

# Related Helper Classes



# Outline

- Timers
- Random Number Generator
- Built-in Error Model
- Bit Operations
- Summary

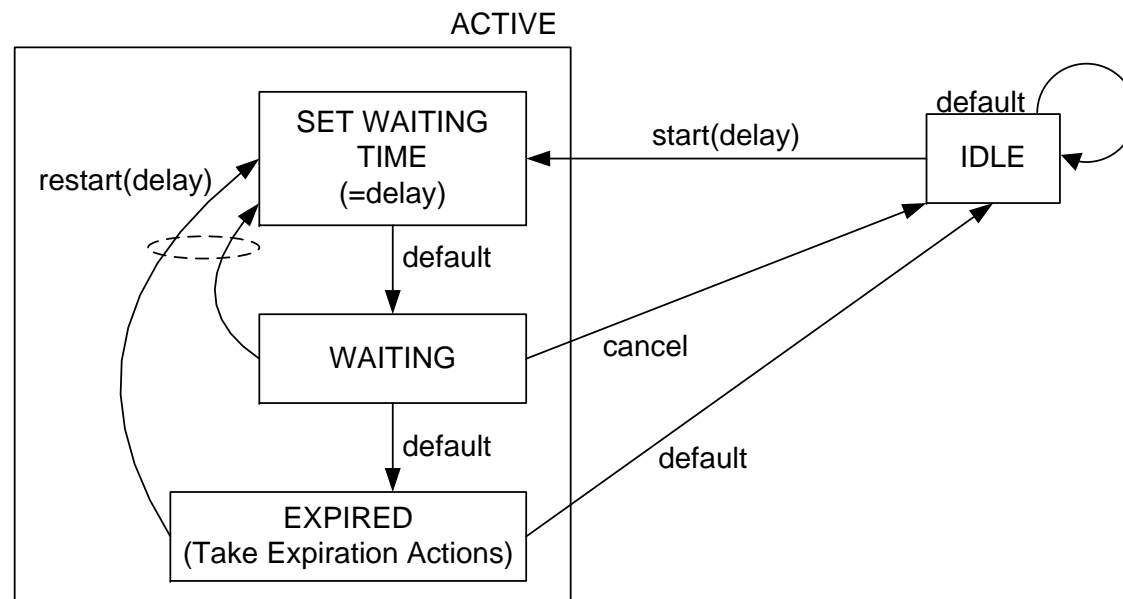
# Timers: Outline

- Overview
- OTcl implementation
- C++ implementation
- Guideline to Implement a New Type of Timer



# Timers

- Implement a waiting process:
  - Wait until the timer expires (i.e., counts to zero).
  - Take expiration actions.



# Timers

- Three main components: Common to all timers  
→ Base Class

1. Waiting mechanism

2. Interface to start, restart, and cancel the timer

3. Expiration actions

Unique to each timer → Derived class

- NS2 Implementation
  - Both OTcl and C++
  - Standalone; Not bound together

# OTcl Timers

- OTcl class `Timer`
- 1. Waiting mechanism → `Simulator::at{...}`
- 2. Interface to
  - `start` → `sched{delay}`,
  - `restart` → `resched{delay}`, and
  - `cancel` → `cancel{}`, `destroy{}`
- 3. Expiration action
  - `Instproc timeout{}`
  - Defined in the derived class



# OTcl Timers

- C++ class TimerHandler

```
//~/ns/tcl/mcast/timer.tcl
Class Timer
Timer instproc init { ns } {
    $self set ns_ $ns
}
Timer instproc sched delay {
    $self instvar ns_
    $self instvar id_
    $self cancel
    set id_ [$ns_ after $delay "$self timeout"]
}
//~/ns/tcl/lib/ns-lib.tcl
Simulator instproc after {ival args} {
    eval $self at [expr [$self now] + $ival] $args
```

A Simulator object

# OTcl Timers

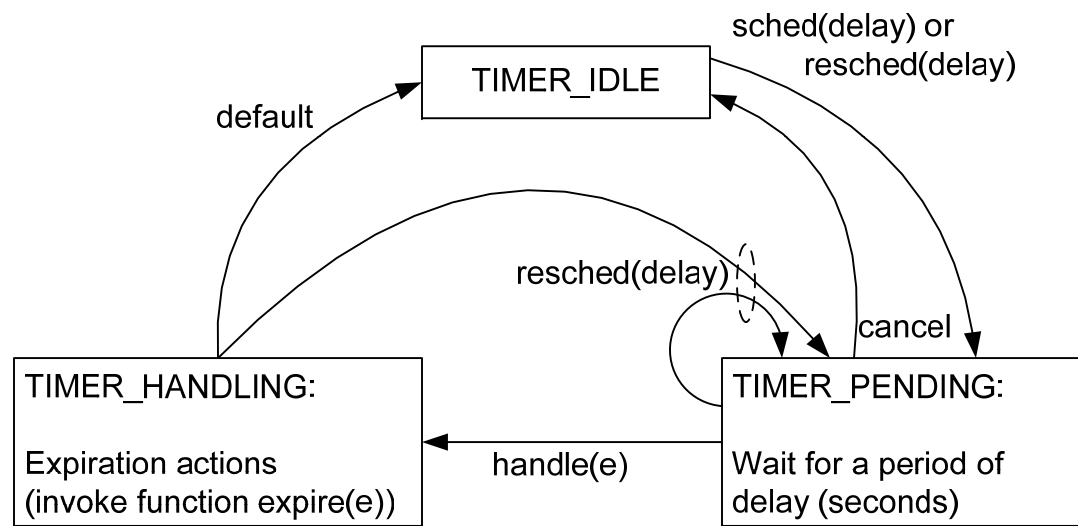
- Instproc `timeout` is defined in the derived classes
- Instprocs `resched`, `cancel`, and `destroy`

```
//~/ns/tcl/mcast/timer.tcl
Timer instproc resched delay {
    $self sched $delay
}
```

```
//~/ns/tcl/mcast/timer.tcl
Timer instproc cancel {} {
    $self instvar ns_
    $self instvar id_
    if [info exists id_] {
        $ns_ cancel $id_
        unset id_
    }
}
Timer instproc destroy {} {
    $self cancel
}
```

# C++ Timers

- C++ Timer Life Cycle



- Tree main states:

{TIMER\_IDLE, TIMER\_PENDING, TIMER\_HANDLING}

# C++ Timers

- C++ class TimerHandler

```
//~/ns/common/timer-handler.h
class TimerHandler : public Handler {
public:
    TimerHandler() : status_(TIMER_IDLE) { }
    void sched(double delay); void resched(double delay); void cancel();
    enum TimerStatus { TIMER_IDLE, TIMER_PENDING, TIMER_HANDLING };
    int status() { return status_; };
protected:
    virtual void expire(Event *) = 0; virtual void handle(Event *);
    int status_; Event event_;
private:
    inline void _sched(double delay) {
        (void)Scheduler::instance().schedule(this, &event_, delay);
    }
    inline void _cancel() {
        (void)Scheduler::instance().cancel(&event_);
    }
};
```

Textbook: T. Issariyakul and E. Hossain, *Introduction to Network Simulator NS2*, Springer 2008.

# C++ Timers

- Variables of class `TimerHandler`

Variable	Meaning
<code>status_</code>	Store the current status of the timer
<code>event_</code>	An expiration event, placed on the simulation timeline

- Three main mechanisms:
  1. Waiting mechanism → `_sched(delay)`, `_resched(delay)`
    2. Interface to → `sched(delay)`, `resched(delay)`, and `cancel()`
    3. Expiration action → `expire(e)`, `handle(e)`

# C++ Timers

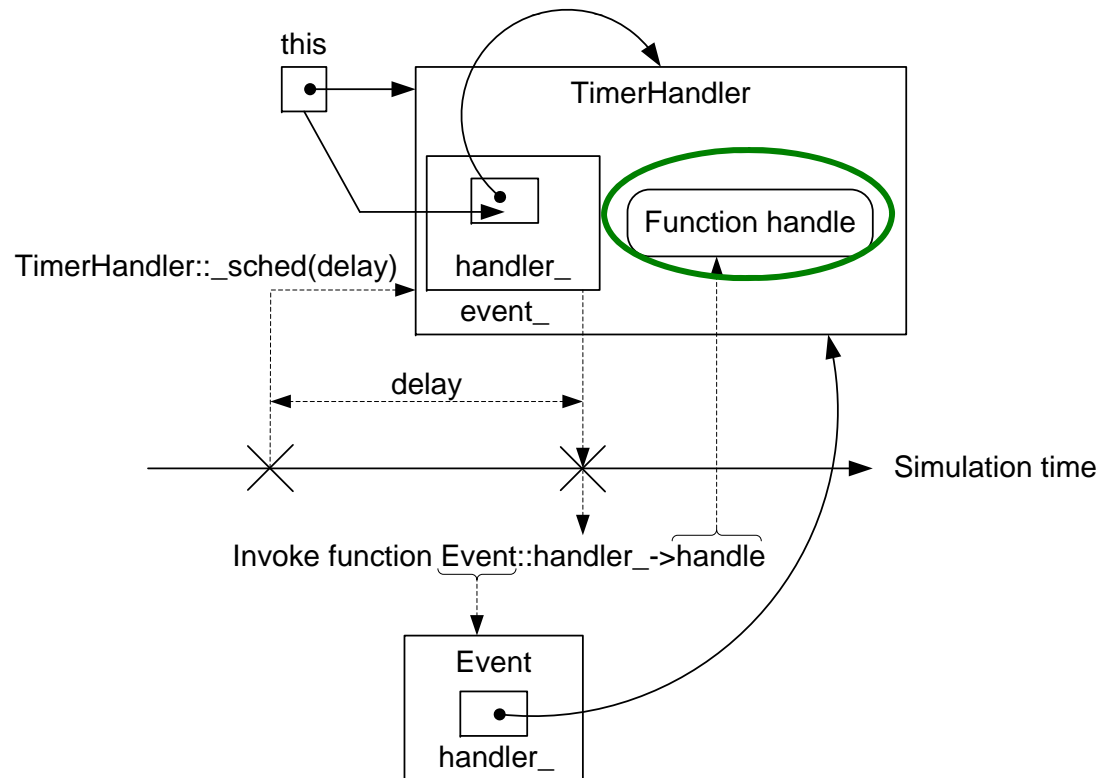
- Functions of class `TimerHandler`

Variable	Meaning
<code>sched(delay)</code>	(Public) Starts the timer and set the timer to expire at <code>delay</code> seconds in future.
<code>_sched(delay)</code>	(Private) Places a timer expiration event on the simulation time line at <code>delay</code> seconds in future.
<code>resched(delay)</code>	(Public) Restarts the timer and set the timer to expire at <code>delay</code> seconds in future.
<code>cancel()</code>	(Public) Cancels the pending timer.
<code>_cancel()</code>	(Private) Removes a timer expiration event from the simulation time line.
<code>status()</code>	Returns variable <code>status_</code> , the current state of the timer.
<code>handle(e)</code>	Invokes function <code>expire(e)</code> . It is used by the Scheduler to dispatch a timer expiration event
<code>expire(e)</code>	Takes expiration actions. It is a pure virtual function, and must be implemented by the child classes of class <code>TimerHandler</code> .

# C++ Timers

## 1. Waiting Mechanism

- Execute `_sched(delay)` {  
(void)Scheduler::instance().schedule(this, &event\_, delay);



# C++ Timers

## 2. Interface to schedule, reschedule and cancel expiration actions

```
//~/ns/common/timer-handler.cc
void TimerHandler::sched(double
delay)
{
    if (status_ != TIMER_IDLE) {
        fprintf(stderr, "Couldn't
schedule timer");
        abort();
    }
    _sched(delay);
    status_ = TIMER_PENDING;
}
```

```
//~/ns/common/timer-handler.cc
void TimerHandler::resched(double
delay)
{
    if (status_ == TIMER_PENDING)
        _cancel();
    _sched(delay);
    status_ = TIMER_PENDING;
}

void TimerHandler::cancel()
{
    if (status_ != TIMER_PENDING) {
        ...
        abort();
    }
    _cancel();
    status_ = TIMER_IDLE;
}
```

# C++ Timers

## 3. Expiration actions

- Defined in `handle(event_), expire(e)`
- Invoked by the `Scheduler`

```
//~/ns/common/timer-handler.cc
void TimerHandler::handle(Event *e)
{
    if (status_ != TIMER_PENDING)
        abort();
    status_ = TIMER_HANDLING;
    expire(e);
    if (status_ == TIMER_HANDLING)
        status_ = TIMER_IDLE;
}
```

- Function `expire(e)` is a pure virtual function  
→ defined in the derived class



# C++ Timers

3. Expiration actions → Two approaches

3.1 Define expiration actions in the timer class

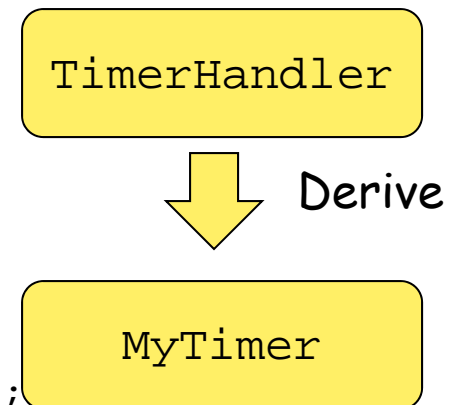
- Derive class `MyTimer` from class

`TimerHandler`

- Define function `expire(e)`

```
void MyTimer::expire(Event *e)
{
    printf("MyTimer has just expired!!\n");
}
```

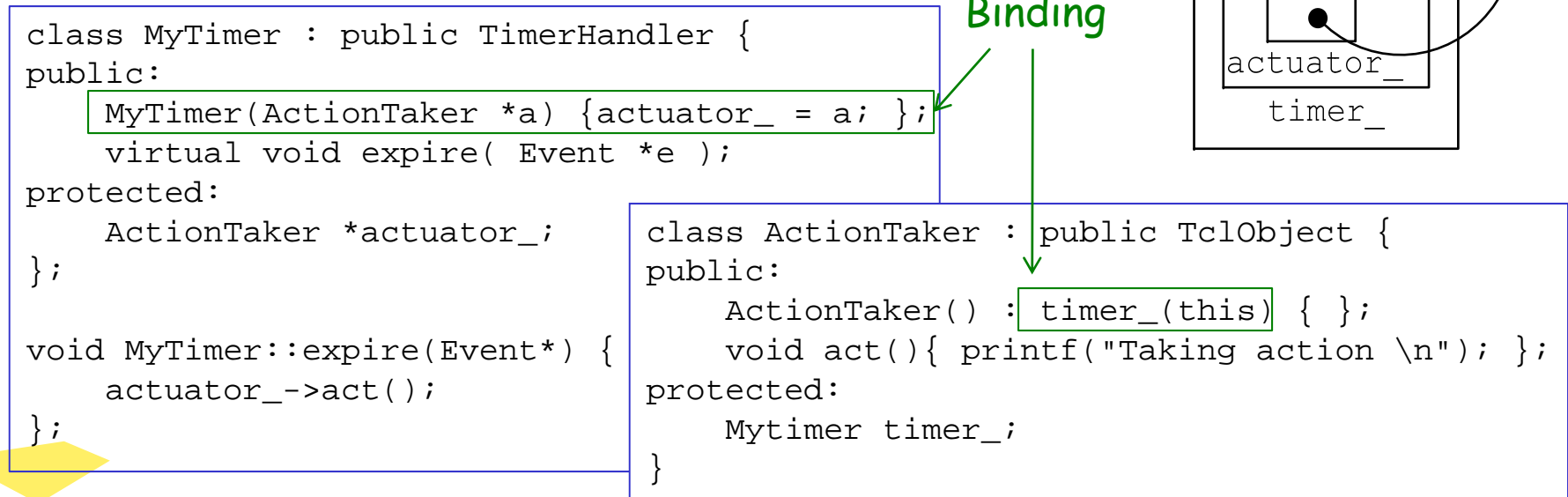
- This is useful but cannot do much!!



# C++ Timers

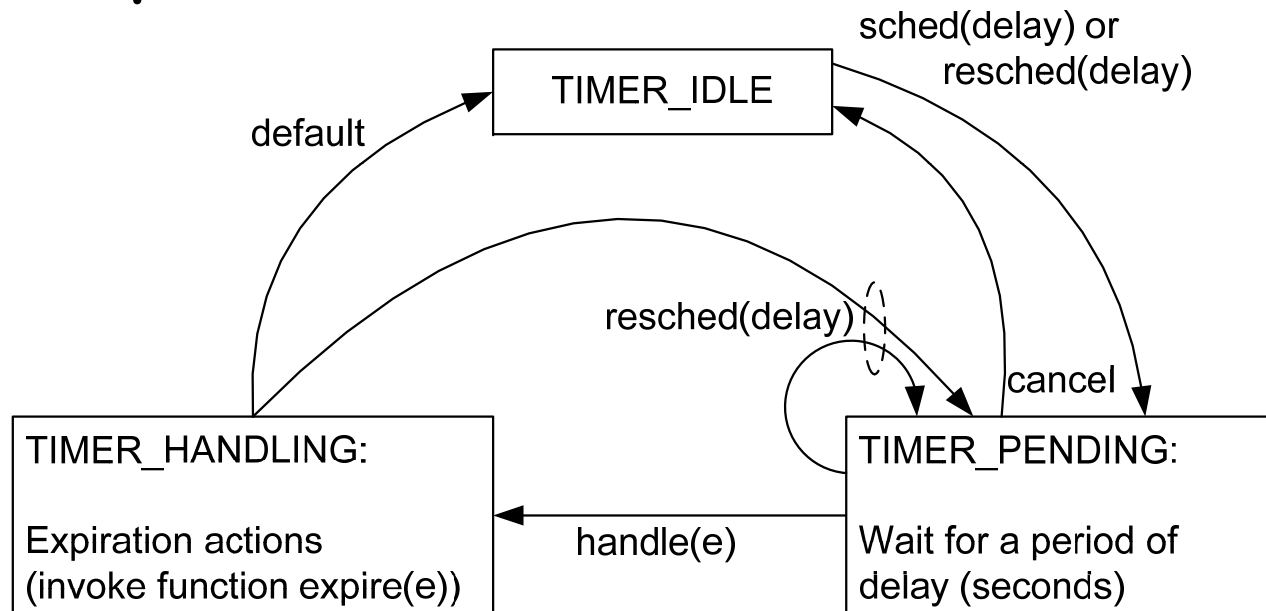
## 3.2 Define expiration actions in other classes

- Two classes: MyTimer and ActionTaker
- MyTimer models the timer
- ActionTaker defines default actions (e.g., print out texts)



# C++ Timers

- Recap



# A Guideline to Implement a New Type of Timers

- Based on the example `MyTimer-ActionTaker`
- **Class `MyTimer`**
  1. Derive class `MyTimer` from class `TimerHandler`.
  2. Declare a pointer `actuator_` to an `ActionTaker` object.
  3. Create a link to `actuator_` from the constructor.
  4. Define expiration action in function `expire(e)`.
- **Class `ActionTaker`**
  1. Declare a pointer `timer_` to the `MyTimer` object.
  2. Instantiate `timer_` with its `this` pointer from the constructor.



# Outline

- Timers
- Random Number Generator
- Built-in Error Model
- Bit Operations
- Summary

# Random Number Generators (Outline)

- Overview
- Seeding RNG
- Generating Random Numbers
- Randomization Scenarios
- Random Variables



# Generating Random Numbers

- non-Deterministic
- An example:
  - Tcl command `ns-random`
- Implementation
  - Random is not random!!
  - Predefined deterministic sequence of numbers → **Pseudo-random**
  - E.g., { ..., **72936**, **1193744747**, 30504276, ... }
  - Pick the number **sequentially**
  - **Seed** → Starting point

```
>>ns
```

```
>>ns-random
```

```
729236
```

```
>>ns-random
```

```
1193744747
```

```
>>exit
```

# RNG in NS2

- Combined Multiple Recursive Generator (MRG23k3a)
  - Contains  $1.8 \times 10^{19}$  streams (i.e., sequence)
  - Each contains  $2.3 \times 10^{15}$  substreams
  - Each contains  $7.6 \times 10^{22}$  numbers
  - Total random number = ?
- Seed → Specify the starting location



# Seeding

- NS2 is deterministic by default

```
>>ns
>>$defaultRNG next-random
729236
>>exit
```

```
### RESTART NS2 ###
>>ns
>>$defaultRNG next-random
729236
```

- Default seed = 1, → 

```
>>ns
>>$defaultRNG seed 101
>>$defaultRNG next-random
72520690
```

```
1
```

# Seeding in NS2

- Set seed to 0
  - Use current time and counter
- "new RNG",
  - Create new RNG
  - The seed is set to the new substream

Q: Hierarchy = ( )

- Default seed = 1, → change to 101

# Generating Random Numbers

- 3 Ways to Generate Random Numbers
  1. Command prompt
    - Use TclCommand: `ns-random`
  2. Random numbers in OTcl
  3. Random numbers in C++



## OTcl Class RNG

- Bound to C++ class RNG
- `$defaultRNG` is an OTcl global variable of class RNG
- To create new: "new RNG"



# OTcl Class RNG

- Instprocs

Instproc	Meaning
<code>seed{ }</code>	return seed
<code>seed{ &lt;n&gt; }</code>	set the seed to be <n>
<code>next-random{ }</code>	return a random number
<code>next-substream{ }</code>	go to the next substream
<code>exponential{ &lt;mu&gt; }</code>	exponentially distributed with mean <mu>
<code>uniform{ &lt;min&gt;, &lt;max&gt; }</code>	uniformly distributed within [ <min> <max> ].

- Example:

```
>>ns
>>$defaultRNG seed 101
>>$defaultRNG next-random
72520690
```

# C++ Class RNG

- File: `~ns/tools/rng.h, cc`

Function	Meaning
<code>set_seed(n)</code>	Set seed to be <code>&lt;n&gt;</code>
<code>seed()</code>	Return seed
<code>next()</code>	Return a random int
<code>next_double()</code>	Return a random double
<code>reset_next_substream()</code>	Move to the next substream
<code>uniform(k)</code> , <code>uniform(a,b)</code>	Return a random number uniformly distributed in $\{0, \dots, k-1\}$ , $[a, b]$
<code>exponential(mean)</code>	Return a random number exponentially distributed with mean " <code>&lt;mean&gt;</code> "
<code>normal(avg, std)</code>	Return a random number normally distributed with mean " <code>&lt;avg&gt;</code> " and standard deviation " <code>&lt;std&gt;</code> "

# Randomization Scenarios

## 1. Deterministic Setting

- Default Setting
- Useful for Debugging

## 2. Single-Stream Setting

- Most widely used
- Collect statistic (e.g., computes a mean from multiple runs)
- Add one line "`$defaultRNG seed <n>`"



# Randomization Scenarios

## 3. Multi-Stream Setting

- Need several RNGs
- Used to associate RNGs to random variables by using

```
<ranvar> use-rng <rng>
```

- `<ranvar>` is a `RandomVariable` instance and
- `<rng>` is an RNG instance



# Example: Multi-Stream/Random Variable

```
$defaultRNG seed 101
set arrivalRNG [new RNG]
set sizeRNG [new RNG]

set arrival_ [new RandomVariable/Exponential]
$arrival_ set avg_ 5
$arrival_ use-rng $arrivalRNG

set size_ [new RandomVariable/Uniform]
$size_ set min_ 100
$size_ set max_ 5000
$size_ use-rng $sizeRNG

puts "Inter-arrival time    Packet size"
for {set j 0} {$j < 5} {incr j} {
    puts [format "%-8.3f %-4d" [$arrival_ value] \
              [expr round([$size_ value])]
}
}
```

result

Inter-arrival time	Packet size
1.048	1880
7.919	116
8.061	3635
4.675	2110
7.201	1590

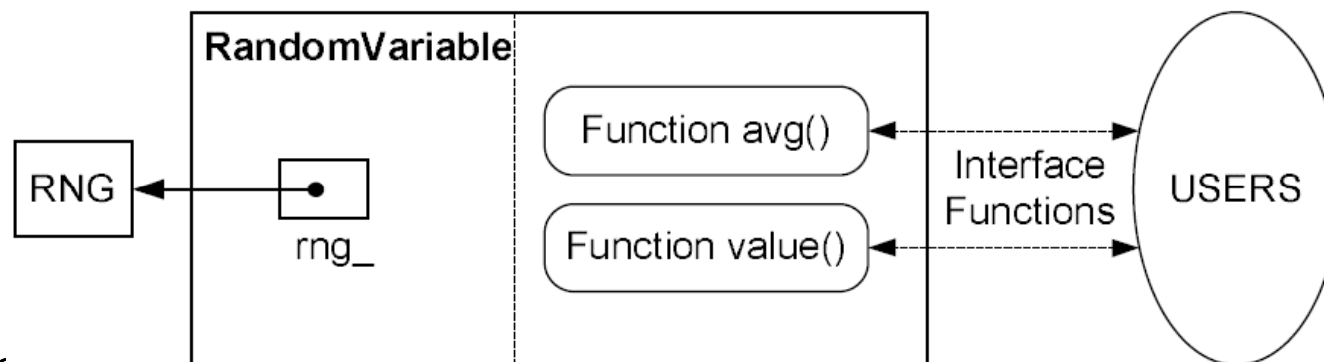
# Random Variables

- Definition
  - Mathematics: A function which measures how big a particular event when compared to the entire sample space.
  - NS2: A class generating random numbers according to the underlying distribution
- What about RNG?
  - Generate random number
  - No distribution!!
- Random variable = RNG + distribution
- See `~ns/tools/ranvar.h, cc`



# Random Variables

- Base class = RandomVariable
- C++ function
  - `avg()`: return/set the average value
  - `value()`: generate a random number
- OTcl instprocs
  - `value{}` : generate a random number
  - `use-rng{rng}` : set RNG



# Random Variables

- Examples of derived classes

C++ Class	OTcl Class
UniformRandomVariable	RandomVariable/Uniform
ExponentialRandomVariable	RandomVariable/Exponential
ParetoRandomVariable	RandomVariable/Pareto
ParetoIIRandomVariable	RandomVariable/ParetoII
NormalRandomVariable	RandomVariable/Normal
LogNormalRandomVariable	RandomVariable/LogNormal
ConstantRandomVariable	RandomVariable/Constant
HyperExponentialRandomVariable	RandomVariable/HyperExponential
WeibullRandomVariable	RandomVariable/Weibull
EmpiricalRandomVariable	RandomVariable/Empirical



# Guideline for Randomization

1. Determine the randomization scenario
  - Deterministic: Use static seed
  - Single-stream: Use `$defaultRNG`
  - Multi-stream: See (2)-(4)
2. Create RNGs
3. Define random variable
4. Associate RNGs for each random variable



# Outline

- Timers
- Random Number Generator
- Built-in Error Model
- Bit Operations
- Summary

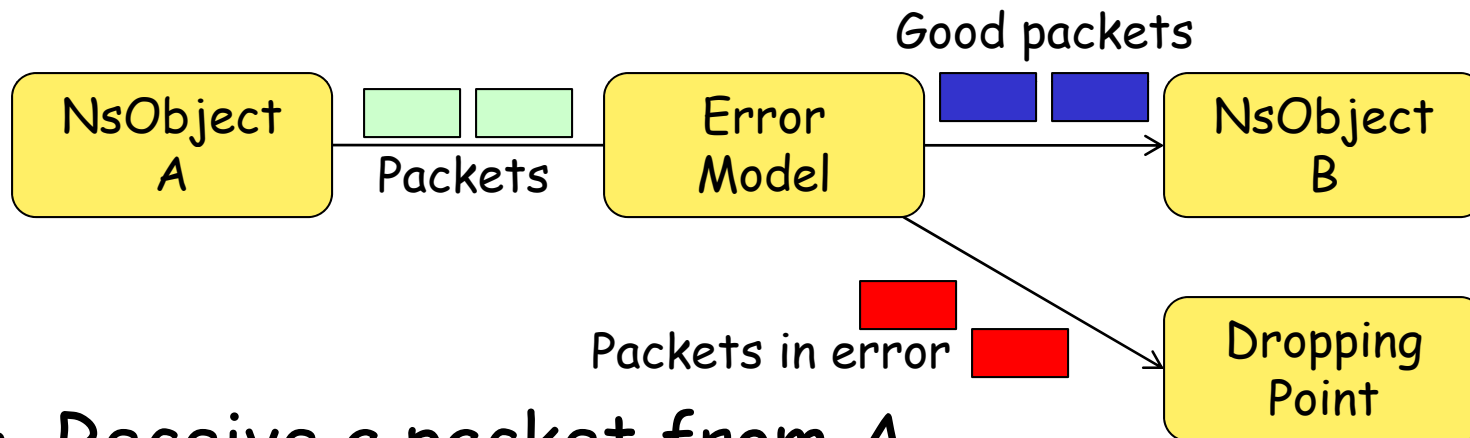
# Built-in Error Model (Outline)

- Overview
- OTcl: Configuration interface
  - Parameter setting
  - Network configuration
- C++: Internal process
- Creating your own error models



# Built-in Error Model

- Imposing error on packet transmission.



- Receive a packet from A
- Simulate error
  - Good packet → B
  - Packets in error → Dropping point

# Built-in Error Model

- OTcl implementation
- OTcl classes `ErrorModel`
- Bernoulli error: Error occurs w.p. "rate\_"
- Parameter configuration

Instvar	Meaning
<code>enabled_</code>	Set to 1 if this error model is active, and set to 0 otherwise.
<code>rate_</code>	Error probability
<code>delay_pkt_</code>	If set to <code>true</code> , the error model will delay (rather than drop) the transmission of corrupted packets.
<code>delay_</code>	Delay time in case that <code>delay_pkt_</code> is set to <code>true</code> .
<code>bandwidth_</code>	Used to compute packet transmission time

# Built-in Error Model

- Parameter configuration

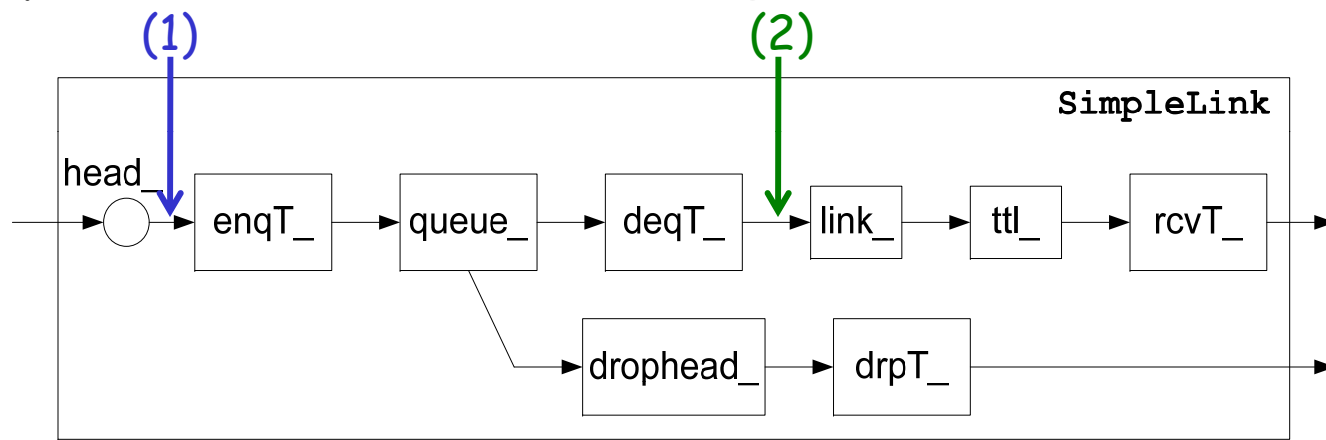
Instproc	Meaning
<code>ranvar{r}</code>	Specify Random Variable as <code>r</code>
<code>unit{u}</code>	Specify the unit of the error model as <code>u</code>

- Network configuration

Instproc	Meaning
<code>target{o}</code>	Specify target (derived from class Connector)
<code>drop-target{d}</code>	Specify drop target (derived from class Connector)

# Built-in Error Model

- Two types of network configuration:

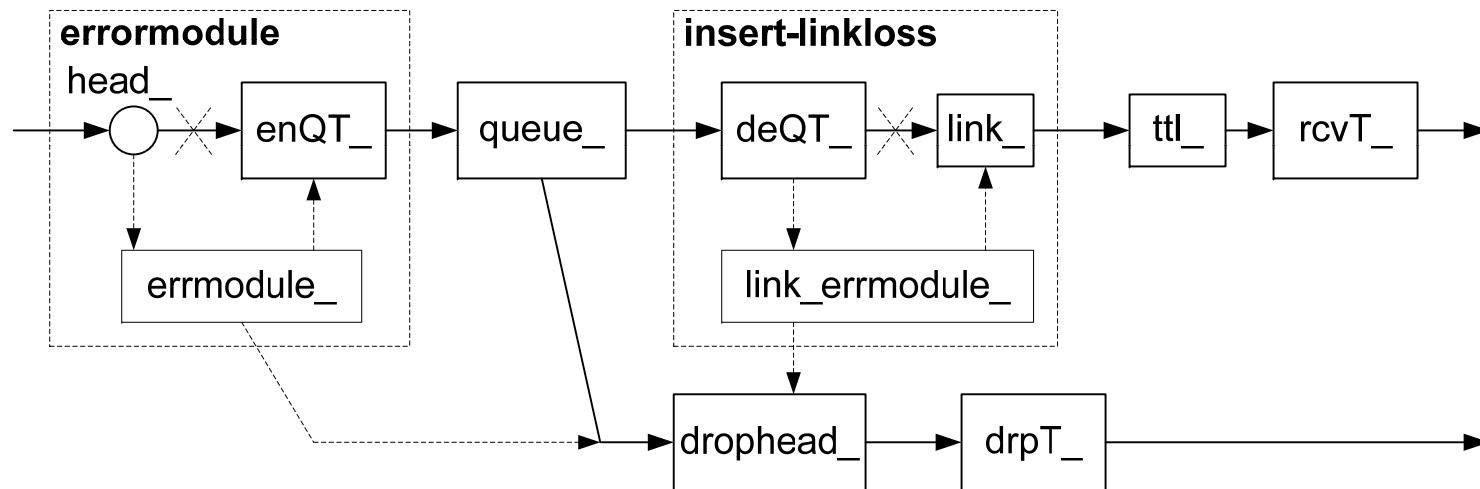


See file `~ns/lib/tcl/ns-lib.tcl`, `ns-link.tcl`.

Insert to Location	Class Simulator	Class SimpleLink
1	<code>lossmodel{lossobj from to}</code>	<code>errormodule{em}</code>
2	<code>link-lossmodel{lossobj from to}</code>	<code>insert-linkloss{em}</code>

# Built-in Error Model

- Result



Insert to Location	Class Simulator	Class SimpleLink
1	lossmodel{lossobj from to}	errormodule{em}
2	link-lossmodel{lossobj from to}	insert-linkloss{em}

# Built-in Error Model


- C++ class `ErrorModel`
  - Base class = `Connector`
  - Bound to OTcl class `ErrorModel`
- Main function
  - `recv(p, h)`: Packet reception
  - `corrupt(p)`: 1 if packet is in error, and 0 otherwise.



# Class ErrorModel

```
//~/ns/queue/errmodel.h
enum ErrorUnit { EU_TIME=0, EU_BYTE, EU_PKT, EU_BIT };

class ErrorModel : public Connector {
public:
    ErrorModel();
    virtual void recv(Packet*, Handler*);
    virtual int corrupt(Packet*);
protected:
    int enable_;
    ErrorUnit unit_;
    RandomVariable *ranvar_;
    Event intr_;
    double rate_; double delay_; double bandwidth_;
    int CorruptPkt(Packet*); int CorruptByte(Packet*); int CorruptBit(Packet*);
    double PktLength(Packet*);
    double* ComputeBitErrProb(int);
};
```



# Function recv(p,h)

```
//~/ns/queue/errmodel.cc
void ErrorModel::recv(Packet* p, Handler* h)
{
    hdr_cmn* ch = hdr_cmn::access(p);
    int error = corrupt(p);
    if (h && ((error && drop_) || !target_)) {
        double delay = Random::uniform(8.0*ch->size()/bandwidth_);
        if (intr_.uid_ < 0)
            Scheduler::instance().schedule(h, &intr_, delay);
    }
    if (error) {
        ch->error() |= error;
        if (drop_) { drop_->recv(p); return; }
    }
    if (target_) { target_->recv(p, h); }
}
```

- Packet is dropped after "delay" seconds  
- Let the NsObject \*h resumes whatever they are doing after "delay"seconds.

- Return here!!


# Function corrupt(p)

```
//~/ns/queue/errmodel.cc
int ErrorModel::corrupt(Packet* p)
{
    hdr_cmn* ch = HDR_CMN(p);
    if (enable_ == 0) return 0;
    switch (unit_) {
    case EU_TIME:
        return (CorruptTime(p) != 0);
    case EU_BYTE:
        return (CorruptByte(p) != 0);
    case EU_BIT:
        ch = hdr_cmn::access(p); ch->errbitcnt() = CorruptBit(p);
        return (ch->errbitcnt() != 0);
    default:
        return (CorruptPkt(p) != 0);
    }
    return 0;
}
```

# Class ErrorModel

```
//~/ns/queue/errmodel.h
enum ErrorUnit {EU_BYTE, EU_PKT, EU_BIT };

class ErrorModel : public Connector {
public:
    ErrorModel();
    virtual void recv(Packet*, Handler*);
    virtual int corrupt(Packet*);
protected:
    int enable_;
    ErrorUnit unit_;
    RandomVariable *ranvar_;
    Event intr_;
    double rate_; double delay_; double bandwidth_;
    int CorruptPkt(Packet*); int CorruptByte(Packet*); int CorruptBit(Packet*);
    double PktLength(Packet*);
    double* ComputeBitErrProb(int);
};
```



# Unit of Errors

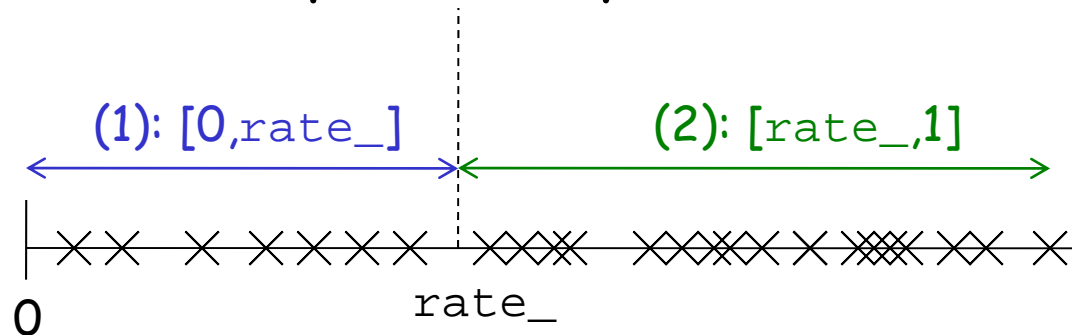
- `unit_` = Unit of errors for `rate_`

<code>unit_</code>	Meaning	Invoke this function upon invoking <code>corrupt(p)</code>
<code>EU_BIT</code>	Bit error probability	<code>CorruptBit(p)</code>
<code>EU_BYTE</code>	Byte error probability	<code>CorruptByte(p)</code>
<code>EU_PKT</code>	Packet error probability	<code>CorruptPkt(p)</code>
<code>EU_TIME</code>	not define here	<code>CorruptTime(p)</code>



# Error Simulation

- Suppose error probability =  $rate\_$

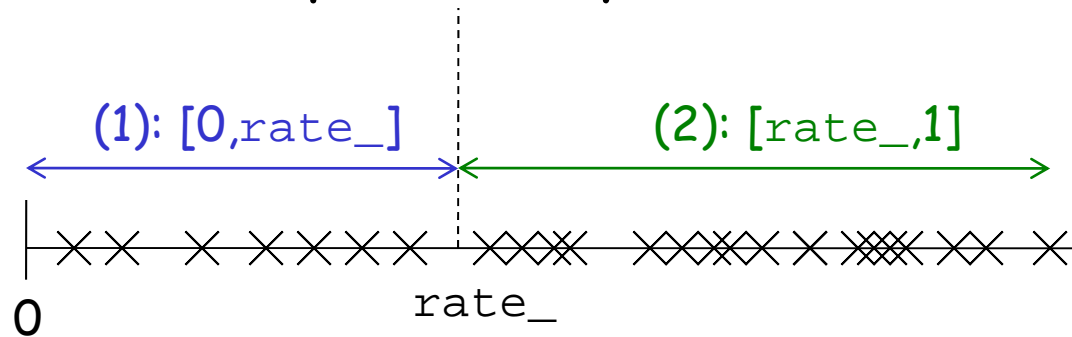


- Consider  $[0,1]$ .
- Randomly pick a point from  $[0,1]$  (Uniformly)  
→ every point in  $[0,1]$  is picked with equal probability.
- The ratio area (1):(1+2) =  $rate\_ : 1$
- A point you pick would be in
  - (1) with probability  $rate\_$  and
  - (2) with probability  $1-rate\_$



# Error Simulation

- Suppose error probability =  $rate\_$



- To generate a Bernoulli RV with  $rate\_$ 
  1. Create  $u = \text{unif}(0,1)$
  2. If  $u < rate\_$ , return 1 (error). Otherwise, return 0 (not in error).



# NS2 Implementation

- Function `CorruptPkt(p)`:

→ True if the packet is in error.

```
int ErrorModel::CorruptPkt(Packet*)
{
    double u = ranvar_ ? ranvar_>value() : Random::uniform();
    return (u < rate_);
}
```

- Function `CorruptByte(p)`:

→ True if the packet is in error.

```
int ErrorModel::CorruptPkt(Packet*)
{
    double per = 1 - pow(1.0 - rate_, PktLength(p));
    double u = ranvar_ ? ranvar_>value() : Random::uniform();
    return (u < rate_);
}
```

# NS2 Implementation

- Function `CorruptBit(p)`:

→ True if the packet is in error.

```
//~/ns/queue/errmodel.cc
```

```
int ErrorModel::CorruptBit(Packet* p)
{
```

```
    double u, *dptr; int i;
```

```
    if (firstTime_ && FECstrength_) {
```

```
        cntrlprb_ = ComputeBitErrProb(cntrlpktsize_);
```

```
        datapr_b_ = ComputeBitErrProb(datapktsize_);
```

```
        firstTime_ = 0;
```

```
    }
```

```
    u = ranvar_ ? ranvar_>value() : Random::uniform();
```

```
    dptr = (hdr_cmn::access(p)->size() >= datapktsize_)
           ? datapr_b_ : cntrlprb_;
```

```
    for (i = 0; i < (FECstrength_ + 2); i++)
```

```
        if (dptr[i] > u) break;
```

```
    return(i);
```

```
}
```

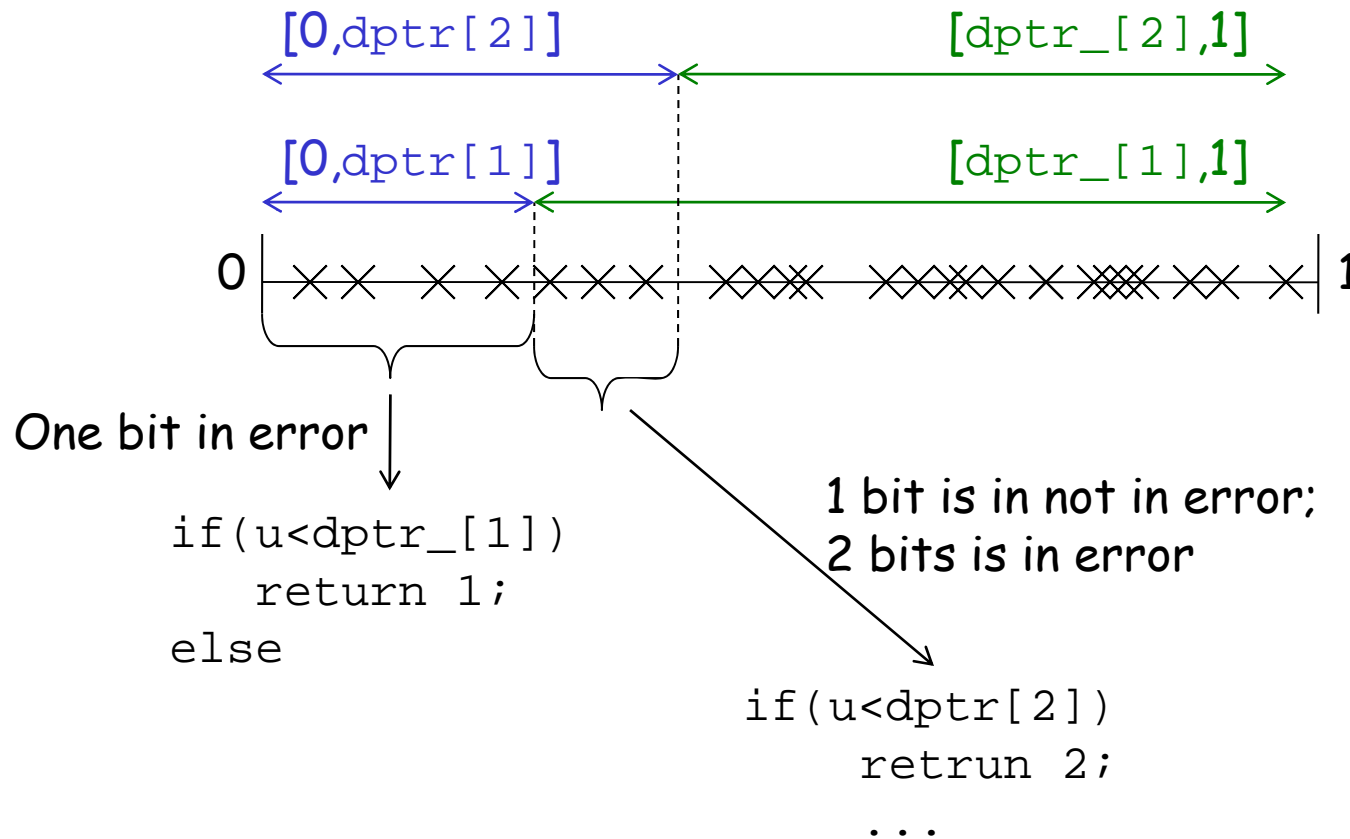
**$i$  = the number of bits in error** → Assume that a data packet is bigger in size.

Compute the probability of having at most  $i$  bits in error

Bit error probability

A packet can be either a data packet or a control packet

# NS2 Implementation



# NS2 Implementation

- **Function** `ComputeBitErrProb(size)`:  
→ Return an array whose *i*th element is the probability of having at most *i* bit errors.

```
double* ErrorModel::ComputeBitErrProb(int size)
{
    double *dptr; int i;
    dptr = (double *)calloc((FECstrength_ + 2), sizeof(double));
    for (i = 0; i < (FECstrength_ + 1) ; i++)
        dptr[i] = comb(size, i) * pow(rate_,
            (double)i) * pow(1.0 - rate_, (double)(size - i));
    for (i = 0; i < FECstrength_ ; i++)
        dptr[i + 1] += dptr[i];
    dptr[FECstrength_ + 1] = 1.0;
    return dptr;
}
```

$$\binom{\text{size}}{i} (\text{rate}_-)^i (1 - \text{rate}_-)^{\text{size}-i}$$

# Creating a New Error Model

1. Design an create error model: C++, OTcl, or both
2. Configure `unit_`, `rate_`, and `ranvar_`
3. Inserting the error model (e.g., using `Simulator::lossmodel{...}`)



# Outline

- Timers
- Random Number Generator
- Built-in Error Model
- Bit Operations
- Summary

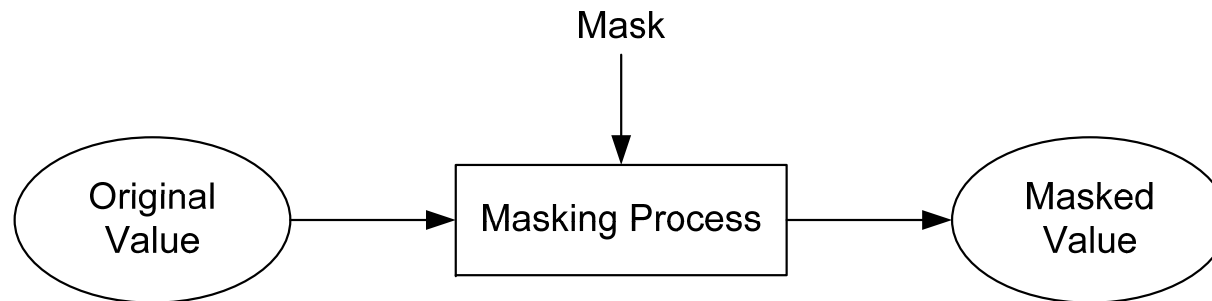
# Bit Operations in NS2 (Outline)

- Bit Masking
- Bit Shifting and Decimal Multiplication



# Bit Masking

- Masking: Original  $\rightarrow$  Masked
- Unmasking: Masked  $\rightarrow$  Original



- Two examples:
  - Subnet masking
  - Modulo masking

# Subnet Masking

- Internet Protocol (IP) [ipconfig]

```
C:\WINDOWS\system32\cmd.exe
Lease Expires . . . . . : Thursday, April 30, 2009 2:46:15 PM
Ethernet adapter Local Area Connection:
    Media State . . . . . : Media disconnected
    Description . . . . . : Intel(R) PRO/100 VE Network Connecti
on
    Physical Address. . . . . : 00-01-4A-5E-75-51
C:\Documents and Settings\I_Bear>ipconfig
Windows IP Configuration

Ethernet adapter Wireless Network Connection:
    Connection-specific DNS Suffix . :
    IP Address . . . . . : 10.0.5.132
    Subnet Mask . . . . . : 255.255.255.0
    Default Gateway . . . . . : 10.0.5.1
Ethernet adapter Local Area Connection:
    Media State . . . . . : Media disconnected
C:\Documents and Settings\I_Bear>
```

Q: How do we compute a subnet?

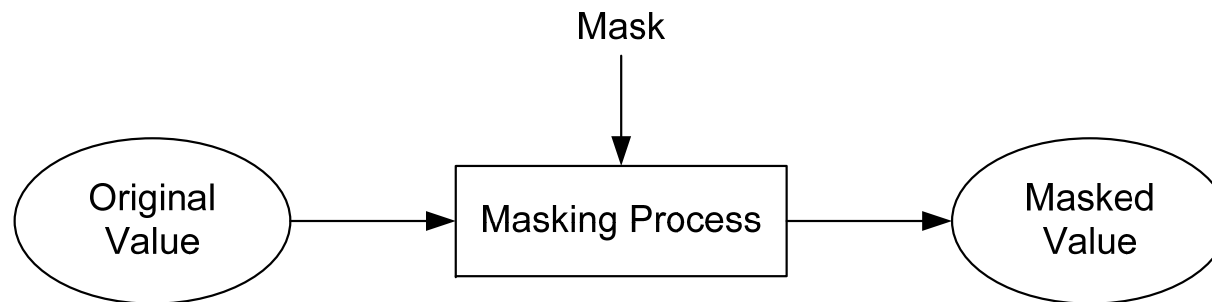
- Host IP Address: 10.0.5.132
- Subnet mask: 255.255.255.0 (class C)
- Subnet: 10.0.5.0



# Subnet Masking

- Subnet computation:

Subnet = Host IP Address & Subnet Mask



- Original value = Host IP Address
- Masking process = &
- Mask = Subnet mask
- Masked value = Subnet

# Modulo Masking

- Modulo
  - Denoted by "%"
  - Let  $a = b \times c + d$ . Then  $a \% b = a \% c = d$
- If  $c = 2^n$ , where  $n$  is a positive integer
$$a \% c = a \& (2^n - 1)$$
- To see this, let write  $a$  in binary where  $a = \text{xxx...xx}$ 
  - Original value =  $x \ x \ x \ \dots \ x \ \boxed{x \ x \ \dots \ x} = a$
  - Mask =  $\underline{0 \ 0 \ 0 \ \dots \ 0 \ 1 \ 1 \ \dots \ 1} = c$
  - Masked value =  $\underline{0 \ 0 \ 0 \ \dots \ 0 \ x \ x \ \dots \ x} = a \% c$



# Bit Shifting and Decimal Multiplication

- Consider  $y = 11_2 = 3$
- Shifting ( $y \ll n$ )
  - $n = 1, y \ll n = 110_2 = 6 = 3 \times 2$
  - $n = 2, y \ll n = 1100_2 = 12 = 6 \times 2$
  - $n = 3, y \ll n = 11000_2 = 24 = 12 \times 2$
- Therefore, we have
  - Left shift by  $n$  bits  $\Leftrightarrow$  Multiplication by  $2^n$
  - Right shift by  $n$  bits  $\Leftrightarrow$  Division by  $2^n$



# Outline

- Timers
- Random Number Generator
- Built-in Error Model
- Bit Operations
- Summary

# Summary

- Timer:
  - Purpose: delay actions
  - Main components:
    1. Waiting mechanism
    2. Interface to start/restart/cancel
    3. Expiration actions
  - Useful scenario: Define expiration actions in another object



# Summary

- Random Number Generator (RNG):
  - Pseudo-random number with seeds
  - Scenarios
    1. Deterministic: Default, For debug
    2. Single stream: Set the seed
    3. Multi stream: Create several RNGs
  - Random variable = RNG + Distribution



# Summary

- Built-in Error Model
  - A module derived from a Connector
    - Packet in error → downstream object
    - Packet not in error → dropping object
  - OTcl/C++ classes: `ErrorModel`
  - Inserting error model
    - `lossmodel{lossobj from to}`
    - `link-lossmodel{lossobj from to}`
  - C++ main function:

`recv(p, h) → corrupt(p)`

# Summary

- Bit Operations

1. Bit masking

- Subnet masking
- Modulo masking

2. Bit shifting

- Decimal multiplication
  - Right shift by  $n$  bits  $\Leftrightarrow$  Multiplication by  $2^n$
  - Left shift by  $n$  bits  $\Leftrightarrow$  Division by  $2^n$

