Announcements

- Lab 1 is being graded. We will try to get scores/feedback out to you soon.

- Lab 2 is **due** Oct 12; Oct 14 (late).

- Writing assignment 2 is **due** Oct 9.
Outline

- Lab 1 vs Lab 2
- Lab 2 strategies
- Lab 2 gotchas
- Lecture review
- Questions
Lab 1 vs Lab2: Differences

- Multiple connections
- Window size $> 1$
Multiple Connections

Lab 1

Sender

Receiver
Multiple Connections

Lab 1

Sender

Receiver

Lab 2

Server

Client 1

Client 2

rel_t_1

rel_t_2

Stream 1

Stream 2
Window Sizes

Lab 1: Window size 1

Sender

data

Receiver
Window Sizes

Lab 1: Window size 1

Lab 2: Window size $n$
Outline

- Lab 1 vs Lab 2
- Lab 2 strategies
- Lab 2 gotchas
- Lecture review
- Questions
Before you begin: backup Lab 1 code

```
cp -r reliable reliable.bak.0
```
Before you begin: backup Lab 1 code

cp -r reliable reliable.bak.0

git commit -a -m "Lab 1 submission"

...
Decomposition

- Break problem into pieces
- Build each piece separately
- Test each piece individually (as much as possible)
Lab 1 vs Lab2: Differences

- Multiple connections
- Window size > 1
Lab 1 vs Lab2: Differences

- Multiple connections
- Window size > 1
Multiple Connections

Lab 1

Sender

Receiver

Lab 2

Server

Client 1

Client 2

Stream 1

Stream 2

rel_t_1

rel_t_2
Code differences

rel_create:
- Lab 1: SYSTEM calls
- Lab 2: YOU call

rel_recvpkt:
- Lab 1: SYSTEM calls
- Lab 2: YOU call

rel_demux:
- Lab 1: NO ONE calls
- Lab 2: SYSTEM calls
Lab 2: rel_demux

pkt → rel_demux (pkt, pkt_size, sockaddr, ...)

 pkt

Lab 2: rel_demux

pkt → rel_demux (pkt, pkt_size, sockaddr, ...)
data → rel_demux (pkt, pkt_size, sockaddr, ...)
ack → rel_demux (pkt, pkt_size, sockaddr, ...)
seqno=1 → rel_demux (pkt, pkt_size, sockaddr, ...)
eof → rel_demux (pkt, pkt_size, sockaddr, ...)
cksum: garbage → rel_demux (pkt, pkt_size, sockaddr, ...)
What should you do when you receive an ack packet in `rel_demux`?
What should you do when you receive an ack packet in rel_demux?

- Walk rel_list global variable
- For each rel_t in rel_list, check if sockaddr_storage matches.
What should you do when you receive an ack packet in rel_demux?

- Walk rel_list global variable
- For each rel_t in rel_list, check if sockaddr_storage matches.
What should you do when you receive an ack packet in `rel_demux`?

- Walk `rel_list` global variable
- For each `rel_t` in `rel_list`, check if `sockaddr_storage` matches.

How?
`addreq` function provided in `rlib.h`
rel_demux: ack pkt

What should you do when you receive an ack packet in rel_demux?

- Walk rel_list global variable
- For each rel_t in rel_list, check if sockaddr_storage matches.
- If matches, call rel_recvpkt for matching rel_t and packet.
rel_demux question

- Checksums for Lab 1 were in rel_recvpkt. Does this work in Lab 2?
Checkpoint question

- Checksums for Lab 1 were in rel_recvpkt. Does this work in Lab 2?

Hint: Are there any cases where you would perform an action based on a packet before rel_recvpkt?
Checksums for Lab 1 were in rel_recvpkt. Does this work in Lab 2?

*Hint: Are there any cases where you would perform an action based on a packet before rel_recvpkt?*

rel_create must be called before rel_recvpkt

Perform a checksum in rel_demux.
Window Sizes: Decomposition
Window Sizes: Decomposition I

Fundamental questions:

- *When should I send a data packet?*
- *What should I do when I receive a packet that I cannot instantly output?*
Window Sizes: Decomposition I

Fundamental questions:

- *When should I send a data packet?*
- *What should I do when I receive a packet that I cannot instantly output?*

Question: Can a sender with window 10 communicate with a receiver with window 1?
Window Sizes: Decomposition I

Fundamental questions:

- *When should I send a data packet?*
- *What should I do when I receive a packet that I cannot instantly output?*

Question: Can a sender with window 10 communicate with a receiver with window 1?

Yes  Decompose: Build one before the other.
Window Sizes: Decomposition 2

- Student's window implementation:

```c
struct rel_t {
    ...
    packet_t[] send_window;
    int []    send_window_map;
    int      send_window_map_index;
    int      send_window_size;
    ...
};
```

**Warning:** Example code doesn't have correct logic or syntax. It's just to demonstrate a point.
Window Sizes: Decomposition 2

rel_recvpkt(rel_t* r, packet_t* pkt, ...) 
{
    ...
    onAck:
      // check if acking a packet
      if (send_window_map[pkt->ackno % send_window_size])
      {
        // try to send if have any data available
        if (send_window_map[send_window_map_index])
          free(send_window[send_window_map_index]);

        (send_window_map_index) %= send_window_size;
      }
    ...
}

rel_timer(rel_t* r, packet_t* pkt, ...) 
{
    onExpired:
      if (send_window_map[send_window_map_index])
        conn_sendpkt(send_window[send_window_map_index]);
      ...
}

Warning: Example code doesn't have correct logic or syntax. It's just to demonstrate a point.
Window Sizes: Decomposition 2

```c
rel_recvpkt(rel_t* r, packet_t* pkt, ...) {
    ...
    onAck:
        // check if acking a packet
        if (send_window_map[pkt->ackno % send_window_size]) {
            // try to send if have any data available
            if (send_window_map[send_window_map_index])
                free(send_window[send_window_map_index]);
            (send_window_map_index) += send_window_size;
        }
    ...
}
rel_timer(rel_t* r, packet_t* pkt, ...) {
    onExpired:
        if (send_window_map[send_window_map_index])
            conn_sendpkt(send_window[++send_window_map_index]);
    ...
}
```

**Warning:** Example code doesn't have correct logic or syntax. It's just to demonstrate a point.

Putting all this logic directly into your code is going to make Lab 2 very difficult to debug.
Sending window functionality

List required functionality of sending window:

- Check if have room in sending window
- Check if have any partial packets in flight
- Add packet to window
- Etc.
Sending window functionality

List required functionality of sending window:

- Check if have room in sending window
- Check if have any partial packets in flight
- Add packet to window
- Etc.

All of these can/should be split into separate functions.
Sending window functionality

List required functionality of sending window:

– Check if have room in sending window
– Check if have any partial packets in flight
– Add packet to window
– Etc.

Examples:

//returns 0 if no room, 1 otherwise
int sendingWindowHasRoom(rel_t* r) ...

//returns 0 if partials in flight, 1 otherwise
int partialPktsInFlight(rel_t* r) ...

//returns 0 if successfully added, 1 otherwise;
//also sends added packet
int addPktToSend(rel_t* r, packet_t* pkt) ...
Window decomposition

- Before writing code:
  - Define interfaces for both sending window and receiving window
- Write code
- Test code
  - Test individually
  - Use tester
Outline

- Lab 1 vs Lab 2
- Lab 2 strategies
- Lab 2 gotchas
- Lecture review
- Questions
Setup

- See post 178 from last year.
- Should be able to send from server and client.
Nagle-like algorithm
Nagle-like algorithm

- For Lab 2, can only send a packet without a full payload if do not already have a packet without a full payload in flight.
Nagle-like algorithm

• For Lab 2, can only send a packet without a full payload if do not already have a packet without a full payload in flight.

• Before writing code, think about how this will affect the structure of your program.
Outline

- Lab 1 vs Lab 2
- Lab 2 strategies
- Lab 2 gotchas
- Lecture review
- Questions
Fragmentation

Maximum packet size: 3000B

Maximum packet size: 1500B
Fragmentation

Maximum packet size: 3000B

Pkt: 2000B

Maximum packet size: 1500B
Fragmentation

Maximum packet size: 3000B

Packet: 2000B
Packet: 520B
Packet: 1500B

Maximum packet size: 1500B
Fragmentation

Maximum packet size: 3000B

Pkt: 2000B

Pkt: 520B

Pkt: 1500B

Why 520?

Maxmum packet size: 1500B
What fields are going to be different for the fragmented packets vs the original?
**IPv4 packet format**

<table>
<thead>
<tr>
<th>Vers</th>
<th>hdr len</th>
<th>TOS</th>
<th>Total Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identification</td>
<td>Identification</td>
<td>Identification</td>
<td>Identification</td>
</tr>
<tr>
<td>TTL</td>
<td>Protocol</td>
<td>hdr checksum</td>
<td>hdr checksum</td>
</tr>
<tr>
<td>Source IP address</td>
<td>Destination IP address</td>
<td>Source IP address</td>
<td>Destination IP address</td>
</tr>
<tr>
<td>Options</td>
<td>Padding</td>
<td>Options</td>
<td>Padding</td>
</tr>
<tr>
<td>Data</td>
<td>Data</td>
<td>Data</td>
<td>Data</td>
</tr>
</tbody>
</table>
Fragmentation: Fill in the fields

Pkt: 1500B
Total Length: ________
MF: ________
Offset: ________

Pkt: 520B
Total Length: ________
MF: ________
Offset: ________
Fragmentation: Fill in the fields

**Pkt: 1500B**
- Total Length: __________
- MF: __________
- Offset: __________

**Pkt: 520B**
- Total Length: __________
- MF: __________
- Offset: __________

Note units: are 8B
Fragmentation: Fill in the fields

Pkt: 1500B
Total Length: 1500
MF: 1
Offset: 0

Pkt: 520B
Total Length: 520
MF: 0
Offset: 185 = (1480/8)
Receiver's view of fragmentation

How does the receiver know that two packets are fragments of the same larger packet?
### IPv4 packet format

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>vers</td>
<td>hdr len</td>
<td>TOS</td>
<td>Total Length</td>
</tr>
<tr>
<td>Identification</td>
<td></td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>TTL</td>
<td>Protocol</td>
<td>hdr checksum</td>
<td></td>
</tr>
<tr>
<td>Source IP address</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Destination IP address</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Options</td>
<td>Padding</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Receiver's view of fragmentation

How does the receiver know that two packets are fragments of the same larger packet?

*Identifier* field.
Go-back-n vs selective repeat

As a **sender**, what should I do when a packet I send times out?
Go-back-n vs selective repeat

As a **sender**, what should I do when a packet I send times out?

(In particular, if I have a send window of 5 packets, and the first has timed out, which packets do I re-transmit?)
As a **sender**, what should I do when a packet I send times out?

(In particular, if I have a send window of 5 packets, and the first has timed out, which packets do I re-transmit?)

How does GBN answer this question?
Go-back-n vs selective repeat

As a **sender**, what should I do when a packet I send times out?

(In particular, if I have a send window of 5 packets, and the first has timed out, which packets do I re-transmit?)

How does GBN answer this question?

-->Retransmit all packets in send window.
Go-back-n vs selective repeat

As a **sender**, what should I do when a packet I send times out?

(In particular, if I have a send window of 5 packets, and the first has timed out, which packets do I re-transmit?)

How does GBN answer this question?

--> Retransmit all packets in send window.

How does selective repeat answer this question?
Go-back-n vs selective repeat

As a **sender**, what should I do when a packet I send times out?

(In particular, if I have a send window of 5 packets, and the first has timed out, which packets do I re-transmit?)

How does GBN answer this question?

-->Retransmit all packets in send window.

How does selective repeat answer this question?

-->Retransmit only the packet that timed out.
Go-back-n vs selective repeat

As a **sender**, what should I do when a packet I send times out?

(In particular, if I have a send window of 5 packets, and the first has timed out, which packets do I re-transmit?)

How does GBN answer this question?
--> Retransmit all packets in send window.

How does selective repeat answer this question?
--> Retransmit only the packet that timed out.

When should I use GBN and when should I use selective repeat?
Go-back-n vs selective repeat

As a **sender**, what should I do when a packet I send times out?

(In particular, if I have a send window of 5 packets, and the first has timed out, which packets do I re-transmit?)

How does GBN answer this question?
--->Retransmit all packets in send window.

How does selective repeat answer this question?
--->Retransmit only the packet that timed out.

When should I use GBN and when should I use selective repeat?

*Hint: consider how errors can occur on the transmission channel.*
Go-back-n vs selective repeat

As a **sender**, what should I do when a packet I send times out?

(In particular, if I have a send window of 5 packets, and the first has timed out, which packets do I re-transmit?)

How does GBN answer this question?
---
> Retransmit all packets in send window.

How does selective repeat answer this question?
---
> Retransmit only the packet that timed out.

When should I use GBN and when should I use selective repeat?
If few, independent errors on channel, consider selective repeat.
If highly-correlated bursts of errors, consider GBN.
End-to-end/strong end-to-end
End-to-end/strong end-to-end

“The network’s job is to transmit datagrams as efficiently and flexibly as possible. Everything else should be done at the fringes. . .”

-RFC 1958
End-to-end/strong end-to-end

“The network’s job is to transmit datagrams as efficiently and flexibly as possible. Everything else should be done at the fringes. . .”

-RFC 1958

This is a principle. It is not a rule. As you go through the course, pay attention to different protocols and technologies. Which of them follow the principle? When they do not, why do they not?
End-to-end/strong end-to-end

“The network’s job is to transmit datagrams as efficiently and flexibly as possible. Everything else should be done at the fringes…”

-RFC 1958

This is a principle. It is not a rule. As you go through the course, pay attention to different protocols and technologies. Which of them follow the principle? When they do not, why do they not?

- Fragmentation?
- Forged RST packets?
- NATs?
- Network coding?
Outline

- Lab 1 vs Lab 2
- Lab 2 strategies
- Lab 2 gotchas
- Lecture review
- Questions
Questions?