PackMime-HTTP: Web Traffic Generation in NS-2

The PackMime Internet traffic model was developed by researchers in the Internet Traffic Research group at Bell Labs, based on recent Internet traffic traces. PackMime includes a model of HTTP traffic, called PackMime-HTTP. The traffic intensity generated by PackMime-HTTP is controlled by the rate parameter, which is the average number of new connections started each second. The PackMime-HTTP implementation in ns-2, developed at UNC-Chapel Hill, is capable of generating HTTP/1.0 and HTTP/1.1 (persistent, non-pipelined) connections.

PackMime-HTTP in ns-2 uses DelayBox, a module developed at UNC-Chapel Hill for delaying and/or dropping packets in a flow according to a distribution. DelayBox can be used with PackMime-HTTP to simulate a "cloud" of clients and servers that have different round-trip times, bottleneck link speeds, and amounts of packet loss.

The PackMime HTTP traffic model is described in detail in the following paper: J. Cao, W.S. Cleveland, Y. Gao, K. Jeffay, F.D. Smith, and M.C. Weigle , “Stochastic Models for Generating Synthetic HTTP Source Traffic”, *Proceedings of IEEE INFOCOM*, Hong Kong, March 2004.

Implementation Details

PackMimeHTTP is an ns object that drives the generation of HTTP traffic. Each PackMimeHTTP object controls the operation of two types of Applications, a PackMimeHTTP server Application and a PackMimeHTTP client Application. Each of these Applications is connected to a TCP Agent (Full-TCP). **Note:** PackMime-HTTP only supports Full-TCP agents.

Each web server or web client cloud is represented by a single ns node that can produce and consume multiple HTTP connections at a time (Figure **??**). For each HTTP connection, PackMimeHTTP creates (or allocates from the inactive pool, as described below) server and client Applications and their associated TCP Agents. After setting up and starting each connection, PackMimeHTTP sets a timer to expire when the next new connection should begin. The time between new connections is governed by the connection rate parameter supplied by the user. New connections are started according to the connection arrival times without regard to the completion of previous requests, but a new request between the same client and server pair (as with HTTP 1.1) begins only after the previous request-response pair has been completed.

PackMimeHTTP handles the re-use of Applications and Agents that have completed their data transfer. There are 5 pools used to maintain Applications and Agents – one pool for inactive TCP Agents and one pool each for active and inactive client and server Applications. The pools for active Applications ensure that all active Applications are destroyed when the simulation is finished. Active TCP Agents do not need to be placed in a pool because each active Application contains a pointer to its associated TCP Agent. New objects are only created when there are no Agents or Applications available in the inactive pools.

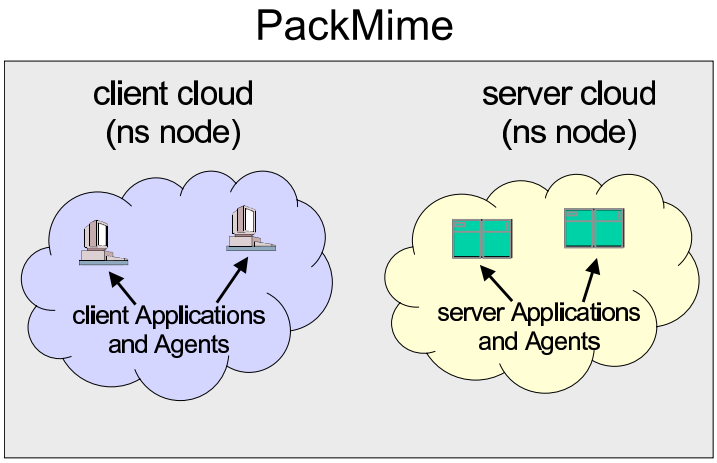

Figure 1: PackMimeHTTP Architecture. Each PackMimeHTTP object controls a server and a client cloud. Each cloud can represent multiple client or server Applications. Each Application represents either a single web server or a single web client.

PackMimeHTTP Client Application

Each PackMimeHTTP client controls the HTTP request sizes that are transferred. Each PackMimeHTTP client takes the following steps:

- sample the number of requests for this connection from the number-of-requests distribution (if the number of requests is 1, this is a non-persistent connection)
- sample the inter-request times from the inter-request-time distribution, if there will be more than 1 request
- sample the HTTP request sizes from the request-size distribution
- send the first HTTP request to the server
- listen for the HTTP response
- when the entire HTTP response has been received, the client sets a timer to expire when the next request should be made
- when the timer expires, the next HTTP request is sent, and the above process is repeated until all requests have been completed

PackMimeHTTP Server Application

Each web server controls the response sizes that are transferred. The server is started by when a new TCP connection is started. Each PackMimeHTTP client takes the following steps:

- listen for an HTTP request from the associated client
- when the entire request arrives, the server samples the server delay time from the server delay distribution
- set a timer to expire when the server delay has passed
- when the timer expires, the server samples the HTTP response sizes from the HTTP response size distribution

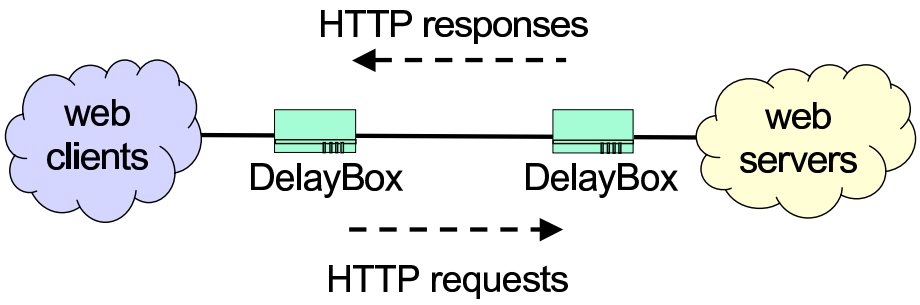

Figure 2: Example Topology Using PackMimeHTTP and DelayBox. The cloud of web clients is a single ns node, and the cloud of web servers is a single ns node. Each of the DelayBox nodes is a single ns node.

- this process is repeated until the requests are exhausted – the server is told how many requests will be sent in the connection
- send a FIN to close the connection

PackMimeHTTP Random Variables

This implementation of PackMimeHTTP provides several ns RandomVariable objects for specifying distributions of PackMimeHTTP connection variables. The implementations were taken from source code provided by Bell Labs and modified to fit into the ns RandomVariable framework. This allows PackMimeHTTP connection variables to be specified by any type of ns RandomVariable, which now include PackMimeHTTP-specific random variables. The PackMimeHTTP-specific random variable syntax for TCL scripts is as follows:

- `$ns [new RandomVariable/PackMimeHTTPFlowArrive <rate>]`, where `rate` is the specified PackMimeHTTP connection rate (number of new connections per second)
- `$ns [new RandomVariable/PackMimeHTTPFileSize <rate> <type>]`, where `type` is 0 for HTTP requests and 1 for HTTP responses
- `$ns [RandomVariable/PackMimeHTTPXmit <rate> <type>]`, where `type` is 0 for client-side delays and 1 for server-side delays. **Note:** This random variable is only used in conjunction with DelayBox. It returns 1/2 of the actual delay because it is meant to be used with 2 DelayBox nodes, each of which should delay the packets for 1/2 of the actual delay.

Use of DelayBox with PackMime-HTTP

PackMimeHTTP uses ns to model the TCP-level interaction between web clients and servers on the simulated link. To simulate network-level effects of HTTP transfer through the clouds, we implemented a new ns module called DelayBox (see **??**). DelayBox is an ns analog to dummynet, often used in network testbeds to delay and drop packets. The delay times model the propagation and queuing delay incurred from the source to the edge of the cloud (or edge of the cloud to destination). Since all HTTP connections in PackMimeHTTP take place between only two ns nodes, there had to be an ns object to delay packets in each flow, rather than just having a static delay on the link between the two nodes. DelayBox also models bottleneck links and packet loss on an individual connection basis. Two DelayBox nodes are used as shown in Figure **??**. One node is

placed in front of the web client cloud ns node to handle client-side delays, loss, and bottleneck links. The other DelayBox node is placed in front of the web server cloud ns node to handle the server-side delays, loss, and bottleneck links.

Example

More examples (including those that demonstrate the use of DelayBox with PackMime) are available in the `tcl/ex/packmime/` directory of the ns source code. The validation script `test-suite-packmime.tcl` is in `tcl/test/` and can be run with the command `test-all-packmime` from that directory.

```
# test-packmime.tcl

# useful constants
set CLIENT 0
set SERVER 1

remove-all-packet-headers;            # removes all packet headers
add-packet-header IP TCP;             # adds TCP/IP headers
set ns [new Simulator];               # instantiate the Simulator
$ns use-scheduler Heap;               # use the Heap scheduler

# SETUP TOPOLOGY
# create nodes
set n(0) [$ns node]
set n(1) [$ns node]
# create link
$ns duplex-link $n(0) $n(1) 10Mb 0ms DropTail

# SETUP PACKMIME
set rate 15
set pm [new PackMimeHTTP]
$pm set-client $n(0);                 # name $n(0) as client
$pm set-server $n(1);                 # name $n(1) as server
$pm set-rate $rate;                   # new connections per second
$pm set-http-1.1;                     # use HTTP/1.1

# SETUP PACKMIME RANDOM VARIABLES
global defaultRNG

# create RNGs (appropriate RNG seeds are assigned automatically)
set flowRNG [new RNG]
set reqsizeRNG [new RNG]
set rspsizeRNG [new RNG]

# create RandomVariables
set flow_arrive [new RandomVariable/PackMimeHTTPFlowArrive $rate]
set req_size [new RandomVariable/PackMimeHTTPFileSize $rate $CLIENT]
set rsp_size [new RandomVariable/PackMimeHTTPFileSize $rate $SERVER]

# assign RNGs to RandomVariables
$flow_arrive use-rng $flowRNG
$req_size use-rng $reqsizeRNG
```

```
$rsp_size use-rng $rspsizeRNG

# set PackMime variables
$pm set-flow_arrive $flow_arrive
$pm set-req_size $req_size
$pm set-rsp_size $rsp_size

# record HTTP statistics
$pm set-outfile "data-test-packmime.dat"

$ns at 0.0 "$pm start"
$ns at 300.0 "$pm stop"
$ns at 301.0 "exit 0"

$ns run
```

Commands at a Glance

The following commands on the PackMimeHTTP class can be accessed from OTcl:

`[new PackMimeHTTP]`
Creates a new PackMimeHTTP object.

`$packmime set-client <node>`
Associates the node with the PackMimeHTTP client cloud

`$packmime set-server <node>`
Associates the node with the PackMimeHTTP server cloud

`$packmime set-rate <float>`
Set the average number of new connections started per second

`$packmime set-req_size <RandomVariable>`
Set the HTTP request size distribution

`$packmime set-rsp_size <RandomVariable>`
Set the HTTP response size distribution

`$packmime set-flow_arrive <RandomVariable>`
Set the distribution of time between two consecutive connections starting

`$packmime set-server_delay <RandomVariable>`
Set the web server delay for fetching pages

`$packmime start`
Start generating connections

`$packmime stop`
Stop generating new connections

`$packmime set-http-1.1`
Use HTTP/1.1 distributions for persistent connections instead of HTTP/1.0.

`$packmime set-run <int>`
Set the run number so that the RNGs used for the random variables will use the same substream (see Chapter **??** on RNG for more details).

`$packmime active-connections`
Output the current number of active HTTP connections to standard error

`$packmime total-connections`
Output the total number of completed HTTP connections to standard error

`$packmime get-pairs`
Return the number of completed HTTP request-response pairs. See `tcl/ex/packmime/pm-end-pairs.tcl` for an example of using `get-pairs` to end the simulation after a certain number of pairs have completed.

`$packmime set-TCP <protocol>`
Sets the TCP type (Reno, Newreno, or Sack) for all connections in the client and server clouds - Reno is the default

`$packmime set-outfile <filename>`
Output the following fields (one line per HTTP request-reponse pair) to `filename`:

- time HTTP response completed
- HTTP request size (bytes)
- HTTP response size (bytes)
- HTTP response time (ms) – time between client sending HTTP request and client receiving complete HTTP response
- source node and port identifier
- number of active connections at the time this HTTP request-response pair completed

`$packmime set-debug <int>`
Set the debugging level:

- 1: Output the total number of connections created at the end of the simulation
- 2: Level 1 +
  output creation/management of TCP agents and applications
  output on start of new connection
  number of bytes sent by the client and expected response size
  number of bytes sent by server
- 3: Level 2 +
  output when TCP agents and applications are moved to the pool
- 4: Level 3 +
  output number of bytes received each time client or server receive a packet