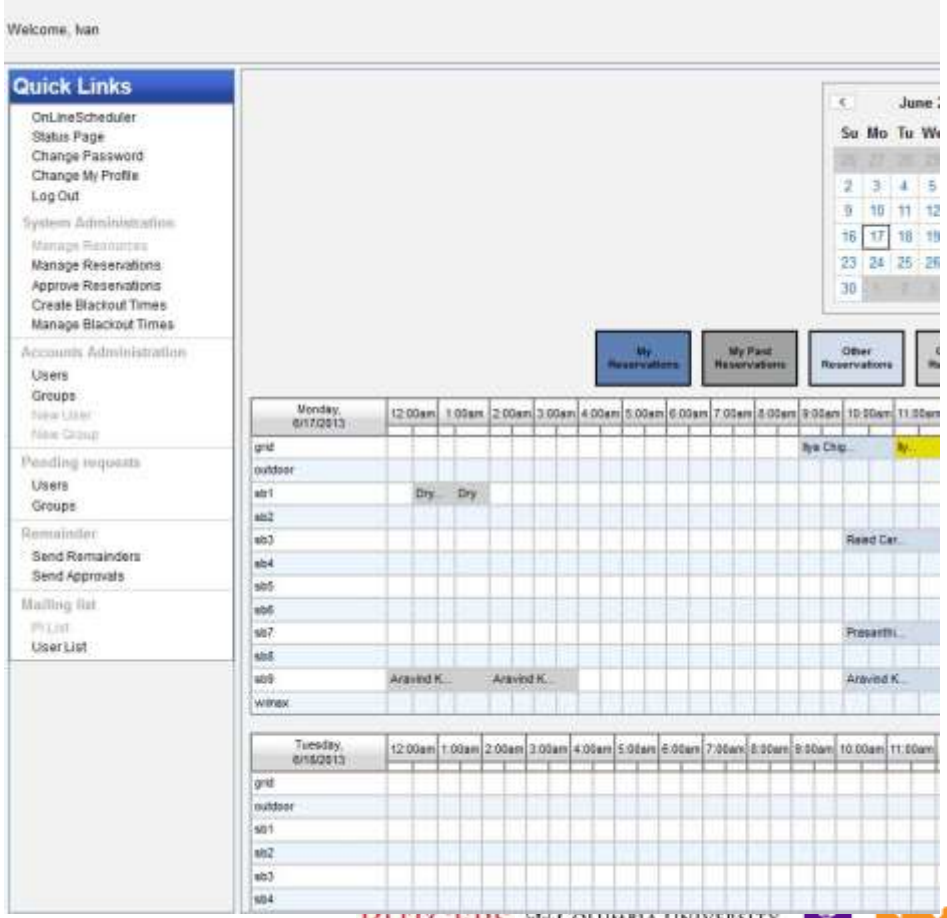
COSMOS/ORBIT Tutorials (Introduction)

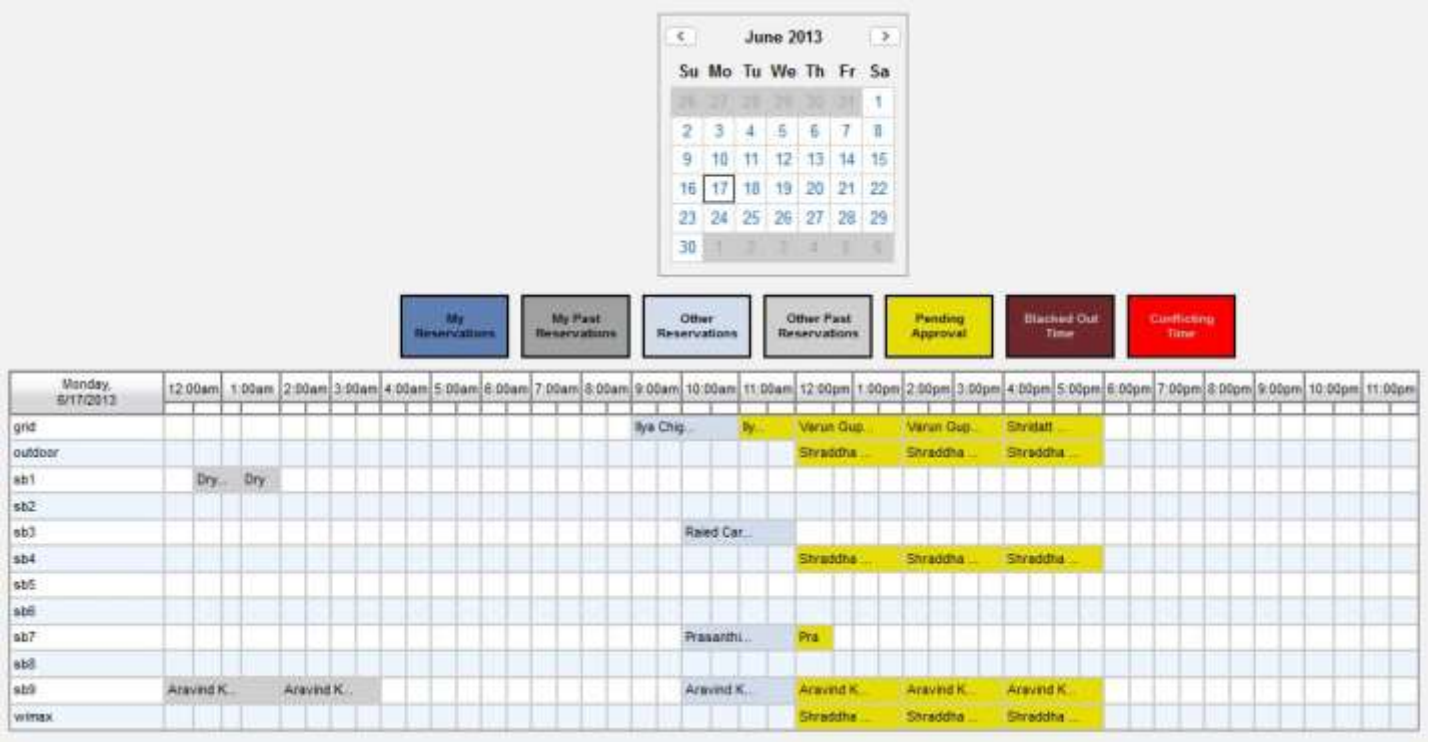
Control Panel



COSMOS

RUTGERS COLUMBIA UNIVERSITY IN THE CITY OF NEW YORK NYU NYC IBM

Reservation System



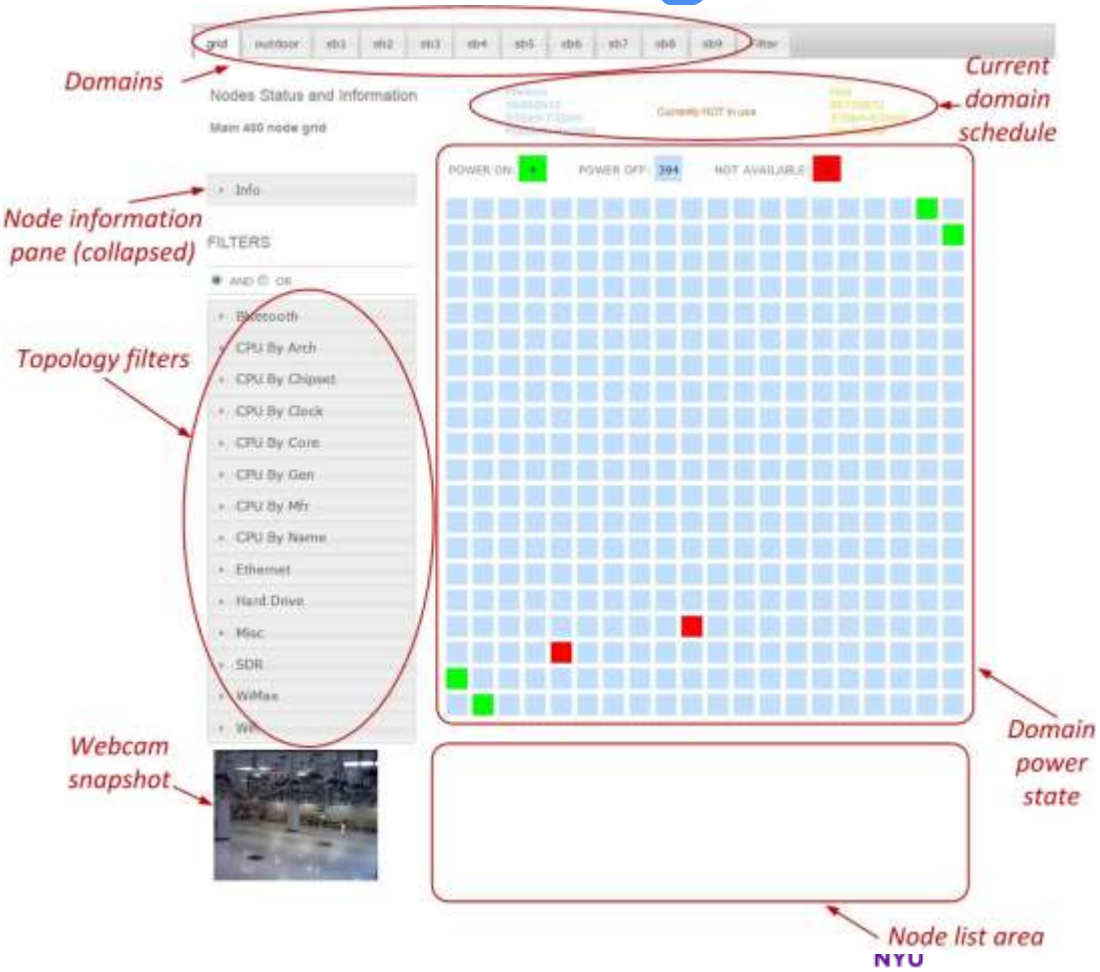
COSMOS

RUTGERS COLUMBIA UNIVERSITY IN THE CITY OF NEW YORK NYU NYC IBM

Auto-approval

- Two stage algorithm:
  - "Early bird" – runs once a day (at 2 PM) and resolves conflicts and approves first two hours for all users for the next day
    - (e.g if you ask for your first slot daily slot from 10-12 the next day , at 2 PM a day earlier you will know wheather you got it).
  - "Just in time" – for reservations made after 2 PM or for more than 2 hours per day per domain, the slots will be automatically approved at the beginning of the slot.
- Conflicts are resolved based on usage in the last three weeks
  - (the less you (ab)use it the more likely you are to get it ☺).
- Be aware of major (conference) deadlines

Status Page


Domains
Current domain schedule
Node information pane (collapsed)
Topology filters
Webcam snapshot
Domain power state
Node list area
Nodes Status and Information
Main 400 node grid
Info
FILTERS
POWER ON:
POWER OFF:
394
NOT AVAILABLE
Bluetooth
CPU By Arch
CPU By Chipset
CPU By Clock
CPU By Core
CPU By Gen
CPU By Mfr
CPU By Name
Ethernet
Hard Drive
Misc
SDR
WiMax

First Exercise: Account Creation, Scheduler, Login (ssh) and Status Page

ORBIT Management Framework

OMF is a framework to **use** and **manage** experimental platforms (testbeds)

Use

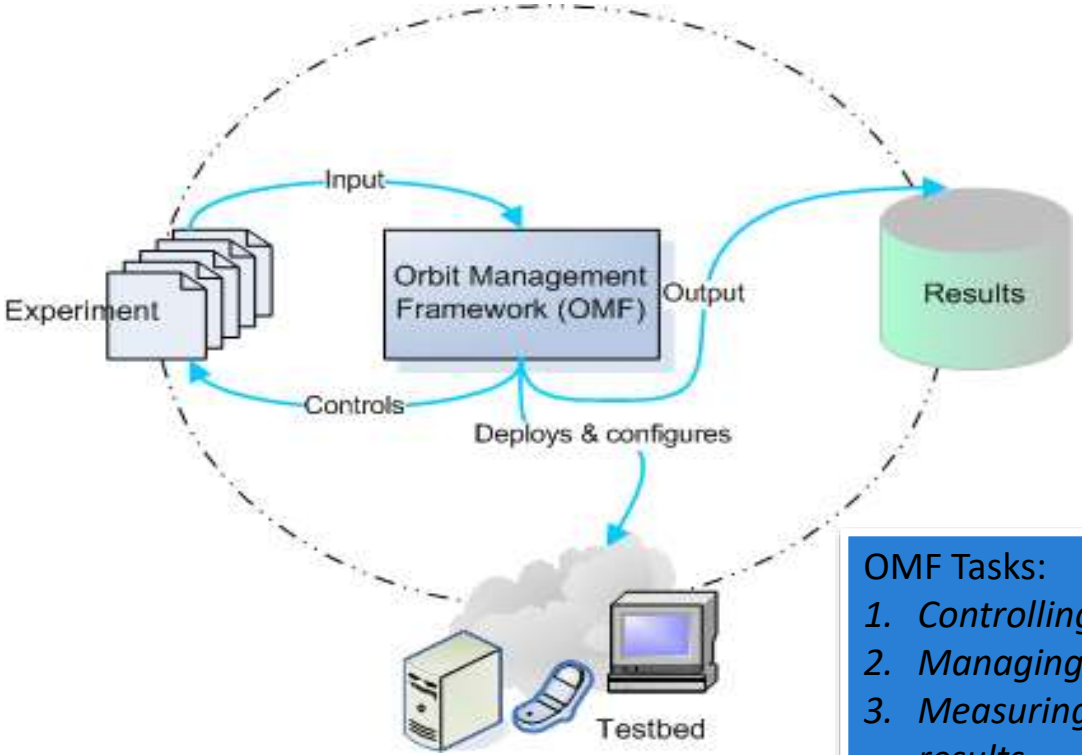
- support "*experiment cycles*" & scientific rigor
- validation, accuracy & reproducibility

Manage

- ease operation and maintenance tasks
- optimize resource utilization inside / across testbeds

Written mostly in Ruby

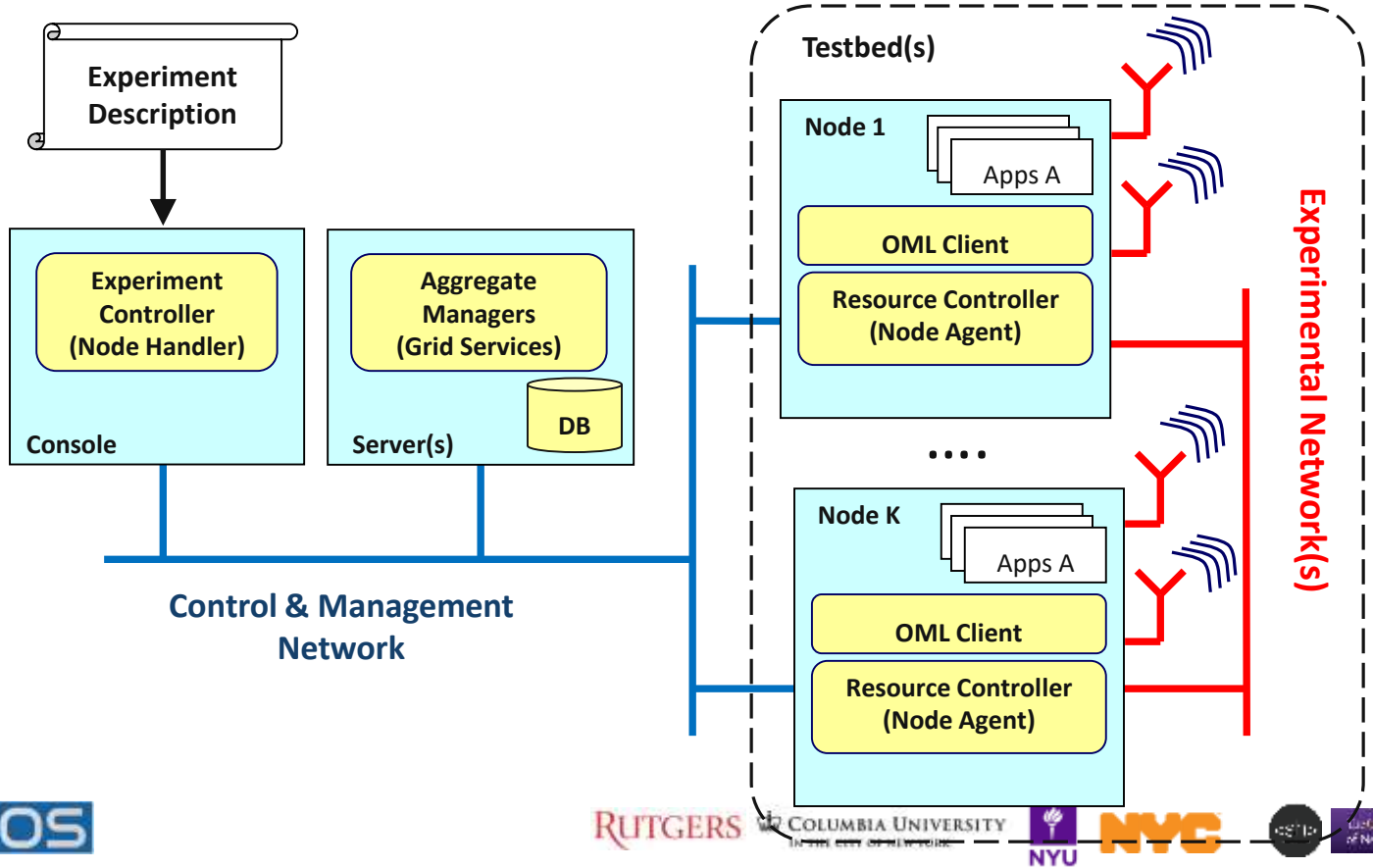
OMF Workflow



OMF Tasks:

1. *Controlling experiments*
2. *Managing the testbed*
3. *Measuring and collecting results*

OMF - Experimenter View



OMF Command

(aka “NodeHandler”)

omf [SUBCOMMAND] [ARGUMENT]...

Subcommand	***Description***
omf help	Display the help for using omf commands.
omf exec	Execute an experiment script.
omf load	Load a disk image on a given set of nodes.
omf save	Save a disk image from a given node into a file.
omf tell	Switch a given set of nodes ON/OFF.
omf stat	Returns the status of a given set of nodes

Second Exercise: Basic OMF commands

omf {tell, stat, load, save}

OMF Experiment Description Language (OEDL)

- Domain-specific Language based on Ruby
- Two parts of experiment description (ED):
  - **Resource requirements and configuration**: specifies experimental resources
  - **Task description**: *state-machine* that enumerates tasks to perform

OEDL Commands

8 groups:

- Top-level commands
- Topology-specific commands
- Group-specific commands
- Prototype-specific commands
- Application-specific commands
- Execution-specific commands
- Resource Paths
- Testbed-specific commands

OEDL Top-level Commands: defProperty

```
defProperty(name, initialValue, description)
```

- **name**: name of the property. This name will be used to refer to this property in any consecutive OEDL commands.
- **initialValue**: the initial value of the property. This also determines the type of the property.
- **description**: Textual description. Used in Experiment Controller's help message, as well as for the default web interface.

Usage:
defProperty('rate', 300, 'Bits per second sent from sender')
defProperty('packetSize', 1024, 'Size of packets sent from sender')

OEDL Top-level Commands: prop

```
prop.propName
prop.propName = newValue
```

- propName: Name of experiment property.
- newValue: New value to assign to the property.

Usage:

```
defProperty('rate', 300, 'Bits per second sent from
sender') …
'rate' => prop.rate
…
[500, 1000, 2000].each { |newRate|
prop.rate = newRate 14
}
```

OEDL Top-level Commands: logging

```
debug(arg1, ...)
info(arg1, ...)
warn(arg1, ...)
error(arg1, ...)
```

- **arg1**: None or more strings to be logged

Usage:

```
info("Starting")
debug(i, " resource(s) are up")
```

Note: DEBUG and INFO log normal progress and can be ignored, while WARNING and ERROR report on abnormal behavior.

OEDL Top-level Commands: wait

wait(time)

- **time**: pause experiment execution for time seconds

Usage:

```
whenAllInstalled {
...
   [500, 1000, 2000].each { |newRate|
      prop.rate = newRate
      wait 30
   }
}
```

OEDL Topology Commands: defTopology

Used to specify topology consisting of a set of nodes and links each with certain characteristics

```
defTopology( name , arrayOfNodes = nil , &block = nil )
```

- **name**: Name of the defined topology.
- **arrayOfNodes**: (optional) array of resources (e.g. nodes) to include in this topology.
  - the list of valid definition patterns are:
    - [x,y]: Describes a single node at location x@y
    - [x1..x2, y]: Describes a set of nodes along a line starting at x1@y and ending at x2@y. For instance, [2..4, 5] defines the nodes [2,5], [3,5], [4,5].
    - [x, y1..y2]: Same as previous, but for the y coordinate.
    - [x1..x2, y1..y2]: This defines a rectangle area of nodes within the grid.
    - [[x1,y1], [x2,y2], [x3,y3]]: An arbitrary long list of single nodes.
- **block**: (optional) a block of commands that can be used to build/configure this topology.

OEDL Topology Commands: defTopology (cont'd)

Topology Sub-Commands	*Description*
addNode(x,y)	Add node at location x@y to the topology.
removeNode(x,y)	Remove node at location x@y from the topology.
addLink (x, y, spec)	Adds a link between nodes x and y and configures it with the characteristics defined in the 'spec'.'spec' is a hash with the following valid keys {:rate , :per, :delay, :asymmetric}
RemoveLink (x, y)	Severs the link between nodes x and y.
size()	Return the number of nodes in this topology.
getNode(index)	Return the node at the position index in this topology. Return nil if index is greater than the number of nodes in the topology.
getFirstNode()	Return the node at the 1st position in this topology.
getLastNode()	Return the node at the last position in this topology.
getRandomNode()	Return a random node from this topology.
getUniqueRandomNode()	Return a unique random node from this topology. When all the available nodes in this topology have been drawn, this method will return nil and output a warning message to the console.
eachNode(&block)	Execute the commands in block on each node within this topology.
setStrict()	Set the strict flag for this topology. By default, the strict flag is NOT set for a topology.
unsetStrict()	Clear the "strict" flag. By default, the strict flag is NOT set for a topology.
hasNode(x, y)	Return true if the node at location x@y is part of this topology, return false otherwise.

OEDL Topology Commands: defTopology (cont'd)

```
defTopology('test:topo:circle') { |t|
    nodeNum = 8
    xCenter = 10
    yCenter = 10
   radius = nodeNum
   # use simple 4-way algorithm to pick the nodes
   r2 = radius * radius
   t.addNode(xCenter, yCenter + radius)
   t.addNode(xCenter, yCenter - radius)
   (1..radius).each { |x|
       y = (Math.sqrt(r2 - x*x) + 0.5).to_i
       t.addNode(xCenter + x, yCenter + y)
       t.addNode(xCenter + x, yCenter - y)
       t.addNode(xCenter - x, yCenter + y)
       t.addNode(xCenter - x, yCenter - y)
   }
}
```

OEDL Group Commands: defGroup

defGroup(groupName, selector, &block = nil)

- **groupName**: name of the defined set of resources
- **selector**: selects the resources to be contained in this set. Group selector can be also defined with topology URI (i.e. set of nodes that form the topology)
- **block**: instructions for all resources in the group

Usage:

```
defGroup('sender1', [1, 1])      # set contains 1 resource
defGroup('sender2', [2, 1..8]) # set contains  8 resources [2,1], [2,2], ... [2,8]
defGroup('sender', ['sender1', 'sender2', [3, 1..8]]) {|node|
   node.prototype("test:proto:sender", {
      'destinationHost' => '192.168.1.1',
      ...
   }
   node.net.w0.mode = "master" #802.11 Master Mode
}
```

OEDL Group Commands: defGroup (cont'd)

addApplication	Install an application on a node
exec	Execute a command on all nodes in this group.
image	Check whether a node boots in the required image. (not available in version 4.4 of the NH)
netmask	This is the network mask resource path.
onNodeUp	Execute a block of commands when a node is up.
pxeImage(...)	Instructs a resource to boot from a network PXE image (recommended for expert users only).

OEDL Group Commands: group and allGroups

```
group(groupSelector).command()
group(groupSelector).resource_path = value
group(groupSelector).resource_path {...}
```

- **groupSelector:** set of resources to use.
- **command**: command to run for that set.
- **resource_path:** is the parameter to be set
- **value:** is the value to assign to the resource path parameter

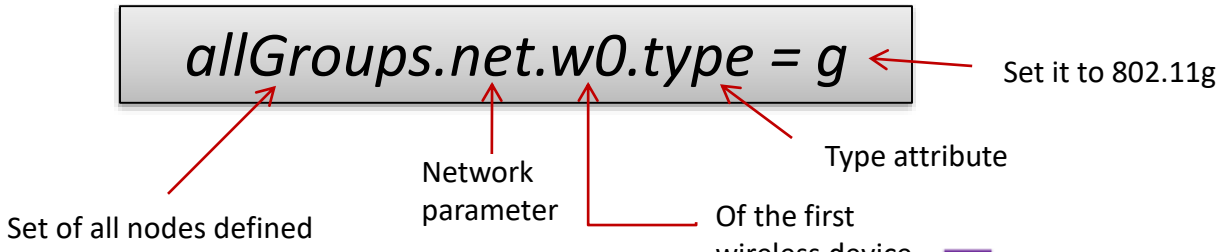
Usage:

```
group('sender1').startApplications
group(['s1', 'r1']).net.w0.essid = "orbit"
allGroups.net.w0 { |w|
   w.essid = "orbit“
}
```

Resource Paths

- A resource path allows the access and the value assignment of a specific configuration parameter of a resource
- **Can be** used in any section of the ED.
- Follow a hierarchical organization:

  <resource_selector>.<hierarchical_path>



net - network resource path

- {e0, e1} Ethernet interface
  - arp = true|false En/disable ARP
  - forward = true|false Enable forwarding
  - ip = address/netmask IP address of interface
  - up = true|false En/disable interface
- {w0, w1} Wireless interface
  - All the above
  - channel (intel only) = 1..11; 36, 40, 44, 48, 52, 56, 60, 64, 149, 153, 157, 161
  - frequency (intel only) = 2.412..2.462GHz (5 Mhz steps); 5.18GHz (20Mhz steps)
  - essid = arbitrary string
  - mode = master|managed|monitor, ad-hoc (intel only)
  - rts (atheros only) = packetSizeThreshold [bytes]
  - rate (intel only) = 1, 5, 11; 6, 9, 12, 18, 24, 36, 48, 54
  - tx_power = -12..15 dBm (intel), 0..20 dBm (atheros)
  - type = a/b/g

hello-world-wireless.rb

```
defProperty('res1', 'node1-1.grid.orbit-lab.org', "ID of sender node")
defProperty('res2', 'node1-2.grid.orbit-lab.org', "ID of receiver node")
defProperty('duration', 60, "Duration of the experiment")
```

```
defGroup('Sender', property.res1) do |node|
  node.addApplication("test:app:otg2") do |app|
    app.setProperty('udp:local_host', '192.168.0.2')
    app.setProperty('udp:dst_host', '192.168.0.3')
    app.setProperty('udp:dst_port', 3000)
    app.measure('udp_out', :samples => 1)
  end
  node.net.w0.mode = "adhoc"
  node.net.w0.type = 'g'
  node.net.w0.channel = "6"
  node.net.w0.essid = "helloworld"
  node.net.w0.ip = "192.168.0.2"
end
```

```
defGroup('Receiver', property.res2) do |node|
  node.addApplication("test:app:otr2") do |app|
    app.setProperty('udp:local_host', '192.168.0.3')
    app.setProperty('udp:local_port', 3000)
    app.measure('udp_in', :samples => 1)
  end
  node.net.w0.mode = "adhoc"
  node.net.w0.type = 'g'
  node.net.w0.channel = "6"
  node.net.w0.essid = "helloworld"
  node.net.w0.ip = "192.168.0.3"
end
```

```
onEvent(:ALL_UP_AND_INSTALLED) do |event|
  info "This is my first OMF experiment"
  wait 10
  allGroups.startApplications
  info "All my Applications are started now..."
  wait property.duration
  allGroups.stopApplications
  info "All my Applications are stopped now."
  Experiment.done
end
```

COSMOS Summary

- Focus on ultra high bandwidth, low latency, edge cloud
- Open platform (building on ORBIT) integrating mmWave, SDR, and optical x-haul
- 1 sq mile densely populated area in West Harlem
- Local community outreach
- Research community:
  - Develop future experiments, provide input
  - (short term) get involved in the educational outreach

More information:

http://advancedwireless.org http://www.orbit-lab.org http://www.cosmos-lab.org
http://omf.orbit-lab.org http://oml-doc.orbit-lab.org

COSMOS RUTGERS COLUMBIA UNIVERSITY IN THE CITY OF NEW YORK NYU NYC IBM