

Network Objects: Creation, Configuration, and Packet Forwarding



Outline

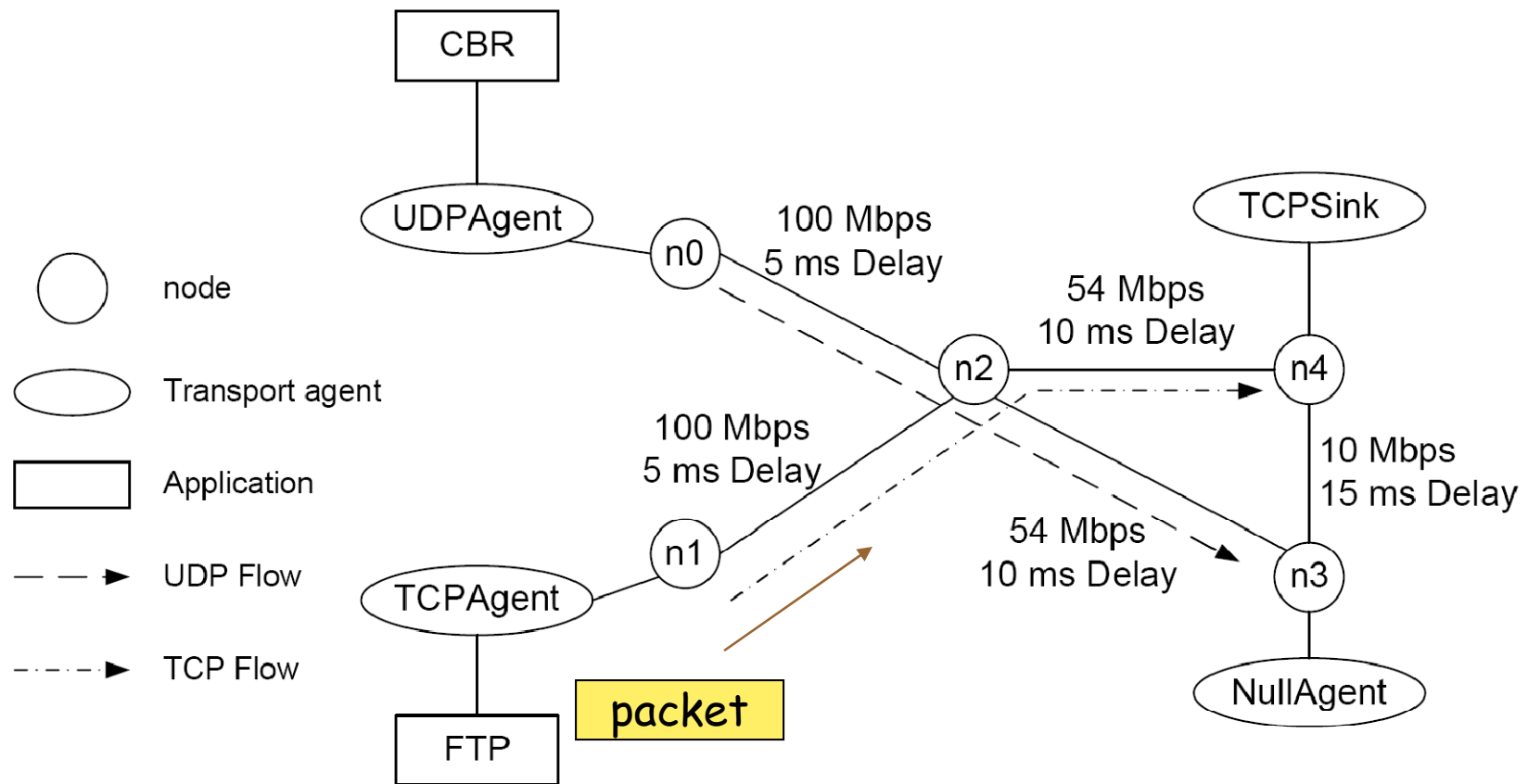
- NS2 Main Components and NsObjects
- Connector: An Example of NsObjects
- Packet Forwarding Mechanism
- Summary

NS2 Network Components

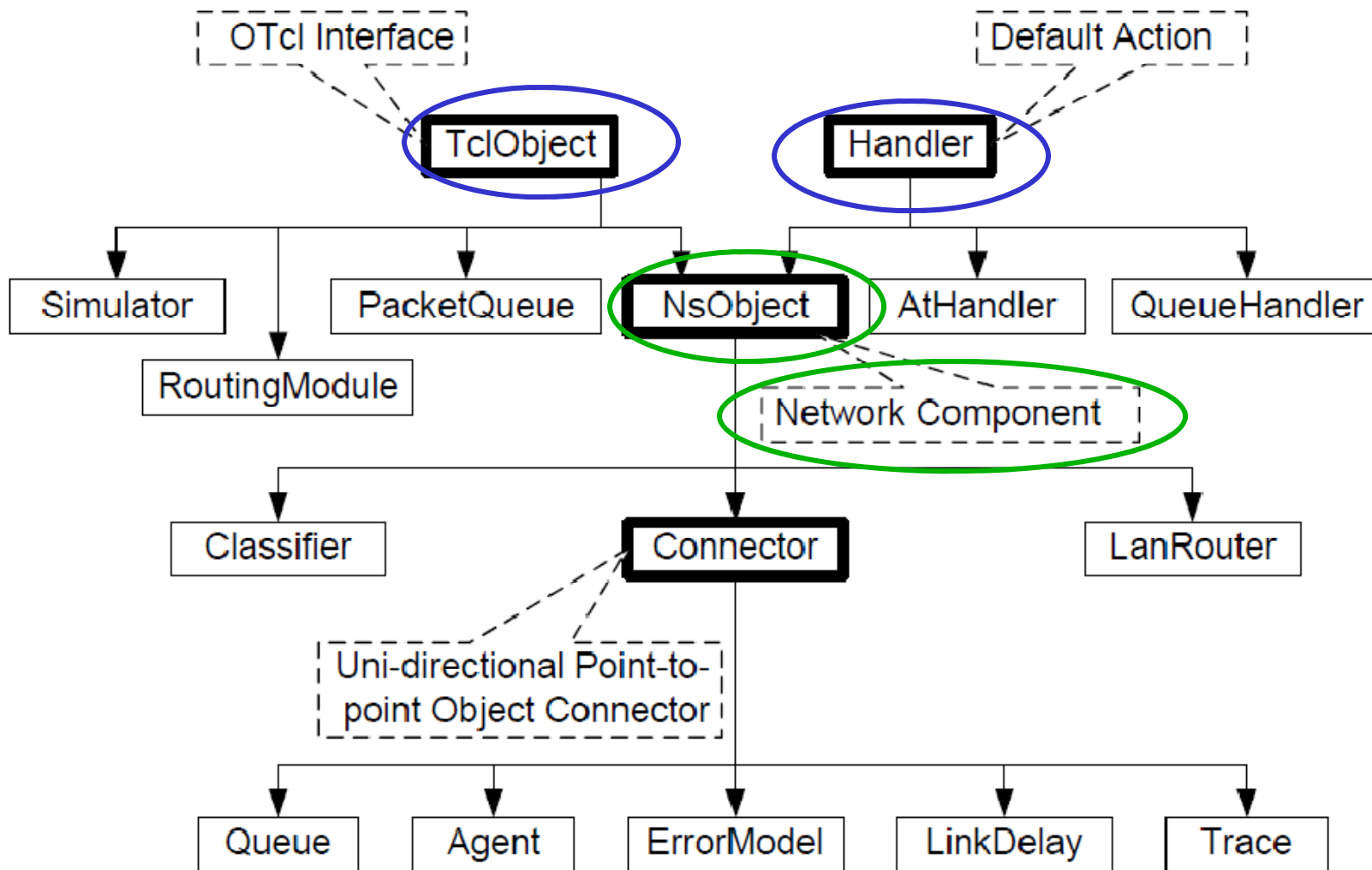
- 4 Main Components
 1. Network objects
 2. Packet-related objects
 3. Simulation-related objects
 4. Helper objects



NS2 Network Components



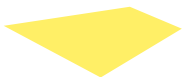
NS2 Network Components



C++ Class NsObject

- Sending and receiving objects
- Base class
 - TclObject → (OTcl interface)
 - Handler → (default actions)
- In NS2, every network object can only receive packet

IT DOES NOT SEND!!



C++ Class NsObject

```
//~/ns/common/object.h
class NsObject : public TclObject, public Handler {
public:
    NsObject();
    virtual ~NsObject();
    virtual void recv(Packet*, Handler* callback = 0) = 0;
    ...
protected:
    void handle(Event*);
    ...
};
```

- A () function `recv(p,h)`

What does this type of function do?

()

Default Actions

- Function `handle(e)`
 - Invoked to **dispatch** an event
 - `NSObject` → **A packet is received**

```
//~/ns/common/object.cc  
void NSObject::handle(Event* e)  
{  
    recv((Packet*)e);  
}
```



Outline

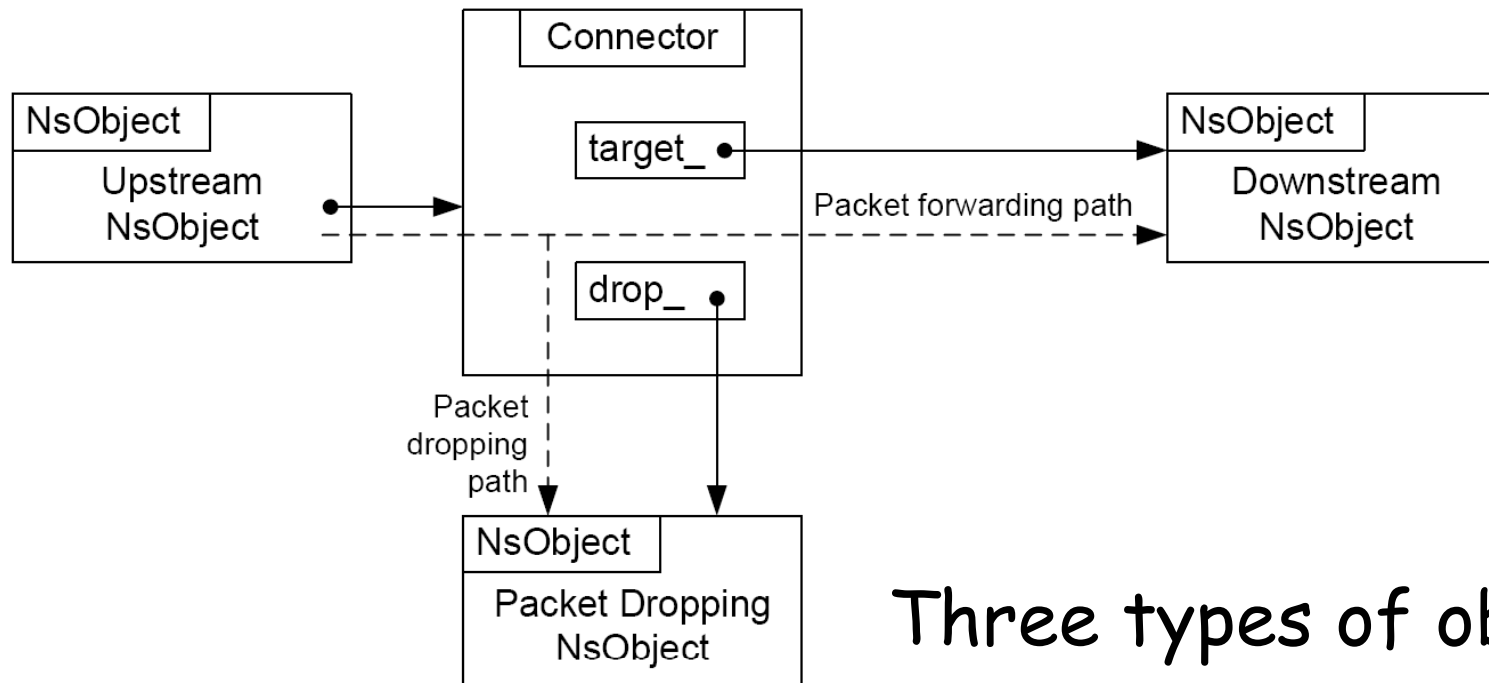
- NS2 Main Components and NsObjects
- Connector: An Example of NsObjects
- Packet Forwarding Mechanism
- Summary

Connector

- NsObject has the following features
 - An object
 - OTcl interface
 - Default action
 - Receive function template
- Connector
 - *Derive* from NsObject
 - *Connect* two NsObjects
 - *Override* function recv(p,h)



C++ Class Connector

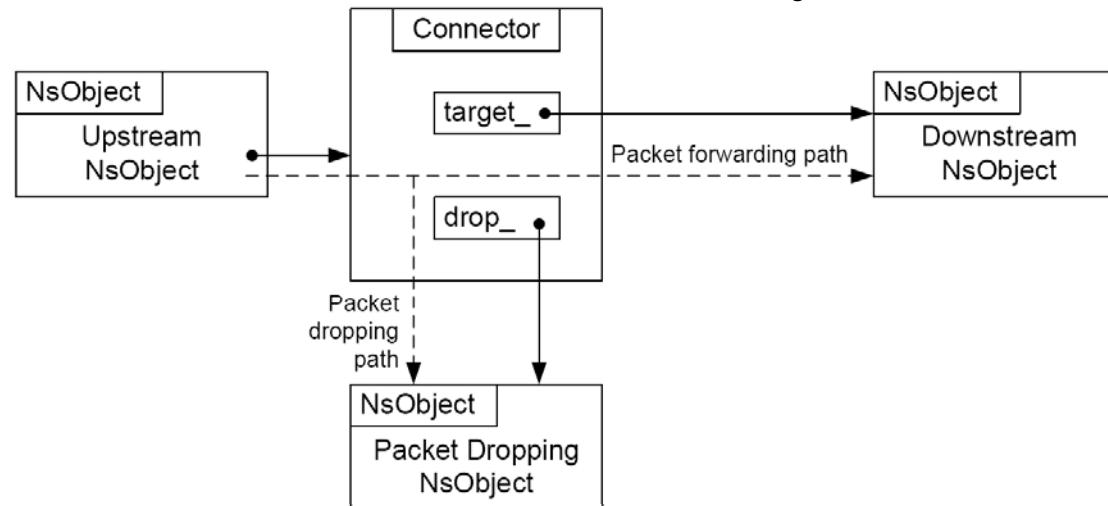


Three types of objects

1. An **upstream** object
2. A **downstream** object, and
3. A **dropping** object

Class Connector

- Derived from class NsObject
- NsObject **does not send** packets
- It asks the downstream object to **receive**

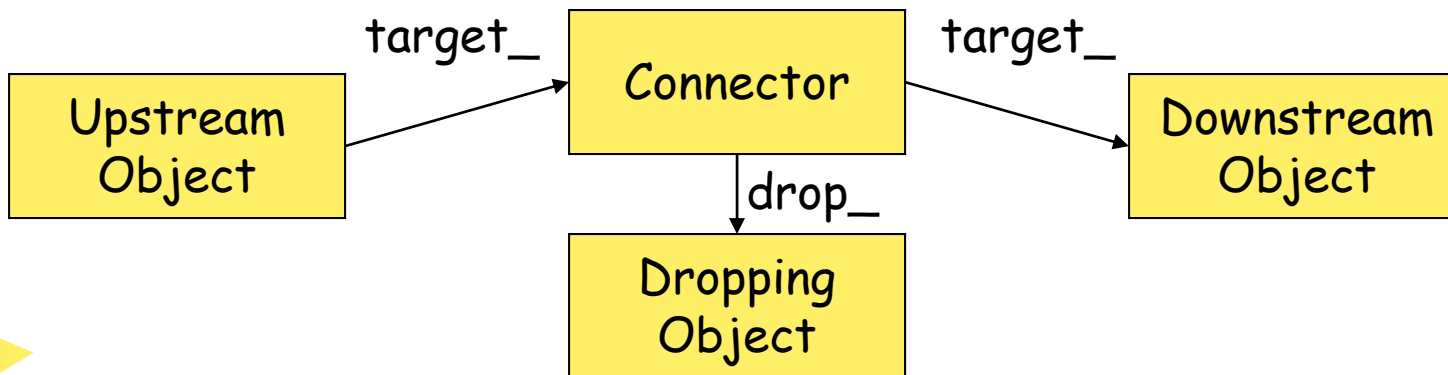


- Connects an upstream object to
 - A **downstream** object, and
 - A **dropping** object

Class Connector

```
//~/ns/common/connector.h
class Connector : public NsObject {
public:
    Connector();
    virtual void drop(Packet* p);
protected:
    void recv(Packet*, Handler* callback = 0);
    inline void send(Packet* p, Handler* h){target_->recv(p, h);}
    NsObject* target_;
    NsObject* drop_;
};
```

```
//~/ns/common/connector.cc
void Connector::recv(Packet* p, Handler* h){send(p, h);}
```



Connector: OTcl Configuration

- Defined in function `Connection::command(...)`
- Suppose a connector `$conn`
- Specify a **target**

```
$conn target <target_name>
```

- Specify a **dropping point**

```
$conn drop-target <drop_name>
```

- Retrieve a **target**

```
set my_target [ $conn target ]
```

- Retrieve a **dropping point**

```
set my_drop [ $conn drop-target ]
```



Outline

- NS2 Main Components and NsObjects
- Connector: An Example of NsObjects
- Packet Forwarding Mechanism
- Summary

Packet Forwarding Mechanism

- Forward to downstream objects only!! (i.e., **Receive only**)
- **Two** types of packet forwarding
 1. **Immediate** Packet Forwarding
 - Invoke fn **recv(p,h)** of a downstream object
 - e.g., see Connector (`target_ ->recv(p,h)`)
 2. **Delayed** Packet Forwarding
 - The packet will be received in future.
 - E.G., The downstream object will receive a packet after "**propagation delay**".



Delayed Packet Forwarding

- Put a packet forwarding event on the simulation timeline → Invoke function
`Scheduler::schedule(target, p, delay)`
- Move forward in time to the next event
- Execute the event → At time t , invoke function
`dispatch(p, t)` of class Scheduler

Packet 1 is received
by Node 2



Packet 2 is received
by Node 2



Delayed Packet Forwarding

- The event has a pointer `handler_` to an `NSObject` which receives the packet.
- Event execution: Invoke `handler_ -> handle(p)`
- For `NSObject`,

```
//~/ns/common/object.cc  
void NSObject::handle(Event* e)  
{  
    recv((Packet*)e);  
}
```

Packet 1 is received }
by Node 2

Packet 2 is received
by Node 2



Delayed Packet Forwarding: Recap

- Use `schedule(target, p, delay)`
- Send packet `*p` to object `*target` at `delay` seconds in future
- 3 Main Steps:
 1. Associate a packet with a downstream object

Type casting: `Handler` → `NSObject` → `DownstreamObject (*target_)`

2. Put the packet (as an event) on the schedule

Type casting: `Event` → `Packet (*p)`

3. Execute the packet at a dispatching time
(`current_time + t`)

`Scheduler::dispatch(p, current_time+t)`

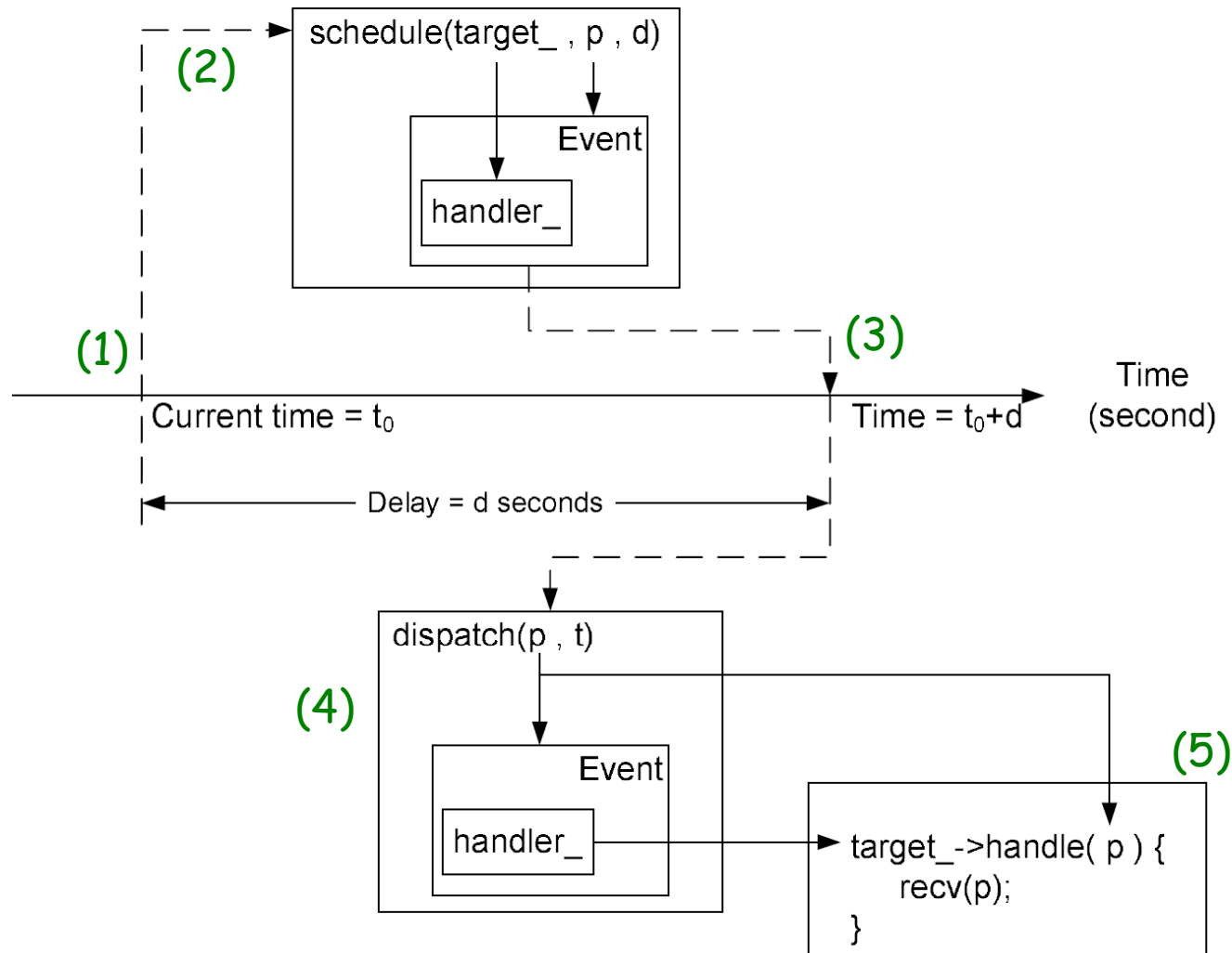
Example: Class LinkDelay

- Simulate packet transmission time and packet delay

```
//~ns/link/delay.cc
void LinkDelay::recv(Packet* p, Handler* h)
{
    double txt = txttime(p);
    Scheduler& s = Scheduler::instance();
    if (dynamic_) { /* See ~ns/link/delay.cc */ }
    else if (avoidReordering_) { /* See ~ns/link/delay.cc */ }
    else {
        s.schedule(target_, p, txt + delay_);
    }
    s.schedule(h, &intr_, txt);
}
```

Delayed Packet Forwarding

`schedule(target_, p, d)`



Outline

- NS2 Main Components and NsObjects
- Connector: An Example of NsObjects
- Packet Forwarding Mechanism
- Summary

Summary

- TclObject: An object + OTcl interface
- NsObject:
 - A TclObject + Network functionality (i.e., function ())
- Connector:
 - ()
 - NsObject + () + ()
 - Define how to receive packets (i.e.,)
- Packet forwarding
 - Downstream only
 - Immediate: use ()
 - Delayed: use ()