WASP Manual

1 Introduction

WASP is a programming language for a subset of wireless sensor network applications that use stationary nodes to periodically sample data and transmit them to a base station. Data can be processed locally and aggregated at the network level.

When you start the test, the main window of the **WASP programming environment** will be open for you on the desktop, as shown in Figure 1. Please keep it open during the test. If you close the main window by accident, please ask your instructor to reopen it. The **WASP programming environment** is where your program is edited and simulated. A network composed of four nodes is simulated in this test.

2 WASP Programming Interface

The WASP main window is shown in Figure 1. It contains a menu bar and two tabs. The **Code** tab is where your program is composed. The **Results** tab displays the simulation results.

2.1 Compose WASP program in the Code tab

The **Code** tab has two parts: **Node-level Code** and **Network-level Code**. The **Node-level Code** area lets programmers describe functionality of each node, which includes sampling and data processing. Instructions in the **Node-level Code** area are executed on every node in the network. The **Network Code** area lets programmers specify how node-level data are aggregated at the network level and transmitted to the base station.

Node-level Code

To add code for data sampling, click the **Sample** button. It will pop up a dialog as shown in Figure 2. To compose a sampling instruction, first select the sensor type to read data from, then specify whether a

Table 1: Node-Level Data Processing Functions

Function name	Arguments	Returned result
max_time	*variable[m:n]*	the maximum value in *variable[m:n]*
min_time	*variable[m:n]*	the minimum value in *variable[m:n]*
avg_time	*variable[m:n]*	average of the values in *variable[m:n]*
sum_time	*variable[m:n]*	sum of the values in *variable[m:n]*

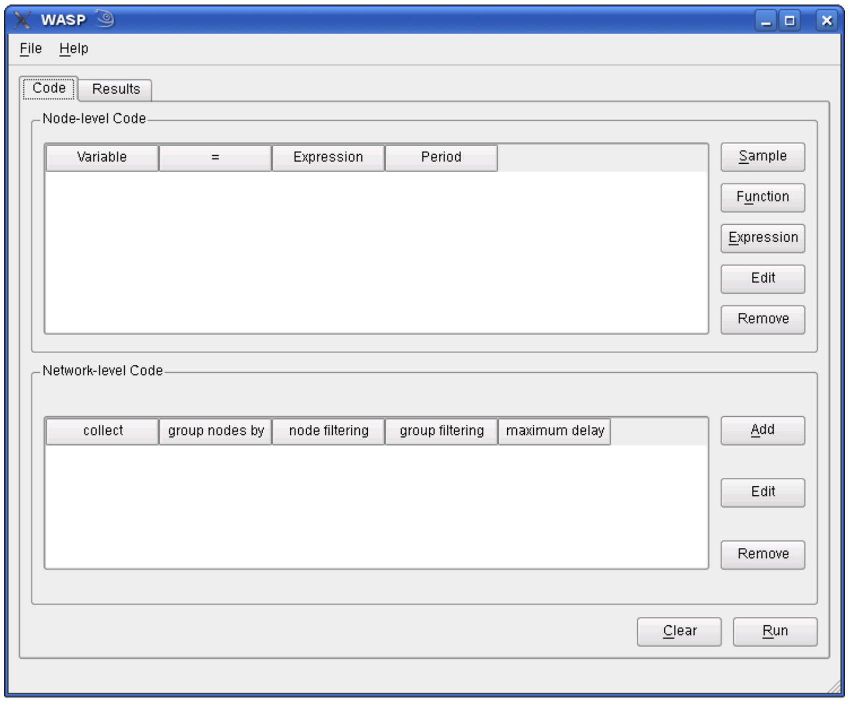


Figure 1: WASP programming and simulation environment.

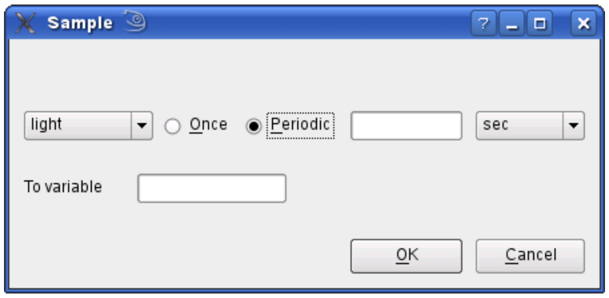


Figure 2: Sample dialog.

single sample or periodically-repeated samples should be taken by clicking the **Once** or **Periodic** radio button. Generally, single-point sampling is used for constant sensor readings. For example, location-related sensor data do not change in a stationary network, so you only need to read the sensor data once to determine the node's location. For data that constantly change, users may want to sample at an appropriate frequency to monitor the change. If periodic sampling is selected, you will be required to specify the interval between sampling events using a positive number with a time unit. Finally, enter a name to label the stored samples so you can refer to them in other instructions. Variable names are composed of alphanumeric characters and underscores (_). The first character of a variable name must be a letter or an underscore. For example, *light_level* and *_sound_5* are legal variable names, but *32light* is not. Variable names are case-sensitive. For example, *VAR* is a different variable from *Var*. Finally, click the **OK** button to add the instruction to the node-level code segment. You will see a new row representing the newly-created instruction added to the node-level instruction table.

Variables are locations for storing data. A variable can be viewed as an array. A variable created

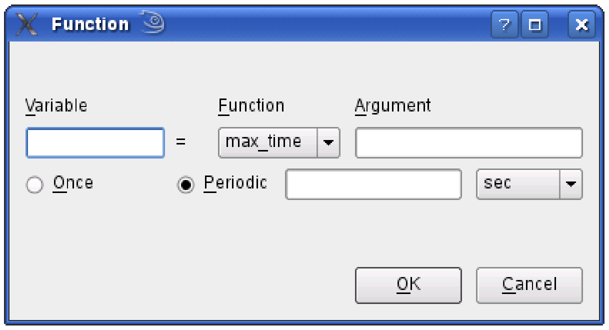


Figure 3: Function dialog.

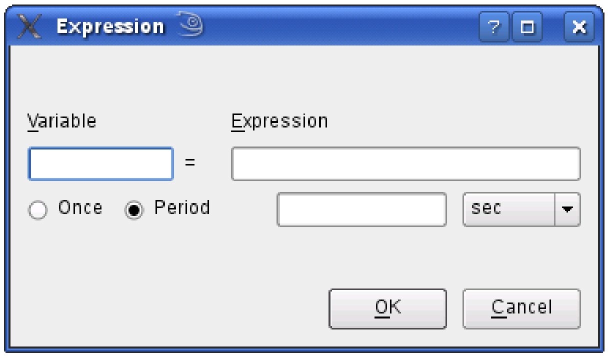
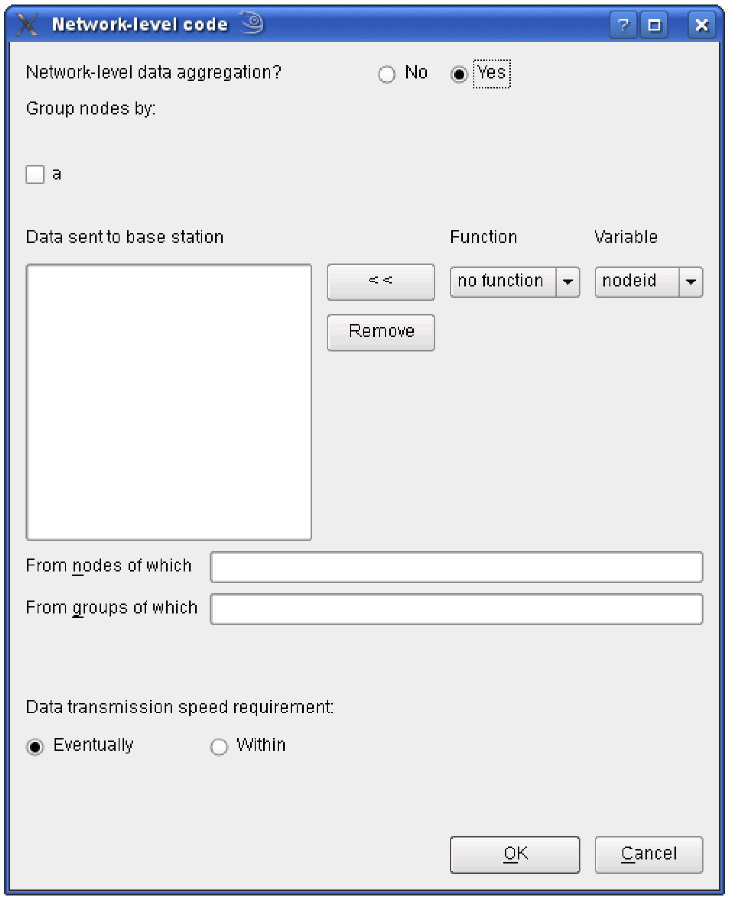


Figure 4: Expression dialog.

with single-point sampling is an array with only one data item in it. A variable created with periodic sampling is an array with multiple data items stored in order of time. For example, if variable *mylight* is created to sample light every 2 seconds, a light sensor reading will be appended to the *mylight* array every 2 seconds. Data items in the array can be referred to via indexing. Index 0 represents the most-recent sample, while index n represents the n most-recent data item. A sequence of data items can be referred to using two indices, indicating a range. For example, *mylight*[0:9] returns the most-recent 10 samples. A variable name alone stands for the most-recent sample in its array. For example, *mylight* means the most recent sample in array *mylight*; it is the same as *mylight*[0].

Already-defined variables can be processed to generate other variables using functions. **Functions** are portions of code that perform specific tasks. A function takes a fixed number (zero or more) of parameters and returns a value. All the functions have already been implemented in WASP. Clicking the **Function** button will open a dialog window as shown in Figure 3. First, enter a variable name to save the result, then select a function from the function list, and specify its argument in the following blank. The descriptions of functions provided in WASP are in Table 1. The argument you provide for the function should have the same type shown in the table. If the computation needs to repeat as its input updates periodically, "periodic" mode should be used by clicking the *Periodic* radio button. Computation period is specified in a similar way in this case. Otherwise, select the "once" mode. Finally, click the **OK** button to add the composed instruction to the node-level code table.

Data can also be processed using arithmetic expressions. Click the **Expression** button to open the expression dialog. First, enter a variable name for the result. Then in the expression box enter an arithmetic expression. Arithmetic expressions are composed of operands, binary operators, and parenthesis. A binary operator can be +, -, *, or /. An operand can be a variable or a number. For


Network-level code

Network-level data aggregation? No [✓ Yes]

Group nodes by:

☐ a

Data sent to base station

Function: no function

Variable: nodeid

From nodes of which

From groups of which

Data transmission speed requirement:

[✓ Eventually] Within

Figure 5: Network dialog.

example, $(mylight * 2 + 100)/10$ is an arithmetic expression. Numbers are integers and decimals. For example, 32 and -5 are integers; -2.5 and 3.22E-2 are decimals. Specify the computation mode in the way described for the function dialog. Finally, click the **OK** button to add the instruction.

All the node-level instructions are shown in the **Node-level Code** table. If you want to modify an instruction, you can select it by clicking on that row, and then click the **Edit** button. To remove an instruction, select it and click the **Remove** button.

2.2 Network-Level Code

The network-level code segment lets programmers view the entire sensor network as a whole and use network-level operations to extract the data they want from the sensor network.

Figure 6 demonstrates node filtering and data aggregation using a simple example. Each circle represents one node with nodeid labeled inside the circle. Each of the nodes has two variables: x and y. Their values are shown beside the circle. Assume we want to get the average of x among nodes that have the same value of y, and we only want to consider nodes with x values larger than 1. The node filtering condition, $x > 1$, selects nodes 1, 2, and 3. The grouping operation groups node 2 and 3 together (because they have the same value of y), and node 1 forms a group by itself. Then the average of x is computed within each group. The results are shown in Figure 6.

The dialog for composing network-level instructions is opened by clicking the **Add** button. The

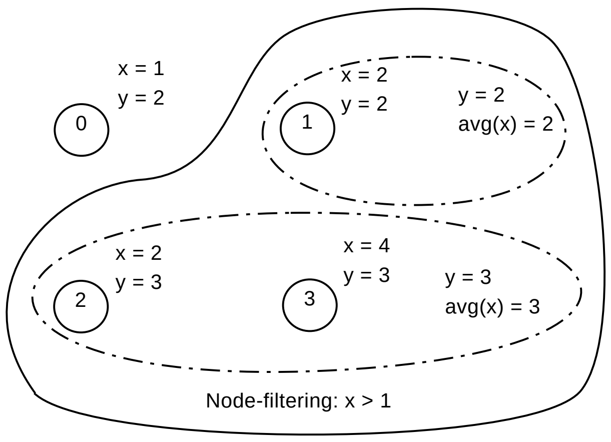


Figure 6: Filtering and aggregation.

Table 2: Network-Level Aggregation Functions

Function name	Arguments	Returned result
max	*attribute*	the maximum value of *attribute* in each group
min	*attribute*	the minimum value of *attribute* in each group
avg	*attribute*	the average of *attribute* in each group
sum	*attribute*	the sum of *attribute* in each group

dialog is shown in Figure 5. First, decide whether you need to aggregate data from multiple nodes using aggregation operations such as *sum* and *average*. If you only want to transmit the data from each node to the base station, you should click the **No** radio button. Otherwise, click the **Yes** radio button. Network-level aggregation functions apply to a set of nodes that are grouped together. If you check the **Yes** radio button, the dialog will show the options to group the nodes. Check the node variables that you want to group the nodes with. Nodes in the same group will have the same values for the selected variables. Table 2 contains descriptions of the network-level aggregation functions. Please be aware of the difference between the functions in Table 1 and those in Table 2. The node-level functions apply to temporal data items from the same node. On the contrary, the network-level functions aggregate data with the same time step across different nodes.

The box labeled “Data sent to base station” contains the data fields that are transmitted to the base station. You can add items to it by selecting from the function and variable list and clicking the << button, or remove an item by selecting it and clicking the **Remove** button.

If you want to collect data from certain nodes that meet some criteria, you can filter the nodes by entering a node-filtering-condition expression in the blank labeled **From nodes of which**. The expression is composed of a single comparison operation or multiple comparison operations connected with “and” and “or”. A comparison operation compares the values of two variables or compares the value of a variable with a number. It takes the form, *variable operator variable* or *variable operator* *number.* The operator can be > (greater than), < (less than), == (equal), >= (greater than or equal), <= (less than or equal), or <> (not equal). For example, "nodeid > 10 and mylight < 50" is a valid node-filtering-condition expression. If you do not need node-filtering, leave the blank empty.

If you want to collect aggregated data from certain groups that meet some criteria, you can filter the groups by entering a group-selection-condition expression in the blank labeled **From groups of which**. The group-selection-condition is composed of a single comparison operation or multiple comparison operations connected with "and" and "or". A comparison operation compares aggregated result with a number. For example, "avg(x) > 100" selects groups with average x larger than 100. If you do not need group-filtering, leave the blank empty.

The **Data transmission latency requirement** widgets let you specify the maximum latency between data sampling and its arrival at the base station. If you just want the data to be eventually collected at the base station and have no latency requirement, check the **Eventually** radio button. Otherwise, you should check the **Within** radio button and enter the maximum latency.

The network-level code segment can contain multiple instructions. An instruction will be executed with the smallest period of the variables involved in the computation. If you sample different types of sensor data with different periods, you may want to have one network-level instruction for each of them in order to collect the data at a frequency consistent with its sampling frequency.

3 Simulate and Check Results in the Results Tab

When you complete the program in the **Code** tab, you can simulate it by clicking the **Run** button. If it executes successfully, the **Results** tab will be brought to the front and data will be displayed there. The **Node samples** area shows the sample readings on each node. The **Base station** area shows the data received at the base station. To terminate the simulation, click the **Stop** button.

4 An Example Using WASP

In this section, we will use an example to demonstrate how to write a sensor network program in WASP.

Assume we want to deploy sensor nodes around a redwood tree to study its microclimate. The nodes are able to sense temperature and barometric pressure. The height level of a node can be computed from the pressure by *pressure/100 + 2.* Sensor nodes are static, so we only need to compute their height levels once. Write a program to sample the average light at each height level and transfer this value to the base station. Light will be sampled every 1 second on each node. Each node first averages its own samples within 4 second intervals. Then these data are averaged for nodes within the same height levels.

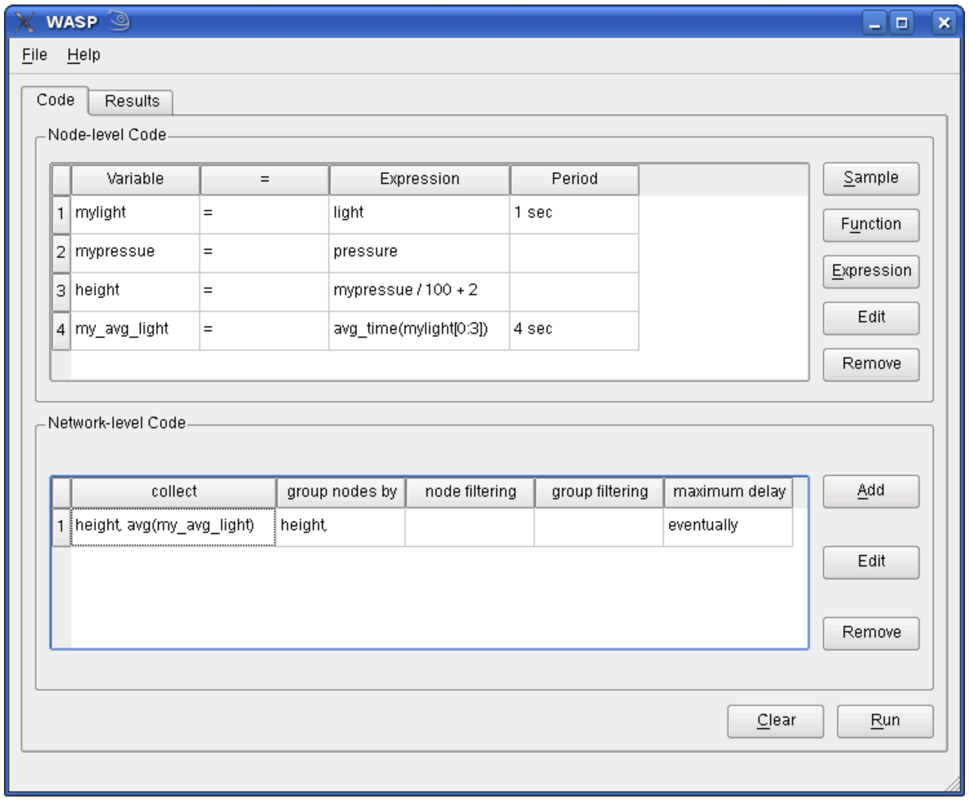
Figure 7 shows the completed program. We have created four node-level variables. We used the **Sample** dialog for creating the *mylight* and *mypressure* variables, the **Expression** dialog for creating the *height* variable, and the **Function** dialog for creating the *my_avg_light* variable. For the network-level code, we checked "Yes" for data aggregation and selected *height* as grouping variable.

Finally, click the **Run** button to simulation the program. Figure 8 shows the simulation results.

The **Node samples** area shows the sensor readings for each node at each time stamp. Data values in the "[ ]" are listed in the order of the *nodeid.* For example,

```
Time 0 s, network sample pressure: [489, 777, 766, 995]
```

indicates that at time 0, node 0 samples light level 489, node 1 samples light level 777, etc.



Node-level Code

	Variable	=	Expression	Period
1	mylight	=	light	1 sec
2	mypressue	=	pressure	
3	height	=	mypressue / 100 + 2	
4	my_avg_light	=	avg_time(mylight[0:3])	4 sec

Network-level Code

	collect	group nodes by	node filtering	group filtering	maximum delay
1	height, avg(my_avg_light)	height,			eventually

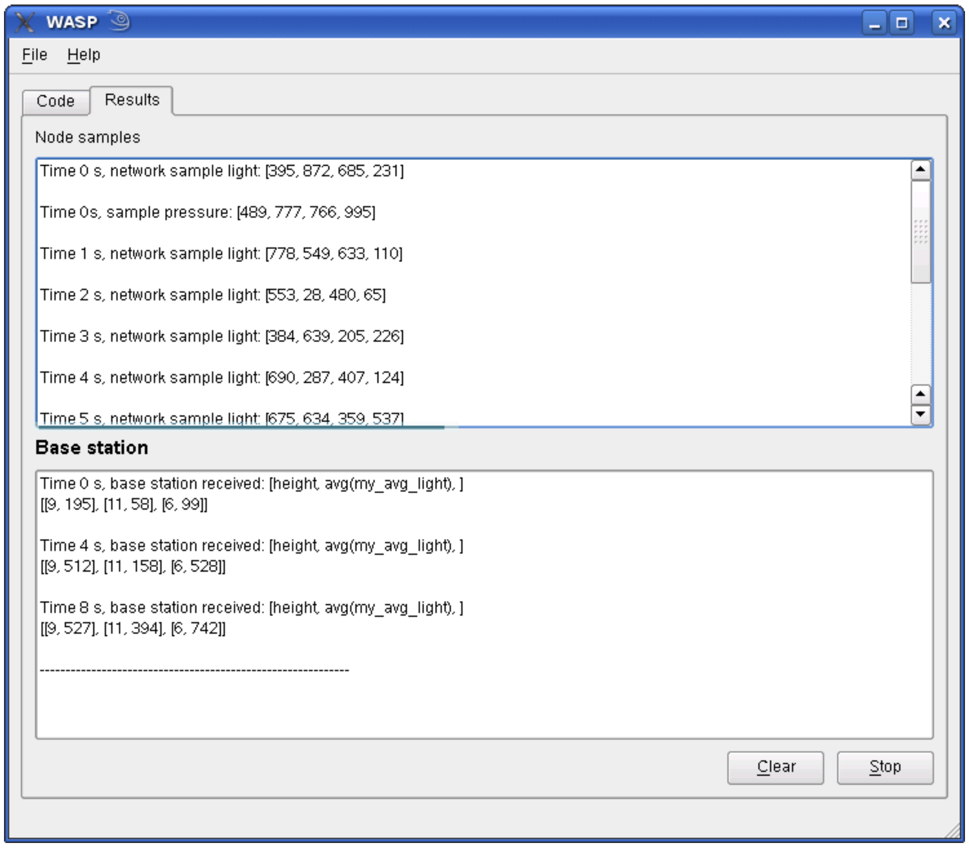
Figure 7: Example program.

The **Base station** area shows the data received at the base station. The data values in the inner square brackets follows the format in the preceding line. For example,

Time 8 s, base station received: [height, avg(my_avg_light),]

[[9, 527], [11, 394], [6, 742]]

indicates that at time 8 s, the base station received 3 groups of data. The first group has height value 9 and average light value 527, the second group has height value 11 and average light value 394, etc.



```
Time 0 s, network sample light: [395, 872, 685, 231]

Time 0s, sample pressure: [489, 777, 766, 995]

Time 1 s, network sample light: [778, 549, 633, 110]

Time 2 s, network sample light: [553, 28, 480, 65]

Time 3 s, network sample light: [384, 639, 205, 226]

Time 4 s, network sample light: [690, 287, 407, 124]

Time 5 s, network sample light: [675, 634, 359, 537]
```

```
Time 0 s, base station received: [height, avg(my_avg_light), ]
[[9, 195], [11, 58], [6, 99]]

Time 4 s, base station received: [height, avg(my_avg_light), ]
[[9, 512], [11, 158], [6, 528]]

Time 8 s, base station received: [height, avg(my_avg_light), ]
[[9, 527], [11, 394], [6, 742]]

--------------------------------------------------------------
```

Figure 8: Simulation output of example.