**Keynote speech**

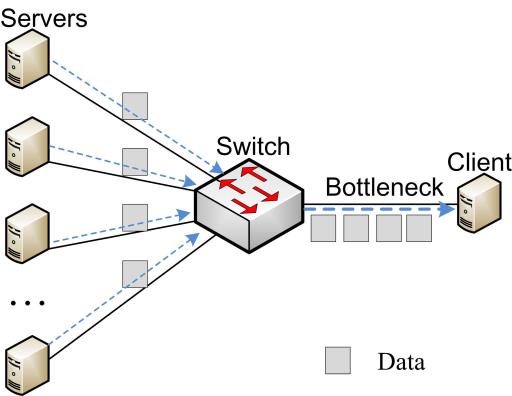
**at**

**The International Conference on Computer Communication 2020**

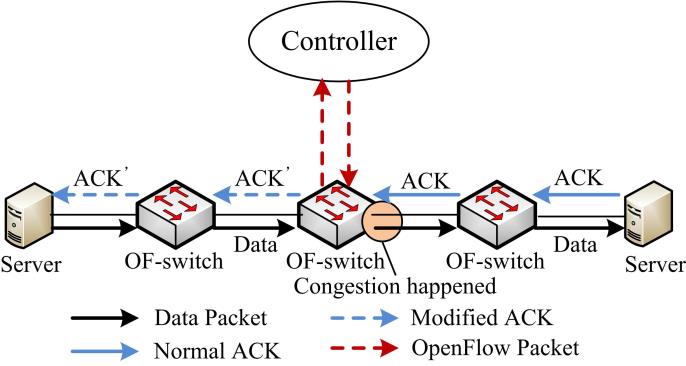
**Daheng Yin**

(All the figure in this script will be included into PowerPoint for presentation)

Good evening everyone, thanks for attendance. I’m Yin Daheng from Key Laboratory of Computer Network and Information Integration, and I’ll introduce an important research projects: *SDTCP: Towards Datacenter TCP Congestion Control with SDN for IoT Applications*, undertaken by our lab. This introduction will summarize the research projects undertaken by our research team in recent years. The progress made in solving the practical problems related to TCP incast based on a software defined network-based TCP congestion control mechanism will be presented.

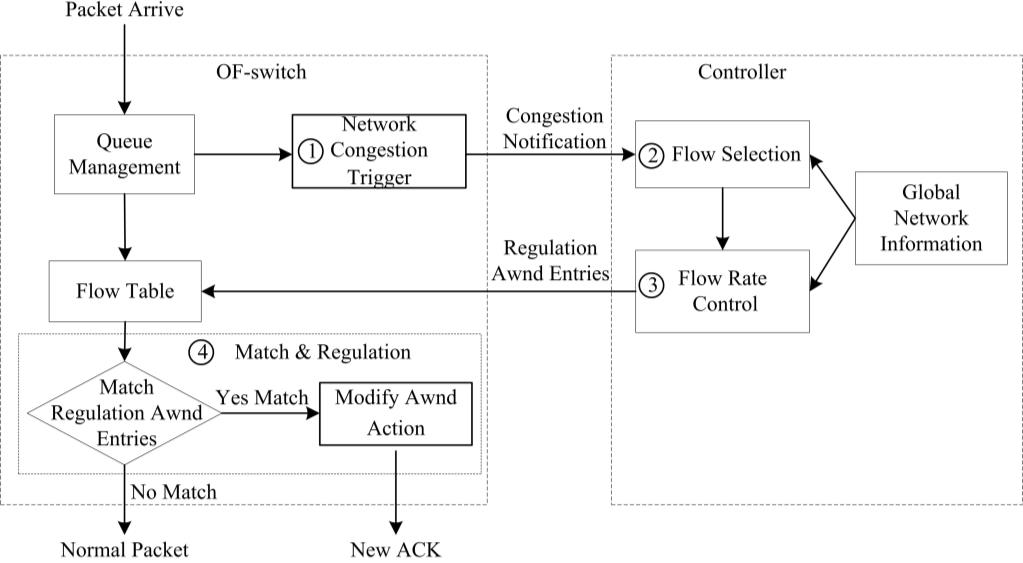
The Internet of Things, or called IoT in brief, has become more and more popular in recent years. Nowadays, IoT applications are increasingly deployed in cloud platforms to perform Big Data analytics, which adds a lot of load to the server. In cloud data networks, or DCN in brief, TCP incast usually happens when multiple senders simultaneously communicate with a single receiver as is shown in Figure 1. The TCP incast make DCN suffer from both TCP burst flows and starvation for TCP background flows. This kind of TCP incast was initially identified in distributed storage clusters and has nowadays become a practical issue in DCN, which is a tremendous interest among researchers in developing techniques to process TCP incast and solve this issue.

**Figure 1 TCP incast scenario**

Before our research, there are solutions for TCP incast focused on reducing the waiting time for packet loss recovery and avoid overflow by controlling switch buffer occupation to using ECN. However, this solution is to complicated. They not only need to modify TCP stack, but also, ignore the characteristics of distinct flows in DCN, so these methods cannot distinguish burst flows and background flows. This equitable treatment of all network flows still makes the DCN suffer from poor performance when incast happens.

**Figure 2 The workflow of SDTCP**

To fundamentally solve this problem, we propose a novel software-defined-network-based congestion control mechanism, referred to as SDTCP. The basic idea of SDTCP is to reduce bandwidth of background flows to guarantee burst flows which are usually more urgent. As is shown in Figure 2, besides the sender and receiver, the SDTCP is related to a series of switches. and a controller monitoring the whole network. Our customized OpenFlow protocol is running on these switches. We refer to these switches as OF-switches.

As we all know, in traditional TCP protocol, the bandwidth of the dataflow was controlled by a slide window, which can be controlled by a window size field in ACK message, so our OF-switches can control the flow from senders by edit the window size in ACK message. To find suitable window size, SDTCP follows 4 steps as is shown in Figure 3.

**Figure 3 SDTCP overall architecture**

The first step is network congestion trigger. We design a network congestion trigger module at the OF-switch. This module can determine whether the network is congested by counting the queue length in OF-switch’s buffer. Once it discovered that the network is congested, it will send a congestion notification message to our controller.

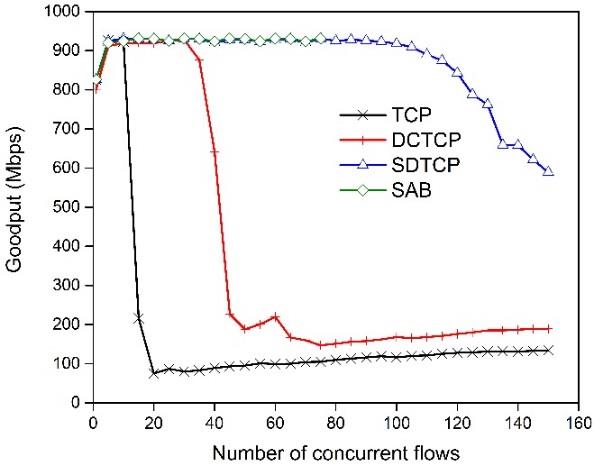
The second step is Flow Selection. There is a flow selection module running on the controller. This module can analyze all of the TCP flow information, e.g., TTL (time-to-live), flow size, and IP addresses of TCP flows, gained from OF-switches through the OpenFlow protocol. Through this analysis, this module can differentiate the background flows and burst flows. With this module, our controller will select all of the background flows passing through the OF-switch once received a congestion notification message from a congested OF-switch.

When the background flows have been selected, controller will perform Flow Rate Control. At this step, a flow rate control module at the controller will at first calculate a desired bandwidth depends on the network congestion level, then estimates the current bandwidth of the chosen background flows and then degrades their bandwidth to the desired one and generates new flow table entries that is used to regulate the background flow bandwidth to our desired one and sends them to the OF-switch.

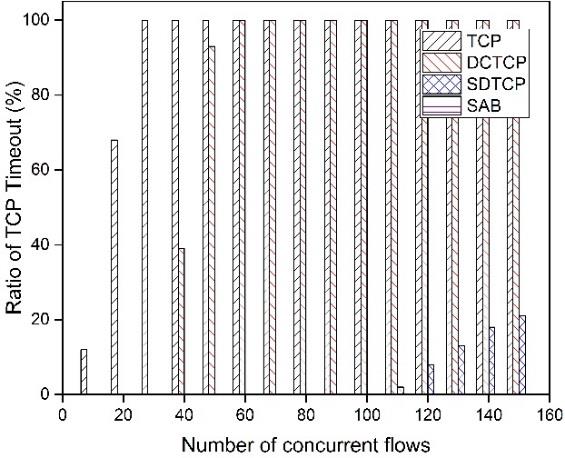
Once the flow table entries on the OF-switchs updated, they will start to perform Flow Match and Regulation. Once TCP ACK packets from the receiver match the regulation entry at OF-switch, the window size field of these packets will be modified to the desired one and then forwarded to the sender. After receiving these modified ACK packets, the sender will adjust the slide window into the desired size. In this way, the bandwidth of background flows can be decreased to our desired one.

That is how our SDTCP runs.

In our experiment, SDTCP gives a good result. Figure 4 shows the goodput of SDTCP, TCP, DCTCP, and SAB as we vary the number of concurrent flows from 1 to 150. As shown in this figure, SDTCP can easily handle 100 concurrent flows and significantly improve the network performance over the incast scenario. With the same experiment, we also present the timeout ratio for these protocols, as is shown in Figure 5. We can see that timeout greatly decreased when performing SDTCP.



**Figure 4 Goodput of SDTCP, TCP, DCTCP, and SAB**



**Figure 5 Ratio of timeout with fixed flow size**

To sum up, SDTCP reduces the sending rate of background flows proactively to guarantee burst flows by adjusting the advertised window of TCP ACK packets of the corresponding background flows. Our results demonstrate that the SDTCP mechanism guarantees high throughput for burst flows effectively without starving background flows.

Thank you very much for your attention. If there are any questions, please feel free to ask. I'd be happy to answer any questions.