**Lab 1: Understanding Logging system in ns3**

There are currently seven levels of log messages of increasing verbosity defined in the system.

• LOG\_ERROR — Log error messages (associated macro: NS\_LOG\_ERROR);

• LOG\_WARN — Log warning messages (associated macro: NS\_LOG\_WARN);

• LOG\_DEBUG — Log relatively rare, ad-hoc debugging messages (associated macro: NS\_LOG\_DEBUG);

• LOG\_INFO — Log informational messages about program progress (associated macro: NS\_LOG\_INFO);

• LOG\_FUNCTION — Log a message describing each function called (two associated macros:

NS\_LOG\_FUNCTION, used for member functions, and NS\_LOG\_FUNCTION\_NOARGS, used for static functions);

• LOG\_LOGIC – Log messages describing logical flow within a function (associated macro: NS\_LOG\_LOGIC);

• LOG\_ALL — Log everything mentioned above (no associated macro).

Note: LOG\_LEVEL\_TYPE = LOG\_TYPE + all the levels above it.

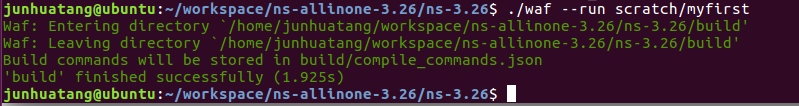
1. Disable logging by commenting out the following two lines in myfirst.cc

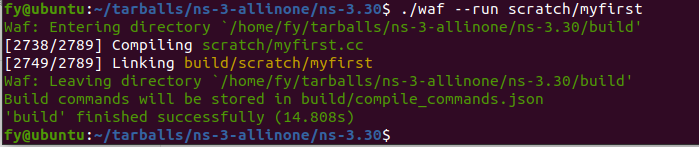
//LogComponentEnable ("UdpEchoClientApplication", LOG\_LEVEL\_INFO);

//LogComponentEnable ("UdpEchoServerApplication", LOG\_LEVEL\_INFO);

$./waf --run scratch/myfirst

No logging massage is printed on the screen.



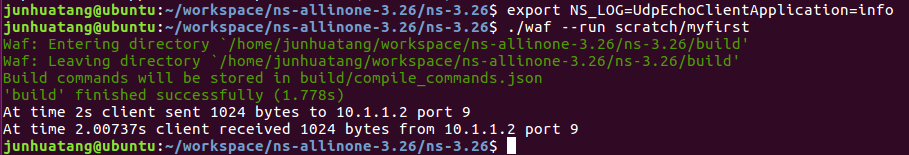


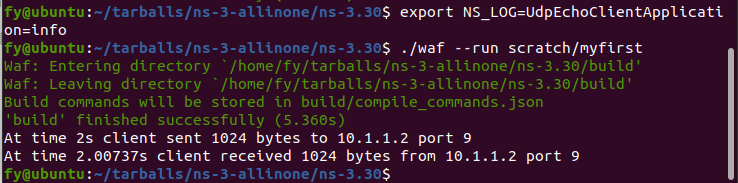
1. Control the logging by setting NS\_LOG environment variable

**2.1 Turn on UdpEchoClientApplication**

$export NS\_LOG=UdpEchoClientApplication=info

$./waf --run scratch/myfirst

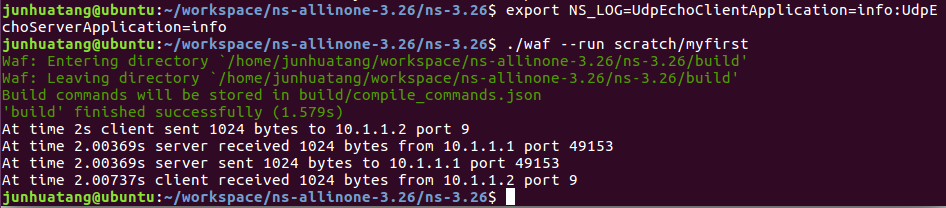


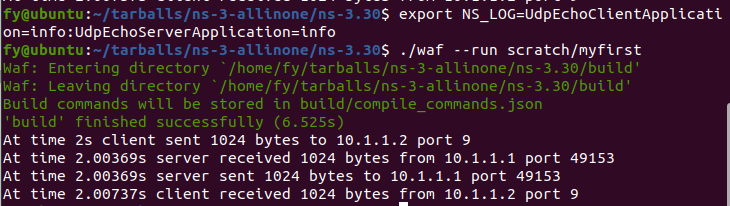


**2.2 Turn on both UdpEchoClientApplication and UdpEchoServerApplication**

$export NS\_LOG=UdpEchoClientApplication=info:UdpEchoServerApplication=info

$./waf --run scratch/myfirst

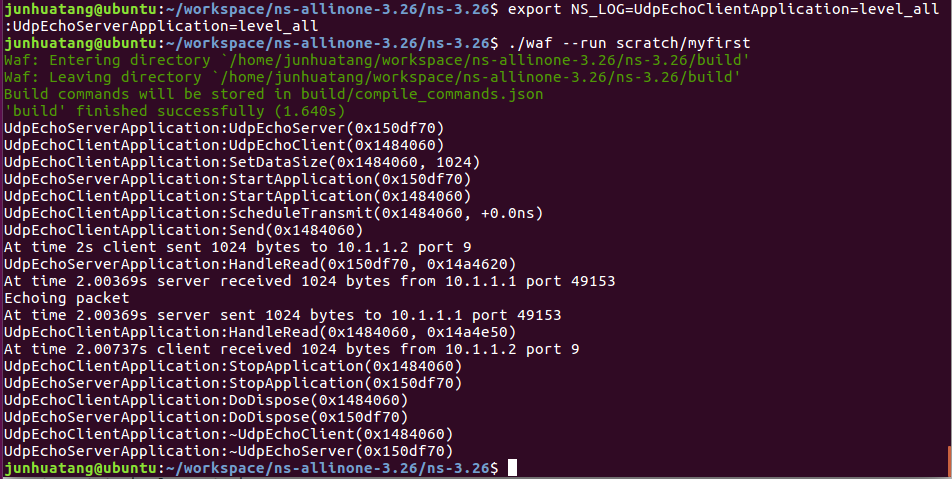


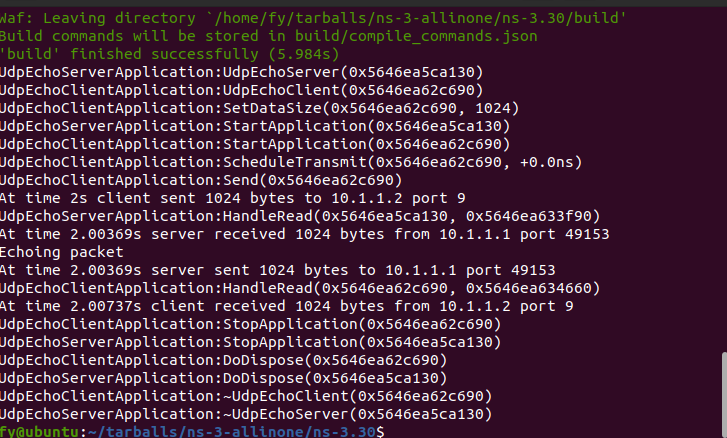


* 1. **Increase the logging level and get more information without changing the script and recompiling**

$export NS\_LOG=UdpEchoClientApplication=level\_all:UdpEchoServerApplication=level\_all

$./waf --run scratch/myfirst





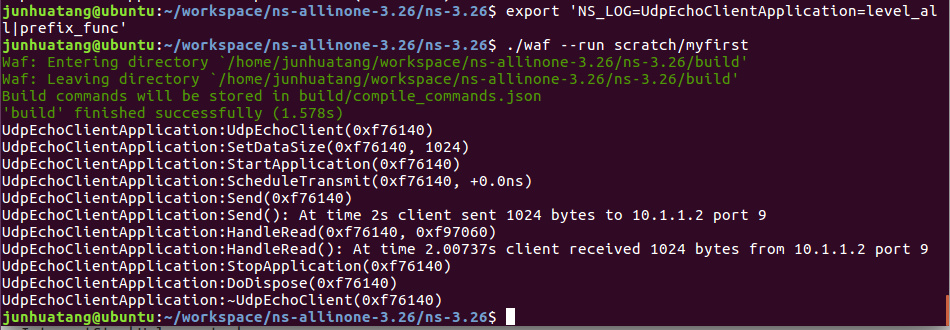
* 1. **Adding the function prefix to logging messages**

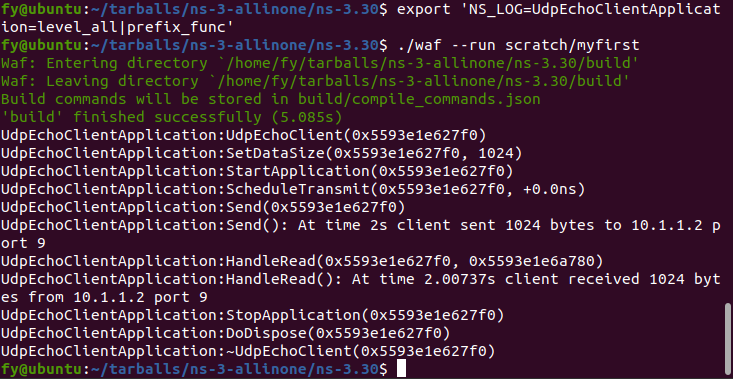
$ export 'NS\_LOG=UdpEchoClientApplication=level\_all|prefix\_func'

$./waf --run scratch/myfirst

Note: the quotes (‘ ‘ ) are required since the vertical bar we use to indicate an OR operation is also a Unix pipe connector.

Also note that only UdpEchoClientApplication is turned on.





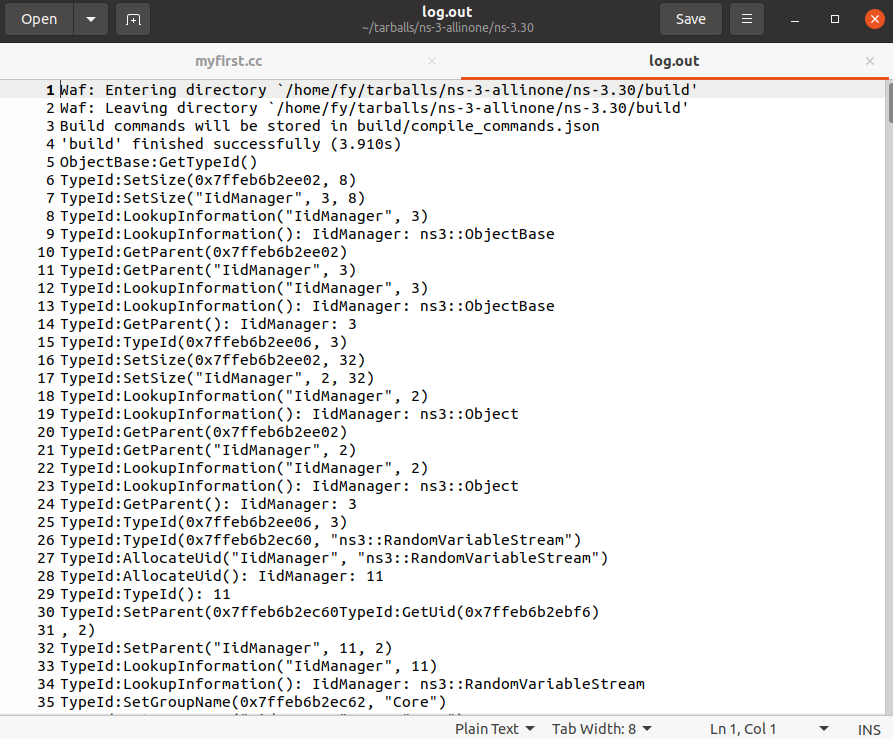
**2.5 Turn on all of the logging components in the system**

$export 'NS\_LOG=\*=level\_all|prefix\_func|prefix\_time'

$./waf --run scratch/myfirst > log.out 2>&1

（0 stdin，1 stdout，2 stderr，2>&1表示标准错误重定向到标准输出）

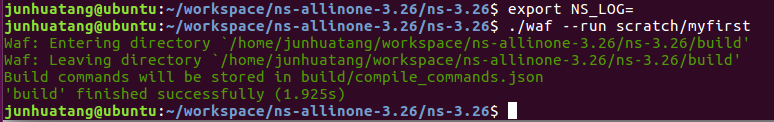
Open the file log.out in the ns-3.28 directory and go through the contents

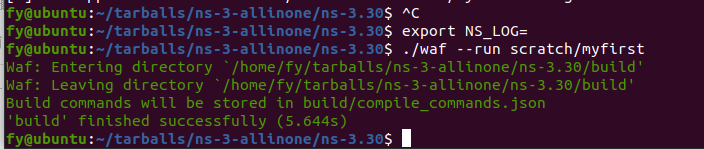


**2.6 Clear NS\_LOG settings**

$ export NS\_LOG=

$./waf --run scratch/myfirst





1. **Adding logging to your code**

We have defined a logging component in myfirst.cc by:

NS\_LOG\_COMPONENT\_DEFINE ("FirstScriptExample");

Add the following lines

NS\_LOG\_UNCOND ("Welcome! Ns3 Simulator! ");

NS\_LOG\_INFO ("Creating Topology");

right before the lines,

NodeContainer nodes;

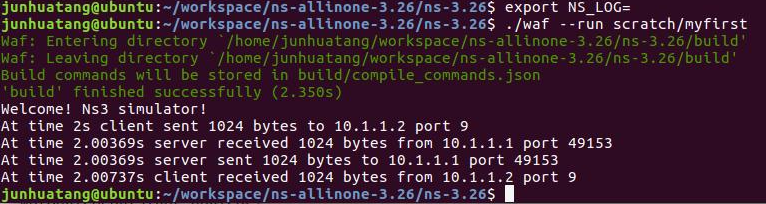
nodes.Create (2);

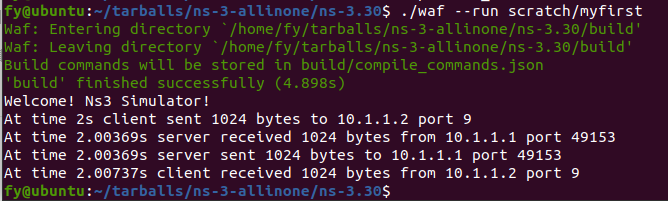
Save the file myfirst.cc, compile and run the script:

$./waf

$export NS\_LOG=

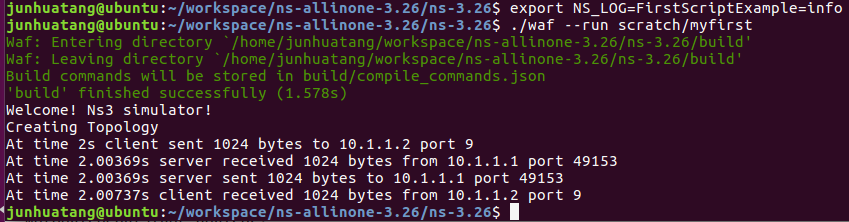
$./waf --run scratch/myfirst

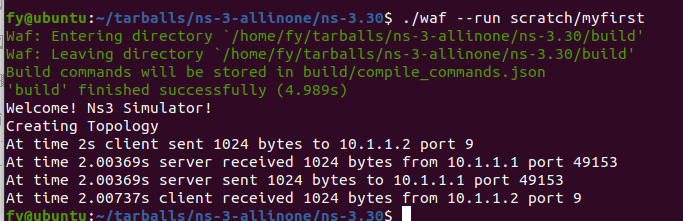




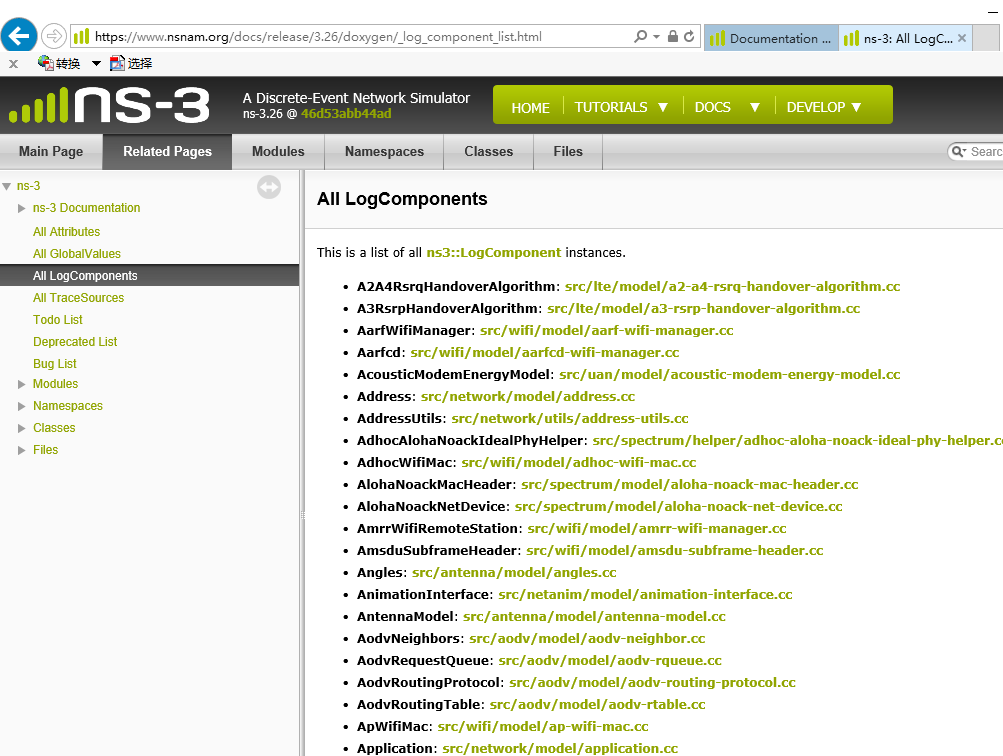
$export NS\_LOG=FirstScriptExample=info

$./waf --run scratch/myfirst





Now go to <https://www.nsnam.org/docs/release/3.28/doxygen/_log_component_list.html> and go through All LogComponents



1. **Using Command Line Arguments**

Go ahead and change the scratch/myfirst.cc script to start with the following code,

**int**

main (**int** argc, **char** \*argv[])

{

**uint32\_t** nPackets = 1;

CommandLine cmd;

cmd.AddValue("nPackets", "Number of packets to echo", nPackets);

cmd.Parse (argc, argv);

...

Scroll down to the point in the script where we set the MaxPackets Attribute and change it so that it is set to the variable nPackets instead of the constant 1 as is shown below.

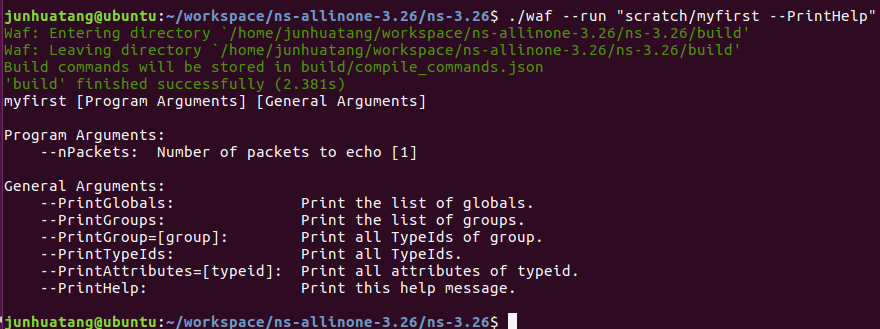
echoClient.SetAttribute ("MaxPackets", UintegerValue (nPackets));

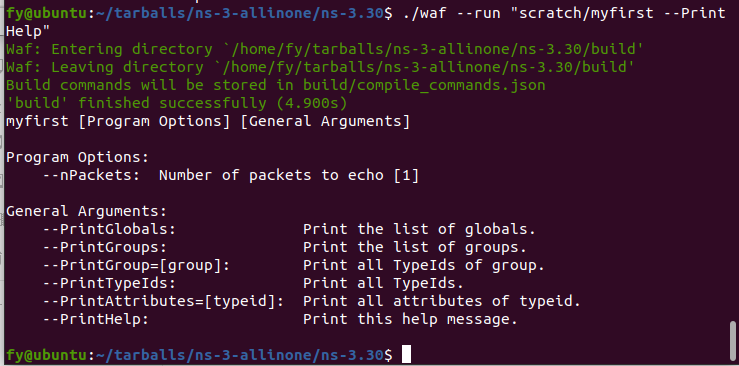
Now if you run the script and provide the --PrintHelp argument, you should see your new User Argument listed in the help display.

Try,

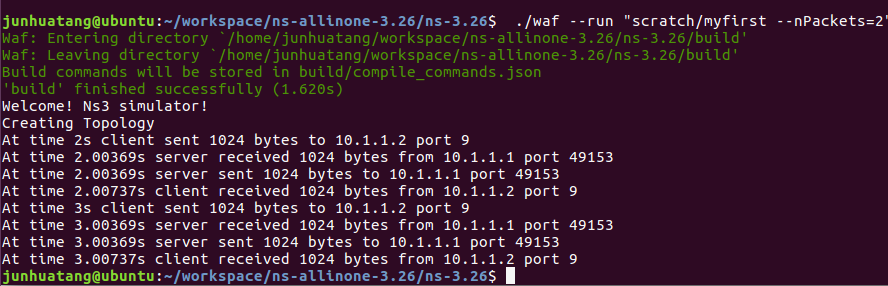
$./waf

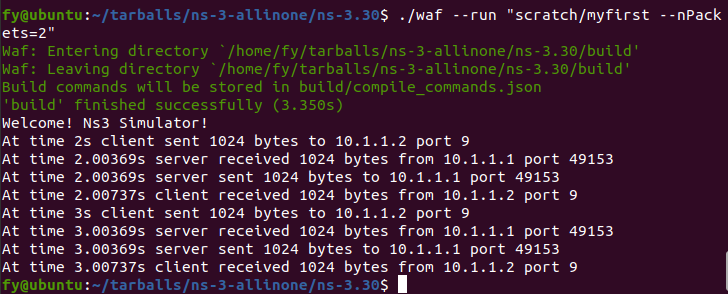
$ ./waf --run "scratch/myfirst --PrintHelp"





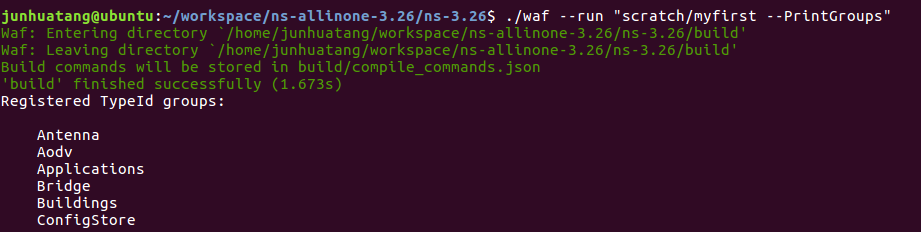
$ ./waf --run "scratch/myfirst --nPackets=2"





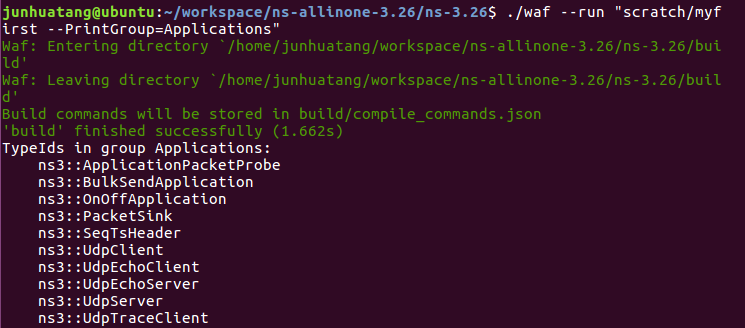
Find out more on general arguments:

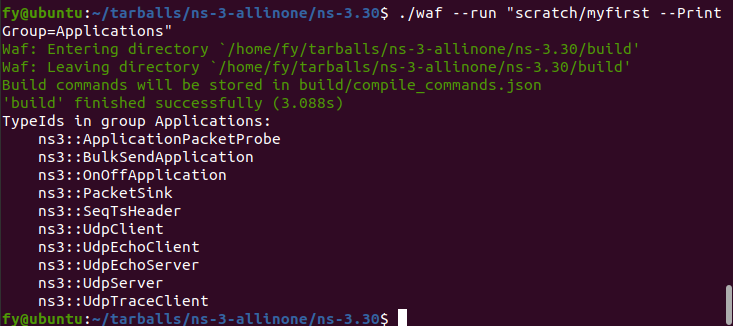
$ ./waf --run "scratch/myfirst --PrintGroups"



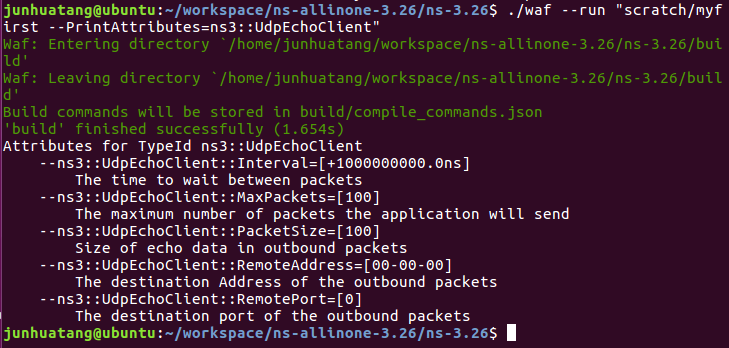


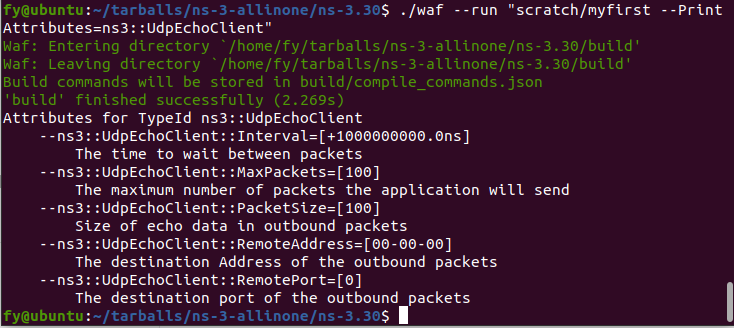
$ ./waf --run "scratch/myfirst --PrintGroup=Applications"





$ ./waf --run "scratch/myfirst --PrintAttributes=ns3::UdpEchoClient"





1. **Using the tracing system**

**5.1 ASCII tracing**

Let’s just jump right in and add some ASCII tracing output to our scratch/myfirst.cc script. Right before the call to Simulator::Run (), add the following lines of code:

AsciiTraceHelper ascii;

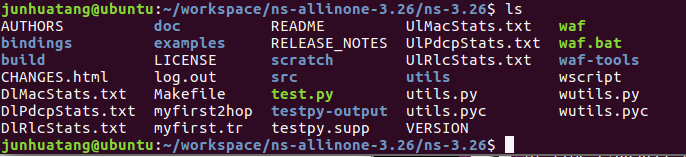
pointToPoint.EnableAsciiAll (ascii.CreateFileStream ("myfirst.tr"));

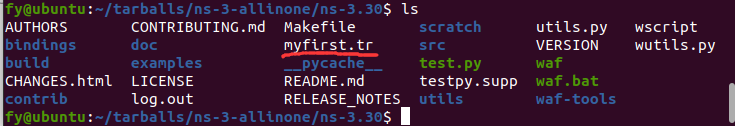
The outside call, to EnableAsciiAll(), tells the helper that you want to enable ASCII tracing on all point-to-point devices in your simulation; and you want the (provided) trace sinks to write out information about packet movement in ASCII format.

You can now build the script and run it from the command line:

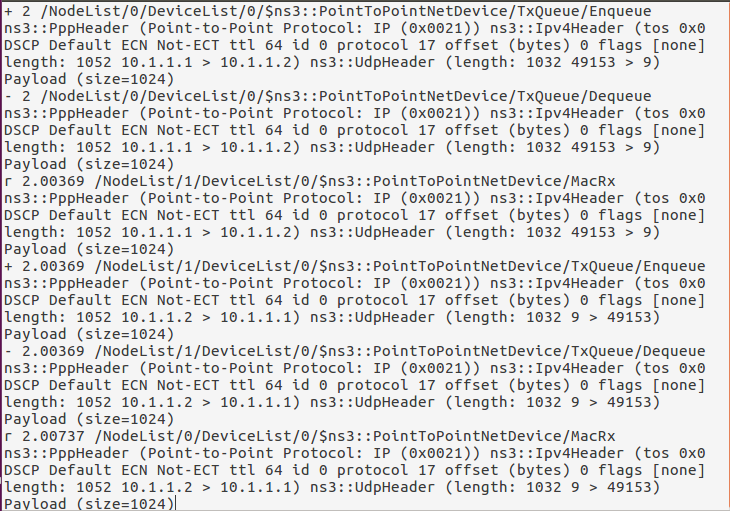
$ ./waf --run scratch/myfirst

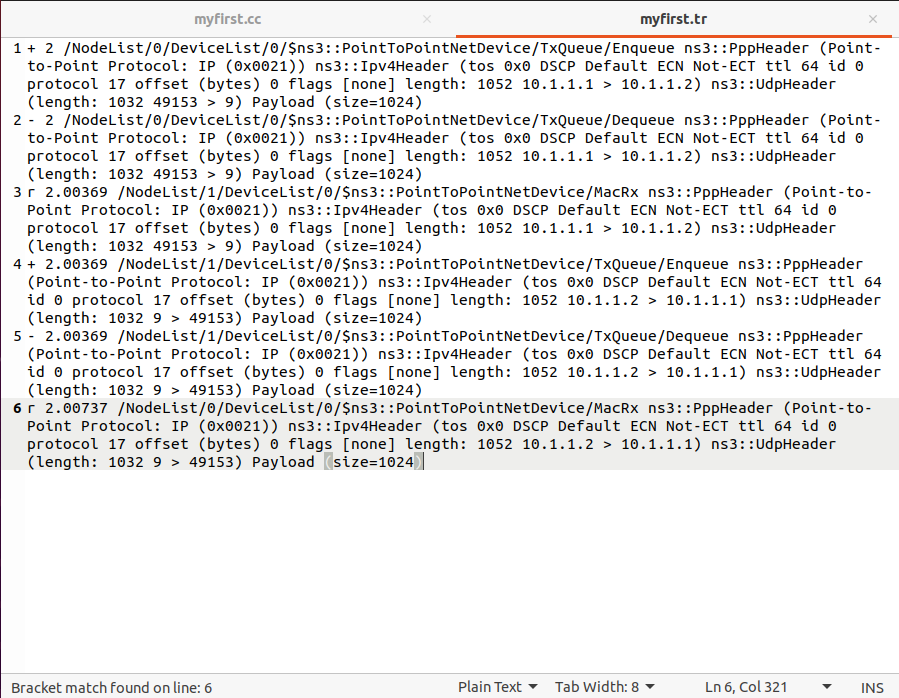
An ASCII trace file myfirst.tr will be created in the directory of ns-3.26, open the file in your favorite editor.





myfirst.tr





Each line in the file corresponds to a trace event. In this case we are tracing events on the transmit queue present in every point-to-point net device in the simulation. The transmit queue is a queue through which every packet destined for a point-to-point channel must pass. Note that each line in the trace file begins with a lone character (has a space after it). This character will have the following meaning:

+: An enqueue operation occurred on the device queue;

-: A dequeue operation occurred on the device queue;

d: A packet was dropped, typically because the queue was full;

r: A packet was received by the net device.

**5.2 PCAP Tracing**

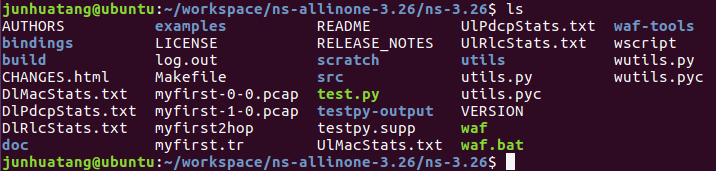
Go ahead and insert this line of code after the ASCII tracing code we just added to scratch/myfirst.cc.

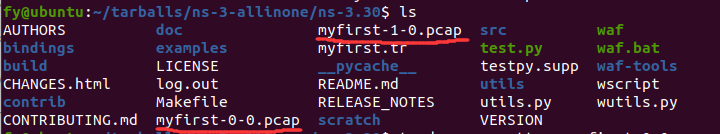
pointToPoint.EnablePcapAll ("myfirst");

$./waf

$ ./waf --run scratch/myfirst

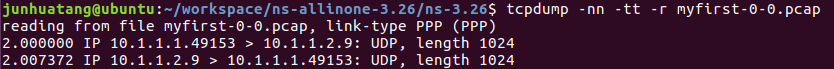
In our example script, we will eventually see files named “myfirst-0-0.pcap” and “myfirst-1-0.pcap” which are the pcap traces for node 0-device 0 and node 1-device 0, respectively, in the directory of ns-3.26.

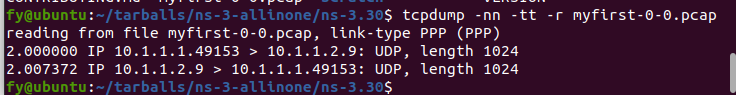




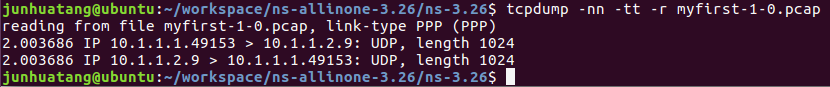
Use tcpdump to look at the pcap files.ls

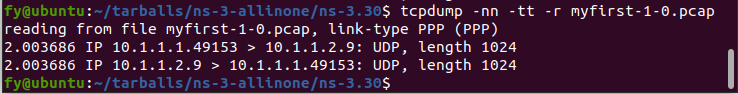
$ tcpdump -nn -tt -r myfirst-0-0.pcap (trace file for node 0 device 0)





$ tcpdump -nn -tt -r myfirst-1-0.pcap (trace file for node 1 device 0)





**Lab1 turn in:**

lab1-report.doc (lab1-report.pdf)

1. Draw a time line diagram to show the packet exchanges illustrated by the following two pcap files. Explain how the timestamps are obtained numerically.

