Transmission Control Protocol (TCP) Agents

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

Outline

- Transmission Control Protocol (TCP)
- An Overview of NS2 Implementation
- TCP Receiver
- TCP Sender
- Summary

Introduction

- Recall: Transport Layer Protocol
  - Implemented at the end points
  - Flow control
  - Error Control
  - Application-NW bridge
- UDP (User Datagram Protocol)
- Transmission Control Protocol (TCP)
- Suggested Reading: J. F. Kurose and K. W. Ross, Computer Networking: A Top-Down Approach. Pearson Addison-Wesley, 2008

Comparison with UDP

	UDP	TCP
Implement at	Source only	Source and Destination
Flow control	None	Window based
Error control	None (non-reliable)	ACK based (reliable)
Connection type	Connection less	Connection oriented
App-NW Bridge	Yes	Yes

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

TCP: Main Features

- Implemented at
  - Source → Packet Transmission
  - Destination → ACK Transmission
- Window-based flow (speed) control

	A window	**TX window**
Things to flow through	Air	Data
Small window	Less air can flow through	Less data can flow through
Window is closed	No air can flow through	No data can flow through

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

TCP: Main Features

- Reliable TL protocol:
  - All data must be received
  - Use ACK
- Connection Oriented TL Protocol:
  - 3 Phrases of data transfer

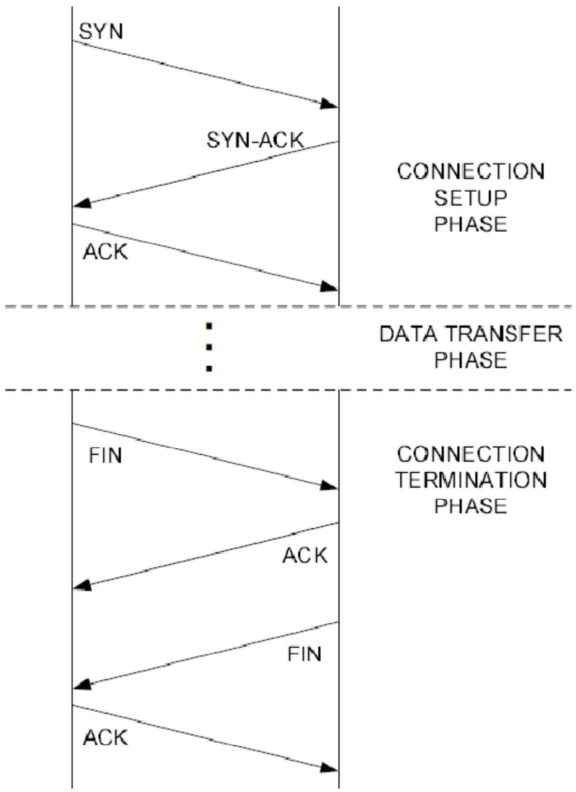
1. Connection setup
2. Data transfer:

(Main Part of TCP; We will focus on this part)

3. Connection termination

Connection setup and termination

- Connection setup
  - Three way handshake
  - SYN/SYN-ACK/ACK
- Connection Termination
  - Four way handshake
  - 2 x FIN/ACK



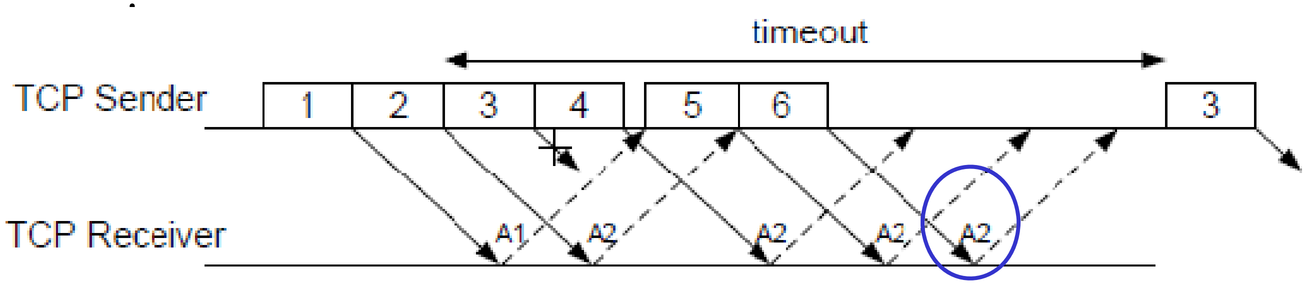
Textbook: T. Issariyakul and E. Hossain, *Introduction to Network Simulator NS2*, Springer 2008.

Data Transfer

- Error control
  - Provide reliability
  - Acknowledgement
  - Timeout
- Flow control
  - Control transmission speed
  - At the source node
  - Window-based

Error Control

- Acknowledgement
  - The receiver: Send an Acknowledgement (ACK) packet for every received packet.
  - ACK with highest seq. no. which has been received.
- Timeout
  - A TCP sender start a timeout counter at after sending each packet
- Loss detection
  - Timeout: An ACK is not received prior to the expiration of the timeout counter.
  - Fast Retransmit: The sender receives the 3rd ACK



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

Timeout Value Adjustment

- Timeout facts
  - Loss detection
  - Long timeout ➔ Long latency in detection loss
  - Short timeout ➔ Unnecessary packet transmission
  - A function of round trip time
- TCP timeout [RFC2988]: For the k^{th} packet
  - Store a sending time in the packet
  - Compute the round trip time (RTT) $t(k)$, when the ACK for the k^{th} packet returns.
  - Compute average RTT, $\bar{t}(k)$.
  - Compute standard variation of RTT, $\sigma_t(k)$.
  - Compute retransmission timeout (RTO), $RTO(k)$.

Timeout Value Adjustment

- RFC 2988

$$\bar{t}(k+1) = \alpha \times \bar{t}(k) + (1-\alpha) \times t(k+1), \alpha \in (0,1)$$
$$\sigma_t(k+1) = \beta \times \sigma_t(k) + (1-\beta) \times |t(k+1) - \bar{t}(k+1)|$$
$$RTO(k+1) = \min\{ub, \max\{lb, \gamma \times [\bar{t}(k+1) + 4 \times \sigma_t(k+1)]\}\}$$

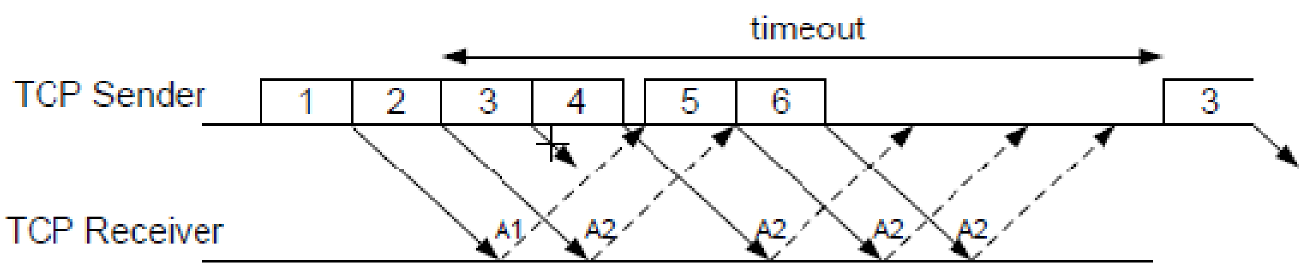
- α = 7/8, β = 3/4
- lb = lower bound (0.2 seconds in NS2)
- ub = upper bound (10^5 seconds in NS2)
- γ : - Initialized to 1
  - Double for every timeout event
  - Reset to 1 upon reception of ACK
- Timeout granularity = 0.5 s (default), 0.1 s (NS2)

Q: How does NS2 setup these initial values?
A:

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

Flow Control

- Window-based: Window size = W
- Control TX rate, ⇔ Controlling the window size
  1. Increasing TX rate
  2. Decreasing TX rate
- W = no. of packets that can be TX without being ACKed

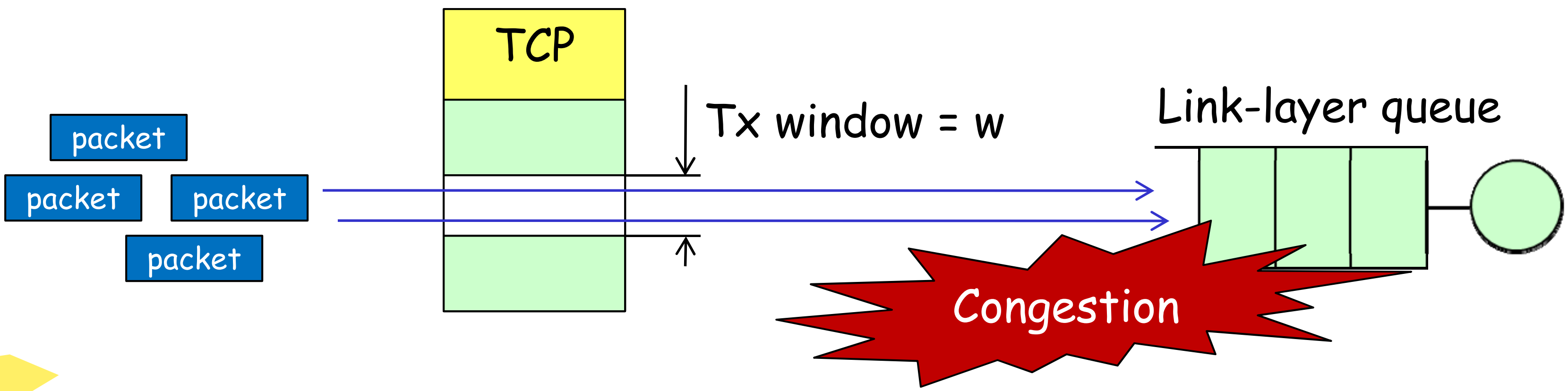


W = ? (4)

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

Window Adjustment Rational

- Large W ➔ Send more data without waiting
  ➔ Higher data rate
- W is too large ➔ Congestion ➔ reduce W
- Note: TCP assumes all losses come from congestion



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

Increasing TX Rate

- Increase or widen the TX window
- A.K.A. Open the TX window
- More data can flow through
- Until $W = W_{max}$, the window increament process has two phases
  1. Slow start:
     - $W < W_{th}$ (i.e., slow start threshold)
     - $W = W+1$ for every received ACK
  2. Congestion avoidance:
     - $W >= W_{th}$
     - $W = W+1/W$ for every received ACK

Decreasing TX Rate

- Decrease or narrow down the TX window
- A.K.A. Close the TX window
- To avoid congestion
- Two methods

1. Reset to 1
2. Fast Recovery:
   - Set W and W_{th} to half of its current value
   - Increase W by 1 for every received **duplicated** ACK
   - After receiving a new ACK ➔ Congestion Avoidance (i.e., $W = W_{th}$)

Data Transfer: Recap

- Error control
  - Acknowledge every packet
  - Loss detection
    - Timeout: Not receiving ACK for a long time
    - Fast Retransmit: 3 Duplicated ACK
  - Packet Retransmission
- Flow control
  - Increase TX rate
    - Slow-start: $W < W_{th}$; $W=W+1$ for every ACK
    - Congestion avoidance: $W >= W_{th}$; $W=W+1/W$ for every ACK
  - Decrease TX rate
    - Reset to 1
    - Fast Recovery: Half W and W_{th}

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

Typical TCP Variants

TCP Variant	Loss Detection	
OUR FOCUS!!	Timeout	Fast Retransmit
Old-Tahoe	Reset w to 1	N/A
Tahoe	Reset w to 1	Reset w to 1
Reno	Reset w to 1	Fast Recovery (single packet)
New Reno	Reset w to 1	Fast Recovery (all packets)

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

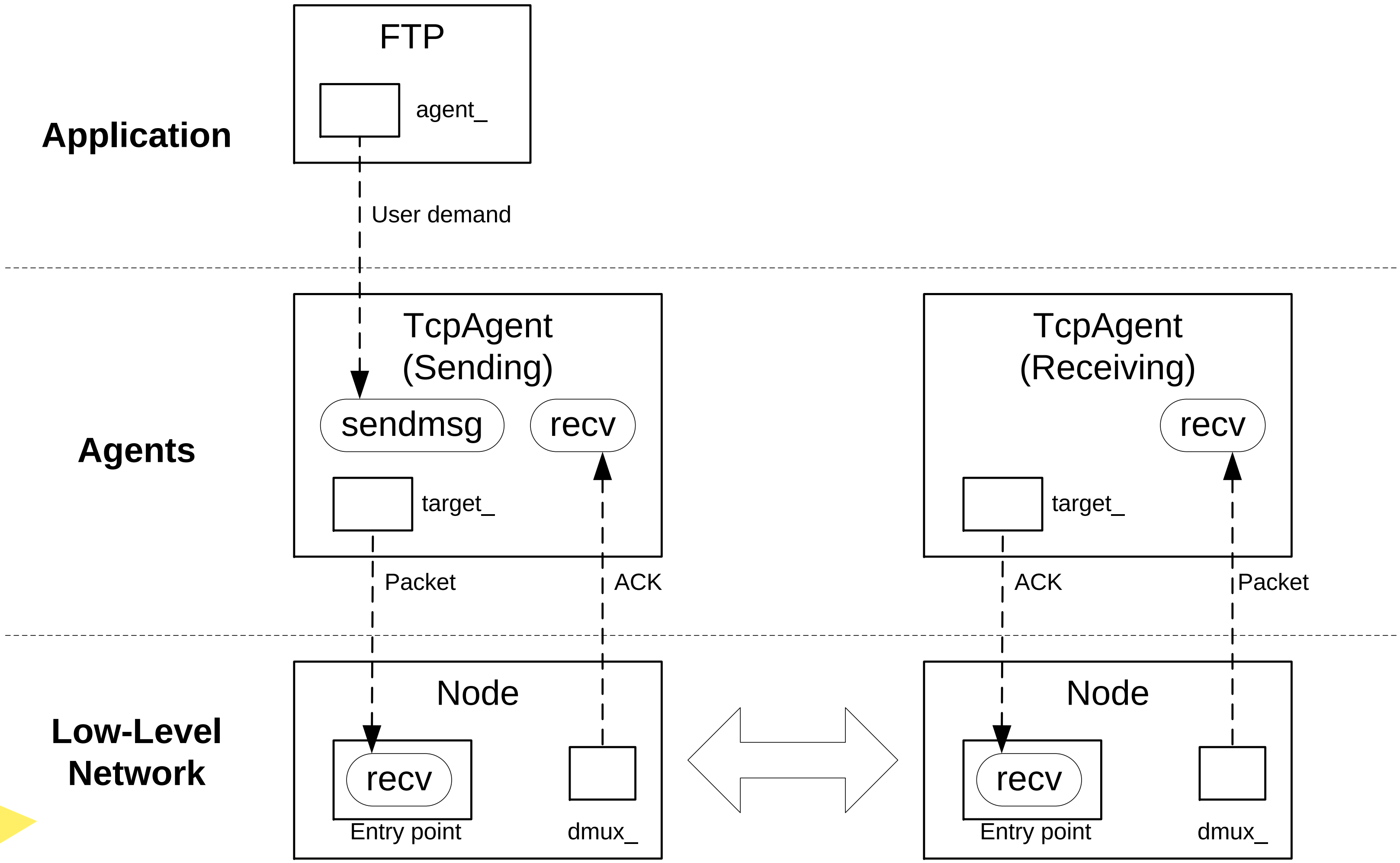
Outline

- Transmission Control Protocol (TCP)
- An Overview of NS2 Implementation
- TCP Receiver
- TCP Sender
- Summary

NS2 Implementation of TCP

- TCP Old-Tahoe
- TCP sender:
  - C++: `TcpAgent` ⇔ OTcl: `Agent/TCP`
- TCP receiver
  - C++: `TcpSink` ⇔ OTcl: `Agent/TCPSink`

TCP: Connection to Application and Low-Level Network



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

TCP Implementation in NS2

- TCP Receiving Agent
  - Responsibilities: Acknowledging packets
  - C++: `Agent` ➔ `TcpSink`
  - OTcl: `Agent` ➔ `Agent/TCPSink`
- TCP Sending Agent
  - Responsibilities: Sending packet, flow control, error control
  - C++: `Agent` ➔ `TcpAgent`
  - OTcl: `Agent` ➔ `Agent/TCP`
- TCP Header: `cmn_hdr`

Variable	Meaning
`seqno_`	Sequence number
`ts_`	Timestamp
`ts_echo_`	Timestamp echo
`reason_`	TX reason (e.g., 0 = normal)

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

Outline

- Transmission Control Protocol (TCP)
- An Overview of NS2 Implementation
- TCP Receiver
- TCP Sender
- Summary

A Guide for Creating a New Agent

1. Define the hierarchy: Based/derived classes
2. Define C++ and OTcl class variables
3. Define the constructor in both the hierarchy (bind the C++/OTcl variables here)
4. Implement the following key functions
   `sendmsg(nbyte)`, **`recv(p,h)`**, `timeout(tno)`
5. Define OTcl commands
6. Define timer (if necessary)

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

TCP Receiving Agent

- Two main classes

	Main	Helper
C++ class	TcpSink	Acker
OTcl class	Agent/TCPSink	None
Responsibility	TCP receiving agent	Support ACK creating

```
//~/ns/tcp/tcp-sink.cc
class TcpSink : public Agent {
public:
    TcpSink(Acker*);
    void recv(Packet* pkt, Handler*);
    int command(int argc, const char*const* argv);
protected:
    void ack(Packet*);
    Acker* acker_;
};
```

One main function

One helper function

One key variable

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

TCP Receiving Agent

- The constructor

```
TcpSink::TcpSink(Acker* acker) : Agent(PT_ACK), acker_(acker)  {...}
```

- Function recv(p,h):

```
//~/ns/tcp/tcp-sink.cc
void TcpSink::recv(Packet* pkt, Handler*)
{
    int numToDeliver;
    int numBytes = hdr_cmn::access(pkt)->size();
    hdr_tcp *th = hdr_tcp::access(pkt);
    numToDeliver = acker ->update(th->seqno(), numBytes);
    if (numToDeliver)
        recvBytes(numToDeliver);
    ack(pkt);
    Packet::free(pkt);
}
```

Tell acker_ that a new packet has arrived

Ack-ing the incoming packet, and destroy the packet

TCP Receiving Agent

- Function ack(p):

```
//~/ns/tcp/tcp-sink.cc
void TcpSink::ack(Packet* opkt)
{
    Packet* npkt = allocpkt();
    hdr_tcp *otcp = hdr_tcp::access(opkt);
    hdr_tcp *ntcp = hdr_tcp::access(npkt);
    ntcp->seqno() = acker_->Seqno();
    double now = Scheduler::instance().clock();
    ntcp->ts() = now;
    hdr_ip* oip = hdr_ip::access(opkt);
    hdr_ip* nip = hdr_ip::access(npkt);
    nip->flowid() = oip->flowid();
    send(npkt, 0);
}
```

Q: What are the types of `(npkt,0)`?
A: (   )

Helper Class Acker

- Responsibility: Maintain the status of received packets
- Declaration:

```
//~/ns/tcp/tcp-sink.h
class Acker {
public:
    Acker();
    inline int Seqno() const { return (next_ - 1); }
    int update(int seqno, int numBytes);
protected:
    int next_; int maxseen_; int wndmask_; int *seen_;
    int is_dup_;
};
```

Two key functions

Five key variables

Helper Class Acker

Variable	Meaning
seen_	An array: Index = Seq. No; Value = packet size
next_	Expected sequence number
maxseen_	Highest sequence number ever received
wndmask_	Modulus mask, initialized to maximum window size-1
is_dup_	True if the latest received TCP packet was received earlier

- Constructor

```
//~/ns/tcp/tcp-sink.cc
Acker::Acker() : next_(0), maxseen_(0), wndmask_(MWM)
{
    seen_ = new int[MWS];
    memset(seen_, 0, (sizeof(int) * (MWS)));
}
```

```
//~/ns/tcp/tcp-sink.cc
#define MWS 64
#define MWM (MWS-1)
```

Helper Class Acker

- Function `Seqno( )`: Return the seqno. under which all packets are received (i.e., `next_ -1`)

- Function `update(seq,numbyte)`
  - `seq` = Seq no. of the received packet
  - `numbyte` = packet size
  - Return no. of in-seq bytes ready to be delivered to the upper layer
  - Three cases :

    I) `seq < next_`,

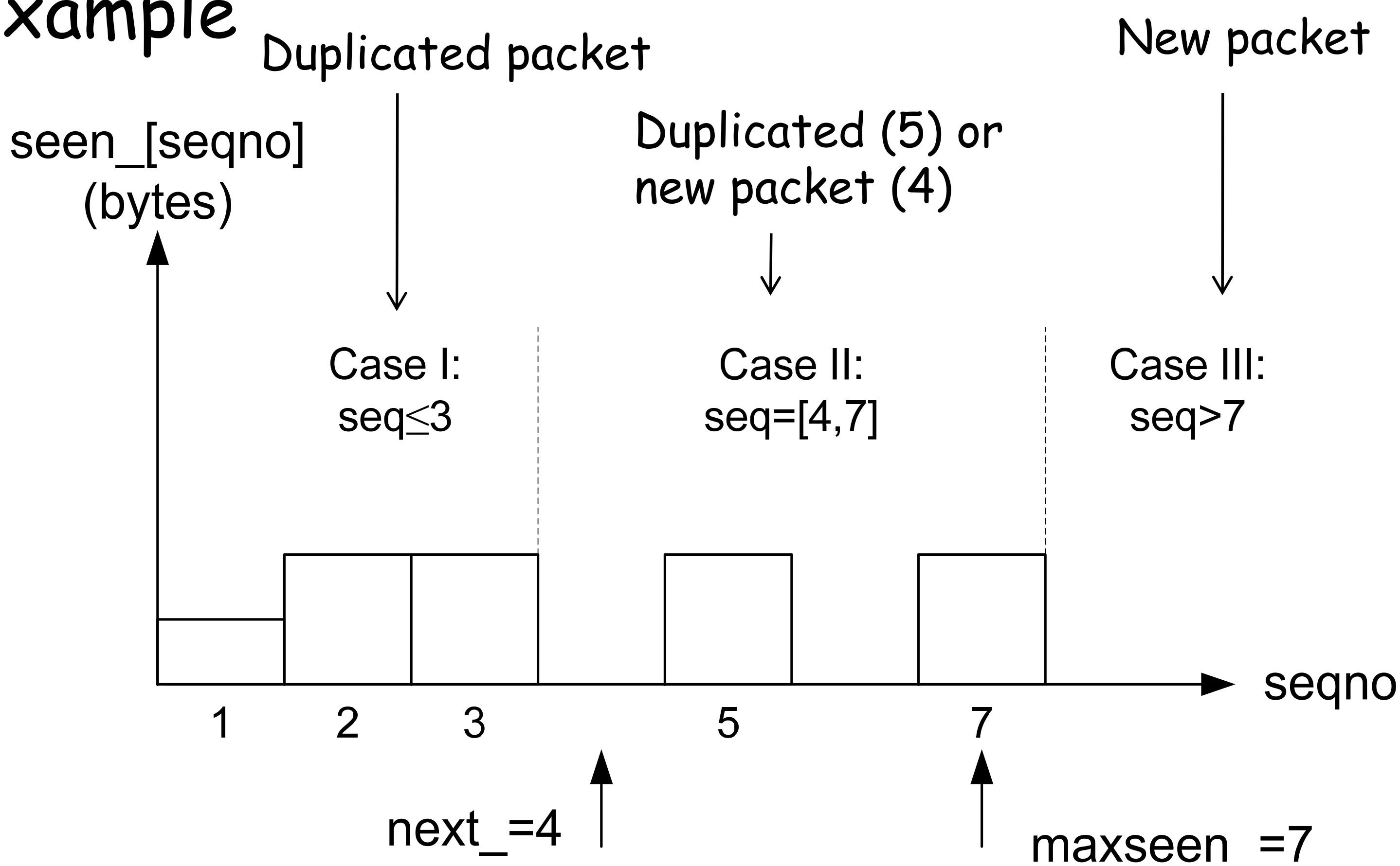
    II) `next_<= seq <=maxseen_`, and

    III) `seq > max_seen_`

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

Helper Class Acker

- Example



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

Helper Class Acker

- Function `update(seq,numbyte)`

  I) `seq < next_`: This is a duplicated packet

```
//~/ns/tcp/tcp-sink.cc
int Acker::update(int seq, int numBytes)
{
    if (seq < next)
        is_dup_ = TRUE;
    ...
}
```

Helper Class Acker

- Function update(seq,numbyte)

  III) seq > max_seen_: This is a new packet

```
//~/ns/tcp/tcp-sink.cc
int Acker::update(int seq, int numBytes)
{
    if (seq > maxseen_) {
        int i;
        for (i = maxseen_ + 1; i < seq; ++i)
            seen_[i & wndmask_] = 0;
        maxseen_ = seq;
        seen_[maxseen_ & wndmask_] = numBytes;
        seen_[(maxseen_ + 1) & wndmask_] = 0;
        just_marked_as_seen = TRUE;
    }
    ...
}
```

- Clear the spaces bet. maxseen_ and seq
- Update maxseen_

Update seen_

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

Helper Class Acker

- Function `update(seq,numbyte)`

  `II) next_<=seq <=maxseen_:` *A missing packet*

Why are these being "&" with `wndmask_`?
Hint: `wndmask_` = max. window size – 1.

This is a dup. packet

```
//~/ns/tcp/tcp-sink.cc
int Acker::update(int seq, int numBytes)
{
    ...
    if (seq >= next && seq <= maxseen_) {
        if (seen_[seq & wndmask_] && !just_marked_as_seen)
            is_dup_ = TRUE;
        seen_[seq & wndmask_] = numBytes;
        while (seen_[next & wndmask_]) {
            numToDeliver += seen_[next & wndmask_];
            ++next;
        }
        next_ = next;
    }
}
```

Update `seen_`

Advance `next_`

Outline

- Transmission Control Protocol (TCP)
- An Overview of NS2 Implementation
- TCP Receiver
- TCP Sender
- Summary

TCP Implementation in NS2

- TCP Receiving Agent
  - Responsibilities: Acknowledging packets
  - C++: `Agent` ➔ `TcpSink`
  - OTcl: `Agent` ➔ `Agent/TCPSink`
- TCP Sending Agent
  - Responsibilities: Sending packet, flow control, error control
  - C++: `Agent` ➔ `TcpAgent`
  - OTcl: `Agent` ➔ `Agent/TCP`

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

A Guide for Creating a New Agent

1. Define the hierarchy: Based/derived classes
2. Define C++ and OTcl class variables (see Tables 10.1-10.4 in the book)
3. Define the constructor in both the hierarchy (bind the C++/OTcl variables here)
4. Implement the following key functions
   `sendmsg(nbyte), recv(p,h), timeout(tno)`
5. Define OTcl commands
6. Define timer (if necessary)

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

Constructor

```
//~/ns/tcp/tcp.h
lass TcpAgent : public Agent {
   ...
   protected:
   RtxTimer rtx_timer_;
   ...

//~/ns/tcp/tcp.cc
TcpAgent::TcpAgent() : Agent(PT_TCP), rtx_timer_(this), ...
{   ...    }
```

Q: What is the C++ base class of NS2 timers?
A:

- Initialize Agent with PT_TCP
- Associated `rtx_timer_` with itself

(Discuss later in this lecture)

Function Overview

- 4 Categories

1. Packet transmission functions
2. ACK processing functions
3. Timer related functions
4. Window adjustment functions

Packet transmission functions

- `sendmsg(nbytes)`: Sends `nbytes` of application payload. If `nbytes=-1`, the payload is assumed to be infinite. The only public pkt TX function.

Put this in `hdr_tcp::reason_`

- `sendmuch(force,reason,maxburst)`: Sends as many packets as (but not more than ``maxburst'' packets) the TX window allows.

- `send_one()`: Sends one TCP packet with a sequence number `t_seqno_`.

- `output(seqno,reason)`: Creates and sends a packet with a sequence number and a transmission reason as specified by `seqno` and `reason`, respectively

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

Packet transmission functions

- Possible reason:
  - 0 = Regular Tx
  - 1 = Timeout
  - 2 = Duplicated ACK
  - 3 = Rate based pacing
  - 4 = Partial ACK
- Defined in `//~/ns/tcp/tcp.h`

Packet transmission functions

- Function sendmsg(nbytes):
  - Compute the no. of packets to be transmit, curseq_
  - Tell send_much(…) to transmit until the seq. no reaches curseq_

```
//~/ns/tcp/tcp.h
#define TCP_MAXSEQ 1073741824

 //~/ns/tcp/tcp.cc
void TcpAgent::sendmsg(int nbytes, const char* /*flags*/)
{
    if (nbytes == -1 && curseq_ <= TCP_MAXSEQ)
        curseq_ = TCP_MAXSEQ;
    else
        curseq_ += (nbytes/size_ + (nbytes%size_ ? 1 : 0));
    send_much(0, 0, maxburst_);
}
```

ACK Processing Functions

- `recv(p,h)`:
  - Main ACK reception function.
  - Determines whether the received packet `p` is a new ACK packet or a duplicated ACK packet, and acts accordingly.

- `recv_newack_helper(p)`:
  - Invoked from by `recv(p,h)` when a new ACK packet is received.
  - Invokes `newack(p)`
  - Opens the TX window if necessary.

ACK Processing Functions

- `newack(p)`:
  - Invoked from within `recv_newack_helper(p)`
  - Update variables
  - Restart the retransmission timer.

- `dupack_action()`:
  - Fast Retransmit
  - Invoked by `recv(p,h)` when a duplicated ACK packet is received.
  - Cut down the TX window,
  - Prepare the seq. no. of the lost packet for retransmission
  - Resets the retransmission timer.

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

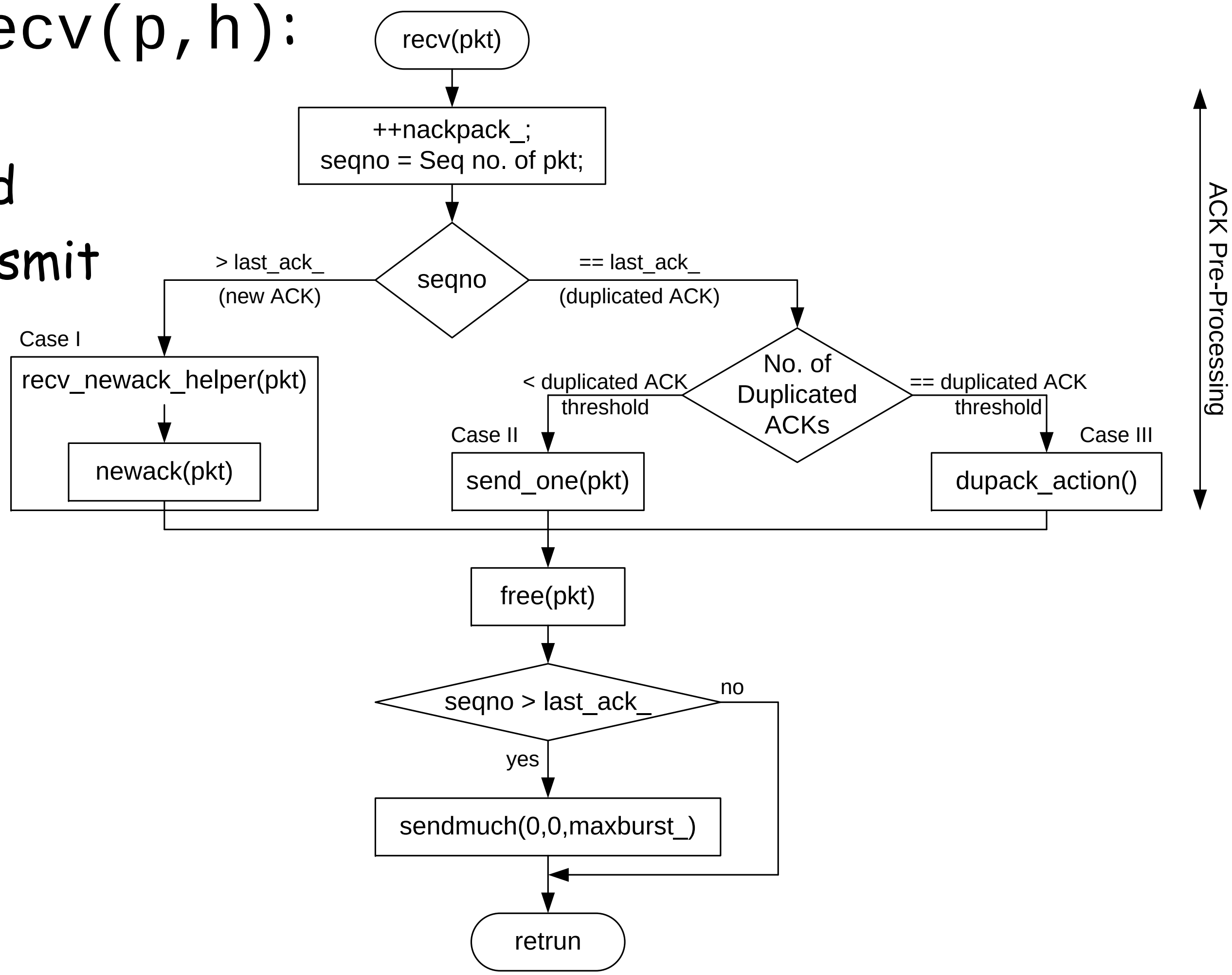
ACK Processing Functions

- Function `recv(p,h)`:

I) New ACK,

II) Dup. ACK, and

III) Fast Retransmit



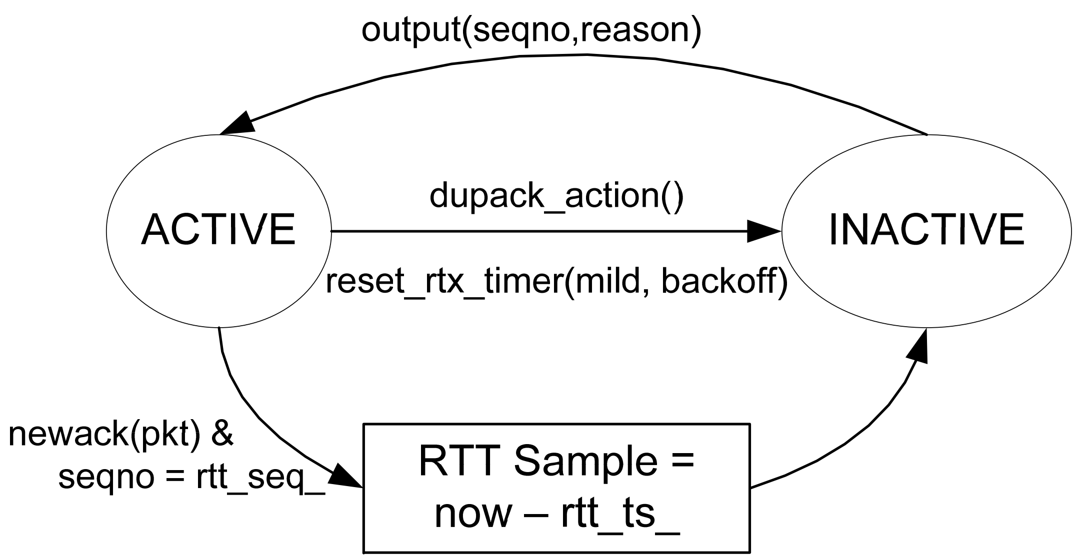
Textbook: T. Issariyakul and E. Hossain, *Introduction to Network Simulator NS2*, Springer 2008.

Timer Related Functions

- RTT Sample Collection
- RTT Estimation
- State Variables
- Retransmission Timers
- Function Overview

RTT Sample Collection

- RFC: Collect RTT for all packets
- NS2:
  - One set of RTT collection.
  - Active/Inactive



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

RTT Estimation

- Recall:

$$\bar{t}(k+1) = \alpha \times \bar{t}(k) + (1-\alpha) \times t(k+1), \alpha \in (0,1)$$
$$\sigma_t(k+1) = \beta \times \sigma_t(k) + (1-\beta) \times |t(k+1) - \bar{t}(k+1)|$$
$$RTO(k+1) = \min\{ub, \max\{lb, \gamma \times [\bar{t}(k+1) + 4 \times \sigma_t(k+1)]\}\}$$

- Equivalently,

$$\begin{aligned} \bar{t}(k+1) &= \frac{1}{8}\left(8\bar{t}(k) + \Delta\right) \\ \sigma_t(k+1) &= \frac{1}{4}\left(-\sigma_t(k) + 4\sigma_t(k) + |\Delta|\right) \\ RTO_u(k+1) &= \gamma \times [t(k+1) + 4\sigma_t(k+1)] \end{aligned}$$

Δ = ?

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

State Variables

Variable	Meaning
t_srtt_	$\bar{t}(k)$
t_rtt_var	$\sigma_t(k)$
T_SRTT_BITS	α
T_RTTVAR_BITS	β
t_backoff_	γ
t_rtxcur_	unbounded RTO

```
//~/ns/tcl/lib/ns-default.tcl
Agent/TCP set T_SRTT_BITS 3     #in bits
Agent/TCP set T_RTTVAR_BITS 2   #in bits
Agent/TCP set srtt_init_ 0      #in seconds
Agent/TCP set rttvar_init_ 12   #in seconds
Agent/TCP set rtxcur_init_ 3.0 #in seconds
Agent/TCP set T_SRTT_BITS 3     #in bits
Agent/TCP set T_RTTVAR_BITS 2   #in bits
Agent/TCP set rttvar_exp_ 2     #in bits
Agent/TCP set tcp_tick_ 0.1     #in seconds
Agent/TCP set maxrto_ 100000    #in seconds
Agent/TCP set minrto_ 0.2       #in seconds
```

```
//~/ns/tcp/tcp.cc
void TcpAgent::rtt_init()
{
    t_rtt_ = 0;
    t_srtt_ = int(srtt_init_ / tcp_tick_) << T_SRTT_BITS;
    t_rttvar_ = int(rttvar_init_ / tcp_tick_) << T_RTTVAR_BITS;
    t_rtxcur_ = rtxcur_init_;
    t_backoff_ = 1;
}
```

Q: What does this do?
A:

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

Retransmission Timer

- Implement timeout.
- Retransmit the lost packets when timeout
- Retransmitting all packets starting from `highest_ack_`
- Reset when receive an *ACK*.
- Implemented using `TimerHandler`

A Guideline to Implement a New Type of Timer

- Class `MyTimer`→`RtxTimer`; `actuator_`→`a_`
  1. Derive class `RtxTimer` from class `TimerHandler`.
  2. Declare a pointer `a_` to an `TcpAgent`object.
  3. Create a link to `a_` from the constructor.
  4. Define expiration action in function `expire(e)`.

- Class `ActionTaker`→`TcpAgent`; `timer_`→`rtx_timer_`
  1. Declare a pointer `rtx_timer_` to an `RtxTimer` object.
  2. Instantiate `rtx_timer_` with its `this` pointer from the constructor.

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

Retransmission Timer

- NS2 Implementation

```
//~/ns/tcp/tcp.h
class RtxTimer : public TimerHandler {
public:
    RtxTimer(TcpAgent *a) : TimerHandler() { a_ = a; }
protected:
    virtual void expire(Event *e);
    TcpAgent *a_;
};
```

```
//~/ns/tcp/tcp.cc
TcpAgent::TcpAgent() :
        Agent(PT_TCP),
        rtx_timer_(this), ...
{
        ...
}
```

```
//~/ns/tcp/tcp.cc
void RtxTimer::expire(Event*)
{
    a_->timeout(TCP_TIMER_RTX);
}

void TcpAgent::set_rtx_timer()
{
    rtx_timer_.resched(rtt_timeout());
}
```

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

Function Overview

- `rtt_update(tao)`:
  - Takes an RTT sample `tao` as an input
  - Updates `t_srtt_`, `t_rttvar_`, and `t_rtxcur_`
- `rtt_timeout()`:
  - Computes the bounded RTO value
  - Based on `t_rtxcur_`, `minrto_`, and `maxrto_`.
- `set_rtx_timer()`:
  - Restarts the retransmission timer.
- `reset_rtx_timer(mild,backoff)`:
  - Restart the retransmission timer
  - Cancel the RTT sample collecting process.
  - Set `t_seqno_` to `highest_ack_+1`, if `mild =0`.
  - Invoke `rtt_backoff()` if `backoff` is nonzero.

Function Overview

- `rtt_backoff()`:
  - Doubles the BEB multiplicative factor `t_backoff_`.
- `newtimer(pkt)`:
  - Takes an ACK packet `pkt` as an input argument.
  - Start the retransmission timer if TCP connection is active
  - Cancel the timer, otherwise.
- `timeout(tno)`:
  - Called by the retransmission timer
  - Close the congestion window,
  - Adjust `t_backoff_`,
  - Restart the retransmission timer and set up the seq. no. (`t_seqno_`) of the packet to be retransmitted.
  - Retransmits the lost packet (using `send_much(…)`)

Window Adjustment Functions

- Two main functions:

1. `opencwnd()`:

- Open the TX window.
- Slow-start or congestion avoidance: Depending on `cwnd_` and `ssthresh_`.

2. `slowdown(how)`:

- Close the TX window
- How much? ➔ See `how`.

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

Window Adjustment Functions

- Functions opencwnd():

```
//~/ns/tcp/tcp.cc
void TcpAgent::opencwnd()
{
    if (cwnd_ < ssthresh_) {
        cwnd_ += 1;
    } else {
        double increment = increase_num_ / cwnd_;
        cwnd_ += increment;
    }
    if (maxcwnd_ && (int(cwnd_) > maxcwnd_))
        cwnd_ = maxcwnd_;
}
```

W_{th}

W

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

Window Adjustment Functions

- Functions `slowdown(how)`:

```
//~/ns/tcp/tcp.cc
void TcpAgent::slowdown(int how)
{
    ...
    if (how & CLOSE_SSTHRESH_HALF)
            ssthresh_ = (int) halfwin;
    else if (how & THREE_QUARTER_SSTHRESH)
        ssthresh_ = (int)(3*cwnd_/4);

    if (how & CLOSE_CWND_HALF)
            cwnd_ = halfwin;
    else if (how & CWND_HALF_WITH_MIN) {
         cwnd_ = decreasewin;
    ...
}
```

Window Adjustment Functions

- Possible values of how

```
//~/ns/tcp/tcp.h
#define CLOSE_SSTHRESH_HALF       0x00000001
#define CLOSE_CWND_HALF           0x00000002
#define CLOSE_CWND_RESTART        0x00000004
#define CLOSE_CWND_INIT           0x00000008
#define CLOSE_CWND_ONE            0x00000010
#define CLOSE_SSTHRESH_HALVE      0x00000020
#define CLOSE_CWND_HALVE          0x00000040
#define THREE_QUARTER_SSTHRESH    0x00000080
#define CLOSE_CWND_HALF_WAY       0x00000100
#define CWND_HALF_WITH_MIN        0x00000200
#define TCP_IDLE                  0x00000400
#define NO_OUTSTANDING_DATA       0x00000800
```

Window Adjustment Functions

- The use of how
- If how = CLOSE_SSTHRESH_HALF

  ➔(how&CLOSE_SSTHRESH_HALF)=1

```
//~/ns/tcp/tcp.cc
void TcpAgent::slowdown(int how)
{
    ...
    if (how & CLOSE_SSTHRESH_HALF)
            ssthresh_ = (int) halfwin;
    else if (how & THREE_QUARTER_SSTHRESH)
        ssthresh_ = (int)(3*cwnd_/4);

    if (how & CLOSE_CWND_HALF)
            cwnd_ = halfwin;
    else if (how & CWND_HALF_WITH_MIN) {
         cwnd_ = decreasewin;
    ...
}
```

Window Adjustment Functions

- Due to the structure of how, we can also put several how-to into how
- For example, we can set

  how = CLOSE_SSTHRESH_HALF&CLOSE_CWND_HALF

  ➔(how&CLOSE_SSTHRESH_HALF)=1

  ➔(how&CLOSE_CWND_HALF)=1

```
void TcpAgent::slowdown(int how)
{
    ...
    if (how & CLOSE_SSTHRESH_HALF)
            ssthresh_ = (int) halfwin;
    if (how & CLOSE_CWND_HALF)
            cwnd_ = halfwin;
    else if (how & CWND_HALF_WITH_MIN) {
         cwnd_ = decreasewin;
    ...
}
```

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

Outline

- Transmission Control Protocol (TCP)
- An Overview of NS2 Implementation
- TCP Receiver
- TCP Sender
- Summary

Summary

- TCP (Transmission Control Protocol)
  - App-NW layers bridge
  - Control TX rate using flow control
  - Provide reliability using error control
- Flow control
  - Increase TX Rate: 1. and 2.
  - Decrease TX Rate: 1. and 2.
- Error Control
  - Acknowledgement
  - Timeout
  - Fast Retransmit

Summary

- TCP Receiver ( )
- A Helper Class Acker
  - Maintain packet reception status
  - Generate ACK number
- TCP Sender ( )
- 4 Main class of functions
  - Packet transmission functions
  - ACK processing functions
  - Timer related functions
  - Window adjustment function

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