COMP3331/COMP9331 LAB 5 Report

written by Heng-Chuan Lin z5219960

EX1Q1: What is the maximum size of the congestion window that the TCP flow reaches in this case? What does the TCP flow do when the congestion window reaches this value? Why? What happens next?

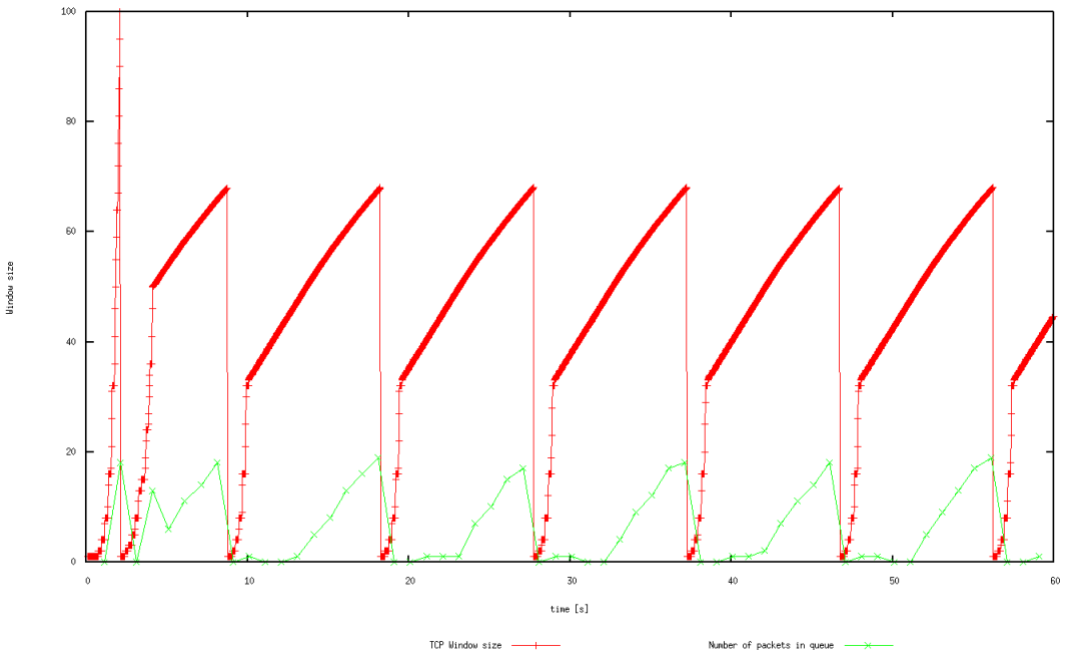
Ans:

The maximum size of the CWND would be 100 MSS. We can observe that receiver can only put around 20 MSS into queue, incoming packets would trigger packet loss. After first timeout or DupACK, the ssthresh would be set to the half value of CWND_max and CWND would be set to 1.

In the beginning, slow start phase would be initiated with an initial ssthresh and CWND = 1 MSS.

Due to the feature of slow start phase, the CWND grew up exponentially. Once the CWND >= initial ssthresh, the stage would move to addictive increase (I.e. congestion avoidance) for slowly increasing the CWND.

After buffer on receiver side overflowed (I.e. pkt loss), receiver then sent triple dupACK or sender the timeout occurred would trigger the phase shifted back to slow-start with CWND = 1, ssthresh = CWND/2.



EX1Q2: From the simulation script we used, we know that the payload of the packet is 500 Bytes. Keep in mind that the size of the IP and TCP headers is 20 Bytes, each. Neglect any other headers. What is the average throughput of TCP in this case? (both in number of packets per second and bps)

Ans:

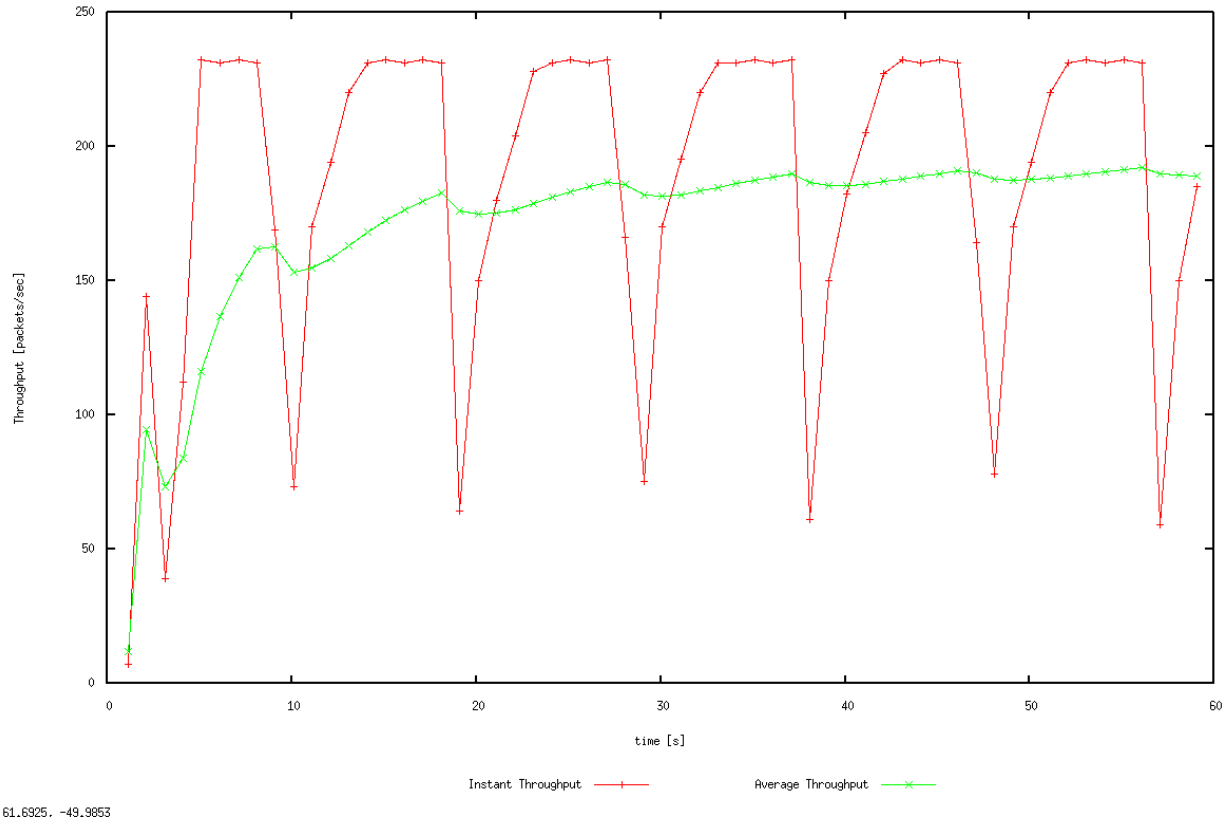
in format of packet/sec:

avg throughput is around 190 packets/sec (after 30 sec quite stable)

in format of bps/sec:

Total size of a single packet: 500 + 20(TCP header) + 20(IP header) = 540 Bytes

540 Bytes/packet * 8 bits/Byte * 190 packets/sec = 820800 bits/sec = 820.8 kbps

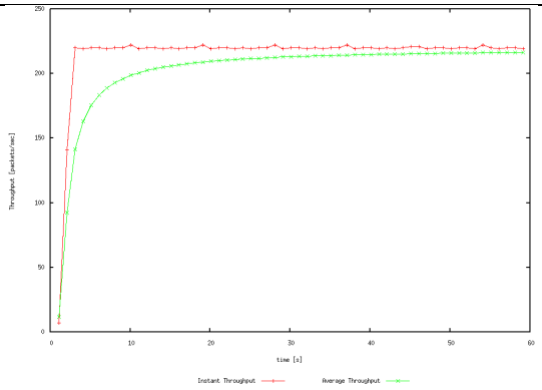


Question 3: Rerun the above script, each time with different values for the max congestion window size but the same RTT (i.e. 100ms). How does TCP respond to the variation of this parameter? Find the value of the maximum congestion window at which TCP stops oscillating (i.e., does not move up and down again) to reach a stable behaviour. What is the average throughput (in packets and bps) at this point? How does the actual average throughput compare to the link capacity (1Mbps)?

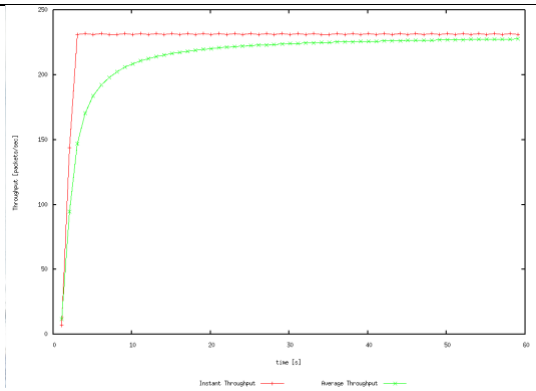
Ans:

The Max Window sizes without oscillating would be 50. According to the table below, 51 would trigger pkt loss, so 50 is the maximum value. The avg throughput would be 230 packets/sec

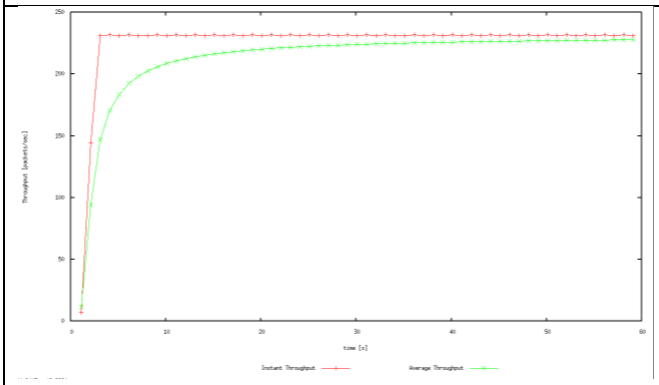
(500+20+20) * 230 * 8 = 993600 bits/sec = 0.9936 Mbps
Margin of error: |(0.9936 -1) / (1/100)| = 0.64%
According to value margin of error above, simulation is quite close to the reality.



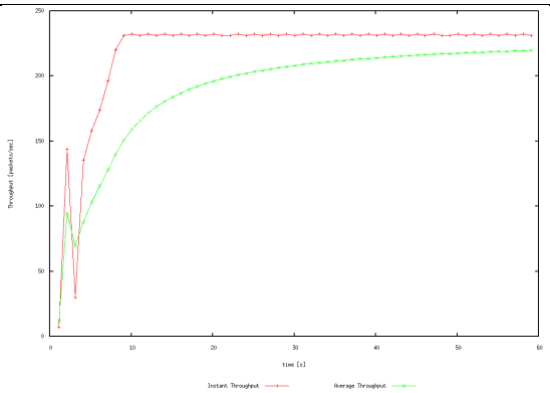
Max window sizes = 45



Max window sizes = 49



Max window sizes = 50



Max window sizes = 51

EX1Q4: Repeat the steps outlined in Question 1 and 2 (NOT Question 3) but for TCP Reno. Compare the graphs for the two implementations and explain the differences. (Hint: compare the number of times the congestion window goes back to zero in each case). How does the average throughput differ in both implementations?

Ans:

TCP-Reno would set the CWND to ½ previous CWND + 3 MSS and scale down the ssthresh to ½ previous CWND after received triple DupACK and further moved to Fast Recovery state. Then, ½ previous CWND would set to 1 and ssthresh would be half the CWND again due to timeout occurred. (Fast Recovery to Slow Start state). Besides we can observe that in Reno case, DupACK happened more frequently than timeout.Hence, compared to Tahoe, the CWND differed from: → timeline

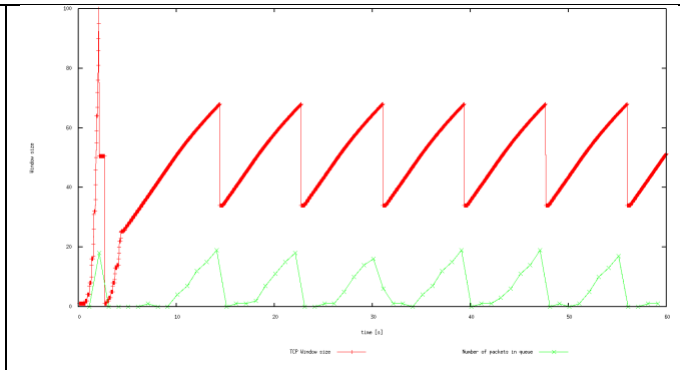
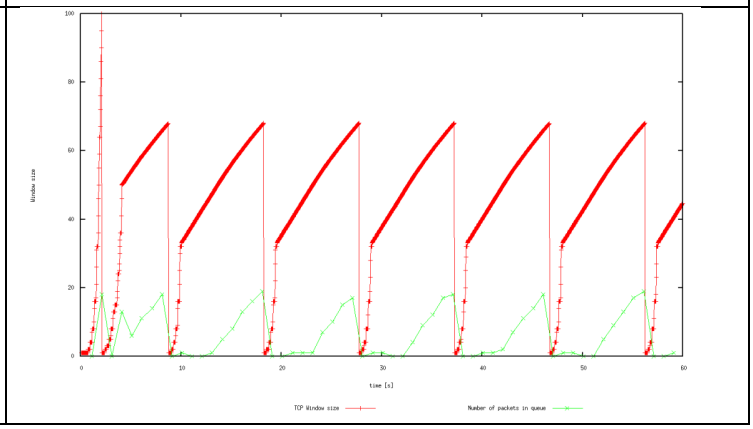
CWND: 100. >> 50+3 (3dupACK) >> 1 (timeout) >> slow start >> reach 26 >> additive increase

ssthresh: Default >> 50 (3dupACK) >> ~26 (timeout)

CWND: 100 >> 1(3dupACK) >> slow start >> slow start >> reach 50 >> additive increase

ssthresh: Default >> 50 (3dupACK) >> 50

Window Size

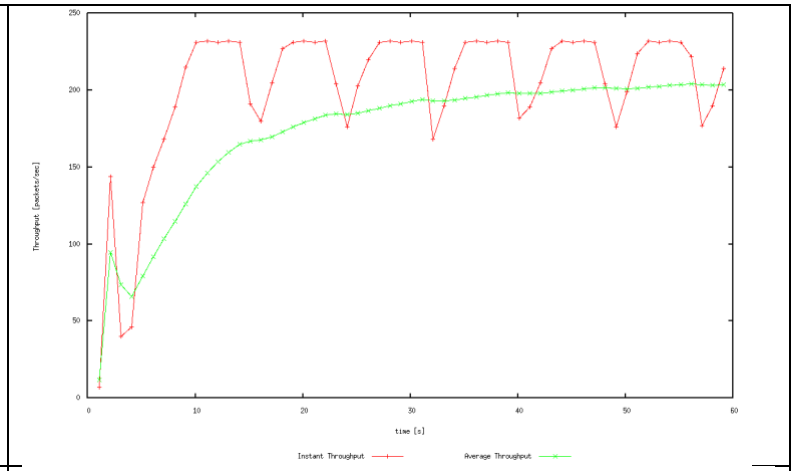
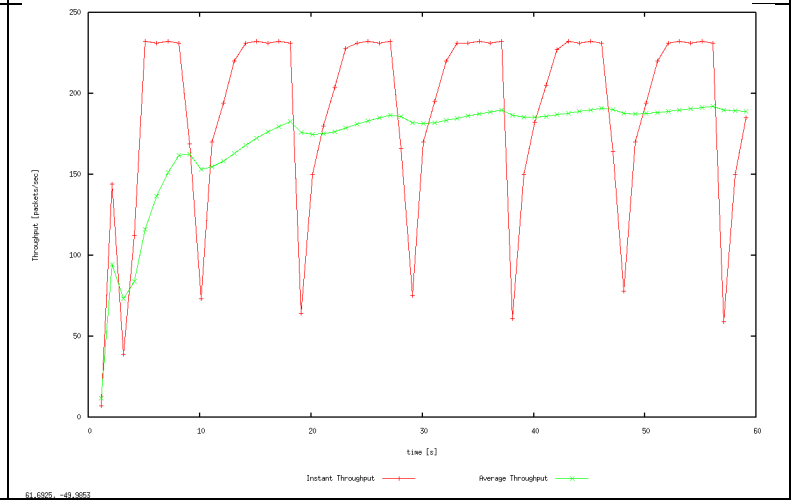
Reno	
Tahoe	

Reno: avg would be around 200 packet/sec. 540*8*200 = 864000 bits/sec = 864kbps

Tahoe: 820.8 kbps

Reno is 43.2kbps higher than Tahoe. More efficient for applying Reno in TCP.

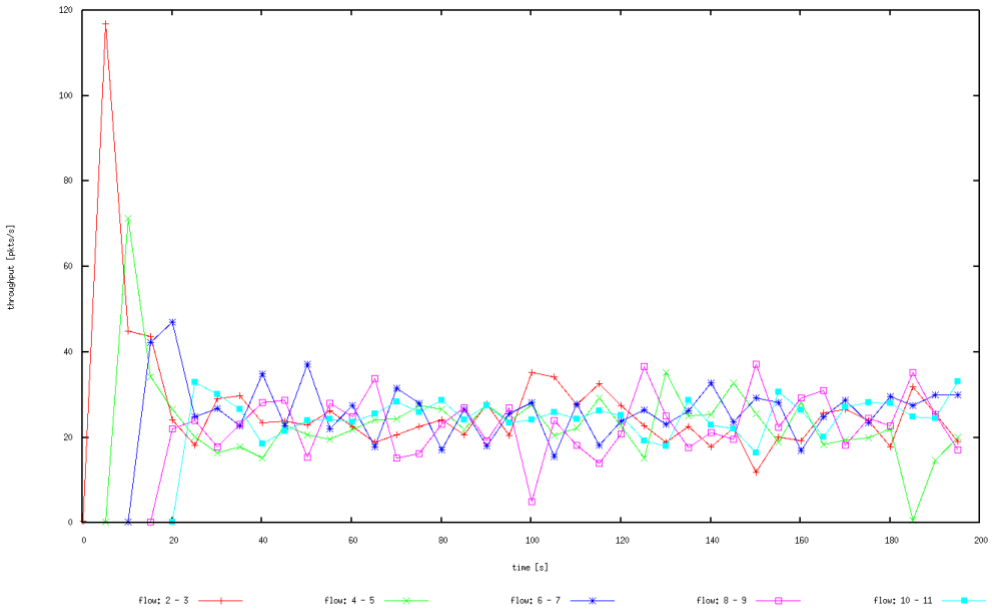
Throughput

Reno	
Tahoe	

EX2Q1: Does each flow get an equal share of the capacity of the common link (i.e., is TCP fair)? Explain which observations lead you to this conclusion.

Ans:

In the very first (flow 2-3), when a new TCP (flow 4-5) connection was established and then transfer data through this link, the throughput of those established connection would drop down a lot in order to balance the throughput of each TCP connection. After all TCP connection had connected and transferred through the link, we can observe that throughput of each connection actually was balanced. Hence, it's quite fair for each connection in the link.



EX2Q2: What happens to the throughput of the pre-existing TCP flows when a new flow is created? Explain the mechanisms of TCP which contribute to this behaviour. Argue about whether you consider this behaviour to be fair or unfair.

Ans:

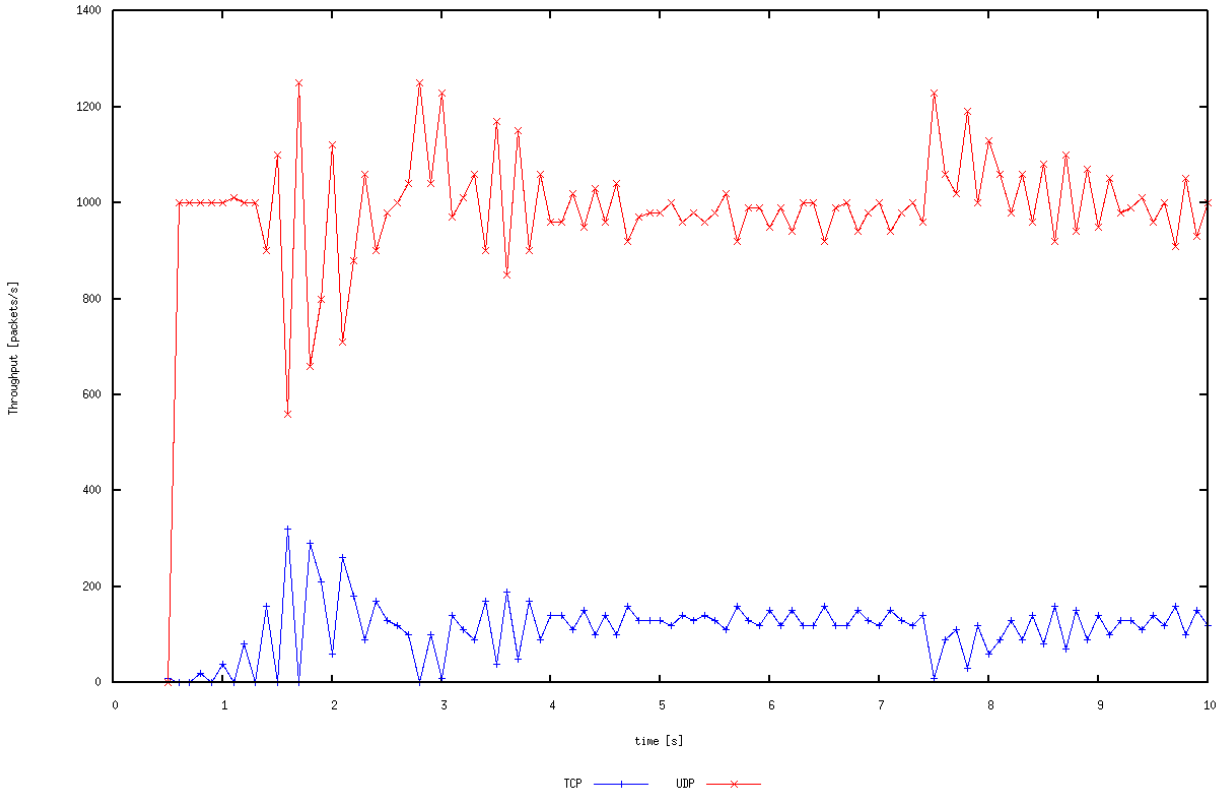
In the beginning, only flow2-3 was transferring data through the link, flow2-3 in slow start state would tend to increase the congestion window exponentially. When throughput of flow2-3 exceed the capacity of link, triple DupACK were received and state changed to Fast Recovery. Once the throughput of flow 2-3 decreased, now flow 4-5 could establish connection and transfer the data in slow start state. Then flow 2-3 and flow 4-5 would keep increasing throughput until packet loss. Therefore, if 3dupACK or Timeout occurred, the pre-existing TCP flows would decrease window. This behaviour could let link have some capacity available to accept new connection flow next.

As we can observe that the throughput actually oscillating dynamically: 3dupACK, timeout and different stage transfer could happen simultaneously. Ideologically it's fair on the assumption of same MSS and RTT of each TCP flow. But in reality, flow connection with smaller MSS and RTT might rapidly occupy the link if 3DupACK or timeout from other flow.

EX3Q1: How do you expect the TCP flow and the UDP flow to behave if the capacity of the link is 5 Mbps?

Ans:

I would expect that UDP would get the majority capacity of link due to its simpleness without congestion control. if it's the worst case, UDP connection would exclude TCP connection mercilessly out of the Link because TCP would decrease the window sizes when packet loss which UDP would not care about. Finally, the simulation plot can match with my expectation.



EX3Q2: Why does one flow achieve higher throughput than the other? Try to explain what mechanisms force the two flows to stabilise to the observed throughput.

Ans:

TCP have its own congestion control. This would allow TCP connection evaluating the entire network and cooperate with each other. TCP would tend to decrease the window sizes while packet loss in order to release the stress of congestion.

However, UDP doesn't have this congestion control and have no side effect if packet loss. Hence, UDP typically just keep sending datagram during times.

Hence, in the simulation plot, UDP flow could generally keep the throughput (rate of sending packets) in the higher value compared to throughput of TCP flow.

EX3Q3: List the advantages and the disadvantages of using UDP instead of TCP for a file transfer, when our connection has to compete with other flows for the same link. What would happen if everybody started using UDP instead of TCP for that same reason?

Ans:

general pros & cons:

pros	cons
Could be transfer fast without consideration of congestion control	File might be damaged or inexecutable due to some pkt loss.

Personally, for file transfer, if there's no implementation of reliable data transfer, I wouldn't consider UDP an option for transferring file.

If everyone started using UDP rather than TCP, I would say that Network would become unpredictable based on my knowledge. Network would keep full loaded all the time. Even implementation of reliable data transfer does not work, because tons of timers would be trigger and then send a lot of redundant packet to crash the network which has already crashed.

Hence, sender might not know the exact time the entire file would be transferred completely