

Timer Usage (In NS2)

AODV method:

1. define your timer in the *.h* file such as in the *aodv.h* file

```
class PrintTimer : public Handler {
public:
    PrintTimer(AODV* a) : agent(a) {}
    void handle(Event*);
private:
    AODV *agent;
    Event intr;
};
```

2. declare the timer in your *routing agent* in the *.h* file

```
class AODV : public Agent {
    .....
    .....
    PrintTimer ptimer;;
    ....
    friend class PrintTimer;;
    ....
};
```

3. In the *aodv.cc* file, implement your *handle* function which will do what you want to do when the timer expires

```
void PrintTimer::handle(Event*) {
    fprintf(stderr, "This is a test for the usage of
timer.\n");
    //if you want to schedule this timer periodically
    //#define DEFINED_DELAY 1.0 //sec
    // Scheduler::instance().schedule(this, &intr,
```

Another method:

1. define your timer in your *.h* file such as in my *sensor-agent.h* file

```
class WakeupTimer : public TimerHandler {
public:
    WakeupTimer(SensorAgent* agent) :
TimerHandler() { agent_ = agent; }
    void expire(Event* event);
private:
    SensorAgent*    agent_;
};
```

2. declare the timer in your *agent* as in my *sensorAgent* in the *.h* file

```
class sensorAgent : public Agent {
    .....
    .....
    WakeupTimer
wakeupTimer_;
    .....
    friend class WakeupTimer;
    ....
};
```

3. In *sensor-agent.cc* file, implement your *expire* function which will do what you want to do after the delay

```
void WakeupTimer::expire(Event* event) {
    // wakeup timer has expired. switch state to probing
and determine if
    // this node needs to start working.
    agent_->switchState(SENSOR_PROBING);
```

```
DEFINED_DELAY);
```

```
}
```

4. remember to initialize it

```
AODV::AODV(nsaddr_t id) : ..., ptimer(this), ... {
```

```
}
```

5. Then you can use it.

For example, in

```
void
```

```
AODV::recvReply(Packet *p) {
```

```
....
```

```
....
```

```
//when it receives a reply, schedule the print timer
for 0.5 s
```

```
//Scheduler::instance().schedule(&ptimer,
p->copy(), 0.5); //if want to process the packet
```

```
Scheduler::instance().schedule(&ptimer, new
Event(), 0.5); //nothing needs to be processed
```

```
// you can also directly call the handle function,
no delay
```

```
//ptimer.handle((Event*) 0);
```

```
// or
```

```
//Scheduler::instance().schedule(&ptimer, new
Event(), 0.0);
```

```
....
```

```
....
```

```
}
```

```
}
```

4. remember to initialize it

```
sensorAgent::sensorAgent() : ..., wakeupTimer_(this),
```

```
... {
```

```
}
```

5. Then you can use it

```
wakeupTimer_.sched(sleeping_duration);
```

```
wakeupTimer_.cancel();
```

With stop, restart

in .h

```
class helloTimer : public Handler {
```

```
public:
```

```
helloTimer(OppFwd* a) : agent(a) { busy_ =
0; }
```

```
void handle(Event*);
```

```
void start(double time);
```

```
void stop(void);
```

```
inline int busy(void) { return busy_; }
```

```
private:
```

```
OppFwd *agent;
```

```
Event intr;
```

```
int busy_;
```

```
};  
  
in .cc  
  
void helloTimer::handle(Event*) {  
  
    busy_ = 0;  
    agent->sendHello();  
    double interval = MinHelloInterval +  
((MaxHelloInterval - MinHelloInterval) *  
Random::uniform());  
    assert(interval >= 0);  
  
    Scheduler::instance().schedule(this, &intr,  
interval);  
}  
  
void helloTimer::start(double time) {  
    Scheduler &s = Scheduler::instance();  
    assert(busy_ == 0);  
  
    busy_ = 1;  
    s.schedule(this, &intr, time);  
}  
  
void helloTimer::stop(void) {  
    Scheduler &s = Scheduler::instance();  
    assert(busy_);  
  
    s.cancel(&intr);  
  
    busy_ = 0;  
}
```

1. "Couldn't schedule timer", check common/timer-handler.cc

[RETURN](#)