

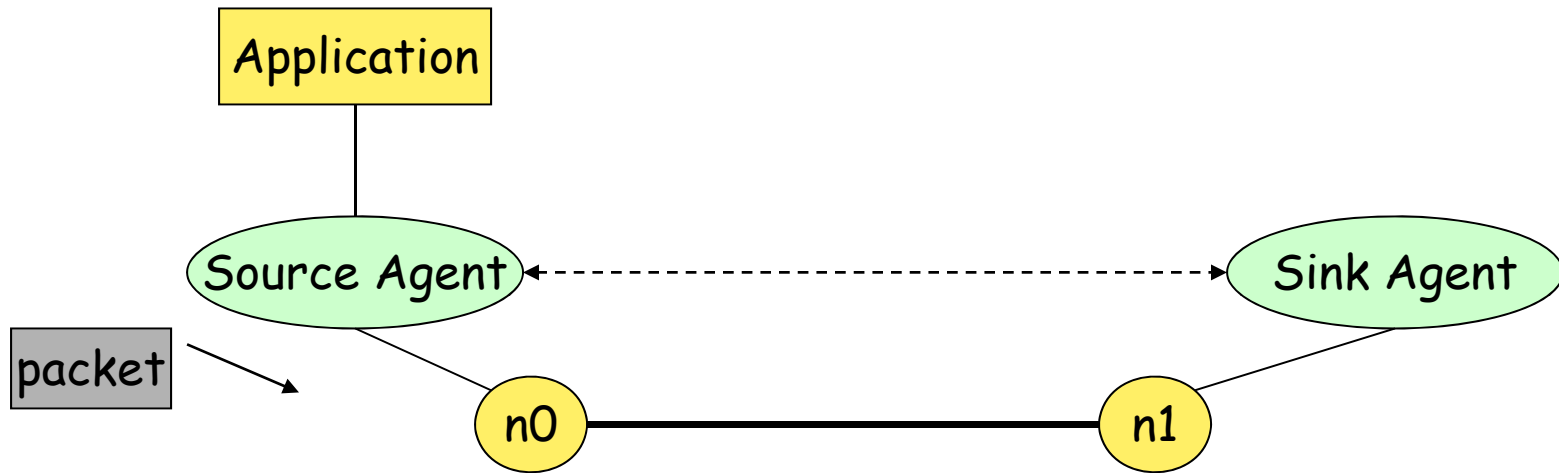
Application: User Demand Indicator



Outline

- Application Overview
- Two Derived Classes
 - Traffic Generator
 - Simulated Application
- Summary

A Two Node Network



Agent:
Packet Creation and
Destruction

Link:
Connecting Nodes
Buffer Management

Node:
A computer host +
A router

Application:
Demand Indication

Application

- Indicate user demand
- Two NS2 Application
 1. Traffic generator
 - Conform to a predefined schedule
 - E.G., CBR
 2. Simulated applications
 - Create demand as if the application is running
 - E.G., FTP



NS2 Implementation

- C++ Class Application
- Main variables: agent_

```
//~/ns/apps/app.h
class Application : public Process {
public:
    Application();
    virtual void send(int nbytes);
    virtual void recv(int nbytes);
    virtual void resume();
protected:
    virtual void start();
    virtual void stop();
    Agent *agent_;
    int enableRecv_;int enableResume_;
};
```

Functions of Class Application

- 3 main **public** functions
- `send(nbytes)`:
 - Tell TL to send out `nbytes` of data.

```
//~/ns/apps/app.cc  
void Application::send(int nbytes)  
{  
    agent_->sendmsg(nbytes);  
}
```
- `recv(nbytes)`:
 - Be informed that `nbytes` are received.
 - Invoked by an Agent object.
- `resume()`:
 - Invoked by sending agent when it has sent out all its packets

Functions of Class Application

- 2 main **protected** functions
- No implementation
- To be overridden by the derived classes

- `start()`: the application
- `stop()`: stop the application



OTcl Commands of Class Application

OTcl Command	Meaning
<code>start { }</code>	Invoke <code>start ()</code>
<code>stop { }</code>	Invoke <code>stop ()</code>
<code>agent { }</code>	Return the name of the attached agent
<code>send {nbytes }</code>	Ask the attached agent to send out <code>nbytes</code> of data
<code>attach-agent {agent }</code>	Create a two-way connection between itself and <code>agent</code> .

Outline

- Application Overview
- Two Derived Classes
 - Traffic Generator
 - Simulated Application
- Summary

Traffic Generator

- Create packets based on a pre-defined schedule
- Example
 - Constant bit rate
 - Exponential ON/OFF:
 - CBR During ON; Stop during OFF
 - On/Off period is exp. Distb.
 - Pareto On/OFF
 - CBR During ON; Stop during OFF
 - On/Off period is exp. Distb.
 - Traffic Trace

Traffic Generator

- An abstract C++ class TrafficGenerator

```
//~/ns/tools/trafgen.h (1)
class TrafficGenerator : public Application {
public:
    TrafficGenerator(); (2)
    virtual double next_interval(int &) = 0;
    virtual void init() {}
    virtual double interval() { return 0; }
    virtual int on() { return 0; }; virtual void timeout();
    virtual void recv() {}; virtual void resume() {}
protected: (3)
    virtual void start(); virtual void stop();
    double nextPkttime_; int size_; (4)
    int running_; TrafficTimer timer_;
};
```



Traffic Generator

- Key Variables

Variable	Meaning
<code>timer_</code>	A <code>TrafficTimer</code> object, which determines when a new burst of payload is created.
<code>nextPkttime_</code>	Simulation time that the next payload will be passed to the attached transport layer agent
<code>size_</code>	Application payload size
<code>running_</code>	True if the <code>TrafficGenerator</code> object is running

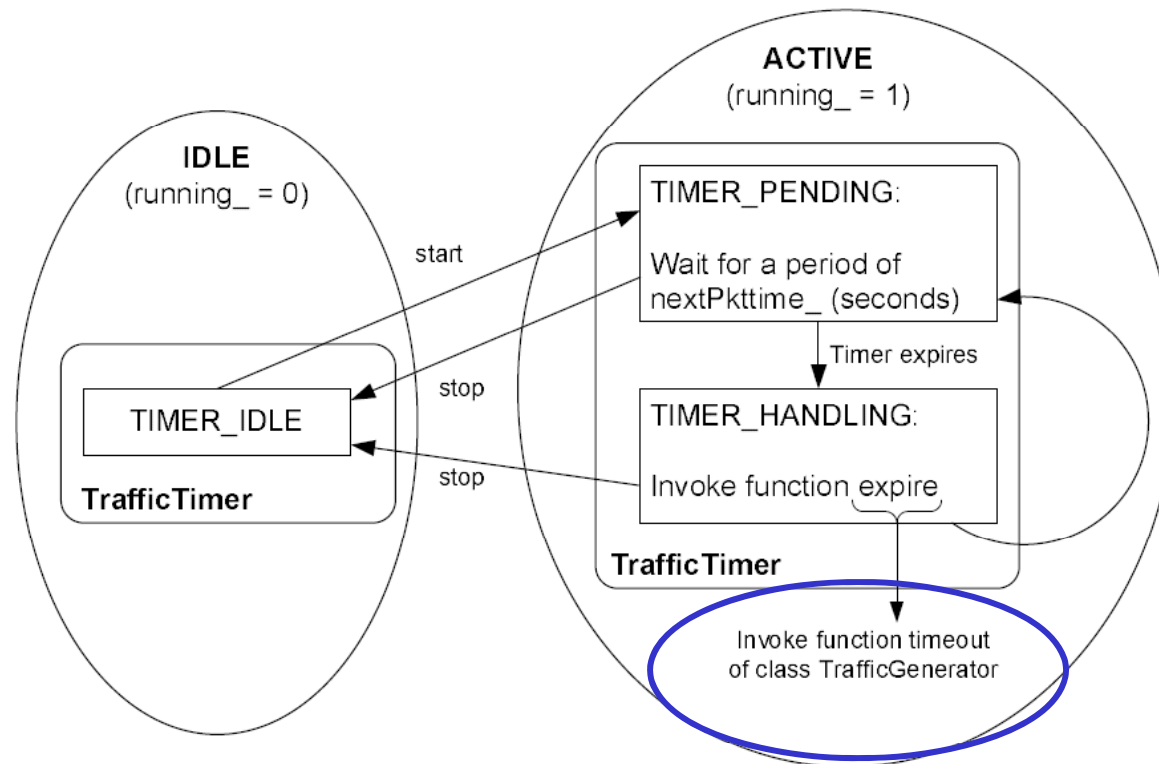
Traffic Generator

- New functions

Variable	Meaning
<code>next_interval(size)</code>	<ul style="list-style-type: none">- Pure virtual- Input = size- Return the time till the next payload is generated.
<code>init()</code>	<ul style="list-style-type: none">- Initializes the traffic generator- Does nothing in this class.
<code>timeout()</code>	<ul style="list-style-type: none">- Sends a user payload to the attached application- Restart <code>timer_</code>.- Invoked when <code>timer_</code> expires

Traffic Timer

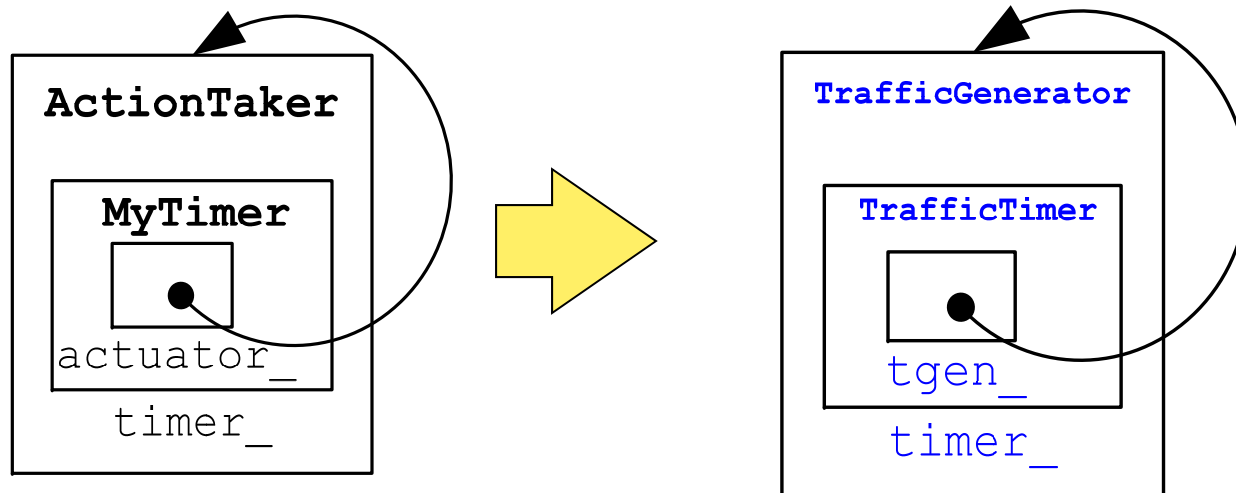
- Model packet generation schedule



Generate a user demand

Traffic Timer

- Variable `timer_` of `TrafficGenerator`
- C++ Class `TrafficTimer`



Traffic Timer

- C++ Class TrafficTimer

```
//~/ns/tools/trafgen.h
class TrafficTimer : public TimerHandler {
public:
    TrafficTimer(TrafficGenerator* tg) : tgen_(tg) {}
protected:
    void expire(Event*);
    TrafficGenerator* tgen_;
};
```

```
//~/ns/tools/trafgen.cc
void TrafficTimer::expire(Event *)
{
    tgen_->timeout();
}

TrafficGenerator::TrafficGenerator() :
    nextPkttime_(-1), running_(0), timer_(this)
{}
```

Traffic Generator

- **Function** TrafficGenerator::timeout()

```
//~/ns/tools/trafgen.cc
void TrafficGenerator::timeout()
{
    if (! running_)
        return;
    send(size_);
    nextPkttime_ = next_interval(size_);
    if (nextPkttime_ > 0)
        timer_.resched(nextPkttime_);
    else
        running_ = 0;
}
```



Constant Bit Rate

- Derived from class `TrafficGenerator`
- C++ class `CBRTrafficClass`
- OTcl class `Application/Traffic/CBR`
- Create a payload of `size_ bytes` for every fixed (or random) interval
- Instvars

Instvar	Default Value	Description
<code>packetSize_</code>	210	Application payload size in bytes
<code>rate_</code>	488×10^3	Sending rate in bps
<code>random_</code>	0 (false)	If true, introduce a random time (either positive or negative) to the inter-burst transmission interval.
<code>maxpkts_</code>	16^7	Maximum number of application payload that CBR can send

Constant Bit Rate

- Override function `next_interval(size)`

```
//~/ns/tools/cbr-traffic.cc
double CBR_Traffic::next_interval(int& size)
{
    interval_ = (double)(size_ << 3) / (double)rate_;
    double t = interval_;
    if (random_)
        t += interval_ * Random::uniform(-0.5, 0.5);
    size = size_;
    if (++seqno_ < maxpkts_)
        return(t);
    else
        return(-1);
}
```

Outline

- Application Overview
- Two Derived Classes
 - Traffic Generator
 - Simulated Application
- Summary

Simulated Application

- No predefined schedule
- Act as if the application is running
- Example: FTP and Telnet
- Let focus on FTP



File Transfer Protocol

- Need no input file
- Tell the attached agent of the file size
- Implemented in OTcl domain only.
- `Application` → `Application/FTP`



File Transfer Protocol

- Key instprocs

Instproc	Meaning
<code>attach-agent {agent}</code>	Attach itself to agent
<code>start{}</code>	Start send a file with infinite size (by executing <code>send -1</code>).
<code>stop{}</code>	Stop the pending file transfer session.
<code>send{nbytes}</code>	Start send a file with infinite size <code>nbytes</code> (by invoking <code>sendmsg(nbytes)</code> of the attached agent).



Summary

- Applications: The main purpose =
()
- Two derived classes
 - Traffic generator (e.g.,)
 - Simulated application (e.g.,)

