## I. INTRODUCTION

From the birth of computer networks, congestion control and flow control has always been a very popular research aspect, which’s purpose is to make full use of the network device performance and to schedule traffic. In view of the hierarchical design of the computer network, the related implement is almost on TCP (Transmission Control Protocol) layer. Implementation of TCP contain four intertwined algorithms: slow-start, congestion avoidance, fast retransmit, and fast recovery0. But with the development of the network technology , the performance of the these basic algorithm is unsatisfactory[1] , so people introduce some complement algorithm to polish these legacy, such as SACK (Selective Acknowledgement)、ECN (Explicit Congestion Notification)、DCTCP(Data center TCP)、etc. [3][4] But it is not difficult to find that the end nodes still dominate the intermediate nodes. For instance, even though the intermediate node can use ECN to notify the congestion, whether decrease send rate and how much to induct still depend on the end nodes. The reason is nothing less than being compatible with existing algorithm and the limitation of the intermediate node hardware. Compared with the intermediate nodes, the terminal nodes tend to have more customizability, So the relatively complex works often complete by end nodes.

All these limitation and sewing have brought about the complexity of the network device and hampered the network further development of the current application trend of the cloud computing、big data and server virtualization [5] . SDN[6](Software Defined Network) as the wind indicator of the next generation network is expected to solve above problems. Using hierarchy module SDN is designed with control layer and data layer. The control layer, including a logical center and programmable controller, master the global network information to facilitate administrator and researchers to configure network and deploy new protocol, etc. The data layer, including programmable switch (unlike the traditional Layer 2 switch, specifically refers to the device used to forward data. ) which could consume P4 program[7], do what control layer indicate which present by OpenFlow protocol[8]. The advantage bring by SDN goes without saying, besides it provides a wholly new perspective to perfect what traditional network architecture want to do, which is compatible with the legacy and is more concise, flexible, extensible, programmable.

Using all these excellent features provide by SDN, this article not only illustrates the convenience to solve legacy compatible problem and to optimize existing algorithm, but also put forward a new method to control congestion and flow output. The remainder of this paper is structured as follow: Section II describes how to mediate ECN and Non-ECN flow in a SDN environment, and introduce our Linux kernel patch. In section III, we discuss remaining problem we have in section II, then we raise a new approach ISAW (Intermediate node Set Advertisement Window) to control congestion and flow output in a compatible and convenient way. The methods of section II and section III are all verified at mininet simulation environment which present expected result. Section IV discusses related work with INT (In‐band Network Telemetry), and section V concludes the paper.

## II MEDITATE ECN AND NON-ECN

Conventionally, TCP/IP networks signal congestion by dropping packets. When timer expire sender still don’t receive the ACK, it will decrease it’s send rate because of the missing package (no matter the data package or the ACK package) indicate the bad condition of the network, while ECN (Explicit Congestion Notification) allow end-to-end notification of network congestion without dropping package. If the endpoints and underlying network infrastructure both support it, an ECN-aware router may set CE (Congestion Encountered) in the IP header instead of dropping a package when encounter congestion. The receiver of these package will inform sender to reduces its transmission rate as if it detected a dropped packet. However Non-ECN may suffer when compete with ECN flow, as the switch may drop long sequences of Non-ECN package, causing timeouts in non-ECN flows. This effect is particularly pronounced with a higher proportion of ECN flows, which typically leads to a higher ECN traffic rate. As a result, it will take longer to drain the queue below the marking threshold as more ECN traffic keeps arriving, and therefore may cause a longer congestion period[9].

Firstly, we conduct an experiment to illustrate the unfairness between ECN and non-ECN flows, for various numbers of them. The topology of our experiment as Figure1.



Figure Simulation topology of both section II and section III

Ten senders are connected to a single receiver through a which is a programmable device (simulated by BMV2[10]). We customized the switch use P4, let it could process both ECN and non-ECN flow like traditional switch. All links have a bandwidth of 100Mbps, 200Mbps and a delay of 0.25ms, so the RTT is 1ms. As to other configuration and tools, we just use the same as paper[9].

|  |  |
| --- | --- |
| Figure Unfairness between 5 ECN flow and 5 non-ECN flow with 100Mbps | Figure Unfairness between 5 ECN flow and 5 non-ECN flow with 200Mbps |

Figure 2 demonstrate the unfairness between ECN flow and non-ECN flow by plotting time-series of their goodput. It shows that while ECN flows coexist with each other, non-ECN flows may significantly starve.

To solve the problem Paper [9] bring up the conception of vCC layer



Figure vCC Layer Demonstration

The vCC layer just like an agency, on one hand it will modify the hand-shake process and package transmission of non-ECN flow to help non-ECN pretend to be a ECN flow, on the other when encounter congestion vCC layer will translate the message to non-ECN hosts to help them deduct send rate lest network switch drop their package thus starve. In practical application, vCC layer implement in VMware’s ESXi hypervisor[11], thus in physical multi-hosts environment such as a company, a school, it’s almost impossible to induct such a vCC layer to help eliminate unfairness between ECN and non-ECN flow. And to translate, vCC layer has to establish a whole ECN-aware calculation model for each non-ECN flow which will definitely consume amount of computing resource especially in high speed link.

Besides the problem above is just an epitome of the network development: new demands midwife new strategy and new methods which bring the compatible problems. It’s hard to update old hardware and software timely. But if we could free end nodes from heavy work, let programmable intermediate nodes be decisive, the situation will become better. By that time, intermediate nodes (in SND, is the controller) could sentence the behavior of every flow and when these are demands to update, it’s no need to bother end nodes and uses. The good news is that SDN has provided the programmability of the controller and switch, combined with above thinking, we conduct an experiment to transplant vCC to intermediate, so that we could implement it on almost all environment as long as the hardware is capable.

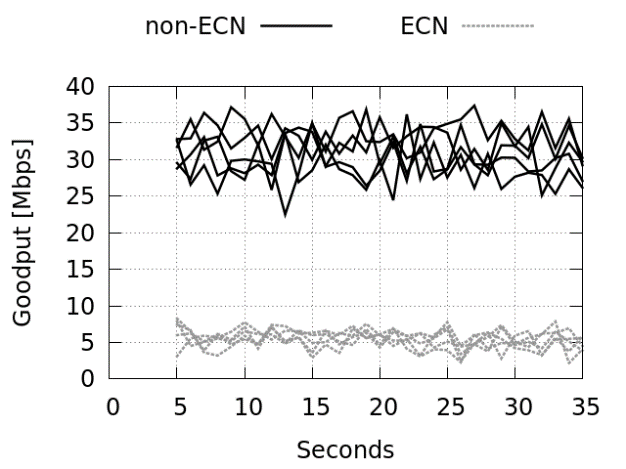
As mentioned above, because that switch is more likely to drop non-ECN package when encounter congestion, ECN flow always overwhelm non-ECN flow. What will happen if switch mark CE (Congestion Encountered) of ECN package while don’t drop non-ECN package when queue is crowed. Obviously when the link is congested ECN flow will be dampened while non-ECN flow feel “every is good” and increase its send rate continuously, thus more congestions more suppresses. Benefit from programmability and flexibility we conduct the experiment.

Figure Reversal unfairness between 5 ECN flow and 5 non-ECN flow with 200Mbps

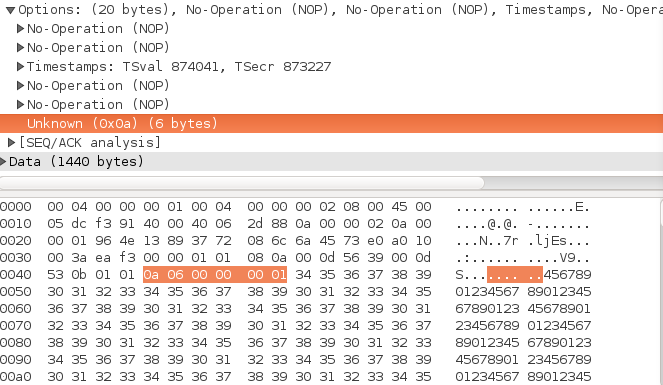
The vCC layer build a calculation model for each non-ECN flow which is a heavy task for lack-computation-resource intermediate nodes. But the main idea is let non-ECN flow behavior like a ECN flow, so we modified Linux kernel and build a patch. The main work of the patch is that if user turn this patch on by “sysctl” system call, every package send from his hosts will contain that time CWND (Congestion Window) in form of a TCP Option.

Figure The effect when turn on the patch. 0x0a present the TCP option type, 0x06 is the total length and 0x01 is the value of CWND. A plugin could let wireshark understand our customized tcp option.

Then switch will recognize a ECN flow which informed by a controller as a normal flow and store this ECN flow’s CWND as . When package which is from receiver to a non-ECN sender enter the queue, the switch will set its  to  so that the non-ECN sender can be aware of the congestion indirectly. Note that the switch will not drop non-ECN package this time and the interesting thing is we can easily determine which non-ECN flow pretend to be a ECN flow and which ECN flow is the normal just by one command.

|  |  |
| --- | --- |
| Figure Choose one non-ECN flow behavior like a ECN flow with 200Mbps | Figure Let all non-ECN flow behavior like a ECN flow with 200Mbps |

## Section III ISAW

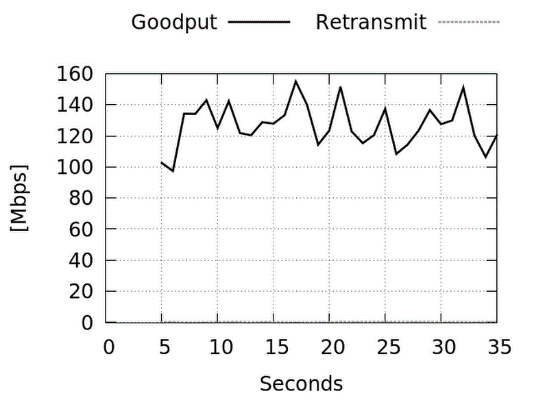
The approach of section II do have some flaws, it poisons total output. As Figure 9 Toxic total output shows that the retransmit is satisfying 0Mbps while total goodput is about 140Mbps which should be about 170Mbps (although the parameter is 200Mbps, because of the simulation environment 170Mbps is what we get from pressure test). Actually, our implement of RED[12] (Random Early Detection) is very shabby due to the lack of expression ability of this version P4 language. When queue length over the threshold we set, switch will drop or mark the package according to the flow type and the command we give. So, although non-ECN behavior like a ECN flow now, it seems that the output is already is the load-capacity of the bottleneck switch until next version of P4 language.

Figure Toxic total output

But what exhilarate us is that we could control a flow behavior by the method ISAW (Intermediate node Set Advertisement Window). Using ISAW we could do some interesting things:

1. We could use it to solve some unfairness like what we do in section II.
2. In the process of IoT promotion, the computation capacity is still a limitation of some embedded device. So, we could simplify TCP implement and let intermediate nodes still could control the behavior of end nodes under the premise being compatible with existing TCP implement. The method ISAW is quit concise and eligible is this situation.
3. In the scenario of paid service, users will get the bandwidth according the package they pay. When the actual bandwidth beyond expected bandwidth in unit time, intermediate nodes will drop package to slow down end nodes send rate. No doubt, this will consume a lot of computation resource and is harmful to total bandwidth (dropping package). In SDN, ISAW should be a good method to solve to problem.

To verify our theory, we write another P4 program. After consuming this P4 program, BMV2 could receive command from controller. The command is something like



Where  is the value we want set, and we define  as tcp flow connected to . So the command means if a package’s egress port is  then set package’s  to . We use port as our granularity in this experiment while in practical we could use varied granularity. After apply a command, if the TCP flow

1. Stevens W．TCP Slow Start，Congestion Avoidalice．Fast Re transmit．and Fast Recovery Algorithms RFC 2001．Jan．1997
2. Paxson V．ct a1．Knowfl TCP Implementation Problems RFC 2525．March 1999
3. Robit Ramani．Abhay Karandikar Explicit Congestion Notification(EcN) in TCP over Wiceless NetwOrk ICPWC’2000，O一7803—5893—7 IEEE．
4. Alizadeh M, Greenberg A, Maltz D A, et al. Data center TCP (DCTCP)[C]// ACM SIGCOMM 2010 Conference. ACM, 2010:63-74.
5. Jain R. Internet 3.0: Ten problems with current Internet architecture and solutions for the next generation. In: Proc. of the IEEE MILCOM. 2006. 1-9. [doi: 10.1109/MILCOM.2006.301995]
6. McKeown N. Software-Defined metworking. In: Proc. of the INFOCOM Key Note. 2009.<http://infocom2009.ieee-infocom.org/technicalProgram.htm>
7. The, P4, Language, Consortium. The P4 Language Specification[J/OL]. http://p4.org/.
8. Mckeown N, Anderson T, Balakrishnan H, et al. OpenFlow: enabling innovation in campus networks[J]. Acm Sigcomm Computer Communication Review, 2008, 38(2):69-74.
9. Virtual Congestion Control

1. <https://github.com/p4lang/behavioral-model>

1. <http://www.vmware.com/products/vsphere-hypervisor.html>
2. 早期随机检测文献