ECE 158B Project 1 - Protocol Distribution Analysis

Jiawei Lian (PDF/CDF Plotting)
Sanjit Joseph (Packet Parsing)
Praveen Swaminathan (Packet Capturing)
(Group contribution to complete this analysis doc)

Packet Capture Procedure

For this project, four packet captures needed to be conducted for the following protocols: HTTP, FTP, VoIP, and BitTorrent. All packet captures were for at least one minute. For all packet captures, after the initial capture was complete, the pcap was filtered to only my IP address, making sure that no other device traffic was showing up in the file. Below are the steps taken to capture the packets for the specific protocols:

HTTP: To demonstrate this protocol, I went to yahoo.com, and browsed the website for a minute. During this time, I clicked on videos, links, articles, images, and new sections. I also imitated common user behavior on a website by clicking and scrolling through the various media and articles.

FTP: To demonstrate this protocol, I used the FileZilla client. First, I set up a new site, connecting to the server test.rebex.net, to get access to some demo files. The server consisted of 16 demo files. Once the packet capture was started, I downloaded the files one by one over the course of a minute. NOTE: All demo files were extremely small in size (in the low KB range).

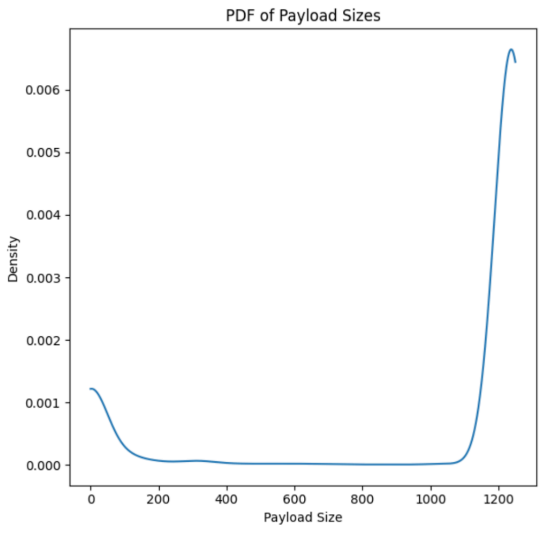
VoIP: To demonstrate this protocol, I used Zoom. To get my remote host, I called my family back home on Zoom. During the one minute, I made sure that my video (as well as the host's video) was on, talking was done by both sides, and periods of silence existed.

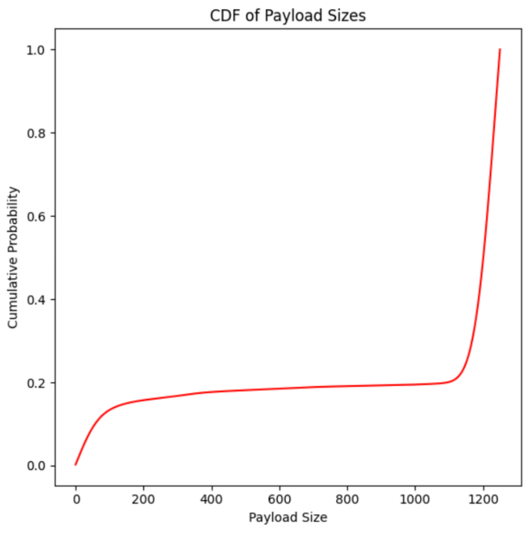
BitTorrent: To demonstrate this protocol, I first had to download a BitTorrent client, BitTorrent Web. Then, I downloaded a torrent of a video from online. After putting the torrent file into BitTorrentWeb, I ran my packet capture, through the duration of the file download. Since the video file was very large, it lasted the entire minute of the packet capture (and more after).

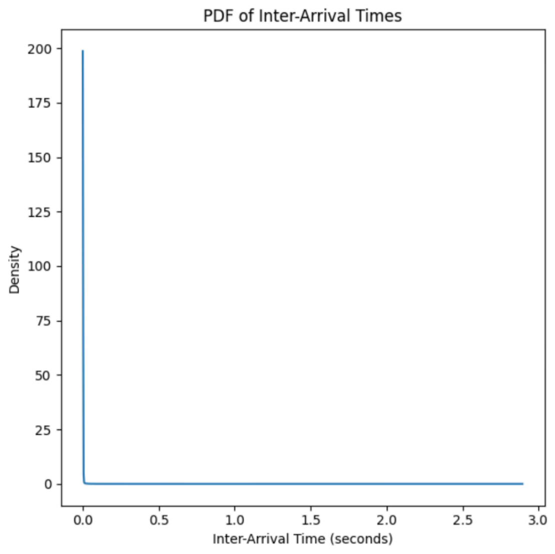
Packet Parsing Procedure

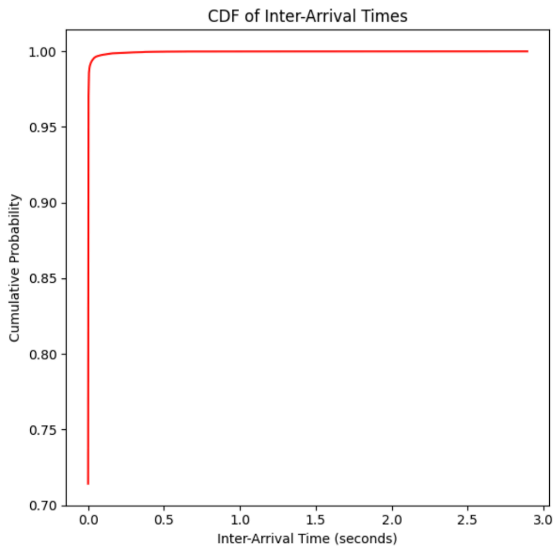
As shown in the code, packet parsing was accomplished through a script that imports dpkt. The option was given to the user to output to a text file or directly to the terminal. The packet data was accessed in much the same way that is shown in the dpkt examples, we accessed each frame along with its payload size and timestamp. The Decimal module was used for accurate time calculation, since we have to subtract the current timestamp from the previous one for every packet. We only had to skip the fragmented packets since they don't have a payload size, but all other packets were displayed–there was no need to do filtering in the script since the packet capture section already captured separate pcaps for HTTP vs FTP, etc. The output was checked against the WireShark display to make sure it was working properly.

HTTP





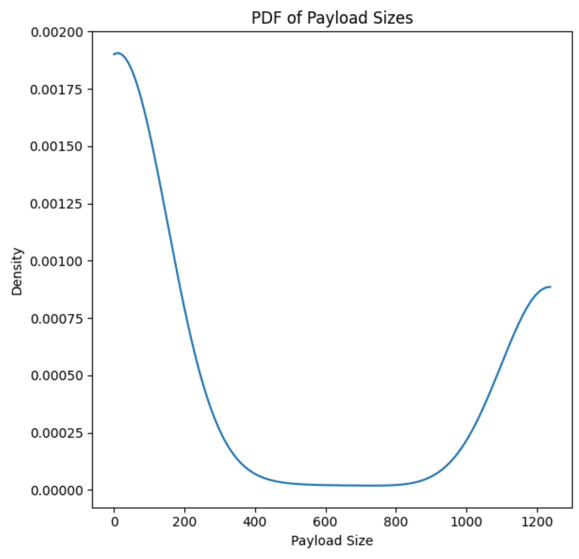


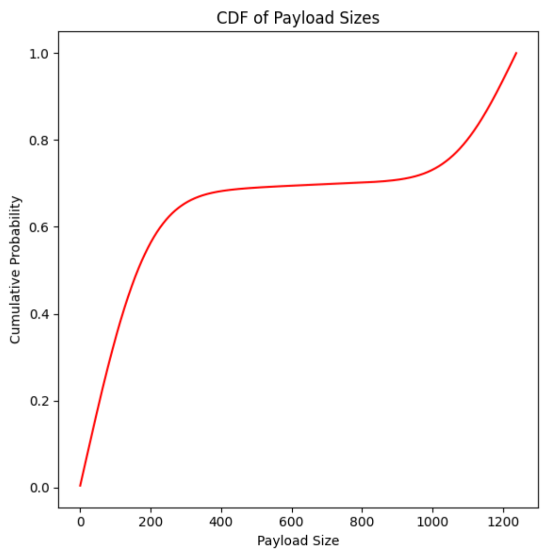


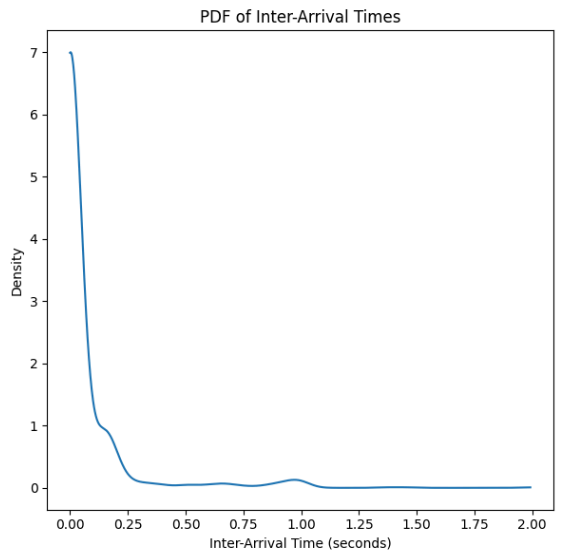
Explanation

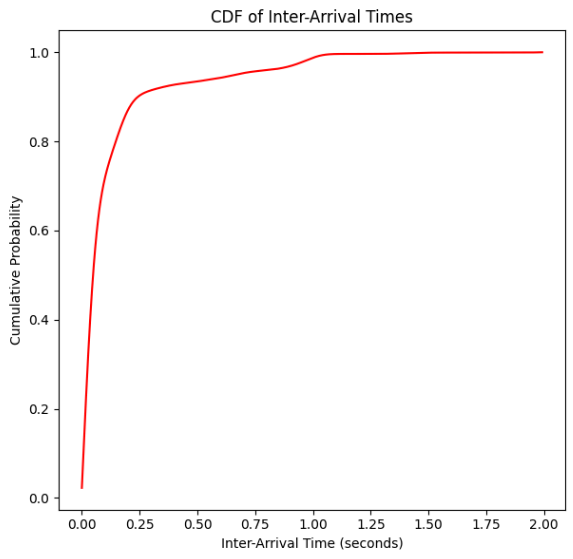
For the HTTP protocol, the PDF of the payload sizes is bimodal, with a low density peak at a very small payload size, and a high density peak, with a much larger payload size. The smaller peak represents any requests, or acknowledgement packets, which are small in size, and less common than the responses, which contain videos, HTML files, and images, all of which require packets of a higher payload. This observation is justified by the CDF function, in which the chance of finding a request packet is much lower (at 20% of packets) than a larger payload packet, which most likely contains media. As for the inter-arrival times, almost all of the packets arrive in a very short amount of time, nearing 0 seconds for transmission between consecutive packets. This makes sense as the response time when a user is on a website is extremely low, and oftentimes the user does not realize any delay when migrating through a website.

FTP









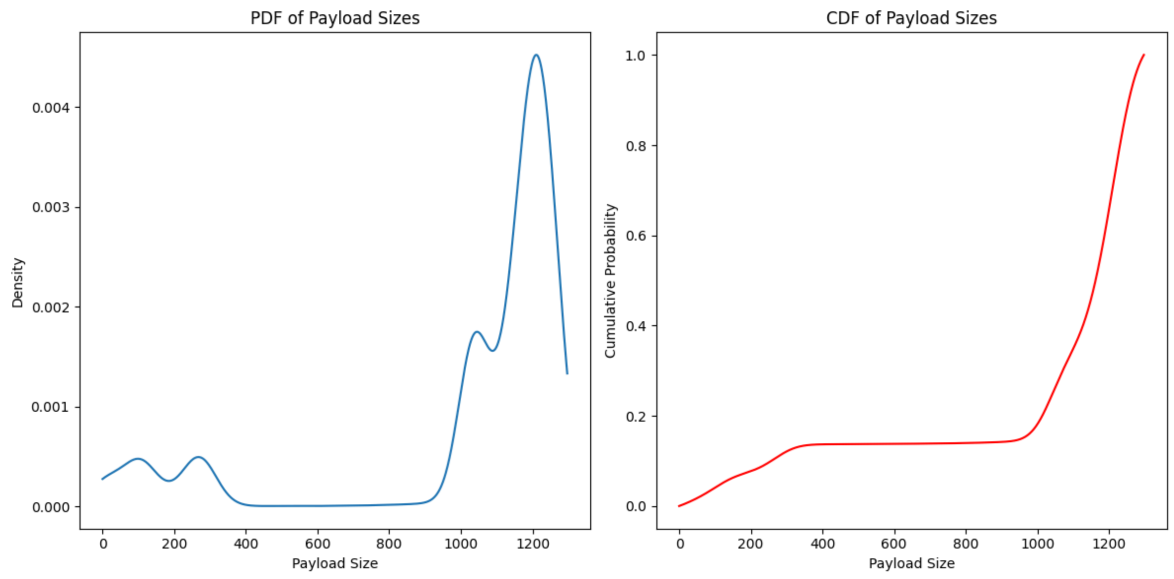
Explanation

In the FTP protocol, as studied before, there is a separation of TCP connections between control and data, and each has different specifications and operations. For example, the control connection needs to be low latency, and able to exchange frequent messages between the client and the server, while the data connection is at a high rate, but with intermittent request/response and will disconnect once a data file has been transmitted. This is important to understand when looking at these distributions. First, some context: the files downloaded through ftp are very small files. This explains the low density in the higher payload sizes, which contain the actual data. The high density in the lower payload sizes is explained by the numerous requests/responses that will be sent in the control connection each time a data connection is opened to download one of the files. In this case, since there are many files downloaded that are so small, the control messages outweigh the density of the data packets itself. This is proven in the CDF function as around 70% of the packets are less than 400-450 bytes. As for the inter-arrival times, in the PDF distribution while most of the times are close to 0, there are some outliers up until almost 1 second. In the CDF, 90% of the packets have an inter-arrival time of less than 0.25s, but 10% of packets are between 0.25 and 1 second. This may also be attributed to the intermittent behavior of the data connection every time a file is downloaded from the server. Since this happens around 16 times (16 files), there is bound to be some extra delay when the data TCP connection opens and closes.

Extra Observations:

(PDF of HTTP and FTP) HTTP is faster than FTP. This may be due to the fact that during the FTP session, only small files were downloaded (as compared to larger files which it is optimized for). This causes extra delay (due to data TCP connections) which is not necessary in this case, and thus makes it slower than HTTP.

VoIP

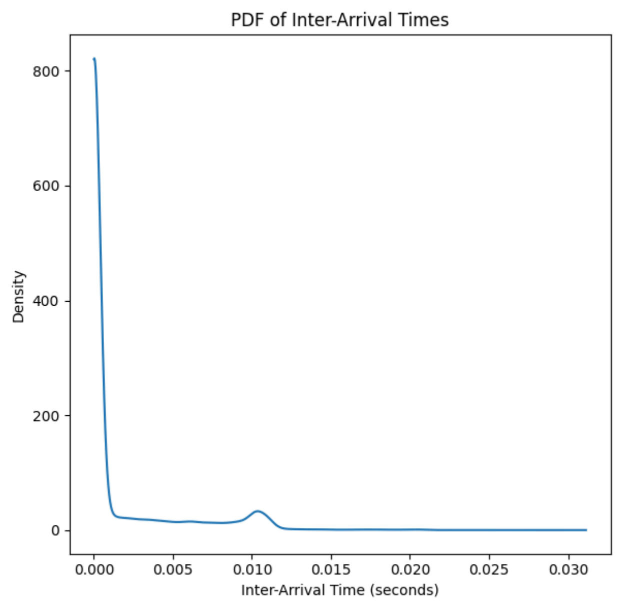


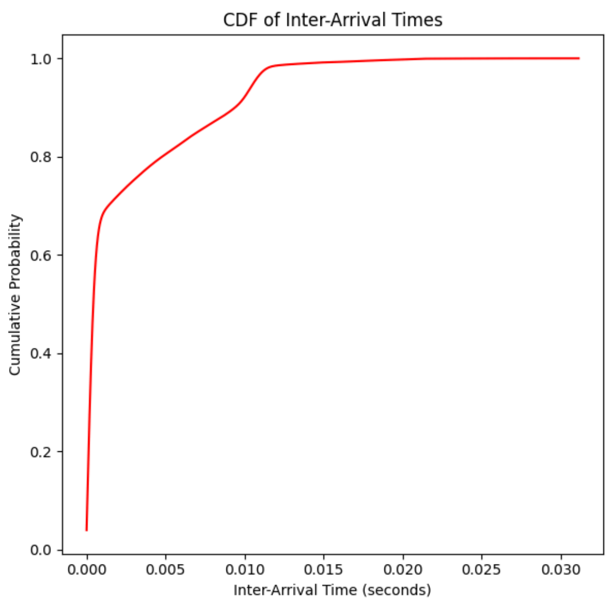
Observations

For the VoIP protocol, for the PDF figure, the chart shows multiple peaks in the density of payload sizes, with significant peaks around 1000 to 1200. There are smaller peaks at lower payload sizes. For the CDF figure, the cumulative probability increases steadily with payload size, with notable jumps corresponding to the peaks observed in the PDF. The curve starts slow, has a steady incline, and rises steeply as it approaches the higher payload sizes. From the CDF graph, around 15% of the packets are less than 1000 bytes in size.

Explanation

Larger packets (1000 to 1200) are likely used to transmit voice data and video data more efficiently and with more quality. This reduces the overhead and increases data transmission efficiency, which is crucial for maintaining the quality of voice communication. As for the smaller packets, from the packet capture itself, we figured out that these correspond to TCP and TLS packets. Most of these packets do not seem to be for the actual communication (as it would be inefficient in VoIP) but rather could be control messages, encoding or encryption schemes, since Zoom is a secure calling platform.





<u>**Observations**</u>

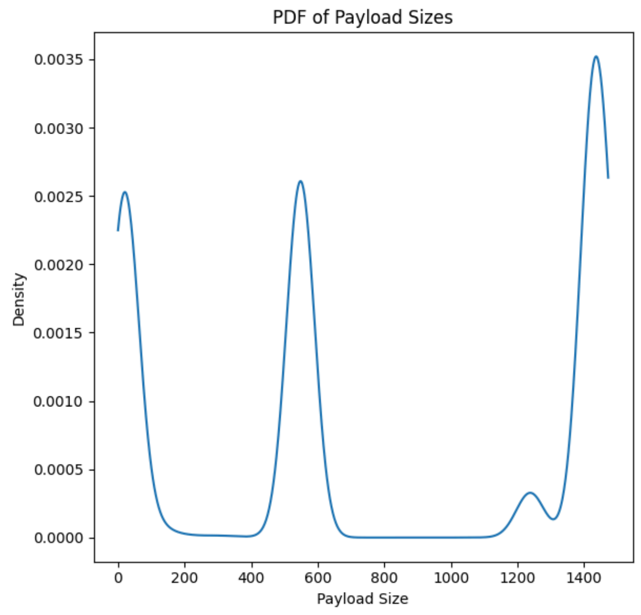
The PDF figure shows a very high peak at very short inter-arrival times, indicating that most packets arrive within a very short interval of each other. The density sharply decreases and then flattens out as the inter-arrival time increases. There is a very small peak at 0.01s.

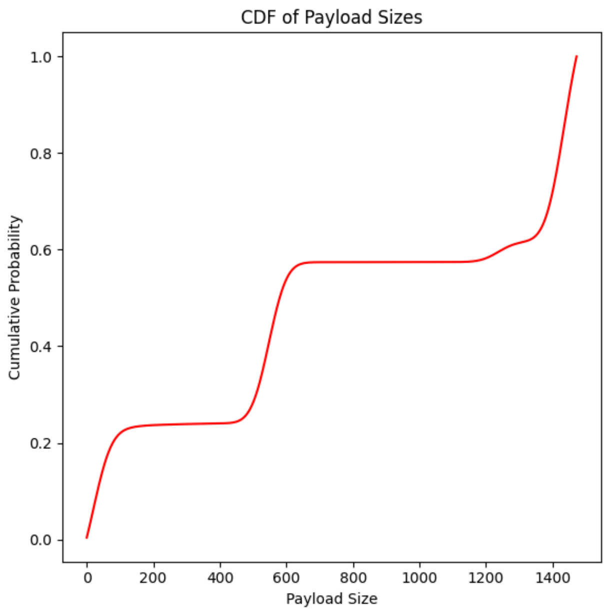
For the CDF figure, the cumulative probability increases rapidly at the beginning, reaching close to 1 within a very short inter-arrival time. This also indicates that most packets arrive within a short time interval. It is important to note that all packets are received within 0.01 seconds (through CDF graph).

<u>**Explanation**</u>

From both figures, we can conclude that most data packets arrived in a very short time period, which is expected for real-time voice and video applications. VoIP is designed to minimize delay, as through UDP transmission, it does not matter if one or two packets are dropped, so long as the video streaming continues. Furthermore, we predict the reason why the inter-arrival time has a small peak much higher than the majority of the distribution may be due to network jitter. From the user end in the Zoom call, this may manifest itself as a loss of frames for a small amount of time, or some audio cutting out during speech. However, in this case, since the density of the 0.01s peak is so low, this jitter was most likely not realized during the call.

BitTorrent





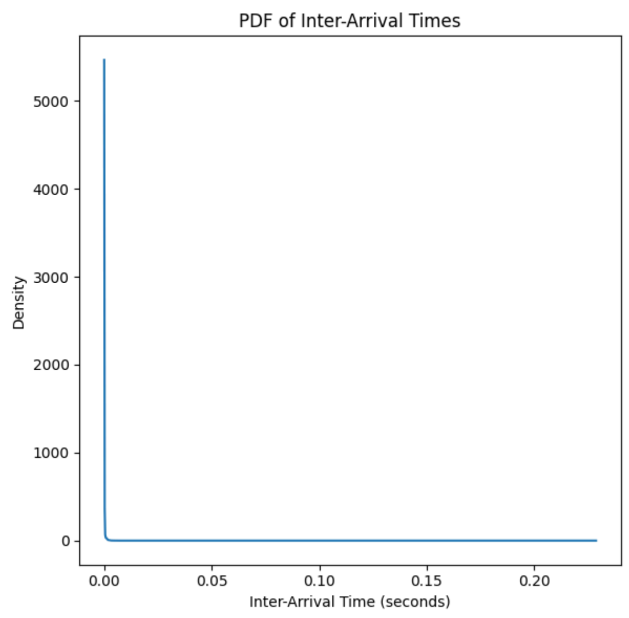
Observations

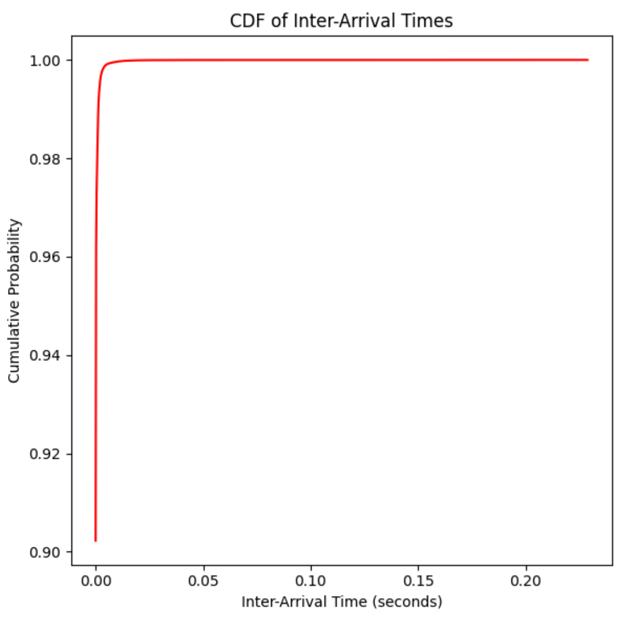
The PDF figure shows multiple significant peaks, with the highest density observed around 1300 and 1400. There are noticeable dips between these peaks.

For the CDF figure, the cumulative probability increases in steps, corresponding to the peaks observed in the PDF. Steep increases at around 600 and 1400 align with the high-density regions in the PDF. Around 20% of the packets are expected to be small payload sizes, 40% medium file chunk sizes, and 40% large packet chunk sizes.

Explanation

BitTorrent divides files into chunks for distribution. Therefore, the payload size may vary based on the file being shared and where the data is coming from at any given time. The observed peaks likely correspond to the common chunk sizes used in the protocol. For instance, 600 and 1400 bytes might represent optimal file chunk sizes for efficient transmission and reassembly. As for the smaller payload sizes, these may be the BitTorrent exchange messages, which are packets that are not large in size, especially compared to the data being downloaded.





<u>Observations</u>

The PDF figure shows an extremely high peak at short inter-arrival times, indicating that most packets arrive almost immediately after each other. The density rapidly decreases and flattens out as the inter-arrival time increases.

For the CDF figure, the cumulative probability increases quickly, reaching almost 1 within a very short inter-arrival time. This indicates that nearly all packets arrive within a very short time frame.

<u>Explanation</u>

From both figures, we can see that packets are transmitted rapidly. This is consistent with the protocol's behavior, which aims to maximize data transfer efficiency. Through the rarest chunk first and tit-for-tat (with peers sending at the highest rate) strategies, file transmission speeds are extremely quick, resulting in the inter-arrival time between packets at almost 0. This is reflected by the observation of the torrent download itself, where a 2GB file was downloaded in a little over a minute.

Conclusion

From previous observations and analysis, we can conclude that for the payload size:

- HTTP: Bimodal distribution, with very small density for HTTP requests, and large density for responses (media/content, etc)
- FTP: Bimodal distribution with distinct peaks at small (control data) and large (file data) payload sizes.
- VoIP: High, narrow peak at large payload sizes, indicating optimization for efficient data transmission. Small peak(s) for control messages and/or encryption schemes
- BitTorrent: Multiple peaks at standardized chunk sizes, reflecting the chunk-based nature of file distribution. Small peaks for BitTorrent exchange messages

And for the time-intervals:

- HTTP: The time interval between packets was reasonably short–we assumed this is because there shouldn't be too much latency when clicking on links and such.
- FTP: We can see the inter-arrival times are longer and much less consistent than HTTP. This is because the end goal is just efficient transfer of a large file, and less attention is paid to packet latency.
- VoIP: Inter-arrival times are extremely short compared to the other protocols, as we expected. They are not perfectly consistent, i.e. perfectly spaced, but of course this is what the buffer mechanism in VoIP is designed to solve (in the presence of jitter)
- BitTorrent: The inter-arrival times here are similar to HTTP, probably because they are both designed to be fast and efficient on the user side. Compared to HTTP we used a much larger file and got smaller packets, so it makes sense that the inter-arrival time is generally a bit lower as well (but not by much). Of course, downloading from multiple peers also contributes to this–we can receive lots of packets in parallel over multiple connections. As mentioned before, with the 2 schemes that BitTorrent uses to exchange and download information, we can get packet-arrival times similar to HTTP, even though we are downloading much larger files as compared with simply accessing and browising a website.