Application: User Demand Indicator

Outline

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

A Two Node Network

Application

Source Agent

Packet

Sink Agent

Node:
A computer host +
A router

Link:
Connecting Nodes
Buffer Management

Agent:
Packet Creation and
Destruction

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_

```cpp
//~/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.
    ```c++
    //~/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

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

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

class TrafficGenerator : public Application {
public:
  TrafficGenerator();
  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:
  virtual void start(); virtual void stop();
  double nextPkttime_; int size_; int running_; TrafficTimer timer_;
};

Traffic Generator

• Key Variables

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

# Traffic Generator

- **New functions**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Meaning</th>
</tr>
</thead>
</table>
| next_interval(size) | - Pure virtual  
                        - Input = size  
                        - Return the time till the next payload is generated. |
| init()              | - Initializes the traffic generator  
                        - Does nothing in this class. |
| timeout()           | - Sends a user payload to the attached application  
                        - Restart timer.  
                        - Invoked when timer expires |
Traffic Timer

• Model packet generation schedule

Traffic Timer

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

Traffic Timer

- **C++ Class** TrafficTimer

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

```cpp
//~/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**

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

Constant Bit Rate

- **Override function** `next_interval(size)`

```cpp
//~/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**

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

Summary

• Applications: The main purpose =

• Two derived classes
  - Traffic generator (e.g., )
  - Simulated application (e.g., )