Explicit congestion notification-Based Congestion Control Algorithm for High-Performing Data Centers

Ezekia Gilliard*, Kashif Sharif†, Arif Raza†, and Md Monjurul Karim†

* Mwalimu J.K. Nyerere University of Agriculture and Technology, Mara, Tanzania
† Beijing Institute of Technology, Beijing, China

***Abstract*—A high-performing data center is vital for the smooth operation of organizations and institutions. However, the widespread utilization of data centers has resulted in heavy traffic and various impairments such as TCP-Incast, buffer overflow, and long queues. The reliability of the Transmission Control Protocol (TCP) has been called into question due to its unsatisfactory performance in the data center environment in maintaining high throughput and meeting deadlines. Several approaches have been implemented to address these impairments, including TCP modifications, increased buffer sizes, and improved congestion algorithms.**

This paper presents an enhanced algorithm that employs the Explicit Congestion Notification (ECN) mechanism to inform network devices of the extent and location of congestion, with the aim of minimizing congestion and improving flow completion time. The simulation results demonstrate that the proposed algorithm is capable of meeting flow deadlines, with the average Flow Completion Time significantly better than other state-of-the-art solutions.

***Index Terms*—Congestion control, Datacenter, Flow Control, ECN**

I. Introduction

Data centers (DC) are now the backbone and growing infrastructures to serve many new services in our world due to their capability and importance. Numerous companies have been relying on them to handle their day-to-day activities. These businesses' big part of their profits depend on their availability and well function [1]. This technology is significant to the revenue since many companies like Google, and Amazon have reported that even a millisecond delay can affect enormously in their revenue [2]. The growth of DC is inevitable due to the high demands of internet services and high interest in the industry, academia, and researchers [3].

Cisco whitepaper predicts that mobile connectivity will reach over 70% of the world's population this year, and expects the total number of global mobile subscribers to increase from 5.1 billion in 2018 to 5.7 billion by 2023, representing 71% of the population. [4]. All of these networks need data centers to run smoothly.

At present, there are numerous Congestion Control (CC) algorithms developed to address the impact of congestion on Data Center Networks (DCN). While many of these algorithms are TCP-based [1], [5], [6], others are Non-TCP-based [7]–[10] and aim to improve specific aspects of DCN at the expense of others. However, these algorithms are not universally applicable to DCN, as they rely on tradeoffs between DCN metrics to achieve their objectives. To ensure the optimal functionality of DCN, these challenges must be addressed as it evolves.

In this paper, we propose an ECN-Based Congestion Notification Solution algorithm improved for relieving congestion while minimizing flow completion time. The proposed model utilizes Explicit Congestion Notification (ECN) to notify network devices of the extent and location of congestion.

Explicit Congestion Notification is one of the features that comes with the Transmission Control Protocol (TCP/IP). It works as an extension that marks the packets with information that shows the extent of congestion experienced in the path. It is instrumental since instead of dropping the packets, it notifies the sender. When the receiver receives the marked data packet, it informs the source to reduce the transmission rate.

Most of the algorithms deployed in the data center need new network infrastructures and deployments of algorithms in the source or middle devices or at the destination. ECN implementation is cheap and requires fewer changes in the network devices. ECN is very beneficial to short-flows as more than 80% of data center flows are mice flow. Deploying it, you equipped the data center with real-time status of the flows in the network through marking and notifications. Most of the algorithms depend on the timeouts to detect packet loss or congestion hence the slow reaction to the extent of congestion.

Implementation can be archived using the IP header fields, transmitting information through two bits in the header to show the congestion experienced in the network(see figure 1). The fields in the DiffServ conceal four codepoints to notify three states, which are Non-ECN-Capable or Non-ECT, ECN Capable, and Transport, or ECN (0) [8], [11].

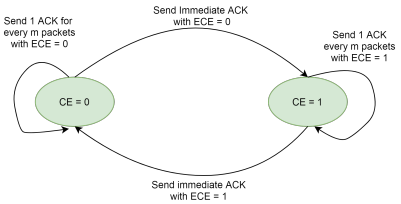

Fig. 1. Two state ACK generation state of the machine.

II. Data center Workload Architecture

In data center architecture, the worker-aggregator model is commonly deployed for task distribution among workers with a completion deadline. Workers complete their tasks and return them to the aggregator. Servers are arranged in clusters, each with multiple tiers, where server workers are located within the rack and connected to top-of-rack layer two switches. These switches are further connected to upper-tier aggregator switches in a mesh to provide redundancy, decrease latency, and offer a non-blocking switching fabric. Figure 3 shows a standard leaf-spine topology implementing a worker-aggregator model. There are various optimized topological configurations that have been extensively studied and published, such as DCell [12], BCube [13], Facebook's 4-post [14], Google's Clos [15], VL2 [16], etc. These different architectures indicate that data centers vary depending on the services they offer and whether they are third-party or owner-operated.

2023 IEEE AFRICON | 979-8-3503-3621-4/23/$31.00 ©2023 IEEE | DOI: 10.1109/AFRICON55910.2023.10293272

979-8-3503-3621-4/23/$31.00 ©2023 IEEE

A study by Munir et al. [17] shows that every data center has a distinct traffic flow pattern depending on its architecture and the type of applications running on it [18]. This finding indicates that data centers with different architectures and running various apps generate different traffic patterns. Reports suggest that some data centers have 80% of traffic within the rack, while others have shown that most of the traffic is shuffle traffic among racks and clusters [19]. Therefore, the amount and type of traffic, length of flows, size of packets, location of source and destination pairs, demand concentration, and several concurrent connections are all heavily dependent on the services provided and the arrangement of servers inside a rack/cluster.

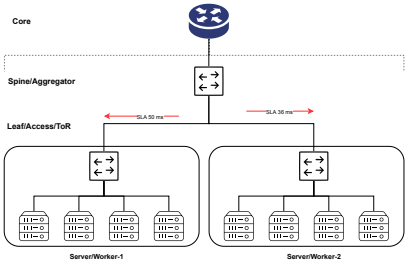

Fig. 2. Worker-Aggregator model in leaf-spine topology.

III. The Proposed Algorithm

The proposed algorithm is designed to address the challenge of keeping congestion to the minimum while minimizing the flow completion percentage. Many users have been using the flow completion rate to measure how good the network is. A complete transaction in the shortest time possible is the goal of every user. The algorithm approaches these problems by restraining congestion to the most modest level while ensuring a short flow completion time. The algorithm works in two principal segments in a source (sender side) and, layer three Devices & Receive. In the following sections, we have explained its architecture by showing the working principle, followed by functionality details.

A. Working Principle

The algorithm employs the use of ECN marking in the packet to show the extent of the congestion at intermediate routers and layer three switches. Network devices residing in this layer are responsible for creating paths and transmitting data from one node to another. Through employing ECN technology, the location of congestion is also indicated by the intermediary device in the Diffserv field of the IP header. When the receiver receives the marked packets, returns this field in the acknowledgment packet to the source without any modification of this information.

When the source receives these packets, it changes its rate based on the received ECN values. Due to the hierarchical nature of the data center topology, the congestion is indicated using the 3-bit value at Top of Rack (ToR) switch (of layer three), spine switches, or core routers as shown in Table I.

Usually, the congestion at the spine switches does not require rate control of flows within the rack. Likewise, congestion at the core level should not impact flows contained within a cluster. By applying this principle, the algorithm dictates the initial rate of new flows destined to pass the congested level. For the existing flows, the algorithm decides according to priority to either continue as they are or are rate controlled by changing the priority. The packet prioritization process is done based on the remaining flow size and remaining deadline time. Three weighted fair queues (Weight:- High:3, Medium:2, Low:1) are maintained at source and intermediate layer three devices, where high-priority data gets three times the bandwidth of lowest priority.

B. Congestion Notification

In the running of this algorithm, the DiffServ field in the IP header is redefined. The DiffServ field contains 6 bits, which we have divided into 3 bits for carrying priority information and 3 bits for congestion location information. The remaining 2 bits in the Type of Service (ToS) field, which are bits six and seven, stay the same as per ECN definition in internet protocol glossary [20], [21].

However, the criteria for packet marking in this algorithm have changed to use a packet queue where a packet is marked when arriving. [22], [23]. The algorithm uses sojourn-time-based marking of ECN bits at all layer three devices in the data center. As described in Table I, the value of the 3-bit field is determined based on the source-destination address regarding marking the device's location. An layer three switch working at the spine with the same subnet ID as of source will mark 011, whereas if its subnet ID matches that of the destination, then it marks 100. Once a device sets the ECN bit, subsequent devices shall not override it even if they experience congestion. Hence, the source is only notified of the congestion point closest to it.

TABLE I
3-bit Congestion-Location Values.

bits	Location
000	Non-Compliance/No Congestion
001	Access ToR - Local $(if\ layer3)$
010	Access ToR - Remote $(if\ layer3)$
011	Spine - Local Cluster
100	Spine - Remote Cluster
101	Core - light
110	Core - Heavy
111	Beyond Core

C. Sender-side Functionality

The proposed algorithm on the sender side has three rules. These rules are ECN measurement, new flows, and existing flow. These rules are described as follows

- ECN Measurement
  In this part, congestion is detected by computing a running fraction of last k ECN marked acknowledgment packets received on the per-flow basis at the sender $(k=10$ in this work). The congestion threshold is set to 0.7 of k, after which the rate control mechanism is executed for that specific flow. Furthermore, the transport layer maintains a congestion table for tracking the flows that are in congestion, and capture destination addresses, destination subnets, ECN congestion-location values, and the state of the flows before congestion.
- New Flows
  In the other component of the sender's side, all new Flows that are initiated can only start if and only if they can be finished (theoretically) within their given deadline. Using information from the congestion table, all new flows that are initiated originated from the same subnets/destinations, which are already reporting congestion starting with the traditional slow start mechanism of TCP. This method will help in controlling congestion in these destinations. Contrarily, if the flows are initiated, and it is not listed

in the congestion table, the new flow skips slow start, and the rate is set to SS_{thresh} followed by an additive increase. The initial priority of each flow is calculated as a function:

$$P = f(T_L, D_L, \alpha_n) \quad (1)$$

where, T_L is time left for flow completion, D_L is data left in flow, α_n is average flow rate of n^{th} priority flows. Thereafter the priority of flow remains the same unless, (a) If congestion threshold is reached, or (b) Congestion-window size data has been transmitted.

- Existing Flows

The priority of flow is recomputed once a congestion window worth of packets has been transmitted. Utilizing the time remaining and data left, the priority may be increased to meet the deadline. In the case of congestion indication for a flow, the rate and priority are controlled as per policy defined in Table II where priorities are represented as P1, P2, and P3 in order from high to low.

The congestion-location and current priority of flow are used to either degrade the priority (affecting the rate) or to introduce inter-packet spacing in the case of P3. At this point, the current congestion window also freezes. The whole policy is only in effect until the ECN markings are above the congestion threshold for that flow, after which they return to the state before congestion was experienced.

TABLE II
DECISION MATRIX AT SOURCE FOR CONGESTED FLOWS.

Priority/ C-Value	P1	P2	P3
000	NA	NA	NA
001	P1 → P2	P2 → P3	IPS1
010	NA	P2 → P3	IPS2
011	P1 → P2	P2 → P3	IPS1
100	NA	NA	IPS2
101	NA	NA	IPS2
110	P1 → P2	P2 → P3	IPS1
111	NA	NA	NA

- NA: No Action, continue as per current policy

IPS1: Inter-Packet Spacing of 1 RTT
IPS2: Inter-Packet Spacing of 0.5 RTT

D. Layer three Devices & Receiver

In layer three devices and receiver, there are no significant changes required at the receiver end other than ECN compatibility. ECN is compatible with all devices since its feature comes with TCP. The only minor modification needed is to copy the whole ToS byte in the packet's IP header to the acknowledgment packet. All layer three devices in the network are also ECN compatible, with ECN marking based on sojourn-time, rather than the buffer size. Moreover, the devices use the priority defined in the modified DiffServ field to allocate forwarding bandwidth to the flows

IV. ALGORITHM DISCUSSION

ECN is a proactive method to detect congestion early in the network compared to an RTT-based mechanism. In this sense, ECN uses less time to inform the source of the extent of the congestion compared with the time that RTT can use to detect packet loss. Also with RTT, the sender can not know the location where there is congestion. Although RTT increases can be treated as congestion notifications, it requires more time and computation to actively perform the measurements [24].

In current, the data center architectures and accesses at ToR are kept in layer two to process incoming packets at line speed and avoid blocking. The proposed algorithm does not require the ToR switch to be at layer three, but if they are set, the algorithm will support them.

The priority selection in this algorithm is directly related to the current flow rate, and the minimum required flow rate to minimize flow completion time.

Equation 1 is a generic function that can be replaced by any prioritization algorithm, as long as the priorities are limited to eight levels (3-bit representation). In our implementation, the initial priority is set, if

$$R_m^i > \alpha_n \quad satisfied\ for\ some\ n \quad (2)$$

where, R_m^i is the minimum required rate to complete i^{th} flow within deadline, and α_n is the average flow rate of all current n^{th} -priority flows. Hence, $P = n$, for n that satisfies equation 2. For continuing flows, priority may change only if:

$$R_m^i \geq R_c^i \rightarrow P = P - 1 \quad (3)$$

where, R_c^i is the current rate of i^{th} flow, and P represents priority as integer in [1,3] with 1 being the highest.

We have also redefined the use of Diffserv bits in the ToS field of IP the header. Doing this requires satisfying two main questions:

1) **What if a source needs to use the Diffserv code point?**
In our literature survey, we have not found any variant of TCP which uses Diffserv code points. In our study of more than sixteen algorithms, we have found that most of the research uses it to indicate priorities that range from 0-7 for IP. For this work, we have limited it to 3 bits, but the primary objective is the same as giving priorities.
2) **What about TCP backward compatibility?**
Data center implementations are highly customized to provide the best possible performance at the production level. Most of the outside world communication is limited to a few (web) interface servers, which can run generic TCP. Moreover, we have not modified the 2-bit ECN field, which resolves the backward compatibility question. A more radical approach would be to increase ECN usage, and assume that all layer three devices are ECN compatible, hence not requiring the ECN bits for representing availability. The full TOS field can then be used to mark locationspecific congestion, by bit positions representing different locations (ToR, spine, core, etc.).

The algorithm is fundamentally dependent on the leaf-spine topological architecture of data centers. The study done in 2015 concludes that the alternate topological designs do not make up for the complexity, management, and cabling required for changing the topology [15]. Hence for the foreseeable future, the proposed solution can work without any significant changes.

V. EXPERIMENT AND EVALUATION

The proposed algorithm is implemented in the NS3 simulator to evaluate implement-ability, scalability, response to realistic traffic, and overall performance [25]. NS3 is an open-source, powerful network simulator for research and education. NS3 is written in C++ programming at the core, and it has python language bindings to facilitate the framework in compiling and simulating applications.

NS3 has a lot of libraries and tools for network simulation, which covers topologies, model development, performance analysis, and other aspects of simulation [26].

We have also used Netanim as a network visualizer for animating the simulation using the XML generated files created from the data from the simulation. In the simulation, other tools we have used are Wireshark, the best protocol analyzer for inspecting packets and flows

TABLE III
SIMULATION PARAMETERS.

Parameter	Value
Server-Server delay inside Rack	50μs
Server-Server delay across Rack	80μs
Buffer size at switch	300KB/port
ECN threshold	65%
SS_{thresh}	16pkts
Initial RTO (for all)	10msec
Total servers	4×64
Sojourn-time Threshold	78μs
Total connections	7680
No. of Core Router	4
No. of Spine Switches	16
No. Leaf/Access ToR Switches	32
No. of Servers	256
Link between Leaf and Spine Switches	10Gbps
Link between ToR Switches and Servers	10Gbps

in the simulated data center, and Gnuplot for plotting graphs from the generated data from the simulation.

The simulation was done on the system with Ubuntu operating system version 18.04. The server is powered with 16GB of RAM, and a Core i7 3rd Generation CPU (3.0 GHz).

A. Overview Data center Architecture

In this simulation, we have used a fat-tree topology architecture to simulate data center architecture. K-ary fat tree is the rooted tree with no more than k children for each node. It follows the traditional data center architecture, which includes the core, aggregate, and the edge.

The network topology is designed to represent a 4-post leaf-spine architecture, with a core router at the top. There are eight leaf/access ToR switches connected to 4 spine switches. The links between leaf and spine are 10 Gbps connected in a full mesh, i.e., each ToR is connected to 4 spine switches, as shown in Figure 3. The ToR switches are connected to 8 servers, each with 10 Gbps links inside the rack. The whole network consists of 4 clusters with core routers connecting them with 10 Gbps links, as shown in Figure 3. Additional parameters for topology are shown in Table. III.

The simulated data center uses a fat tree unique addressing patterning scheme for the switches and the pods in the topology. This addressing pattern makes it easy to know the pod and switch location in the topology. The pods are numbered from left (k=0) to right (k-1), and switches are also from right to left and up and down across data center layers.

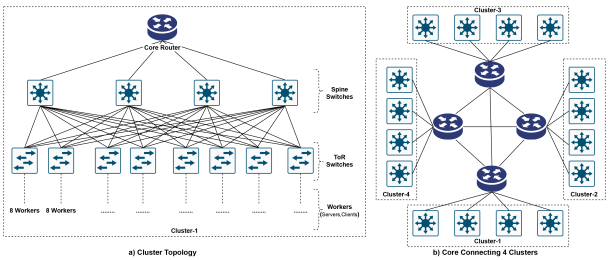

Fig. 3. 4 Post Leaf-Spine Architecture.

B. Nodes and Links setup

The components of topology, like nodes, are created and put in the container, so the container's attributes can apply to all nodes in that container. In this simulation, pods, and all other network devices are created. After creating the nodes, links for connection of point to point must be created with the speed and queue delay parameters. NS3 library installs links with their specified properties. Then all nodes and other network devices are given IP addresses as per the location of nodes and pods. NetDeviceContainer instantiates the net devices and connects each layer device with the above and below layers with the MAC, IP address, and subnet.

Four clusters of Server and client insistence are created and assigned IP addresses. They include workers and clients who are connected to the TOR switch. They communicate with each other and generate the mice, and elephant flows with the use of applications installed in all servers and clients to allow sending and receiving requests.

Furthermore, we have implemented RED (see code 3) as both queuing discipline and scheduler with global routing. It uses Equal-cost multi-path routing (ECMP) to enable the selection of random best paths to the destination. ECMP helps in ensuring the utilization of link bandwidth. Later on, we have implemented ECN(see code 2) for marking of the packets and some other TCP models for performance comparison. Other experiment parameters are shown in Table III.

```
Config::SetDefault ("ns3::Ipv4GlobalRouting:
:RespondToInterfaceEvents", BooleanValue (true));
Config::SetDefault ("ns3::Ipv4GlobalRouting::EcmpMode",
StringValue ("ECMP_RANDOM"));
```

Listing 1. RED Implementation

```
        Config::SetDefault ("ns3::TcpSocket:
␣:SegmentSize", UintegerValue (packet_size));
        Config::SetDefault ("ns3::TcpL4Protocol:
␣:SocketType", StringValue ("ns3::TcpNewReno"));
        Config::SetDefault ("ns3::TcpSocket:
␣:DelAckCount", UintegerValue (1));
        Config::SetDefault ("ns3:
␣:TcpSocketBase::UseEcn", BooleanValue (true));
        Config::SetDefault ("ns3:
␣:RedQueueDisc::UseEcn", BooleanValue (true));
        Config::SetDefault ("ns3:
␣:TcpL4Protocol::SocketBaseType", TypeIdValue(TypeId:
 :LookupByName ("ns3::DctcpSocket")));
        Config::SetDefault ("ns3::DctcpSocket:
␣:DctcpWeight", DoubleValue (1.0 / 16)););
```

Listing 2. Other TCP model and ECN Implentation

```
        Config::SetDefault ("ns3::Ipv4GlobalRouting:
␣:RespondToInterfaceEvents",BooleanValue (true));
        Config::SetDefault ("ns3::Ipv4GlobalRouting:
␣:EcmpMode", StringValue ("ECMP_RANDOM"));
```

Listing 3. RED Implementation

C. Traffic Simulation

The data center houses various applications and services that generate different types of flows. To simulate this diversity, we generate mice and elephant flows representing different types of traffic. Additionally, clients randomly generate short and long flows to simulate the real-world data center environment. To trace the congestion window and slow start threshold, we set a data rate of 1 Mb per second and use callback attributes in NS3. After configuring the settings and creating the object, we initiate the startup and stop time of the applications. The server initiates a maximum of 30 connections for destinations in the rack, cluster, and cross-cluster, with an even distribution. In NS3, we have implemented this by setting these flows, as shown in the code 4.

```
        int MB = 1024;
        int KB = 1;
        int size=std::rand() % 50 + 1;
// random data generation
        int short_flow=50*KB;
        int long_flow=size*MB;
        std::string data_rate="1Mbps";
```

Listing 4. Taffic Simulation

VI. Performance Metrics & Protocols

This experiment's ultimate goal is to measure the flow completion time and deadline met of the proposed algorithm against other TCP models. Our primary metric is Flow Completion Time (FCT), as it is an excellent metric for measuring the data center's performance [27]. In doing so, we are comparing the FCT of overall data center flows, and then we have separately produced the FCT of mice and elephant flows. An additional parameter is the measurement of the flows which completed within the deadline.

The proposed algorithm is compared to three different data center rate control and scheduling algorithms, as these are the closest works to our proposed scheme. DCTCP is the baseline congestion control algorithm for data centers [28]. D^3 is a deadline-aware algorithm for bandwidth allocation that required changes in switching fabric, which has been implemented in the simulated switch node [29]. D^2 TCP is a congestion and deadline aware algorithm that uses gamma correction function to adjust the congestion window size [30].

VII. Experiments Results and Discussions

The simulated experiments results were obtained from the measurement of Flow Completion Time (FCT) of the master output file that has the record of the flow measurement from the source to the destination. Also, we have calculated the flow to know that all flows are completed within their deadline. In the following sections, we present and evaluate the results of the proposed algorithm compared with other TCP models.

- **Flow completion time**: After the simulation, the test shows that compared with D3, D2TCP, and DCTCP, the proposed algorithm was able to maintain the overall flow completion time and achieve stability for all flows. This significant FCT stability is attributed to ECN implementation and Multi-path routing. The use of Multi-path in the proposed algorithm also significantly improves the balance of the flow completion time, and the results confirm that not only the elephant flows but also mice flows also have a stable FCT.

  ECN notifies and ensures that the congestion is kept to a minimum, thus significantly improving the data center's traffic conditions. The FCT remains stable, demonstrating that the types of flow, especially long flows, did not impact the data center network's performance. It provides the data center with the ability to tune the network traffic patterns to suit the network condition. This implementation has improved the stability of the network and provided a stable system throughout the simulation.

  Figures 4, the flow completion time for all flows. The figure shows the proposed algorithm ensures that the congestion in the data center is minimal while providing full utilization of bandwidth and stability compared to another TCP model. Figure 5 continues to demonstrate the ability of the proposed algorithm to maintain stability in the elephant flows even when the load is increased compared to other TCP models where the FCT increases as the load increases. Figure 6 shows the reaction of mice flows in the simulation because most of the time in the data center, the FCT of Mice flows is affected by elephant flows. The proposed algorithm manages to maintain the flow completion time by providing a firm and stable completion time compared to DCTCP, D2TCP, and D3.

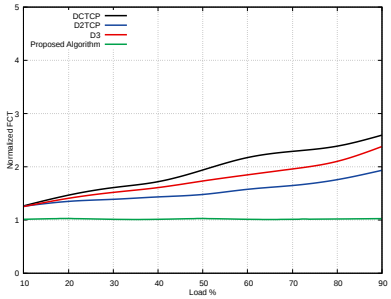

Fig. 4. FCT of All flows

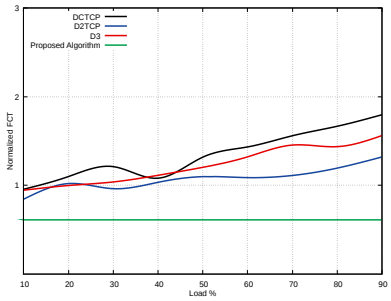

Fig. 5. FCT of Mice flows

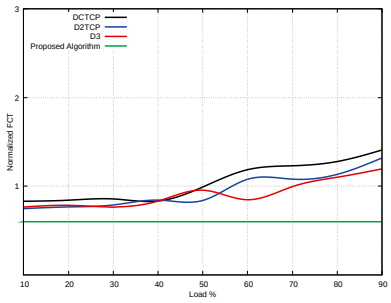

Fig. 6. FCT of Elephant flows

- **Flow completion time within their deadline**: Figure 7 show how the proposed algorithm performed compared to other TCP models. The newly improved priority calculation has significantly increased deadline completion. With this modification, the proposed algorithm met 15% more deadline than DCTCP for the flows destined to cross-Cluster in the simulated data center. Flows destined In-Rack, In-Cluster, and cross-Cluster, are realized due to the contribution of the mechanism of not allowing flows to start if the cant finish their flows in the deadline.

  Most of the algorithms like D3 estimate FCT and deadline but the proposed algorithm architecture makes sure the flow meets deadlines by ensuring first no new flow can be initiated if it can't finish within the timeframe, then if the flow is destined in the area that has been reported to have congestion start with slow start threshold. Furthermore, all the current flow utilizes data left and

remaining time to change the priority so that they can meet the deadline. Though these approaches, the proposed algorithm has been able to ensure all deadlines of the flows destined in different areas in the data center are met.

Multipath and ECN help the proposed algorithm to have the ability to self-adjust, and the congestion table helps to understand the flows and its status. The experiment results demonstrated that it is possible to meet deadlines and FCT of both long and short-flows. Furthermore, the policy helps in the case of congestion indications for a flow, the rate and priority are controlled, and deadlines are met.

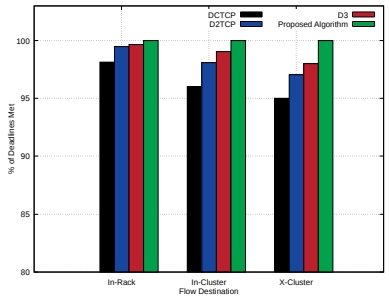

Fig. 7. Flow completion time within their deadline.

VIII. Conclusion and Future Works

In conclusion, the rapid growth of technology and the introduction of 5G networks have increased the demand for high-quality services that can cope with users' demands. The growth of interactive applications in data centers requires low-latency networks, and congestion control plays a crucial role in ensuring that the network runs smoothly. The proposed algorithm in this paper improves congestion control and flow completion time by using ECN to notify network devices of congestion extent and location. Other technologies like software-defined networks, and wireless transmission technologies like WiFi and LiFi could help to solve some of the current challenges in data center networks, such as over-subscription and congestion. Further research can explore the deployment of these algorithms and protocols using other technologies like LiFi to improve the overall health of the data center network.

References

[1] "A survey on TCP Incast in data center networks - Ren - 2014 - International Journal of Communication Systems - Wiley Online Library." [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/dac.2402

[2] Z. Zhao, Z. Jiang, C. Lu, Y. Cai, and J. Bi, "A Congestion Control Algorithm for Datacenters," in *2013 IEEE Eighth International Conference on Networking, Architecture and Storage*, Jul. 2013, pp. 98–104.

[3] E. Baccour, S. Foufou, R. Hamila, and A. Erbad, "Green data center networks: a holistic survey and design guidelines," in *2019 15th International Wireless Communications Mobile Computing Conference (IWCMC)*, June 2019, pp. 1108–1114.

[4] "Cisco Global Cloud Index: Forecast and Methodology, 2018–2023 White Paper - Cisco." [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html

[5] A. Munir, I. A. Qazi, Z. A. Uzmi, A. Mushtaq, S. N. Ismail, M. S. Iqbal, and B. Khan, "Minimizing flow completion times in data centers," in *2013 Proceedings IEEE INFOCOM*, April 2013, pp. 2157–2165.

[6] M. Alizadeh, S. Yang, M. Sharif, S. Katti, N. McKeown, B. Prabhakar, and S. Shenker, "pfabric: Minimal near-optimal datacenter transport," in *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, ser. SIGCOMM '13. New York, NY, USA: ACM, 2013, pp. 435–446. [Online]. Available: http://doi.acm.org/10.1145/2486001.2486031

[7] S. Osada, D. Izumi, S. Kishimoto, Y. Fukushima, and T. Yokohira, "Backoff algorithms to avoid tcp incast in data center networks," in *2018 International Conference on Information and Communication Technology Convergence (ICTC)*, Oct 2018, pp. 515–520.

[8] S. D. Pawar and P. V. Kulkarni, "Article: A survey on congestion notification algorithm in data centers," *International Journal of Computer Applications*, vol. 108, no. 20, pp. 30–38, December 2014, full text available.

[9] C. So-In, R. Jain, and J. Jiang, "Enhanced forward explicit congestion notification (e-fecn) scheme for datacenter ethernet networks," in *2008 International Symposium on Performance Evaluation of Computer and Telecommunication Systems*, June 2008, pp. 542–546.

[10] J. Bao, J. Wang, Q. Qi, and J. Liao, "ECTCP: An Explicit Centralized Congestion Avoidance for TCP in SDN-based Data Center," in *2018 IEEE Symposium on Computers and Communications (ISCC)*, Jun. 2018, pp. 00 347–00 353.

[11] "Standardization of Low-Latency TCP with Explicit Congestion Notification: A Survey - IEEE Journals & Magazine." [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7839868

[12] C. Guo, H. Wu, K. Tan, L. Shi, Y. Zhang, and S. Lu, "Dcell: A scalable and fault-tolerant network structure for data centers," vol. 38, 10 2008, pp. 75–86.

[13] V. Asghari, R. F. Moghaddam, and M. Cheriet, "Performance analysis of modified bcube topologies for virtualized data center networks," *Computer Communications*, vol. 96, pp. 52–61, 2016.

[14] N. Farrington and A. Andreyev, "Facebook's data center network architecture," in *2013 Optical Interconnects Conference*. Citeseer, 2013, pp. 49–50.

[15] A. Singh, J. Ong, A. Agarwal, G. Anderson, A. Armistead, R. Bannon, S. Boving, G. Desai, B. Felderman, P. Germano *et al.*, "Jupiter rising: A decade of clos topologies and centralized control in google's datacenter network," *ACM SIGCOMM computer communication review*, vol. 45, no. 4, pp. 183–197, 2015.

[16] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta, "Vl2: A scalable and flexible data center network," *SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 4, pp. 51–62, Aug. 2009. [Online]. Available: http://doi.acm.org/10.1145/1594977.1592576

[17] A. Munir, T. He, R. Raghavendra, F. Le, and A. X. Liu, "Network scheduling aware task placement in datacenters," in *Proceedings of the 12th International on Conference on emerging Networking EXperiments and Technologies*, 2016, pp. 221–235.

[18] A. Roy, H. Zeng, J. Bagga, G. Porter, and A. C. Snoeren, "Inside the social network's (datacenter) network," in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, 2015, pp. 123–137.

[19] M. Noormohammadpour and C. S. Raghavendra, "Datacenter traffic control: Understanding techniques and trade-offs," *CoRR*, vol. abs/1712.03530, 2017. [Online]. Available: http://arxiv.org/abs/1712.03530

[20] K. Ramakrishnan, S. Floyd, D. Black *et al.*, "The addition of explicit congestion notification (ecn) to ip," 2001.

[21] J. Postel *et al.*, "Internet protocol," 1981.

[22] W. Bai, K. Chen, L. Chen, C. Kim, and H. Wu, "Enabling ecn over generic packet scheduling," in *Proceedings of the 12th International on Conference on emerging Networking EXperiments and Technologies*, 2016, pp. 191–204.

[23] K. Nichols and V. Jacobson, "Controlling queue delay," *Communications of the ACM*, vol. 55, no. 7, pp. 42–50, 2012.

[24] Y. Zhu, M. Ghobadi, V. Misra, and J. Padhye, "Ecn or delay: Lessons learnt from analysis of dcqcn and timely," in *Proceedings of the 12th International on Conference on emerging Networking EXperiments and Technologies*, 2016, pp. 313–327.

[25] T. R. Henderson, M. Lacage, G. F. Riley, C. Dowell, and J. Kopena, "Network simulations with the ns-3 simulator," *SIGCOMM demonstration*, vol. 14, no. 14, p. 527, 2008.

[26] "NS3 — Tutorial." [Online]. Available: https://www.nsnam.org/docs/tutorial/html/getting-started.html#overview

[27] N. Dukkipati and N. McKeown, "Why flow-completion time is the right metric for congestion control," *ACM SIGCOMM Computer Communication Review*, vol. 36, no. 1, pp. 59–62, 2006.

[28] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data center tcp (dctcp)," in *Proceedings of the ACM SIGCOMM 2010 Conference*, ser. SIGCOMM '10. New York, NY, USA: ACM, 2010, pp. 63–74. [Online]. Available: http://doi.acm.org/10.1145/1851182.1851192

[29] A. Shymyrbay, A. Zhanbolatov, A. Amankhan, A. Bakambekova, and I. A. Ukaegbu, "Meeting Deadlines in Datacenter Networks: An Analysis on Deadline-Aware Transport Layer Protocols," in *2018 International Conference on Computing and Network Communications (CoCoNet)*, Aug. 2018, pp. 152–158.

[30] B. Vamanan, J. Hasan, and T. Vijaykumar, "Deadline-aware datacenter tcp (d2tcp)," *SIGCOMM Comput. Commun. Rev.*, vol. 42, no. 4, pp. 115–126, Aug. 2012. [Online]. Available: http://doi.acm.org/10.1145/2377677.2377709