# Simulation scenarios for Vertical handoff algorithm for different wireless technologies

#

#####################################################################

# General Parameters

#####################################################################

set opt(title) zero ;

set opt(seed) -1 ;

set opt(stop) 100 ;# Stop time.

set opt(ecn) 0 ;

#####################################################################

# Topology

#####################################################################

set opt(type) gprs ;#type of link:

set opt(bwUL) 0 ;# speed of congested link in kbps

set opt(bwDL) 0 ;# speed of congested link in kbps

set opt(propUL) 0 ;# uplink delay of congested link in s

set opt(propDL) 0 ;# downlink delay of congested link in s

set opt(secondDelay) 55 ;# average delay of access links in ms

set opt(rtts) 1 ;# 1 for a range of delays for the access links.

set opt(numWebNodes) 10 ;# number of access links for web traffic

set opt(accessLink) 1 ;# 1 for a range of bandwidth for the access links.

 # NOT IMPLEMENTED YET

#####################################################################

#

#####################################################################

set opt(minth) 30 ;

set opt(maxth) 0 ;

set opt(adaptive) 1 ;# 1 for Adaptive RED, 0 for plain RED

set opt(queue) DT ;# 0 for DropTail

 ;# 1 for RED

set opt(qsize) 0 ;# Queue size in packets.

#####################################################################

# Traffic generation.

#####################################################################

set opt(pingInt) 0 ;# ping interval, 0 = off

set opt(flows) 1 ;# number of long-lived TCP flows

set opt(shortflows) 0 ;# two short flows in the beginning

set opt(flowsRev) 0 ;# number of reverse long-lived TCP flows

set opt(flowsTfrc) 0 ;# number of long-lived TFRC flows

set opt(webers) 0 ;# number of web users

set opt(window) 30 ;# window for long-lived traffic

set opt(smallpkt) 10 ;# inverse of fraction of TCP connections

 # with smaller packets

set opt(reverse) 1 ;# reverse-path traffic

set opt(web) 2 ;# number of web sessions

set opt(numPage) 10 ;# number of pages per session

set opt(pagesize) 10 ;# number of objects per page

set opt(objSize) 60 ;# average size of web object in pkts.

set opt(shape) 1.05 ;# shape parameter for Pareto distribution of web size

 # a larger parameter means more small objects

set opt(interpage) 1 ;# interpage parameter for web traffic generator.

#####################################################################

# Plotting statistics.

#####################################################################

set opt(printRTTs) 0 ;# 1 to print link delays

set opt(quiet) 0 ;# popup anything?

set opt(wrap) 90 ;# wrap plots?

set opt(srcTrace) is ;# where to plot traffic

set opt(dstTrace) bs ;# where to plot traffic

#####################################################################

# Link characteristics.

#####################################################################

set opt(delayInt) "" ; # interval of delays

set opt(delayLen) "" ; # length of delays

set opt(allocLenUL) "" ; # delay of allocating a channel in sec in uplink

set opt(allocHoldUL) "" ; # idle time after channel is released in uplink

set opt(allocLenDL) "" ; #

set opt(allocHoldDL) "" ; #

# currently only in downlink

set opt(bwLowLen) 0 ; # length of period of low bandwidth in sec

set opt(bwHighLen) 0 ; # length of period of high bandwidth in sec

set opt(bwScale) 0 ; # factor by which bandwidth is decreased

# currently only in downlink

set opt(reorderLen) 0 ; # reordering delay in sec

set opt(reorderRate) 0 ; # per-packet reordering rate

set opt(errUnit) packet ; # unit of errors, per packet or per bit

set opt(errRateUL) 0 ; # the rate of errors in uplink

set opt(errBurstUL) 0 ; # coefficient 0..1 of how bursty errors are

set opt(errSlotUL) 0 ; # time in sec of low error period

set opt(errRateDL) 0 ; #

set opt(errBurstDL) 0 ; #

set opt(errSlotDL) 0

set opt(vhoTarget) none ; # to which network handover occurs

set opt(vhoTime) 30 ; # when handover occurs

set opt(vhoLoss) 0 ; # fraction of packets lost during handover

set opt(vhoDelay) 0 ; # delay in sec caused by handover

set opt(gprsbuf) 10 ; # buffer size for gprs

set opt(wlan\_duplex\_buf) 10 ; # buffer size for wlan\_duplex

set opt(nodeDist) 2 ; # distance in meters between WLAN nodes

#####################################################################

set opt(tfrcFB) 1 ; #number of feedback reports in TFRC per RTT

proc getopt {argc argv} {

 global opt

 for {set i 0} {$i < $argc} {incr i} {

 set arg [lindex $argv $i]

 if {[string range $arg 0 0] != "-"} continue

 set name [string range $arg 1 end]

 set opt($name) [lindex $argv [expr $i+1]]

 puts "opt($name): $opt($name)"

 }

}

getopt $argc $argv

if {$opt(seed) > -1} {

 puts "Seeding Random number generator with $opt(seed)\n"

 ns-random $opt(seed)

}

Agent/TFRCSink set NumFeedback\_ $opt(tfrcFB)

#default downlink bandwidth in bps

set bwDL(gsm) 9600

set bwDL(gprs) 30000

set bwDL(umts) 384000

set bwDL(geo) 2000000

set bwDL(wlan\_duplex) 650000 ;# 650000

set bwDL(wlan\_ether) 1000000

set bwDL(wlan\_complex) 1000000

#default uplink bandwidth in bps

set bwUL(gsm) 9600

set bwUL(gprs) 10000

set bwUL(umts) 64000

set bwUL(geo) 200000

set bwUL(wlan\_duplex) 650000

#default downlink propagation delay in seconds

set propDL(gsm) .500

set propDL(gprs) .100

set propDL(umts) .150

set propDL(geo) .250

set propDL(wlan\_duplex) .01

set propDL(wlan\_ether) .01 ;#0.01

#default uplink propagation delay in seconds

set propUL(gsm) .500

set propUL(gprs) .100

set propUL(umts) .150

set propUL(geo) .250

set propUL(wlan\_duplex) .01 ;#0.01 0.003

#default buffer size in packets

set buf(gsm) 10

set buf(gprs) $opt(gprsbuf)

set buf(umts) 20

set buf(geo) 20

set buf(wlan\_duplex) $opt(wlan\_duplex\_buf)

set buf(wlan\_ether) 10

set buf(wlan\_complex) 10

#####################################################################

proc cell\_topo {} {

 global ns nodes qm

 $ns duplex-link $nodes(bs) $nodes(ms) 1 1 $qm

 $ns duplex-link $nodes(lp) $nodes(ms) 3Mbps 10ms DropTail

 $ns duplex-link $nodes(bs) $nodes(is) 3Mbps 50ms DropTail

 puts "cell topology"

}

#####################################################################

proc sat\_topo {} {

 global ns nodes qm

 $ns duplex-link $nodes(bs) $nodes(ms) 1 1 $qm

 $ns duplex-link $nodes(lp) $nodes(ms) 3Mbps 10ms DropTail

 $ns duplex-link $nodes(bs) $nodes(is) 3Mbps 50ms DropTail

 puts "sat topology"

}

#####################################################################

proc wlan\_topo\_simple {} {

 global qm ns nodes opt bwDL propDL buf

 switch $opt(type) {

 wlan\_duplex {

 puts "duplex wlan topology"

 $ns duplex-link $nodes(ms) $nodes(bs) 1 1 $qm

 }

 wlan\_ether {

 puts "ethernet wlan topology"

 # Mac/Csma/Ca Mac/802\_3

 set lan [$ns make-lan "$nodes(bs) $nodes(ms)" $bwDL(wlan\_ether) $propDL(wlan\_ether) LL Queue/$qm Mac/802\_3 Channel "Phy/WiredPhy" $buf(wlan\_ether)]

# [$nodes(bs) queue] set limit\_ $buf(wlan\_duplex)

 }

 }

 $ns duplex-link $nodes(lp) $nodes(ms) 100Mbps 1ms $qm

 $ns duplex-link $nodes(bs) $nodes(is) 10Mbps 50ms DropTail

}

#

proc wlan\_topo\_complex {} {

 global ns nodes qm opt buf bwDL

 puts "complex wlan topology"

#Configuration for Orinoco 802.11b 11Mbps PC card with ->22.5m range

#Phy/WirelessPhy set Pt\_ 0.031622777

#Phy/WirelessPhy set bandwidth\_ 11Mb

#Mac/802\_11 set dataRate\_ 11Mb

#Mac/802\_11 set basicRate\_ 1Mb #for broadcast packets

#Phy/WirelessPhy set freq\_ 2.472e9 #channel-13.2.472GHz

#Phy/WirelessPhy set CPThresh\_ 10.0

#Phy/WirelessPhy set CSThresh\_ 5.011872e-12

#Phy/WirelessPhy set L\_ 1.0

#Phy/WirelessPhy set RXThresh\_ 5.82587e-09

 Mac/802\_11 set dataRate\_ $bwDL(wlan\_complex)

 $ns node-config -addressType hierarchical

 AddrParams set domain\_num\_ 2

 lappend cluster\_num 1 1

 AddrParams set cluster\_num\_ $cluster\_num

 lappend eilastlevel 2 2

 AddrParams set nodes\_num\_ $eilastlevel

 set topo [new Topography]

 $topo load\_flatgrid 1000 1000

 # god needs to know the number of all wireless interfaces MN+BS

 create-god 2

 set nodes(bs) [$ns node {0.0.0}]

 set nodes(is) [$ns node {0.0.1}]

 # DSDV DSR TORA AODV

 $ns node-config -adhocRouting DSDV \

 -llType LL \

 -macType Mac/802\_11 \

 -ifqType Queue/$qm \

 -ifqLen $buf(wlan\_complex) \

 -propType "Propagation/TwoRayGround" \

 -antType "Antenna/OmniAntenna" \

 -phyType "Phy/WirelessPhy" \

 -wiredRouting ON \

 -channel [new "Channel/WirelessChannel"] \

 -agentTrace ON \

 -routerTrace OFF \

 -topoInstance $topo \

 -macTrace OFF \

 -movementTrace OFF

 #create base station

 set nodes(ms) [$ns node {1.0.0}]

 $nodes(ms) random-motion 0

 $nodes(ms) set X\_ $opt(nodeDist)

 $nodes(ms) set Y\_ 0.0

 $nodes(ms) set Z\_ 0.0

 #configure for mobilenodes

 $ns node-config -wiredRouting OFF

 set nodes(lp) [$ns node {1.0.1}]

 $nodes(lp) random-motion 0

 $nodes(lp) set X\_ 2.0

 $nodes(lp) set Y\_ 2.0

 $nodes(lp) set Z\_ 0.0

 $nodes(lp) base-station [AddrParams addr2id [$nodes(ms) node-addr]]

 $ns duplex-link $nodes(ms) $nodes(bs) 10Mbps 30ms DropTail

 $ns duplex-link $nodes(bs) $nodes(is) 10Mbps 50ms DropTail

 # establish route

 set ping9 [$ns create-connection Ping $nodes(is) Ping $nodes(lp) 9]

 $ping9 oneway

# $ns after 0.001 "$ping9 send"

}

#

proc set\_link\_params {t} {

 global ns nodes bwUL bwDL propUL propDL buf

 $ns bandwidth $nodes(bs) $nodes(ms) $bwDL($t) simplex

 $ns bandwidth $nodes(ms) $nodes(bs) $bwUL($t) simplex

 $ns delay $nodes(bs) $nodes(ms) $propDL($t) simplex

 $ns delay $nodes(ms) $nodes(bs) $propDL($t) simplex

 $ns queue-limit $nodes(bs) $nodes(ms) $buf($t)

 $ns queue-limit $nodes(ms) $nodes(bs) $buf($t)

}

#remove-all-packet-headers ; # removes all except common

#add-packet-header Flags IP TCP ; # hdrs reqd for TCP

if {$opt(queue) == "DT"} {

 set qm DropTail

}

if {$opt(queue) == "RED"} {

 set qm RED

}

set tcpTick\_ 0.01

set pktsize 1460

Agent/TCP set tcpTick\_ $tcpTick\_

set out $opt(title)

proc stop {} {

 global nodes opt

 set wrap $opt(wrap)

 set sid [$nodes($opt(srcTrace)) id]

 set did [$nodes($opt(dstTrace)) id]

 if {$opt(srcTrace) == "bs"} {

 set a "-a out.tr"

 } else {

 set a "out.tr"

 }

 set GETRC "../../../bin/getrc"

 set RAW2XG "../../../bin/raw2xg"

 exec $GETRC -s $sid -d $did -f 0 out.tr | \

 $RAW2XG -s 0.01 -m $wrap -r > plot.xgr

 exec $GETRC -s $did -d $sid -f 0 out.tr | \

 $RAW2XG -a -s 0.01 -m $wrap >> plot.xgr

 exec $GETRC -s $sid -d $did -f 1 out.tr | \

 $RAW2XG -s 0.01 -m $wrap -r >> plot.xgr

 exec $GETRC -s $did -d $sid -f 1 out.tr | \

 $RAW2XG -s 0.01 -m $wrap -a >> plot.xgr

 exec ./xg2gp.awk plot.xgr

 if {!$opt(quiet)} {

 exec xgraph -bb -tk -nl -m -x time -y packets plot.xgr &

 }

 exit 0

}

#

proc pingSend {ag int} {

 global ns

 $ag send

 $ns after $int "pingSend $ag $int"

}

#Define a 'recv' function for the class 'Agent/Ping'

Agent/Ping instproc recv {from seq ff bf} {

 $self instvar node\_

 global rttf

# puts "node [$node\_ id] received ping answer from \

 $from seq $seq with forward delay $ff and backward $bf ms."

 puts $rttf "$seq [expr $ff/1000] [expr $bf/1000]"

}

#

source web.tcl

#

proc insertDelay {} {

 global ns dl\_dist di\_dist delayerUL delayerDL

 $delayerUL block

 $delayerDL block

 set len [$dl\_dist value]

 $ns after $len "$delayerUL unblock"

 $ns after $len "$delayerDL unblock"

 set next [expr $len + [$di\_dist value]]

 $ns after $next "insertDelay"

# puts "[$ns now]: delay for $len, next after $next"; flush stdout

}

#

proc bwOscilate {bw\_up} {

 global ns opt nodes

 set dl\_link [[$ns link $nodes(bs) $nodes(ms)] link]

 set bw [$dl\_link set bandwidth\_]

 if {$bw\_up} {

 set bw [expr $bw / $opt(bwScale)]

 set bw\_up 0

 set next $opt(bwLowLen)

 } else {

 set bw [expr $bw \* $opt(bwScale)]

 set next $opt(bwHighLen)

 set bw\_up 1

 }

 $dl\_link set bandwidth\_ $bw

 $ns after $next "bwOscilate $bw\_up"

 puts "[$ns now]: new bandwidth $bw, next after $next"; flush stdout

}

#

proc setError {low errmodel rate slot burst} {

 if {$low} {

 set newrate [expr $rate / $burst ]

 set next [expr $slot \* $burst]

 } else {

 set newrate $rate

 set next $slot

 }

 puts "setError: newrate $newrate slot $slot low $low"

 $errmodel set rate\_ $newrate

 ns after $next "setError [expr !$low] $errmodel $rate $slot $burst]"

}

# perform vertical handover

proc makeVho {} {

 global ns opt nodes delayerUL delayerDL buf

 if {$opt(vhoLoss)} {

 $ns queue-limit $nodes(bs) $nodes(ms) [expr $buf($opt(type))\*(1-$opt(vhoLoss))]

 $ns queue-limit $nodes(ms) $nodes(bs) [expr $buf($opt(type))\*(1-$opt(vhoLoss))]

 [[$ns link $nodes(bs) $nodes(ms)] queue] shrink-queue

 [[$ns link $nodes(ms) $nodes(bs)] queue] shrink-queue

 }

 set\_link\_params $opt(vhoTarget)

 $delayerUL block; $delayerDL block

 $ns after $opt(vhoDelay) "$delayerUL unblock; $delayerDL unblock"

}

proc parseDist {s} {

 set d ""

 set k [scan $s "%c(%f,%f)" dist arg1 arg2]

 if { $k != 3 } {

 set k [scan $s "%c(%f)" dist arg1]

 if { $k != 2 } {

 puts "wrong distribution $s"

 exit 1

 }

 }

 switch $dist {

 85 {

 set d [new RandomVariable/Uniform]

 $d set min\_ $arg1

 $d set max\_ $arg2

 }

 69 {

 set d [new RandomVariable/Exponential]

 $d set avg\_ $arg1

 }

 default {

 puts "unkown distribution"

 exit 1

 }

 }

# puts "$arg1 $arg2"

 return $d

}

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* MAIN \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

set ns [new Simulator]

variable delayerUL

variable delayerDL

#

# RED and TCP parameters

#

if {$opt(ecn) == 1} {

 Queue/RED set setbit\_ true

}

Queue/RED set summarystats\_ true

Queue/DropTail set summarystats\_ true

Queue/RED set adaptive\_ $opt(adaptive)

Queue/RED set q\_weight\_ 0.0

Queue/RED set thresh\_ $opt(minth)

Queue/RED set maxthresh\_ $opt(maxth)

Queue/DropTail set shrink\_drops\_ true

Agent/TCP set ecn\_ $opt(ecn)

Agent/TCP set packetSize\_ $pktsize

Agent/TCP set window\_ $opt(window)

DelayLink set avoidReordering\_ true

# Create trace, rttf is for latency measured with ping

set tf [open out.tr w]

set rttf [open rtt.tr w]

$ns trace-all $tf

# Complex WLAN needs nodes with hierarhical routing and has no params

if {$opt(type) != "wlan\_complex"} {

 set nodes(lp) [$ns node]

 set nodes(ms) [$ns node]

 set nodes(bs) [$ns node]

 set nodes(is) [$ns node]

 # Overwrite defaults from command line params

 if {$opt(bwUL)} {set bwUL($opt(type)) $opt(bwUL)}

 if {$opt(bwDL)} {set bwDL($opt(type)) $opt(bwDL)}

 if {$opt(propUL)} {set propUL($opt(type)) $opt(propUL)}

 if {$opt(propDL)} {set propDL($opt(type)) $opt(propDL)}

}

if {$opt(qsize)} {set buf($opt(type)) $opt(qsize)}

# Create topology

switch $opt(type) {

 gsm -

 gprs -

 umts {cell\_topo}

 geo {sat\_topo}

 wlan\_complex {wlan\_topo\_complex}

 wlan\_duplex -

 wlan\_ether {wlan\_topo\_simple}

}

if {$opt(type) != "wlan\_complex" && $opt(type) != "wlan\_ether"} {

 set\_link\_params $opt(type)

 set delayerDL [new Delayer]

 set delayerUL [new Delayer]

# $delayerDL set debug\_ true

# $delayerUL set debug\_ true

 $ns insert-delayer $nodes(ms) $nodes(bs) $delayerUL

 $ns insert-delayer $nodes(bs) $nodes(ms) $delayerDL

}

#$ns trace-queue $nodes(is) $nodes(bs) $tf

#$ns trace-queue $nodes(bs) $nodes(is) $tf

#

# Set up forward TCP connection

#

if {$opt(flows) > 0} {

 set tcp1 [$ns create-connection TCP/Sack1 $nodes(is) TCPSink/Sack1 $nodes(lp) 0]

 set ftp1 [[set tcp1] attach-app FTP]

 $ns at 0.8 "[set ftp1] start"

}

if {$opt(shortflows) > 0} {

 set tcp1 [$ns create-connection TCP/Sack1 $nodes(is) TCPSink/Sack1 $nodes(lp) 0]

 set ftp1 [[set tcp1] attach-app FTP]

 $tcp1 set window\_ 100

 $ns at 0.0 "[set ftp1] start"

 $ns at 3.5 "[set ftp1] stop"

 set tcp2 [$ns create-connection TCP/Sack1 $nodes(is) TCPSink/Sack1 $nodes(lp) 0]

 set ftp2 [[set tcp2] attach-app FTP]

 $tcp2 set window\_ 3

 $ns at 1.0 "[set ftp2] start"

 $ns at 8.0 "[set ftp2] stop"

}

#

# Set up forward TFRC connection

#

if {$opt(flowsTfrc) > 0} {

 set tfrc1 [$ns create-connection TFRC $nodes(is) TFRCSink $nodes(lp) 0]

 set ftp3 [[set tfrc1] attach-app FTP]

 $ns at 0.1 "[set ftp3] start"

}

#

# Set up ping for delay measurement

#

if {$opt(pingInt)} {

 set ping1 [$ns create-connection Ping $nodes(is) Ping $nodes(lp) 0]

 $ping1 oneway ; # enable ping extensions

 $ns after 1 "pingSend $ping1 $opt(pingInt)"

}

#

# Traffic on the reverse path.

#

# Reverse-path traffic is half the number of flows as for the forward path.

if {$opt(flowsRev) > 0} {

 set tcp2 [$ns create-connection TCP/Sack1 $nodes(lp) TCPSink/Sack1 $nodes(is) 1]

 set ftp2 [[set tcp2] attach-app FTP]

 $ns at 2 "[set ftp2] start"

}

#

# Add forward web traffic.

#

set req\_trace\_ 0

set count $opt(webers)

if ($count) {

 add\_web\_traffic $opt(secondDelay) $opt(web) $opt(interpage) $opt(pagesize) $opt(objSize) $opt(shape) 1

 add\_web\_traffic $opt(secondDelay) [expr $opt(web)/2] $opt(interpage) $opt(pagesize) $opt(objSize) $opt(shape) 0

}

#

# Set up channel allocation delay

#

# Downlink

if {$opt(allocLenDL) != "" && $opt(allocHoldDL) != ""} {

 set al\_dl [parseDist $opt(allocLenDL)]

 set ah\_dl [parseDist $opt(allocHoldDL)]

 $delayerDL alloc $ah\_dl $al\_dl

}

# Uplink

if {$opt(allocLenUL) != "" && $opt(allocHoldUL) != ""} {

 set al\_ul [parseDist $opt(allocLenUL)]

 set ah\_ul [parseDist $opt(allocHoldUL)]

 $delayerUL alloc $ah\_ul $al\_ul

}

#

# Set up delay variation (due to ARQ or delay)

#

if {$opt(delayInt) != "" && $opt(delayLen) != ""} {

 set di\_dist [parseDist $opt(delayInt)]

 set dl\_dist [parseDist $opt(delayLen)]

 $ns after [$di\_dist value] "insertDelay"

}

#

# Set up bandwidth oscillation

#

if {$opt(bwLowLen) && $opt(bwHighLen) && $opt(bwScale)} {

 $ns after $opt(bwHighLen) "bwOscilate 1"

}

#

# Set up reordering

#

if {$opt(reorderLen) && $opt(reorderRate)} {

 ErrorModel set delay\_pkt\_ true

 ErrorModel set drop\_ false

 ErrorModel set delay\_ $opt(reorderLen)

# set errmodelDL [new ErrorModel]

# $errmodelDL set rate\_ $opt(errRateDL)

# $errmodelDL set unit\_ $opt(errUnit)

# $ns lossmodel $errmodelDL $nodes(bs) $nodes(ms)

 set em [new ErrorModule Fid]

 [$ns link $nodes(bs) $nodes(ms)] errormodule $em

 set errmodel [new ErrorModel/Uniform $opt(reorderRate)]

 $errmodel unit pkt

 $em insert $errmodel

 $em bind $errmodel 0

}

#

# Set up error losses (bursty or uniform)

#

if {$opt(errRateUL)} {

 set errmodelUL [new ErrorModel]

 $errmodelUL set rate\_ $opt(errRateUL)

 $errmodelUL set unit\_ $opt(errUnit)

 $ns lossmodel $errmodelUL $nodes(ms) $nodes(bs)

 $ns after $opt(errSlotUL) "setError 1 $errmodelUL $opt(errRateUL) $opt(errSlotUL) $opt(errBurstUL)"

}

if {$opt(errRateDL)} {

 set errmodelDL [new ErrorModel]

 $errmodelDL set rate\_ $opt(errRateDL)

 $errmodelDL set unit\_ $opt(errUnit)

 $ns lossmodel $errmodelDL $nodes(bs) $nodes(ms)

 $ns after $opt(errSlotDL) "setError 1 $errmodelDL $opt(errRateDL) $opt(errSlotDL) $opt(errBurstDL)"

}

## set emL [new ErrorModule Fid]

## set linkL [$ns link $nodes(bs) $nodes(ms)]

## set errmodelL [new ErrorModel/List]

## $errmodelL droplist {5}

## $linkL errormodule $emL

## #$emL insert $errmodelL

## #$emL bind $errmodelL 1

#

# Set up vertical handover

#

if {$opt(vhoTarget) != "none"} {

 set hoTime $opt(vhoTime)

 $ns after $hoTime makeVho

}

$ns at $opt(stop) "stop"

$ns run