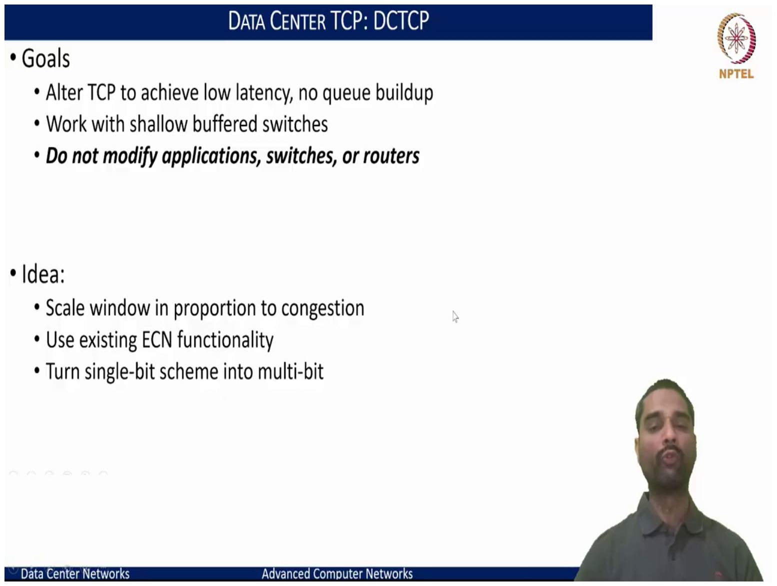
Advance Computer Networks
Instructor Doctor Sameer Kulkarni
Department of Computer Science and Engineering
Indian Institute of Technology Gandhinagar
Lecture 54
Data Centre Networking – Protocol Innovations Part 2

(Refer Slide Time: 00:17)



Data Center TCP: DCTCP

- Goals
  - Alter TCP to achieve low latency, no queue buildup
  - Work with shallow buffered switches
  - ***Do not modify applications, switches, or routers***
- Idea:
  - Scale window in proportion to congestion
  - Use existing ECN functionality
  - Turn single-bit scheme into multi-bit

Data Center Networks Advanced Computer Networks

So, if we revisit and try to see what way we can work on this problem. Our major goal here would be that we want to alter the TCP. So, that we can achieve low latency communication, and also be able to adapt at a very small time scale. But we want to ensure that the queue build up in the first place does not occur so that there is no queue and no congestion that we can lead in. And we want to work with shallow buffered switches. That means we are not expecting the switches to have more buffers where the queues would build up. If the queues build up, we would end up having a lot of latency. And this latency would result in having for us to miss the deadlines.

And hence, we want to ensure that we work with shallow buffered switches. And more importantly, when we think of the workloads, there are a lot of applications that readily use the TCP. So we do not want to alter or modify anything at the application space nor do we want to change significantly any aspects within the switches or routers, because these are the devices that are going to be operating at the L2 or L3 stacks.

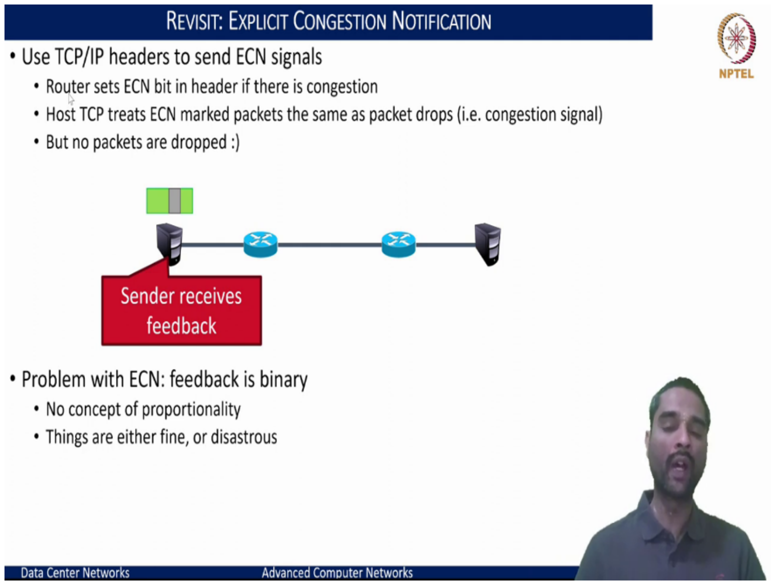
If there are any changes, then that would mean that we will be customizing and requires a lot of efforts and costs in bringing these things up. And so equally important goal is to ensure that there will be no modifications at the application level, at either ends of the host devices that communicate, nor at the network stream in terms of the switches or routers. And the research in this direction led to some of the key ideas where you will want to address this problem.

And one of them being like to scale the window in proportion to the congestion. So, whenever we have a congestion the problem is we react only to the packet loss. And if there is a means to react much early and adapt our mechanism of how we transmit the packets. And even before the congestion actually sets in, we want to react. And that way we could basically mitigate these packet losses or retransmissions that end up taking a lot of time.

And to do this we had seen earlier in the transport layers that we have, we can either use the packet drops using the TCP, which will proactively drop the packets even before the queue gets full or we could use the explicit congestion notification as a scheme where we could instead of dropping the packet, we mark the packet and send them back to the sender so that he can react as if there was a packet loss and lower the rate at which you would transmit the packets in the next round trip time. And this ECN readily helps ensure that we do not even have to experience the packet loss, we could react from the sender side and slow down the number of packets that were sent into the network. And this has been experienced and has been seen to work exceptionally well for the internet case, then where is the concern of applying this in the data centers? And the only concern here is the reaction happens in a round trip time and we are talking of round trip times on internet in several milliseconds orders 200, 300 to 500 milliseconds and more, while in data centers, these would be in the microseconds space.

And this is to our advantage that this round trip times are very short in data center and we could use this to adapt very quickly in a microsecond time scale. But then, with the traditional ECN what we see is it is a single bit that is being added which either slows down the sender or makes the sender to ramp up as you would otherwise do.

(Refer Slide Time: 04:01)

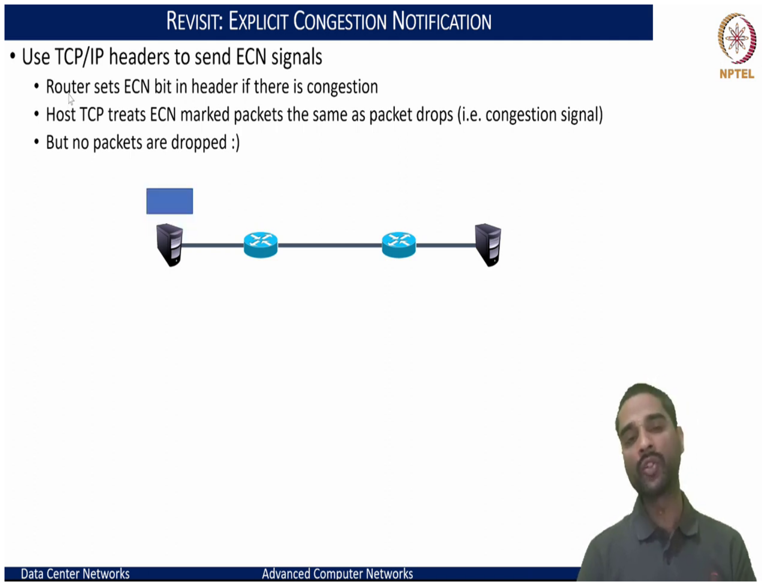


Revisit: Explicit Congestion Notification

- Use TCP/IP headers to send ECN signals
  - Router sets ECN bit in header if there is congestion
  - Host TCP treats ECN marked packets the same as packet drops (i.e. congestion signal)
  - But no packets are dropped :)

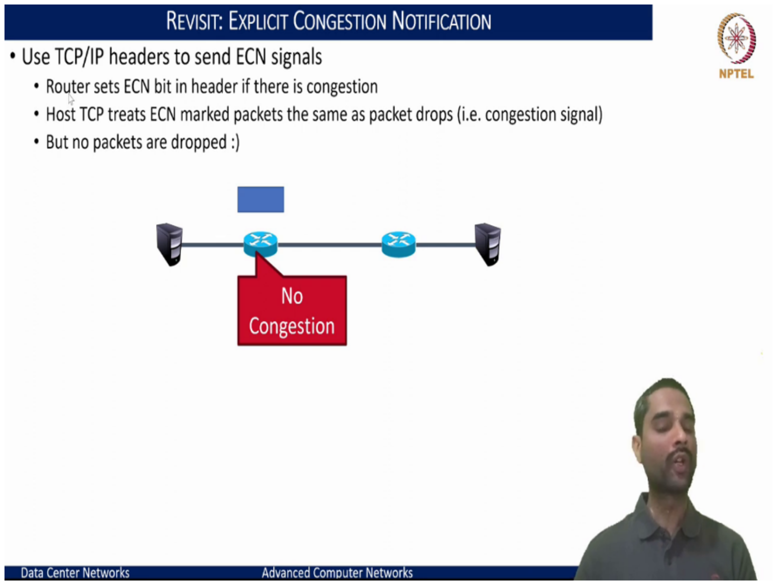
Sender receives feedback

- Problem with ECN: feedback is binary
  - No concept of proportionality
  - Things are either fine, or disastrous



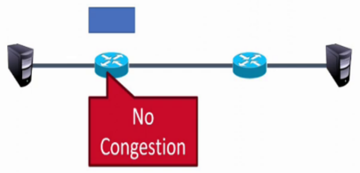
Revisit: Explicit Congestion Notification

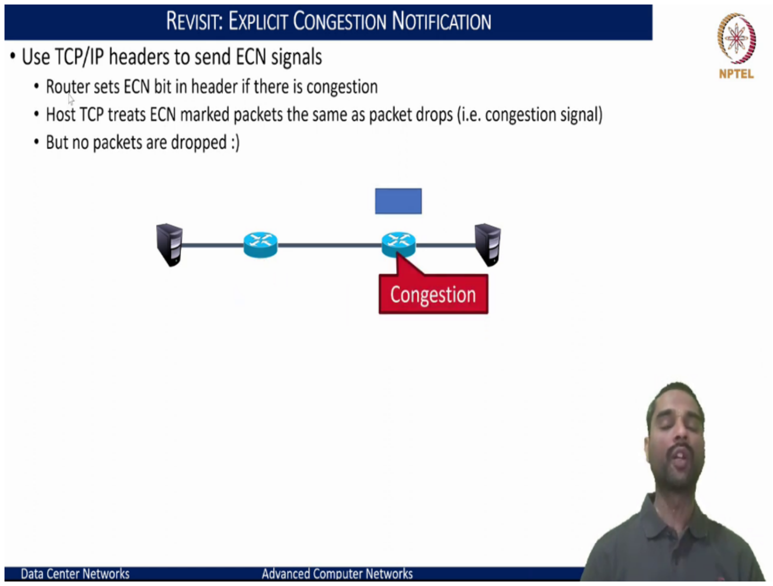
- Use TCP/IP headers to send ECN signals
  - Router sets ECN bit in header if there is congestion
  - Host TCP treats ECN marked packets the same as packet drops (i.e. congestion signal)
  - But no packets are dropped :)



Revisit: Explicit Congestion Notification

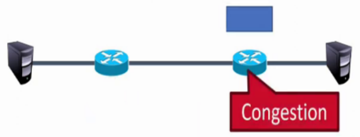
- Use TCP/IP headers to send ECN signals
  - Router sets ECN bit in header if there is congestion
  - Host TCP treats ECN marked packets the same as packet drops (i.e. congestion signal)
  - But no packets are dropped :)

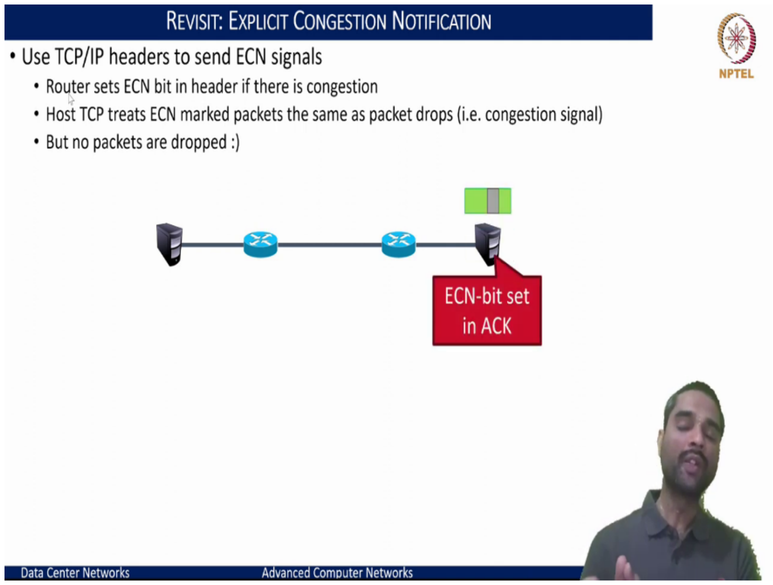




Revisit: Explicit Congestion Notification

- Use TCP/IP headers to send ECN signals
  - Router sets ECN bit in header if there is congestion
  - Host TCP treats ECN marked packets the same as packet drops (i.e. congestion signal)
  - But no packets are dropped :)

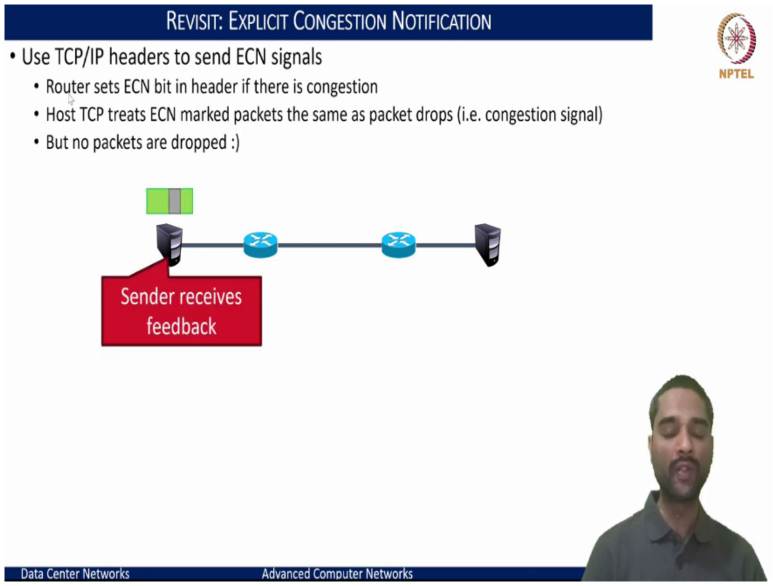




Revisit: Explicit Congestion Notification

- Use TCP/IP headers to send ECN signals
  - Router sets ECN bit in header if there is congestion
  - Host TCP treats ECN marked packets the same as packet drops (i.e. congestion signal)
  - But no packets are dropped :)

Data Center Networks Advanced Computer Networks



Revisit: Explicit Congestion Notification

- Use TCP/IP headers to send ECN signals
  - Router sets ECN bit in header if there is congestion
  - Host TCP treats ECN marked packets the same as packet drops (i.e. congestion signal)
  - But no packets are dropped :)

Data Center Networks Advanced Computer Networks

So, it is necessary now to understand how this ECN works and then see what is the problem in adopting this as it is to the data center networks. So if we think let us consider a very simple topology as the source and destination connected via are two routers and whenever we have these source and destination connects over, they would basically exchange a series of packets from source and destination at which each of the router would see what is its current status and accordingly update the state on the packet that is sent towards the receiver and if there is no congestion, the router would send the packet unaltered without marking any of the bits.

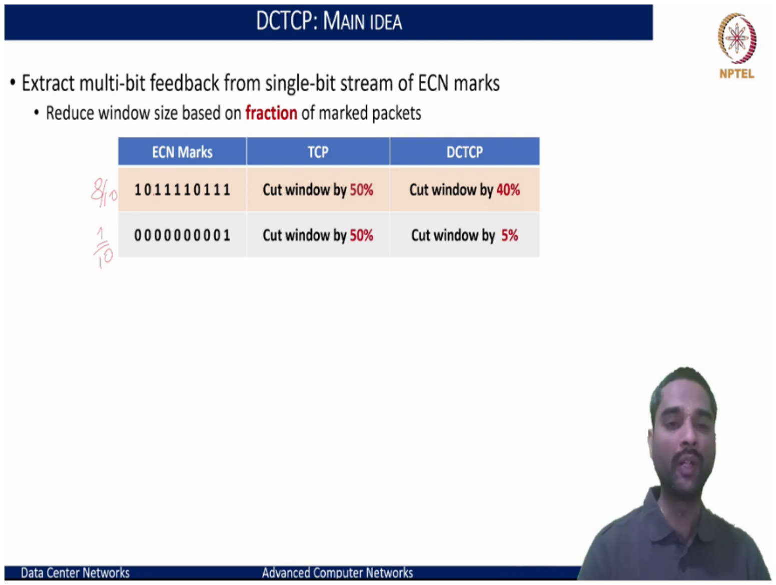
And if there is a congestion at a particular router, then it would mark up ECN bits in the IP header of the packets and then transmit it to the receiver. And once the receiver sees that the packets that it received were marked with the ECN bits, it would basically send this information back in the acknowledgment packet that it would send to the sender. And this once it is received by the sender and sees that there was a ECN flag that was set that means now there was a congestion perceived at the transport layer, the sender would understand that there was a congestion signal, and then he would slow down the packet rate at which he would send the next set of packets in the next RTT. And that is how the host TCP reacts to the congestion signal and ensures that the rate is slowed down when it sends the packets into next RTT.

And thus, without any packet drops, we are able to ensure that the congestion is being worked out from the sender side. But the problem here is that this feedback is completely binary. What that means is if any of the packets that you receive, end up seeing that there is a ECN marked bit, then you react that means over one RTT, even if I receive just 1 packet or 100 packets with ECN marking I react in the same way.

And that means there is no concept of proportionality with respect to the number of packets that were marked ECN. And the way I would react is either that everything is fine. And I would ramp up as usual or I slow down and result in basically making the next round trip to be slow by basically cutting down the congestion window by half.

And this experience would say that now we would end up having a lot of operations in terms of all the transmissions would happen.

(Refer Slide Time: 06:36)

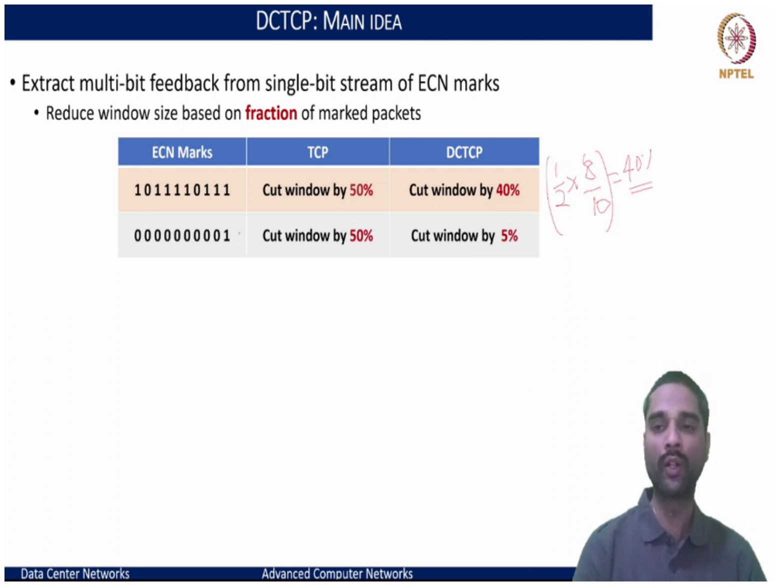


DCTCP: Main Idea

- Extract multi-bit feedback from single-bit stream of ECN marks
  - Reduce window size based on **fraction** of marked packets

	ECN Marks	TCP	DCTCP
$8/10$	1011110111	Cut window by 50%	Cut window by 40%
$\frac{1}{10}$	0000000001	Cut window by 50%	Cut window by 5%

Data Center Networks Advanced Computer Networks



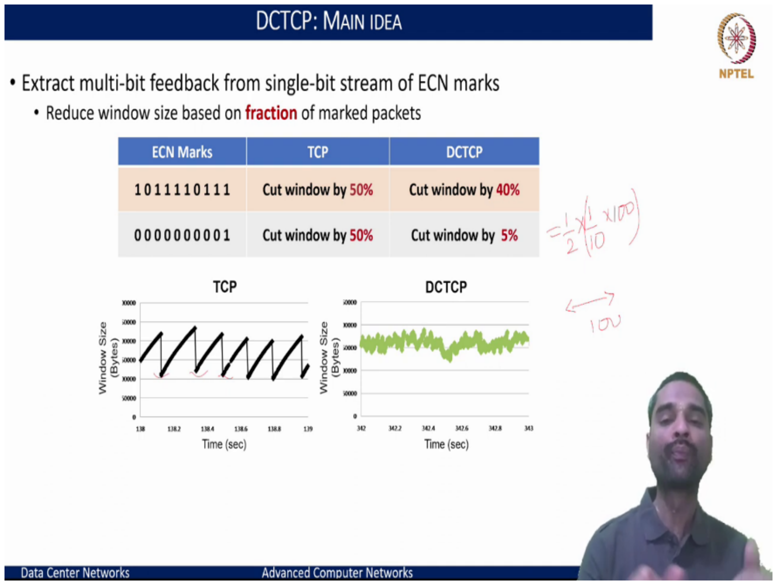
DCTCP: Main Idea

- Extract multi-bit feedback from single-bit stream of ECN marks
  - Reduce window size based on **fraction** of marked packets

ECN Marks	TCP	DCTCP
1011110111	Cut window by 50%	Cut window by 40%
0000000001	Cut window by 50%	Cut window by 5%

$\left(\frac{1}{2} \times \frac{8}{10}\right) = 40\%$

Data Center Networks Advanced Computer Networks



DCTCP: Main Idea

- Extract multi-bit feedback from single-bit stream of ECN marks
  - Reduce window size based on **fraction** of marked packets

ECN Marks	TCP	DCTCP
1011110111	Cut window by 50%	Cut window by 40%
0000000001	Cut window by 50%	Cut window by 5%

$=\frac{1}{2}\times\left(\frac{1}{10}\times 100\right)$

$\longleftrightarrow$ 100

So in DCTCP, what the essense of bringing like how we could adapt this for data centers was brought out to see if we can extract this feedback, which is a single bit ECN mark, and extrapolate that to make it a multi bit information or basically trying to treat several of the single bit information of the ECN marks to accumulate and create a multi-bit feedback mechanism.

And to think of we could do this by saying if I receive n number of packets that are ECN marked over given RTT with n number of packets that were actually sent, then I can use that fraction to say, how much is the congestion really given an instance and how should I react, rather than completely reducing the congestion window by 2, I would reduce the window size based on the fraction of the packets that were marked in a given RTT.

And this way, you will be able to adapt how much you would go in a slow like how much you would have ramp done rather than ramping down all the way to 50 percent. And this was exactly the core idea of DCTCP. To put it in perspective, if we see that there were a bunch of packets that were exchanged, that is 10 of the packets that were sent in a given round trip time. And on the sender side, if it sees that there were 10 packets that it sent, it received basically 8 of the packets were marked with ECN out of these 10, then you would want to react differently, as opposed to saying if there were only just 1 packet out of 10 that were marked with ECN. And if it were the default ECN and TCP that we use, in either of the cases, it would cut down the window by 50 percent. But now our idea is to utilize the single-bit information to make a multi-bit feedback. And we can readily do that by saying my congestion that I see is basically 8 out of 10 packets have experienced ECN, that means the condition is, in fact, bad. But, how should I react to this information, and that we can do in proportion to the number of ECN bits that were being marked.

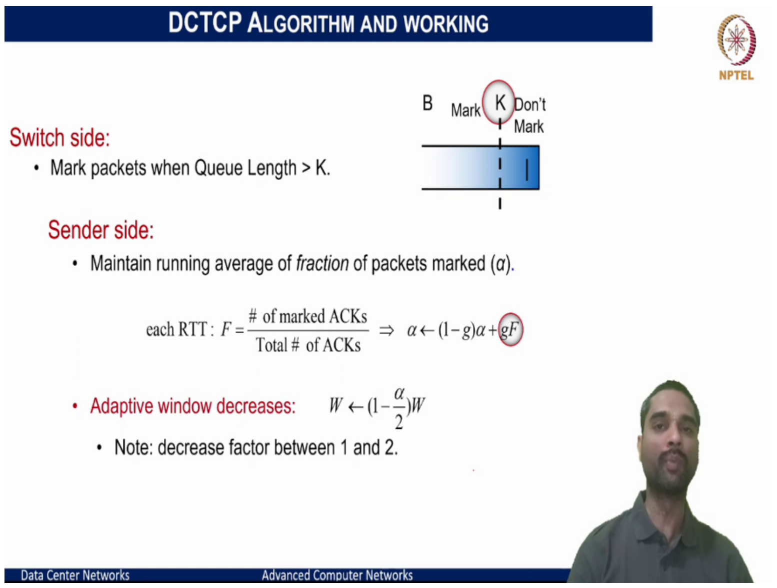
So if I see that I had these ECN marked of 8 over 10, the way I would want to cut my window is basically saying, I will do this with respect to saying instead of cutting the window size by half, what I would now do is basically cut the window size in proportion to the number of packets that I received as ECN marked. So this way, what it would really result in is basically saying that I want to cut the window not by half, but basically by 40 percent. So if I make this in percent, so I basically reduce the window size by 40 percent of my current window so that I slow down and send, but not completely slow down to 50 and cut to half.

And likewise in the lower case when we see that the congestion is really indication of saying if there was a microburst which resulted in only one of the other of the packets to be marked as perceiving the congestion at the router, I could basically see again and do that in proportion to the number of packets that I just transmit and that would be half of the one tenth of the packets that we received. And that would mean that we would cut down the window by just 5 percent and not 50 percent.

Now, if we See how the reaction would look in terms of when we see the window size variations that happen in this TCP versus the DCTCP, we will see that in the regular TCP, you would have a ramp up. And then because of packet loss, you are slowing down, cutting the window by half and then ramping up again and the typical sawtooth that we would see, but now, not that this is happening, because we are always cutting the window by half. And instead in the DCTCP, what we really care about is how much of the ECN packets that we really received. And if we received a very small fraction, then the amount of window that we would ramp down to is much smaller, like around 5 percent.

And within an RTT, if we are sending a lot more packets 100 or more, then you get a better scale in which you are able to like a finer scale at which the window size would also be reduced. And thus DCTCP ensure to not have the issue of having to fall down to a very small and then ramp up again, when you try to mitigate the aspect of how the congestion would be contained and you are able to react. In fact, using the ECN approach helps us react much more quickly.

(Refer Slide Time: 11:12)



DCTCP ALGORITHM AND WORKING

Switch side:

- Mark packets when Queue Length > K.

Sender side:

- Maintain running average of *fraction* of packets marked (α).

$$\text{each RTT}: F = \frac{\#\text{ of marked ACKs}}{\text{Total }\#\text{ of ACKs}} \Rightarrow \alpha \leftarrow (1-g)\alpha + gF$$

- Adaptive window decreases: $W \leftarrow (1-\frac{\alpha}{2})W$
    - Note: decrease factor between 1 and 2.

Data Center Networks Advanced Computer Networks

To put this in terms of an algorithm and see how it really works. On the switch side, it would be a usual ECN switch, where it would just configure the parameter K, which says if any time the queue length increases at the switch beyond this threshold of K, you would mark the packet with ECN and if the queue becomes full, and the only way is to drop the packet.

But the notion here is that once you set this value K any time a packet arrives and it has exceeded the queue length of value K, you would mark that packet and send it forward it to the destined receiver. And if anytime the buffer size is lower than K, then you would not mark, and it would be considered as normal traffic.

And on the sender side, whenever you would receive the packets that are going to be marked with K, you would keep track of how many packets within a given RTT were sent. And what is the fraction of the packets the fraction F here to say how many of the packets that he sent and received in this RTT were marked with ECN flag.

And if there is any mark of that ECN packets that he received with the ECN flag being marked, then we would want to react and adapt the window in proportion to the number of packets that we would have received. So, the receiver echoes the actual ECN bits back to the sender. And the sender accumulates all the packets with the ECN bits to estimate first, within each RTT what is the fraction of packets that he received and use this fraction of packets as an information to say how much we should be adapting the window or lower the window. And if there is no congestion, no packets that are being marked, we can see that the fraction α would always basically be 0. And what that means is we would always honour whatever the traditional TCP in terms of ramping up.

As alpha becomes 0, you are only going to be using g times of the aspect to say it is a traditional TCP approach of increasing the congestion window if you are in a slow start phase or if you are in a condition avoidance phase, it would again accordingly increase the window size over the RTT.

But, if there is any time we see that there is a packet that is being received with ECN flag, then we would want to adapt the congestion window. And this is where we use this running average that is the α variable, which indicates by how much should we take into consideration the current aspect because many a times when you transmit the packets, the packets that you sent in one RTT may overlap with the packets that you receive in other RTT.

So we do not want to just make the decision based on the current RTT afresh, we want to keep the information in a history and do that using the weighted average approach that we typically follow in TCP. And the weight g can be defined to say how much of the history we want to remember and how much of it that we want to react to the current instance. So if we say that g is a very large value, then we are reacting more impulsively to the information that we receive in our current RTT window.

And if g happens to be a small value, then we also take into account in equal essense 1 - g the factor would pronounce in terms of what is the running average that we are having and take into account the history or hysteresis of what is gone in with respect to the adaptation and account the current window and past windows information in deducing what should be the window decrease.

And this is a very important characteristic because, whenever we said that there will be information that we want from the workers to be sent out, they should be able to react based on what was observed earlier rather than what would be observed now, and in that essense can be brought in by maintaining this g and arbitrarily good value of 0.4 or 0.5 would ensure that you are able to keep the history information and then accordingly signed.

And if we see that, a very large history can even tell us to say that we should slow down even before sending the packets upfront. And that is more desirable when we saw and want to understand how this incast can be addressed in a correct fashion. And this is a very simple algorithm, very simple mechanism and we can see that there is no as such any tunings that are required, specifically for the switches, because the switches were already capable of having the RED or RED random early drops, as well as ECN marking. So the same can be leveraged as it is at the switches. And very little has to be done on the source side, because source is now instead of trying to react quickly, it is trying to accumulate over the RTT and then react.

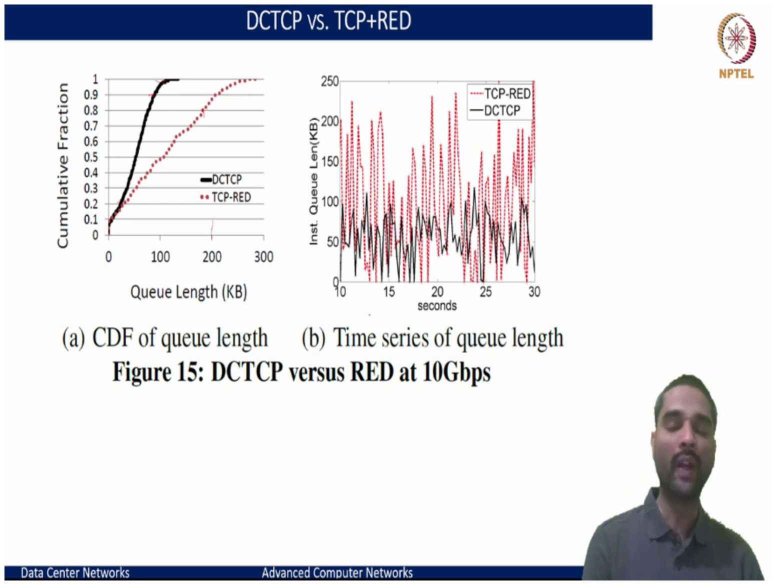
And this is a change that would happen at the transport layer, like within the TCP protocol. So the applications would again be agnostic to the kind of change that you would bring in at this TCP layer. So, these changes could be readily adapted for the TCP in the transport stack of the source and destination sites. And it is only the source that will maintain this information so the only new parameters that this information adds is maintaining a running average of the number of packets that are being perceived as a multibit information, and then change in how we react to the adaptive window. And note, if this α is 0, then there is no change whatsoever to the traditional TCP approach.

And if we do not want to react, you want to react to the ECN as in the earlier indication, then we can have g as 0, that means you are always trying to react with respect to whatever is the current information that you see. And you can again make it as a one-bit information. And this decrease factor that we are trying to bring is always like less than half. And that is where this reduction of window would be like w to w/2 any range in this, we can get in a small time fine grained steps, which will depend based on the fraction of the packets that we want to use. And the decrease factor that will want to set.

Thus, this fine grained adaptation of reducing the congestion window in a very final scales time just ramping down by half ensures that you have smooth variations for the congestion window and this smooth RTT variations that ensure that we are able to operate with shallow buffers in a much more final sense than reacting in a manner of TCP.

(Refer Slide Time: 17:58)

DCTCP vs. TCP+RED



(a) CDF of queue length (b) Time series of queue length

Figure 15: DCTCP versus RED at 10Gbps

Data Center Networks Advanced Computer Networks

And, to put this in perspective and try to understand what really happens when we try to compare the DCTCP with the traditional TCP and RED approach. The paper shows several of the evaluations but I am trying to put in the key aspects that will help us understand and distinguish the benefits that we see from DCTCP.

And if we observe like that a Broadcom router where they would observe basically what is the queue occupancy for a given set of packets that were transmitted. So, if we had one flow that is run over a long enough instants of time just to show the benefits that this DCTCP would bring and try to compare that with the TCP RED variant. And we can see here from the figure on the left, for all the dotted lines here correspond to how the queue length was perceived five different fraction of the packets as they were transmitted out of that switch. And here, you can see that for almost the 30% of the packets had observed around nearly 100 Kilobytes of the queue length that is being set, that means if you have considered 1 Kilobyte packets, there are already a queue size that is built up of about 100 packets, before they could even be transmitted.

And only for very small 0.1 to 0.2 is where the queue build is much smaller, right less than 10 packets. But as we go further, if we consider around 80% of the packet, you will see that there were more than around 200 packets at this point with 200 KB of the queue length that is being utilized.

And if you go further, more than 90% of the packets experienced to see that they were already queued up at well above 200 KB that means the packet will only be delivered after 200 KB of the queue is drained out. That means it would be the 200+ packet to be sent out from that link. And this essentially results if you had a very small RTO you would end up doing lots of retransmissions and if you had a large enough RTO you would end up missing the deadlines and ensure the packets would reach in a stale fashion.

And as opposed to what we see with the DCTCP approach where the packets are now being sent and if you see the cumulative fraction of the packets that experienced what is the queue length as they go, you can see that up till 20 percent, you both experience the same amount of packets, there is no change. But beyond that, when the K factor that is being configured kicks in, you would start to see that the reactions are much more quicker.

And every packet that is being transmitted for a given flow as it goes, it will not observe the queue to build up drastically at the switch, and all of the even at the 90%or 99 percentile of the time, we are observing that the queue build up is less than 100. That means the amount of time that it would need to drain the queue out and have the retransmissions done, you are reducing the chances to retransmit the packets because you are trying to meet the deadline and most of the packets will then whatever would have been sent would be delivered within the deadline.

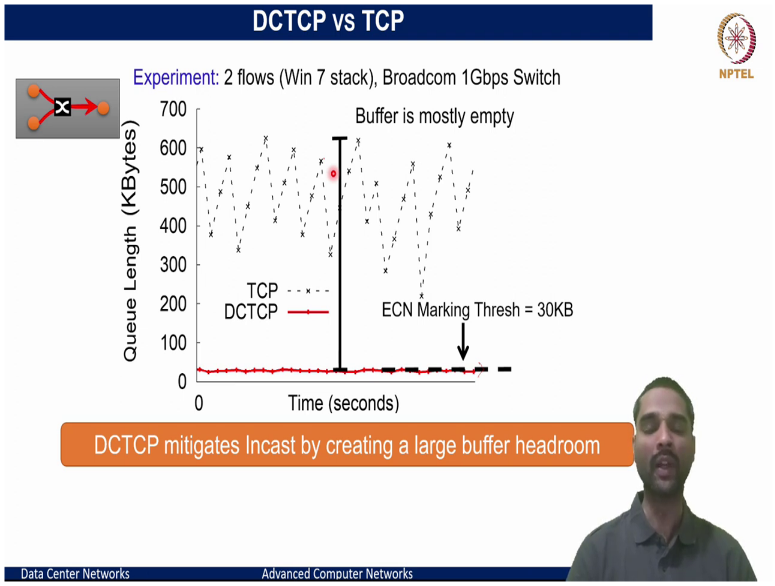
And this measure of the instantaneous queue length also, we can see in the case of the TCP with RED, you can see a lot of aberrations. And that is because whenever you see there is a loss, you would wait for the sender to slow down and then send, that would happen over RTT time and the RTT is being very small, you would see that the sender would not react in a very quick fashion and it would ramp down pretty quickly making the packet instantaneous queue length also to go down much more rapidly.

While with DCTCP, what you would see is we are not cutting down the packets in the same proportion, we are not cutting down the window size by 2 rather we are cutting down in the proportions of how the congestion is observed. So, it would follow not as much drastic drop because even if we have to operate in a coarse and fine timescale, our adaptation is much more controlled.

And in that way, you will see that the queue length on average is maintained between 50 and 100. And rarely it will drop in cases where there is only one of the flows maybe where it experienced and it has a slow start phase and it ran down to the 0.

So, in a nutshell, what we see here now, with this queue length visualization of how this has achieved, we can see that the DCTCP flows would experience less of a congestion as they are trying to react much more quickly to the congestion and ensure that the queue build up would not go beyond the certain size where it would end up in having the delayed or make the flows to miss the deadline.

(Refer Slide Time: 22:37)


DCTCP vs TCP

Experiment: 2 flows (Win 7 stack), Broadcom 1Gbps Switch

DCTCP mitigates Incast by creating a large buffer headroom

Data Center Networks Advanced Computer Networks

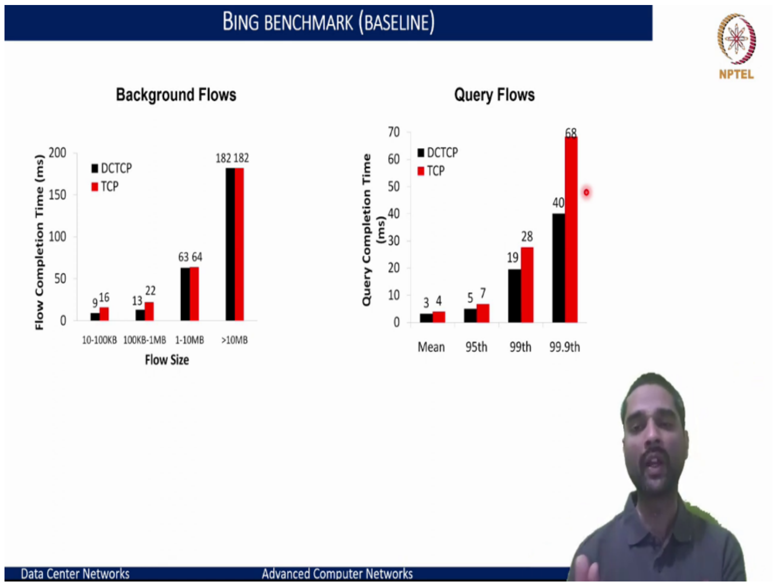
So if we consider this experiment in a much more closer view, where there were two flows that were run on a Windows 7 networking stack with a Broadcom 1 Gbps switch, and you will see the queue lengths that would be maintained at the DCTCP over a course of time as opposed to the queue length that will be maintained by the traditional TCP, you will see that the DCTCP is maintaining almost a very small value of queue length wherein you can see that throughout the DCTCP was able to maintain the queue length at a switch at around roughly 30 kilobytes were very small and aberrations around this 30 Kilobyte range. While this typically means roughly around 30 packets of 1 Kilobyte that would be basically having to transmit the packets at the switch.

While with TCP, you can see that the queue length measured goes all the way up to 600 Kilobytes, and it might drop all the way to less than around 200 Kilobytes that inferring that with the ECN approach, we are trying to keep the buffering consistent and trying to keep the buffering smooth at the switch without having to add any additional delay. And in a way, you are trying to keep the buffers empty to the extent that you are not overflowing anytime and not conceiving any loss.

And that is the most important aspect that we need to address or mitigate the incast problem. And this we can see that DCTCP helps mitigate the incast problem by ensuring that we are creating a large headroom to accommodate the bursts. So, we are now saying that we can accommodate a burst of packet but not an average queue that exceeds for a larger duration of time to take more packets.

And that is the most important characteristic because the main intention for us was to ensure that we can absorb the microbursts without having to have the real losses, and this is exactly where the DCTCP heads.

(Refer Slide Time: 24:42)



And if we compare now from an end-to-end application point of view, how this really results in terms of the flow completion times, as we vary the flow sizes and if we see that DCTCP is more applicable in the aspect of when we have very small flows or the mice flows and we want it to give the advantage to the mice flows in essence of trying to complete without having to experience much of losses.

And we can see that for the 10 to 100 KB flows that we are seeing here, the flow completion times are almost half as opposed to what you would see in the regular TCP version. That is basically 9 milliseconds is what the DCTCP took for completion of 10 to 100 kilobyte flow as opposed to TCP taking around 16 milliseconds.

And as we increase the flow sizes making mice to move towards the elephant flows, we see that there is no adverse effect either like when we increase the flow sizes from 100 KB to 1 MB range, we are still getting the better benefit of DCTCP in terms of having better flow completion time, but although in less proportion, as opposed to what we saw with the 10 to 100 KB flows.

And as we increase further to 1 to 10 MB, that is we are entering into the elephant domain, where all the flows are now large flows, we will see that the TCP and DCTCP will start to behave the same. And that is understandable in essence that now, we are keeping just the sufficient amount of packets in the queue neither more nor less, but the links are always going to be occupied in terms of delivering the packet.

So, there is nowhere where the link would start off the packets and it would have just sufficient number of packets in the link to send the packets. And likewise, you would see that when the flow size increases further, TCP and DCTCP would almost behave the same way. So the takeaway here is with TCP, what would happen because you react by cutting the congestion window by half, you are basically making the queue at the switch to start and not accommodate the microbursts.

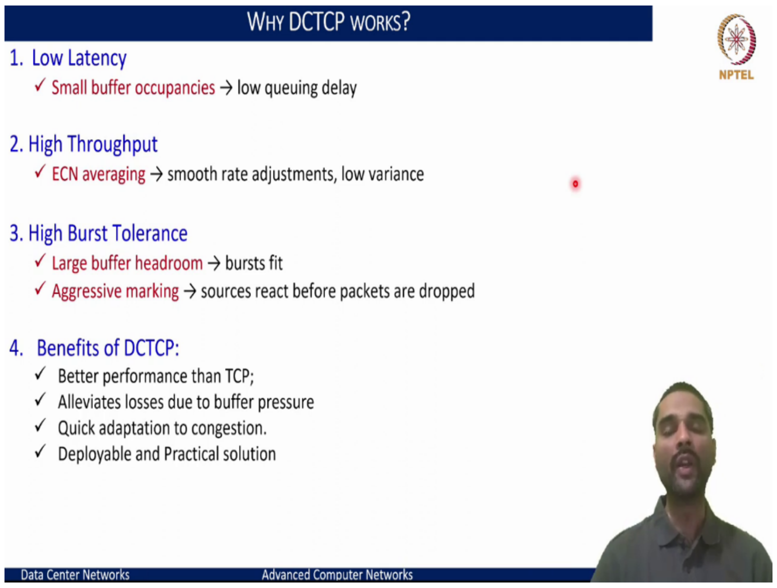
And with DCTCP, on the other hand, what we have ensured is that the buffer occupancy at a queue is contained and it will be sustained at a value K that you would configure. And by doing that, you have built the headroom to accommodate microbursts at any given time, where the packets can still be accommodated within the queue and only if the average queue length increases for a duration then they will be marked and you would whenever there is a marking you are not reacting instantaneously either to cut down the window by half. Instead, we are jumping in proportion to the number of packets that really absorb, that means, if there were a microburst of packets, you would experience 1 out of 10 or 1 out of 4 or 5 packets to say that we would react accordingly.

And thus, you are able to ensure that we are not having any overheads at the queue in terms of overflowing the buffers, providing them a space to accommodate the microbursts at the sender side, you are able to react in a proactive manner and send just the right amount of packets that a queue would handle. And thus, this DCTCP is able to ensure that provides the better benefits, especially with respect to mice flows.

And on the right-hand side here, what we are trying to see is the query completion times which is an end-to-end applications where you are trying to make the queries and see how they really behave. And we can see that these queries typically include basically lots of short flows and the effect is in essence, when you look at the higher percentile of the queries that complete within a given time.

And you can see that most of the queries with the DCTCP 99.9% of the queries were able to be completed within 40 milliseconds while with TCP, the numbers jumped up to 68 milliseconds and thus, we are able to contain the time and meet the deadlines in a soft approach using this DCTCP as a protocol.

(Refer Slide Time: 28:48)



Why DCTCP works?

1. Low Latency
   - ✓ Small buffer occupancies → low queuing delay
2. High Throughput
   - ✓ ECN averaging → smooth rate adjustments, low variance
3. High Burst Tolerance
   - ✓ Large buffer headroom → bursts fit
   - ✓ Aggressive marking → sources react before packets are dropped
4. Benefits of DCTCP:
   - ✓ Better performance than TCP;
   - ✓ Alleviates losses due to buffer pressure
   - ✓ Quick adaptation to congestion.
   - ✓ Deployable and Practical solution

So, to summarize, the key benefits of TCP is better performance in terms of low latency and high throughput. And it alleviates the losses that can happen due to the buffer pressure at a switch because we are now able to contain several of the microbursts. And it also provides a means to quickly adapt to the congestion for the end hosts in terms of how to react and react in a much more controlled fashion than in aggressive fashion of reducing the congestion window.

And most important is that because it does not require any additional changes at any of the network elements, nor at the application layer, but just the TCP stack that we can build in the Windows or Linux network stacks, this becomes readily deployable and a very practical solution in addressing the TCP incast. But I do want to question that this DCTCP is a better approach of handling the incast but in essence, this does not solve the incast problem as it is.

And if we have to really address the incast problem, what really requires wherever we thought asked about the switches that are having the buffer sharing, we want complete isolation of the large elephant flows and the small mice flows. And this is something that is only possible if we address at the scheduling of the packets and marking of the packets as either elephant and mice and accordingly handle the packets, which is not the intent here that the DCTCP does. But it does in generalized way to accommodate microbursts.

So there were a lot of other works that came after this in terms of what were the shortcomings in this DCTCP and what are the means to do this by using better packet scheduling approaches to work around the TCP incast. But for this context, we will stick to this DCTCP and you may follow several of these works that came including some of the work from Google and Facebook timely as one of the papers that came in 2017 SIGCOMM and DCQCN approach was another Microsoft approach. In fact, which will try to improvise on this approach of DCTCP as a means to address the congestion problems. So, there are several other works interesting works to look at.

But for the scope, we will contain here and hope we have gotten a good understanding of how this DCTCP works and why there was a need for a variant of TCP to be set up for the Data Centers.