01 - Lesson 1 Intro  
  
>> In this section of the course, we'll learn about resource control and content distribution. Resource control deals with handling bandwidth constraints on links. And in this first section, we'll focus on controlling congestion along links. >> To learn more about TCPA and congestion control, your Minia project will explore what happens when congestion control goes wrong.

02 - Congestion Control  
  
Okay, we're starting course two on congestion control and streaming. And we'll first talk about congestion control. In particular, what is congestion control and why do we need it? Simply put, the goal of congestion control is to fill the Internet's pipes without overflowing them. So to think about this in terms of an analogy, suppose you have a sink. And you're filling that sink with water. Well, how should you control the faucet? Too fast and the sink overflows. Too slow and you are not efficiently filling up your sink. So what you would like to do is to fill the bucket as quickly as possible without overflowing. The solution here is to watch the sink. And as the sink begins to overflow, we want to slow down how fast we're filling it. That's effectively how congest control works.

03 - Congestion  
  
Let's suppose that in a network we have three hosts shown as squares at the edge of the network that are connected by links with capacities as shown. The two senders on the left can send at rates of 10 megabits per second and 100 megabits per second, respectively. But the link to the host on the right is only 1.5 megabits per second. So these different hosts on the left are actually competing for the same resources inside the network. So the sources are unaware of each other and also of the current state of whatever resource they are trying to share. In this case How much other traffic is on the network. This shows up as lost packets or long delays and can result in throughput that's less than the bottleneck link, something that's also known as congestion collapse.

04 - Congestion Collapse  
  
In congestion collapse, an increase in traffic load suddenly results in a decrease of useful work done. As we can see here, up to a point, as we increase network load, there is an increase in useful work done. At some point, the network reaches saturation, at which point increasing the load no longer results in useful work getting done. But at some point, actually increasing the traffic load can cause the amount of work done or the amount of traffic forwarded to actually decrease. There are many possible causes. One possible cause is the spurious re-transmissions of packets that are still in flight. So when senders don't receive acknowledgements for packets in a timely fashion, they can spuriously re-transmit, thus resulting in many copies of the same packets being outstanding in the network at any one time. Another cause of congestion collapse is simply undelivered packets, where packets consume resources and are dropped elsewhere in the network. The solution to spurious re-transmissions is to have better timers and to use TCP congestion control, which we'll talk about next. The solution to undelivered packets is to apply congestion control to all traffic. Congestion control is the topic of the rest of this lesson.

05 - Congestion Collapse Quiz  
  
Lets take a quick quiz. What are some possible causes of congestion collapse? Faulty router software? Spurious retransmissions of packets in flight? Packets that are travelling distances that are too far between routers? Or, undelivered packets? Please check all options that apply.

06 - Congestion Collapse Solution  
  
Congestion collapse, is caused by spurious retransmissions of packets in flight and, undelivered packets that consume resources in the network but achieve no useful work.

07 - Goals of Congestion Control  
  
Congestion control has two main goals. The first is to use network resources efficiently. Going back to our sink analogy, we'd like to feel the sink as quickly as possible. Fairness, on the other hand, ensures that all the senders essentially get their fair share of resources. A final goal, of course, is to avoid congestion collapse. Congestion collapse isn't just a theory. It's actually been frequently observed in, in many different networks.

08 - Two Approaches to Congestion Control  
  
There are two basic approaches to congestion control. End-to-end congestion control and network assisted congested control. In end-to-end congestion control the network provides no explicit feedback to the senders about when they should slow down their rates. Instead, congestion is inferred typically by packet loss, but potentially also by increased delay. This is the approach taken by TCP congestion control. In network assisted congestion control routers provide explicit feedback about the rates that end systems should be sending in. So they might set a single bit indicating congestion, as is the case in TCP's ECN or explicit congestion notification extensions, or they might even tell the sender an explicit rate that they should be sending at. We're going to spend the rest of the lesson talking about TCP congestion control.

09 - TCP Congestion Control  
  
In TCP congestion control, the senders continue to increase their rate until they see packet drops in the network. Packet drops occur because the senders are sending at a rate that is faster than the rate at which a particular router in the network might be able to drain its buffer. So you might imagine, for example, that if all three of these senders are sending at a rate that is equal to the rate at which the router is able to send traffic downstream, then eventually this buffer will fill up. TCP interprets packet loss as congestion. And when senders see packet loss, they slow down as a result of seeing the packet loss. This is an assumption. Packet drops are not a sign of congestion in all networks. For example, in wireless networks, there may be packet loss due to corrupted packets as a result of interference. But in many cases, packet drops do result, because some router in the network has a buffer that has filled up and can no longer hold anymore packets and hence it drops the packets as they arrive. So senders increase rates until packets are dropped. Periodically probing the network to check whether more bandwidth has become available. Then they see packet loss, interpret that as congestion, and slow down. So, congestion control has two parts. One is an increase algorithm, and the other is a decrease algorithm. In the increase algorithm, the sender must test the network to determine whether the network can sustain a higher sending rate. In the decrease algorithm, the senders react to congestion to achieve optimal loss rates, delays in sending rates. Let's now talk about how senders can achieve these increase and decrease algorithms.

10 - Two Approaches to Adjusting Rate  
  
What approach is a window based [UNKNOWN]? In this approach, a sender can only have a certain number of packets outstanding or quote, in flight. And the sender uses acknowledgement from the receiver to clock the retransmission of new data. So let's suppose that the sender's window was four packets. At this point, there are four packets outstanding in the network. And the sender cannot send additional packets, until it has received an acknowledgement from the receiver. When it recieves an acknoweldgemnt on ACK from the reciever. The sender, can then, send another packet. So at this point there are still four outstanding or four unacknowledged packets in flight. In this case if a sender wants to increase the rate at which it's sending, it simply needs to increase the window size. So for example, if the sender wants to send at a faster rate, it can increase the window size from four, to five. A sender might increase it's rate anytime it sees an acknowledgement from the receiver. In TCP every time a sender receives an acknowledgement, it increases the window size. Upon a successful receipt, we want the sender to increase its window by one packet per round trip. So for example in this case, if the sender's window was initially four packets, then at the end,of a single round trip's worth of sending, we want the next set of transmissions to allow five packets to be outstanding; this is call Additive Increase. If a packet is not acknowledged, the window size is reduced by half, this is called Multiplicative Decrease. So TCP's congestion control is called additive increase multiplicative decrease, or AIMD. The other approach to adjusting rates is an explicit rate-based congest control album. In this case the sender monitors the loss rate and uses a timer to modulate the transmission rate. Window based congestion controal, or AIMD, is the common way of performing congestion control in today's computer networks. In the next lesson we will talk about the two goals of TCP congestion control further, efficiency and fairness. And explore how TCP achieves those goals.

11 - Window Based Congestion Control Quiz  
  
Let's have a quick quiz on window based congestion control. Suppose the round trip time between the sender and receiver is 100 milliseconds, each packet is 1 kilobyte, and the window size is 10 packets. What is the rate at which the sender is sending? Please put your answer here in terms of kilobits per second, keeping in mind that 1 byte is 8 bits.

12 - Window Based Congestion Control Solution  
  
The sending rate of the sender is approximately 800 kbps. With a window size of ten packets and a round trip time of 100 milliseconds the sender can send 100 packets per second. With each packet being one kilobyte Or 8000 bits, that gives us 800,000 bits per second or about 800 kilobits per second.

13 - Fairness and Efficiency in Congestion Control  
  
The two goals of congestion control are fairness; meaning every send gets their fair share of the network resources. And efficiency meaning that the network resources are used well. In other words we shouldn't have a case where there are spare capacity or resources in the network And senders have data to send, but are not able to send it. So, we'd like the network to be used efficiently, but we'd also like it to be shared among the senders. We can represent fairness and efficiency in terms of a phase plot, where each axis representers a particular user or particular senders. Allocation. In this case we just have two users, one and two, and we represent their allocations with x1 and x2. If the capacity of the network is C, then we can represent the optimal operating line. As X1 + X2 being some constant, C. Anything to the left of this diagonal line represents under utilization of the network, and anything to the right of the line represents overload. We can also represent another line, X1 = X2 as some notion of fair allocation. So the optimal point, is where the network is neither under or over utilized, and when the allocation is fair. So being on this diagonal line represents efficiency, and being on the green diagonal line represents fairness. We can use the phase plot to understand why senders who use additive increase, multiplicative decrease, converge to fairness. The senders also converge, to the efficient operating point. Let's suppose that we start at the operating point shown in blue. At this point both senders will additively increase their sending rates. Additive increase results in moving along a line that is parallel to x1 and x2, since both senders increase their rate by the same amount. Additive increase will continue. Until the network becomes overloaded. At this point the senders will see a loss and perform multiplicative decrease. In multiplicative decrease each sender decreases its rate by some constant factor of its current sending rate. For example, suppose each one of these senders decreases its sending rate by half. The resulting operating point, is shown by this second blue dot. Note that that new operating point, as a result of multiplicative decrease, is on a line between the point on the efficiency line, that the centers hit, and the origin. At this point the sender's will again increase their sending rate along a line that's parallel to X1 equals X2 until they hit over load again. At which point they will again retreat towards the origin. You can see that eventually, the senders will reach this optimal operating point through the path. That's delineated by the blue line. To think about this a bit more you can see that every time additive increase is applied, that increases efficiency. Every time multiplicative decrease is applied that improves fairness because every time. We apply multiplicative decrease, we get closer to this x1 equals x2 line.

14 - AIMD  
  
The result is the additive increase multiplicative decrease congestion control algorithm. The algorithm is distributed, meaning that all the senders can act independently, and we've just shown using the phase plots that it's both fair and efficient. To visualize this sending rate over time, the sender's sending rate looks roughly as shown. We call this the TCP sawtooth behavior or simply the TCP sawtooth. TCP periodically probes for available bandwidth by increasing its rate using additive increase. When the sender reaches a saturation point by filling up a buffer in a router somewhere along the path, it will see a packet loss, at which point it will decrease its sending rate by half. You can thus see that a TCP sender sends, a sending rate shown by the dotted green line that is halfway between the maximum window size at which the sender sends, and half that rate, which it backs off to when it sees a loss. You can see that between the lowest sending rate and the highest is w m over 2 plus 1 round trips. Now given that rate we can compute the number of packets between periods of packet loss, and compute the loss rate from this. The number of packets sent for every packet lost is the area of this triangle. So the lost rate is on the order of the square of the maximum window divided by some constant. Now, the throughput is the average rate, 3 4ths w max divided by the RTT. Now if we want to relate the throughput to the loss rate, where we call the loss rate p, and the throughput lambda, we simply need to solve for w m. And I'm just going to get rid of the constant. So a loss occurs once for this number of packets, so the loss rate is simply 1 over that quantity. And then when we solve for w m and plug in for throughput, we see that the throughput is inversely proportional to both the round trip time and the square root of the loss rate.

15 - Data Centers and TCP Incast  
  
We'll now talk about TCP congestion control in the context of modern datacenters. And we'll talk about a particular TCP throughput collapse problem called the TCP incast problem. A typical data center consists of a set of server racks. Each holding a large number of servers, the switches that connect those racks of servers, and the connecting links that connect those switches to other parts of the topology. So the network architecture is typically made up of some sort of tree. And, switching elements that progressively are more specialized and expensive as we move up the network hierarchy. Some of the characteristics of a data center network include a high fan in, there is a very high amount of fan in between the leaves of the tree and the top of the root. Workloads are high bandwidth and low latency, and many clients issue requests in parallel, each with a relatively small amount of data per request. The other constraint that we face, is that the buffers in these switches can be quite small. So when we combine the requirements of high bandwidth and low latency for the applications, the presence of many parallel requests coming from these servers. And the fact that the switches have relatively small buffers, we can see that potentially there will be a problem. The throughput collapse that results from this phenomenon is called the TCP Incast problem. Incast is a drastic reduction in application throughput that results when servers using TCP. All simultaneously request data, leading to a gross underutilization of network capacity in many-to-one communication networks like a datacenter. The filling up of the buffers here at the switches result in bursty retransmissions that overfill the switch buffers. And these bursting retransmissions are cause by TCP timeouts. The TCP timeouts can lasts hundreds of milliseconds. But the roundtrip time in a data center network, is typically less than a millisecond. Often just hundreds of microseconds. Because the roundtrip times are so much less than TCP timeouts. The centers will have to wait for the TCP timeout. Before they retransmit. An application throughput can be reduced by as much as 90% as a result of link idle time.

16 - Barrier Synchronization and Idle Time  
  
A common request pattern in data centers today is something called barrier synchronization whereby a client or an application might have many parallel threads. And no forward progress can be made until all the responses for those threads are satisfied. For example, a client might send a synchronized read with four parallel requests, but suppose, that the fourth is dropped. At this point, we have a request, sent at time zero, then we see a response less than a millisecond later, and at this point, threads one to three complete but TCP may time out on the fourth. In this case, the link is idle for a very long time while that fourth connection is timed out. The addition of more servers in the network induces an overflow of the switch buffer, causing severe packet loss, and inducing throughput collapse. One solution to this problem is to use fine grained TCP retransmission timers, on the order of microseconds, rather than on the order of milliseconds. Reducing the retransmission timeout for TCP, thus improves system throughput. Another way to reduce the network load is to have the client acknowledge every other packet rather then every packet. Thus reducing the overall network load. The basic idea here, and the premise, is that the timers need to operate on a granularity that's close to the round-trip time of the network. In the case of a data center that's hundreds of microseconds or less.

17 - TCP Incast Quiz  
  
As a quick review what are some solutions to the TCP incast problem? Having senders send smaller packets. Using finer granularity TCP timeout timers. Having the clients send fewer ackowledgements or having fewer TCP senders?

18 - TCP Incast Solution  
  
Using finer granularity timers and having the clients acknowledge only every other packet, as oppose to every packet, are possible solutions to the TCP Incast problem.

19 - Multimedia and Streaming  
  
In this lesson, we'll talk about multimedia and streaming. We'll talk about digital audio and video data, multimedia applications, in particular, streaming audio and video for playback. Multimedia transfers over a best-effort network, in particular, how to tolerate packet loss delay and jitter, and quality of service. So we use multimedia and streaming video very frequently on todays internet. YouTube streaming videos are an example of multimedia streaming as are applications for video or voice chat such as Skype or Google Hangout. In this lecture we'll talk about the challenges for streaming these types of applications over best effort networks. As well has how to solve those challenges. We'll also talk about the basics of digital audio and video data. First of all, let's talk about the challenges for media streaming.

20 - Challenges  
  
One challenge is that, there's a large volume of data. Each sample, is a sound or an image and there are many samples per second. Sometimes because of the way data is compressed, the volume of data that's being sent may vary over time. In particular the data may not be set at a constant rate. But in streaming, we want smooth playout. So the variable volume of data can pose challenges. Users typically have a very low tolerance for delay variation. Once playout of a video starts for example, you want that video to keep playing. It's very annoying if once you've started playing, that the video stops. The users might have a low tolerance for delay period. So in cases like games or Voice over IP, delay is typically just unacceptable. Although users can tolerate some loss. Before we get into how the network solves these challenges. Let's talk a little bit about digitizing audio and video.

21 - Digitizing Audio and Video  
  
Suppose we have an analog audio signal that we'd like to digitize, or send as a stream of bits. What we can do is sample the audio signal at fixed intervals, and then represent the amplitude of each sample with a fixed number of bits. For example, if our dynamic range was from 0 to 15, we could quantize the amplitude of this signal such that each sample could be represented with four bits.

22 - Digitizing Audio and Video Quiz 1  
  
Lets take a couple of examples. So with speech you might take 8000 samples per second, and you might have 8 bits per sample. So what is the sampling rate in this case? Please give your answer in kilobits per second.

23 - Digitizing Audio and Video Solution 1  
  
At 8,000 samples per second, and eight bits for every sample, the rate of digitized speech would be 64 kbps which is a common bit rate for audio.

24 - Digitizing Audio and Video Quiz 2  
  
suppose we have a MP3 with 10,000 samples per second and 16 bits per sample. What's the resulting rate in this case? In kbps?

25 - Digitizing Audio and Video Solution 2  
  
The resulting rate in this case is, a 160 kbps.

26 - Video Compression  
  
Video compression works in slightly different ways. Each video is a sequence of images and each image can be compressed with spacial redundancy, exploiting aspects that humans tend not to notice. Also there is temporal redundancy. Between any two video images, or frames, there might be very little difference. So if this person was walking towards a tree, you might see a version of the image that's almost the same, except with the person shifted slightly to the right. Video compression uses a combination of static image compression on what are called reference frames, or anchor frames, sometimes called I frames, and derived frames, sometimes called P frames. The P frame can be represented as the I frame, compressed. If we take the I frame and divide it into blocks, we can then see that the P frame is almost the same except for a few blocks here that can be represented in terms of the original I frame blocks, plus a few motion vectors. A common video compression format that's used on the internet is called MPEG.

27 - Streaming Video  
  
In a streaming video system, where the server streams stored audio and video, the server stores the audio or video files, the client requests the files and plays them as they download. It's important to play the data at the right time. The server can divide the data into segments and then label each segment with a time stamp. Indicating the time at which that particular segment should be played so the client knows when to play that data. The data must arrive at the client quickly enough, otherwise the client can't keep playing. The solution is to have a client use what's called a playout buffer, where the client stores data as it arrives from the server, and plays the data for the user in a continuous fashion. Thus, data might arrive more slowly or more quickly from the server, but as long as the client is playing data out of the buffer at a continuous rate, the user sees a smooth playout. A client may typically wait a few seconds before it starts playing the stream to allow data to be built up in this buffer. To account for cases when the server might have times where it is not sending at a rate that's sufficient to satisfy the client's playout rate.

28 - Playout Delay  
  
Looking at this graphically, we might see packets generated at a particular rate and the packets might be received at slightly different times, depending on network delay. These types of delays are the types that we want to avoid when we playout, so if we wait to receive several packets. And fill the buffer before we start playing, say to here. Then, we can have a playout schedule that is smooth, regardless of the erratic arrival times that may result from network delays. So this playout delay or buffering allows the client to achieve a smooth playout. Some delay at the beginning of the playout is acceptable. Startup delays of a few seconds are things that users can typically tolerate, but clients cannot tolerate high variation in packet arrivals if the buffer starves or if there aren't enough packets in the buffer. Similarly, small amount of loss or missing data does not disrupt the playback but retransmitting a lost packet, might actually take too long and result in delays or starvation of the playout buffer.

29 - Streaming Quiz  
  
So as a quick review which of these pathologies can streaming audio and video tolerate? Packet loss, delay, variation in delay, or jitter.

30 - Streaming Solution  
  
Some delay at the beginning of a packet stream is acceptable. And similarly, some small amount of missing data is okay. We can tolerate small amounts of missing data that result in slightly reduced quality of the audio or video stream. On the other hand, a receiver is not very good at tolerating variability in packet delay in the packet stream, particularly if the client buffer is starved.

31 - TCP is Not a Good Fit  
  
It turns out that TCP is not a good fit for congestion control for streaming video, or streaming audio. TCP retransmits lost packets, but retransmissions may not always be useful. TCP also slows down its sending rate after packet loss, which may cause starvation of the client. There's also a fair amount of overhead in the protocol. A TCP header of 20 bytes for every packet is large for audio samples. And sending acknowledgements for every other packet may be more feedback than is needed. Instead, one might consider using UDP. UDP does not retransmit lost packets and it does not automatically adapt the sending rate. It also has a smaller header. Because UDP does not automatically retransmit packets or adjust the sending rate, many things are left to higher layers. Potentially the application, such as when to transmit the data, how to encapsulate it, whether to retransmit, and whether to adapt the sending rate, or to adapt the quality of the video or audio encoding. So higher layers must solve these problems. In particular, the sending rate still needs to be friendly or fair to other TCP senders, which may be sharing the link. There are a variety of video streaming and audio streaming transport protocols that are built on top of UDP, that allows senders to figure out when and how to retransmit lost packets and how to adjust sending rates.

32 - More Streaming  
  
We're going to talk a little bit more about streaming and in particular, specific applications and how they work. We'll talk about a streaming video application, YouTube, and a Voice over IP or streaming audio application, Skype. With YouTube, all uploaded videos are converted to Flash or html5 and nearly every browser has a Flash plugin. Thus, every browser can essentially play these videos. HTTP and TCP are implemented in every browser and the streams easily get through most firewalls. So, in this case, even though we've talked about why TCP is suboptimal for streaming, the designers of YouTube decided to keep things simple, potentially at the expense of video quality. When a client makes an HTTP request to youtube.com, it is redirected to a content distribution network server located in a content distribution network, such as Limelight or perhaps even YouTube's own content distribution network. We will talk about content distribution networks later on in this course. When the client sends an HTTP get message to this CDN server, the server responds with a video stream. Similarly with Skype, or Voice Over IP, your analog signal is digitized through an A to D conversion and this resulting digitized bit stream is sent over the internet. In the case of Skype, this A to D conversion happens by way of the application. And in the case of Voice over IP this conversion might be performed with some kind of phone adapter that you actually plug your phone into. An example of that might be Vonage. In VoIP with an analogue phone the adapter converts between the analogue and digitial signal. It sends and recieves data packets and then communicates with the phone in a standard way. Skype, on the other hand, is based on what's known as peer-to-peer technology. Where individual users, using Skype, actually route voice traffic through one another. We will talk more about peer-to-peer content distribution, later in this course.

33 - Skype  
  
So Skype has a central log in server but then use a pier to pier to exchange the actual voice streams and compresses the audio to achieve a fairly low bit rate t around 67 bytes per packet and 140 packets per second. We see Skype uses about 40 kilobites per second in each direction. Data packets are also encrypted in both directions. Good audio compression and avoiding hops through faraway hosts can improve the quality of the audio, as can for arrow correction. Ultimately, though, the network is a major factor. Long propagation delays High congestion, or disruptions as a result of routing changes can all degrade the quality of a voice over IP call. To ensure that some streams achieve acceptable performance levels, we sometimes use what's called Quality of Service. One way of doing Quality of Service is through explicit resservations. But, another way is to simply mark certain packet streams as higher priorities than others. Let's first take a look at marking and policing of traffic sending rates.

34 - Marking and Policing  
  
So we know from before that applications compete for bandwidth. Consider a Voice over IP application and a file transfer application that want to share a common network link. In this case, we'd like the audio packets to receive priority over the file transfer packets. Since the user's experience can be significantly degraded by lost or delayed audio packets. In this case, we want to mark the audio packets as they arrive at the router, so that they receive a higher priority. Then the file transfer packets. You might imagine implementing this with priority queues where the void packets were put in one queue and the file transfer packets are put in a separate queue that is served less often than a high priority queue. An alternative is to allocate fixed ban width per application, but the problem with this alternative. Is that it can result in inefficiency if one of the flows doesn't fully utilize its fixed allocation. So the idea, in addition to marking and policing, is to apply scheduling. One way of applying scheduling is to use what's called weighted fair queuing, whereby the queue with the green packets is served more often than the queue with the red packets. Another alternative is to use admission control, whereby an application declares its needs in advance, and the network may block the application's traffic if the application can't satisfy the needs. A busy signal on a telephone network is an example of admission control. But can you imagine how annoying it would be if you attempted to load a web page and were blocked. This blocking or the user experience that results from it, is a negative consequence of admission control and is one of the reasons it's not commonly applied for internet applications.

35 - QoS Quiz  
  
So as a quick quiz what are some commonly used Quality of Service techniques for streaming audio and video? Marking packets and putting them into queues with different priorities, scheduling the service of these queues at different rates depending on the priority of the application, blocking flows with admission control if the network can't satisfy their rates, or performing fixed bandwidth allocations? Please check all that apply.

36 - Qos Solution  
  
A common way of applying QoS for streaming applications is to mark the packets as a higher priority, put those packets into a higher priority queue, and then schedule that queue so that it's serviced more regularly, or more frequently, than traffic or applications that are lower priority.

37 - Parking Lot Problem  
  
Recall that congestion collapse occurs when packets consume valuable network resources only to get dropped later at a downstream link. And although the network is forwarding packets, none of them actually reach the destination and no useful communication occurs, resulting in what we called congestion collapse. We talked about congestion collapse in an earlier lesson in this course. Recall that the goal of TCP is to prevent congestion collapse. In this assignment, you will become familiar with the Mininet environment, creating custom network topologies, running programs in virtual hosts that generate TCP traffic, and learn about TCP congestion control, an the TCP sawtooth. You'll also see how bandwidth is shared across multiple flows. We are going to explore these concepts in a particular topology that we'll just call a parking lot. So in the assignment, you'll create the topology below with the following parameters. N will be the number of receivers connected via this switch like topology. The data rate of each link will be 100 megabits per second and the delay of each link will be one millisecond. Now your going to used iperf to generate simultaneous TCP flows from each of the senders. To the lone receiver. Then you going to plot the time series of throughput versus time for each of these senders. For each of your experiments as you vary the value of N. Your plot should run for 60 seconds. We have given an initial file called parkinglot.py. Your goal is to fill in the gaps. You'll need to both fill in the part that sets up the topology using a parameter of N, and then you'll need to fill in the part that actually generates the TCP flows between the senders, and the receiver using Iperf, and monitors the throughput of each of these flows.