Network Programming

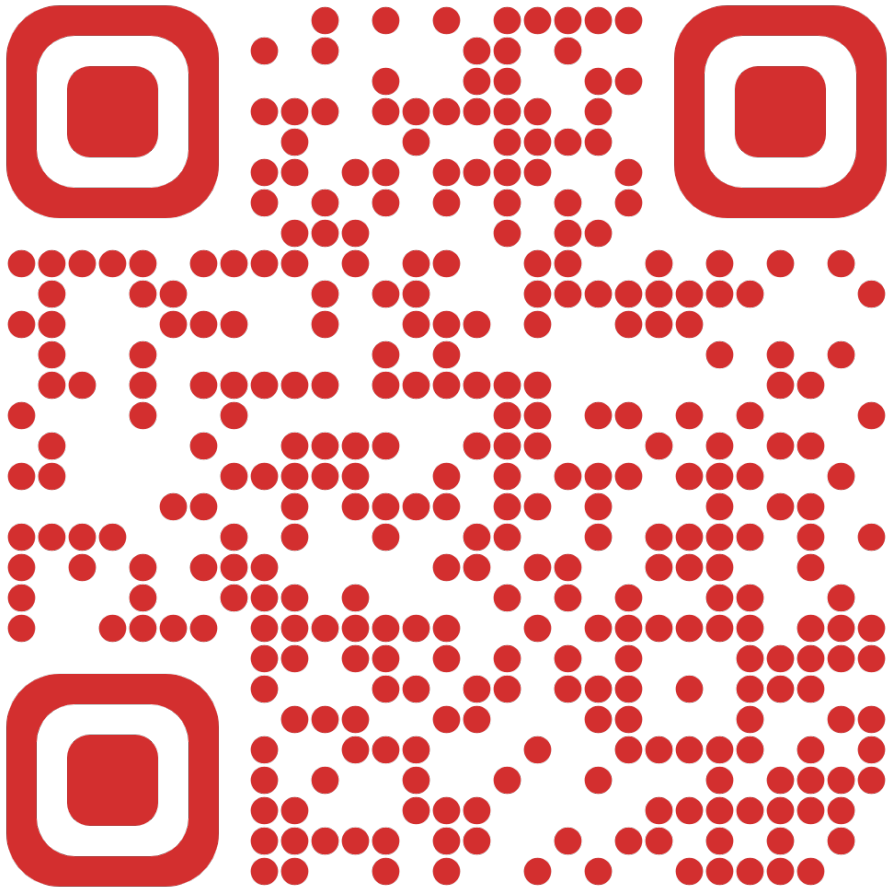
Simulating Flow Control using Socket Programming

Rathindra Nath Dutta

Senior Research Fellow
Advanced Computing & Microelectronics Unit
Indian Statistical Institute, Kolkata

October 30, 2023

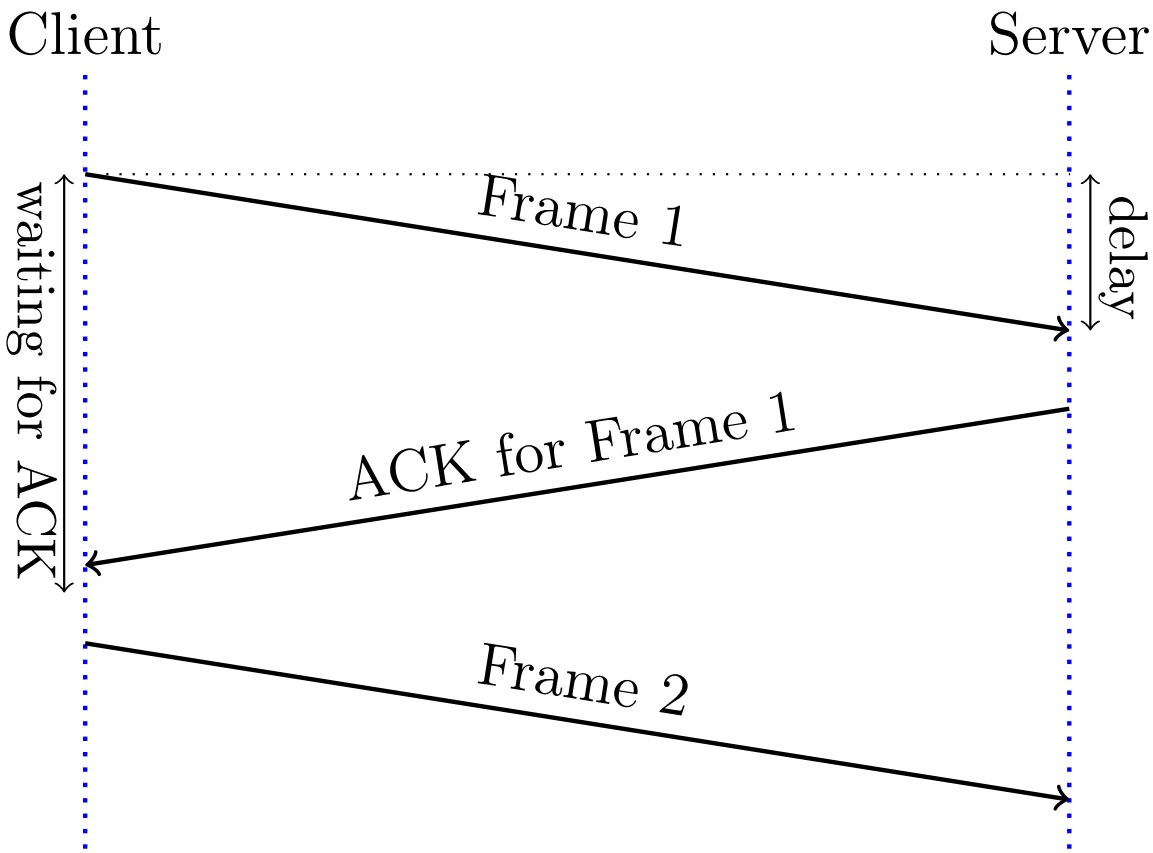
https://www.isical.ac.in/~rathin_r/uploads/CN/



WEB PAGE

Stop-and-Wait Protocol

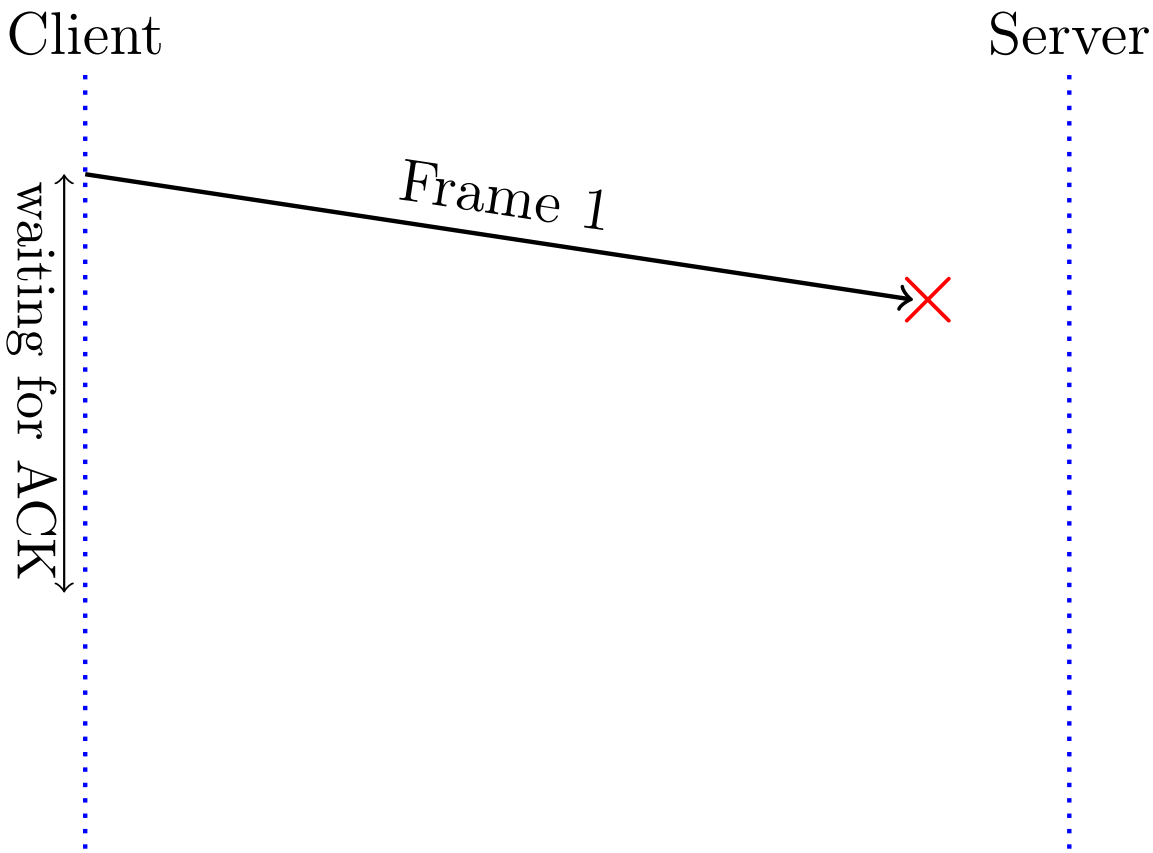
Simplest case of sliding window protocols



Normal operation

Stop-and-Wait Protocol

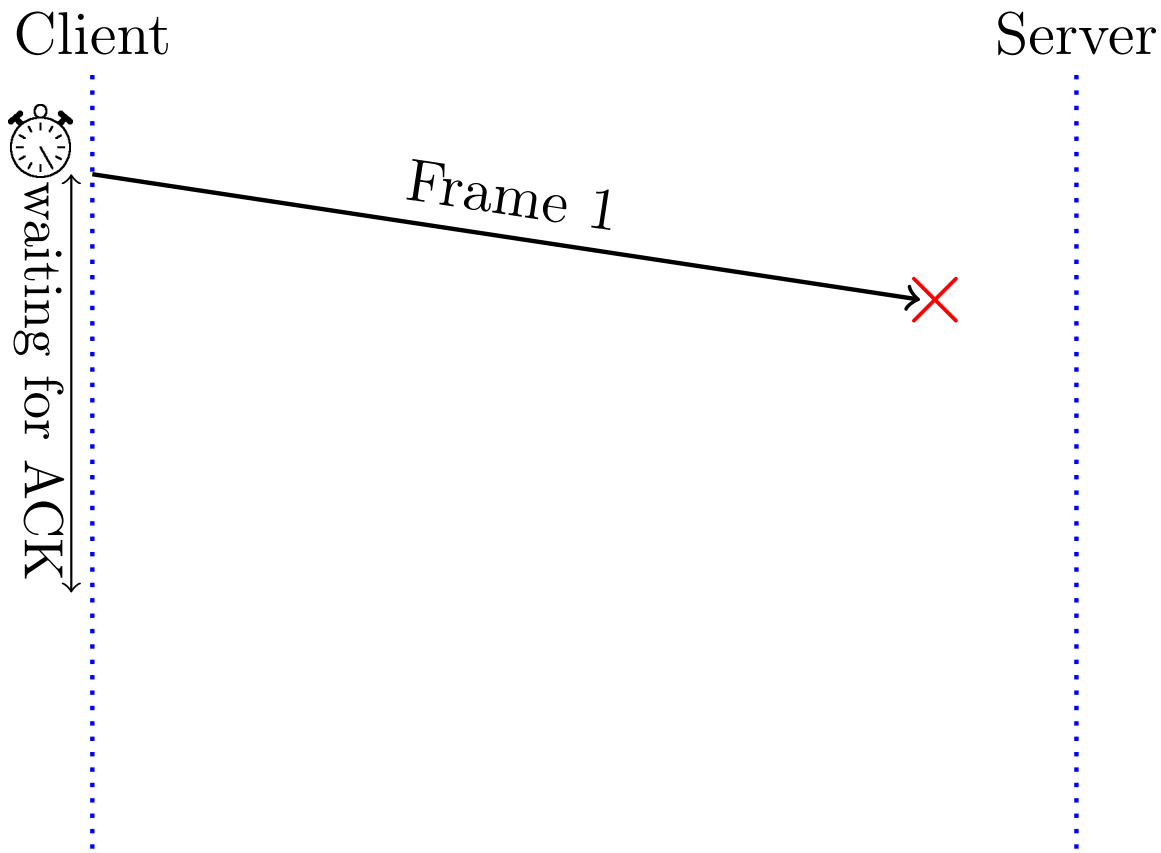
Simplest case of sliding window protocols



Dealing with lost frames

Stop-and-Wait Protocol

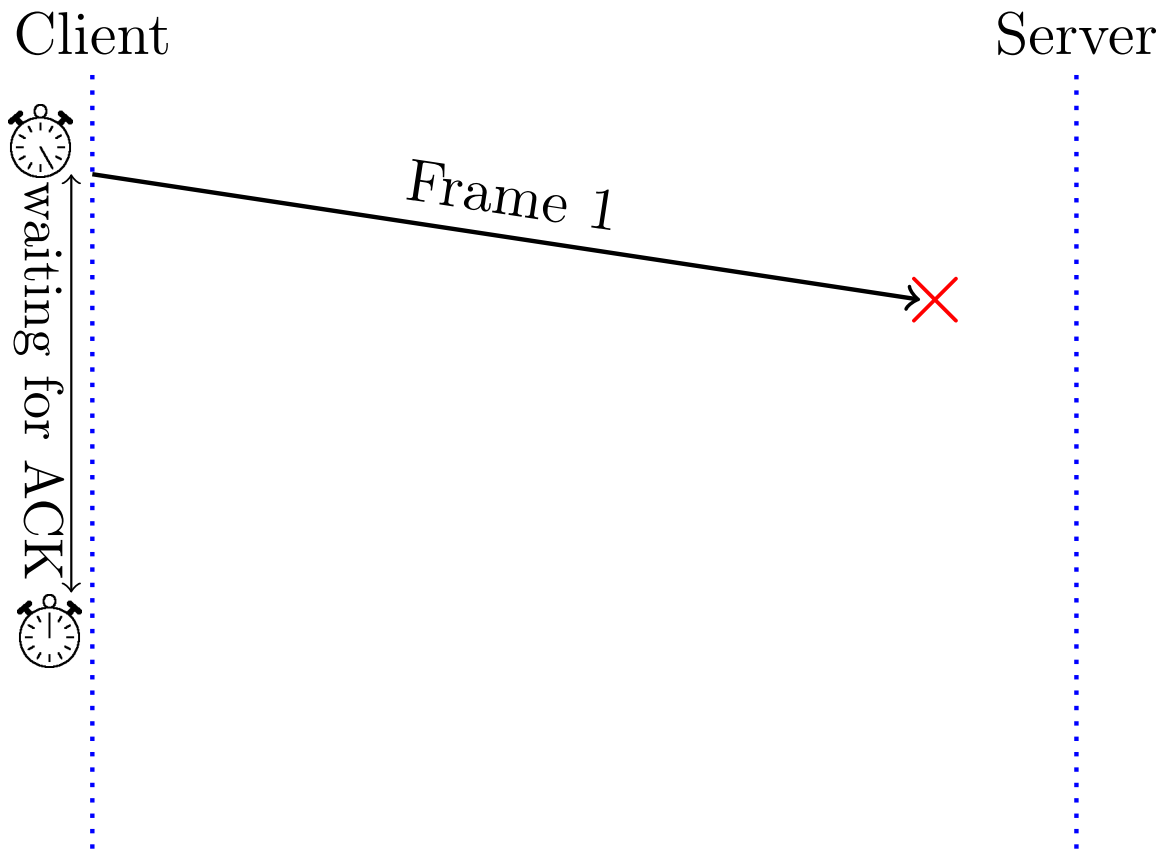
Simplest case of sliding window protocols



Dealing with lost frames

Stop-and-Wait Protocol

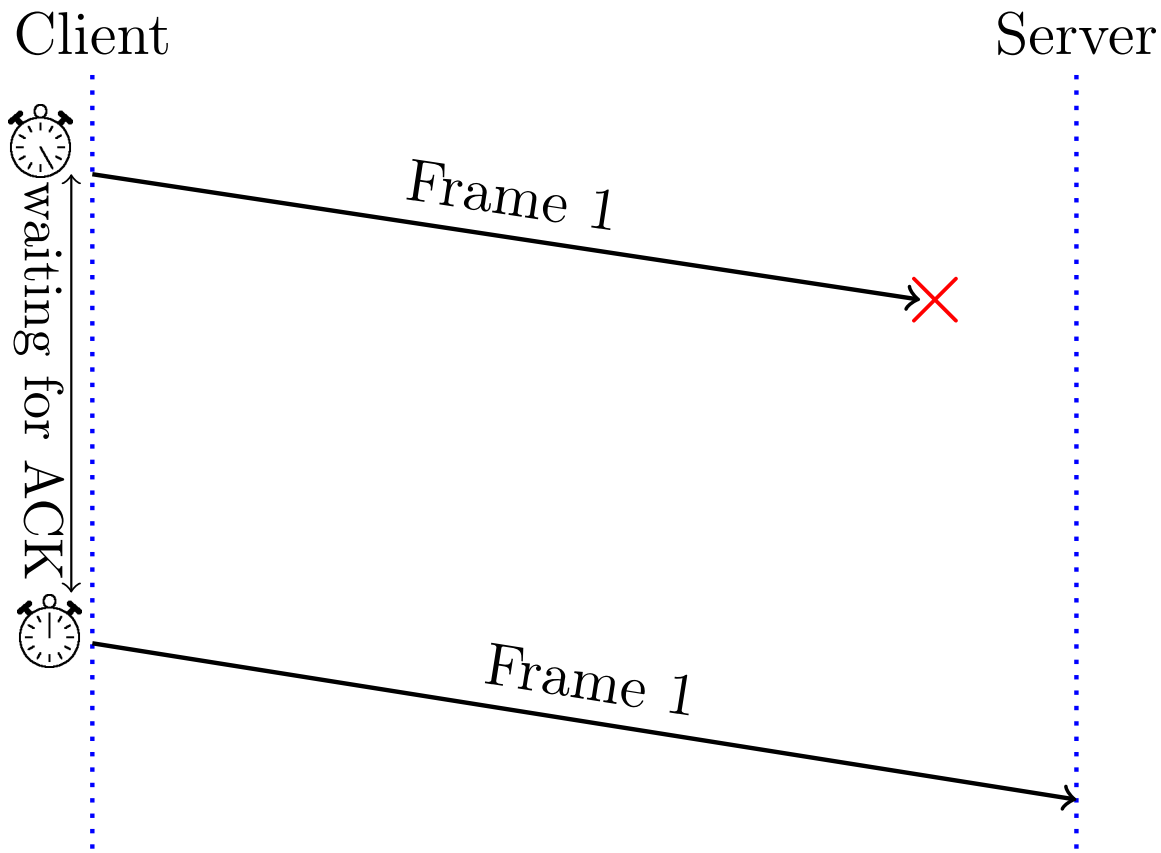
Simplest case of sliding window protocols



Dealing with lost frames

Stop-and-Wait Protocol

Simplest case of sliding window protocols



Dealing with lost frames

Prerequisite

- Some mechanism for simulating network delays

Prerequisite

- Some mechanism for simulating network delays
  – making a process wait

Prerequisite

- Some mechanism for simulating network delays
  – making a process wait
- Some mechanism to detect a timeout

Prerequisite

- Some mechanism for simulating network delays
  – making a process wait
- Some mechanism to detect a timeout
  – monitoring fds using `select()`

Prerequisite

- Some mechanism for simulating network delays
  – making a process wait
- Some mechanism to detect a timeout
  – monitoring fds using `select()`
- Some mechanism for simulating packet loss

Prerequisite

- Some mechanism for simulating network delays
  – making a process wait
- Some mechanism to detect a timeout
  – monitoring fds using `select()`
- Some mechanism for simulating packet loss
  – deterministic/random

Making a Process Wait

- The simplest way is to keep the process busy doing some 'useless' computation/work

Making a Process Wait

- The simplest way is to keep the process busy doing some 'useless' computation/work

```
int i = 100000 // some large number
while (i--); // keep the process busy
/* busy waiting ended, do useful work */
```

- It is also known as *busy waiting*

Making a Process Wait

- The simplest way is to keep the process busy doing some 'useless' computation/work

```
int i = 100000 // some large number
while (i--); // keep the process busy
/* busy waiting ended, do useful work */
```

- It is also known as *busy waiting*
- The time spend on the `while` loop is platform dependent
- How to wait for some 'exact' amount of time

Making a Process Wait

- The simplest way is to keep the process busy doing some 'useless' computation/work

```
int i = 100000 // some large number
while (i--); // keep the process busy
/* busy waiting ended, do useful work */
```

- It is also known as *busy waiting*
- The time spend on the `while` loop is platform dependent
- How to wait for some 'exact' amount of time

  – utilize the system clock[1]

```
time_t before = time(NULL);
while(time(NULL) - before < 10); // wait 10 seconds
```

[1] See man page for `time()`: https://man7.org/linux/man-pages/man2/time.2.html

Avoiding Busy Waiting

- The last solution wastefully consumes precious CPU resource

Avoiding Busy Waiting

- The last solution wastefully consumes precious CPU resource
- A better alternative is to simply block the process for the desired amount of time, and after the desired wait time has been elapsed the process resumes its work

Avoiding Busy Waiting

- The last solution wastefully consumes precious CPU resource
- A better alternative is to simply block the process for the desired amount of time, and after the desired wait time has been elapsed the process resumes its work
- UNIX provides `sleep()` system call[1]

[1] See man page for `time()`: https://man7.org/linux/man-pages/man2/time.2.html

Avoiding Busy Waiting

- The last solution wastefully consumes precious CPU resource
- A better alternative is to simply block the process for the desired amount of time, and after the desired wait time has been elapsed the process resumes its work
- UNIX provides `sleep()` system call[1]

```
sleep(10); // wait 10 seconds
```

[1] See man page for `time()`: https://man7.org/linux/man-pages/man2/time.2.html

Avoiding Busy Waiting

- The last solution wastefully consumes precious CPU resource
- A better alternative is to simply block the process for the desired amount of time, and after the desired wait time has been elapsed the process resumes its work
- UNIX provides `sleep()` system call[1]

```
sleep(10); // wait 10 seconds
```

Take a look at `waiting.c`

[1] See man page for `time()`: https://man7.org/linux/man-pages/man2/time.2.html

Monitoring File Descriptors

- Recall that invoking a `read()` call indefinitely blocks the current process until some input arrives, or some error occurs

Monitoring File Descriptors

- Recall that invoking a `read()` call indefinitely blocks the current process until some input arrives, or some error occurs

- To avoid this indefinite wait we can monitor a file descriptor for some desired I/O event
- And only invoke `read()` when some input arrives

Monitoring File Descriptors

- Recall that invoking a `read()` call indefinitely blocks the current process until some input arrives, or some error occurs

- To avoid this indefinite wait we can monitor a file descriptor for some desired I/O event
- And only invoke `read()` when some input arrives

- For this monitoring we will use the `select()` API[1]

[1] See man page: https://man7.org/linux/man-pages/man2/select.2.html

The `select()` API

```
#include <sys/select.h>
int select(int nfds, fd_set *readfds, fd_set *writefds,
           fd_set *exceptfds, struct timeval *timeout);
```

Return value: Returns the total number of fds are that ready.
On timeout, zero is returned. On error, -1 is returned.

Parameters:
`nfds`: set to the highest-numbered file descriptor in any of the three sets, plus 1
`readfds`: set of fds that are watched to see if they are ready for reading
`writefds`: set of fds that are watched to see if they are ready for writing
`exceptfds`: set of fds that are watched for 'exceptional conditions'
`timeout`: specifies the interval that `select()` should block waiting for a file descriptor to become ready

[1] `fd_set` is a special type that can represent a set of file descriptors

The `select()` API (contd.)

```
struct timeval {
    time_t      tv_sec;  /* seconds */
    suseconds_t tv_usec; /* microseconds */
};
```

The select() API (contd.)

```
struct timeval {
    time_t      tv_sec;  /* seconds */
    suseconds_t tv_usec; /* microseconds */
};
void FD_ZERO(fd_set *set);
// removes all file descriptors from the set

void FD_SET(int fd, fd_set *set);
// adds the file descriptor fd to the set

void FD_CLR(int fd, fd_set *set);
// removes the file descriptor fd from the set

int  FD_ISSET(int fd, fd_set *set);
// test if a file descriptor is still present in a set
```

All the three fd sets supplied to the `select()` call are updated
They must be reinitialized before making successive `select()` call

Timed Input

Take a look at `timeout.c`

Stop-and-Wait (without any packet loss)

Take a look at `stop_wait_server.c` and `stop_wait_client.c`

Simulating Frame Loss

- A simple strategy is to conditionally send a frame

Simulating Frame Loss

- A simple strategy is to conditionally send a frame

- The condition can be *randomized*

```
if( rand()  <  0.5 * RAND_MAX ) { // some threshold
    /* send the frame */
} else { /* do not send */ }
```

Simulating Frame Loss

- A simple strategy is to conditionally send a frame

- The condition can be *randomized*

```
if( rand()  <  0.5 * RAND_MAX ) { // some threshold
    /* send the frame */
} else { /* do not send */ }
```

- The condition can be *deterministic* as well

```
int counter = 0; // global scope or value-result argument
...
if( counter % 3 == 0 ) { // some condition
    /* do not send every third frame*/
} else { /* send the frame */ }
counter = (counter + 1) % 3 // update counter
```

Practice Problems

- Modify `stop_wait_client.c` to incorporate frame loss
  Assume no ACK is lost

Practice Problems

- Modify `stop_wait_client.c` to incorporate frame loss
  Assume no ACK is lost

- Extend these ideas to simulate Go-Back-N Protocol for flow control