Recitation 7: PA2

Outline

Important Dates

- Midterm Exam
  - Due Date: Monday, November 16 at 11:30 AM
- Wireshark Lab 4
  - Due Date: Monday, November 23 at 11:59 PM
- Programming Assignment 2
  - Due Date: Wednesday, December 9 at 11:59 PM

Part 1:
Overview of Reliable Transport Protocols

Objectives

- Objective 1: In a given simulator, implement three reliable data transport protocols.
  - Alternating-Bit 3.0 (ABT), Go-Back-N (GBN), and Selective-Repeat (SR)
  - IMPORTANT: You must use the pseudocode from the textbook/lectures to implement the above protocols.
- Objective 2: Afterwards, you'll test, analyze, and discuss your results in the project report

- Code-wise, PA2 is much easier than PA1.
  - There is NO socket programming.

Reliable Data Protocols

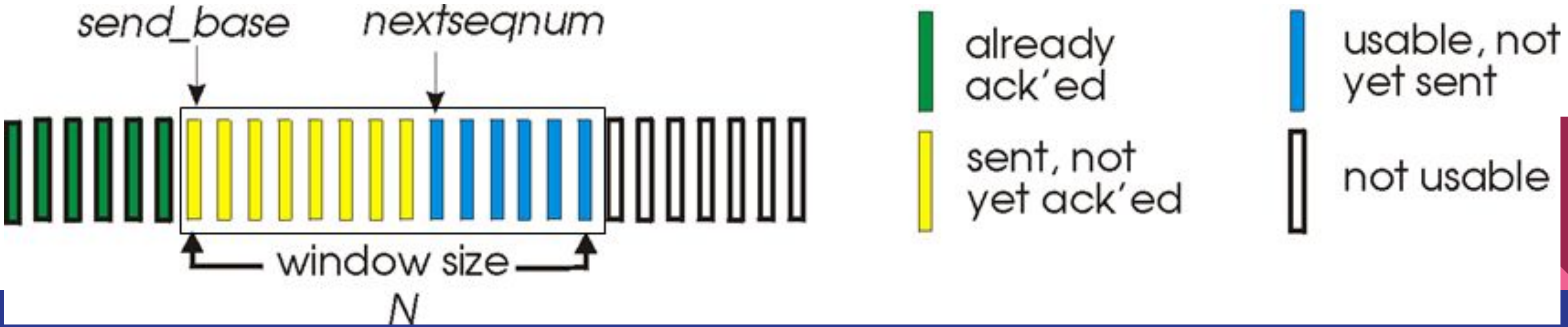
- Reason for Reliable Data Protocols: underlying channel can lose and corrupt packets (data or ACKs)
    - checksum, seq. #, ACKs, retransmissions will be of help.
- In order to implement a reliable data protocol, you'll have to use basic checksums, sequence numbers, and acknowledgements.
- When the sender detects a packet loss or packet corruption, it will retransmit the packet.

The Alternating-Bit (rdt 3.0)

- Main Approach: sender waits “reasonable” amount of time for ACK
  - Retransmits if no ACK received in this time
  - If pkt (or ACK) just delayed (not lost):
    - retransmission will be duplicate, but use of seq. #’s already handles this
    - receiver must specify seq # of pkt being ACKed
  - Requires countdown timer
- Sender Window:
  - Only 1 unAck’ed packet is allowed in transit at any given time
- timer for the single unAcked pkt:
  - on timeout: retransmit the single unAcked pkt
- Receiver sends back an ACK for each packet:
  - Sender must wait for ACK before sending the next packet.
- Implement rdt 3.0 sender and rdt 2.2 receiver
  - Pseudocode found on pages 216 and 217 of the course textbook.

Go-Back-N

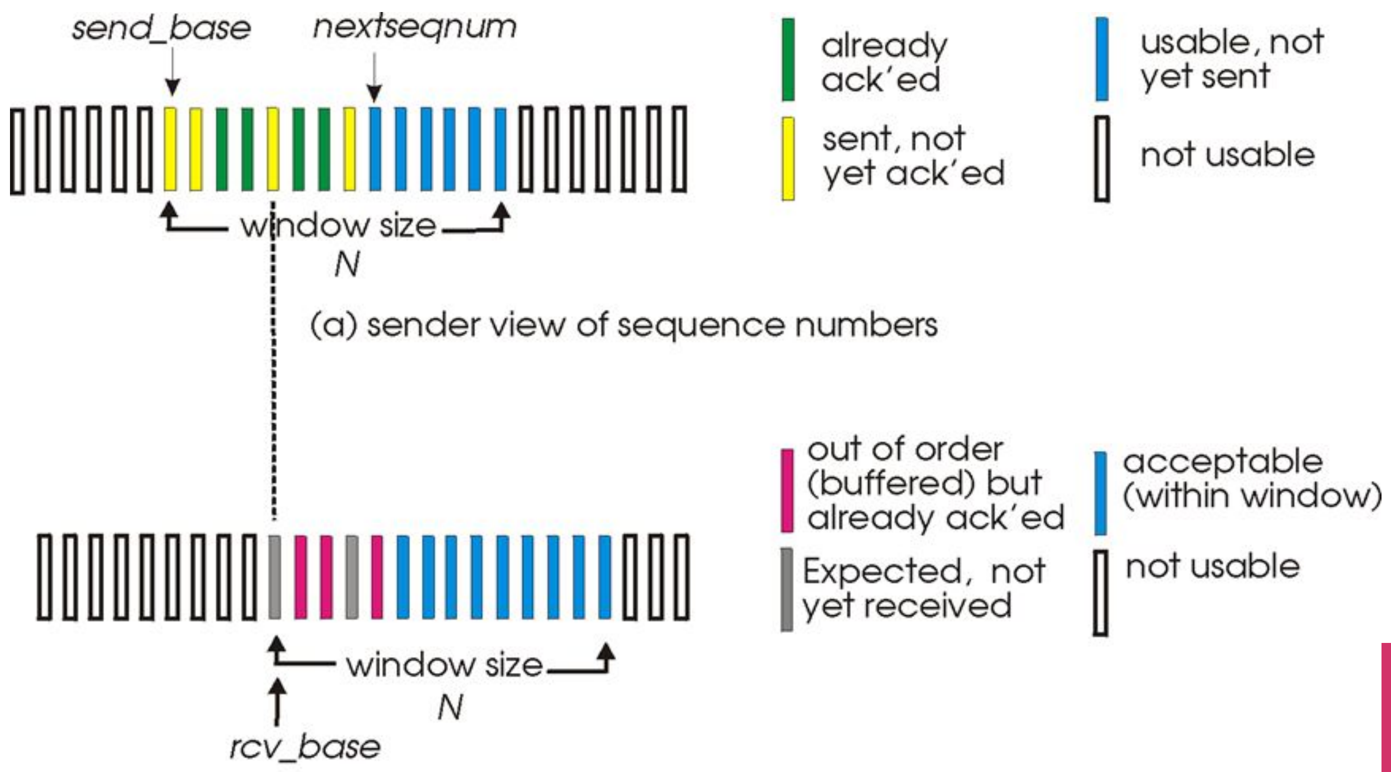
- Sender Window:
  - “window” of up to N, consecutive unAck’ed packets are allowed in transit at any given time
- timer for oldest unAcked pkt:
  - on timeout: retransmit oldest unAcked pkt and all higher seq # pkts in window
- Receiver sends back cumulative ACKs:
  - may receive duplicate ACKs (see receiver)
  - Receiver ignores out-of-order packets
- Refer to page 223 in textbook for pseudocode



Selective-Repeat

- Sender Window (like GBN):
  - “window” of up to N, consecutive unAck’ed packets are allowed in transit at any given time
  - again limits seq #s of sent, unACK’ed pkts
- timer for each sent, but currently unAcked pkt:
  - on timeout for packet p: retransmit packet p only
  - sender only resends the packet whose timer has expired
- Receiver individually acknowledges all correctly received pkts
  - receiver buffers pkts, as needed, for eventual in-order delivery to upper layer
- Refer to page 228 for pseudocode

Selective-Repeat Diagram



(a) sender view of sequence numbers

(b) receiver view of sequence numbers

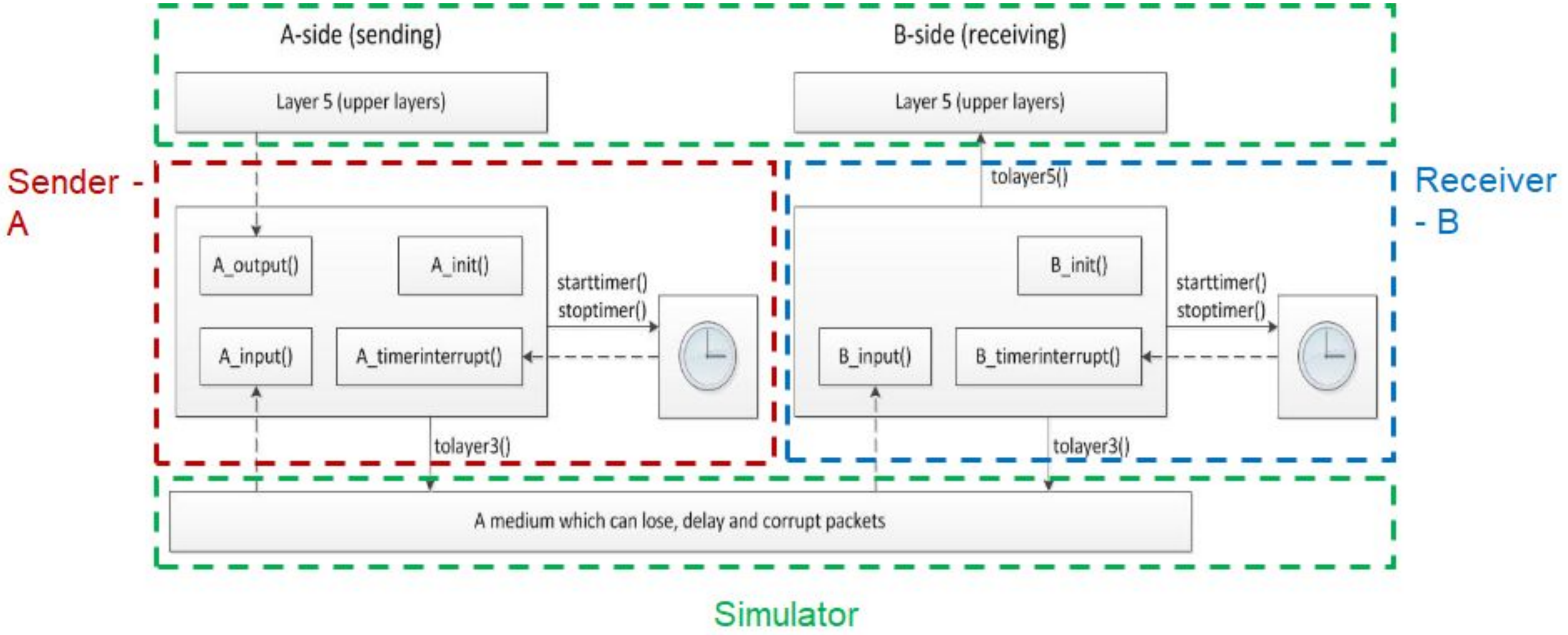
Selective-Repeat

- Unlike ABT and GBN, you’re not given all of the pseudocode for SR.
- You need to make your own algorithm that simulates N logical timers with one physical timer.
    - You’ll probably have to associate a retransmission time with each sent, but unACKed packet currently in the window.
- It’s not difficult and you’re encouraged to check with one of the TA’s to make sure your implementation is correct

Part 2:
Implementing the Protocols

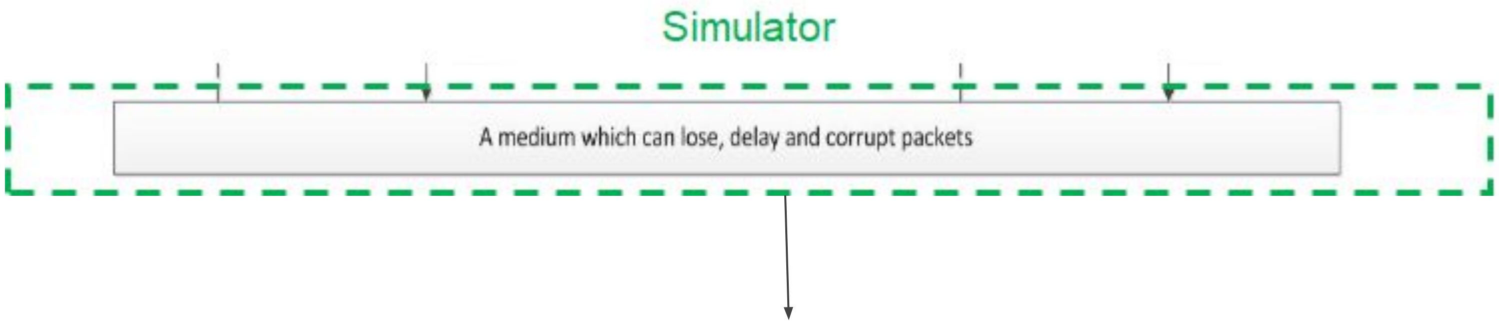
Programming Environment

- You'll be working in a simulated network environment
  - Even though it's simulated, the environment closely mirrors an actual UNIX environment



Design Goals

- To ensure the data generated from sender A's application layer is delivered in-order and correctly to the receiver B's application layer
  - Provide reliability over an unreliable channel



Data packets will be lost and/or corrupted at the lower layers

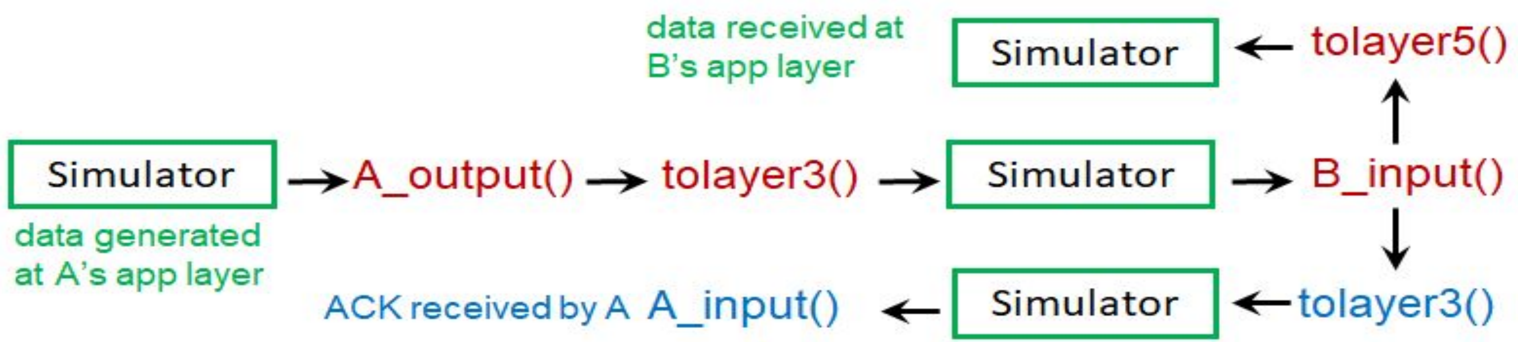
The Simulator

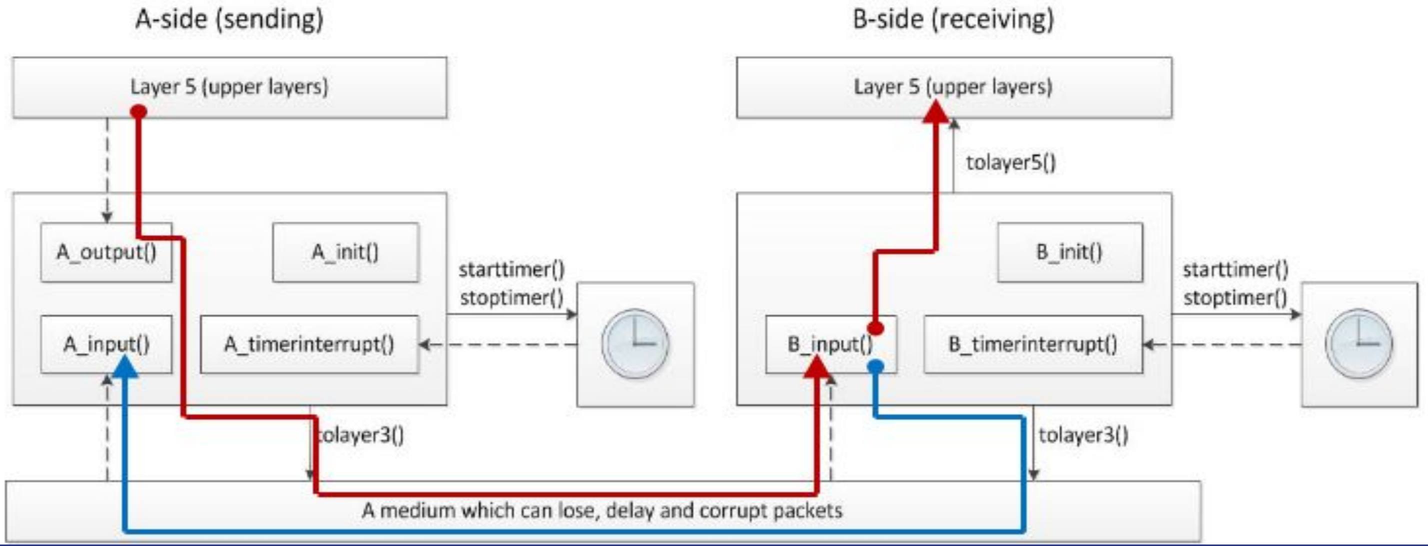
- Simulated hosts A and B run on single real host
- Hosts A and B “communicate” over simulated network
  - Packet loss/corruption and other parameters are specified as command-line arguments
- Time is self-contained in the simulated environment
  - Both simulated hosts have the same time measured in “time units”
  - Simulated time different from real time!
  - Use get_sim_time()
  - Don’t use functions from the time.h library

Code Structure and Flow

- The A-side will always be the sender
- The B-side will always be the receiver
- When A receives data from layer 5, perform necessary processing and forward it the B-side by passing the data to layer 3
- When B receives data from layer 3, perform necessary processing, and either discard the data if it’s corrupted or forward it to layer 5 and generate an ACK.
- When A receives data from layer 3, process the ACK and adjust window as necessary (may have to send next packet in new window as well)
- There’s also the timeout event that triggers A to retransmit packet(s).

Code Structure and Flow

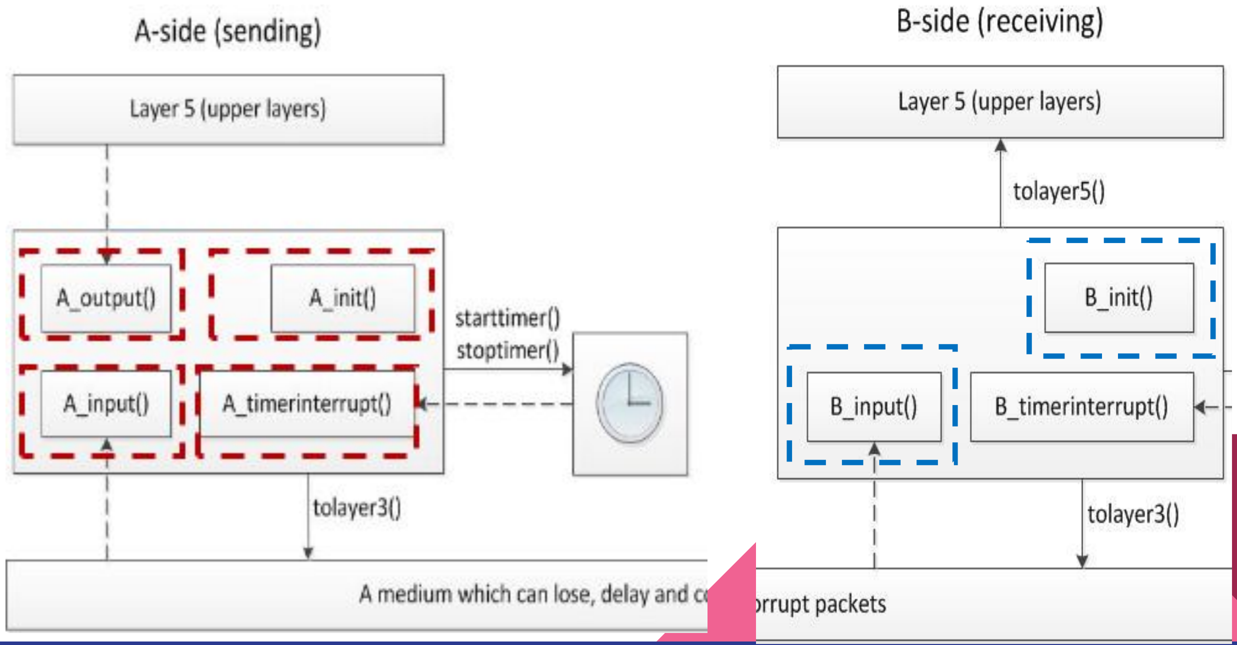




Functions You Will Write

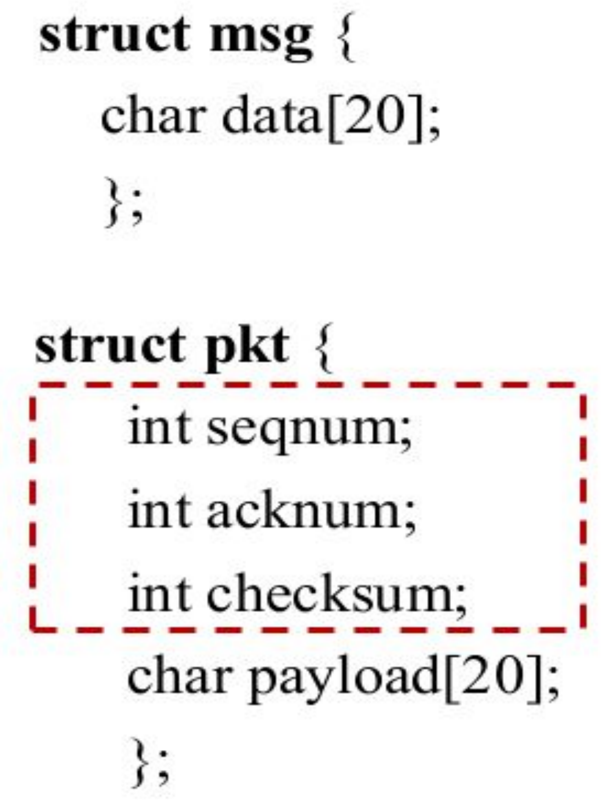
- A_output(message);
- A_input(packet);
- A_timerinterrupt();
- A_init();

- B_input(packet);
- B_init();



Messages and packets structure

- There are two different predefined data-types
  - struct msg
  - struct pkt
- struct msg comes from layer 5 and that needs to be put inside of struct pkt
- The only two types of data that can be recognized by the simulator.
- Both data packets and ack packets are of struct pkt type.
- Both data packets and ack packets need checksum
- Don't change the definitions of struct msg or struct pkt

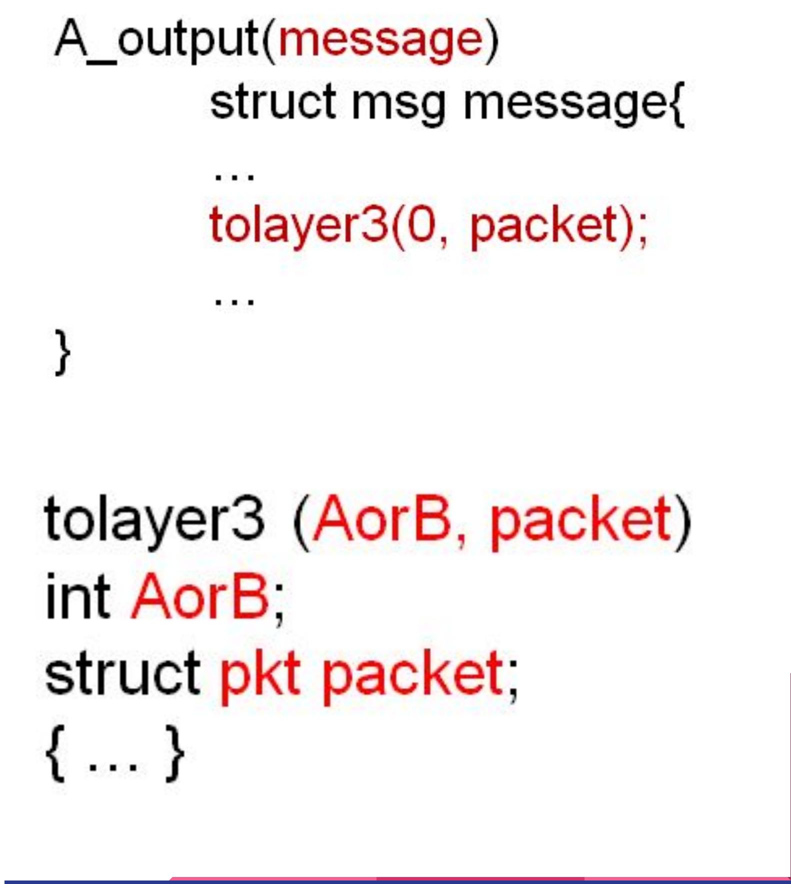


```
struct msg {
    char data[20];
};

struct pkt {
    int seqnum;
    int acknum;
    int checksum;
    char payload[20];
};
```

Functions

- A_output()
  - Called whenever the upper layer at sender A has a message to send
  - Function parameters - struct msg message
  - Will call tolayer3() inside after filling out sequence # and checksum in packet.
    - tolayer3() will put the struct pkt into the unreliable channel
- tolayer3() - Implemented by Simulator
  - Function parameters – AorB, packet
    - AorB parameter
      - int AorB
      - AorB = 0 [when being called by A]
      - AorB = 1 [when being called by B]
    - packet parameter
      - struct pkt packet



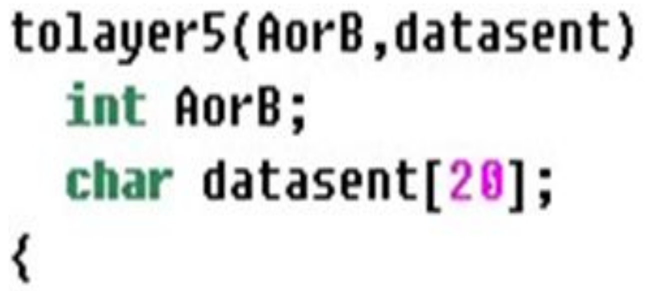
```
A_output(message)
        struct msg message{
        ...
        tolayer3(0, packet);
        ...
}

tolayer3 (AorB, packet)
int AorB;
struct pkt packet;
{ ... }
```

Functions

- B_input()
  - Function parameters - packet
  - Validate the checksum first.
  - May call tolayer3() and tolayer5() inside
- tolayer5() - Implemented by Simulator
  - Function parameters – AorB, datasent
    - AorB
      - int AorB
      - AorB = 0 // called by A
      - AorB = 1 // called by B
    - datasent // different from tolayer3() here
      - char datasent[20] // payload

```
B_input(packet) {
        // validate the checksum;
        ...
        // if the checksum is correct
        // send ack to A
        tolayer3(1, ack);
        // deliver the data to upper layer
        tolayer5(1, payload);
        ...
}
```



```
tolayer5(AorB,datasent)
  int AorB;
  char datasent[20];
{
```

Timer Functions

- If packet corruption and loss happen at the lower layers, what should we do?
  - Retransmit with timer!
- The timer and the functions to manipulate it are implemented by the simulator.
- starttimer(int calling_entity, float increment)
  - Starts a timer.
  - Function parameters – AorB, increment
    - AorB
      - int AorB
      - AorB = 0 // called by A
      - AorB = 1 // called by B
    - Increment
      - float increment
  - Only ONE timer provided by the simulator!

Timer Functions

- stoptimer(int calling_entity)
  - Its usage is similar to starttimer()
  - Stops the timer
- get_sim_time()
  - Returns the current simulator time as a float value.

- A_timerinterrupt()
  - Have to implement this function.
  - Will be called when A's timer expires
  - This function is used to handle packet retransmission.

Other Simulator Implemented Functions

- getwinsize()
    - Returns the window size as an int value.

Common Questions

- What value should we use in the timer?
    - Hint #1: A packet sent into the network takes an average of 5 time units to arrive at the other side when there are no other packets in the medium.
    - Hint #2: Experiment with this value! This is the only area where you can potentially optimize your protocol implementation.
        - Experimenting may lead to interesting results that will improve your report.
- If the data generating speed at A's app layer is higher than the speed at which lower layers send packets, what should we do?
    - Buffer!
    - You may assume that the buffer will grow no larger than a size of 1000

Common Questions

- How do I compute the checksum?
  - The checksum must include the data, seqnum and acknum fields for both data and ACK packets.
  - checksum = sum of the ascii values in message + acknum + seqnum
  - Recall that any content - payload, seqnum, acknum or checksum - can be corrupted by the simulator’s lower layers.
- When will the simulation end?
  - The simulation is controlled by the number of messages generated by A’s app layer – the number of packets you type in before the simulation start
  - The simulator will stop as soon as that number of messages have been passed down from A’s app layer.

Debugging

- The easiest way to debug your code is to add print statements and run the protocol executables directly.
- Generate these executables (`abt` & `gbn` & `sr`) by compiling your code with the template-provided `Makefile`.
- Example Run:
  - ./gbn -s 200 -m 50 -t 30 -c 0.2 -l 0.1 -w 20 -v 3
- These executables take the following arguments:
  - `-s 200` : The seed value is set to 200. Simulator runs with the same seed value should produce the same output.
  - `-m 50` : 50 messages are delivered to the transport layer on the sender side.
  - `-t 30` : An average of 30 seconds between messages arriving from the sender's application layer.
  - `-c 0.2` : An average of 20% of sent packets are corrupted.
  - `-l 0.1` : An average of 10% of sent packets are lost.
  - `-w 20` : The window size is set to 20 packets.
  - `-v 3` : Many debugging statements from the simulator are printed to the screen. Can use lower integer values to print less statements.

Running the Tests

- In order to test whether you implemented a reliable data protocol, run the three test suites included with the project template.
  - `./basic_tests` & `./sanity_tests` & `./advanced_tests`
- Example Run for ABT:
  - `./basic_tests -p ../<UBIT_NAME>/abt -r ./run_experiments`

- WARNING: The included tests just tell you whether you have implemented a reliable data protocol. They will not tell you whether you implemented ABT, GBN, or SR correctly.
  - In addition to running the tests, we will manually check your code in order to make sure that you actually implemented ABT, GBN, and SR correctly.

Additional Project Requirements

- Make sure your project successfully compiles and runs on the CSE Server assigned to you (stones, underground, embankment, euston, or highgate)
    - Do not compile your code manually. Use the Makefile provided by the template.
    - Use the same local directory path you used for PA1: `/local/Fall_2020/<Your-UBIT-Name>/`
    - See Piazza post @364 for your host assignment: https://piazza.com/class/kegjk8a65qz79w?cid=364
- C or C++
    - No Disk I/O
- Use the PA2 template **[MANDATORY]**
    - Template Instructions: https://goo.gl/G4cPfH
- This presentation only covers a subset of the entire project.
    - View the project description document for more details: https://goo.gl/KzTh0J

Part 3:
Experiment Analysis and Report

Experiments & Report

- For the second part of the project, you will have conduct 2 experiments and each experiment consists of multiple parts.
- You have to use the `./run_experiments` script on the dedicated host assigned to you when conducting the experiments.
- After running the experiments you will have to graph and analyze your results in a report.
- Every time the run_experiments script is run, you will receive 10 results in the output CSV file.
    - You can take the average of these 10 results for plotting the graphs.
    - Use the value of the data field called ‘Throughput’ for calculating the average.

Experiments & Report

- Example Experiment Run:
  - `./run_experiments -m 1000 -t 50 -c 0.2 -l 0.1 -w 0 -p ../<UBIT_NAME>/abt -o e1_l1_abt.csv`
- This command runs 10 experiments with the following parameters:
  - `-m 1000` : 1000 messages are delivered to the transport layer on the sender side.
  - `-t 50` : An average of 50 seconds between messages arriving from the sender's application layer.
  - `-c 0.2` : An average of 20% of sent packets are corrupted.
  - `-l 0.1` : An average of 10% of sent packets are lost.
  - `-w 0` : The window size is set to 0 packets (any value is fine for ABT).
  - `-p ../<UBIT_NAME>/abt` : The path to the executable to run for the experiment.
  - `-o e1_l1_abt.csv` : The name of the file to write the experiment results to.
- Just supply the `-h` parameter to see a listing of all available parameters.

Experiments & Report

- Once you have finished running the experiments, you will have to graph and analyze your results.
- The required graphs and required explanations are listed in the project specification document.
- NOTE: Running all of the experiments on the dedicated hosts will take quite some time. The amount of time it takes depends on how efficiently you implemented packet retransmissions as well as how many users are currently using the dedicated servers. You should budget at least 24 hours for all of the experiments to finish running.

Submitting

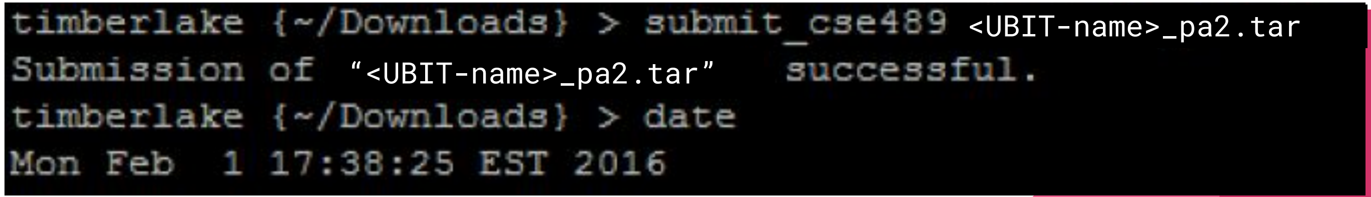
- Before submitting, you must first package your code into a tarball (.tar file).
- To do this, use the script included with the template:
    - `./assignment2_package.sh`
- Do NOT package manually
- This ONLY packages; **Does NOT SUBMIT**

- WARNING: In your submission, you MUST include the academic integrity statement in your report in order to receive a grade for the project.

Submitting

- After packaging your code, you must submit it using the CSE 489/589 submission scripts.
- Submitting your assignment follows the same procedure as labs and homeworks.
  - Undergraduates: submit_cse489 <ubit_pa2.tar>
  - Graduates: submit_cse589 <ubit_pa2.tar>

- If you’re working with another student, only one group member needs to submit
  - But make sure both UBIT names are located in the `abt.c/cpp` file.



```
timberlake {~/Downloads} > submit_cse489 <UBIT-name>_pa2.tar
Submission of "<UBIT-name>_pa2.tar"   successful.
timberlake {~/Downloads} > date
Mon Feb  1 17:38:25 EST 2016
```

Resources

- Reliable Data Protocols:
    - Textbook (7th Edition): ABT (Pages 216-217) & GBN (Page 223) & SR (Page 228)
    - Lecture Slides: “Lecture13_Transport2” & “Lecture14_Transport3”
- Programming Assignment 2:
    - Description: https://goo.gl/KzTh0J
    - Template & Packaging and Submission: https://goo.gl/G4cPfH
    - Grading Rubric: https://goo.gl/s74dAe

Next Week

- Wireshark Lab 4

Any Questions?

Acknowledgements

- Some of the slides were adapted from material made by Jim Kurose, Keith Ross, and the previous authors of the CSE 489 recitation slides.