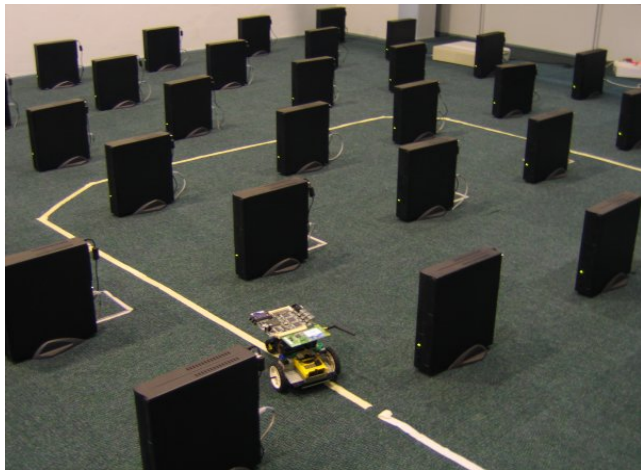


Meraka Wireless Mesh Testbed



**Workshop on building
experiments**

Day 1: Learning the tools

Presented by David Johnson and Kevin Duff

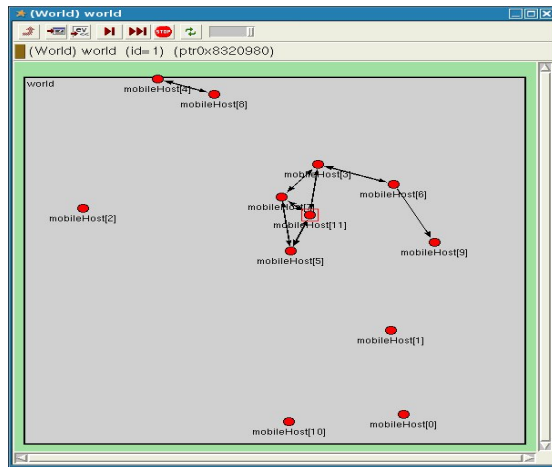
Contents of Talk

- ◆ **Overview of the lab environment**
 - + Background
 - + Accessing the lab
 - + Booting OS on nodes
 - + Controlling the nodes
 - + Installing your own software on the nodes
- ◆ **The wireless interface**
 - + Modes/rates/channels ...
 - + iwconfig, ifconfig, iwpriv, iwlist
- ◆ **Network simulation tools**
 - + The Traffic Control (tc) tool
 - + The firewall (iptables) tool

Outline

- ◆ **Measurement/traffic generation**
 - + iperf tool
- ◆ **Mesh protocols**
 - + OLSR
 - + AODV
 - + B.A.T.M.A.N.
 - + Dymo
 - + Srcr
- ◆ **Putting it all together**
 - + Writing control scripts (Perl, Python)
 - + Plotting (Gnuplot)
- ◆ **Tutorials**
 - + Your turn to do some work

Background: Motivation for the mesh testbed



◆ Problems with Simulations

- + Physical/MAC model oversimplified
- + Network simulation model inconsistencies [1]
- + Not all 802.11 characteristics built in
- + Perfectly symmetrical links



◆ Problems with real world networks

- + Time consuming to build
- + Difficult to upgrade/debug/maintain
- + Measuring process effects measurement
- + Users complain if the network is down for experiments

Background: Similar labs



- ◆ **Rutgers – Winlab – Orbit**
 - + 8x8 Grid and 20x20 Grid
 - + 802.11 based (Atheros 5212)
 - + Can extend to Bluetooth/UWB/Zigbee
 - + Range limited by Raising the noise floor with AWGN or MAC filtering [3]
 - + Available to any researchers worldwide using scheduler
 - + Mobility supported by virtual mobility and in the future using robots
 - + Routing protocols or entire OS can be changed

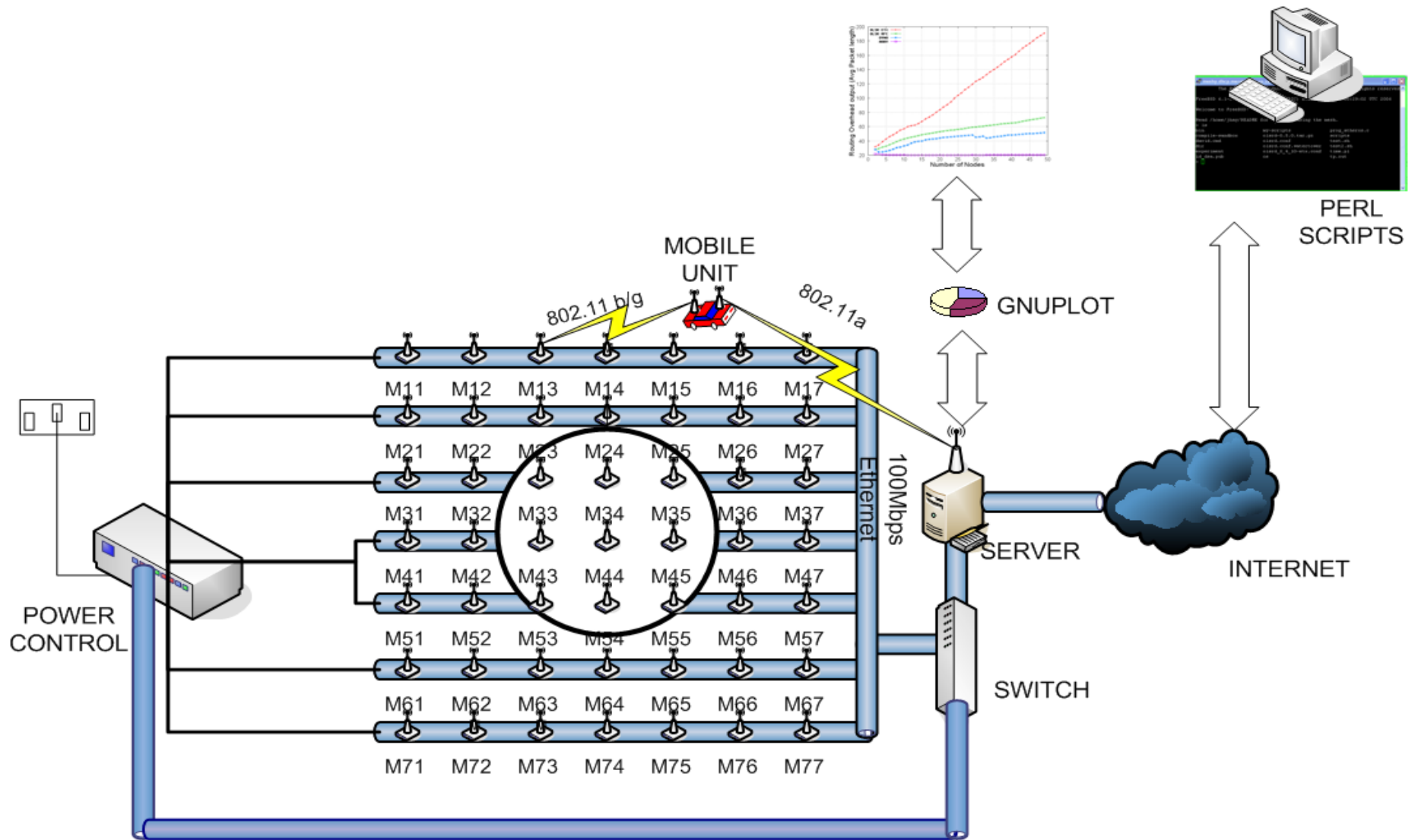
[3] S.K. Kaul, M. Gruteser, and I. Seskar, “Creating Wireless Multi-hop Topologies in Space-Constrained Indoor Testbeds Through Noise Injection”, <http://www.winlab.rutgers.edu/~sachin/papers/Topology-noise.ps>

Author: D L Johnson and K. Duff

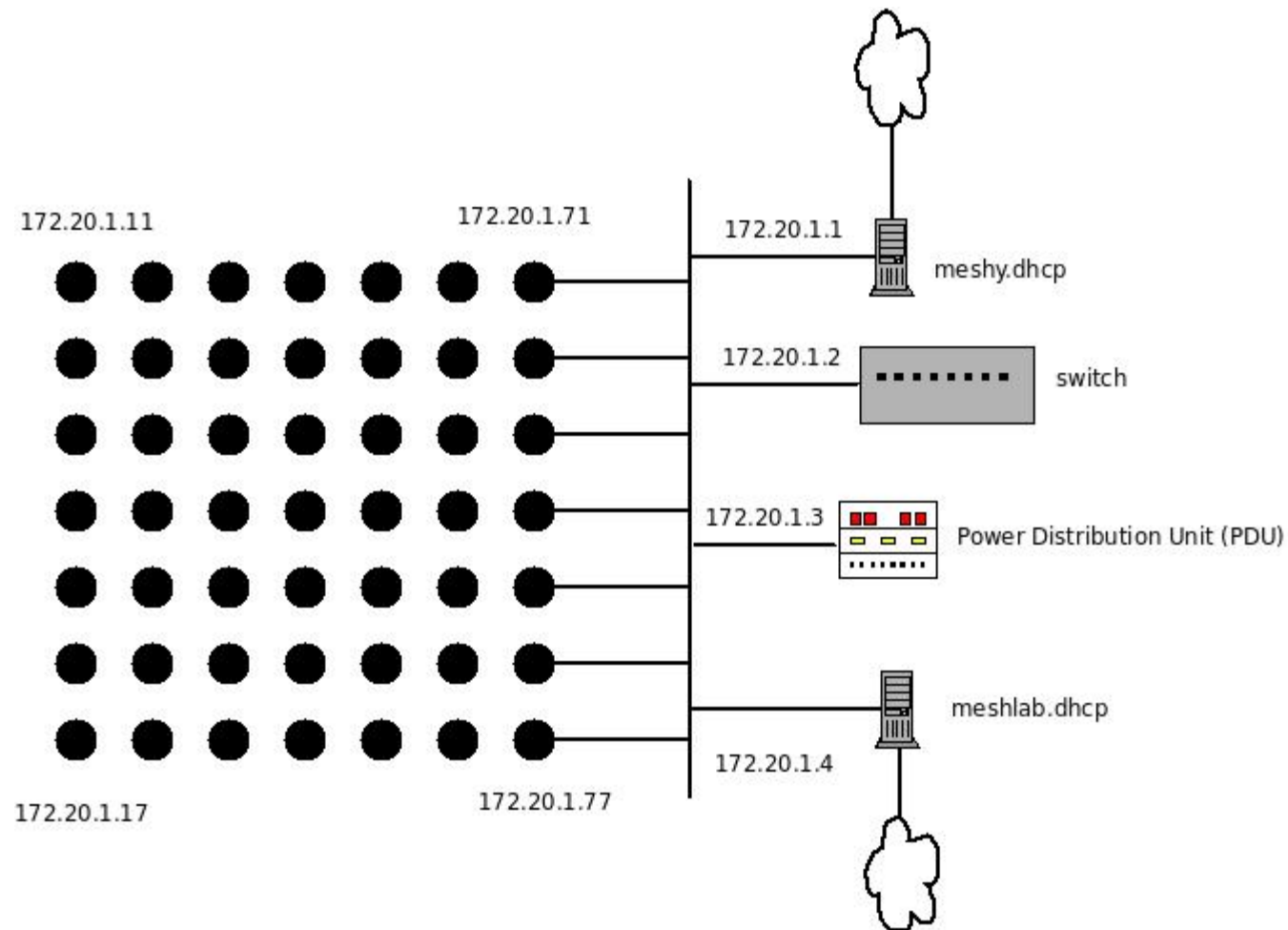
Background: What can the mesh testbed do

- ◆ **Benchmark ad hoc routing protocols**
 - + Throughput, packet loss and delay
 - + Test route optimization and hop count
 - + Routing overhead
 - + Test gateway location mechanisms
- ◆ **Testing network behaviour under load**
 - + Effect of load on ad hoc routing
 - e.g. Number of supported VoIP calls over a multi hop network
- ◆ **Test various radio technologies**
 - + WiFi (802.11 a/b/g)
 - + If other radios are plugged in (Bluetooth, Zigbee, UWB)
 - + Cognitive radios to be supported in the future
 - + Currently supports 2.4GHz - upgrading to dual band (2.4/5 GHz)

Background: Construction

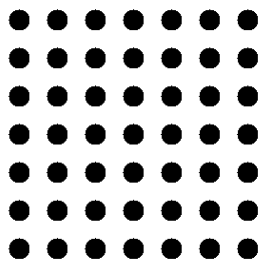


Background: Architecture of the lab

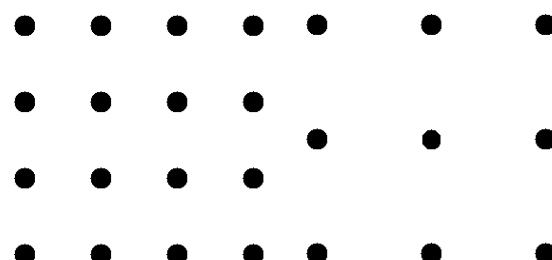


What can the mesh testbed do

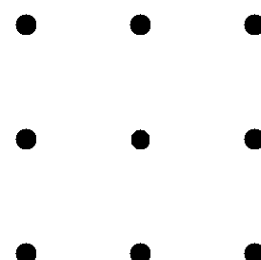
◆ Test various topologies



(a) Full 7x7 grid



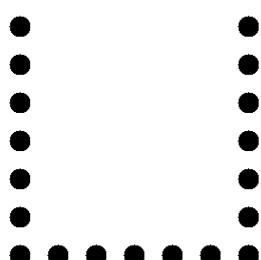
(b) Sparse 4x4 grid



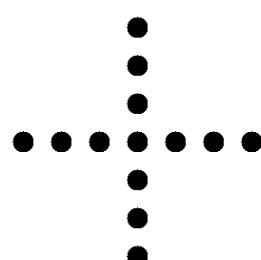
(c) Sparse 3x3 grid



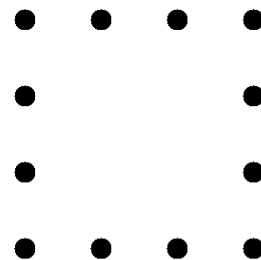
(d) String of pearls



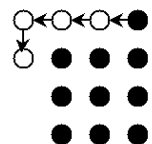
(e) Horse shoe



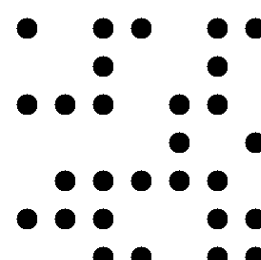
(f) Maltese Cross



(g) Ring



(h) Growing spiral



(i) Random

Accessing the lab (1)

You will need to ask the administrators of the mesh lab to create an account for you. Once you have credentials, you can log into the server as follows:

If you are located at Meraka :

Linux meshlab.dhcp server

ssh **username@meshlab.dhcp.meraka.csir.co.za**

Free-BSD meshy.dhcp server

ssh **username@meshy.dhcp.meraka.csir.co.za**

Accessing the lab (2)

The Linux and free-BSD mesh servers are not visible from the Internet, nor any network outside of Meraka. If you require remote access you will need to ask the Administrators to set-up a reverse tunnel to your server.

If you are remotely located and a tunnel has been configured for you, ask which tunnel port was used and then use:

```
ssh -p <tunnel port> username@localhost
```

e.g. if you were given a tunnel port of 10002

```
ssh -p 10002 username@localhost
```

Booting OS: Background

- ◆ **The nodes are only equipped with 128M of flash memory and can either boot their operating system off the flash memory or using their built in network interface.**
- ◆ **Network boot is the default boot mode of the nodes. The Bios has been set-up so that when a node starts it will automatically try to boot off the network card first and then try to boot off the flash card if the network boot fails.**
 - + All settings in `/etc/ltsp/dhcpd.conf`
 - + Gets IP address using fixed table in DHCP server
 - + Gets bootstrap program using PXE
 - + Bootstrap program loads operating system specified in `dhcpd.conf`

Botting OS: Changing the OS to boot (1)

- ◆ **Command to change operating system**

- + `mk-dhcp-classes [-d <default_class>] <range> <class>`

- ...

- **Available classes**

- Linux-hardcore ... Linux

- Ubuntu-hardy

- Ubuntu-hardy-click

- freebsd-6-stable

- freebsd-7-stable

- **range is in the form: x1y1-x2y2**

- Where x1y1 and x2y2 are grid coordinaes

- ◆ ***To install your own operating system see the lab manual***

Booting OS: Changing the OS to boot (2)

- ◆ **Example 1: Half the grid (rows 1-3) runs fbsd6 FreeBSD and half the grid (rows 4-7) runs ubuntu-hardy linux**
+ **mk-dhcp-classes 11-37 fbsd6 41-77 linux-hardy**
- ◆ **Example 2: The whole grid runs fbsd7 FreeBSD except for node 71 which runs hardcore linux**
+ **mk-dhcp-classes -d fbsd7 71 linux-hardcore**
- ◆ **Booting off the flash disk (disable the dhcp server so that the PXE boot-loader gets no IP addresses leases from the server)**
+ **/etc/init.d/dhcp3-server stop**
– **To boot off the network server again (/etc/init.d/dhcp3-server start)**

Booting OS: Monitoring boot process

- ◆ Once you have changed the operating system you will need to reboot all the nodes
- ◆ A useful tool to monitor the boot process when the operating system is booting is to use the console tool on the serial port.
- ◆ If the operating system is set for serial monitoring (all the default OS's have this installed) it will send it's console output to the serial port
 - + `APPEND console=ttyS0,115200`
- ◆ Start a terminal emulator on serial port 1
 - + `Picocom -b 115200 /dev/ttyS0`
- ◆ Node 77 is connected to the serial port, so you will need to power cycle this node to check the boot process

Controlling the nodes: Power on/off (1)

- ◆ The nodes are controlled by a relay bank which can switch each of the 7 banks on and off in the grid
- ◆ Using the relay is useful when changing operating system or when a node hangs due to some instability
- ◆ To check the power status of each row
 - + meshpower status
 - Power: "On Off On On On On On Off "
- ◆ To switch banks on or off use
 - + meshpower <bank|all> <on|off> [time(s)]
 - all switches all banks on
 - bank is a number from 1 to 8 (8 is not used yet)
 - time (in seconds) is used in combination with all to specify time between each bank switching on
 - It is useful to specify a large interval like 60 seconds between banks switching on to prevent overwhelming the server with too much NTP traffic

Controlling the nodes: Power on/off (2)

- ◆ **example1:**

- + meshpower 2 off
- turns off bank 2.

- ◆ **example2:**

- + meshpower all on 30
- turns on all banks with a 30 second interval between rows switching on

Controlling the nodes: logging into nodes

- ◆ To login to a particular node at a specific xy coordinate use the following command
 - + `ssh root@172.20.1.xy`
- ◆ A short hand script has also been created which allows you to abbreviate this to
 - + `msh <xy>`
- ◆ Example - to log into a node at position 3,3 you would use the command
 - + `msh 33`
- ◆ You will then be logged into the node and can issue any Unix command at the command line which will be executed on that node.

Controlling the nodes: remote control (1)

- ◆ There are often times where you may want to execute the same command on many nodes at once to avoid logging into each node
- ◆ Use the following command to execute a command on a node
 - + `lexec |<all>| or |<range>| or |<x><y>| <command>`
 - all sends a command to all 49 nodes in the grid
 - range is of the form `<x1y1-x2y2>`
 - command contains a shell command you want to execute on the node
- ◆ **Example 1:**
 - + `lexec 24 "reboot"`
 - Reboot node 24
- ◆ **Example 2:**
 - + `lexec 11-22 "iwconfig ath0 channel 6"`
 - Change nodes 11,12,13,14,15,16,17,21,22 to channel 6

Installing Your Own Software on Nodes

- ◆ **First chroot into the host file system:**
 - + Login to the server meshlab.dhcp.meraka.csisr.co.za
 - + chroot into the desired file system:
 - `sudo chroot /export/ubuntu-hardy`
 - + mount the /proc file system
 - `mount -t proc /proc /proc`
- ◆ **To install from an Ubuntu Repository:**
 - `apt-get install packagename`
- ◆ **To install from Source:**
 - + Download source using a command like **wget** or with **cv**s or **git**.
 - + Extract archive and copy to a folder in /usr/src or to /home/user
 - + Compile and install the package (Refer to package documentation)

Testing Your Software Installation

- ◆ **Exit out of chroot**
 - `exit`
- ◆ **Login to any node that booted this operating system to test the compiled code.**
 - `msh 47`

The wireless interface: control

- ◆ **There are many parameters that can be changed on a wireless interface using command line tools**
 - + Mode: 802.11b, 802.11g, 802.11a
 - + Rate: 1,2,5,6,11 54 Mbps
 - + Channel:
 - 802.11b/g: 1-13 (2412-2472MHz) most countries, 1-14 Japan
 - 802.11a: 36-64 (5180-5320MHz), 100-165 (5500-5825MHz),
- ◆ **The core tools are:**
 - + iwconfig
 - Configure a wireless network interface
 - + iwpriv
 - Configure optional (private) parameters of a wireless network interface
 - + iwlist
 - Get detailed wireless information from a wireless interface



Configuring The Wireless Interface

- ◆ **Set Ad-Hoc Mode**
 - + `iwconfig ath0 mode Ad-Hoc`
 - Because there is no AP in the mesh.
- ◆ **Select 802.11 a/b or g Modulation**
 - + `iwpriv ath0 mode 11b`
 - To lock to Interface to 802.11b only
 - + `iwpriv ath0 get_mode`
 - To query the currently selected mode
- ◆ **Channel / Frequency**
 - + `iwlist ath0 channel`
 - View available channels
 - + `iwconfig ath0 channel 3`
 - Set channel 3

Configuring The Wireless Interface

◆ Transmit Rate

+ iwlist ath0 rate

– To view available transmit rates

+ iwconfig ath0 rate 11M

– Configure 11Mb/s transmit rate

◆ Broadcast/Multicast Transmit Rate

+ iwpriv ath0 mcast_rate 11000

– Set 11Mb/s (Rate in Kb/s)

Configuring The Wireless Interface

◆ Transmit power

+ iwlist ath0 txpower

– View available Tx power settings

+ iwconfig ath0 txpower 15

– set the transmit power in dBm. If W is the power in Watt, the power in dBm is $P = 30 + 10 \cdot \log(W)$.

– The next two slides correlate the relationship between power and physical transmission range!

◆ To disable a Wireless Interface:

+ iwconfig ath0 txpower 0

– Disables radio transmissions.

+ ifconfig ath0 down

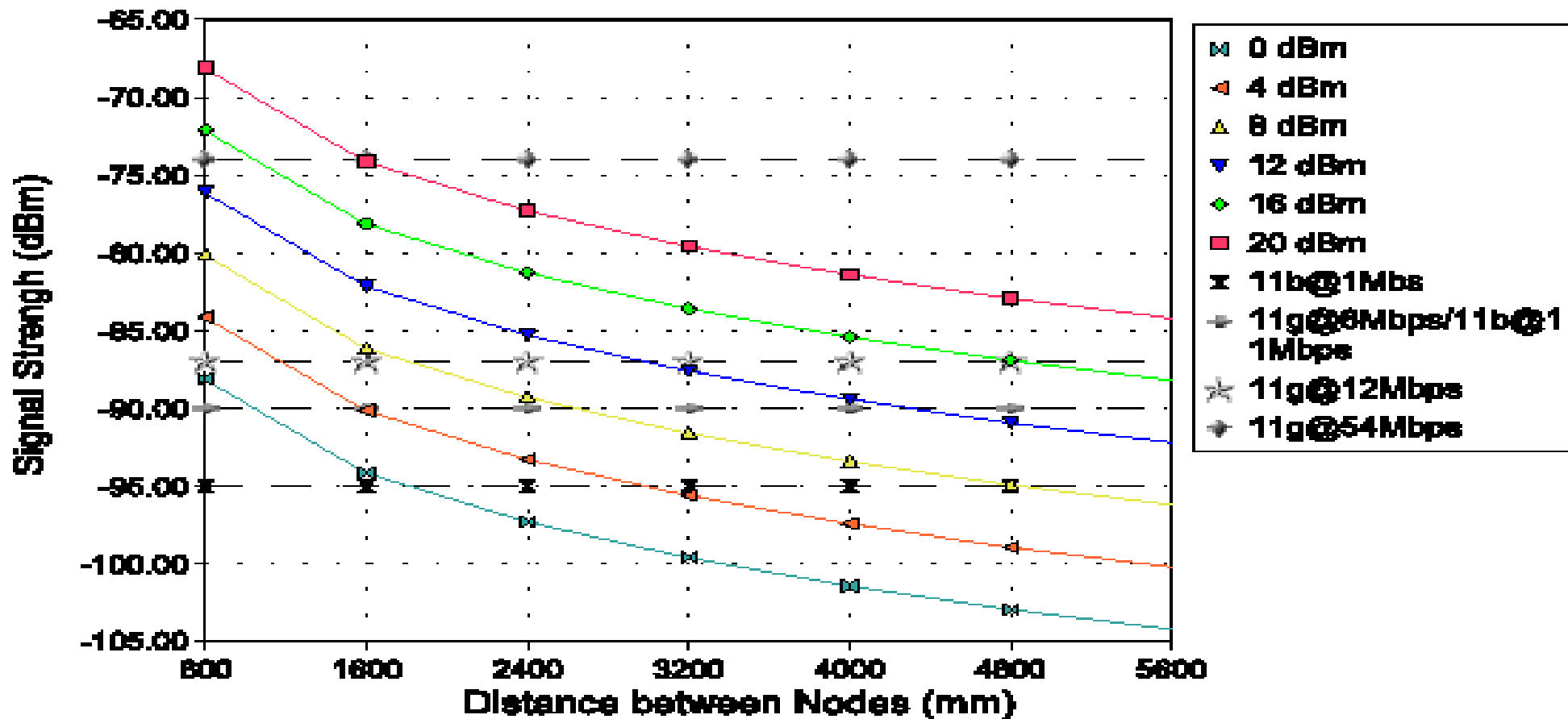
– causes the driver for this interface to be shut down.

Transmit Power vs Range

Power level (dBm)	radio mode	data rate (Mbps)	Receive sensitivity (dBm)	Range (m)
0	802.11g	54	-74.00	0.15
0	802.11g	36	-79.00	0.27
0	802.11g	24	-82.00	0.39
0	802.11g	12	-87.00	0.69
0	802.11g	6	-90.00	0.97
0	802.11b	11	-90.00	0.97
0	802.11b	5.5	-92.00	1.22
0	802.11b	2	-94.00	1.54
0	802.11b	1	-95.00	1.73
20	802.11g	36	-79.00	2.74
20	802.11g	6	-90.00	9.71
20	802.11b	1	-95.00	17.26

Possible radio range of a node when configured with various radio parameters using 30dB of attenuation on each radio.

Received Signal Strength vs Distance



Received signal strength vs. distance between nodes in the grid spaced 800 mm apart. The horizontal lines show the receive sensitivity of the Atheros 5213 wireless network card. If the received signal strength curve is above this line, there will be connectivity between the nodes.

Configuring The Wireless Interface

- ◆ **MAC retransmissions behaviour:**
 - + `iwconfig ath0 retry 16`
 - Retry up to 16 times
 - + `iwconfig ath0 retry lifetime 300m`
 - Retry for up to 300ms

- ◆ **RTS/CTS handshaking:**
 - + `iwconfig ath0 rts 512`
 - Use RTS/CTS for packets bigger than 512 octets
 - + `iwconfig ath0 rts off`
 - Disable RTS/CTS

Configuring The Wireless Interface

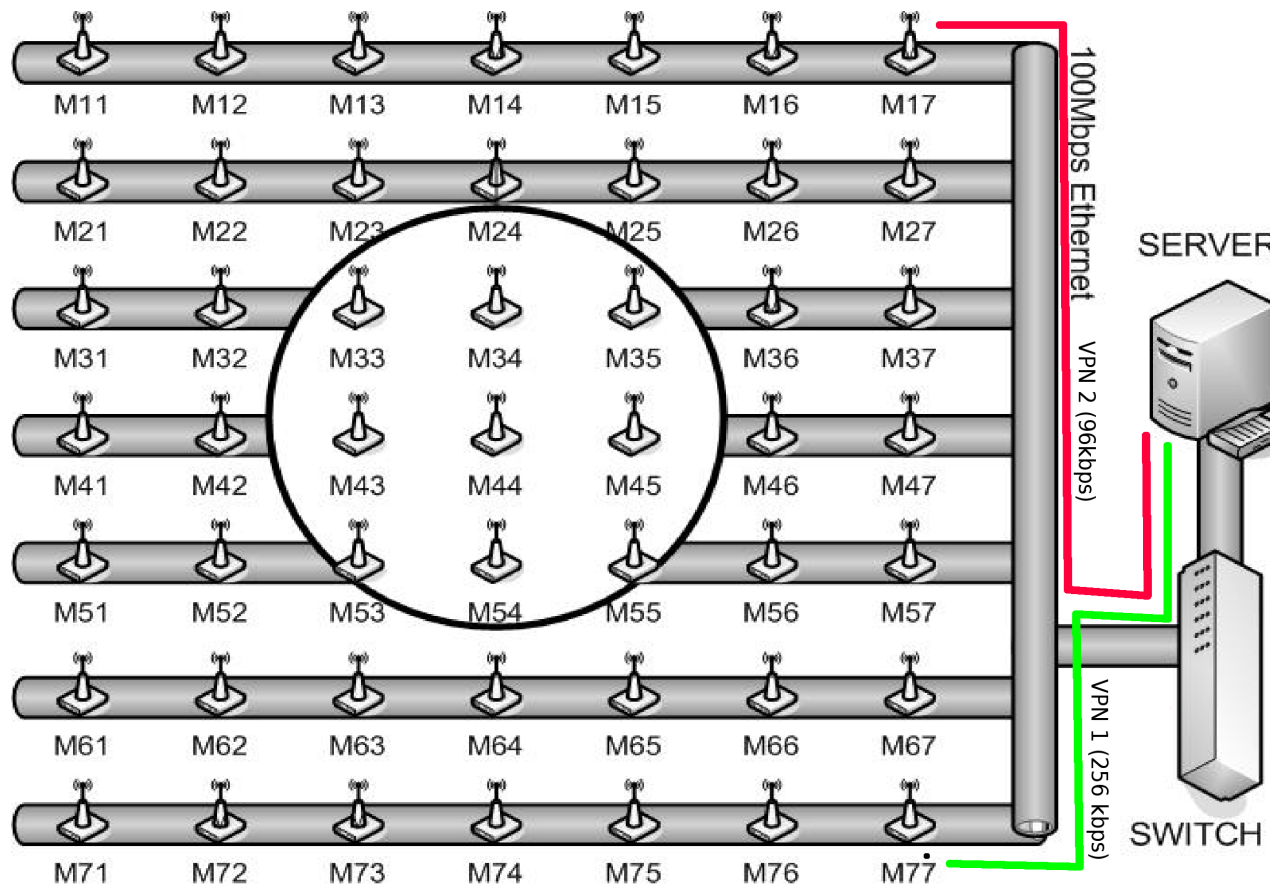
- ◆ **Fragmentation (split large IP packets into smaller fragments):**
 - + iwconfig ath0 frag 512
 - Fragment packets larger than 512 octets
 - + iwconfig ath0 frag off
 - Disable fragmentation
- ◆ **Force card to apply all pending changes immediately:**
 - + iwconfig ath0 commit

Traffic control and tunneling

- ◆ **The traffic control tool (TC)**
- ◆ **Useful when you need to emulate a network link with a specific delay, throughput or queuing discipline.**
- ◆ **Traditionally it is used to shape and schedule traffic to guarantee some degree of fairness when multiple users are using the network**
- ◆ **In order to use the traffic control tool to emulate links you will also need to know how to build tunnels with the “ip tunnel” command**

Traffic control and tunneling

- ◆ Lets say we wanted to emulate two gateways in the mesh
 - + 1 gateway at node 17 with a capacity of 96kbps
 - + 1 gateway at node 77 with a capacity of 256kbps



Traffic control and tunneling (1)

- ◆ There are commands that need to be executed on the server and on node 17 to set up this emulated tunnel.
- ◆ On the server we need to create a tunnel device
 - Need to be root
 - + `ip tunnel add vsat17 mode gre remote 172.20.1.17`
 - Builds tunnel between server and node 17 – creates a device
 - + `ip addr add 10.17.1.1 dev vsat17`
 - Adds an IP address to tunnel device
 - + `ifconfig vsat17 up`
 - Brings up the tunnel interface
 - + `ip route add 10.17.2.0/24 dev vsat17`
 - Ensures that all traffic to tunnel subnet is routed through tunnel

Traffic control and tunneling (2)

- ◆ **On node 17 we also need to build a tunnel device**
 - + `Ip tunnel add vsat1 mode gre remote 172.20.1.4`
 - Build tunnel from node 17 to server – create device
 - + `Ip addr add 10.17.2.1 dev vsat1`
 - Add IP address to tunnel device
 - + `Ifconfig vsat1 up`
 - Bring up tunnel interface
 - + `Ip route add 10.17.1.0/24 dev vsat1`
 - Ensures that all traffic to tunnel subnet is routed through tunnel
 - + `Ip route add default via 172.17.2.1`
 - Make this tunnel a default gateway on node
 - + `Iptables --table nat -a postROUTING -o vsat1 -j MASQUERADE`
 - Do nat on this interface to get to real internet (see notes later on iptables)

Traffic control and tunneling (3)

- ◆ **Once the tunnels are set up we can shape the traffic that goes through these tunnels without effecting the underlying link that the tunnel is built on (do this on node17)**
 - + **Tc qdisc add dev vsat1 root handle 1:0 htb default 30**
 - Add a queuing discipline to tunnel device (htb = hierachical token bucket)
 - + **Tc class add dev vsat1 parent 1: classid 1:2 96kbit**
 - Add a child class to parent 1 for downlink of 96kbps
 - + **Tc class add dev vsat1 parent 1: classid 1:3 96kbit**
 - Add a child class to parent 1 for uplink of 96kbps
 - + **Tc qdisc add dev vsat1 parent 1:2 handle 2: sfq perturb 10**
 - Add a queing discipline to downlink child with stochastic fair queuing
 - + **Tc qdisc add dev vsat1 parent 1:3 handle 3: sfq perturb 10**
 - Add a queing discipline to uplink child with stochastic fair queuing

Traffic control and tunneling (4)

- + Tc filter add dev vsat1 protocol ip parent 1:0 prio 1 u32 match ip dst 10.17.2.1/32 flowid 1:2
 - Add filter to mark packets coming into tunnel as downlink flow
- + Tc filter add dev vsat1 protocol ip parent 1:0 prio 1 u32 match ip src 10.17.2.1/32 flowid 1:3
 - Add filter to mark packets leaving tunnel as uplink flow

iptables Firewall

- ◆ **iptables is a user space application program that allows a system administrator to configure the tables provided by the Linux kernel firewall.**
- ◆ **The iptables framework enables:**
 - + packet filtering
 - + network address [and port] translation (NA[P]T)
 - + packet mangling.
- ◆ **We'll demonstrate iptables for:**
 - + Network Address Translation (NAT)
 - Enable mesh nodes to share the server's Internet connection.
 - + Topology Control
 - Iptables allows us to dynamically break the network links between pairs of nodes, thus controlling network topology.

Iptables: Simple Steps to NAT

- ◆ **Enable kernel IP forwarding:**
 - + `echo 1 > /proc/sys/net/ipv4/ip_forward`
- ◆ **Configure iptables to forward the packets from your internal network, on /dev/eth1, to your external network on /dev/eth0:**
 - + `/sbin/iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE`
 - + `/sbin/iptables -A FORWARD -i eth0 -o eth1 -m state --state RELATED,ESTABLISHED -j ACCEPT`
 - + `/sbin/iptables -A FORWARD -i eth1 -o eth0 -j ACCEPT`
- ◆ **Test NAT by pinging an external address from one of the mesh nodes.**
- ◆ **Save these commands to a bash script, as iptables changes will be lost after the next reboot.**

Iptables: Topology Control

- ◆ **To break connectivity between two nodes (to be executed on a node, not the server!):**
 - + iptables -D INPUT -s 172.20.1.12 -j DROP
 - Drop all incoming packets received from node 12
 - + iptables -A OUTPUT -d 172.20.1.33 -j DROP
 - Drop all outgoing packets destined for node 33
 - + iptables -A FORWARD -d 172.20.1.34 -j DROP
 - Drop any packets from other nodes being forwarded to node 34
- ◆ **To remove an iptables rule, simply replace -A with -D:**
 - + iptables -A INPUT -s 172.20.1.12 -j DROP

Traffic generation/measurement (1)

- ◆ There are many traffic generation and measurement tools
 - + Ping is the simplest and is still the most commonly used tool to check for network problems
 - Ping -s Set packets size
 - Ping -p ff fill data with pattern of ff's
 - Ping -r display route taken by packet
- ◆ Good survey of performance tools by Jan Barton – CESNET technical report 18/2003

Support	DBS	IPerf	NetPerf	NetPIPE	NetSpec	RUDE & CRUDE	Treno	TTCP6
User defined frame format	✗	✓	✓	✗	✗	✗	✗	✗
Verifying received frames	✓	✓	✗	✗	✗	✗	✓	✗
Bidirectional traffic	✓	✓	✗	✗	✓	✗	✗	✗
Setting inter-frame time gap	✓	✗	✓	✗	✓	✓	✗	✗

✓ Full Support

✓ Partial Support

✗ No Support

Traffic generation/measurement (2)

◆ Chose iperf because of its ability to

- Measure delay/jitter/packet loss
- Determine MTU size, TCP window size
- Run tests by time or amount transferred
- UDP streams with specified bandwidth (also multicast mode)
- bidirectional testing mode
- use a representative stream (from file or stdin) to measure bandwidth (could use to check how compression affects available bandwidth)

◆ Examples (server side)

```
+ node2> iperf -s -u -l 32k -w 128k -i 1
```

- s = run IPerf in server mode
- u = use UDP instead TCP
- l 32k = buffer length
- w 128k = largest receivable datagram size
- i 1 = interval time in seconds between periodic reports

Traffic generation/measurement (3)

◆ Examples (client side)

```
+ node1> iperf -c node2 -b 10m -l 32k -w 128k
```

- c = run IPerf in client mode
- node2 = server address
- b 10m = UDP bandwidth to send at, in bits/sec – 10Mbit/sec
- l 32k = buffer length
- w 128k = largest receivable datagram size

Mesh protocols: Which one?

◆ Pro-active (Table-driven)

- + AWDS (Ad-hoc Wireless Distribution Service)
- + Babel (AODV with ETX)
- + CGSR (Clusterhead Gateway Switch Routing protocol)
- + DFR ("Direction" Forward Routing)
- + DBF (Distributed Bellman-Ford Routing Protocol)
- + **DSDV (Highly Dynamic Destination-Sequenced Distance Vector routing protocol)**
- + Guesswork (Robust Routing in an Uncertain World)
- + HSR (Hierarchical State Routing protocol)
- + IARP (Intrazone Routing Protocol/pro-active part of the ZRP)
- + LCA (Linked Cluster Architecture)
- + **MMRP (Mobile Mesh Routing Protocol)**
- + **OLSR (Optimized Link State Routing Protocol)**
- + STAR (Source Tree Adaptive routing protocol)
- + TBRPF (Topology Dissemination based on Reverse-Path Forwarding routing protocol)
- + WRP (Wireless Routing Protocol)

◆ Reactive (On-demand)

- + **ANTNET (Ant-based Routing Algorithm for Mobile Ad-Hoc Networks)**
- + ACOR (Admission Control enabled On demand Routing)
- + Ariadne (A Secure On-Demand Routing Protocol for Ad Hoc Networks)
- + Associativity-Based Routing
- + **AODV (Ad-hoc On-demand Distance Vector)**
- + BSR (Backup Source Routing)
- + CHAMP (CachIng And MultiPath routing)
- + **DSR (Dynamic Source Routing)**
- + DSRFLOW (Flow State in Dynamic Source Routing)
- + DNVR (Dynamic Nix-Vector Routing)
- + **DYMO (DYnamic Manet On-demand Routing)**
- + MAODDP (Mobile Ad-hoc On-Demand Data Delivery Protocol)
- + FOR (Flow Oriented Routing)
- + IERP (Interzone Routing Protocol/reactive part of the ZRP)
- + LBR (Link life Based routing)

◆ Reactive (On-demand) .. continued

- + LMR (Lightweight Mobile Routing protocol)
- + LQSR (Link Quality Source Routing)
- + LUNAR (Lightweight Underlay Network Ad hoc Routing)
- + MOR (Multipath On-demand Routing Protocol)
- + MPRDV (Multipoint Relay Distance Vector protocol)
- + QuaSAR (QoS aware source initiated ad-hoc routing)
- + RDMAR (Relative-Distance Micro-discovery Ad hoc Routing protocol)
- + **SrcRR (DSR and ETX based, optimized for performance)**
- + SSR (Signal Stability Routing protocol)
- + TORA (Temporally-Ordered Routing Algorithm routing protocol)
- + PLBR (Preferred link based routing)

◆ Hierarchical

- + CBRP (Cluster Based Routing Protocol)
- + CEDAR (Core Extraction Distributed Ad hoc Routing)
- + DART (Dynamic Address Routing)
- + DDR (Distributed Dynamic Routing Algorithm)
- + FSR (Fisheye State Routing protocol)
- + GSR (Global State Routing protocol)
- + HARP (Hybrid Ad Hoc Routing Protocol)
- + HSR (Host Specific Routing protocol)
- + HSR (Hierarchical State Routing)
- + LANMAR (Landmark Routing Protocol for Large Scale Networks)
- + OORP (OrderOne Routing Protocol)

Mesh protocols: Which one?

◆ Geographical

- + ALARM (Adaptive Location Aided Routing Protocol - Mines)
- + BGR (Blind Geographic Routing)
- + DREAM (Distance Routing Effect Algorithm for Mobility)
- + GLS(Grid) (Geographic Location Service)
- + LAR (Location-Aided Routing protocol)
- + GPSAL (GPS Ant-Like Routing Algorithm)
- + ZHLS (Zone-Based Hierarchical Link State Routing)
- + GPSR (Greedy Perimeter Stateless Routing)
- + GFG (Greedy Face Greedy)
- + SiFT (Simple Forwarding over Trajectory)

◆ Power aware

- + ISAIH (Infra-Structure Aodv for Infrastructured Ad Hoc networks)
- + PARO (Power-Aware Routing Optimization Protocol)
- + EADSR (Energy Aware Dynamic Source Routing Protocol)
- + PAMAS (PAMAS-Power Aware Multi Access Protocol with Signaling Ad Hoc Networks)

◆ Geographical Multicast (Geocasting)

- + LBM (Location Based Multicast)
- + GeoGRID (Geographical GRID (see GLS))
- + GeoTORA (Geographical TORA (see TORA))
- + MRGR (Mesh-Based Geocast Routing)
- + MOBICAST (Mobile Just-in-time Multicasting)

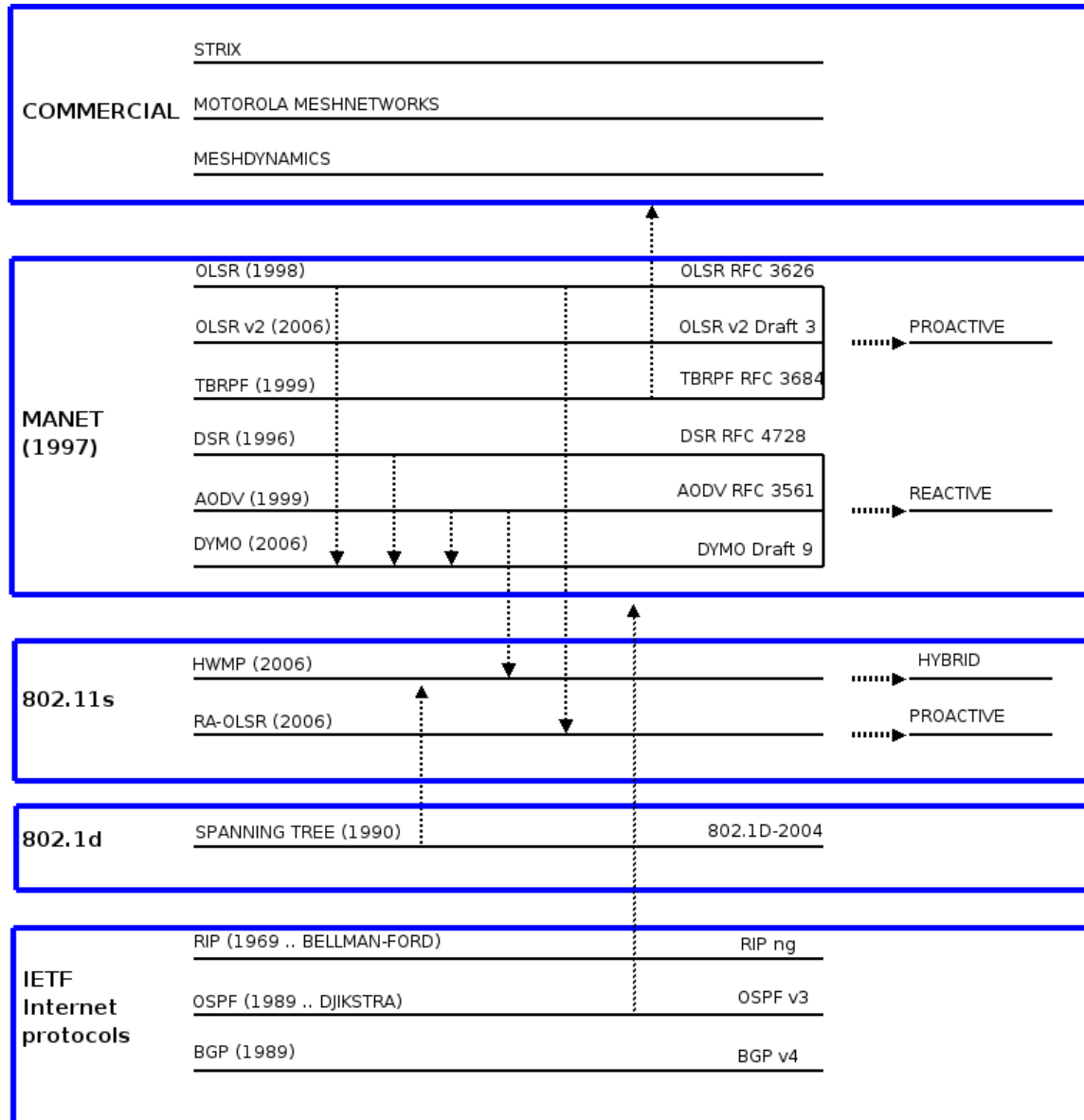
◆ Multicast

- + ABAM (On-Demand Associativity-Based Multicast)
- + ADMR (Adaptive Demand-Driven Multicast Routing)
- + AMRIS (Ad hoc Multicast Routing protocol utilizing Increasing id-numberS)
- + AMRoute (Adhoc Multicast Routing Protocol)
- + AQM (Ad Hoc QoS Multicast)
- + CAMP (Core-Assisted Mesh Protocol)
- + CBM (Content Based Multicast)
- + DCOMP (Dynamic Core Based Multicast Routing Protocol)
- + DDM (Differential Destination Multicast)
- + DSR-MB (Simple Protocol for Multicast and Broadcast using DSR)
- + FGMP (Forwarding Group Multicast Protocol)
- + LAM (Lightweight Adaptive Multicast)
- + MAODV (Multicast Ad-hoc On-Demand Distance Vector routing)
- + MCEDAR (Multicast Core-Extraction Distributed Ad hoc Routing)
- + MZR (Multicast Zone Routing)
- + ODMRP (On-Demand Multicast Routing Protocol)
- + SMF (Simplified Multicast Forwarding)
- + SPBM (Scalable Position-Based Multicast)
- + SRMP (Source Routing-based Multicast Protocol)

◆ Other

- + EraMobile (Epidemic-based Reliable and Adaptive Multicast)
- + FQMM (Flexible QoS Model for MANET)
- + SMP (Skewed Map Forwarding)
- + INSIGNIA (In-band signaling support for QoS in Mobile Ad hoc Networks)
- + IMEP (Internet Manet Encapsulation Protocol)
- + ANMP (Adhoc Network Management Protocol)
- + Terminode Routing
- + B.A.T.M.A.N. (Better approach to mobile adhoc networking)
- + ODLW (On-Demand Link-Weight routing protocol for Ad Hoc Networks)

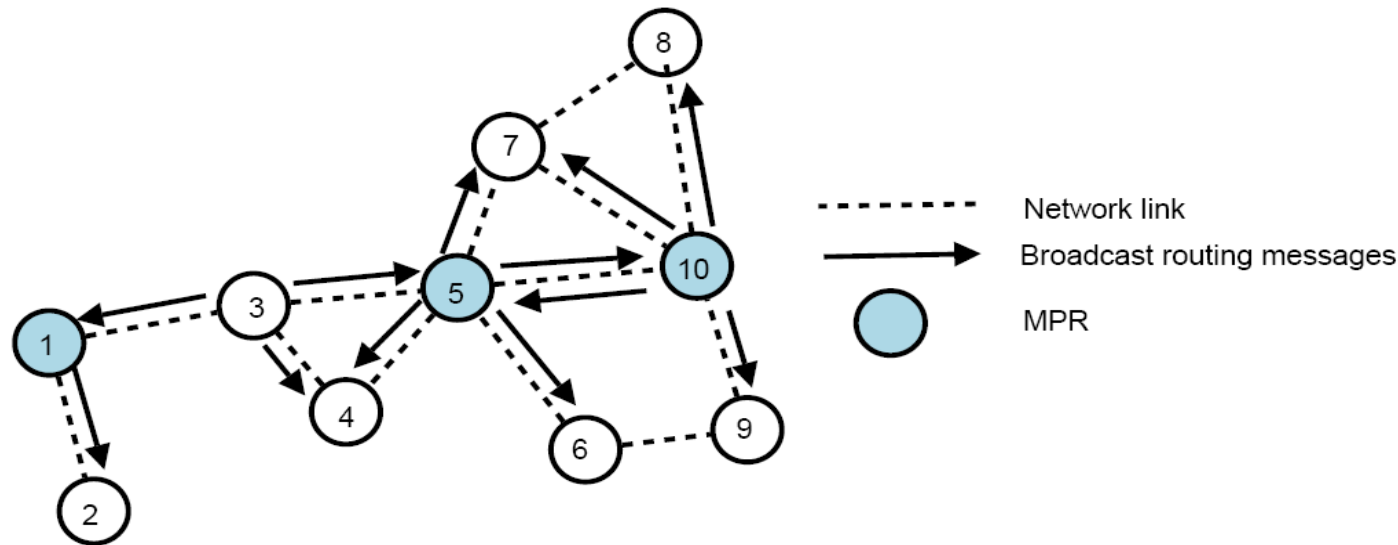
Mesh protocols: Which one?



Mesh protocols available on the grid - OLSR

◆ OLSR (Optimized Link State Routing)

+ Currently running May 2009 release 0.5.6-r4 from www.olsr.org



+ All configuration done in `/etc/olsrd.conf`

+ Can change configuration file with `olsrd -f <filename>`

+ Can set debug level with `olsrd -d <0...9>`

Mesh protocols available on the grid - OLSR

◆ Configuration file (olsrd.conf)

+ Ipversion <4/6>

+ LinkqualityLevel <0...2>

- 0 = use RFC OLSR
- 1 = use ETX for MPR selection
- 2 = use ETX for all routing selection

+ LinkQualityWinSize <number of messages in window>

- Defaults to 10
- 100 good choice for robust network

+ TCRedundancy <0...2>

- How much neighbour info is sent in TC (Topology control) messages

+ MPRCoverage <1...>

- Specifies how many MPRs a node should select to reach every 2 hop neighbour

Mesh protocols available on the grid - olsr

+ Interface “<wireless interface names>”

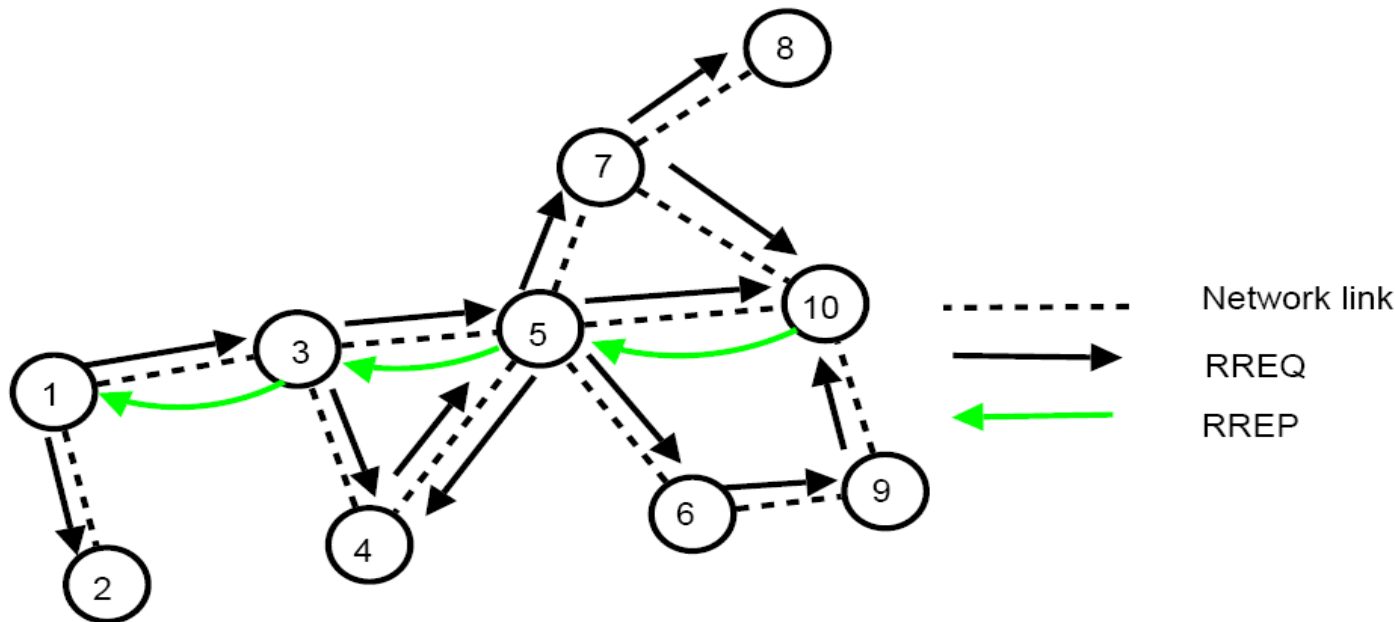
- Sets the interface to send olsr messages on, can be more than one for a multi-radio node

+ Timer settings

- HelloInterval <seconds> ... recommended value = 5
- HelloValidityTime <seconds> ... recommended value = 300
- TCInterval <seconds> ... recommended value = 2
- TCValidityTime <seconds> ... recommended value = 300

Mesh protocols available on the grid - aodv

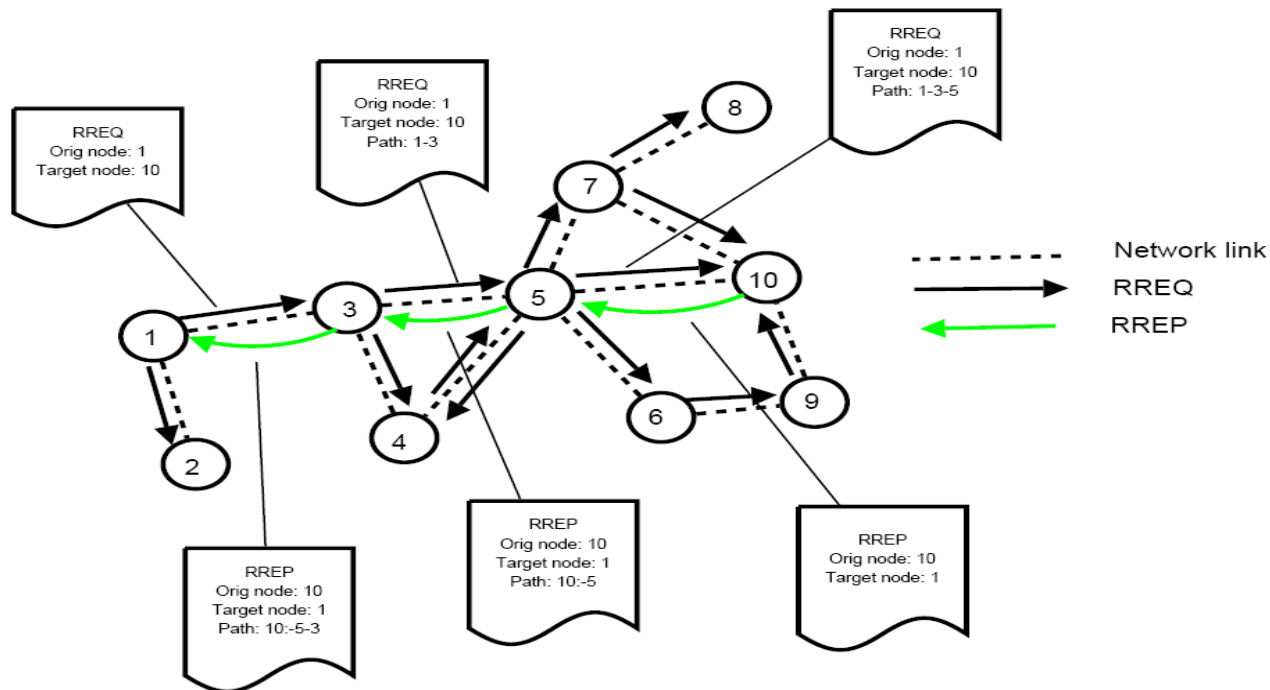
- ◆ **AODV (Ad hoc on demand distance Vector) algorithm**
 - + Currently running aodv-uu July 2007 release 0.9.5 from Urpsala University <http://core.it.uu.se/core>



- + Configuration done from the command line

Mesh protocols available on the grid - DYMO

- ◆ **DYMO (Dynamic MANET On-demand routing) algorithm**
 - + Currently running dymod 0.3 release by Francisco Ros <http://masimum.dif.um.es/?Software:DYMOUM>

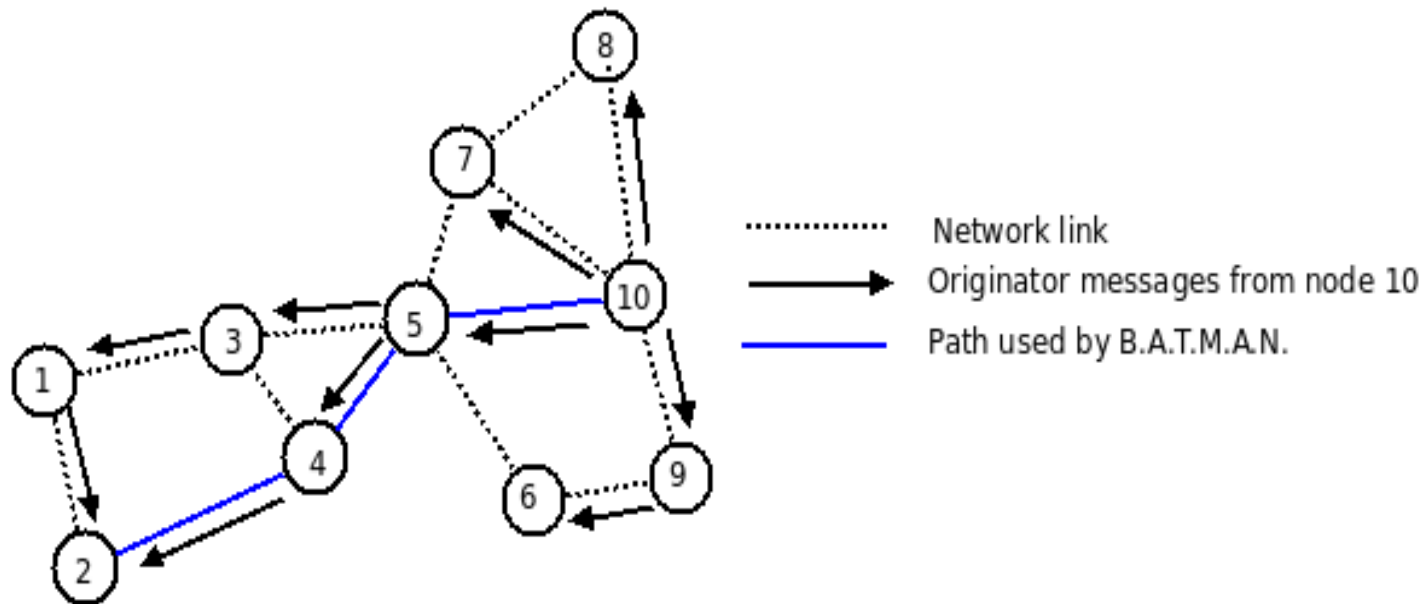


+ Configuration done from the command line

Mesh protocols available on the grid - DYMO

- ◆ **B.A.T.M.A.N. (Better Approach to mobile ad-hoc networking)**

- + Currently running February 2009 batman experimental 0.3-alpha rv1197 release by axel <http://www.open-mesh.net/wiki>



Pulling it all together - scripts

- ◆ **In a typical lab experiment there are five steps**
 - + Setup → Send commands to nodes → log data → Process data → Plot data
- ◆ **An experimental script or program can be written in any programming language or script language as long as you can send system commands.**
- ◆ **A scripting language such as Python or Perl is a good choice because they can act as a batch language and have good text parsing capabilities.**
- ◆ **Cannot teach a scripting language in the context of this workshop. Python will be shown as an example because of its power and clear indentation based syntax**

Pulling it all together - Python script (1)

- ◆ **Python script to start olsr on all nodes and measure throughput between node 11 and 77 and node 17 and 71**

```
#!/usr/local/bin/python
import os,sys,time

# Define traffic sources
# node_src_xy, node_sink_xy, measurement_period
traf_source = [
    ["11","172.20.1.77",10],
    ["17","172.20.1.71",10]
]

cmd = "lexec all olsrd -d 0"
os.system(cmd)
time.sleep (30)

for block in traf_source:
    cmd = "lexec " + block[0] + " 'iperf -t " + block[4]
    cmd += " -c " + block[1] + " -fk > /root/tp" + block[0] + ".out &"
    os.system(cmd)
```


Pulling it all together - Python script (1)

◆ Python script to process output from iperf file

```
#!/usr/local/bin/python
import os,sys,time,commands,re

f1 = open("average_tp.log","w")
# pattern for packet loss
ptp = re.compile(".* ([\d.]* Kbits)")

cmd = "find tp*.out"
res = commands.getoutput(cmd)
filelist = res.splitlines()

for thefile in filelist:
    xy = thefile[len(thefile)-6:len(thefile)-4]
    f1.write(str(xy) + " ")
    f=open(thefile, 'r')
    for line in f:
        if ptp.match(line):
            tp = m.group(1)
            f1.write(tp + "\n")
    f.close()

f1.close()
```

Plotting with gnuplot

- ◆ **gnuplot is a command-driven interactive function and data plotting program which enables you to make attractive looking graphs.**
- ◆ **Can operate in interactive mode via a command-line .**
 - + Useful for developing, testing and fiddling your graphs!
- ◆ **Alternately, gnuplot can read commands from a file and produce graphs in batch mode.**
 - + Useful if you are running a series of experiments and need to graph the results after each run, or when you need to modify some aspect long after the graph was originally generated.
- ◆ **help is available by typing help <command>**

The plot Command











- ◆ The ``plot`` command is used to generate 2D plots:
 - + `plot "data.csv"`
- ◆ You can specify which columns in the data file are used:
 - + `plot "data.csv" using 1:2`
 - Plots field 1 (x-axis) vs field 2 (y-axis)
- ◆ Use multiple `'plot'` commands to plot multiple curves on a single graph.
- ◆ `gnuplot` allows a great deal of control over the appearance of the graph...

Plotting: Legends and Axes

- ◆ **Enable a key (or legend) describing plots on a graph:**
 - + `set key`
- ◆ **Allow grid lines to be drawn on the plot:**
 - + `set grid`
- ◆ **Label the x-axis:**
 - + `set xlabel "Distance between nodes (mm)"`
- ◆ **Label the y-axis (using a larger font):**
 - + `set ylabel "Average Throughput (kbps)" font "Arial,16"`
- ◆ **Fine control of the major (labelled) tics on the x axis is possible with the ``set xtics`` command. The tics may be turned off with the ``unset xtics`` command, and may be turned on (default state) with ``set xtics``:**
 - + `set xtics font "Arial,12"`
`unset ytics`

Plotting: Changing Colour

- ◆ By default, gnuplot will apply a different colour to each curve.
- ◆ Colours can be explicitly set using hexadecimal RGB values:
 - + plot "data.csv" with lines linecolor xff0000
 - Plot this curve in Red lines
 - + plot "data.csv" with points pointcolor x00FF00
 - Plot this curve using Green points

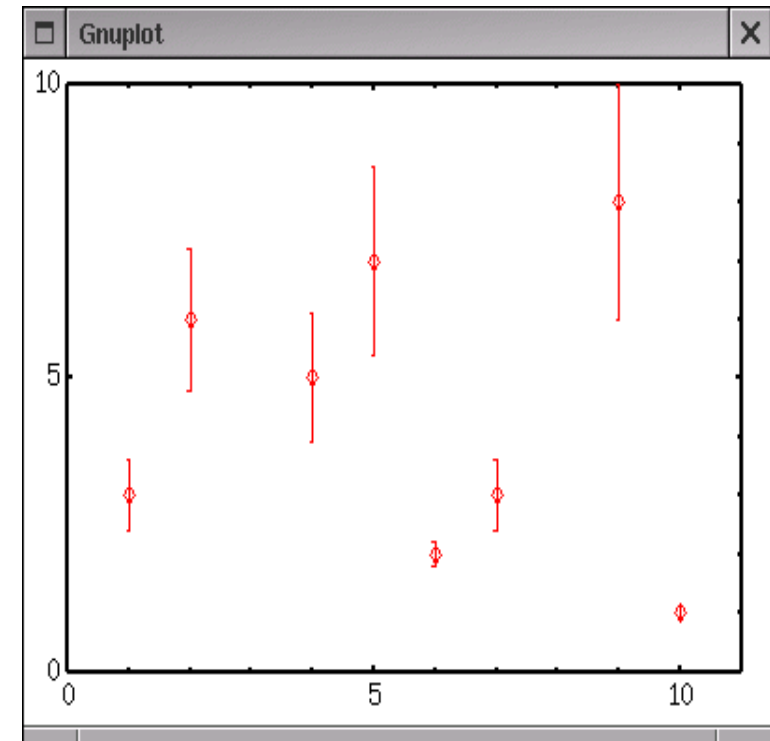
	green	x00ff00		black	x000000
	turquoise	x00ffff			x404040
	blue	x0000ff		grey	x808080
	magenta	xff00ff			xc0c0c0
	red	xff0000		white	xffffffff

Plotting: Style and Thickness

- ◆ **By default, each function and data file will use a different line type and point type, up to the maximum number of available types.**
- ◆ **Of course, they can be explicitly specified:**
 - + plot "data.csv" with lines linewidth 2
 - Use double-thickness lines
 - + plot "data.csv" with lines linetype 2
 - Specify line type as an integer.
 - + plot "data.csv" with points pointsize 3
 - Use triple-size points
 - + plot "data.csv" with points pointtype 3
 - Specify point-type as an integer.

Plotting: Error Bars

- ◆ When input data contains uncertainties for the measured (dependent) quantity, we can create y error bars or error lines.
- ◆ Your data will require one or more extra column for error bars or lines.
 - + plot "data.csv" using 1:6:7 with errorbars
- ◆ If the data file has 3 columns, the third column is used as the Y-error, Y plus/minus dY.
- ◆ If there are 4 columns, the third and fourth columns are used, Y plus dY1 minus dY2.

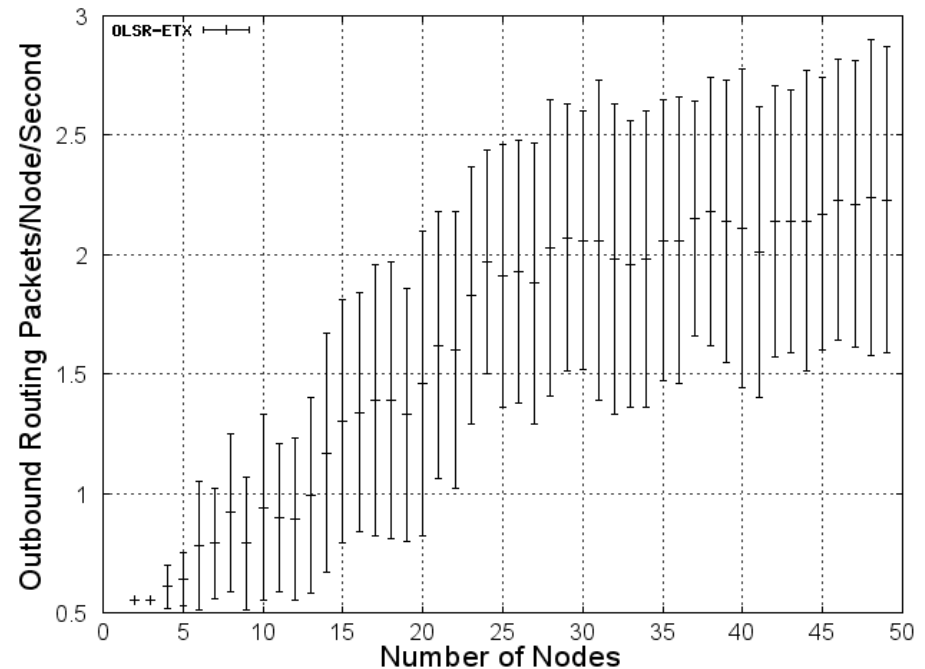
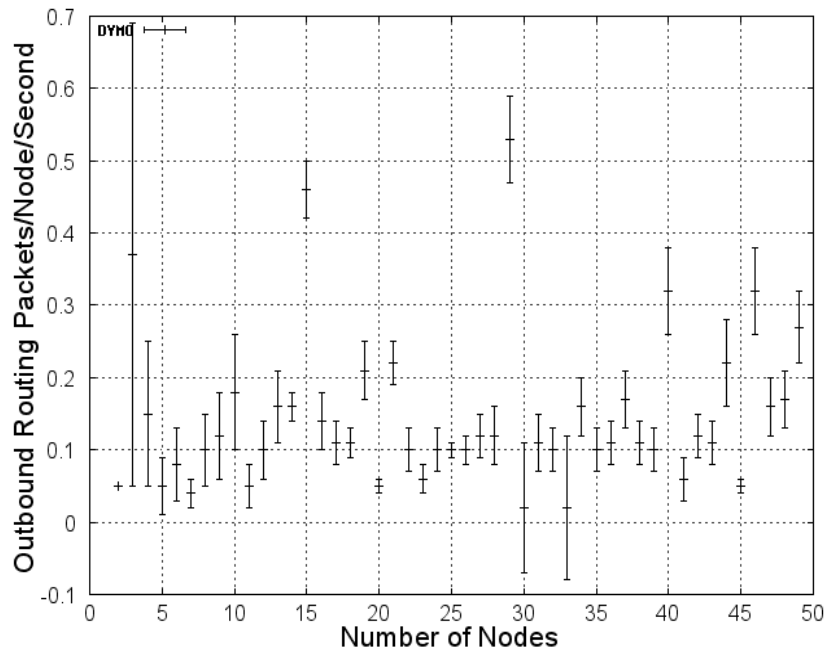


Plotting: Output

- ◆ **Set the terminal output format:**
 - + set terminal postscript
 - Generate a .ps PostScript file
 - + set terminal png enhanced
 - Output Portable Network Graphics (.png)
 - `enhanced` enables enhanced text mode features (subscripts, superscripts and mixed fonts).
- ◆ **Specify an output filename:**
 - + set output "route_traffic_dymo_out.png"

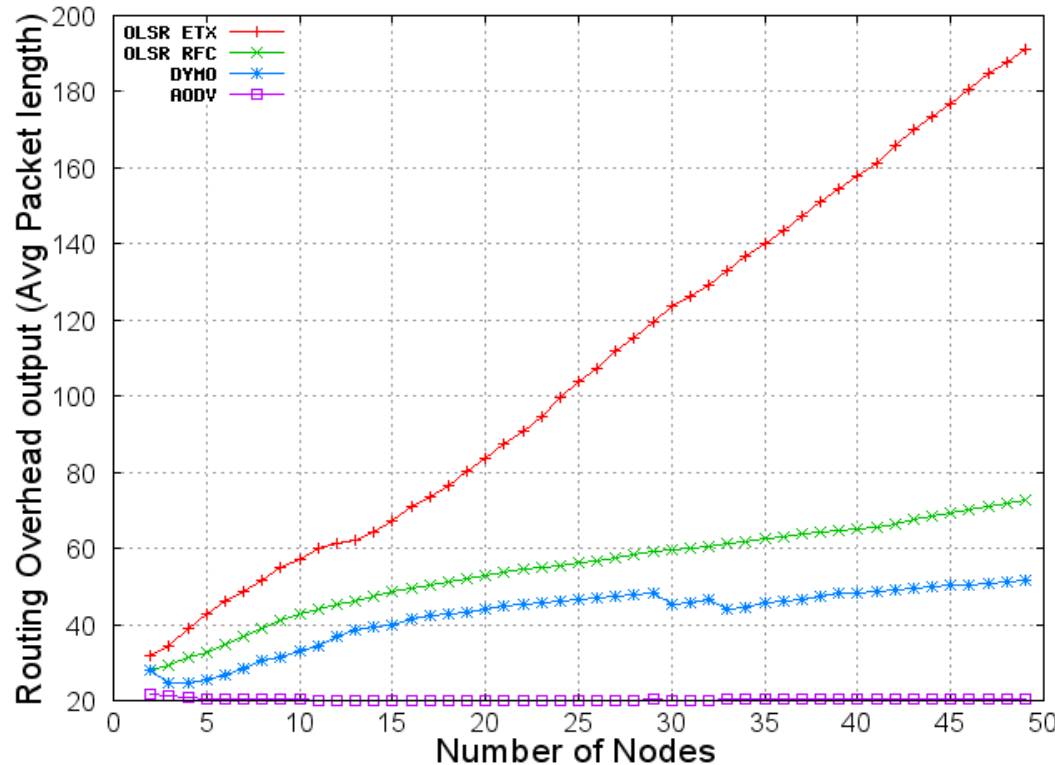
Example graphs - gnuplot (1)

- ◆ **Outbound traffic vs number of nodes for DYMO and OLSR-ETX with error bars.**



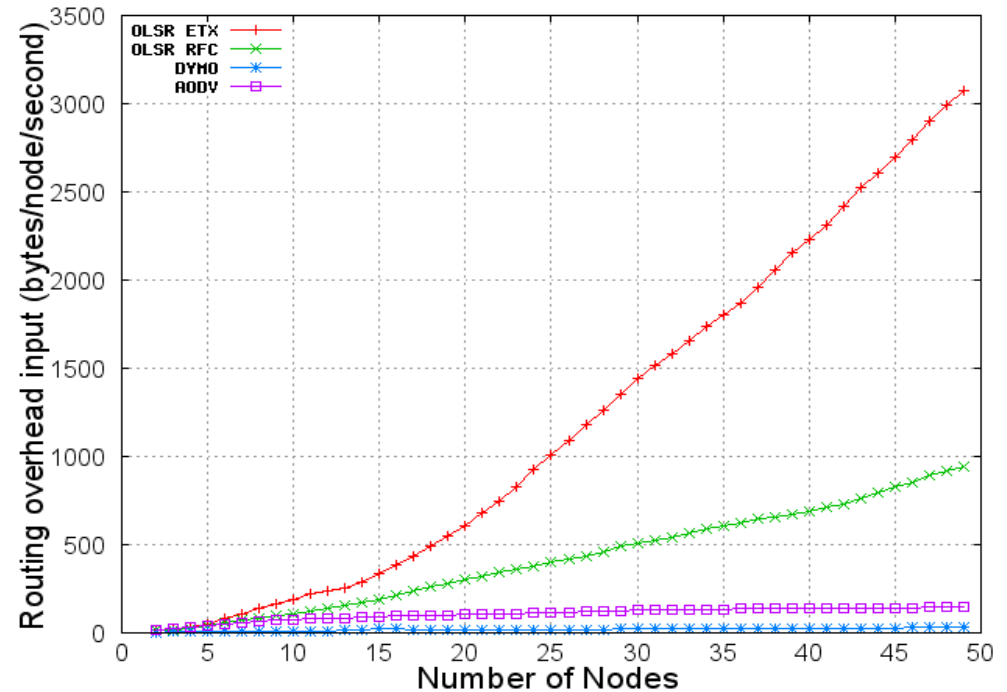
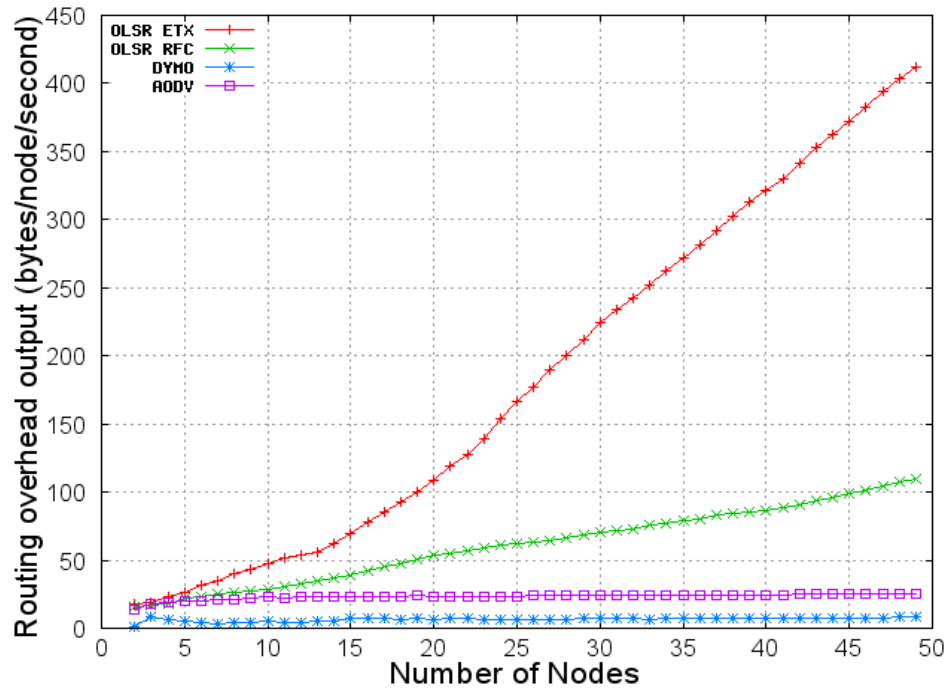
Example graphs - gnuplot (2)

- ◆ Routing packets don't tell the whole story, the length of these packets is also crucial to understanding scalability

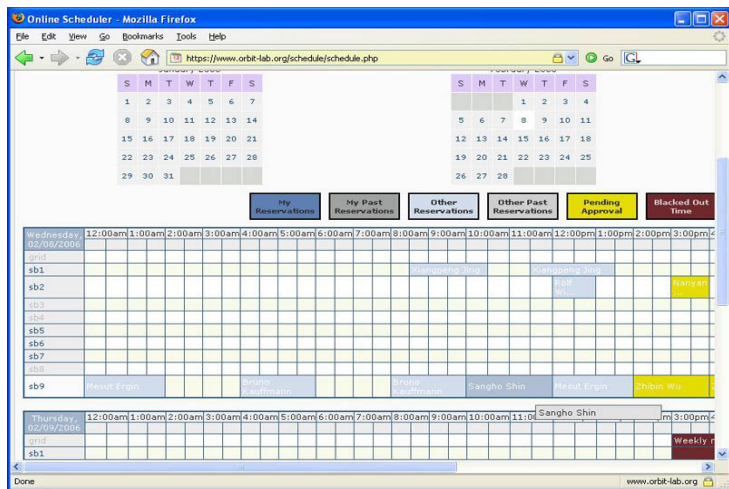


Example graphs - gnuplot (3)

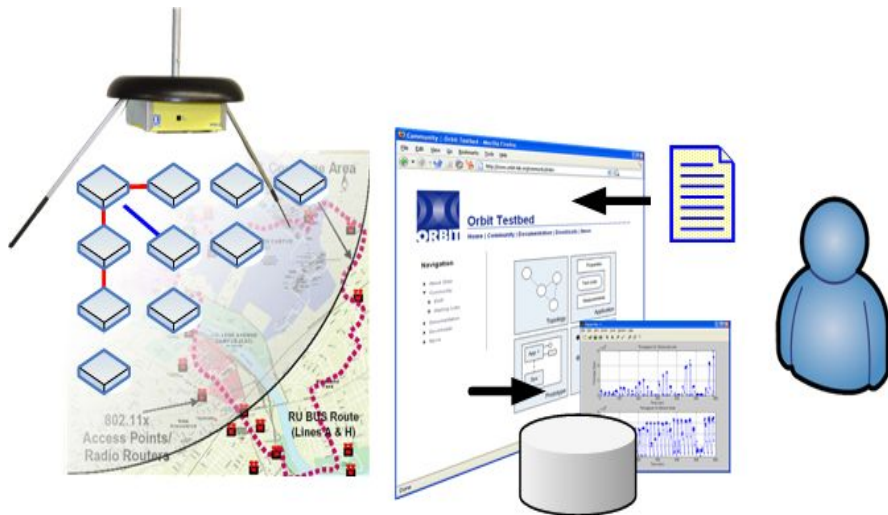
- ◆ If we multiply the number of routing packets by the packet length we get the true overhead of the routing protocol on the channel



Future work



- ◆ Migrate to dual band (2.4/5 GHz) ... in process
- ◆ Add a lab sandbox
- ◆ Test virtual mobility by virtualizing a node – allowing it move in discrete steps



Tutorial 1: pinging with different power levels

- ◆ **Write a script which measures the ping time and checks the number of hops from node x1 to node x7 with two different power settings (x is the row assigned to you)**
 - + Set power to 8dBm ... write to log file /root/pingx8.log
 - + Set power to 19dBm ... write to log file /root/pingx19.log
 - + Set channel to your row (x)
- ◆ **Use the OLSR routing protocol (give it time to settle before starting the pings)**
- ◆ **Use 2 second pings with a packet size of 1000 bytes and run the ping tool for 30 counts.**
- ◆ **Record the average ping time and average number of hops**
- ◆ **Use the ping -R tool to see the route taken for both power settings – is it symmetrical, if not can you explain why not? Does the route change over the 1 minute period? Why? (kill olsrd on your nodes when you are finished)**

Tutorial 2: Throughput vs hop count

- ◆ Using your allocated row (x). Setup a set of static routes on your nodes so that you build a chain from node x1 to node x7
 - + x1->x2->x3->x4->x5->x6->x7
 - Hint use: `route add <dest> gw <gw node>`
- ◆ Now start a TCP iperf server on each node with a 32k buffer length
- ◆ Run a TCP iperf client test from x1->x2, x1->x3 ... x1->x7 for 20 seconds on each test (hint: `man iperf`) ... log each result to `/root/iperfx1_x2`, `/root/iperfx1_x3`
- ◆ Plot the throughput vs hop count using gnuplot
- ◆ What can you say about the trend – does it decrease logarithmically/linearly ...