


Advanced Computer Networks
Professor Doctor Sameer Kulkarni
Department of Computer Science Engineering
Indian Institute of Technology, Gandhinagar
Lecture 53
Data Center Networking - Protocol Innovations - Part 1

(Refer Slide Time: 0:17)


DATA CENTER NETWORKS: PROTOCOL INNOVATIONS



- **link layer:**
 - RoCE: remote DMA (RDMA) over Converged Ethernet, RoCEv2, DCQCN

- **transport layer:**
 - ECN (explicit congestion notification) used in transport-layer congestion control (DCTCP)
 - experimentation with hop-by-hop (backpressure) congestion control
 - [RFC 8257 - TCP Congestion Control for Data Centers](#)
 - QUIC – an Application layer transport?

- **routing, management:**
 - SDN widely used within/among organizations' datacenters
 - place related services, data as close as possible (e.g., in same rack or nearby rack) to minimize tier-2, tier-1 communication



Data Center NetworksAdvanced Computer Networks

Let us look at some of the key protocol innovations that happened due to the data center networks. And when we speak of the link layer at the lowest of the layers there was an emergence of RoCE or what we call as the remote DMA over Converged Ethernet.

And this is essentially a network protocol that leverages the RDMA capabilities to accelerate communication between the applications that could be hosted on clusters of servers or including the storage arrays and whenever there is a need to copy the data from the application space from one device onto the other within a data center, we could do that without expanding the whole CPU by directly configuring the aspects within the network interface card to ensure that this communication can be done.

And this is done through what we use as an infiniband transmission association semantics for RDMA, wherein the direct memory-to-memory transfer at the application level can happen without involving any copies from the host CPU. And coming to the transport layer, there were several of the extensions that did happen and several of the protocols that were being targeted, specifically for data center workloads.

And one of them is being the TCP itself that had modified to meet the data center characteristics, and that is what we know as the data center transport control protocol. And here it reworks on how the ECN is being used within the data centers, unlike the way the ECN gets used in the traditional networks. So we will learn about this in a bit and we can cover this DCTCP as a protocol that how it led to the acceleration of addressing several of the data center congestion works within the data center.

And next is also like there were several of the experimentations that happened with hop-by-hop backpressure for congestion control as a variant means through which we could control and mitigate the condition within the microsecond granularities, and these works also converged into what we see as an IDF RFC 8257, which essentially is the initial work that came out DCTCP in the early 2010s, got standardized in terms of how the DCTCP should be put and that came out as an RFC 257 around 2017.

And also with HTTP workloads that were being dominant and in terms of how the connections for the page loads that we wanted to improvise, there was an emergence of what we see as a QUIC, as an alternative protocol, which basically is both an interplay of transport and application layers. And application builds on top of this, hence it is not necessarily an application, but it also builds on top of an existing UDP to facilitate a newer transport layer, so we can look at this QUIC in bits later.

Besides in terms of routing and management aspects, there were also many of the challenges when we had to deal with lot of scales of these devices to handle. And that is where we saw the emergence of SDN could readily plugin and help, how the data centers could be managed and configured on a centralized fashion. And also in terms of the topologies that we saw, there were a lot of aspects that also follow in terms of how we manage the racks and how you can even offload management of these racks to different tenants when you lease them.

So the communications in themselves like how the services could be moved and what are the means where the interacting services could be placed close to each other and how to build such an infrastructure all of these innovations, in terms of how to automate and make sure that things can be done in a easier fashion, we will solve it to several of the protocol innovations in network automation.

So let us try to focus, and understand what it means with respect to the data center TCP and how is it different from the traditional TCP that we learned in our networks.

(Refer Slide Time: 4:17)

Data Center TCP (DCTCP)

Mohammad Alzadeh¹, Albert Greenberg², David A. Maltz¹, Jitendra Padhye¹,
Parveen Patel¹, Balaji Prabhakar¹, Sudipta Sengupta¹, Murari Sridharan¹

¹Microsoft Research ²Stanford University
(albert, dmaltz, padhye, parveen, sudipta, murari)@microsoft.com
(alzadeh, balaji)@stanford.edu

DCTCP – SIGCOMM 2010

ABSTRACT

Cloud data centers host diverse applications, mixing workloads that require small predictable latency with others requiring large sustained throughput. In this environment, today's state-of-the-art TCP protocol falls short. We present measurements of a 6000 server production cloud and several requirements that lead to high application latencies, needed in TCP's demands on the limited buffer space available in data center switches. For example, head-of-line binary "background" flows build up queues at the switches, and this impacts the performance of latency sensitive "foreground" traffic.

To address these problems, we propose DCTCP, a TCP that is optimized for data center networks. DCTCP leverages Explicit Congestion Notification (ECN) in the network to provide multi-bit feedback to congestion. We evaluate DCTCP at 10 and 100 Gbps using commodity, shallow buffered switches. We find DCTCP delivers the same or better throughput than TCP while using 50% less buffer space. Unlike TCP, DCTCP also provides high-bandwidth and low latency for short flows. In handling workloads derived from operational measurements, we find DCTCP enables the applications to handle 10X the current background traffic, without impacting foreground traffic. Further, a 10% increase in foreground traffic does not cause any timeouts, thus largely eliminating access problems.

Categories and Subject Descriptors: C.2.2 [Computer-Communication Networks]: Network Protocols

General Terms: Measurement, Performance

Keywords: Data center network, ECN, TCP

1. INTRODUCTION

In recent years, data centers have transformed computing, with large scale consolidation of compute IT into data center hubs.

IEETF RFC 8257

2017

entire recent research proposals envision creating economical, easy-to-manage data centers using novel architectures built atop these commodity switches [12, 13].

In this vision, switches handle the traffic of real data center applications. In this paper, we focus on well-tuned applications, supporting web search, email, advertising, and recommendation systems that have diverse needs for data center connections. These applications generate a diverse mix of short and long flows, and require three things from the data center network: **low latency for short flows**, **high bandwidth**, and **high utilization for long flows**.

The first two requirements stem from the **Performance Aggregate** identified in [23], a workload pattern that many of these applications use. The near real-time analytics for end users transfer into latency targets for the individual tasks in the workload. These **latency targets** must be met, and tasks are completed **before their deadline** are cancelled, affecting the final result. Thus, application requirements for low latency directly impact the quality of the real-time result and user experience. Reducing network latency allows application developers to insert more cycles in the algorithms that require relevance and real-time experience.

The third requirement, **high utilization for long flows**, stems from the need to continuously update internal data structures of these applications, as the freshness of the data also affects the quality of the results. Thus, high throughput for these long flows is as essential as low latency and head-of-line.

In this paper, we make two major contributions. First, we **measure and analyze production traffic** (>15TB of compressed data), collected over the course of a month from >6000 servers (12), extracting application patterns and needs (in particular, low latency needs), from data centers whose network is comprised of commodity switches. Implications that their performance are identified and

Data Center Networks

we will see emergence in using computing services providers that Amazon, Microsoft and Google, all competing for the data center market.

Source: the proposed Data Center TCP (DCTCP), which addresses

TRANSPORT ON THE INTERNET

• TCP is optimized for the WAN

- Fairness:
 - Slow-start
 - AIMD convergence
- Defense against network failures:
 - Three-way handshake
 - Reordering
- Zero knowledge congestion control:
 - Self-induces congestion
 - Loss always equals congestion
- Delay tolerance:
 - Ethernet, fiber, Wi-Fi, cellular, satellite, etc.

Additive Increase:
 $W \rightarrow W+1$ per round-trip time

Multiplicative Decrease:
 $W \rightarrow W/2$ per drop or ECN mark

Data Center Networks

Advanced Computer Networks

And this is a SIGCOMM 2010 paper by Microsoft Research at Stanford University that came and said what aspects of TCP were the shortcomings that it would not work for data centers and what aspects would need to be changed so that the TCP could be adapted for Data Center workloads.

And this is a very nice paper, so I would encourage you to go read it. But we will try to summarize in the next half an hour or so about this paper and what contributions it really brought about. And to fill ourselves in the shoes of the networks, first let us try to recap what the transport on the internet really meant.

And here essentially, the TCP as a protocol, how it was built for the Internet infrastructure, and why many of the workloads that we see over the Internet, like almost 90 percent of the traffic or 80 to 90 percent, is often the TCP and that is because of its strengths in terms of how it is able to adapt to the internet bandwidth, how it is able to adapt to the latency that we see the variable variations in the latency. In terms of how it ensures connection-orientedness and reliability for providing the connection, and most important aspect when multiple of the users are trying to connect, is also the fairness in terms of ensuring that all the users get equal fair share of the resources within the internet.

And this was done primarily through two of the important phase aspects, one is a slow start where you would ramp up to see what is the capacity that you could reach. And to do that, you would basically do additive increase, and then once you have reached the capacity and you start to perceive the losses you want to do the multiplicative decreases so that you are able to quickly drop down and utilize the remaining network capacity and allow the other users to also utilize the capacity in the meantime.

And the robustness came with respect to having a three-way handshake that ensured that there is a mechanism to ensure that the two ends that wanted to communicate on up and running. And also the mechanisms like reordering that ensured that even if the networks were to send the packets out of order, the transport layer would ensure that the packets would be reordered before delivering it to the application ensuring in-order packet delivery.

And the congestion control which was basically just the window-based or which was triggered based on the losses, and then it would do this AIMD approach what we just discussed. So all of these aspects made TCP a very popular choice and a default choice for most of the communications, for most of the applications to use over the internet.

And also there were aspects that brought about in wireless, in satellite networks in terms of how you manage the delays and you could have the parameters to tune in terms of what should be the

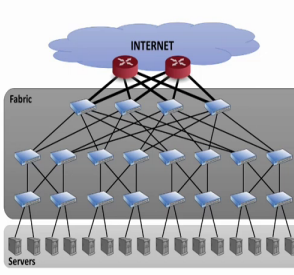
time scales on which you would want to timeout, what should be the time scales on which you would want to resend or re-transmit the packets, all of these could be adapted and hence, the TCP worked seamlessly for the cases of the internet.

But now when we looked at the internet and the data center networks, we saw there were specific characteristics where the data centers differ from the internet.

(Refer Slide Time: 7:35)

DATACENTER IS NOT THE INTERNET

- The good:
 - Possibility to make unilateral changes
 - Homogeneous hardware/software
 - Single administrative domain
 - Low error rates
- The bad:
 - Latencies are very small (250μs)
 - Agility is key!
 - Little statistical multiplexing
 - One long flow may dominate a path
 - Cheap switches have queuing issues
 - Incast



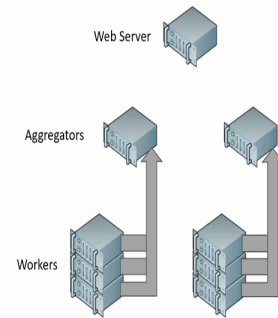
How should we design the transport for data centers?

NPTEL

Data Center Networks Advanced Computer Networks

PARTITION/AGGREGATE PATTERN

- Common pattern for web applications
 - Search
 - E-mail
- Responses are under a deadline
 - ~250ms



NPTEL

Data Center Networks Advanced Computer Networks

And that is essentially in terms of when we have this large amount of the network fabric that is woven within local control, the amount of latency that we are speaking of when we want to have

the connection within such a data center; this is always within few microseconds, like typically within tens to at best hundreds of microseconds.

And now the aspect that we want to see, TCP brings in reliability, and TCP can work well for ensuring reliable communications even accounting for any of the link failures or whatever, but now how does it do so with respect to the latency requirements that we are speaking about, that becomes an important aspect to consider.

Also like when we have such a data center and we are speaking of the traffic that is both East-West and North-South, which have different latency requirements and different bandwidth requirements, and if they have to be using the same set of links, they may be sharing the resources as switches and links as they flow along. And if these switches were to have statistical multiplexing of the buffers that they have for all of their links through which they transmit the packets, then we may end up also seeing that most of the East-West traffic, which is basically the throughput-oriented large flows like considering the workload of database updates that are happening or migration of the VMS that are happening versus like small queries like a Google search or Bing search that we discussed earlier happening on the North-South traffic.

So then even they intermix, there may be a spike in the latency where the queries may even often be dropped when we spoke about the deadline-driven aspects. It may be becoming hard to meet the deadlines. So what these data centers started to see is what we call as an Incast problem. So we will try to understand what this Incast problem means and how this can be overcome through means of adapting the TCP.

So the question in essence is how should we be designing the transport for data centers? would the data centers be right like using just the traditional TCP be right for data centers or what is the problem that it brings along?

And in this aspect we also discussed about the key kind of North-South traffic that constitutes in the data centers, and that is primarily either you are having the email service for a search that you are trying to do using basic web applications, where it involves basically the partition and aggregation pattern that we looked earlier wherein, anytime a request comes to a web server, the web server would split this request and communicate with multiple of the aggregators, maybe one to ten of the aggregators, and then each of these aggregators would in turn talk to the worker

nodes and split the workload to ensure that all the workers fetch or get the data and then send the information back to the aggregators and aggregators then send the information back to the web server, which would collate all the information and respond back to the user.

And typically like, when we use the internet and we expect to do a search on Google or Bing, we expect the data to be available within seconds of time, less than a second. And if we say a critical deadline of around 250 milliseconds, then we are speaking of having each of these servers to cooperate within such a deadline.

And that means all the connection and communication that happens from the web server to the aggregators and to the workers and the information that flows back from workers to the aggregators and to the web server have to happen in a much smaller and tighter deadline than what we see as an end-to-end deadline so that the response can meet the deadlines.

(Refer Slide Time: 11:20)

TCP INCAST

- Incast is a many-to-one communication pattern.
 - Parent server places a request for data to a cluster of Nodes which all receive the request simultaneously.
 - Cluster of nodes in turn respond to the singular Parent.
 - The result is a micro-burst of many machines simultaneously sending TCP data streams to one machine (many to one).
- “TCP Incast” refers to the detrimental effect on TCP throughput caused by network congestion.
 - Simultaneous many-to-one burst leads egress congestion at the network port attached to the Parent server.
 - Resulting packet loss requires nodes to detect missing ACKs, re-send data (after RTO), and slowly ramp up throughput as per standard TCP behavior.

Parent solicits Nodes
Simultaneous Node Response
Congestion at Parent port
TCP throughput back off
Application performance suffers

Switch

Nodes

Hadoop, Map Reduce, HDFS, GFS

Aggregator /Parent

Workers

Data Center NetworksAdvanced Computer Networks

And this often results to what we call as an incast problem wherein when we see what is happening is when an aggregator tries to request several of the workers, we are expecting all the data from the workers to reach back to this aggregator in a stipulated time, and we saw earlier that this could be within tens of milliseconds.

And what in this what happens is whenever the workers have to reply back to the aggregator more than often they would finish their job and send the updates to the aggregator almost

simultaneously, what it results is at the aggregator now there are many of workers are communicating with the aggregator at the same time.

So if we consider the pattern wherein we have the parent that made a request to all these worker nodes, and these worker nodes are now responding back to the switch, that is the aggregator node, then all of these packets that come from these aggregators are going to be received simultaneously at the switch, and this would then pass on the packets back to the parent or basically the top level aggregator.

And in this case what we need to see is how much of the packets would each of these nodes deliver, because if they are all delivering simultaneously, you would need to build a lot of buffer at the switch. And often if you build a large buffer, then this in turn adds to the latency, which is again not desirable when you have a deadline to meet.

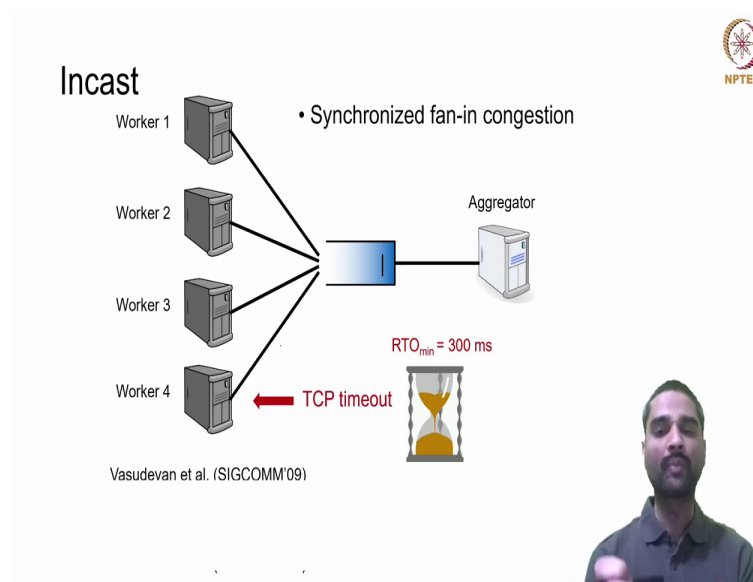
So when the parent node receives the responses simultaneously from all of the nodes, it essentially results in what we call as a micro-burst at the parent node, as many of the packets accumulate at the same time, and this micro-burst often exceeds the capacity of the buffers that you would have at the parent and result in a loss. And if such a loss happens the node that whose packets have been lost has to retransmit the packets again.

And this is what results in what we know as a detrimental effect on the throughput because of this network congestion. At this point it looks like this is a normal phenomenon and the internet that we have seen but how is it any different when we are talking about the data centers. And here, the aspect of TCP is the same in terms of how we do retransmit.

And the way that it would know that there is a loss that the node would perceive a loss, based on the retransmission timer timeout that happens within each of the nodes. And when the retransmission time times out, it would essentially perceive a loss and send the packet back. So the question now is how big is this retransmission timeout, and as we also know, when it times out, and it has to send the packets back, they will again go through the slow start phase to ensure that you are able to not congest the link and in fact, the truth is that there was no congestion of the link at all in the sense of not exceeding the bandwidth, but it was a loss because there was a burst of packets that came to the parent node resulting in the packet loss that happened for that burst alone and hence, we would see that it adds to a detrimental effect on the throughput.

And also, the timeout being on a very large granularity in several of order of milliseconds results in making this even more worse and typically you would have around 500 milliseconds as a timeout, and that means that you are essentially for one packet loss that happens in a micro-burst you are waiting for that much amount of time before even trying to retransmit and that would already have missed the deadlines that we set as around 250 milliseconds for end-to-end communication while at the aggregated level if we said that we had a 10 milliseconds of the deadline, we have missed it by miles and hence, it becomes essential to see how we can address this TCP incast.

(Refer Slide Time: 15:15)

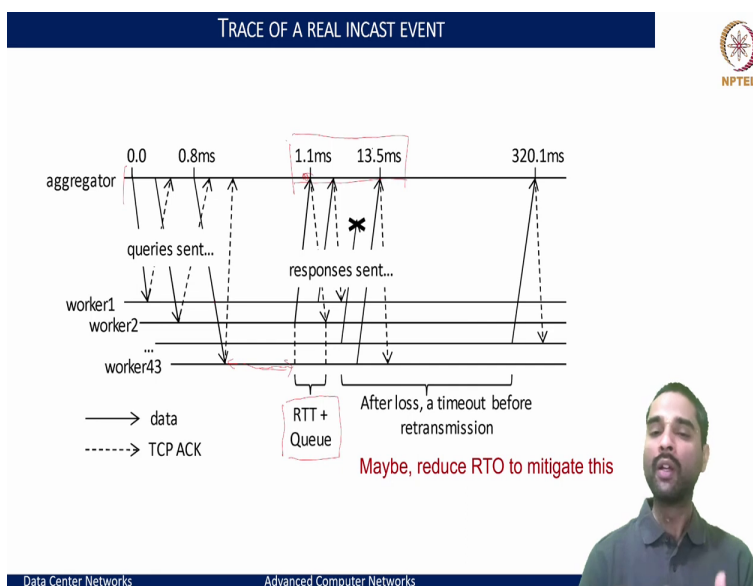


Let us try to visualize and understand how this TCP incast problem really occurs. So if we have a bunch of the model as this aggregator and the workers, and whenever the aggregator makes the requests to a multiple of the workers, the workers would process each of these requests in a stipulated time, typically around a few milliseconds and then send the response back to the aggregator.

And in doing so all of them would try to simultaneously send the packets and whenever we are trying to send the packets, the buffer at the aggregator would overflow and essentially some of the workers' packets might get dropped. And when this occurs the workers put time out and in this case when we see that worker 4's packets were dropped, it would have a TCP timeout that would happen at worker 4.

And RTO time being 300 milliseconds that you could set, it is going to infer that there was a packet loss and retransmit those packets back to the aggregator, and this is what we call as basic incast aspect. And this also is referred to as what we call as a Synchronized fan-in congestion because most of the workers are trying to send the packets in a synchronous fashion at the same time, and that results in a micro-burst of packets resulting in temporary congestion not a congestion indicating that there were sustained congestion at any of the links due to the overhauling of the packets but this is essentially timeline specific aspect that happened due to the synchronized transmission.

(Refer Slide Time: 16:48)



And if we look at in terms of the real trace of what has really occurred, like multiple of the workers received the request from the aggregators over a time and as these worker nodes, worker 1 to 43 try to transmit the packets, and they received their requests and they all started to respond back.

You start to see that all the responses try to come at the aggregator almost within this 1.1 to 13.5 milliseconds time scale. And when lots of packets are being pushed during this time from 40-odd workers we start to see that the round trip time accounting for the communication adds to the queue buildup at this aggregator. And when one of the worker nodes tries to send a packet because of the queue buildup there were packet losses that were that occurred.

And once such a packet loss occurred and then this worker node would have to infer that such a packet loss occurred at some point in time, and that is when basically after the transmission of this packet, the worker mode would have triggered the RTO block and only after the expiry of such an RTO clock that is around 300 milliseconds we can see that right after this point around 320 milliseconds is when the lost packet was sent.

And this gap from 13.5 milliseconds, wherein we expected all the traffic to be delivered, has now been delayed all the way to 300 milliseconds, making the entire aggregators miss their deadlines and also making all of these packets wait until the information comes back, to even reply back and this is where we need now to see what could be the means so that we could mitigate or overcome this.

So one of the very naive approaches would then be, why not just reduce the RTO to a very small timeout? And if we do that, maybe this 320 milliseconds of delay would not happen, and you will be able to retransmit quickly. But let us think again like how close or how short we can have this RTO and what is the impact if we have this.


So the naïve solution of reducing the RTO would, in fact, create a lot more problems as we see that these packets, whatever are going to be delivered are going to be queued. And as the queues build up at the aggregator they would, they would take a lot more time to send the packets out.



Hence, having a very small RTO on one side seems to be a very feasible solution, would result in an adverse aspect of making the workers timeout even when the packets have been really queued and not being dropped. And this would, in fact, have much more adverse effect in trying to unnecessarily re-transmit the packets when there is no need at all.


Hence, marginalizing in terms of how close we can get to RTO, what is the means would be a thing to debate on and it is seen that this is not a solution at all. At best, we could tune it to certain aspects based on the buffers that we would have, but we can clearly see that this would not be a solution either. Hence, we need to think of better alternatives and mechanisms to make this work.

(Refer Slide Time: 20:04)

DATA CENTER WORKLOADS



- Mice and Elephants
- Short messages
(e.g., query, coordination) → **Low Latency** 
 - Deadline sensitive!
 - Never get out of slow start
 - But there is queuing on arrival
- Large flows
(e.g., data update, backup) → **High Throughput** 
 - Ramp up, past slow start
 - Don't stop until they induce queuing + loss
 - Oscillate around max utilization



Data Center Networks
Advanced Computer Networks

To put it into perspective, if we see the typical data center workloads that we discussed, we will have two kinds of workloads, typically one, which we call as the short messages or the mice flows, that span up to a few kilobytes at best within 10 to kilo 10 kilobytes. And if we have like more than 100 kilobytes spanning to several megabytes to gigabytes of flows, we call them as large flows for the elephant flows.

And whenever we speak about the queries that we do or the coordination tasks, the timing sequences that are going to be updated are all basically the short flows that are typically deadline sensitive, and we want them to be served as quickly as possible in essence they are the latency-sensitive flows.


While the large flows, wherever we are having the data updates, the backups, the VM migration, all of these would span several MBs to GBs. And whenever we have such flows, they would have lots of packets that are getting transmitted, and what in essence, they are more throughput sensitive than latency, and they would also be having like, because there are lots of flows, lots of packets that are getting, they would be able to ramp up and go past the slow start phase to the congestion avoidance and continue to operate utilizing the peak bandwidth.

While the short messages, if they are typically like one or two packets or at least 10 packets of one kilobyte, then you would see that they would never get out of the slow start, and these packets would basically be utilizing less of the bandwidth but more in terms of trying to be served at switches as quickly as possible.

And this is where we call these as latency-sensitive short flows, or the mice, and throughput-sensitive large flows, or the elephant flows. And our job is now to accommodate both of these flows in the considerations of how the data centers operate and ensure that we are able to meet the sensitivity of the latencies that these short messages require.

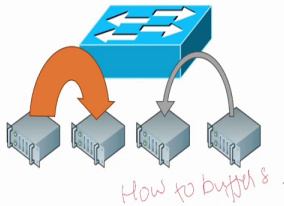
(Refer Slide Time: 22:09)

PROBLEM: BUFFER PRESSURE




- In theory, each port on a switch should have its own dedicated memory buffer

- Cheap switches share buffer memory across ports
 - *The fat flow (elephant) can congest the thin (mice) flow!*



How to buffer?



Data Center NetworksAdvanced Computer Networks

And in this, what is also a crucial aspect to see is both of these flows would be sharing the same resources. In essence, whenever we have these aggregators and the switches that these aggregators and worker nodes go through, they are all going to be sharing shared resources, and the switches in effect, have a dedicated and limited amount of buffer memory that they can use.

So when all the packets, be it the East-West traffic or the North-South traffic they would all share the same buffer space. And if we have several of these East-West traffic or the elephant flows that could be going along, and when it is combined with the East-West traffic of the aggregators and worker nodes trying to exchange the information amongst themselves, then we see the mix of both the elephant flows as well as the mice flows that is happening through the same switch.

And in essence, now the resources over this shared switch are going to be used together, and when we share such a memory space, the elephant flows like the ones in the orange here can basically take the majority of the buffer resources at a switch and make these thin flows to basically wait at the switch get to experience along queuing delay or even make these mice flows to lose the packets due to the overutilization of the queuing buffers at these switches.

Hence, adaptation of this buffer space within this switch and how to isolate these flows also becomes an important consideration. And to summarize what we just mentioned, one, just reducing the RTO timeout would not work because we may end up seeing unnecessary retransmissions due to the queue buildups, which would result in longer delays than the RTO timeout itself.

Two, when we speak about the queues, the queues within these switches, if they are long, then we will experience larger queuing delays, and if we are going to experience larger queuing delays, that means we have to have sufficiently large RTO timeouts to ensure that we do not unnecessarily retransmit the packets. And hence, we do not want the queues to be large, and that also comes at a cost.

And if you are speaking of hundreds and thousands of switches, we want to minimize the cost as well that means we cannot have the switches with large numbers of buffers. And the other end, if we do not have large enough buffers, then we would experience this incast problem with the micro-bursts, resulting in excessive packet losses.

So, having a queue with the memory buffer within each of these switches at a size which is neither long enough nor short enough, I mean, if we have very short buffers, then they may not be able to absorb this micro-burst. So, we want to absorb the micro-burst on one end, but we do not want to add enough delay on the other end. And hence, how to actually design the framework so that how to size the buffers within these switches becomes an important challenge in its own.

And this has been dealt in several of the works, but let us try to keep it to saying that we do not have long enough buffers where we would have all the burst accommodated, and neither do we have a very short buffer where micro-burst cannot be accommodated, so we have some limited space which is now going to be contended between these mice flows and the elephant flows. And our goal is to ensure that we operate and minimize the retransmissions that would occur, and we also do not want to have false retransmissions when the buffers are queued up.

(Refer Slide Time: 25:37)



- Limits search worker responses to one TCP packet
- Uses heavy compression to maximize data.



- Largest memcached instance on the planet
- Custom engineered to use UDP
- *Connectionless responses*
- Connection pooling, one packet queries



And this being a very massive problem, especially for players like Google and Facebook where you are trying to either do some very specific queries or trying to do on Facebook very specific smaller updates that you would want to be looking for. These kinds of workloads demanded that the industry really has to come up with some specific solutions.

And from Google and Facebook, we start to see that they came up with specific solutions, but in essence, they were really workarounds to not address the incast but have a means to get rid of what an incast would really cost. And from the Google's point of view, what it tried to do is whenever you have a search, and the search workers would respond back to the aggregator, they want to limit the amount of information that a search worker would send.

And if you limit to just one TCP packet, then whether that packet is signed or not, the aggregator can then get the information within one packet if it reaches; if it does not, that packet can be dropped, and the information, whatever has been collected up till then could be transmitted out and without going to account for their dropped packet. And this also meant that they would use heavy compression for making sure that the entire data fit in just one TCP packet.

And heavy compression so that you maximize the data that you can transmit in one TCP packet, and the aggregator can accumulate whatever the TCP packets that were received in one go and leave out any of the packets if it was not able to reach and then transmit out the information. And you can clearly see that this is just a workaround because we can still have the TCP incast problem where the packets would really get lost. Only that now at the application layer, we are

able to sidestep these losses and go ahead, but whenever there is a loss, the timeouts would occur, and then they would respond back, but they would just be void at that point.

And alternative from Facebook was to use what they call as a large memcached instances and they custom-engineered to use the UDP. Why use TCP when we have to just update several of the instances with small messages, instead, use the UDP, which is a connectionless framework, and if you deliver the packet, that is fine. If you are not able to deliver, then you just leave it. And then, you can always sync up at a later time and retransmit from the application point of view and keep the transport as just UDP, which is unreliable and connectionless, adding to serve the latency requirements but not really creating anything at the switch or the means to overcome the TCP incast. And the other aspect that also they tried to do is build this connection pooling, which helps basically improve the performance of applications that make multiple of the brief and small connections.

Basically, when you are trying to connect to a database server and make small queries where you will get some key value responses, this connection pool helps avoid trying to re-establish the connection and get the data over the same connection channel, where a pool of connections are already pre-maintained, and you would reuse the same connection pool to get the information which can be just one packet queries and responses that would work.

But nonetheless, these were basically the workarounds, but essentially trying to evade the TCP incast problem and means to get the applications to work, but not really address how we could solve the incast aspect.