very focused experiments are now feasible with the testbed. Currently this is solved by showing simple statistics by default with optional raw output. In a similar manner it is possible to easily monitor packages arriving at each node and collect them.

Security is currently an issue on the TikNet. Because all changes of the wireless card and also some noise generators operate on kernel level, for most of the actions performed, root rights are required. Other testbeds address this problem by the use of an entry server. Complete user management, logging and scheduling of experiments can be done on that single computer. The problem of security was not yet addressed directly mainly because of the lack of time. However some guidelines on how experimenters have to behave when using the testbed are given in the Appendix in the end of this report.

In summary, the goals concerning management and simple experimentation are reached. Concerning usability and visualization capabilities some big steps where taken. The security aspects of the testbed and some automation in scheduling experiments, if needed depending on number of testbed users, will be worth a separate Semester Thesis; we give some hints in the next section.

4. Possible testbed functionality extensions

There is much work to be done in order to improve the tested usability further. About lots of points mentioned here, a whole semester thesis could be written.

Nodecontrol is the central software of this semester thesis and some of its possibilities are described in this work. A first step, which must be taken as soon as many people are using the software, is bug fixing. Several steps are taken to minimize bugs beginning from the script language to excessive testing, however some bugs cannot be avoided. Performance might become an issue for big experiments because most of the results are returned directly to the server node and are kept in RAM until the experiment is finished. In order to facilitate ensuing work on the software, the code is accurately documented and commented. A next step is to further improve its abilities. For example the Click Modular Router[20] could be integrated nicely. In a final step a generic script language describing experiments would be desirable, like it is done in ORBIT. However the simplicity of the current software should be kept.

If the future shows that a better way to collect data is needed, one might reiterate on the software used in Emulab. The tools developed in this work can easily coexist with such a new approach.

The visualization framework is currently mainly usable within a Ruby script. One could add interfaces to use it with other API's. Then it could be further generalized such that it is as simple to use with any network as it is currently with the TikNet. Additionally the visualization of the asymmetric links is far from perfect because they often overlap for dense graphs making it hard to read the numbers out of the picture. Finally, it would be nice to get rid of the xfig libraries and do the whole process of creating the image in Ruby. A possible extension could include the drawing of conflict graphs of a network.

Security is still an open problem. Our recommendation would be to have this issue treated in the context of another semester thesis. A first step could include an actual entry server apart from the testbed. Access on the testbed should only go through this to facilitate logging and user management. Next steps could include an automated experiment scheduling along with complete shutdown of nodes for idle periods.

Reference

[1] D. Kotz, C. Newport, R. S. Gray, J. Liu, Y. Yuan, and C. Elliott. “Experimental evaluation of wireless simulation assumptions.” Proc. Of MSWiM 2004, October 2004.

[2] G. Sotiropoulos, “ Experimentation and evaluation of routing metrics for multi-hop ad-hoc wireless networks on an indoor wireless testbed ”, Semester thesis ETH, June 2007

[3] G. Parissidis. Interference-aware routing in wireless multihop networks. PhD Dissertation, ETH Zurich, April 2008

[4] K. Farkas, T. Hossmann, F. Legendre, B. Plattner. Link Quality Prediction in Mesh Networks, Computer Communications, Elsevier, No. 31, pages 1497-1512, May 2008

[5] Emulab Network Emulation Testbed, http://www.emulab.net

[6] ORBIT, http://orbit-lab.org

[7] Ruby programming language, http://www.ruby-lang.org

[8] P. De, A. Raniwala, S. Sharma, T. Chiueh, “Design Considerations for a Multihop Wireless Network Testbed ”, Stony Brook University, October 2005

[9] Roofnet, http://pdos.csail.mit.edu/roofnet

[10] mafwifi-ng, http://madwifi-project.org

[11] ViTAN, http://www.acticom.de/en/products/vitan/

[12] F. Fitzek P. Seeling M. Reisslein∗M. Zorzi, “Visualization Tool for Ad Hoc Networks - ViTAN v1.1 ”, acticom GmbH, Arizona State University, Università di Ferrara, February 2003

[13] Xfig, http://www.xfig.org/

[14] Fig Format 3.2 specification, http://epb.lbl.gov/xfig/fig-format.html

[15] Webmin, http://www.webmin.com

[16] J. Padhye, S. Agarwal, V. N. Padmanabhan, L. Qiu, A. Rao, B. Zill, “Estimation of Link Interference in Static Multi-hop Wireless Networks”, University of Texas, Austin & University of California, Berkeley, 2006

[17] TikNet wiki, http://tiknet.ee.ethz.ch

[18] Iperf project page, http://sourceforge.net/projects/iperf/

[19] J. C. Bicket, “Bit-rate Selection in Wireless Networks”, Massachusetts Institute of Technology, February 2005

[20] Click project page, http://read.cs.ucla.edu/click/

Appendix

Most parts of this Appendix can be found in the TikNet wiki (tiknet.ee.ethz.ch visited: June 2009).

nodecontrol (user documentation)

```
nodecontrol action [arguments] affectedNodes
```

```
Actions:
   help - get specific help ('nodecontrol help [action]')
   ping - pings nodes from the current machine
   test - tests ssh connection to each node
   silent - kills the wlan interface
   reboot - reboots nodes
   set  - sets new wlan interface
   stat - prints wlan related information
   exec - executes remote commands on nodes
   tudp - broadcast measurements
   iperf - link measurements (under developement)
   tshark - collect packages

Nodes:
   Specifies the affected nodes by a comma separated list (no spaces).
   Accepted values are integers, ranges or simply the keyword 'all'
   Example: 1,5..10

Placeholders:
   Any executed command can contain placeholders affecting each node
differently:
   %id   - will be replaced with the node number (1,2,3...)
   %host - replaced by the node host name of the wired interface (tik-
wifi1, tik-wifi2,...)
   %id0  - replaced with a two digit id (01,02,...20)
```

help

A good starting point. nodecontrol help lists you a list of actions, whereas nodecontrol help [action] prints action specific information.

```
nodecontrol help [action]
```

ping/test/silent/reboot

a first step to get an overview over the shape of the testbed.

```
nodecontrol test all
nodecontrol ping all
```

set

Use set to create a new VAP and interface on a set of nodes. After a reboot any changes are lost.

```
tik-wifi4:~# nodecontrol help set
Usage: nodecontrol set [arguments] [nodes]

Creates a NEW wlaninterface on given nodes and deletes the previous one

Arguments:
   interface  athX
   wlandev    wifiX
   mac        XX:XX:XX:XX:XX:XX
   mode       {sta|adhoc|ap|monitor|wds|ahdemo}
   nosbeacon
   modulation {11a|11b|11g}
   essid
   channel    see 'wlanconfig ath1 list chan'
   rate       54M (or 11M on 11b)
   ip         10.0.0.%id
   netmask    255.255.255.0
   ifconfig   additional arguments for ifconfig
   iwconfig   additional arguments for iwconfig

Any of the following keywords will be replaced for each node:
   %host - specifies the hostname (tik-wifi7)
   %id   - specifies the id (7)
   %0id  - specifies a two digit id (07) good for scripting MAC

Examples:
   nodecontrol set mode adhoc 1..5
      - sets nodes 1-5 in adhoc mode
   nodecontrol set mode ap txpower 15 iwconfig 'frag 512' 10
      - creates AccessPoint on node 10 with limited power
   nodecontrol set mode adhoc nosbeacon 12..16
      - nodes do not send beacons but are still part of the network
```

To use the default options just run set without additional arguments:

```
nodecontrol set [nodes]
```

More correct: ‘set’ assumes default values for every unspecified argument. Currently the default values are as follows (likely to change):

```
interface  ath1
wlandev    wifi0
mac        00:19:5B:00:00:%0id
mode       adhoc
modulation 11b
essid      tik-grid
channel    1
rate       11M
ip         10.0.0.%id
netmask    255.255.255.0
ifconfig   ""
```

```
iwconfig   ""
```

stat

```
Usage: nodecontrol stat [keyword] [nodes]
```

```
Prints a list of wlan related information (iw-/ifconfig stuff).
An optional keyword acts as a filter.

Examples:
   nodecontrol stat freq all
      - prints an overview over used frequencies
```

exec

```
nodecontrol exec [-l][-q][-s] "script" [nodes]
```

```
Executes a (bash) script on every node. This is done in parallel by
default without displayed stdio during execution. Alternatively it can
be processed sequentially with direct output.

Arguments
   -l/--local: Execute on the server and not on the client, (eg. ping)
   -q/--quiet: only print exit state
   -s/--sequential: run node after node with direct output
```

tudp

```
Usage: nodecontrol tudp [arguments] [listeners] senders
```

```
Executes broadcast measurements on a set of nodes. If no listeners are
provided
listeners=senders is assumed

Arguments
   mode {sequential|parallel|pairwise|BIR} defines the order of the senders
      - sequential sending is default. parallel might not perfectly start
at the same time.
        BIR as a special case additionally calculate the broadcast
interference ratio
   sender "tudp sender arguments"
   receiver "tudp receiver arguments"
   log "prefix"
      - creates logfiles of the receivers(listeners) labeled
[prefix]_nodeX.log
   png "pictureName"
      - creates a picture named [pictureName].png/.xfig

Example
   nodecontrol tudp log "myExperiment" png "myPic" 1..4
      - broadcast measurement on nodes 1 to 4 creates logfiles
        labeled myExperiment_nodeX.log and writes a picture to
```

```
      myPic.png containing the a link overview
   nodecontrol tudp mode parallel 1..4 5,6
      - broadcast measurement sent from node 5 and 6 in parallel.
        Packets are received at nodes 1 to 4 and no logfiles are written.
   nodecontrol tudp mode bir 1..4
      - calculated the broadcaste interference ratio (BIR) on all possible
        links between node 1 to 4.
```

tshark

```
Usage: nodecontrol tshark [tshark argument] [nodes]
```

```
Arguments
   per default all packages of the subnet are caputred or whatever
   specified here.

Example
   nodecontrol tshark 1..3
      - caputes all packages of the subnet on nodes 1 to 3 and
        prompts for an output file for the log-file
```

ssh

Every node in the TikNet is accessable via ssh. The hostnames range from tik-wifi1 up to something around tik-wifi20.

Login without password

- Login as user or root to the entry server. If you are within the ETH network this is tik-wifi4 (you need a password). For externals the server is currently pc-4094.ethz.ch

  ```
  ssh -C root@pc-4094.ethz.ch
  ```

- From there you can login as root with ssh root@tik-wifiX to all the machines that belong to the TikNet, without having to type a password.

Note that for now no other combination is possible

- you cannot login to another machine and from there to the others (without password)
- you cannot login as user to the other machines (without password)

Webmin

Webmin is a web-based interface for system administration for Unix. Using any browser that supports tables and forms (and Java for the File Manager module), you can setup user accounts, Apache, DNS, file sharing and so on.

Webmin consists of a simple web server, and a number of CGI programs which directly update system files like /etc/inetd.conf and /etc/passwd. The web server and all CGI programs are written in Perl version 5, and use no non-standard Perl modules.

http://www.webmin.com/

Usage

Webmin is accessable on every node on the default port 10000. You can access the entry node as follows (note that you might get a warning because the ssl certificate is self signed).

- https://pc-4094.ethz.ch:10000/

Coding remarks

The module is located in

```
> cat /etc/webmin/miniserv.conf | grep ^root
root=/usr/local/share/webmin-1.470
```

To install it manually place the nodecontrol folder into the mentioned folder, purge the webmin cache and add the module to the trusted ones.

```
rm /etc/webmin/module.infos.cache
nano /etc/webmin/webmin.acl
```

Basic infos are found under http://doxfer.com/Webmin/ModuleDevelopment.

Setup a new tik node

The following guide is meant to setup a fresh debian environement as part of the tik-net. Most commands need to be executed as root (su).

basic access

To change the hostname permanently edit the hostname file and run the startup script (tik nodes are labeled tik-wifi1 to tik-wifi20):

```
nano /etc/hostname
/etc/init.d/hostname.sh
```

Allow remote access from tik-wifi4 via ssh without tiping the password (not only for comfort reasons, this is required by some scripts to work):

```
scp tik-wifi4:/home/user/.ssh/id_rsa.pub /root/.ssh/user@tik-wifi4.pub
cat /root/.ssh/user@tik-wifi4.pub >> /root/.ssh/authorized_keys
```

keep you system up2date

```
apt-get update && apt-get -y upgrade
```

eventually also upgrade the kernel headers

```
apt-get dist-upgrade
```

Madwifi - wireless drivers

Fist make sure you have the non-free sources: edit the sources.list

```
nano /etc/apt/sources.list
```

...and add the following lines if they are not yet present:

```
# Unstable
deb ftp://ftp.au.debian.org/debian unstable main contrib non-free
deb-src ftp://ftp.au.debian.org/debian unstable main contrib non-free


# Testing
deb ftp://ftp.au.debian.org/debian testing main contrib non-free
deb-src ftp://ftp.au.debian.org/debian testing main contrib non-free


# Stable
deb ftp://ftp.au.debian.org/debian stable main contrib non-free
deb-src ftp://ftp.au.debian.org/debian stable main contrib non-free
```

(re)add madwifi drivers (also if they get lost after kernel upgrade):

```
apt-get update && apt-get -y install madwifi-source madwifi-tools
m-a prepare
m-a a-i madwifi
```

```
reboot (oder modprobe athX)
```

In cases where apt-get fails (because the system is terribly outdated) try to increase its cache: fix apt-get update memory problem: add

```
APT::Cache-Limit "16777216";
to /etc/apt/apt.conf.d/70debconf
```

Additional software

To get a basic set of tools we use the debian dispo.

```
apt-get -y install iperf tshark tcpdump ruby macchanger
```

Additional tools, not so easily available, are the following:

tudp

most simple way is to just use the precompiled version of node 4 (since its standalone & I don’t have a clue where to get the sources from):

```
scp tik-wifi4:/usr/local/bin/tudp /usr/local/bin/tudp
```

click

In order to get the newest click version ckeckout the pit repository. First make sure git and basic labraries for compiling are installed.

```
apt-get install git-core build-essential
```

then get the repo & compile:

```
git clone git://read.cs.ucla.edu/git/click /home/user/click
cd /home/user/click
./configure
make -j2
make install
```

Similary you get the additional click-packages http://www.read.cs.ucla.edu/click/git.

mgen

get the newest version online: http://cs.itd.nrl.navy.mil/work/mgen/ or directly http://downloads.pf.itd.nrl.navy.mil/mgen/. Then move the binary to the local bin folder.

```
wget http://downloads.pf.itd.nrl.navy.mil/mgen/mgen4/linux-
mgen-4.2b4.tgz
```

```
tar -xzf linux-mgen-4.2b4.tgz
cp MGEN/mgen /usr/local/bin/mgen
```

webmin

get the newest version on the official page. get and install the precompiled deb package like

```
wget http://prdownloads.sourceforge.net/webadmin/webmin_1.470_all.deb
dpkg -i webmin_1.470_all.deb
```

Then its accessable over https://tik-wifiX:10000/.

new entry node

If a new node needs to act as an entry server, several additional steps are needed.

ssh

Export the public key to all tik nodes as described above.

nodecontrol

Nodecontrol depends only on Ruby stdlib. However if one wants to use the png output functionality xfig libs are needed.

```
apt-get install ruby xfig
```

Move the script to /usr/local/bin or create a link.

In order to use it with webmin, install the nodecontrol module nodecontrol.wbm.gz manually or

1. Login to Webmin as root, and go to Webmin → Webmin Configuration → Webmin Modules
2. Select the From ftp or http URL option, and enter the URL of nodecontrol.wbm.gz into the adjacent text box
3. Click the Install Module button

Security

Currently security is an issue on the TikNet because every experimentor needs root access. A better solution is shortly described here in order to give some ideas about a future work. Till then everyone working on the testbed is asked to follow the following rules.

guidelines

- On tik-wifi4: do **not** save any data to /root or anywhere else on the system.
  If you need to save files create a new user

```
adduser [your_ee_username]
```

  and then save any data to /home/your_ee_username/

- On any other node: do **not** save data anywhere except /home/user/ and do **not** create new users. /home/user is likely to be purged, so make sure you aggregate all your data on tik-wifi4

privacy and use policy

- **Receiving**: You may listen to and monitor anything you like. You may not disclose, on purpose or accidently, the IP addresses or DNS names of connections used by others. If contemplating disclosing the applications used by others, or similarly private information, you must first get explicit approval from testbed-ops. In case of any doubt about disclosure, contact testbed-ops. If you happen to observe any other information that a user would expect to be private, such as plaintext Web passwords or account names, you will not exploit that information, and you will take care not to let it leak out publicly, e.g., in log files.
- **Transmitting (1)**: Do not transmit on channels that another experiment on the testbed is using, unless it's your own. Check the wiki page for maintenance work/experiments currently done.
- **Transmitting (2)**: Do not flood a wireless network with non-responsive traffic for any significant period of time. The following channels are "production networks" used by others at this location, so are more restricted. You may not send "large" amounts of traffic on them, and may send only low rates of non-responsive traffic.

— *mostly taken from Emulab*

would be nice

- A separated entry server apart from the nodes (no wireless). All user management and logging happens on this node.
- No root login possible, a user account is required (manually or automatically for a given slot)
- login on nodes (root) is only possible from the entry node
- Automated purging of experimenatiton data on nodes.