QUIC Code:

quic\_ack\_notifier:

Called when a packet get ACKed

quic\_connection:

A QUIC's connection. It has many callback functions, when a packet is processed the functions will be called and the data inside will be processed.

quic\_data\_stream:

An implementation of ReliableQuicStream

quic\_dispatcher:

The dispatcher will manage different connections established by a server. Once a connection is established, packets will be processed by the connections.

quic\_fec\_group:

Processing FEC information.

quic\_framer:

It'll process the packets and extract different frames out of the packets and process them. It'll also form the different frames and add them to the packets.

quic\_http\_stream:

Managing http stream.

quic\_packet\_creator:

Packs the different frames into one packet.

quic\_received\_packet\_manager:

The packet received may be out of order. They need this manager to put everything together.

quic\_server:

The server code.

quic\_session:

Managing a session created.

quic\_spdy\_server\_stream:

It maintains spdy streams and does what spdy server does.

quic\_stream\_sequencer:

Receives and buffers data from a stream

quic\_unacked\_pacet\_map:

Maintains a list of unacked packets.

reliable\_quic\_stream:

Maintains a reliable quic stream

How does the server work:

The server starts from quic\_server\_bin, which is a binary wrapper of the quic server. It will create an instance of the class QuicServer and let it listen on the port and ip address designated.

quic\_server: Runs QuicServer::Listen, it first starts listening on the socket address. Then it'll allocate the recv and send buffer for the server. After this it'll initialize a dispatcher for this server. This dispatcher will process all the packets sent and recved by the server and manage the sessions.

The server does mainly simple things. It'll reading bytes from the port listened continuously and send the data recved to dispatcher.

quic\_dispatcher: The dispatcher has a member called framer. The framer will process specific frames in a packet. The framer has a visitor.

The visitor exists in many classes of QUIC code. That's because QUIC uses callback function. When a function calls another function, it doesn't do anything when it returns. Instead, the callee calls another callback function. So the callee needs to know who called this function, that's what visitor does.

When QuicDispatcher recved a packet, it first send it to its framer to parse the packet.

quic\_framer:

The dispatcher's framer's visitor is the dispatcher.

When framer first process a packet it'll call its visitor's OnPacket function. This time it'll do nothing.

Then it'll first parse the public header. It'll extract information from the public header to determine the quic version or other things in the public flag of QUIC.

Then it'll call its callee's OnUnauthenticatedPublicHeader function. What it does is that now the public header is not authenticated. The server needs to see if the packet belongs to the new session or existed session. If not existed, a new session should be created. The visitor will return a boolean to tell the framer whether it needs to process the packet further. Because this time it's the dispatcher's framer, this framer will only process the header and rest of the packet will be processed in the session so it will always return false and stops the framer from further proceeding the packet.

quic\_session: quic\_session has a quic\_connection member. Its visitor is type VisitorShim. Then this session's connection's ProcessUdpPacket method will be called.

quic\_connection: This file contains many important functions of QUIC protocol. It'll determine the behavior of the server/client when proceeding different packets/frames.

It also has a framer member. This framer's visitor is this connection, not the dispatcher mentioned above. When this connection process a UDP packet, it'll call the framer to parse the packet.

The framer will process the packet like the first time dispatcher told it to do so. The only difference is that when calling the visitor's OnUnauthenticatedPublicHeader, since the header has been authenticated, the connection will simply return true, so that the rest of the packet will also be processed by the framer.

quic\_framer: The framer will look at the header to determine whether it's a version negotiation packet or a public reset packet or a data packet. We'll assume the server has received a data packet now.

It'll call ProcessDataPacket. It'll process the packet's header first. This time it'll look at the other information in the header, the sequence number, private\_flag, entropy\_flag, entropy\_flag and so on. It'll call back to connection to validate the packet's header. Then it'll handle the packet's payload. If the packet is in a FEC group, it'll process it. Then it'll process every different frames in the packet.

In ProcessFrameData, it'll first read the frame type byte to determine what type of the frame is. If it's a special frame type (i.e. stream frame, ack frame, congestion feedback frame), it'll handle that in different functions. Otherwise, it'll be processed simply in this function.

Stream Frame:

Stream frames contain contents of streams. The framer will first parse the stream frame in ProcessStreamFrame function:

It'll first read the header of stream frame. It contains the information of StreamID, Offset, Data Length and Fin. Then it'll read bytes equals to data length, which is frame data. Then it'll wrap all of these information into class QuicStreamFrame.

Then it'll call it's visitor's OnStreamFrame to tell it's visitor the frame has been parsed. This time the visitor is connection.

quic\_connection: In OnStreamFrame it'll check whether the stream data frame is encrypted. If is it'll add the frame into a list, then return true to tell the framer to continue further processing.

Ack Frames:

Ack Frames contain the information of which packet is acked. As above, it'll first call ProcessAckFrame to parse the Ack Frame:

It'll check the version and call ProcessReceivedInfo to process it:

First it'll determine the three lengths from the frame type: largest observed length, missing sequence number length, and missing range length.

Then it'll record the delta time largest observed. Then it'll see if there are packet not acked. If yes, it'll update the missing packet list and see if there are packets revived. Then it'll call its visitor's OnAckFrame to inform its visitor that an ack frame has been processed.

quic\_connection: In OnAckFrame it'll check the frame and push the frame into a list. Then it'll tell the framer to continue processing.

Congestion Feedback Frames:

It contains the information needed to do congestion control. As above, it'll first call ProcessQuicCongestionFeedbackFrame:

It'll see the congestion feedback frame's type and do different things. Now all of packet will use type kTCP, which means it'll use TCP's congestion control scheme.

It'll call visitor's OnCongestionFeedbackFrame. This will also push the frame into a list.

When all frames are processed, we can see every frame's content is wrapped in different classes and each type of classes form a list. At last the OnPacketComplete of connection is called. This will resolve all the frames received.

quic\_connection: OnPacketComplete deal with frames received. First, it use received\_packet\_manager to form a in-order frame list. If there were stream frames received, the visitor's OnStreamFrames will be called. Let's recall that connection's visitor is class VisitorShim.

quic\_session: VisitorShim's OnStreamFrames is called, it will call session's OnStreamFrames first.

This function will go through all the stream frames received. It will get the stream\_id of the stream frames and get that stream. The stream is a ReliableQuicStream, but it is actually also QuicSpdyServerStream. The latter inherits the former one. The stream's OnStreamFrame is called.

reliable\_quic\_stream: OnStreamFrame will deal with flow control mainly. It'll call the sequencer's OnStreamFrames. The sequencer is a utility which can process the contents of a stream and make them into a sequence.

quic\_stream\_sequencer: OnStreamFrame will see if the frame has arrived in-order. If not it'll need to buffer the frames first before processing. When processing the data, it'll call stream's ProcessRawData function.

quic\_data\_stream: ProcessRawData will call ProcessData.

quic\_spdy\_server\_stream: ProcessData will call SPDY server's function to parse the request, and generate a SPDY response. That's what the application layer does.

So until now we can see how server works. It reads packets and parse them, and finally send them to application layer. Then it'll save the response, and process the ack frames and congestion feedback frames and so on, and frame them and packs them, and send them out to clients finally,