<http://ns2tutor.weebly.com/throughput-calculation-using-ns2.html>

* [*Home*](http://ns2tutor.weebly.com/)
* [*NS2 in Windows*](http://ns2tutor.weebly.com/ns2-in-windows.html)
* [*Simulators*](http://ns2tutor.weebly.com/simulators.html)
* [*Wired Simulation*](http://ns2tutor.weebly.com/wired-simulation.html)
* [*Wireless Simulation*](http://ns2tutor.weebly.com/wireless-simulation.html)
* [*Useful Links*](http://ns2tutor.weebly.com/useful-links.html)
* [*Contact Me*](http://ns2tutor.weebly.com/contact-me.html)

|  |  |  |  |
| --- | --- | --- | --- |
| [*Network Simulator - ns2*](http://ns2tutor.weebly.com/) | |  |  | | --- | --- | |  |  | |

**THROUGHPUT CALCULATION USING NS2**  
  
  
**Throughput Description:**  
  
            In computer technology, throughput is the amount of work that a computer can do in a given time period. Historically, throughput has been a measure of the comparative effectiveness of large commercial computers that run many programs concurrently. An early throughput measure was the number of batch jobs completed in a day. More recent measures assume a more complicated mixture of work or focus on some particular aspect of computer operation. While "cost per million instructions per second (MIPS)" provides a basis for comparing the cost of raw computing over time or by manufacturer, throughput theoretically tells you how much useful work the MIPS are producing. Another measure of computer productivity is performance, the speed with which one or a set of batch programs run with a certain workload or how many interactive user requests are being handled with what responsiveness. The amount of time between a single interactive user requests being entered and receiving the application's response is known as response time.  
  
              
  
            In communication networks, such as Ethernet or packet radio, network throughput is the average of successful message delivery over a communication channel. This data may be delivered over a physical or logical link, or pass through a certain network node. The throughput is usually measured in bits per second or data packets per time slot. It can be calculated as maximum throughout, maximum theoretical throughput, maximum sustained throughput, peak throughput, normalized throughput and so on.  
  
            In computer networks, goodput is the application level throughput, i.e. the number of useful bits per unit of time forwarded by the network from a certain source address to a certain destination, excluding protocol overhead, and excluding retransmitted data packets. For example, if a file is transferred, the goodput that the user experiences corresponds to the file size in bits divided by the file transfer time. The goodput is generally lower than the throughput (the gross bit rate that is transferred physically), which generally is lower than network access connection speed (the channel capacity or bandwidth).  
  
            Examples of factors that cause lower goodput than throughput are:  
  
*Protocol overhead*: Typically, transport layer, network layer and sometimes datalink layer protocol overhead is included in the throughput, but is excluded from the goodput.  
  
*Transport layer flow control and congestion avoidance*: For example, TCP slow start, may cause a lower goodput than the maximum throughput.  
  
*Retransmission* of lost or corrupt packets due to transport layer automatic repeat request (ARQ), caused by bit errors or packet dropping in congested switches and routers, is included in the datalink layer or network layer throughput but not in the goodput.  
  
  
  
**Maximum Throughput:**  
  
            Users of telecommunications devices, systems designers, and researchers into communication theory are often interested in knowing the expected performance of a system. From a user perspective, this is often phrased as either "which device will get my data there most effectively for my needs?", or "which device will deliver the most data per unit cost?". Systems designers are often interested in selecting the most effective architecture or design constraints for a system, which drive its final performance. In most cases, the benchmark of what a system is capable of, or its 'maximum performance' is what the user or designer is interested in. When examining throughput, the term 'Maximum Throughput' is frequently used. Maximum throughput is essentially synonymous to digital bandwidth capacity. Four different values have meaning in the context of "maximum throughput", used in comparing the 'upper limit' conceptual performance of multiple systems.   
  
            They are 'maximum theoretical throughput', 'Maximum Achievable throughput', and 'peak measured throughput' and 'maximum sustained throughput'. These represent different quantities and care must be taken that the same definitions are used when comparing different 'maximum throughput' values. Comparing throughput values is also dependent on each bit carrying the same amount of information. Data compression can significantly skew throughput calculations, including generating values greater than 100%. If the communication is mediated by several links in series with different bit rates, the maximum throughput of the overall link is lower than or equal to the lowest bit rate. The lowest value link in the series is referred to as the bottleneck.  
  
**Peak Measured Throughput:**  
  
            The above value is theoretical or calculated values. Peak measured throughput is throughput measured by a real, implemented system, or a simulated system. The value is the throughput measured over a short period of time; mathematically, this is the limit taken with respect to throughput as time approaches zero. This term is synonymous with "instantaneous throughput". This number is useful for systems that rely on burst data transmission, however, for systems with a high duty cycle this is less likely to be a useful measure of system performance.  
  
**Bandwidth Description:**  
  
            The bandwidth of a computer network is the total amount of information that can be transmitted over it in a given time. The bandwidth is a measure of available or consumed data communication resources expressed in bits per second or multiples of it. It may refer to bandwidth capacity or available bandwidth in bits, which typically means the net bit rate, channel capacity or maximum throughput of a logical or physical communication path in a digital communication system.  
  
            It may also be referred to as consumed bandwidth, corresponding to achieved throughput or goodput, i.e. average data rate of successful data transfer through a communication path.  
  
            Bandwidth test software is used to determine the maximum bandwidth of a network or internet connection. It is typically undertaken by attempting to download or upload the maximum amount of data in a certain period of time, or a certain amount of data in the minimum amount of time. For this reason, Bandwidth tests can delay internet transmissions through the internet connection as they are undertaken, and can cause inflated data charges. A more accurate method is to use dedicated software such as Netcps,IxChariot, Iperf, Ttcp, netperf or bwping for measuring the maximum throughput of a network access.  
  
  
  
**Formula:**  
  
            Transmission Time = File Size / Bandwidth (sec)  
  
            Throughput = File Size / Transmission Time (bps)  
  
**Network Traffic Measurement:**  
  
            In computer networks, network traffic measurement is the process of measuring the amount and type of traffic on a particular network. This is especially important with regard to effective bandwidth management.  
  
            Various software tools are available to measure network traffic. Some tools measure traffic by sniffing and others use SNMP, WMI or other local agents to measure bandwidth use on individual machines and routers. However, the latter generally do not detect the type of traffic, nor do they work for machines which are not running the necessary agent software, such as rogue machines on the network, or machines for which no compatible agent is available. In the latter case, inline appliances are preferred. These would generally 'sit' between the LAN and the LAN's exit point, generally the WAN or Internet router, and all packets leaving and entering the network would go through them. In most cases the appliance would operate as a bridge on the network so that it is undetectable by users.  
  
            Measurement tools generally have these functions and features:  
  
    \* User interface (web, graphical, console)  
  
    \* Real-time traffic graphs  
  
    \* Network activity is often reported against pre-configured traffic matching rules to show:  
  
          o Local IP address  
  
          o Remote IP address  
  
          o Port number or protocol  
  
          o Logged in user name  
  
    \* Bandwidth quotas  
  
    \* Support for traffic shaping or rate limiting (overlapping with the network traffic control page)  
  
    \* Support website blocking and content filtering  
  
    \* Alarms to notify the administrator of excessive usage (by IP address or in total)  
  
**Network Measurement Tools:**  
  
            Some of the available network measurement tools include:  
  
    \* *Argus* processes packets into detailed network flow audit data for operations, performance and security management.  
  
    \* *Cacti* allows a user to poll services at predetermined intervals and graph the resulting data.  
  
    \* *Cricket* is a tool originally written for WebTV Networks.  
  
    \* *ET/BWMGR* by Emerging Technologies Inc is a software product that is like MRTG; capable of gathering statistics and displaying graphs on multi-gigabit networks.  
  
    \* *Exbander* Precision by DBAM Systems  
  
    \* *FireBeast* is a software firewall that offers bandwidth management and traffic shaping.  
  
    \* *FlowMon* by INVEA-TECH is a complete solution for NetFlow monitoring and analyzis including probes up to 10 Gbit/s, collectors and other supervision systems.  
  
    \* *Infosim* supports all different network flow technologies such as Netflow, sFlow, jFlow, cFlow or Netstream.  
  
    \* *InterMapper* Originally developed for the Macintosh Classic in 1994 by the network manger of Dartmouth College this application uses SNMP, Ping and Netflow to build a graphical network map similar to HP Openview which shows bandwidth usage by port information and protocol. VLAN aware. Supported platforms: MacOS X, Linux and Windows.  
  
    \* *MRTG*.  
  
    \* *NetLimiter* is a traffic monitoring and shaping software for Windows.  
  
    \* *OmniPeek* is an end-to-end network monitoring solution, offering support for many packet adapters and remote collectors.  
  
    \* *PRTG* runs on Windows, with graphical and web interfaces. It captures packets using Cisco Netflow or packet sniffing or uses SNMP to monitor bandwidth usages.  
  
    \* *PacketTrap* Networks - Traffic and Traffic Flow Analyzer  
  
    \* *PathSolutions Switchmonitor Network Performance Monitoring System*.  
  
    \* *SecurActive* by SecurActive is a Network Behavior Analysis solution with network anomalies detection.  
  
    \* *Sandvine* Intelligent Network Solutions measure and manage network traffic using Policy Traffic Switches  
  
    \* *SevOne* Network Performance Monitoring System.  
  
    \* *Solana* Networks builds Layer 2 and 3 network maps and measures BW as well as QoS  
  
    \* *Scrutinizer NetFlow and sFlow Analyzer* provides deep visibility into network traffic behavior and trends. Leveraging NetFlow, J-Flow, and sFlow data, NetFlow Traffic Analyzer identifies which users and applications are consuming the most bandwidth.  
  
**Class Application:**  
  
            Class Application has only one key variable agent\_ which is a pointer to class Agent. Other two variables, enableRecv\_ and enableResume\_, are flag variables, which indicate whether an Application object should react to functions recv(nbytes) and resume(), respectively. These two flag variables are set to zero by default.  
  
After their connection is created, an application and an agent may invoke public functions of each other through the pointers agent\_ and app\_, respectively. The key public functions of class Application include functions send(nbytes), recv(nbytes), and resume(), while those of class Agent are functions send(nbytes), sendmsg(nbytes), close(), listen(), and set\_pkttype(pkttype). Apart from these public functions, class Application also provides protected functions start() and stop() to start and stop an Application object, respectively. Finally, there are five key OTcl commands for class Application which can be invoked from the OTcl domain: start*{}*, stop*{}*, agent*{}*, send*{*nbytes*}*, and attach-agent*{*agent*}*.   
  
            The three following public functions of class Application:  
  
*•*send(nbytes):   Inform the attached transport layer agent that a user needs to send nbytes of data payload. Line 3 sends the demand to the attached agent by executing “agent\_->sendmsg(nbytes)”.  
  
*•*recv(nbytes):    Receive “nbytes” bytes from a receiving transport layer agent. A UDP agent specifies nbytes as the number of bytes in a received packet. In case of UDP, nbytes is equal to packet size; on the other hand,  
  
TCP specifies “nbytes” as the number of in-sequence received bytes. Due to possibility of out-of-order packet delivery, nbytes can be greater than the size of one packet.  
  
*•*resume():   Invoked by a sending agent, this function indicates that the agent has sent out all data corresponding to the user demand. For a TCP sender, this function is invoked when it sends out all the packets regardless of whether the transmitted packets have been acknowledged.  
  
**Public Functions of Class Agent:**  
  
            Class Application may invoke the following functions of class Agent through variable agent\_:  
  
*•*send(nbytes): Send “nbytes” of application payload (i.e., user demand) to a receiving agent. If nbytes=-1, the user demand would be infinite.  
  
*•*sendmsg(nbytes,flags): Similar to function send(nbytes), but also feed flags as an input variable.  
  
*•*close(): Ask an agent to close the connection (applicable only to TCP)  
  
*•*listen(): Ask an agent to listen to (i.e., wait for) a new connection (applicable only to Full TCP)  
  
*•*set\_pkttype(pkttype): Set the variable type\_ of the attach agent to be pkttype.  
  
  
  
**Basic Agents**  
  
            An agent is an NsObject which is responsible for creating and destroying packets. There are two main types of NS2 agents: routing agents and transportlayer agents. A routing agent creates and receives routing control packets, and commands routing protocols to act accordingly. Connecting an application to a low level network, a transport-layer agent controls the congestion and reliability of a data flow based on an underlying transport layer protocol (e.g., UDP or TCP).  
  
  
  
**Agent Configuration**  
  
            Agent configuration consists of four main steps:  
  
(i) Create a sending agent, a receiving agent, and an application using “new*{*...*}*”.  
  
(ii) Attach agents to the application using OTcl Command attach-agent- *{*agent*}*of class Application.  
  
(iii) Attach agents to the a low-level network using instproc attach-agent- *{*node agent*}*of class Simulator.  
  
(iv) Associate the sending agent with the receiving agent using instproc connect*{*src dst*}*of class Simulator.  
  
**Internal Mechanism for Agents**  
  
*• A sending agent*: Receive user demand by having the associated application invoke its function sendmsg(...). From within sendmsg(...), create packets using function allocpkt() and forward the created packets to the low-level network by executing target\_->recv(p,h).  
  
*• A receiving agent*: Receive packets by having a low-level network demultiplexer invoke its function recv(p,h). Destroy received packets by invoking function free(p) of class Packet.  
  
  
  
**UDP and Null Agents**  
  
            UDP (User Datagram Protocol) is a connectionless transport layer protocol, which provides neither congestion control nor error control. In NS2, a UDP agent is used as a sending agent. It is usually peered with a Null (receiving) agent, which is responsible for packet destruction.  
  
  
  
**Null (Receiving) Agents**  
  
            A Null agent is the simplest but one of the most widely-used receiving agents. The main responsibility of a Null agent is to deallocate packets, through function free(p) of class Packet. A Null agent is represented by an OTcl class Agent/Null which is derived directly from an OTcl class Agent (see file ˜*ns*/tcl/lib/ns-agent.tcl).  
  
  
  
**UDP (Sending) Agent**  
  
            A UDP agent is perhaps the simplest form of sending agents. It receives user demand to transmit data by having the attached application invoke its function (e.g., sendmsg(...)), creates packets based on the demand, and forwards the created packet to a low-level network. An application may use three following ways to tell a UDP agent to send out packets: via a C++ function sendmsg(...) of class UdpAgent, via an OTcl command send*{*...*}*of OTcl class Agent/UDP, or via an OTcl command sendmsg*{*...*}*of OTcl  
  
**class Agent/UDP**  
  
            A UDP agent is represented by a C++ class UdpAgent and an OTcl class Agent/UDP. These two classes derive from class Agent in their domains, and are bound by using a mapping class UdpAgentClass .In NS2, UDP is implemented in the C++ class UdpAgent which is bound to an OTcl class Agent/UDP.  
  
  
  
**Built-in Traffic Generators in NS2**  
  
**Constant Bit Rate (CBR)**  
  
            A CBR traffic generator creates a fixed size payload burst for every fixed interval. NS2 implements CBR traffic generators by using a C++ class CBR\_Traffic which is bound to an OTcl class. Application/Traffic/CBR by default the inter-burst transmission interval, which is the interval between the beginning of two successive payload bursts, can be computed by dividing the payload burst size by the sending rate.  
  
**Exponential On/Off**  
  
            An exponential on/off traffic generator acts as a CBR traffic generator during an ON interval and does not generate any payload during an OFF interval. ON and OFF periods are both exponentially distributed. NS2 implements Exponential On/Off traffic generators by using the C++ class which is named EXPOO\_Traffic which is bound to an OTcl class Application/Traffic/Exponential.  
  
**Pareto On/Off**  
  
A Pareto On/Off traffic generator does the same as an Exponential On/Off generator but the ON and OFF periods conform to a Pareto distribution. NS2 implements Pareto On/Off traffic generators by using a C++ class named POO\_Traffic which is bound to an OTcl class Application/Traffic/Pareto.  
  
TCL AND [AWK](http://www.cs.utah.edu/dept/old/texinfo/gawk/gawk_toc.html#SEC9):             Tcl can be invoked from a shell command prompt with the following syntax:  
  
tclsh [<filename> <arg0> <arg1> ...]  
  
Alternatively, since NS2 is written in Tcl, the following invocation would lead to the same result.  
  
>>ns convert.tcl  
  
  
  
**Variable Assignment and Retrieval:**  
  
            Tcl stores a value in a variable using the reserved word “set”. The value stored in a variable can be retrieved by placing a character “$” in front of a variable name. In addition, a reserved word “unset” is used to clear the value stored in a variable.  
  
  
  
**Commands**gets **and**puts:  
  
            The command puts and gets reads and writes, respectively, a message to a specified Tcl channel. In particular, the command “gets” reads a line from a Tcl channel, and passes every character in the line except the end-of-line character to the Tcl running environment. The Tcl channel could be a standard  
  
channel or a file channel. The syntax of the command gets is as follows:  
  
gets <channel> <var>  
  
 all the characters in the current line from the channel channel will be stored in the variable <var>.  
  
The command “puts” writes a string <string> followed by an end-ofline character to a Tcl channel <channel>. If <channel> is not specified, the stdout will be used as a default channel. The syntax of the command puts is as follows:  
  
puts [-nonewline] ]<channel>[ <string>  
  
where  nonewline option above specifies not to write an end-of-line character to the end of the string.  
  
  
  
**Mathematical Expressions:**  
  
            The primary method of doing calculations is by way of Tcl's **expr** command.  
  
we should always enclose the argument expression to [expr] in curly braces. This allows the bytecode compiler to optimise your code more thoroughly since it has fewer possible interpretations and the contents of variables and the results of commands will not unexpectedly modify the meaning of the expression.The part or whole of the expression is not a constant, e.g.  
  
   set v [expr "$a $op $b"]                                                  
  
if the operator $op varies, then surrounding this expression with { } makes the proper evaluation fail.  
  
  
  
**Operators**  
  
**- + ~ !**  
  
Unary operators; specifically a negation operation, a non-negation operation (I see little point in this one), a bit-wise NOT operation (every bit in the input value gets replaced by its inverse) and a logical NOT operation (non-zero maps to zero, and zero maps to one.)  
  
**\* / %**  
  
Multiplication, division and *integer* remainder.  
  
**+ -**  
  
Addition and subtraction.  
  
**<< >>**  
  
Left and right shift. Equivalent to multiplying or dividing by a suitable power of two, and then reducing the result to the range representable in an integer on the host platform.  
  
**< > <= >=**  
  
Ordering relations (less than, greater than, less than or equal, greater than or equal.) Note that these operations work on strings as well as numbers, but you are probably better off testing the result of [string compare] instead as that is more predictable in the case of a string that looks like a number.  
  
**== !=**  
  
Equality and inequality. Note that these operations work on strings as well as numbers, but you are probably better off testing the result of [string equal] instead as that is more predictable in the case of a string that looks like a number.  
  
&  
  
Bit-wise AND. A bit is set in the result when the corresponding bit is set in both the arguments.  
  
**^**  
  
Bit-wise exclusive OR. A bit is set in the result when the corresponding bit is set in *precisely one* of the arguments.  
  
**|**  
  
Bit-wise OR. A bit is set in the result when the corresponding bit is set in either of the arguments.  
  
&&  
  
Logical AND. The result is a one (true) when both of the arguments are non-zero (true), and zero (false) otherwise. Note that this operation is a *short-circuiting* operation, and will only evaluate its second argument when the first argument is non-zero. This includes the expansion of Tcl commands in square brackets, but this delay in evaluation only occurs if the whole expression is enclosed in curly braces.  
  
**||**  
  
Logical OR. The result is a zero (false) when both of the arguments are zero (false), and one (true) otherwise. Note that this operation is a *short-circuiting* operation, and will only evaluate its second argument when the first argument is zero. This includes the expansion of Tcl commands in square brackets, but this delay in evaluation only occurs if the whole expression is enclosed in curly braces.  
  
x?y:z  
  
.  
  
If-then-else, as in C (where x,y,z are expressions). If the value x is non-zero (true) then the expression y is evaluated to produce the result, and otherwise the expression z is evaluated to produce the result. Note that this operation is a*short-cicuiting* operation, and will not evaluate expression z if x is zero (false) and will not evaluate expression y if x is non-zero (true). This includes the expansion of Tcl commands in square brackets, but this delay in evaluation only occurs if the whole expression is enclosed in curly braces. It is usually clearer and easier to maintain (and no slower - the generated bytecode is identical) to use the Tcl [if] command instead of this.  
  
**Awk:**  
  
            AWK can be invoked from a command prompt in two ways based on the following syntax:  
  
>>awk [ -F<ch> ] {<pgm>} [ <vars> ] [ <data\_file> ]  
  
>>awk [ -F<ch> ] { -f <pgm\_file> } [ <vars> ] [ <data\_file> ]  
  
where *{}*and [] contain mandatory and optional arguments, respectively. The bracket <> contains a variable which should be replaced with actual values at the invocation.   
  
These variables include ch Field separator pgm.   
  
An AWK script pgm\_file A file containing an AWK script (i.e., an AWK file)  
  
vars Variables used in an AWK file data\_file.   
  
            An input text file By default, AWK separates records by using a white space (i.e., one or more spaces or tabs). However, if the option “-F is present, AWK will use <ch> as a field separator.1   
  
            The upper invocation takes an AWK script <pgm> as an input argument, while the lower one takes an AWK file <pgm\_file> as an input argument. In both cases, variables <vars> and input text file <data\_file> can be optionally provided. If an input text file is not provided, AWK will wait for input argument from the standard input (e.g., keyboard) line by line.  
  
  
  
**AWK Script:**  
  
            The basic function of awk is to search files for lines (or other units of text) that contain certain patterns. When a line matches one of the patterns, awk performs specified actions on that line. awk keeps processing input lines in this way until the end of the input file is reached. When you run awk, you specify an awk program which tells awk what to do. The program consists of a series of rules.  Each rule specifies one pattern to search for, and one action to perform when that pattern is found.  
  
            Syntactically, a rule consists of a pattern followed by an action. The action is enclosed in curly braces to separate it from the pattern. Rules are usually separated by newlines. Therefore, an awk program looks like this:  
  
*pattern* { *action* }  
  
*pattern* { *action* }  
  
...  
  
            The awk utility reads the input files one line at a time. For each line, awk tries the patterns of all the rules. If several patterns match then several actions are run, in the order in which they appear in the awkprogram. If no patterns match, then no actions are run. After processing all the rules (perhaps none) that match the line, awk reads the next line. This continues until the end of the file is reached. Each rule's action is enclosed in its own pair of braces.  
  
**AWK Programming Structure**  
  
            The general form of an AWK program is shown below:  
  
BEGIN {<initialization>}  
  
<pattern1> {<actions>}  
  
<pattern2> {<actions>}  
  
.  
  
END {<final actions>}  
  
Prior to procession an input text file, AWK performs <initialization> specified in the curly braces located after the reserved word BEGIN. Then, for each record, it performs actions if the records match with the corresponding pattern. After processing the entire file, it performs <final actions> specified in the curly braces located after the reserved word END.  
  
  
  
**Operators and Output in awk**  
  
            The key operators in AWK are shown below.  
  
+ (addition) ++ (increment)  
  
- (subtraction) == (decrement)  
  
\* (multiplication) = (assignment)  
  
/ (division) % (modulo)  
  
            AWK outputs a variable or a string to a screen using either **print**or **printf**, whose syntax are as follows:  
  
print <item1> <item2> ...  
  
printf(<format>,<item1>,<item2>,...)  
  
where <item1>, <item2>, and so on can be either variables or strings,  
  
<format> is the format of the output.   
  
Using print, a string needs to be enclosed within a quotation mark (""), while a variable could be indicated as it is.  
  
  
  
**Control Structure**  
  
            In common with Tcl, AWK support three major types of control structures: if/else, while, and for . The syntaxes of these control structures are as follows:  
  
if(<condition>) <action 1> [else <action 2>]  
  
while(<condition>) <action>  
  
for(<initialization>;<condition>;<end-of-loop-action>) <action>  
  
AWK also contains four unconditional control commands:  
  
break Exit the loop  
  
contine Restart the loop  
  
next Process the next record  
  
exit Exit the program by executing the END operation  
  
**Running awk programs:**  
  
            There are several ways to run an awk program. If the program is short, it is easiest to include it in the command that runs awk, like this:  
  
awk 'program' input-file1 input-file2 ...  
  
where program consists of a series of patterns and actions, as described earlier.  
  
            When the program is long, you would probably prefer to put it in a file and run it with a command like this:  
  
awk -f program-file input-file1 input-file2 ...  
  
Once you are familiar with awk, you will often type simple programs at the moment you want to use them. Then you can write the program as the first argument of the awk command, like this:  
  
awk 'program' input-file1 input-file2 ...  
  
where program consists of a series of patterns and actions, as described earlier.  
  
            This command format tells the shell to start awk and use the program to process records in the input file(s). There are single quotes around the program so that the shell doesn't interpret any awk characters as special shell characters. They cause the shell to treat all of program as a single argument for awk. They also allow program to be more than one line long. This format is also useful for running short or medium-sized awk programs from shell scripts, because it avoids the need for a separate file for the awk program. A self-contained shell script is more reliable since there are no other files to misplace.  
  
Running Long Programs of awk  
  
            Sometimes the awk programs can be very long. In this case it is more convenient to put the program into a separate file. To tell awk to use that file for its program:  
  
awk -f source-file input-file1 input-file2 ...  
  
The `-f' tells the awk utility to get the awk program from the file source-file. Any file name can be used for source-file.  
  
  
  
**Comments in awk Programs:**  
  
            A comment is some text that is included in a program for the sake of human readers, and that is not really part of the program. Comments can explain what the program does, and how it works. Nearly all programming languages have provisions for comments, because programs are hard to understand without their extra help. In the awk language, a comment starts with the sharp sign character, `#', and continues to the end of the line. The awk language ignores the rest of a line following a sharp sign.  
  
**User-defined Functions in awk:**  
  
            Definitions of functions can appear anywhere between the rules of the awk program. Thus, the general form of an awk program is extended to include sequences of rules *and* user-defined function definitions.   
  
The definition of a function named *name* looks like this:  
  
function *name* (*parameter-list*) {  
  
     *body-of-function*  
  
}  
  
The keyword function may be abbreviated func.  
  
*name* is the name of the function to be defined. A valid function name is like a valid variable name: a sequence of letters, digits and underscores, not starting with a digit.  
  
*parameter-list* is a list of the function's arguments and local variable names, separated by commas. When the function is called, the argument names are used to hold the argument values given in the call. The local variables are initialized to the null string.  
  
The *body-of-function* consists of awk statements. It is the most important part of the definition, because it says what the function should actually *do*. The argument names exist to give the body a way to talk about the arguments; local variables, to give the body places to keep temporary values.  
  
            Argument names are not distinguished syntactically from local variable names; instead, the number of arguments supplied when the function is called determines how many argument variables there are. Thus, if three argument values are given, the first three names in *parameter-list* are arguments, and the rest are local variables.  
  
**Xgraph:**  
  
**Syntax:**  
  
xgraph [ options ] [[-geometry —=]WxH+X+Y ] [ -display host:display.screen ] [ file ... ]  
  
**Description:**  
  
            The xgraph program draws a graph on an X display given data read from either data files or from standard input if no files are specified. It can display up to 64 independent data sets using different colors and/or line styles for each set. It annotates the graph with a title, axis labels, grid lines or tick marks, grid labels, and a legend.  
  
            There are options to control the appearance of most components of the graph. A data set consists of an ordered list of points of the form “directive X Y”. For directive “draw”, a line will be drawn between the previous point and the current point. Specifying a “move” directive tells xgraph not to draw a line between the points. “draw” is the default directive. The name of a data set can be specified by enclosing the name in double quotes. Overall graphing options for the graph can be specified in data files by writing lines of the form “¡option¿: ¡value¿”. The interface used to specify the size and location of this window depends on the window manager currently in use.  
  
  
  
            Once the window has been opened, all of the data sets will be displayed graphically with a legend in the upper right corner of the screen. xgraphalso presents three control buttons in the upper left corner of each window: Hardcopy, Close and About xgraph accepts a large number of options most of which can be specified either on the command line, in the user’s .Xdefaults or .Xresources file, or in the data files themselves. A list of these options is given below. The format of the option in the X defaults file is ”program.option: value” where program is the program name (xgraph) and the option name is the one specified below. Option specifications in the data file are similar to the X defaults file specification except the program name is omitted.  
  
*/-geometry WxH+X+Y or /=WxH+X+Y (Geometry)*  
  
Specifies the initial size and location of the xgraph window.  
  
*/-bar (BarGraph)*  
  
Specifies that vertical bars should be drawn from the data points to a  
  
base point which can be specified with -brb. Usually, the -nl flag is used  
  
with this option. The point itself is located at the center of the bar.  
  
*/-fitx*  
  
Translate and scale the x data from all datasets to fit [0. . . 1].  
  
*/-fity*  
  
Translate and scale the y data from all datasets to fit [0. . . 1].  
  
*/-fmtx <printf-format> /-fmty <printf-format>*  
  
Use the format specified to generate the legends for the x or y axis.  
  
*/-bb (BoundBox)*  
  
Draw a bounding box around the data region. This is very useful if you  
  
prefer to see tick marks rather than grid lines (see -tk).  
  
*/-bd <color> (Border)*  
  
This specifies the border color of the xgraph window.  
  
*/-bg <color> (Background)*  
  
Background color of the xgraph window.  
  
*/-brb <base> (BarBase)*  
  
This specifies the base for a bar graph. By default, the base is zero.  
  
*/-brw <width> (BarWidth)*  
  
This specifies the width of bars in a bar graph. The amount is specified  
  
in the user’s units. By default, a bar one pixel wide is drawn.  
  
*/-bw <size> (BorderSize)*  
  
Border width (in pixels) of the xgraph window.  
  
*/-fg <color> (Foreground)*  
  
Foreground color. This color is used to draw all text and the normal  
  
grid lines in the window.  
  
*/-gw (GridSize)*  
  
Width, in pixels, of normal grid lines.  
  
*/-gs (GridStyle)*  
  
Line style pattern of normal grid lines.  
  
*/-lf <fontname> (LabelFont)*  
  
Label font. All axis labels and grid labels are drawn using this font.  
  
A font name may be specified exactly (e.g. ”9x15” or ”-\*-courier-bold-rnormal-\*-  
  
140-\*”) or in an abbreviated form: ¡family¿-¡size¿. The family is  
  
the family name (like helvetica) and the size is the font size in points (like  
  
12). The default for this parameter is ”helvetica-12”.  
  
*/-lnx (LogX)*  
  
Specifies a logarithmic X axis. Grid labels represent powers of ten.  
  
*/-lny (LogY)*  
  
Specifies a logarithmic Y axis. Grid labels represent powers of ten.  
  
*/-lw width (LineWidth)*  
  
Specifies the width of the data lines in pixels. The default is zero.  
  
*/-lx <xl,xh> (XLowLimit, XHighLimit)*  
  
This option limits the range of the X axis to the specified interval. This  
  
(along with -ly) can be used to ”zoom in” on a particularly interesting portion  
  
of a larger graph.  
  
*/-ly <yl,yh> (YLowLimit, YHighLimit)*  
  
This option limits the range of the Y axis to the specified interval.  
  
*/-m (Markers)*  
  
Mark each data point with a distinctive marker. There are eight distinctive  
  
markers used by xgraph. These markers are assigned uniquely to each  
  
different line style on black and white machines and varies with each color  
  
on color machines.  
  
*/-M (StyleMarkers)*  
  
Similar to -m but markers are assigned uniquely to each eight consecutive  
  
data sets (this corresponds to each different line style on color machines).  
  
*/-nl (NoLines)*  
  
Turn off drawing lines. When used with -m, -M, -p, or -P this can be  
  
used to produce scatter plots. When used with -bar, it can be used to produce  
  
standard bar graphs.  
  
*/-ng (NoLegend)*  
  
Turn off drawing Legends. Can be used to increase the drawing area.  
  
*/-t <string> (TitleText)*  
  
Title of the plot. This string is centered at the top of the graph.  
  
*/-tf <fontname> (TitleFont)*  
  
Title font. This is the name of the font to use for the graph title. A font  
  
name may be specified exactly (e.g. ”9x15” or ”-\*-courier-bold-r-normal-\*-  
  
140-\*”) or in an abbreviated form: ¡family¿-¡size¿. The family is the family  
  
name (like helvetica) and the size is the font size in points (like 12). The  
  
default for this parameter is ”helvetica-18”.  
  
*/-x <unitname> (XUnitText)*  
  
This is the unit name for the X axis. Its default is ”X”.  
  
*/-y <unitname> (YUnitText)*  
  
This is the unit name for the Y axis. Its default is ”Y”.  
  
*/-zg <color> (ZeroColor)*  
  
This is the color used to draw the zero grid line.  
  
*/-zw <width> (ZeroWidth)*  
  
This is the width of the zero grid line in pixels.            
  
**Wireless Trace File Format:**  
  
            s 0.500000000 \_0\_ AGT  --- 0 tcp 40 [0 0 0 0] ------- [0:1 49:0 32 0] [0 0] 0 0  
  
            r 0.500000000 \_0\_ RTR  --- 0 tcp 40 [0 0 0 0] ------- [0:1 49:0 32 0] [0 0] 0 0  
  
            The new trace format as seen above can be can be divided into the following fields:  
  
**Event type:**In the traces above, the first field (as in the older trace format) describes the type of event taking place at the node and can be one of the four types:  
  
**s**send  
  
**r**receive  
  
**d**drop  
  
**f**forward  
  
  
  
**General tag:**The second field starting with "-t" may stand for time or global setting  
  
**-t**time  
  
**-t**\* (global setting)  
  
  
  
**Node property tags:**This field denotes the node properties like node-id, the level at which tracing is being done like agent, router or MAC. The tags start with a leading "-N" and are listed as below:  
  
**-Ni:**node id  
  
**-Nx:**node’s x-coordinate  
  
**-Ny:**node’s y-coordinate  
  
**-Nz:**node’s z-coordinate  
  
**-Ne:**node energy level  
  
**-Nl:**trace level, such as AGT, RTR, MAC  
  
**-Nw:**reason for the event. The different reasons for dropping a packet are given below:  
  
**"END"**DROP\_END\_OF\_SIMULATION  
  
**"COL"**DROP\_MAC\_COLLISION  
  
**"DUP"**DROP\_MAC\_DUPLICATE  
  
**"ERR"**DROP\_MAC\_PACKET\_ERROR  
  
**"RET"**DROP\_MAC\_RETRY\_COUNT\_EXCEEDED  
  
**"STA"**DROP\_MAC\_INVALID\_STATE  
  
**"BSY"**DROP\_MAC\_BUSY  
  
**"NRTE"**DROP\_RTR\_NO\_ROUTE i.e no route is available.  
  
**"LOOP"**DROP\_RTR\_ROUTE\_LOOP i.e there is a routing loop  
  
**"TTL"**DROP\_RTR\_TTL i.e TTL has reached zero.  
  
**"TOUT"**DROP\_RTR\_QTIMEOUT i.e packet has expired.  
  
**"CBK"**DROP\_RTR\_MAC\_CALLBACK  
  
**"IFQ"**DROP\_IFQ\_QFULL i.e no buffer space in IFQ.  
  
**"ARP"**DROP\_IFQ\_ARP\_FULL i.e dropped by ARP  
  
**"OUT"**DROP\_OUTSIDE\_SUBNET i.e dropped by base stations on receiving routing updates from nodes outside its domain.  
  
  
  
**Packet information at IP level:**The tags for this field start with a leading "-I" and are listed along with their explanations as following:  
  
**-Is:**source address.source port number  
  
**-Id:**dest address.dest port number  
  
**-It:**packet type  
  
**-Il:**packet size  
  
**-If:**flow id  
  
**-Ii:**unique id  
  
**-Iv:**ttl value  
  
  
  
**Next hop info:**This field provides next hop info and the tag starts with a leading "-H".  
  
**-Hs:**id for this node  
  
**-Hd:**id for next hop towards the destination.  
  
  
  
**Packet info at MAC level:**This field gives MAC layer information and starts with a leading "-M" as shown below:  
  
**-Ma:**duration  
  
**-Md:**dst’s ethernet address  
  
**-Ms:**src’s ethernet address  
  
**-Mt:**ethernet type  
  
**Packet info at "Application level":**The packet information at application level consists of the type of application like ARP, TCP, the type of adhoc routing protocol like DSDV, DSR, AODV etc being traced. This field consists of a leading "-P" and list of tags for different application is listed as below:  
  
**-P arp**Address Resolution Protocol. Details for ARP is given by the following tags:  
  
**-Po:**ARP Request/Reply  
  
**-Pm:**src mac address  
  
**-Ps:**src address  
  
**-Pa:**dst mac address  
  
**-Pd:**dst address  
  
**-P dsr**This denotes the adhoc routing protocol called Dynamic source routing. Information on DSR is represented by the following tags:  
  
**-Pn:**how many nodes traversed  
  
**-Pq:**routing request flag  
  
**-Pi:**route request sequence number  
  
**-Pp:**routing reply flag  
  
**-Pl:**reply length  
  
**-Pe:**src of srcrouting->dst of the source routing  
  
**-Pw:**error report flag ?  
  
**-Pm:**number of errors  
  
**-Pc:**report to whom  
  
**-Pb:**link error from linka->linkb  
  
**-P cbr**Constant bit rate.   
  
Information about the CBR application is represented by the following tags:  
  
**-Pi:**sequence number  
  
**-Pf:**how many times this pkt was forwarded  
  
**-Po:**optimal number of forwards  
  
**-P tcp**Information about TCP flow is given by the following subtags:  
  
**-Ps:**seq number  
  
**-Pa:**ack number  
  
**-Pf:**how many times this pkt was forwarded  
  
**-Po:**optimal number of forwards  
  
This field is still under development and new tags shall be added for other applications as they get included along the way.  
  
  
  
**Description of Tcl Commands used:**  
  
            1. The ‘set’ and ‘val( )’ keywords are used to initialize the configuration parameters, as shown below.  
  
                                    “ set val(chan) Channel/WirelessChannel “  
  
            2. The ‘new’ keyword is used to create a new object reference to a particular class, as shown below.  
  
                                    “ set ns [new Simulator] “  
  
            3. The ‘open’ keyword is used to open a file in the given r/w/x mode. If that particular file does not exist, it is created and opened, as shown below.  
  
                                    “ set tf [open wireless.tr w] “  
  
            4. The ‘trace-all’ function is used to trace the events in the opened trace file (\*.tr).  
  
            5. The ‘namtrace-all-wireless’ function is to trace the events in the nam file created (\*.nam).  
  
            6. The ‘load\_flatgrid’ function is used to load the topography value of the simulation, like 1000 x 1000, as shown below.  
  
                                    “ $topo load\_flatgrid 1000 1000 “  
  
            7. The ‘create-god’ function  is used to create the General Operations Director.  
  
            8. The ‘node-config’ function is used to configure the node by setting in it the configuration parameters.  
  
            9. The ‘attach-agent’ function is used to link one agent/application to another node/agent respectively.  
  
            10. The ‘setdest’ function is used to set the position of the node at a particular time.  
  
            11. The ‘start’ and ‘stop’ keywords are used to start and stop the application respectively.  
  
            12. The ‘proc’ keyword is used to indicate a procedure or a function.  
  
            13. The ‘flush-trace’ function is used to flush the traced events into the trace files.  
  
            14. The ‘run’ keyword is used to run the file.  
  
**TOPOLOGY:**  
  
  
  
Application  
  
starts at 0.5 sec  
  
stops at 159 sec  
  
Simulation Time 160 sec  
  
Topography = 500 x 500  
  
Node Movement:  
  
Time                Node               (x, y, z)  
  
0                      0                      (5, 10, 0)  
  
0                      2                      (300, 300, 0)  
  
30                    1                      (50, 300, 0)  
  
50                    1                      (50, 50, 0)  
  
Configuration Parameters:  
  
Channel Type: Wireless Channel  
  
Radio propagation model: TwoRay Ground  
  
Network Interface type: Wireless Phy  
  
MAC Type: 802-11  
  
Interface Queue Type: PriQueue  
  
LinkLayer type: LL  
  
Antenna model: Omni Antenna  
  
Max. Pts in ifq: 50  
  
No. of mobile nodes: 3  
  
Routing Protocol: DSDV  
  
  
  
**PROCEDURE:**  
  
            1. Initialize the network configuration parameters of the network to be simulated using the 'set' keyword and the 'val()' keyword.  
  
            2. While initializing, the super class of the particular class is also denoted. The example of an initialization is   
  
                        “ set val(chan) Channel/WirelessChannel “  
  
            3. The values of link layer type, interface queue length, the number of nodes and the routing protocol variables can be directly given without any superclass.  
  
                        ex.:  set val(ll) LL  
  
                                set val(nn) 3  
  
            4. Create an object (say, ns) for the Simulator class using the 'set' keyword, and 'new' keyword.  
  
                        “ set ns [new Simulator] “  
  
            5. Create and open a trace file with write mode (i.e., w) using 'set' and 'open' keywords.  
  
                        “ set tf [open wireless.tr w] “  
  
            6. Using the 'trace-all' function, set the ns object to trace all the events and write the trace file created.  
  
            7. Create a nam file in write mode and using 'namtrace-all-wireless' function, with arguments as the nam file handler and the topography values (here, 500 x 500), the events traceds are put in nam file.  
  
            8. Create an object for the Topography class and give the values (500 x 500) in the ‘load\_flatgrid' function.  
  
            9. Pass the value of the number of nodes to the 'create-god' function.  
  
            10. Set the values for the arguments of the 'node-config' function with all the configuration parameter values.  
  
            11. Create the nodes 0, 1 and 2 using the 'set' and '$ns' keywords shown.  
  
                        “ set node0 [$ns node] “  
  
            12. Set the position of the nodes in the X\_, Y\_ and Z\_ variables using 'set' keyword.  
  
            13. Create the UDP agent and attach it to node0 using the 'attach-agent' function.  
  
            14. Similarly create the object for CBR Application and attach it to the UDP object using 'attach-agent' function.  
  
            15. Create the object for the NULL and attach it to the node2 using 'attach-agent' function.  
  
            16. Connect the UDP object to the NULL object using the 'connect' function.  
  
            17. Set the movement of node1 using 'at' keyword, specifying the time in seconds and the position in x, y, z values using 'setdest' function.  
  
            18. The application start time and stop time are specified using the 'at' keyword along with the 'start' and 'stop' keywords.  
  
            19. The Simulation stop time is indicated using the 'at' keyword and call the 'finish' procedure.  
  
            20. Inside the 'finish' procedure, set the values ns, tf and tf1 as 'global'.  
  
            21. Call the 'flush-trace' function to write all the events traced in the trace files.  
  
            22. Close the trace file using 'close' function.  
  
            23. Execute the nam file using 'exec' keyword and put & at the end of the line.  
  
            24. Call the 'exit' function with parameter '0', and close the 'finish' procedure.   
  
            25. Run the simulation using the 'run' keyword.  
  
            26. Close the editor.  
  
            27. In the out.awk file, check if the packet is a received packet and it is of cbr type.  
  
            28. Then increment the ‘packets’ variable with the value in the 8th column of the trace file. This gives the number of bytes received.  
  
            29. Calculate the number of bits received by multiplying the value of ‘packets’ variable with 8.  
  
            30. Divide the value by the corresponding time got from the 2nd column in the trace file.  
  
            31. Multiply the result with 1000000 to get the value in Mbps units. Display the throughput.  
  
            32. In out1.awk file, display the time and throughput values alone. This file is used to create the graph.  
  
            33. Execute the awk files to display the throughput and the corresponding graph.  
  
            34. Thus the throughput is calculated for the defined network simulator.  
  
  
  
**PROGRAM:**  
  
  
set val(chan) Channel/WirelessChannel  
set val(prop) Propagation/TwoRayGround  
set val(netif) Phy/WirelessPhy  
set val(mac) Mac/802\_11  
set val(ifq) Queue/DropTail/PriQueue  
set val(ll) LL  
set val(ant) Antenna/OmniAntenna  
set val(ifqlen) 50  
set val(nn) 3  
set val(rp) DSDV  
set ns [new Simulator]  
  
set tf [open wireless.tr w]  
$ns trace-all $tf  
  
set tf1 [open wireless1.nam w]  
$ns namtrace-all-wireless $tf1 500 500  
  
set topo [new Topography]  
$topo load\_flatgrid 500 500  
  
create-god $val(nn)  
  
$ns node-config -adhocRouting $val(rp) \  
-llType $val(ll) \  
-macType $val(mac) \  
-ifqType $val(ifq) \  
-ifqLen $val(ifqlen) \  
-antType $val(ant) \  
-propType $val(prop) \  
-phyType $val(netif) \  
-channelType $val(chan) \  
-topoInstance $topo \  
-agentTrace ON \  
-routerTrace OFF \  
-macTrace OFF \  
-movementTrace OFF   
  
set node0 [$ns node]  
set node1 [$ns node]  
set node2 [$ns node]  
  
$ns initial\_node\_pos $node0 10  
$ns initial\_node\_pos $node1 10  
$ns initial\_node\_pos $node2 10  
  
$node0 set X\_ 5.0  
$node0 set Y\_ 5.0  
$node0 set Z\_ 0.0  
  
$node1 set X\_ 50.0  
$node1 set Y\_ 50.0  
$node1 set Z\_ 0.0  
  
$node2 set X\_ 100.0  
$node2 set Y\_ 100.0  
$node2 set Z\_ 0.0  
  
set udp1 [new Agent/UDP]  
$ns attach-agent $node0 $udp1  
  
set cbr1 [new Application/Traffic/CBR]  
$cbr1 attach-agent $udp1  
  
set null1 [new Agent/Null]  
$ns attach-agent $node2 $null1  
  
$ns connect $udp1 $null1  
  
$ns at 0.0 "$node0 setdest 5.0 10.0 0.0"  
$ns at 0.0 "$node2 setdest 300.0 300.0 0.0"  
$ns at 30.0 "$node1 setdest 30.0 300.0 0.0"  
$ns at 50.0 "$node1 setdest 50.0 50.0 0.0"  
  
$ns at 0.5 "$cbr1 start"  
$ns at 159 "$cbr1 stop"  
  
$ns at 160 "finish"  
  
proc finish {} {  
global ns tf tf1  
$ns flush-trace  
close $tf  
close $tf1  
exec nam wireless1.nam &  
exit 0  
}  
  
$ns run  
  
**out.awk:**  
  
BEGIN {   
Print “Throughput Calculation”  
}  
  
{  
if (( $1 == “r” && $7 == “cbr” && $3 == “\_2\_“ ))  
{  
pkts = pkts + $8;  
}  
  
}  
  
END {  
Throughput  = pkts \* 8 / $2 /1000000  
print “Throughput = “ Throughput  
}  
  
**out1.awk:**  
  
{  
if (( $1 == “r” && $7 == “cbr” && $3 == “\_2\_“ ))  
{  
pkts = pkts + 8;  
print $2, pkts \* 8/ $2 / 1000000  
}  
}  
  
  
  
**OUTPUT:**  
  
  
  
**- Terminal**  
  
  
  
  
  
**- Nam Output**  
  
  
  
**- Termainal – awk**  
  
  
  
  
  
**- Graph**

**CREATE A**[**FREE WEBSITE**](http://www.weebly.com/?utm_source=internal&utm_medium=footer&utm_campaign=3)

**POWERED BY**

**START YOUR OWN FREE WEBSITE**

A surprisingly easy drag & drop site creator. [Learn more.](http://www.weebly.com/?utm_source=internal&utm_medium=footer&utm_campaign=3)