01 - Traffic Engineering Intro  
  
>> Whew. We just completed our exploration of software defined networking, and now we're jumping into traffic engineering. >> Traffic engineering is a fancy way of describing how network operators deal with large amounts of data flowing through their networks. Companies like Google are doing exciting work in this area, so be sure to pay attention to the instructor note section, where we'll highlight exciting research papers.

02 - Traffic Engineering Overview  
  
This lesson covers traffic engineering. Traffic engineering is the process of reconfiguring the network in response to changing traffic loads to achieve some operational goal. A network operator might want to reconfigure the network in response to changing loads to, for example, maintain traffic ratios in a peering relationship or to relieve congestion on certain links in the network or to balance load more evenly across the available links in the network. In this lesson we will explore different ways of performing traffic engineering. We will start by looking at conventional approaches to traffic engineering and we'll explore how software-defined networking is used to make the process of traffic engineering easier in both data center networks and transit networks. IP networks must be managed. Now, in some sense, IP networks manage themselves. There are several examples of protocols on the internet that in some sense manage themselves. TCP senders send less traffic during congestion and routing protocols will adapt to topology changes. The problem is that even though these protocols are designed to adapt to various changes, the network may not run efficiently. There may be congested links when idle paths exist or there might be a high-delay path that some traffic is taking when a low-delay path, in fact, exists. So key question that traffic engineering tries to address is how routing should adapt to traffic. In particular, traffic engineering attempts to avoid congested links and satisfy certain application requirements such as delay. These are, in some sense, the essential questions that traffic engineering attempts to answer. In the rest of this lesson we will look at how network operators can tune a routing protocol configuration to affect how network traffic traverses the links in the network. And in particular we will look at both intradomain traffic engineering, that is how to reconfigure the protocols within a single autonomous system to adjust traffic flows, as well as interdomain traffic engineering, or how to adjust how traffic flows between autonomous system. We will also look at multi-path routing and how it is used to achieve various traffic engineering goals. Let's start by looking how network operators can tune static link weights in a routing protocol like OSPF or ISIS, to affect how traffic flows through the network.

03 - Interdomain Traffic Engineering  
  
Let's assume that we have a single autonomous system with static link weights as shown. In such a setup, routers will flood information to one another to learn the network topology, including the link weights on, links connecting individual routers. An operator can affect how traffic flows through the network. By configuring the link weights. By changing the link weight configuration, an operator can affect the shortest path between two points in this graph. Thus, affecting the way that traffic flows. In the link weight settings shown here, traffic would flow along the green path. Suppose that the operator would like to shift traffic off of a congested link in the middle of the network. Such as this one. By changing the link weight from one to three. The shortest path between this node and this node, now takes an alternate route. So we can see from this simple example, That, by adjusting the link weights in a intra-domain topology, the operator can affect how traffic flows between different points in the network, thus affecting the load on the network links. In practice, network operators set these link weights in a variety of ways. One could set the link weights inversely proportional to capacity. Proportional to propagation delay, or the operator might perform some network wide optimization, based on traffic.

04 - Measuring, Modeling and Controlling Traffic  
  
Traffic engineering has three steps: measuring the network to figure out the current traffic loads, forming a model of how configuration affects the underlying paths in the network, and then ultimately reconfiguring the network to exert control over how the traffic flows through the network. An operator might measure the topology and traffic, feed the topology and traffic to a what-if model that predicts what will happen under various type of configuration changes, decide which changes to affect on the network, and then, ultimately, control the network behavior by readjusting link weights. So in summary, we have measurement, modeling and control. Each of these three components requires a significant amount of heavy lifting to make both practical and accurate in practice. Intradomain traffic engineering attempts to solve an optimization problem, where the input is the graph G, where R is the set of routers, and L is the set of unidirectional links. Each link L also has a fix capacity. Another input is the traffic matrix or offered traffic load, where Mij represent the traffic load from router i to router j. The output of the optimization problem is a set of link weights, where wl is the weight on any unidirectional link l in the network topology. Ultimately, the setting of these link weights should result in a fraction of the traffic from i to j traversing each link l, such that those fractions satisfy the network-wide objective function. Defining an objective function is tricky. We could talk about, for example, minimizing the maximum congested link in the network, evenly splitting traffic loads across links, and so forth.

05 - Link Utilization Function  
  
What we'd like to represent is that the cost of congestion increases in a quadratic manner as the loads on the links continue to increase. Ultimately becoming increasingly expensive as link utilization approaches one. Solving the optimization problem, however, is much easier if we use a piecewise linear cost function. We can define utilization as the amount of traffic on the link divided by the capacity and our objective might be to minimize the sum of this piecewise linear cost function over all the links in the network. Unfortunately, solving this optimization is still NP-complete, which means that there is no efficient algorithm to find the optimal setting of link weights, even for simple objective functions. The implications of this are that we have to resort to searching through a large set of combinations of link weight settings to ultimately find a good setting. So clearly searching through all the link weights is suboptimal. But the graphs turn out to be small enough in practice such that this approach is still reasonably effective. In practice, we also have other operational realities to worry about. For example, minimizing the number of changes to the network. Often just changing one or two link weights is enough to achieve a good traffic load balance solution. Whatever solution we come up with must be resistant to failure and it should be robust to measurement noise. We also want to limit the frequency of changes that we make to the network. This completes our overview of intradomain routing. And now we will take a look at interdomain routing. Intradomain routing and traffic engineering concerns traffic flow within a single domain, such as an ISP, a campus network, or a data center. In contrast, interdomain routing, and interdomain traffic engineering, concerns routing that ocurs between domains, something that we looked at before in the context of the Borda Gateway Protocol.

06 - Interdomain Routing Quiz  
  
Before we proceed with our discussion of interdomain traffic engineering, let's take a brief quiz reminding ourselves about the differences between intradomain routing and interdomain routing. Which of the following are examples of interdomain routing? Peering between two internet service providers. Peering between a university network, and its ISP. Peering at an internet exchange point. Routing inside a data center. Or routing across multiple data centers. Please again, check all that apply.

07 - Interdomain Routing Quiz Answer  
  
Peering between two ISPs involves inter-domain routing, as does peering between a university network and its ISP. Or any peering at an Internet exchange point, for that matter. Routing between multiple data centers typically involves routing in a wide area, and hence may involve inter-domain routing. Routing within a single data center concerns routing in a single domain and hence does not concern inter-domain routing.

08 - BGP in Interdomain Traffic Engineering  
  
1 00:00:00,025 --> 00:00:03,910 Inter-domain routing concerns routing between 2 00:00:03,910 --> 00:00:06,810 domains or autonomous systems. It involves 3 00:00:06,810 --> 00:00:10,720 the reconfiguration of the border gateway 4 00:00:10,720 --> 00:00:14,280 protocol, policies or configurations that are 5 00:00:14,280 --> 00:00:19,730 running on individual routers in the network. Changing BGP policies at these 6 00:00:19,730 --> 00:00:25,440 edge routers can cause routers inside an autonomous system to direct 7 00:00:25,440 --> 00:00:32,380 traffic to or away from certain edge links. We can also change the set of egress 8 00:00:32,380 --> 00:00:38,310 links for a particular destination. For example, an operator of autonomous 9 00:00:38,310 --> 00:00:41,880 system one might observe. Traffic to destination 10 00:00:41,880 --> 00:00:45,160 D traversing the green path. But by 11 00:00:45,160 --> 00:00:50,550 adjusting B to B policies, the operator might balance load across these two edge 12 00:00:50,550 --> 00:00:52,780 links, or shift all of the traffic for 13 00:00:52,780 --> 00:00:56,340 that destination. To the lower path. An operator might 14 00:00:56,340 --> 00:00:59,290 wish to use inter domain traffic engineering if 15 00:00:59,290 --> 00:01:01,720 an edge link is congested, if a link is 16 00:01:01,720 --> 00:01:05,420 upgraded, or if there's some violation of appearing 17 00:01:05,420 --> 00:01:10,000 agreement. For example, AS1 and AS2 have an agreement 18 00:01:10,000 --> 00:01:12,270 that they only send a certain amount of traffic 19 00:01:12,270 --> 00:01:16,520 load. Over that link in a particular time window. 20 00:01:16,520 --> 00:01:19,810 If the load exceeds that amount, an operator would need to 21 00:01:19,810 --> 00:01:24,040 use BGP to shift traffic from one [INAUDIBLE] link to another.

09 - Interdomain Traffic Engineering Goals  
  
Effective Inter-domain Traffic Engineering has three goals. One is predictability. In other words, it should be possible to predict how traffic flows will change in response to changes in the network configuration. Another goal is to, limit the influence of neighbouring domains. In particular, we'd like to use BGP policies and changes to those policies that. Limit how neighbouring ASes might change their behaviour in response to changes to the PGP configuration that we make in our own network. And finally, we'd like to reduce the overhead of routing changes by achieving our traffic engineering goals with changes to as few IP prefixes as possible. To understand the factors that confound predictability Let's look at how the inter-domain routing choices of a particular autonomous system, can wreak havoc, on predictability. Let's suppose that a downstream neighbor, is trying to reach, the autonomous system at the top of this figure. The AS here might wish to relieve congestion, on a particular peering link. To do so, this AS, might now send traffic to that destination out a different set of atonimous systems. But once this AS makes that change, note that it's choosing a longer AS path. Now taking a path of three half's rather than two. In response, the down stream neighbor might decide not to send its traffic for that destination Through this attonomous system at all. Thus affecting the traffic matrix that this AS sees. So, all the work that went in to optimizing the traffic load balance for this AS is for not, because the change that it made, effectively changed the offered traffic loads and hence the traffic matrix. One way to avoid this type of problem, and achieve predictable traffic flow changes. Is to avoid making changes like this that are globally visible. In particular, note that this change caused a change in the AS path link of the advertisement to this particular destination, from two to three. Thus, other neighbors, such as the downstream neighbor here. Might decide to use an alternate path as a result of that globally visible routing change. By avoiding these types of globally visible changes, we can achieve predicitability. Another way to achieve effective inter-domain traffic engineering is to limit the influence of neighbors. For example, an autonomous system might try to make a path look longer with AS path prepending. If we consider treating paths that have almost the same AS path length as a common group, we can achieve additional flexibility. Additionally, if we enforce a constraint that our neighbors should advertise consistent BGP route advertisements, over multiple appearing links, should multiple appearing links exists? That gives us additional flexibility, to send traffic over different e-egress points to the same autonomous system. Enforcing egress points to the same autonomous system. Enforcing consistent advertisments turns out to be difficult in practice, but it is doable. To reduce the overhead of routing changes. We can group related prefixes. Rather than exploring all combinations of prefixes to move a particular volume of traffic. We can identify routing choices that group routes that have the same AS paths and we can move groups of prefixes according to these groups of prefixes that share an AS path. This allows us to move groups of prefixes by making tweaks to local preference on. Regular expressions on AS path. We can also focus on the small fraction of prefixes that carry the majority of traffic. Ten percent of origin AS is responsible for about 82 percent of outbound traffic, therefore we can achieve significant gains in rebalancing traffic in the network by focusing on the heavy hitters. In summary, to achieve predictability, we effect changes that are not globally visible. To limit the influence of neighbors, we enforce consistent advertisments and limit the influence of a s pass length. And to reduce the overhead of routing changes, we group pre fixed according to those that have common AS paths. And, move traffic, in terms of groups of prefixes.

10 - Multipath Routing  
  
Another way to perform traffic engineering is with Multipath Routing, where an operator can establish multiple paths in advance. This approach applies both to intra-domain routing, and inter-domain routing. The way this is done in intra-domain routing is to set link weights such that multiple paths of equal cost exist between two nodes in the graph. This approach is called equal cost multi path. Thus traffic will be split across paths that have equal costs through the network. A source router might also be able to change the fraction of traffic that's sent along each one of these paths. Sending for example 35% along the top path and 65% along the bottom path. It might even be able to do this based on the level of congestion that's observed along these paths. The way that the router would do this is by having multiple forwarding table entries with different [INAUDIBLE] stops for outgoing packets to the same destination.

11 - Source Router Path Quiz  
  
So quickly, how can a source router adjust paths to a destination when there are multiple paths to the destination? Dropping packets to cause TCB backoff, alternating between multiple forwarding table entries to the same destination, or sending alerts to incoming senders whenever a route changes.

12 - Source Router Path Quiz Answer  
  
A source router can adjust traffic over multiple paths by having multiple forwarding table entries for the same destination, and splitting traffic flows across the multiple next hops, depending on, for example, the hash of the IP packet header.

13 - Data Center Networking  
  
Let's now talk about data center networking and how network operators perform traffic engineering inside a data center. First of all, what characterizes a data center? Data center network has 3 important characteristics. One is multi-tenancy. Multi-tenancy allows a data center provider to advertise the cost of shared infrastructure, but because there are multiple independent users, the infrastructure must also provide some level of security and resource isolation. Data Center network resources are also elastic, meaning that as demand for service fluctuates, the operator can expand and contract these resources. It's also, can be pay per use, meaning that as the need to use more resources arises or disappears, a service provider can adjust how much resources are devoted to the particular service running in the data center. Another characteristic of data center networking is flexible service management, or the ability to move work, or work loads to other locations inside the data center. For example, as load changes for a particular service, an operator may need to provision additional virtual machines, or servers to handle the load for that service or potentially move it to a completely different set of servers inside the infrastructure. This workload movement and virtual machine migration essentially creates the need for traffic engineering solutions inside a data center. A key enabling technology in data center networking is the ability to virtualize servers. This makes it possible to quickly provision, move, and migrate servers and services in response to fluctuations in workload. But while provisioning servers and moving them is relatively easy, we must also develop traffic engineering solutions that allow the network to reconfigure in response to changing workloads and migrating services.

14 - Data Center Networking Challenges  
  
Some of the challenges for data center networking include traffic load balance, support for migrating virtual machines, in response to changing demands. Adjusting server and traffic placement to save power. Provisioning the network, when demands fluctuate, and providing various security guarantees, particularly in scenarios that involve multiple tenants. To understand these challenges, in a bit more detail, let's take a look at a typical data center topology. A topology typically has three layers: an access layer, which connect the servers themselves. An aggregation layer which connects the access layer, and then the core. Historically, the core of the network has been connected with layer three, but increasingly, modern data centers are connected with an entire layer-two topology. A layer-two topology makes it easier to perform migration of services from one part of the topology to another, since these services can stay on the same layer-two network and hence would not need new IP addresses when they moved. It also becomes easier to load balance traffic. On the other hand, a monolithic layer-two topology makes scaling difficult, since now we have tens of thousands of servers on a single flat topology. In other words, layer-two addresses are not topological. So the forwarding tables in these switches can't scale as easily, because they can't take advantage of the natural hierarchy that exists in the topology. Other problems that exist in this type of topology, is that the hierarchy can potentially create single points of failure. And links at the top of the topology, in the core, can be come oversubscribed. Modern data center operators have observed that as you move from the bottom of the hierarchy up towards the core, that the links at the top can carry as much as 200 times as much traffic as the links at the bottom of the hierarchy. So there's a serious capacity mismatch in that the top part of the topology has to carry a whole lot more traffic than the bottom. We'll explore how modern data center network architectures address these various challenges, but let's first take a quick look at one way of solving the scale problem.

15 - Data Center Topologies  
  
Recall that the Scale problem arises because we have tens of thousands of servers on a flat layer two Topology, where all of the servers have a Topology independent MAC or hardware address and thus, in the default case. Every switch in the topology has to store affording table entry for every single MAC address. One solution is to introduce what are called pods and assign sudo MAC addresses to each server corresponding to the pod in which they're location in the Topology. So in addition to having a real MAC address. Each server has what's known as a pseudo-MAC address, as shown in pink. Thus, switches in the Data Centre Topology no longer need to maintain forwarding table entries for every host. They only need to maintain entries for reaching other pods in the Topology. Once a frame answers a pod, the switch then of course has entries for all of the servers inside that pod but they don't need to maintain entries for the MAC addresses for servers outside of each pod. For example, the switch in pod one only needs to maintain entries for these two servers with MAC addresses A and B, but it doesn't need to maintain independent entries with servers with MAC addresses C and D. It only needs to maintain an entry for how to reach pod 2. Likewise for pods 2 and pods 3. Now, in such a Data Centre Topology, of course, these hosts are unmodified. So, their still going to respond to things like ARP queries, with their real MAC addresses, so we need a way of dealing with that, as well as a way of Mapping pseudo MAC addresses to real MAC addresses. The solution is as follows. When a host such as server A issues an ARP query, that query is intercepted by the switch, but instead of flooding that query, the switch intercepts the query and forwards it to an entity called the Fabric Manager. The Fabric Manager then responds with the pseudo-MAC corresponding to that IP address. Host A then sends the frame with the destination pseudo-MAC address and switches in the Topology can forward that frame to the appropriate pod. Corresponding to the pseudo MAC address of the destination server. Once the frame reaches the destination pod, let's say in this case pod 3, the switch at the top of that pod can then Map the pseudo MAC address back to the real MAC address. And the server that receives the frame receives an Ethernet frame with its real destination MAC address, so it knows that the Ethernet frame was intended for it. By intercepting our queries in this way and providing a Mapping between topological pseudo MAC addresses and real, physical MAC addresses, we can achieve hierachial forwarding. In a large Layer 2 Topology, without having to modify any host Software.

16 - Data Center Intradomain Traffic Engineering  
  
In this lesson, we will look at how data center tarffic engineering through customized topologies and special mechanisms for load balance can help reduce linkulization, reduce the number of hops to each the edge of the data center, and make the data center network easier to maintain. We saw in the last lesson how existing data center topologies provide extremely limited server to server capacity because of the over subscription of the links at the top of the hierarchy. Additionally, as services continue to be migrated to different parts of the data center resources can be fragmented, significantly lowering utilization. For example, if the service denoted by green is running mostly in one part of the data center, but there's a little bit running on a virtual machine in another part of the data center, this might cause traffic to traverse. Links of the data center topology hierarchy, thus significantly lowering utilization and cost efficiency, reducing this kind of fragmentation can result in complicated layer two, or layer three, routing reconfiguration, what we'd like to have is just the abstraction is of one large layer to switch, this is the abstraction that VL2 provides. So, VL2 has two main objectives. One is to achieve layer-two semantics across the entire data center topology. This is done with a name-location separation and a resolution service that resembles the fabric manager which we talked about in the last lesson. In which is described in more detail in the paper. To achieve uniform high capacity between the servers and balance load across links in topology, VL2 relies on flow based random traffic interaction using valiant load balancing. Let's take a closer look at how that load balancing works.

17 - Valiant Load Balancing  
  
The goals of Valient load balancing in the VL Two network are to spread traffic evenly across the servers, and to ensure that traffic load is balanced independently of the destinations of the traffic flows. Field two achieves this by inserting an indirection level. Into the switching hierarchy. When a switch at the access layer wants to send traffic to a destination, it first selects a switch at the indirection level to send the traffic at random. This intermediate switch then forwards the traffic to the ultimate destination Depending on the destination MAC address, of the traffic. Subsequent flows might pick different, indirection, points for the traffic, at random. This notion of picking a random indirection point, to balance traffic more evenly, across a topology, actually comes from multi-processor architectures. And has been rediscovered in the context of data centers. So, in this lesson we have explored how valiant load balancing can be used on a slightly modified topology. To achieve better load balance than in traditional fat tree networks, without an interaction layer, and valiant load balancing. In the next lesson we'll look at how a custom random topology can make some of these traffic engineering problems even easier.

18 - Jellyfish Data Center Topology  
  
In this lesson we'll look at Jellyfish, a technique to network data centers randomly. The goals of Jellyfish are to achieve high throughput to support, for example, big data analytics or agile placement of virtual machines and incremental expandability, so that network operators can easily add or replace servers and switches. For example, large companies like Facebook are adding capacity on a daily basis. Commercial products make it easy to expand or provision servers in response to changing traffic load but not the network. Unfortunately, the structure of the data center networks constrains expansion. Structures such as a hypercube require two to the K switches, where K is the number of servers. Even more efficient topologies, like a FAT tree, are still quadratic in the number of servers.

19 - Data Center Topology Quiz  
  
By looking at this figure showing a data center topology where the servers are at the edge of the graph, can you try to figure out where data center topology primarily constrains expansion? Is it at individual servers, aggregation switches or top-level switches?

20 - Data Center Topology Quiz Answer  
  
As we can see from the figure, most of the congestion occurs at the top level. Jellyfishes answer to how data structure constrains expansion is to simply have no structure at all.

21 - Jellyfish Random Regular Graph  
  
Jellyfish's topology is what is called a random regular graph. It's random because each graph is uniformly selected at random from the set of all regular graphs. A regular graph is simply one where each node has the same degree. And a graph in Jellyfish is one where the switches in the topology are the nodes. In contrast to the earlier data center topology diagram we saw, here is a picture of a Jellyfish random graph with 432 servers and 180 switches. Every node in this graph has a fixed degree of 12. Jellyfish's approach is to construct a random graph at the Top of Rack switch layer. Every Top of Rack switch i, has some total number of Ki ports, of which it uses Ri to connect to other Top of Rack switches. The remaining Ki minus Ri ports are used to connect servers. With n racks, the network then supports n times Ki minus Ri servers. And the network is a random regular graph denoted as follows. Formally, random regular graphs are sampled uniformly from the space of all R regular graphs. Achieving such a property is a complex graph theory problem, but there's a simple procedure that produces a sufficiently uniform random graph that empirically have the desired properties.

22 - Constructing a Jellyfish Topology  
  
To construct a jellyfish topology, one can simply take the following steps. First, pick a random switch pair with free ports for which the switch pair are not already neighbors. Next, join them with a link, and repeat this process until no further links can be added. If a switch remains with greater than or equal to two free ports, which might happen during the incremental expansion by adding a new switch, these switches can be incorporated in the topology by removing a uniform random existing link and adding links to that switch. For a particular equipment cost, using identical equipment, the jelly fish topology can achieve increased capacity by supporting twenty five percent more servers. This higher capacity is achieved because the paths through the topology are shorter than they would be in a Fat tree topology. Consider a topology with sixteen servers, twenty switches, and a fixed degree of four for both the fat tree topology and the jellyfish random graph. In the fat tree topology, only four of 16 servers are reachable in less than five hops. In contrast, in the jellyfish random graph, there are 12 servers reachable. By making more servers reachable along shorter paths, jellyfish can increase capacity over a conventional Fat tree topology. So while Jellyfish shows some promise, there are certainly some open questions. First, how close are these random graphs to optimal, in terms of the optimal throughput that could be achieved for a particular set of equipment. Second, what about typologies where switches are heterogeneous with different numbers of ports or link speeds. From a system design perspective, the random topology model could create problems with physically cabling the datacenter network, and there are also questions about how to perform routing or congestion control without the structure of a conventional datacenter network like a fat tree.