

# OMNeT++ - Tutorial 2

Brian Ricks  
Wireless Network Security  
Spring 2014

# Where we left off

- Basic OMNeT++/Inet intro
  - Installing
  - Basics of a simulation model
  - TicToc example
  - (very basic) Analysis



# What Now?

- Create a new project with Inet support
  - Needed for HW2
- Learn how to run batch simulations
  - Needed for HW2
- Learn more analysis
  - ... Needed for HW2?
    - Yes!
- Bonus!
  - Initial node position



# Creating a New Project

- File->new->OMNeT++ Project...
  - Create a name, and keep the default location
  - Select blank project, then finish
- Link Inet project
  - Right-click on your project
  - Select 'properties' → Project References
  - Select Inet from the list
  - What if Project References is not an option?
    - Make sure project is open (right-click project → Open Project)

# Creating a New Project

- Can I use an existing project?
  - Sure, just make sure it is linked to Inet
- Can I create a folder within the Inet project and use that as my project?
  - Technically, yes
    - This would not be a 'project' (simply a folder within the Inet project)
    - Inet makefile would need to be regenerated
    - Your project would not be standalone, would now be part of Inet

# Running Your Project

- What is required?
  - package.ned: defines package for your project
  - omnetpp.ini: configuration file
- Two methods to run:
  - IDE
    - Click the green circle arrow button
  - Command-line
    - Directly run project executable with appropriate args
    - opp\_runall

# Command-line Issues

- Executable won't load
  - Libraries missing
    - In Windows, add the following to your PATH:
      - `<omnet-root>\lib;<omnet-root>\samples\inet\src`
    - Alternatively, run from IDE (preferred)
  - Cannot find NED files
    - Executable by default only search for NED files in CWD
      - `-n` and then paths to other NED files (prolly in Inet folders)
      - Copy and paste these args from IDE console
        - Alternatively, run from IDE (preferred)

# Command-line Issues

- I tried using `opp_run` instead, and now the simulation can't find my classes
  - `opp_run` is used if project compiled as library
    - Since you compiled as a standalone executable, there is no library with your classes



# Batch Simulations

- This is process of executing multiple *runs*
  - What's a run?
    - A single instance of a simulation, using some set of values for parameters
  - Multiple runs are usually used for two reasons:
    - To run the same configuration with different values for parameters
    - To run the same configuration with same parameter values but with different seeds

# Running Batch Simulations

- Command-line only
  - Hey, you told us to use the IDE!!



But, there is a catch...

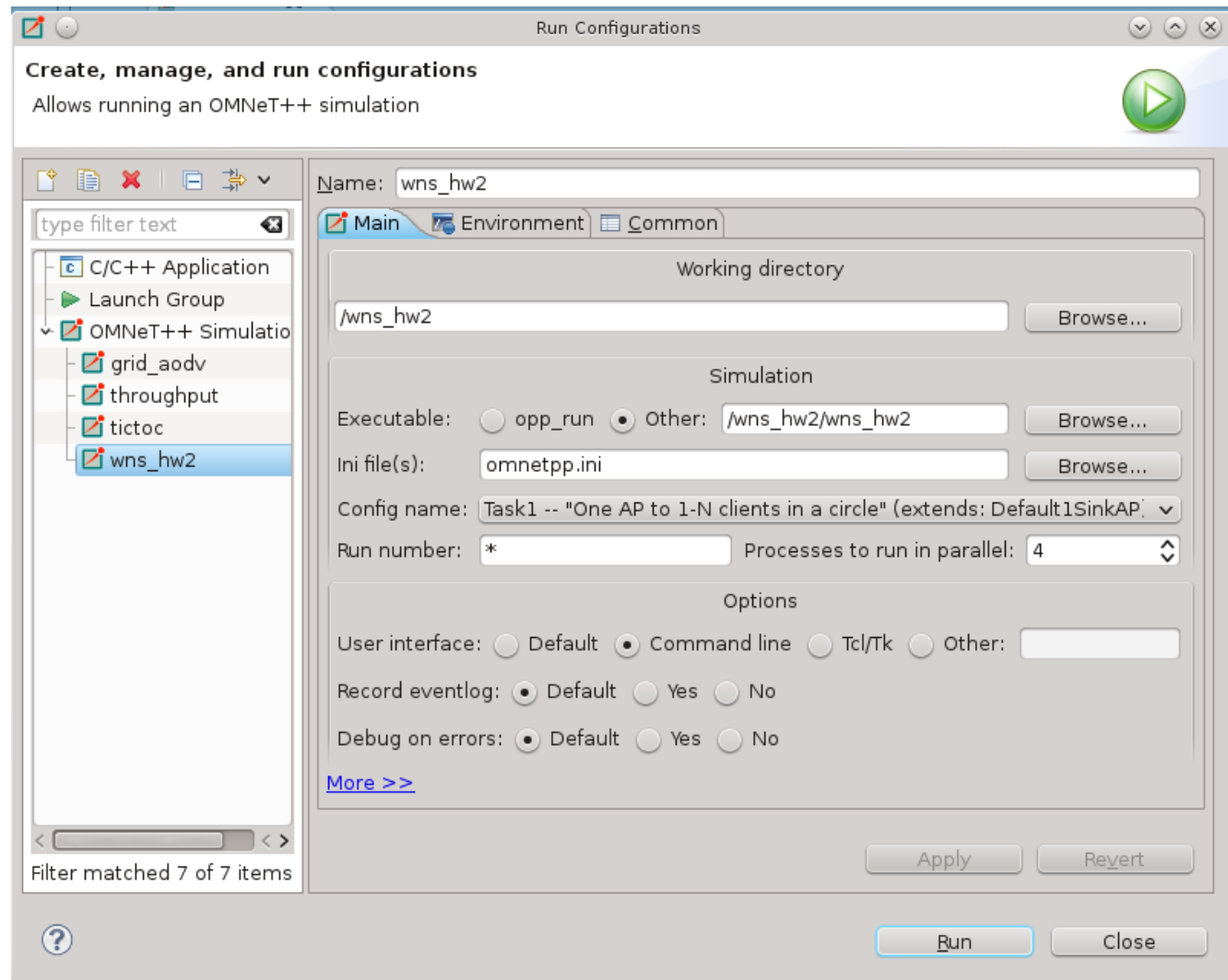
# Running Batch Simulations

- IDE can act as front-end for command-line launcher!
  - This is accomplished by changing the run configuration
  - Run --> Run Configurations
    - Select appropriate configuration
    - Select 'Command Line' for user interface
    - Put '\*' for run number (all runs)
    - (optional) match the processes to run in parallel with your CPU's core count



# Running Batch Simulations

- IDE can act as front-end for command-line launcher!



# Running Batch Simulations

- Changing the run configuration is persistent
  - If you need to go back to the GUI (Tkenv), switch configuration in the ini, or anything else, you need to go back to the Run Configurations dialog...

# Setting up a Batch Simulation

- This is done in the `omnetpp.ini` file
  - We can setup loops, with each loop being one run
    - This is done using *iteration variables*

# Iteration Variables

- Syntax:  $module.parameter = \{var\_name=range\ step\ step\_num\}$ 
  - $module.parameter$  is the same syntax as for any parameter assignment in the ini
  - $var\_name$  is the name of the iteration variable
  - $range$  is the set of numbers we iterate over
    - 1..10 means a range from 1 to 10
    - 1,2,5,8..10 means step through 1,2,5,8,9,10
  - $step$  is optional, specifies to use  $step\_num$ 
    - Default step is 1
    - 1..11 step 2 would step through 1,3,5,7,9,11

# Iteration Variables

- We declare iteration variables in configuration blocks
  - We reference them using this syntax:  $\${var\_name}$
- For each value of the iteration variable, that block is run once with that value
  - Consider this example:  
[Config Task1]  
WifiNetwork.numClients =  $\${N=1..10 \text{ step } 1}$
  - Doing a batch simulation of Task1 over all runs, we will execute Task1 10 times, for each  $\{N\}$  (N = iteration variable)



# Iteration Variables

- Multiple iteration variables can be declared in the same block
  - Consider this example:  
[Config Task1]  
WifiNetwork.numClients =  $\{N\_Cli=1..10 \text{ step } 1\}$   
WifiNetwork.numAPs =  $\{N\_APs=1..5 \text{ step } 1\}$
  - Doing a batch simulation of Task1 over all runs, we will execute Task1 50 times:
    - For each  $\{N\_Cli\}$ , we execute 5 times for each  $\{N\_APs\}$
    - $\{N\_APs\}$  is nested inside  $\{N\_Cli\}$  due to declaration ordering

# Iteration Variables

- Unique seed per run
  - $seed\text{-}set = \{\textit{repetition} \mid \textit{runnumber}\}$ 
    - Default is *runnumber*
      - Produces a unique seed per run
    - *Repetition* produces a unique seed per repetition
      - What is a repetition?
        - A run with a unique value for the repetition iteration variable
        - Declared with *repeat*
          - *Repeat = 20* would cause each run to repeat 20 times
          - Repetitions are in themselves runs, just have a reserved name and meaning in OMNeT++

# Iteration Variables

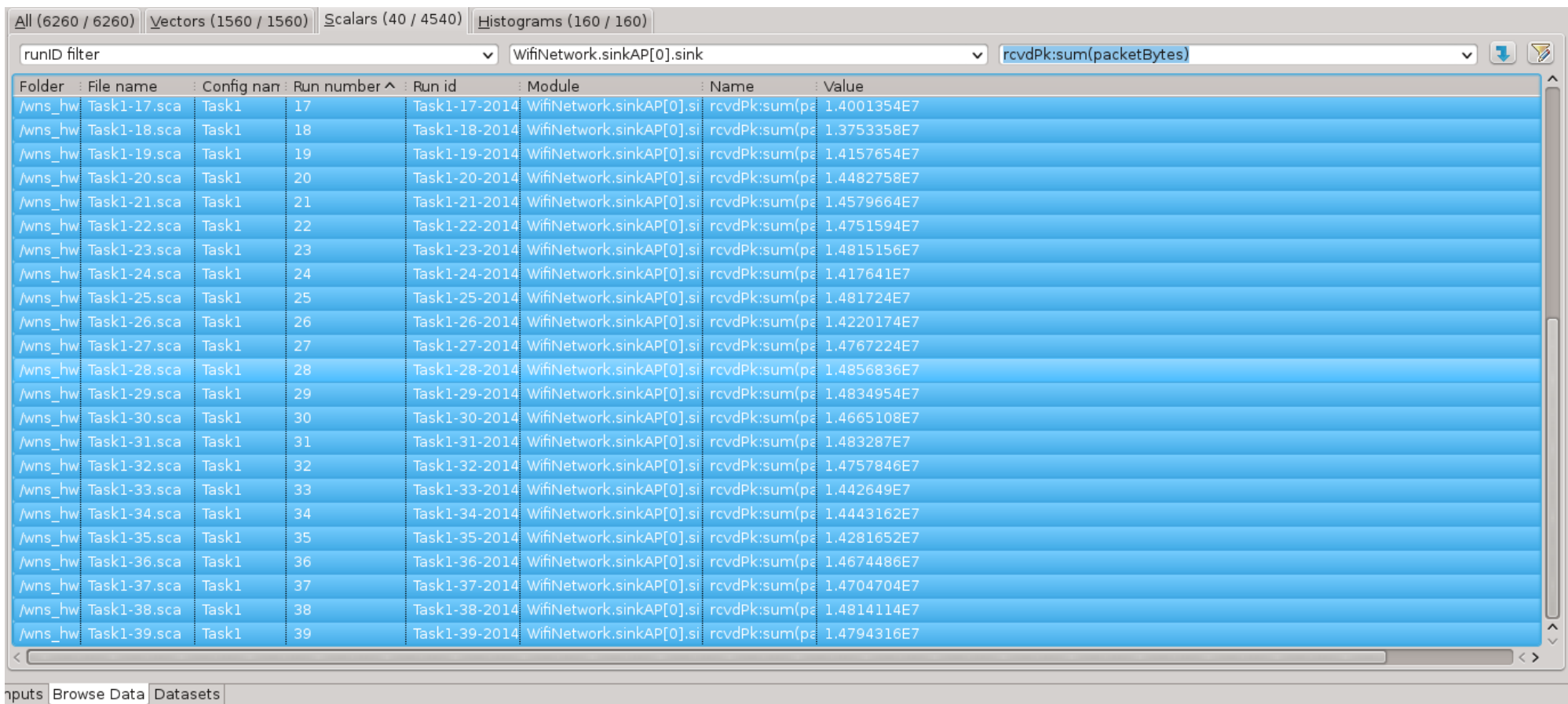
- Multiple runs of a specific configuration, with unique seed per repetition:
  - Using the same example:  
[Config Task1]  
WifiNetwork.numClients =  $\{N=1..10 \text{ step } 1\}$   
repeat = 20  
seed-set =  $\{\text{repetition}\}$
  - For each  $\{N\}$ , run 20 times, each time using a different seed
    - 200 runs in total
    - When using *repeat*,  $\{\text{repetition}\}$  will always be nested lowest

# Grabbing Statistics

- After a batch run, click on any of the \*.vec or \*.sca files to create an ANF
  - Notice how the ANF name does not append a specific run
    - They are grouped together (awesome)!
  - How do I average the data?
    - Use the filters to narrow down the module and scalar you need
      - Select everything, and plot
        - Watch the magic

# Grabbing Statistics

- For example, using the HW2 scenario, with 2 clients and a single AP:

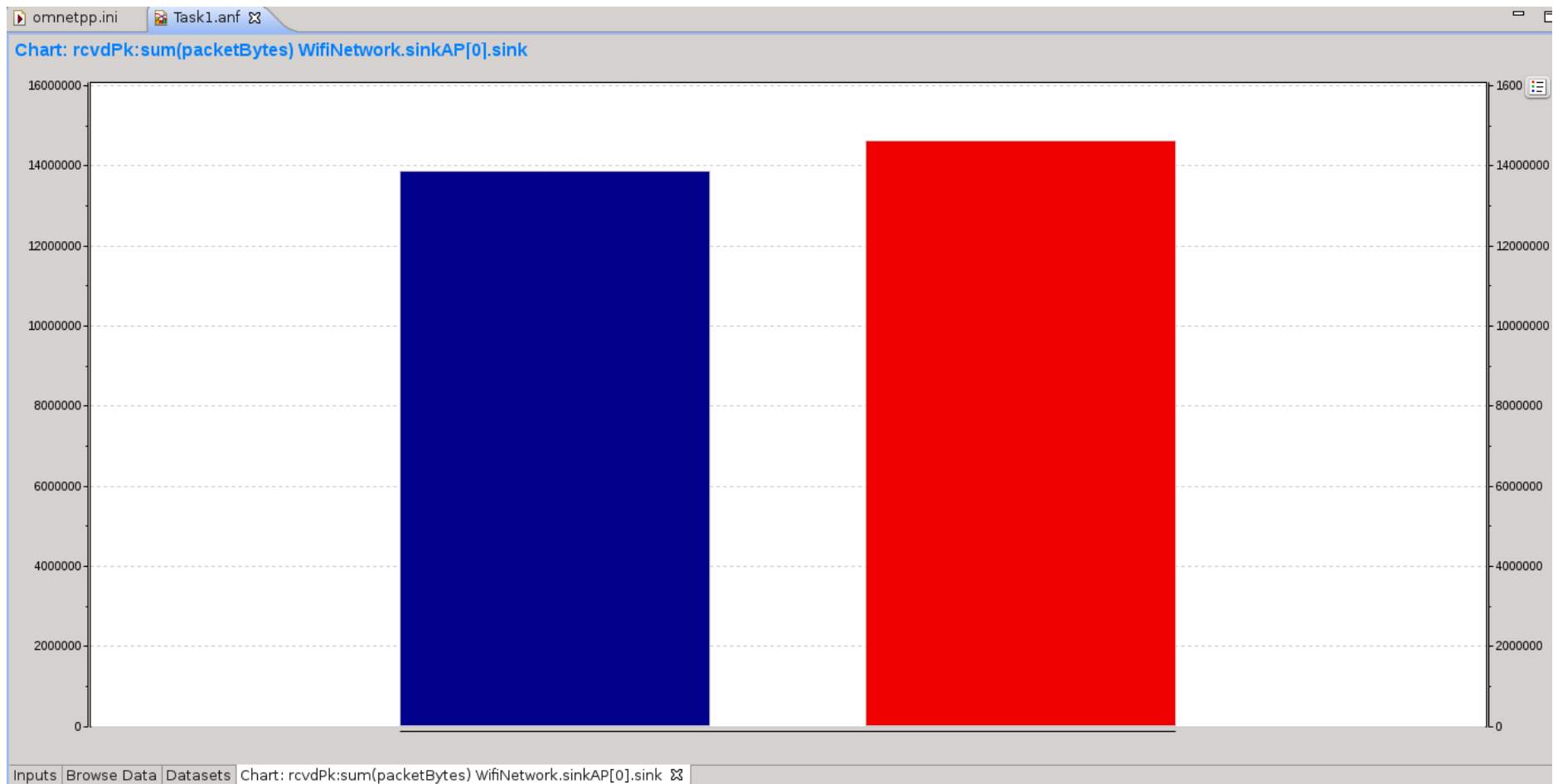


The screenshot shows a software interface with a table of statistics. The table has columns for Folder, File name, Config name, Run number, Run id, Module, Name, and Value. The data is filtered for 'WifiNetwork.sinkAP[0].sink' and shows 'rcvdPk:sum(packetBytes)' values for runs 17 through 39. The values range from approximately 1.4001354E7 to 1.4794316E7.

Folder	File name	Config name	Run number	Run id	Module	Name	Value
/wns_hw	Task1-17.sca	Task1	17	Task1-17-2014	WifiNetwork.sinkAP[0].si	rcvdPk:sum(pa	1.4001354E7
/wns_hw	Task1-18.sca	Task1	18	Task1-18-2014	WifiNetwork.sinkAP[0].si	rcvdPk:sum(pa	1.3753358E7
/wns_hw	Task1-19.sca	Task1	19	Task1-19-2014	WifiNetwork.sinkAP[0].si	rcvdPk:sum(pa	1.4157654E7
/wns_hw	Task1-20.sca	Task1	20	Task1-20-2014	WifiNetwork.sinkAP[0].si	rcvdPk:sum(pa	1.4482758E7
/wns_hw	Task1-21.sca	Task1	21	Task1-21-2014	WifiNetwork.sinkAP[0].si	rcvdPk:sum(pa	1.4579664E7
/wns_hw	Task1-22.sca	Task1	22	Task1-22-2014	WifiNetwork.sinkAP[0].si	rcvdPk:sum(pa	1.4751594E7
/wns_hw	Task1-23.sca	Task1	23	Task1-23-2014	WifiNetwork.sinkAP[0].si	rcvdPk:sum(pa	1.4815156E7
/wns_hw	Task1-24.sca	Task1	24	Task1-24-2014	WifiNetwork.sinkAP[0].si	rcvdPk:sum(pa	1.417641E7
/wns_hw	Task1-25.sca	Task1	25	Task1-25-2014	WifiNetwork.sinkAP[0].si	rcvdPk:sum(pa	1.481724E7
/wns_hw	Task1-26.sca	Task1	26	Task1-26-2014	WifiNetwork.sinkAP[0].si	rcvdPk:sum(pa	1.4220174E7
/wns_hw	Task1-27.sca	Task1	27	Task1-27-2014	WifiNetwork.sinkAP[0].si	rcvdPk:sum(pa	1.4767224E7
/wns_hw	Task1-28.sca	Task1	28	Task1-28-2014	WifiNetwork.sinkAP[0].si	rcvdPk:sum(pa	1.4856836E7
/wns_hw	Task1-29.sca	Task1	29	Task1-29-2014	WifiNetwork.sinkAP[0].si	rcvdPk:sum(pa	1.4834954E7
/wns_hw	Task1-30.sca	Task1	30	Task1-30-2014	WifiNetwork.sinkAP[0].si	rcvdPk:sum(pa	1.4665108E7
/wns_hw	Task1-31.sca	Task1	31	Task1-31-2014	WifiNetwork.sinkAP[0].si	rcvdPk:sum(pa	1.483287E7
/wns_hw	Task1-32.sca	Task1	32	Task1-32-2014	WifiNetwork.sinkAP[0].si	rcvdPk:sum(pa	1.4757846E7
/wns_hw	Task1-33.sca	Task1	33	Task1-33-2014	WifiNetwork.sinkAP[0].si	rcvdPk:sum(pa	1.442649E7
/wns_hw	Task1-34.sca	Task1	34	Task1-34-2014	WifiNetwork.sinkAP[0].si	rcvdPk:sum(pa	1.4443162E7
/wns_hw	Task1-35.sca	Task1	35	Task1-35-2014	WifiNetwork.sinkAP[0].si	rcvdPk:sum(pa	1.4281652E7
/wns_hw	Task1-36.sca	Task1	36	Task1-36-2014	WifiNetwork.sinkAP[0].si	rcvdPk:sum(pa	1.4674486E7
/wns_hw	Task1-37.sca	Task1	37	Task1-37-2014	WifiNetwork.sinkAP[0].si	rcvdPk:sum(pa	1.4704704E7
/wns_hw	Task1-38.sca	Task1	38	Task1-38-2014	WifiNetwork.sinkAP[0].si	rcvdPk:sum(pa	1.4814114E7
/wns_hw	Task1-39.sca	Task1	39	Task1-39-2014	WifiNetwork.sinkAP[0].si	rcvdPk:sum(pa	1.4794316E7

# Grabbing Statistics

- For example, using the HW2 scenario, with 2 clients and a single AP:



# Grabbing Statistics

- For example, using the HW2 scenario, with 2 clients and a single AP:
  - It's magic! It even split up the data by client!
    - Well, not really magic. OMNeT++ groups the data by non-repetition iterator variables (client count being one), and averages according to repetition.
    - Makes your job much easier!\*

\*Only if paying attention right now

# Initial Node Position

- Suppose you want to position your nodes in a circle, but lets say the number of nodes is variable
  - Also suppose they need to be evenly spaced
- Does OMNeT++ have a really easy way to set this up?
  - Yes! Automatic layouts. One tag in the display string will accomplish this.
    - The 'p' tag, third argument is the layout.
      - 'ri' gives a nice ring layout



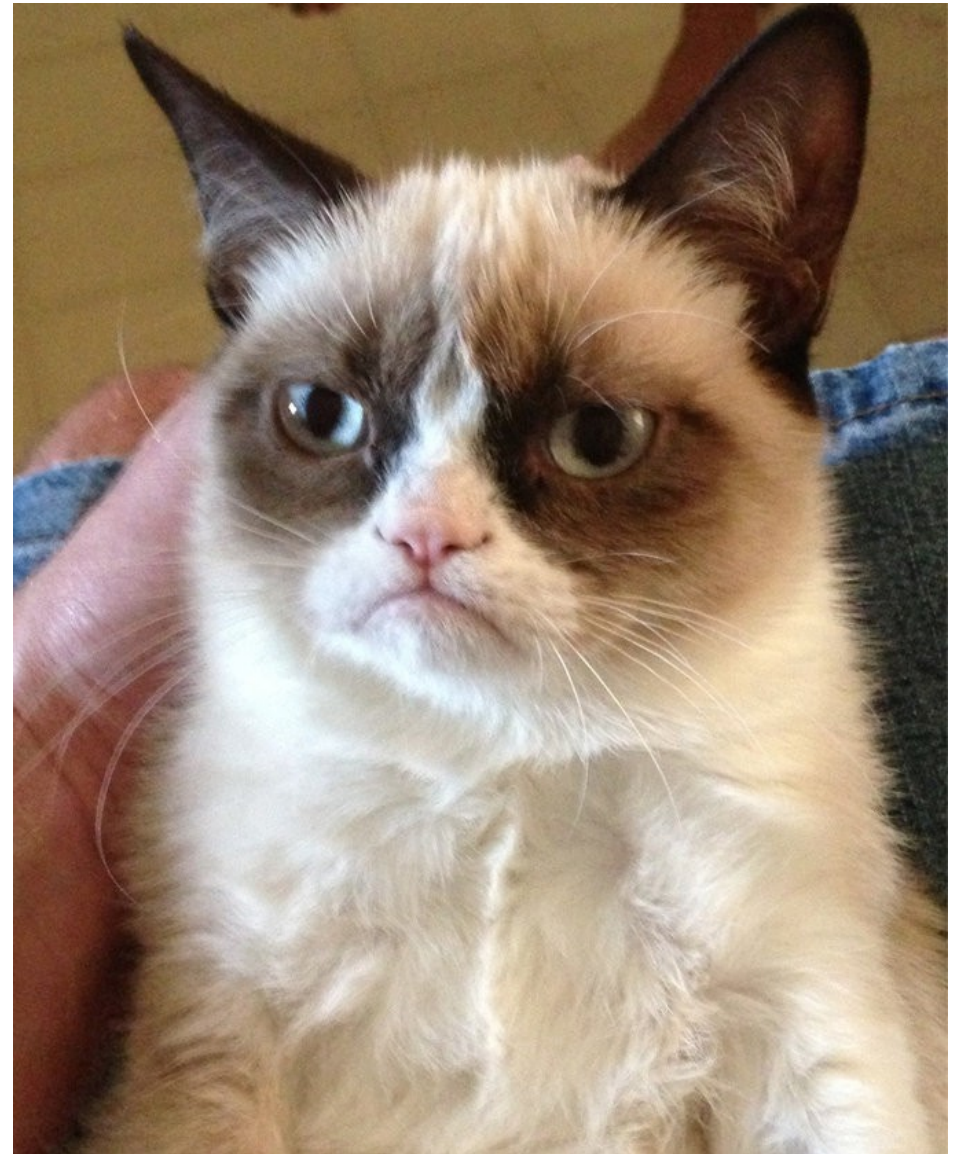
# Initial Node Position

- Does this work in Inet?
  - No



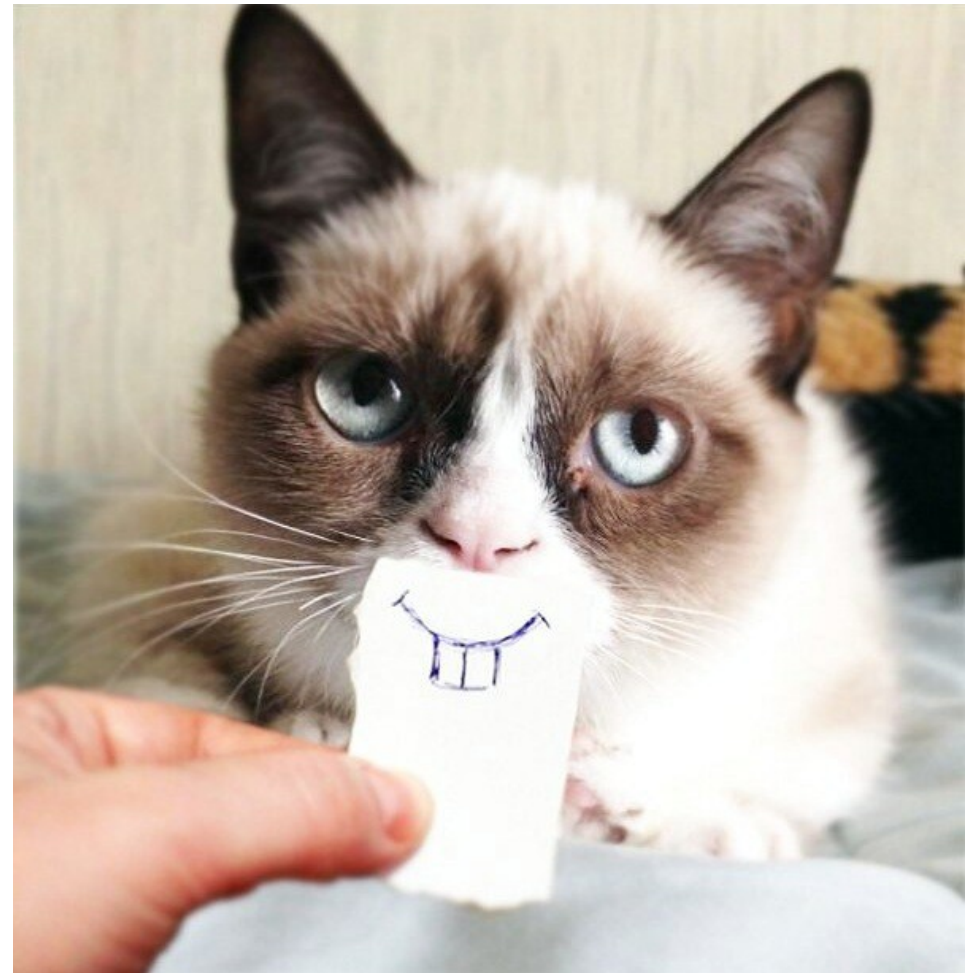
# Initial Node Position

- Why not?
  - Laziness



# Initial Node Position

- Fortunately, there are other ways



# Using CircleMobility Module

- Make sure your nodes are using this module instead of say StationaryMobility
  - For example, the Throughput module's mobility module would need to be changed to CircleMobility

# Using CircleMobility Module

- Parameters of interest:
  - `cx`, `cy`, `cz`: defines center of the circle
    - So to have a circle of clients around an AP, the circle center will have the same coordinates as the AP
  - `r`: defines radius of the circle
  - `StartAngle`: defines initial angle of node, in degrees

# Using CircleMobility Module

- How can we have the startAngle for each client unique such that they are equally spaced?
  - Pretty awesomely





# Using CircleMobility Module

- There are 360 degrees in a circle



# Using CircleMobility Module

- There are 360 degrees in a circle
  - Divide by number of nodes, and we get equal slices around the circle
    - Multiply this by a node's index, and we get a unique slice per node



# Using CircleMobility Module

- For example:
  - `WifiNetwork.node[*].mobility.startAngle = parentIndex() * 360deg/${N}`
    - The '\*' means to apply the assignment to all nodes in the vector
    - `parentIndex()` is a NED function which returns the current index of the node being assigned
    - `${N}` is an iteration variable which holds the number of nodes
      - Refer to earlier slides on iteration variables (`${N}` does not refer to the node vector size, though here they are the same)

# Another Method

- What if I don't want to use CircleMobility?
  - Useful if you don't want to mess with inet modules which don't use CircleMobility (most of them)
  - Also useful if you want your nodes in a circle initially but want to use a different mobility model
- Define initial position of nodes using circle equations for x,y pos, given radius and circle center.

# Another Method

- For example:

$\text{WifiNetwork.node[*]**.initialX} = 100\text{m} + 95\text{m} * \cos(\text{parentIndex() } * (6.283185 / \{N\}))$

$\text{WifiNetwork.node[*]**.initialY} = 100\text{m} + 95\text{m} * \sin(\text{parentIndex() } * (6.283185 / \{N\}))$

- 100m = X, Y pos of circle center
- 95m = radius of circle
- $\cos()$ ,  $\sin()$  = NED functions, they take radians
  - Hence why we multiply the  $\text{parentIndex()}$  by  $2\text{PI}$  approximately

# THE END (for now)

