## Programming Assignment 2 50 pts.

# Sending Photos to a Photo Server

**Due: Thursday, November 17, 2005 1 p.m.**

**Introduction**

The goal of this assignment is to send a set of digital photographs from a single client to a photograph gallery server. The client and server run on separate CCC Linux computers and communicate at the data link layer by *sending* and *receiving* frames. Both the client and server emulate three OSI layers (application/network, data link, and physical layer).

This assignment exposes the student to the concept of network protocol layers by implementing the PAR (Positive Acknowledgement with Retransmission) data link protocol on top of an emulated physical layer ***{real TCP does the actual transmissions for your physical layer}***. Assignment 2 is a major step towards a complete concurrent server implementation in assignment 4.

## *The Client*

The Client should be written to be run on any arbitrary CCC Linux machine. The command line for initiating the client is:

***client servermachine***

where

***servermachine*** indicates the logical name for the server machine (e.g., ccc4.wpi.edu).

and

***client.log*** indicates the file which records significant client events.

The Client communicates with the Server assuming knowledge of a unique "well-known" port for each photo server.

## The client application/network layer

The ***client application layer's*** responsibility is to read five digital photos from files ***photo1.jpg*** to ***photo5.jpg*** and send them one at a time to the photo server. The ***client application layer*** indicates to the ***client network layer*** when it has completely read in a photo by setting the end-of-photo indicator. **{Note, for this assignment the abstraction of separating these two layers is not necessary}.**

Initially, the ***client network layer*** calls the physical layer to establish a connection with the ***server network layer***. Once a connection has been established, the client network layer begins receiving 256 byte "chunks" of photos and depositing each 256 byte chunk into a packet payload. Additionally, the packet payload contains

one byte as an end-of-photo indicator for the application layer. The ***client network layer*** sends the packet to the data link layer and waits for an **ACK** packet from the ***photo server network layer***.

When the last photo has been sent and ACK'ed, the ***client network layer*** calls ***the client physical layer*** to close the connection to the server and terminate the Client.

## The client data link layer

The responsibilities of the data link layer involve error detection and the PAR protocol with a timeout mechanism that causes a frame retransmission when frames are not promptly acknowledged.

**Frame Format**

Information at the data link layer is transmitted between the client and the server in frames. All frames need a frame-type byte to distinguish data and ACK frames. All data frames must have two bytes for the sequence number, two bytes for error-detection, and one end-of-packet byte. The client process sends data frames that contain from 1 to 100 bytes of payload (encapsulated data from the network layer packet). ACK frames consist of **zero** bytes of payload, a two-byte sequence number, and a two-byte error detection field.

The ***client data link layer*** receives packets from the ***client network layer***, converts packets into frames and sends frames to the ***client physical layer***. Upon receiving each packet from the ***client network layer***, the ***client data link layer*** splits the packet into frame payloads. The data link layer builds each frame as follows:

- put the payload in the frame.
- deposit the proper contents into the end-of-packet byte to indicate if this is the last frame of a packet.
- compute the value of the error-detection bytes and put them into the frame.
- start a frame timer.
- send the frame to the ***client physical layer***.

The ***client data link layer*** then waits to *receive* a frame. If the received frame is a data frame, then its payload is a network layer ACK packet. The ***client data link layer*** then sends the valid ACK packet up to the ***client network layer***, and then waits to receive a new packet from the ***client network layer***. If the received frame is an ACK frame successfully received before the timer expires, the client sends the next frame of the packet. When the last frame of a packet has been successfully ACK'ed, the ***client data link layer*** waits to receive a data frame. If an ACK frame is received *in error*, this event is recorded in the log **and the client data link layer continues as if the ACK was never received.** If the timer expires, the ***client data link layer*** retransmits the frame.

## The client physical layer

The ***client physical layer*** sends the frame received from the ***client data link layer*** as an actual TCP message to the ***server physical layer***. The ***client physical layer*** receives frames as actual TCP message from the ***server physical layer***. This triggers a received frame event for the ***client data link layer.***

The Client records significant events in a log file ***client.log***. Significant events include: packet sent, frame sent, frame resent, ACK frame received successfully, ACK packet received successfully, ACK frame received in error, and timer expires. For logging purposes identify the packet and the frame within a packet by number for each event. Begin counting packets and frames at 1 (e.g. "frame 2 of packet 218 was retransmitted").

## *The Photo Server*

The Server also should be written to run on an arbitrary CCC Linux machine. The Server emulates the same three layers as the client process (application/network, data link and physical layer). The Server is always started first.

The command line to start the server is simply:

***server***

where

***photonew1.jpg*** to ***photonew5.jpg*** indicates the name of the files in the photo gallery written out by the server.

and

***server.log*** indicates the file which records significant server events.

### The server application / network layer

The ***server application layer's*** responsibility is take 256 byte photo chunks out of network packets to reconstruct the five photos and write them out to the files in the photo gallery. The ***server application layer*** interrogates the end-of-photo indicator byte in the packet to know when the current packet is the last packet for a photo so the specific photo file can be closed.

After each packet has been processed by the ***server application layer,*** the ***server network layer*** creates an ACK packet and sends it to the ***server data link layer.***

### The server data link layer

The Server begins by waiting for the establishment of a TCP connection from the client. Once the connection is established, the ***server data link layer*** is initiated. The ***server data link layer*** cycles between *receiving* a frame from the ***server physical layer***, assembling a packet and possibly sending the packet up to the ***server network layer***, *receiving* an ACK packet from the ***server network layer*** and *sending* it as a data frame, and *sending* an ACK frame back to the Client via the ***server physical layer***. The ***server data link layer*** sends ACK frames consisting of two bytes of sequence number and the two error-detection bytes.

There is no need for a timer at the server. **Note - the setting of the end-of-packet byte indicates to the *server data link layer* that the current received frame is the last frame of a packet.** When the client closes the connection to the server, the server terminates.

The **data link layer** has to check for transmission errors using the **error-detection** bytes. If the received data frame is in error, this event is recorded and the receiving process waits to receive another frame.

The ***server data link layer*** checks received frames for duplicates and reassembles frames into packets and sends one packet at a time to the ***server network layer***. Note – the ***server data link layer*** needs to send an ACK frame when a duplicate frame is detected due to possibly damaged ACK's. The Server records significant events including frame received, frame received in error, duplicate frame received, ACK frame sent, ACK packet sent and packet sent to the network layer in *server.log*.

**Frame Error Simulation**

Since real TCP guarantees no errors at the emulated physical layer, your program must inject artificial transmission errors into your physical layer.

Force a **client transmission error** in every $8^{th}$ frame sent by flipping any single bit in the error-detection bytes prior to transmission of the frame. Force a **server transmission error** every $6^{th}$ ACK frame sent by using the same flipping mechanism. (i.e., frames 8, 16, 24, … sent by the client will be perceived as in error by the server and ACK frames 6, 12, 18, … sent by the server will be perceived as error by the client.) When the client times out due to either type of transmission error, it resends the same frame with the correct error-detection byte.

Assume for this assignment that all data frames sent by the ***server data link layer*** are transmitted "error free" Therefore, the ***client data link layer*** does **NOT** need to ACK the data frames sent by the ***server data link layer***.

**Assignment Hints**

- **[DEBUG]** Build and debug your programs in stages. Begin by getting the client and server working without processing errors and without a timer. Then add the error generating functions and the timer mechanism on the client. Initially, the Client and Server can exist on the same machine if this simplifies debugging. However, at least one member of the programming team needs to focus on making sure the final project permits the Client and Server to exist and be tested by the TA on any arbitrary CCC Linux machine.

- **[Error-Detection]** While **CRC** at the bit level will be discussed in class, it is recommended that you use a two-byte **XOR folding** algorithm of all the frame bytes to create your error-detection bytes. For the ACK frame, the error-detection bytes simply become a copy of the two-byte sequence number.

- The **correct** way to handle a timer and an incoming TCP message **requires** using a timer and the ***select system call***. You will lose points if you use polling to do this assignment, and you will be unable to do assignment 4 without knowledge of the select function.

- **[Performance Timing]** You **must** measure the total execution time of the complete emulated transfer of all the photos. Be sure to print this result out in readable form in the file *client.log*.

- **[Timer] The PAR protocol can fail if there is a *premature timeout.* Set the timeout period large enough to insure no premature timeouts.**

- **port numbers:** You can "hardwire in " the well-know port number of the server for this assignment.

- The **actual content of the photos** written by the Photo Server should **exactly** match the photos read by the client.

- **[Documentation]** Several small design decisions are deliberately vague in this assignment. The project team MUST explain all these design decisions both as documentation in the code and via a separate README file. **Remember: This is a team project and all routines must specify only a SINGLE primary author for each routine as part of the documentation!! You CANNOT simply attribute routines to all team members!!**

Do not wait for the official test data to work on this assignment. Use your own digital photo to test out the client prior to the TA releasing the official test photos.

**What to turn in for Assignment 2**

The TA will make an official test file available a couple of days before the due date. Turn in your assignment using the *turnin* program. Turn in the two source programs *client.c* and *server.c*, a README file and a make file.