The first step of our solution is creating the .tcl file. We create nn mobile nodes and use UDP for the packet transmission. This portion looks like -

```
for {set i 0} {$i < $val(nn)} {incr i} {
        set audp($i) [new Agent/UDP]
        $ns_ attach-agent $node_($i) $audp($i)
        set acbr($i) [new Application/Traffic/CBR]
        $acbr($i) attach-agent $audp($i)
        $ns_ connect $audp($i) $sink
        $acbr($i) set interval_ 0.02
        $acbr($i) set packetSize_ 16
        set start_time [$rndnum value]
        puts "$start_time time"
        $ns_ at $start_time "$acbr($i) start"
        $ns_ at $val(end_time) "$acbr($i) stop"
}
```

One node is always assigned as the sink and all other nodes are attached to it. In our case, we made the modifications directly in the wireless-simple-mac file.

Once this is done, we also need to make modifications to the mac-simple.h and .cc files. For the .h file, our modifications are the following -

```
public:
          MacSimple();
        void recv(Packet *p, Handler *h);
        void send(Packet *p, Handler *h);
->       void handle(Event *p);
        void waitHandler(void);
        void sendHandler(void);
        void recvHandler(void);
        double txtime(Packet *p);

        // Added by Sushmita to support event tracing (singal@nunki.usc.edu)
        void trace_event(char *, Packet *);
        int command(int, const char*const*);
        EventTrace *et_;

private:
        Packet *        pktRx_;
        Packet *        pktTx_;
        MacState        rx_state_;// incoming state (MAC_RECV or MAC_IDLE)
        MacState        tx_state_;// outgoing state
        int             tx_active_;
        int             fullduplex_mode_;
```

```
        Handler * txHandler_;
        MacSimpleWaitTimer *waitTimer;
        MacSimpleSendTimer *sendTimer;
        MacSimpleRecvTimer *recvTimer;

        int busy_ ;
```

Declaring repeat and interval variables.

```
->      int repeat_;
->      double interval_;
```

We make the following modifications to enable retransmissions of packets at the receiver's side -

Handle function for handling retransmission.

```
void MacSimple::handle(Event *p)
{
        downtarget_->recv((Packet*)p, txHandler_);
}

MacSimple::MacSimple() : Mac() {
        rx_state_ = tx_state_ = MAC_IDLE;
        tx_active_ = 0;
        waitTimer = new MacSimpleWaitTimer(this);
        sendTimer = new MacSimpleSendTimer(this);
        recvTimer = new MacSimpleRecvTimer(this);
        // Added by Sushmita to support event tracing (singal@nunki.usc.edu)
        et_ = new EventTrace();
        busy_ = 0;
        bind("fullduplex_mode_", &fullduplex_mode_);
```

Binding the new variables declared.

```
                                        bind("interval_", &interval_);
        bind("repeatTx_", &repeat_);
}

void MacSimple::send(Packet *p, Handler *h)
{
        double last_packet_time = 0;
        double* packet_intervals = new double[repeat_];

        Scheduler& s = Scheduler::instance();

        hdr_cmn* ch = HDR_CMN(p);
        /* store data tx time */
        ch->txtime() = Mac::txtime(ch->size());
```

Get the Maximum Packet Interval time.

```
        // Added by Sushmita to support event tracing (singal@nunki.usc.edu)
        trace_event("SENSING_CARRIER",p);
        for(int i=0; i<repeat_; i++) {
                packet_intervals[i] = (rand()%40)/40.0 * interval_;
                if(last_packet_time < packet_intervals[i]) {
                  last_packet_time = packet_intervals[i];
                }
        }

Schedule for retransmission.
for(int i=0;i<repeat_;i++) {
        if(packet_intervals[i]!=last_packet_time) {
                s.schedule(this, (Event*)p->copy(), packet_intervals[i]);
        }
}
waitTimer->restart(last_packet_time);

        /* check whether we're idle */
        if (tx_state_ != MAC_IDLE) {
                // already transmitting another packet .. drop this one
                // Note that this normally won't happen due to the queue
                // between the LL and the MAC .. the queue won't send us
                // another packet until we call its handler in sendHandler()
Packet::free(p);
                return;
        }

           pktTx_ = p;
        txHandler_ = h;
        // rather than sending packets out immediately, add in some
        // jitter to reduce chance of unnecessary collisions
        double jitter = Random::random()%40 * 100/bandwidth_;

        if(rx_state_ != MAC_IDLE) {
                trace_event("BACKING_OFF",p);
        }

           if (rx_state_ == MAC_IDLE ) {
                // we're idle, so start sending now
                // waitTimer->restart(jitter);
                // sendTimer->restart(jitter + ch->txtime());
                sendTimer->restart(last_packet_time + ch->txtime());
        } else {
                // we're currently receiving, so schedule it after
                // we finish receiving
                waitTimer->restart(jitter);
```

```
                                                        sendTimer->restart(jitter +
ch->txtime()
                        + HDR_CMN(pktRx_)->txtime());
        }
}
```