

# Lab: The Basic P4 NFP NIC

---

This lab introduces the concepts of P4 by implementing a basic NIC (Network Interface Card) and includes an introduction to some advanced features like load balancing and metering.

The rudimentary NIC is achieved by creating a program to match traffic entering the Network Flow Processor from the physical network port, and sending the traffic to its PCIe interface (i.e. the network device driver in the kernel of the host OS). Conversely traffic received at the PCIe interface is directed to the network interface.

We extend the rudimentary NIC by providing multiple PCIe interface destinations for traffic ingressing on the physical network port based on a simple load balancing implementation in P4. Furthermore meters are added for each independent PCIe destination, configured differently to provide an observable variation between the PCIe destinations.

The program is constructed in three phases. The first phase starts by familiarizing users to Programmer Studio with a simple drop program. The second phase extends this initial program to a basic wire, i.e. a rudimentary NIC. The third phase further modifies the program to implement the more advanced load balancing and metering features.

## Part 0: Download Source

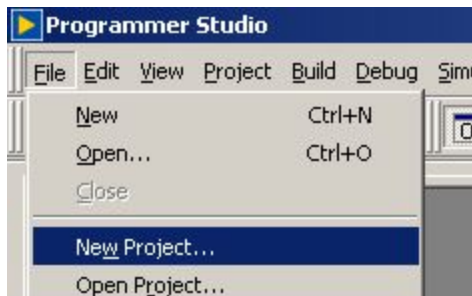
Before starting the lab, please download the source at the following URL:

[dxdd.netronome.com/labs.zip](http://dxdd.netronome.com/labs.zip)

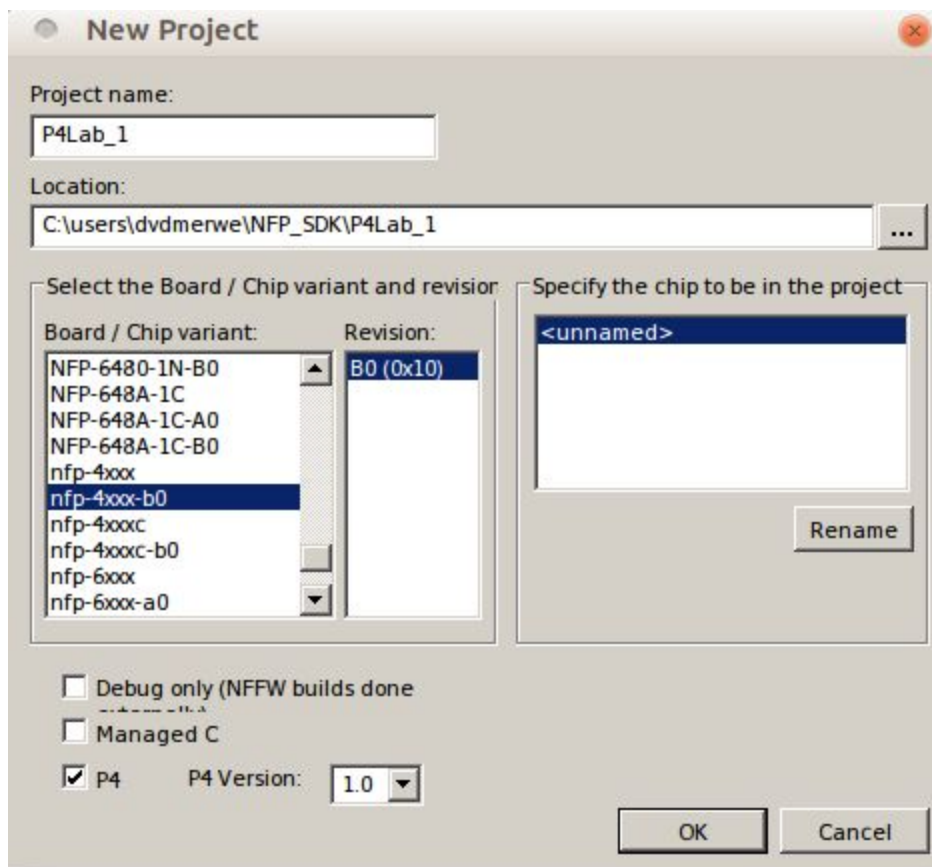
## Part 1: Programmer Studio and Packet Drop

The first step is to create a skeleton Programmer Studio (PS) project, and add the boilerplate drop program.

1. Open Programmer Studio by double clicking on the icon on the desktop.
2. Create a new project by choosing File -> New Project:

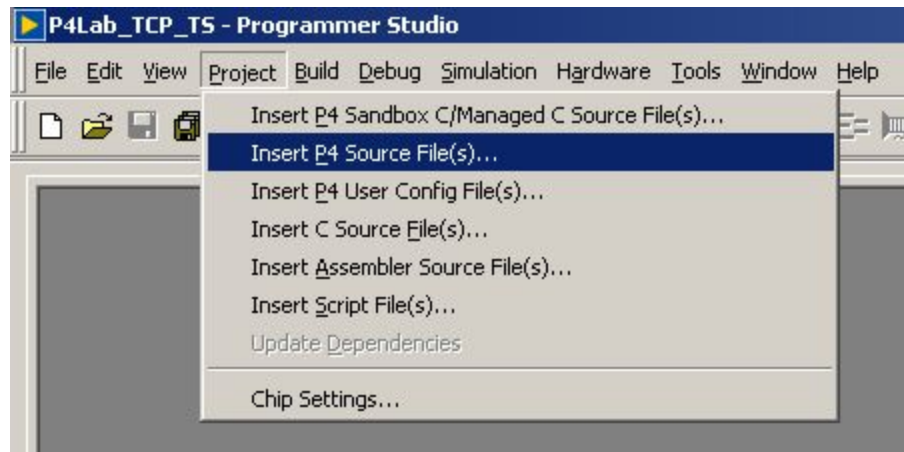


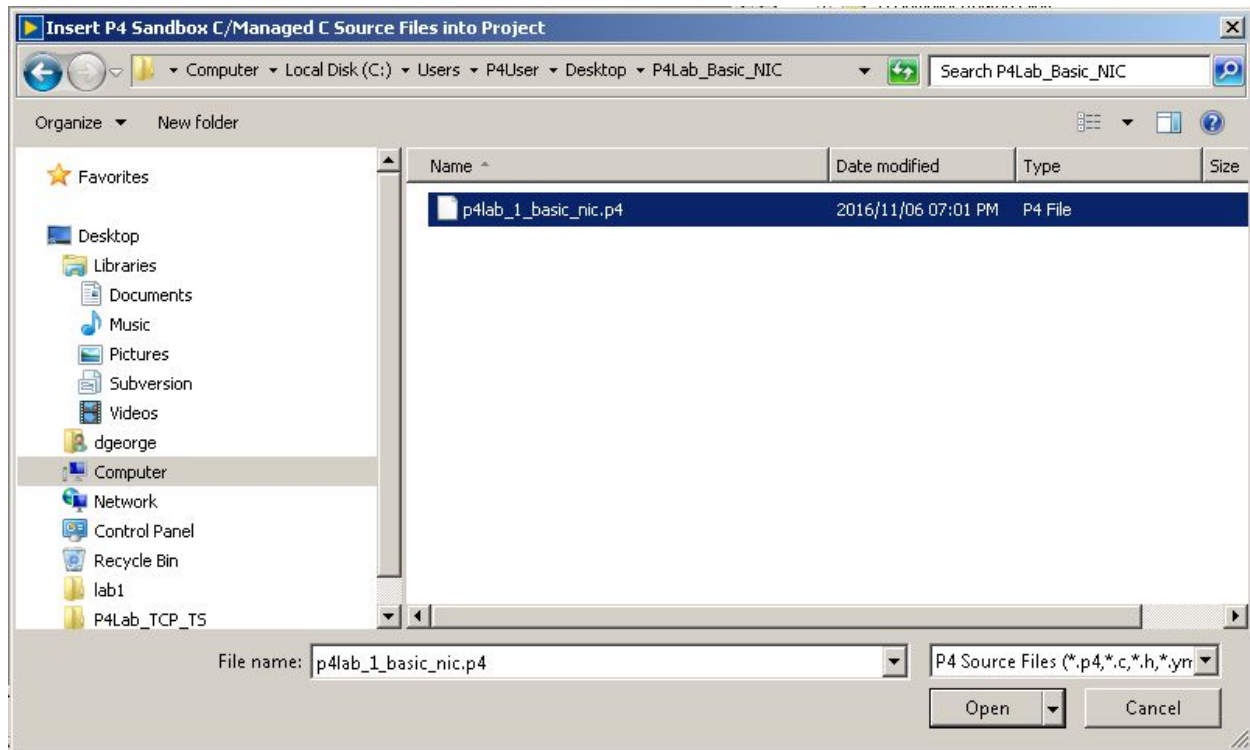
3. Select the '*nfp-4xxx-b0*' platform, set the project type to be P4 using the checkbox, set the P4 version to 1.0, and choose a suitable project name, for example "P4Lab\_1":



Note: Take care to select the P4 checkbox, as otherwise the project will not be using the P4 perspective.

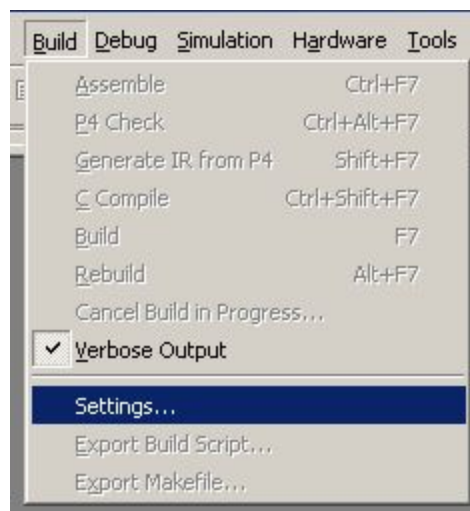
4. Add existing source which should be found in the P4Lab\_Basic\_NIC folder on the desktop (or whichever folder the demo resources were installed). To do this click *Project -> Insert P4 Source File(s)*. Now navigate to the *P4\_Lab\_Basic\_NIC* folder and select the file *p4lab\_1\_basic\_nic.p4*.



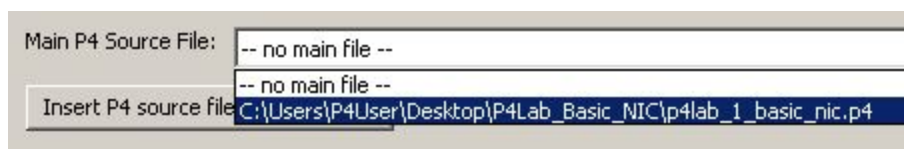


5. Select the P4 source as the main P4 source file.

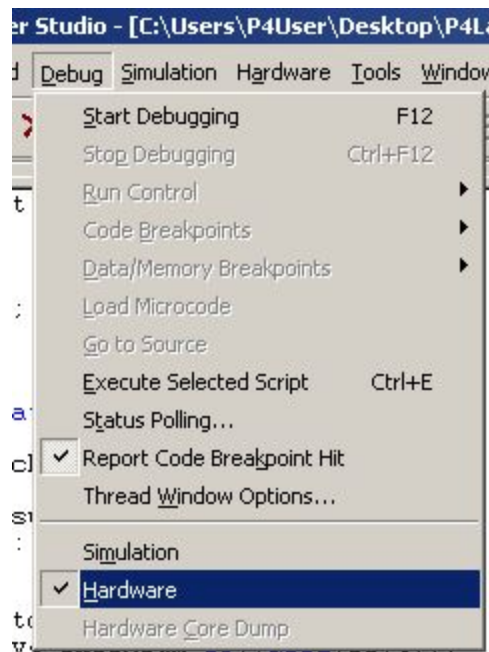
a. First click on *Build -> Settings...*



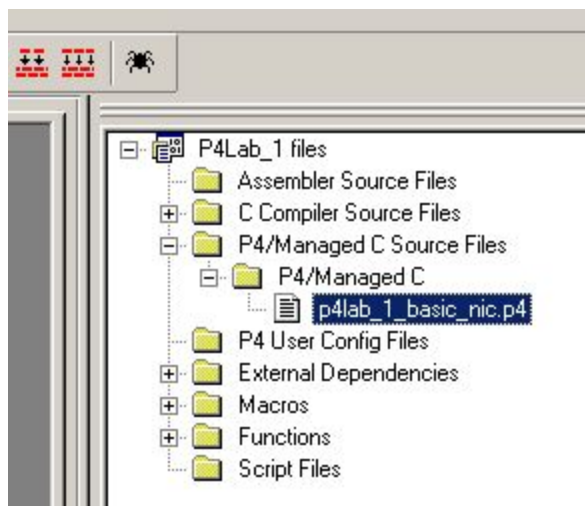
b. Select the P4 source from the *Main P4 Source File* drop down:



6. Configure the project for debugging using hardware:

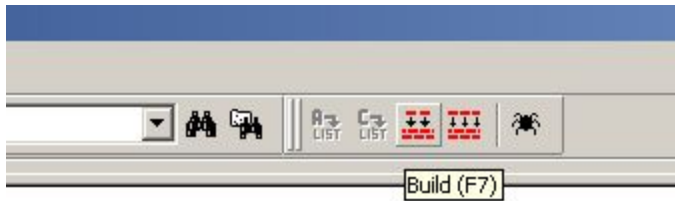


7. Open the source file by clicking *FileView -> P4/Managed C Source Files -> P4/Managed C -> p4lab\_1\_basic\_nic.p4*

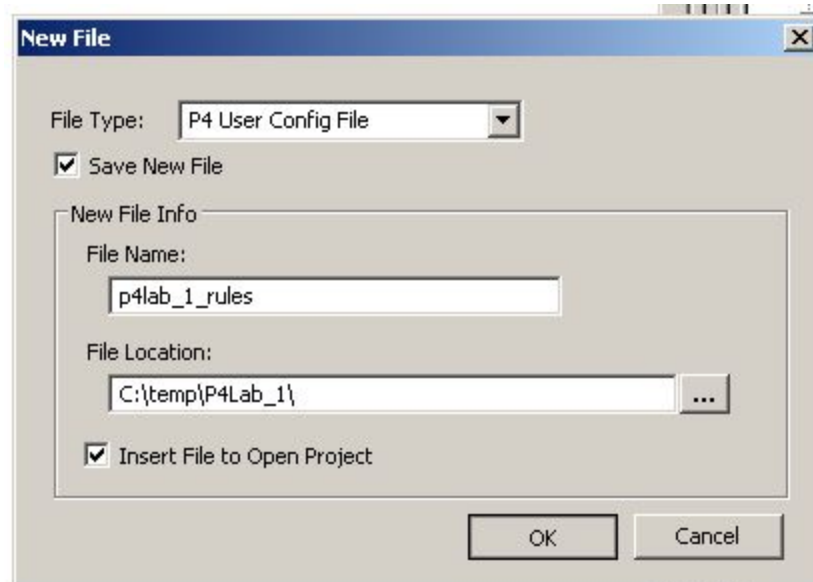


Review the source. You will see that only an Ethernet header is being parsed and a single action *act\_drop* is present for the table *tbl\_forward* which is applied in the *ingress* control flow.

8. Build by clicking the build button on the toolbar or by pressing F7:

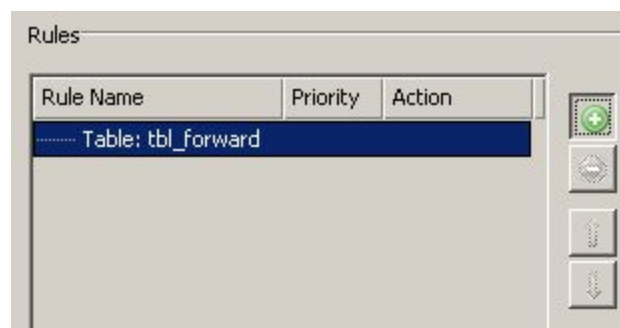


8. Create a P4 user configuration file by clicking on *File -> New* and then select *P4 User Config File* from the *File Type* drop down. Give your file a new and click the *Insert File to Open Project* checkbox:



9. We will now create a default rule and associate it with the drop action. The configuration file should now be open. If it is not click *FileView -> P4 User Config Files -> p4lab\_1\_rules.p4cfg* to open.

a. Click on the table *tbl\_forward* then click the add rule button:



b. Type a rule name, click the checkbox to make it the default rule, and select an action from the drop down list:

**Edit Rule**

Table: tbl\_forward

Rule Name: default

Default: ☒

Static Priority: ☐ Priority:

Match On:

Field	Match Type	-

Action: act\_drop

Action Parameters:

Parameter	Value

c. Now press Ctrl-S to save the file.

10. We will now add the rules file to the project.

a. Open the build settings dialog by clicking *Build -> Settings...*:

b. Select the P4 User Config file from the *User Config File* drop down:

User Config File (.p4cfg):

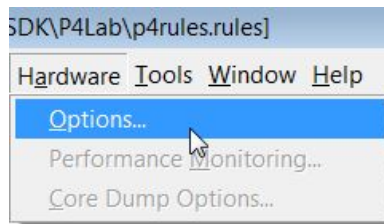
-- no user config file --

Insert P4 user config file: C:\temp\P4Lab\_1\p4lab\_1\_rules.p4cfg

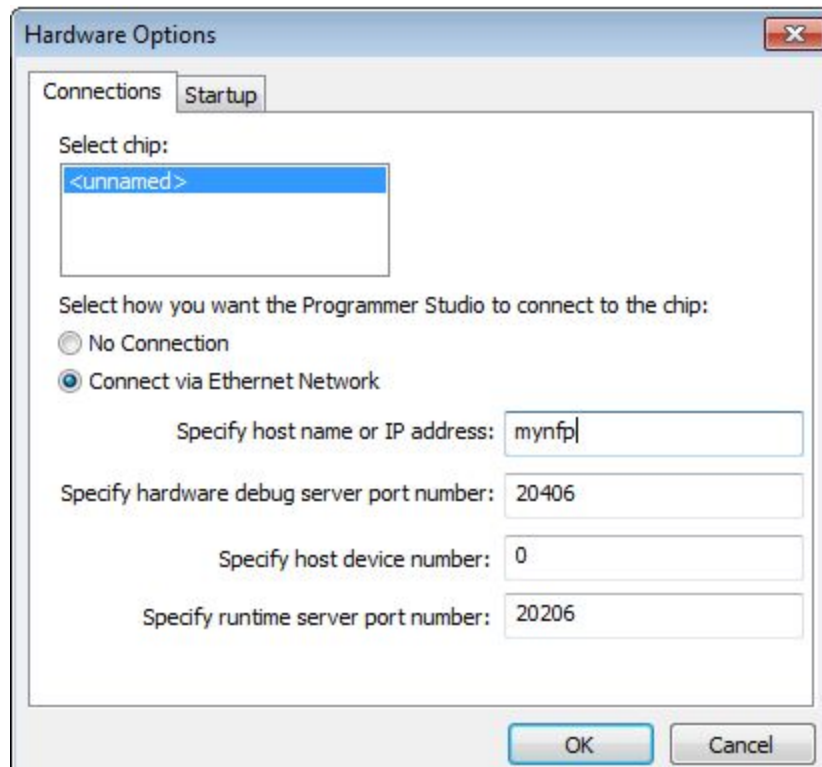
c. Click OK.

11. We now specify hardware debugging options.

a. Select the hardware debugging options dialog:



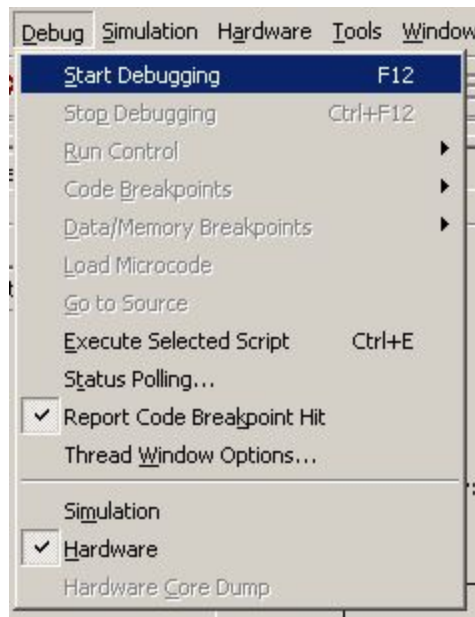
b. Specify the hardware debugging destination host name:



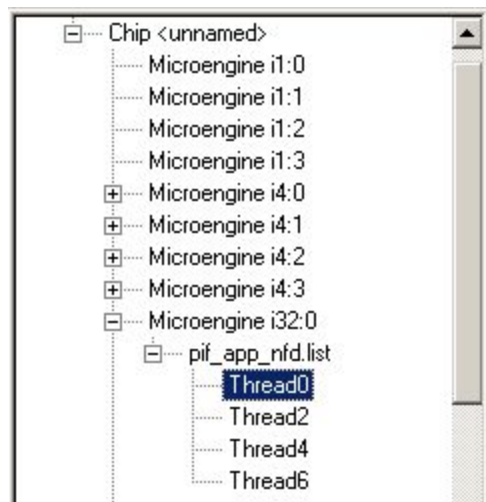
Enter your designated host IP address, hardware debug server port number, SmartNIC number (as the host device number) and runtime server port number.

12. Start debugging by pressing F12, or by using the menu:

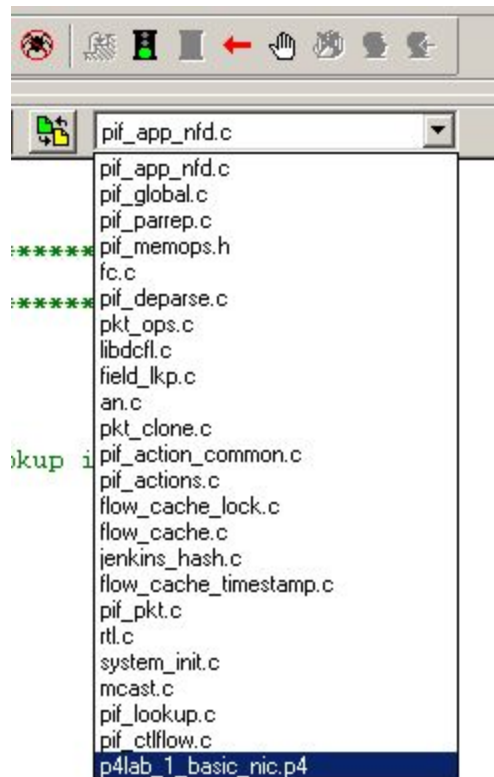




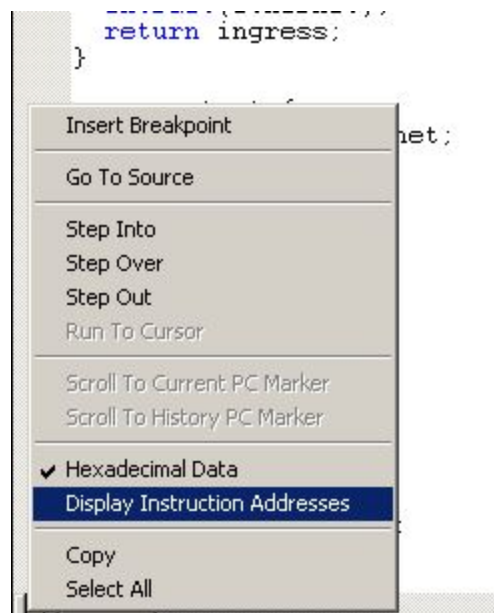
13. Select microengine 0 on island 32 in the thread view pane, and double click “Thread0”:



14. Select the “p4lab\_1\_basic\_nic.p4” file in the dropdown to debug at the P4 level:



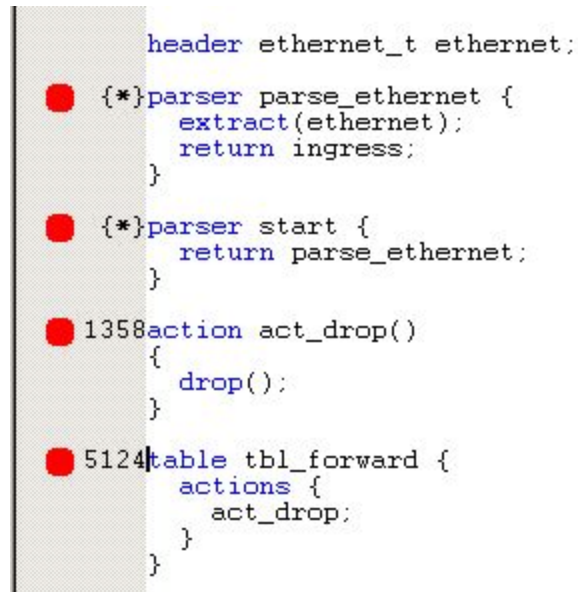
15. Enable displaying instruction addresses to simplify adding breakpoints (they can only be added at lines where instruction addresses are displayed) by right clicking in the left margin and enabling the option:



16. Add a few code breakpoints. Click on the source code line, then either click on the toolbar button, or press F9.



Breakpoints are indicated using red stop icons, as follows:



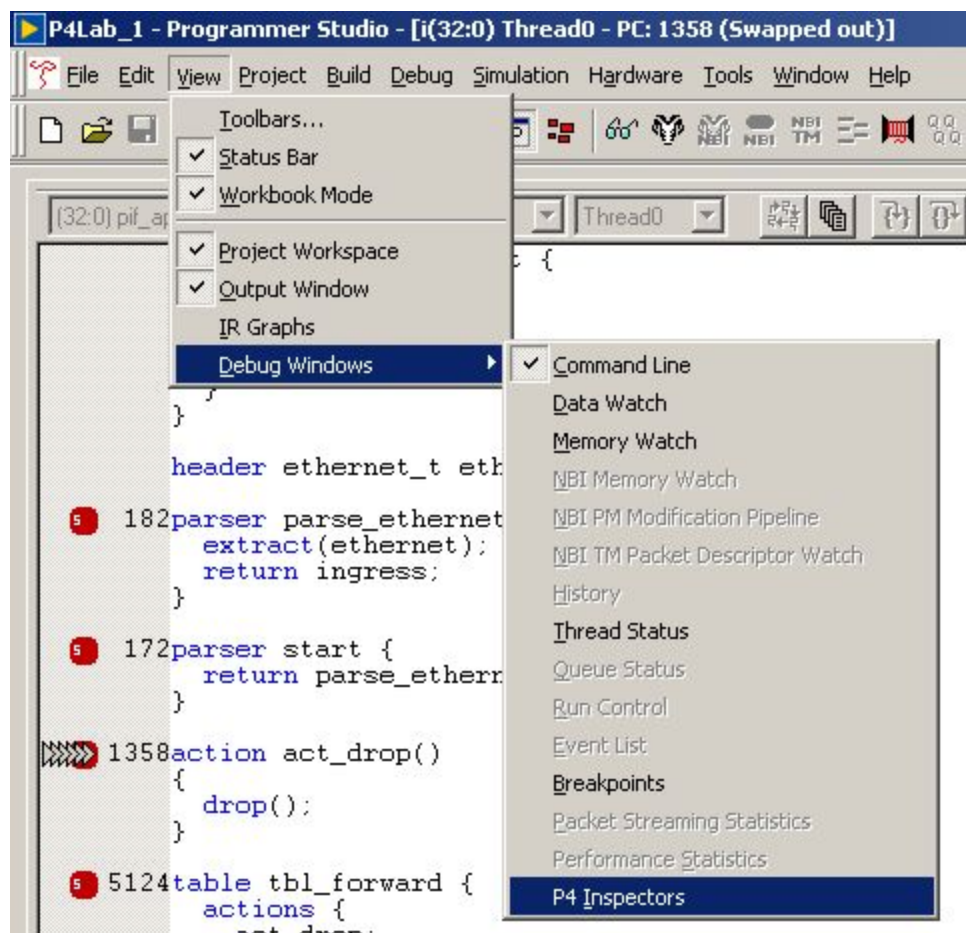
17. Run the program by pressing F5 or clicking on the toolbar icon:



18. Log in to your designated the traffic generation server as your designated user, i.e. “sdkuser<X>” with password “netronome”:



19. Open the P4 Inspector window by clicking *View -> Debug Windows -> P4 Inspectors*:

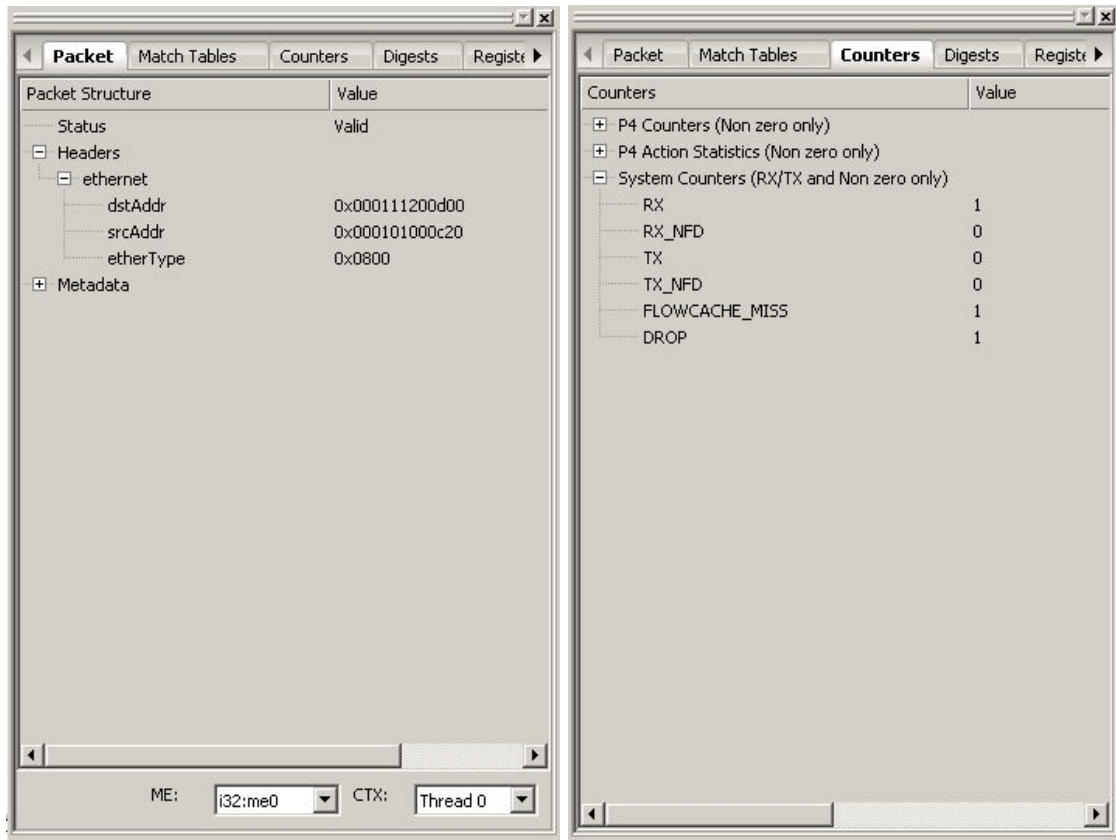


20. Inject a single packet by entering:

```
tcpreplay -L1 -i plp1 /root/udp_traffic.pcap
```

A breakpoint should now be hit. Continue execution by pressing F5 to observe the packet making its way through the P4 execution stages (parsing, matching, and actions).

21. The packet and counters can be observed using the P4 Inspectors to the left of the source code display:



Click Refresh to display updated packet information. The extracted packet headers are only displayed once the packet has been parsed. The standard packet metadata is always available.

Click Refresh to display updated counter values.

22. Stop debugging by pressing Ctrl-F12 or by clicking on the toolbar button:



## Part 2: Packet Wire

The next step involves modifying the program and rules to forward traffic from the network to the host, and the host to the network, i.e. in effect implementing the functionality of a rudimentary Network Interface Card (NIC).

1. Update the P4 program (“*p4lab\_1\_basic\_nic.p4*”) as follows.

a. Introduce an action to forward traffic to a specified port:

```
action act_drop() {
    drop();
}

action act_send_to_port(port) {
    modify_field(standard_metadata.egress_spec, port);
}
```

b. Amend the ingress table declaration to match the ingress port and invoke this action:

```
table tbl_forward {
    reads {
        standard_metadata.ingress_port : exact;
    }
    actions {
        act_send_to_port;
        act_drop;
    }
}
```

2. We will now update the rules to match the ingress port and invoke the forwarding action, to create a “wire” between the physical network interface on the NIC (“p0”) and the virtual interface on the host (“v0.0”), and another wire for the converse direction.

a. Rebuild the program (press F7) to ensure that the configuration editor is aware of the newly added table field.

b. Open your P4 configuration.

c. As before, add a rule to the table *tbl\_forward*. Enter a rule name. Enter “p0” as the match value. Select “act\_send\_to\_port” as the action and enter “v0.0” as the action parameter.

Edit Rule


Table: *tbl\_forward*

Rule Name:

Default: ☐

Static Priority: ☐ Priority

Match On:

Field	Match Type	value	-	
standard_metadata.ingress_port	exact	p0		

Action:

Action Parameters:

Parameter	Value	
port	v0.0	

d. Also add a host to network rule. Click on Add again, and enter a rule name. Enter “v0.0” as the match value. Select “act\_send\_to\_port” as the action and enter “p0” as the action parameter.

Edit Rule

Table: tbl\_forward

Rule Name:

Default: ☐

Static Priority: ☐ Priority

Match On:

Field	Match Type	value	-	
standard_metadata.ingress_port	exact	v0.0		

Action:

Action Parameters:

Parameter	Value	
port	p0	

e. Press Ctrl-S to save changes.

3. Start debugging again by pressing the menu button or by pressing F12.

4. Remove all breakpoints by clicking the remove all breakpoints icon on the toolbar:



5. Press the GO toolbar icon or press F5 to run firmware

6. Open two SSH connections to your designated server.

Use the one SSH session to observe packets being received by the NIC, using the following command:



```
tcpdump -l -n -i v0.0
```

Use the other SSH sessions to inject four packets, using the following command:

```
tcpreplay -L4 -i plp1 /root/udp_traffic.pcap
```

The following should be observed in the tcpdump window

```
19:33:30.913099 IP 5.6.7.8.63 > 1.2.3.4.63: UDP, length 18  
19:33:30.913113 IP 5.6.7.8.63 > 1.2.3.4.63: UDP, length 18  
19:33:30.913119 IP 5.6.7.8.63 > 1.2.3.4.63: UDP, length 18  
19:33:30.913123 IP 5.6.7.8.63 > 1.2.3.4.63: UDP, length 18
```

This concludes the rudimentary NIC functionality. The behavior of the NIC can be customized further by specifying rules that augment the program.

7. Stop debugging by pressing CTRL-F12 or by pressing the Stop Debugging toolbar icon

## Part 3: Metered Load Balancing

The next step involves modifying the program and rules again to implement a simple load balancing switch with meters applied to each destination port individually. The program will now forward packets received on the physical interface to 4 PCIe destinations load balanced on the ethernet address fields.

1. Update the P4 program ("*p4lab\_1\_basic\_nic.p4*") as follows.

a. Introduce the following packet metadata structure:

```
header ethernet_t ethernet;

header_type extended_metadata_t {
    fields {
        hash : 2;
        meter_color : 2;
    }
}

metadata extended_metadata_t extended_metadata;
```

b. Introduce the following field list and field list calculation structures required for the hash calculation:

```
metadata extended_metadata_t extended_metadata;

field_list lb_header_list {
    ethernet.srcAddr;
    ethernet.dstAddr;
}

field_list_calculation lb_hash {
    input {
        lb_header_list;
    }
    algorithm : crc16;
    output_width : 16;
}
```

c. Now add the action to insert the hash result to our metadata:

```
action act_calc_hash() {
    modify_field_with_hash_based_offset(extended_metadata.hash,
                                        0,
                                        lb_hash,
                                        4);
}
```

The action *act\_calc\_hash* will populate the field *extended\_metadata.hash* with a hash value from 0 to 3.

d. Add the 'hash' field to the 'tbl\_forward' table and add the table for the load balancing metadata:

```
table tbl_forward {
    reads {
        standard_metadata.ingress_port : exact;
        extended_metadata.hash : exact;
    }
    actions {
        act_send_to_port;
        act_drop;
    }
}

table tbl_lb_calc {
    actions {
        act_calc_hash;
    }
}
```

e. Now add the meter definition for each of our destinations:

```
@pragma netro meter_drop_red
meter vf_meters {
    type : bytes;
    result : extended_metadata.meter_color;
    instance_count : 4;
}
```

Note: the *netro meter\_drop\_red* pragma forces the packet to be dropped immediately in the pipeline when the meter returns red.

f. Introduce the additional metering action:

```
action act_send_to_port(port) {
    modify_field(standard_metadata.egress_spec, port);
}

action act_meter(meter_idx) {
    execute_meter(vf_meters,
                  meter_idx,
                  extended_metadata.meter_color);
    // Implicit drop when metered to red
}
```

g. Add the metering table:

```
table tbl_egress_meter {
    reads {
        standard_metadata.egress_port : exact;
    }
    actions {
        act_meter;
    }
}
```

h. Finally modify the ingress and egress control for the introduced tables. Note that metering is applied on egress:

```
control ingress {
    apply(tbl_lb_calc);
    apply(tbl_forward);
}

control egress {
    apply(tbl_egress_meter);
}
```

i. Save and rebuild the program

2. Now update the rules to calculate a hash value for each ingress packet, match the ingress port and the hash value and then invoke the forwarding action to a particular virtual interface on the host.

a. Open your P4 configuration file

b. Add the hash calculation rule by adding a new rule to the 'tbl\_lb\_calc' table.

Click on Add again and enter a rule name. Select "act\_calc\_hash" as the action. Select the "Default" tick box. This rule does not require any match or action parameters.

Edit Rule

Table: tbl\_lb\_calc

Rule Name:

Default: ☒

Static Priority: ☐ Priority

Match On:

Field	Match Type	-	-	

Action:

Action Parameters:

Parameter	Value	

c. Now remove the existing rules configured for the “tbl\_forward” table by selecting each rule and click on the Remove button.

d. Now add a new rule to the “tbl\_forward” table. Enter “p0” as the ingress port match value and “0” as the hash match value. Select “act\_send\_to\_port” as the action and enter “v0.0” as the action parameter.

Edit Rule

Table: tbl\_forward

Rule Name: fwd\_lb0

Default: ☐

Static Priority: ☐ Priority

Match On:

Field	Match Type	value	-	
standard_metadata.ingress_port	exact	p0		
extended_metadata.hash	exact	0		

Action: act\_send\_to\_port

Action Parameters:

Parameter	Value	
port	v0.0	

Continue to add three more rules, each matching on physical port “p0” and hash values “1”, “2” and “3” and set the action parameter as “v0.1”, “v0.2” and “v0.3” for each respective hash value.

3. Configure the egress metering.

- a. Add a new rule to the “tbl\_egress\_meter” table. Enter “v0.0” as the egress port match value. Select “act\_meter” as the action and enter “0” as the meter\_idx parameter.

Edit Rule

Table: tbl\_egress\_meter

Rule Name: meter\_vf0

Default: ☐

Static Priority: ☐ Priority

Match On:

Field	Match Type	value	-	
standard_metadata.egress_port	exact	v0.0		

Action: act\_meter

Action Parameters:

Parameter	Value	
meter_idx	0	

- b. Now select the “Meters” tab at the top of the P4 configuration window:

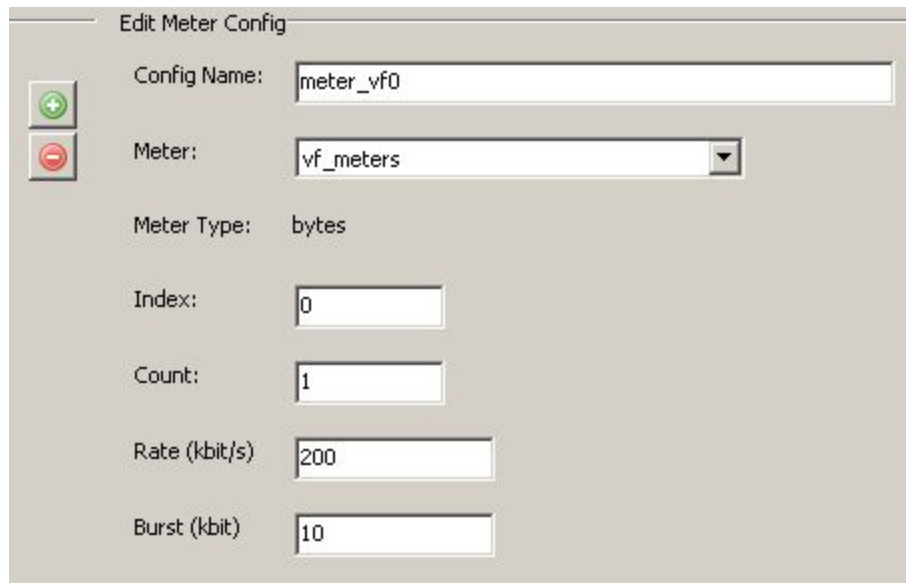
Rules | Multicast | Registers | Meters | Parser Value Sets

Meter Configs

Config Name	Meter	Idx	Cnt	
-------------	-------	-----	-----	--

- c. Click on the “Add” button to create a new meter configuration.

Enter a Config name and select “vf\_meters” in the meter field. Enter “0” for the Index and “1” for the Count fields. Enter “200” in the Rate and “10” in the Burst Size fields.



d. Press Ctrl-S to save changes.

4. Click 'Start Debugging' and then click 'GO' once loading is complete.

5. Open two SSH connections to your traffic generation server.

a. On one terminal run the *nload* tool to display traffic rates for all VF interfaces as follows:

```
nload -m v0.0 v0.1 v0.2 v0.3
```

Ensure your window is large enough to show the details of all port simultaneously.

b. Use the other SSH sessions to inject packets continuously using the *tcpreplay* command as follows:

```
tcpreplay --topspeed -q -K -l0 -i p1p1 \  
/root/udp_traffic.pcap
```

One can modify the meter parameters to observe the differences in behaviour. That concludes the load balance and meter functionality.

THE INFORMATION IN THIS DOCUMENT IS AT THIS TIME NOT A CONTRIBUTION TO P4.ORG.