A Network-centric TCP for Interactive Video Delivery Networks (VDN)

MD Iftakharul Islam and Javed I Khan
Kent State University
Email: mislam4@kent.edu, javed@cs.kent.edu

***Abstract*—Interactive video streaming requires very low latency and high throughput. Traditional latency based congestion control algorithm performs poorly in fairness. This results in very poor video quality to adaptive video streaming. Software defined networks (SDN) enables us to solve the problem by designing a network controller in the routers. This paper presents a SDN-centric TCP where sending rate of the network is calculated from the network rather than the host. Routers along the path uses a *proportional integral (PI)* controller to calculate the sending rate in order to reduce the queuing delay. The routers also divide the available throughput fairly among the flows. The network based controller is found more effective than the sender/receiver based controller in reducing latency and providing fairness. NC-TCP has been designed for interactive video delivery network (VDN) where the interactive video flows compete among themselves. Such differentiated service obviates the need for *TCP-friendliness*. We have implemented NC-TCP in Linux kernel. We have evaluated NC-TCP in Mininet for an interactive video streaming application. Our experimental results shows that NC-TCP outperforms delay-based congestion control in an interactive VDN.**

***Index Terms*—Router-centric Congestion Control; SDN; Interactive Video Streaming; XCP; Linux Kernel Networking**

I. Introduction

Interactive video streaming over the Internet are expected to behave in a TCP-friendly manner. For example, Skype uses a TCP-friendly congestion control algorithm for video conferencing [1]. However competing with TCP flows adversely affect interactive video streaming. Loss-based congestion control of TCP fills up the router buffer (*bufferbloat problem*) which increases delay. Again interactive video streaming also traditionally employs delay based congestion control algorithm which loses throughput to loss-based congestion control algorithm [2]. Recent advances in network virtualization and differentiated service enable us to isolate interactive video traffic from other best effort services. This obviates the need for TCP-friendliness and enables us to design congestion control algorithm where interactive video streaming competes among themselves.

Congestion control for real-time communication aims to minimize the delay. The first proposed delay based congestion control algorithm TCP Vegas [3] used Round Trip Time (RTT) to infer congestion. It has been shown that RTT based congestion control results in lower link utilization in the presence of reverse traffic, which inflates queues in the backward path[2]. To solve this problem, LEDBAT [4] employs one-way delay to rule out the sensitivity to the reverse path congestion. These approach however suffers *latecomer effect* : when two flows share the same bottleneck, the second flow may starve the first one [5]. Recently RTT gradient (change of RTT over time) has been employed to overcome the latecomer effect. Examples of such congestion control are TCP CDG [6] and TCP Igino [7]. TCP CDG regulates the sending rate based on RTT gradient whereas TCP Igino regulates the sending rate based on one-way delay gradient. Google Hangout also uses one-way delay gradient based congestion control algorithm over RTP/RTCP [8].

978-1-5090-6501-1/17/$31.00 © 2017 IEEE

Although delay-gradient based congestion control algorithms have been widely used for interactive video streaming, it has shown that they suffer from fairness issue [9]. It is important to mention that if flows cannot share throughput fairly, it results in poor video quality for the flows. Recent advances in software defined networks (SDN) however has a potential to solve this problem. Programmable switches [10] can provide us with explicit feedback from network. XCP[11] is the first attempt in direction where explicit rate is assigned by the router based on persistent queuing delay and spare capacity of the bottleneck link. It has shown that XCP can be implemented on a router at line-rate [12]. But XCP makes no assumption about the traffic characteristic. In this paper however we designed a network-centric TCP (NC-TCP) for interactive VDN where interactive video streaming competes among themselves. We have utilized the traffic characteristics to optimize throughput allocation. Here routers along the path calculate a sending rate based on PI controller in order to minimize the latency. Routers also divide the throughput fairly among the flows. Our experimental results show that such network centric approach results in small and smooth latency and better fairness in an interactive VDN.

NC-TCP is designed as a rate-based congestion mechanism as opposed to XCP. It uses *TCP-pacing* to regulate the sending rate. That is why, it does not send bursty traffic and it does not send more data than the network is able to deliver. NC-TCP also uses instantaneous queue backlog to calculate the feedback rate whereas XCP calculates the feedback rate based on the standing queue: minimum queue length over a control interval. This is why NC-TCP reacts to congestion quicker than XCP. Our experimental results show that NC-TCP performs better in containing small queue size compared to XCP in an interactive VDN. The main contributions of this paper are as following:

- We have described the importance of TCP for interactive video video network (VDN). We have developed a network-centric TCP where rate is allocated by the routers along the path. We have shown that NC-TCP is stable and fair. We have implemented NC-TCP in Linux kernel.
- Most of the interactive video streaming applications such as Google Hangout uses delay based congestion control

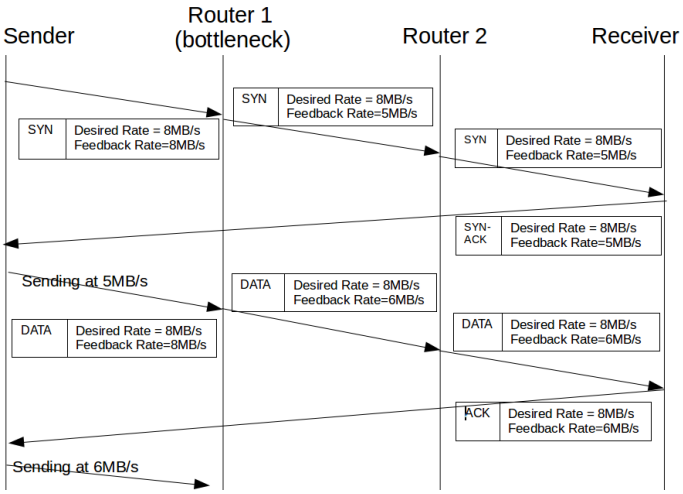


Fig. 1. Work flow of NC-TCP

[8]. However our studies have shown that delay based congestion control algorithm perform very poorly compared to explicit rate control algorithm. We have shown that both NC-TCP and XCP outperforms TCP Inigo [7] which is a one-way delay variant based congestion control algorithm similar to the one used in Google Hangout [8].

II. Why TCP?

Interactive video streaming traditionally uses application layer rate control algorithm over UDP [8]. But this approach has some disadvantages. It forces the application developers to reimplement the common functionality. Different implementations of the congestion control algorithms interact poorly with each other, and the stability of the network is compromised. UDP is also not NAT and firewall friendly which are highly desirable attributes in today's Internet. TCP solves those problems, however it has some disadvantages in the context of interactive video streaming. The ordered delivery of TCP delays some segment delivery if a prior segment is missing (*head-of-line blocking* problem). Retransmission is also not often desirable in interactive video streaming. However it has shown that this problem can be avoided by modifying TCP implementation. Recently TCP Hollywood [13] has been designed as an unordered and time-line oriented TCP. It avoids the *head-of-line blocking* problem by supporting unordered delivery. It also attaches a time-line to data. The data would be ignored after the time-line has been expired. This is how it avoids retransmission. Packet-loss can be further avoided by caching packets on the router. As NICs (network interface controllers) evolved from kbps to Gbps and memory chips from KB to GB, packet loss has become less of an issue.

III. NC-TCP Algorithm

NC-TCP is implemented as a new TCP option. The option consists of two fields: *expected throughput* and *feedback throughput*. TCP Option is the intended mechanism by which TCP can be extended. The workflow of NC-TCP is shown in Figure 1. A sender specifies the required throughput in SYN and DATA packets. The feedback throughput would be set by the routers. The routers allocate a feedback throughput based on the link bandwidth, number of flows and queue occupancy. Receivers copy the feedback throughput to the acknowledgements packets. In this way, the feedback throughput would be returned to the sender. This is how the routers send an explicit feedback throughput to the senders. The feedback throughput would be used as a sending rate in the sender. Traditional TCP uses window based congestion control. Window-based based congestion control produces bursty traffic. Video streaming traditionally uses rate-based congestion control [8]. NC-TCP has been designed as a rate-based congestion control. It uses *TCP-pacing* to regulate the sending rate. That is why, it does not send bursty traffic and it does not send more data than the network is able to deliver.

A video streaming flow may have more than one bottleneck links along the path. This is why a bottleneck router would only assign the feedback rate if the rate is smaller than the feedback calculated by a previous router. This ensures that the sending rate of a sender is adjusted to the most congested link on the path.

IV. Congestion Control

This section uses a *Proportional Integral (PI)* controller in the router to calculate the aggregate feedback throughput based on the queue size. We divide the feedback control system into two parts: Delay Controller and Fairness Controller.

A. Delay Controller

The delay controller is designed based on PI controller. It calculates the aggregate feedback throughput based queue length and link capacity. In PI controller, the aggregate feedback throughput can described by the following equation:

$$\sum r_i(t) = \alpha[c(t) - q(t)] \tag{1}$$

where the $\sum r_i(t)$ is the aggregate feedback throughput, $c(t)$ is the bottleneck link capacity, $q(t)$ is the queue length in bytes and α is the coefficient. We need to determine the value of α for which the system stabilizes. Later we have shown that the system stabilizes for when $\alpha > 0$. The value of α is used to regulate the queuing delay and throughput. There is a trade-off between throughput and queuing delay. Later we have shown that $\alpha = 1$ makes the queuing delay approaches to zero at the same time maximizing the feedback throughput.

B. Fairness Controller

The objective of the fairness controller is to divide the capacity fairly among the flows. Here it uses min-max fairness to divide the available capacity among the flows equally. We can rewrite the equation (1) as $r_i(t) = \frac{\alpha[c(t)-q(t)]}{N(t)}$ where $r_i(t)$ is the feedback throughput for flow i.

C. Stability Analysis

This section shows that NC-TCP always converges to an equilibrium point. The feedback rate set by the router is sent back to the sender via acknowledgment. This is why there is a feedback delay in NC-TCP. Let us assume that the feedback delay is t_f. As the feedback rate becomes the sending rate after t_f,the sending rate at time t can be written as

$$x_i(t) = \frac{\alpha[C - q(t - t_f)]}{N} \tag{2}$$

Without loss of generality similar to all previous works [11], [14], our analysis assumes that all flows have a common, finite, and positive round-trip delay and the link capacity and the number of flows are constant for a session. The queuing-delay gradient can be represented as

$$\dot{q(t)} = \sum x_i(t) - C \tag{3}$$

As flows adapt their sending rate $x_i(t)$ at the same rate, $\sum x_i(t) = N x_i(t)$ for all i. From equation (2) and (3) we get,

$$\dot{q(t)} = N x_i(t) - C \implies \dot{q(t)} = -\alpha q(t - t_f) + C(\alpha - 1) \tag{4}$$

This is an autonomous differential equation which describes the autonomous system. According to the stability theorem of autonomous differential equation, the system is stable if $\ddot{q(t^*)} < 0$ where $q(t^*)$ represents the queuing delay at the equilibrium point. This is why the system is stable if $\alpha > 0$. If the system is perturbed by burst of flows, the $q(t)$ increases. In that case, the queue will be drained at $-\alpha q(t-t_f)+C(\alpha-1)$ rate and eventually reaches the equilibrium point.

At the equilibrium point, $\dot{q(t)} = 0$. Thus from the equation (4), we get $q(t - t_f) = \frac{C(\alpha-1)}{\alpha}$. $\frac{C(\alpha-1)}{\alpha}$ is the queuing delay at the equilibrium point. The value of α controls the queuing delay and feedback throughput. Here we want the queuing delay to be be zero while maximizing the throughput. That is why we set $\alpha = 1$.

D. Encoder Rate Control

One of the main challenges of interactive video streaming is that the encoding rate of video cannot be changed quickly in order to avoid congestion. The video encoder reacts to encoding rate changes over discrete intervals which is usually more than 500 milliseconds. The actual encoder output rate also fluctuates randomly around the input target rate. The actual output rate of a video depends on the content of the video. This section shows how to set the target encoding rate based on the feedback rate.

The encoding rate in an adaptive video streaming is set based on the throughput of the network. Contrary to that, NC-TCP based application sets the target rate based on the feedback rate of the network. NC-TCP uses a rate-based congestion control mechanism to regulate the sending rate. It uses *TCP-pacing* to regulate the sending rate. The sending rate is set to the feedback throughput. This is why NC-TCP does not send more data than network would be able to handle. As a result, although it results in a very small queuing delay, packets

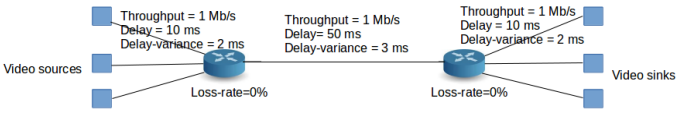


Fig. 2. Topology

may wait in the TCP's sending buffer (congestion window) to be scheduled to sent out. This increases the RTT. This is why RTT produced by the NC-TCP is not exactly same as the propagation delay. Rather RTT/RTT_{min} indicates how much encoder has overshot with respect to the network throughput. Here RTT_{min} is the minimum RTT which is considered as the *propagation delay*. We set the encoder target rate as $R_t(t) = \frac{r_i(t)}{\frac{RTT}{RTT_{min}}}$ where $r_i(t)$ is the feedback throughput received from the network. Note that $\frac{RTT}{RTT_{min}} \geq 1$. This is why the target encoding rate is less than the feedback throughput. This is how NC-TCP based appliactions set the target encoding rate based on the network-feedback and past history of overshooting.

V. Implementation

We have implemented NC-TCP and XCP in Linux kernel[1]. We also have found that TCP Inigo Linux kernel implementation in the Internet [2]. Our software architecture for the both NC-TCP and XCP includes two parts: NC-TCP host and NC-TCP router.

NC-TCP end point is implemented by adding a new option to Linux TCP stack. In contrast to the traditional TCP, NC-TCP uses *TCP-pacing* to regulate the sending rate. Here the sending rate is set to the feedback rate received from the network. The congestion window is also set $\frac{r_i(t)*RTT}{MSS}$ where MSS represents the *maximum segment size*. Linux TCP stack does not have a native *TCP-pacing* implementation. TCP-pacing has been implemented using *fq* (fair queue) Qdisc (queuing discipline). Qdisc is the traffic control mechanism in the Linux kernel. XCP on the other hand uses window based congestion control. XCP sender calculates the congestion window based on the feedback received from the XCP router.

NC-TCP router has been implemented in the Linux router (enabling *ip_forward* option) as a new Qdisc. The Qdisc kernel module inspects the NC-TCP option and sets a feedback rate based on the delay and fairness controller. XCP router is also implemented in the same manner.

VI. Experimental Setup

We have created a topology as Figure 2 using Mininet [15]. We have used *NetEm* Qdisc to set the delay and *htb* (hierarchical token bucket) Qqisc to set the throughputs on the network interfaces. We also developed a GStreamer based application[3] to stream video from the sources. The source nodes act as a TCP server and the sink nodes act as a TCP client. Once the

[1]https://github.com/tamimcse/Linux
[2]https://github.com/systemslab/tcp_inigo
[3]https://github.com/tamimcse/gst-streamer

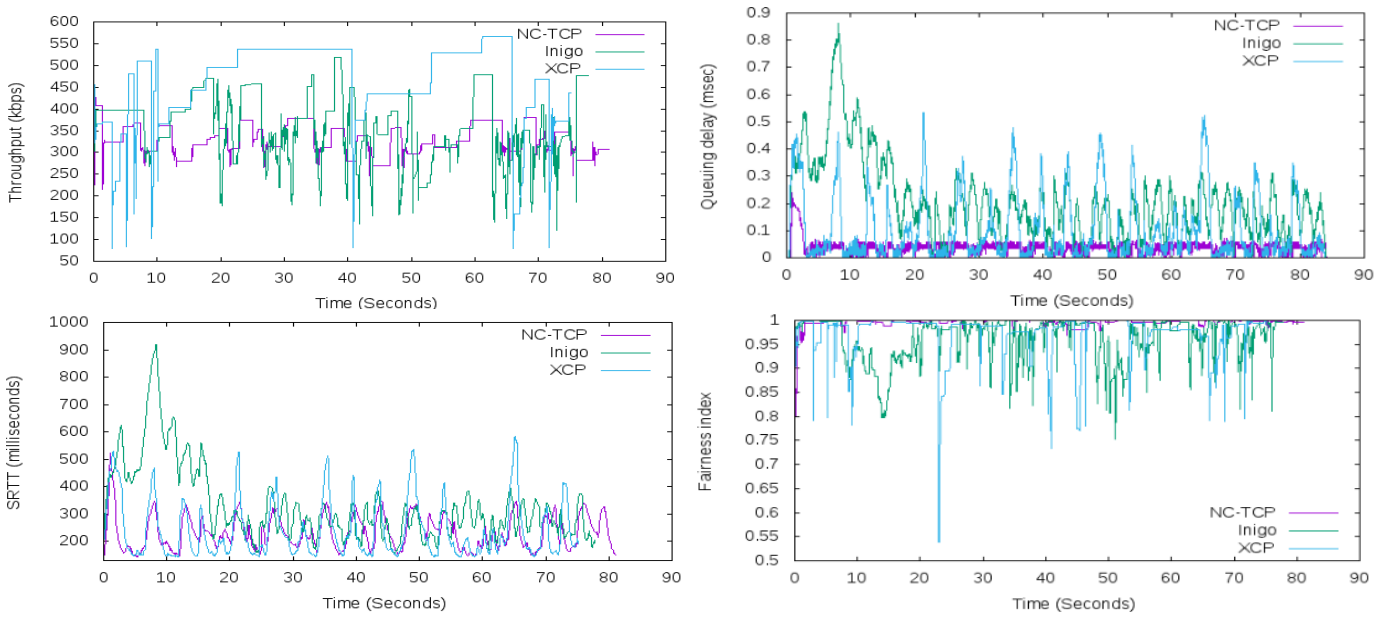


Fig. 3. Comparison among NC-TCP, XCP and TCP Inigo in an interactive VDN

connection has been established, the server streams video to the client. In this experiment, we used a clip from Big Buck Bunny[4]. The source application produces *512X340* video at 30 fps. This video is encoded into H.264 format. Here we used adaptive bitrate streaming where the bitrate of the video is adapted based on the network throughput. We used one-pass constant bitrate (CBR) encoder to produce the desired bitrate. Such one-pass encoder is traditionally used for real-time streaming. The GStreamer source updates the encoding rate in every 500 milliseconds. It uses *getsockopt* system call to get feedback throughput from the TCP stack. It updates the encoding rate based on the feedback. On the receiving side, each node has a GStreamer sink application that plays the video as soon as a new NAL (network abstraction layer) unit is received.

VII. Evaluation

Loss-based TCP such as TCP CUBIC, the default congestion control in Linux, results in substantial delay in the router which is not acceptable for interactive traffic. We found that TCP CDG [6] and TCP Inigo [7] to cause very small queuing delay which make them suitable for interactive video streaming. Moreover we found that TCP Inigo outperforms TCP CDG in achieving higher throughput and lower latency. This is why, in this section we compared the performance of NC-TCP with that of TCP Inigo. This section also compared NC-TCP with XCP [11]. XCP and RCP [14] are explicit rate control mechanism which are promising alternative to TCP. RCP is designed to minimize the flow completion time but it results in higher queuing delay [14]. This is why RCP is not suitable for interactive video streaming. XCP on the other hand results in very small queuing delay. This is why we also compared NC-TCP to that of XCP. This section compared the performance of one-way flows in order to avoid the reverse path affect.

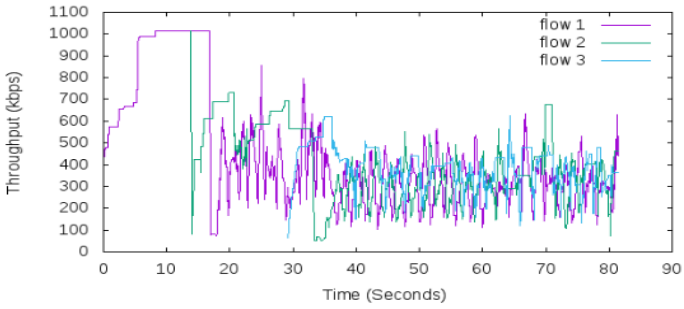
The primary evaluation criteria in this section are the queuing delay on the bottleneck router and the throughput fairness. As NC-TCP, XCP and TCP Inigo fully utilize the bottleneck link, higher fairness indicates that all the flows get higher throughput while lower fairness indicates that some flows get higher throughput while other flows are not receiving higher throughput from the bottleneck link.

Single bottleneck link: We have conducted our experiment on the topology describe in Figure 2. In the experiment, we have started three flows simultaneously and let it run for 80 seconds. We have run the experiment for NC-TCP, XCP and TCP Inigo separately. We have shown the throughput, queuing delay, round-trip time and Jain's fairness index in Figure 3. We have modified the *tcp_probe* kernel module to record throughput and RTT on the sender. Here *tcp_probe* takes sample as soon as the congestion window changes. The network-controller we developed also records the queue backlog using Linux *proc* file system.

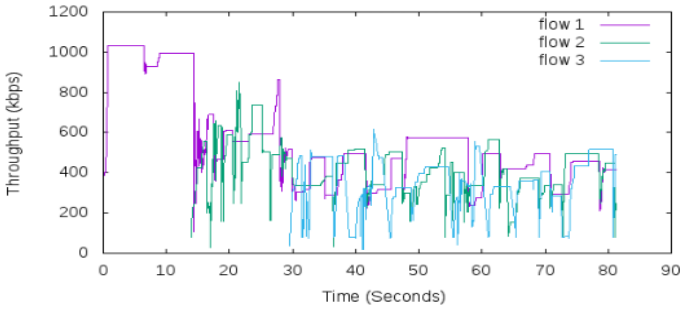
The Figure 3 shows that both NC-TCP and XCP produces smooth throughput. Such smooth throughput of is very suitable for adaptive video streaming. On the other hand, the throughput of TCP Inigo fluctuates very rapidly compared to NC-TCP and XCP.

The Figure 3 has also shown the queuing delay of the bottleneck router. The queuing delay has been calculated using $\frac{q(t)}{B}$ where B is the bottleneck throughput. The figure shows that NC-TCP outperforms XCP and TCP Inigo in queuing delay. TCP Inigo regulates the sending rate based on the delay-gradient observed on the receiver. It has no real network information. It rather tries to estimate the extent of congestion based on the implicit information, the delay observed on the

[4]http://bbb3d.renderfarming.net/download.html



(a) Three TCP Inigo flows 15 seconds apart



(b) Three NC-TCP flows 15 seconds apart

Fig. 4. NC-TCP and TCP Inigo self-fairness comparison

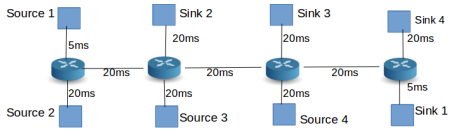


Fig. 5. Parking lot topology with multiple bottleneck links

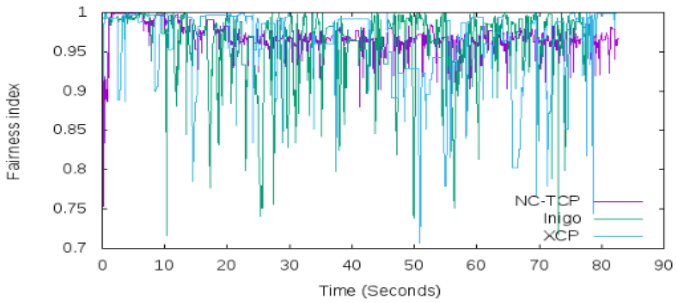


Fig. 6. Evaluating RTT fairness for three flows with different RTTs in interactive VDN

receiver. NC-TCP on the other hand regulates the sending rate based on the actual congestion information. This is why NC-TCP performs significantly better than TCP Inigo. The figure also shows that XCP produces spikes in queuing delay. This is because it uses standing queue in rate calculation. Standing queue is the minimum queue backlog during a control interval. XCP also uses window-based congestion control which causes bursty traffic in the bottleneck router. NC-TCP on the other hand uses instantaneous queue backlog in rate calculation. This is why it reacts sooner to congestion than XCP. NC-TCP also use *TCP-pacing* to regulate the sending rate. This is why, it does not send bursty traffic to network.

The Figure 3 also compared the smoothed RTT (SRTT) experienced by the protocols. It shows that NC-TCP outperforms TCP Inigo and XCP in reducing SRTT. Note that, average RTT in our experimental test-bed is 140 milliseconds. The round-trip time produced by NC-TCP is also close to 140 milliseconds. It is also noteworthy that although queuing delay in NC-TCP is close to zero, the RTT is not equal to the propagation delay. This is because NC-TCP segments may wait in the sender buffer before being scheduled by the *TCP-pacing* Qdisc.

Finally the Figure 3 has compared the Jain's fairness index for the protocols. The Jain's fairness index has been calculated using the formula $\frac{(\sum_{i=1}^{n} x_i)^2}{n\sum_{i=1}^{n} {x_i}^2}$ where x_i is the throughput of flow i and n is the total number of flows. Higher fairness index indicates that the protocol is better in fairness. The figure shows that both NC-TCP and XCP outperforms TCP Inigo in fairness. It is noteworthy that although XCP exhibits occasional downward spikes in fairness index, it performs very fairly most of the time. Both NC-TCP and XCP divides the bottleneck throughput equally among the flows. This results in very high fairness index. On the contrary, TCP Inigo results in very fluctuating queuing delay which cause the congestion window to fluctuate even more. As a result, TCP Inigo performs poorly in fairness compared to XCP and NC-TCP. Note that low fairness results in lower throughput to some flows which results in lower QoE for interactive video streaming.

Self-fairness: Here we have examined the self fairness of NC-TCP and TCP Inigo. We have run the experiment again where each flow is 15 seconds apart. Figure 4 shows three flows competing for 1Mbps bottleneck link. Figure 4a shows that without the present of other flows, TCP Inigo can utilize the available throughput. But it performs poorly when it starts competing with other flows. It changes the congestion window rapidly based on the queuing delay caused by other flows. This is why throughput also fluctuates rapidly. Contrary to that, NC-TCP divides the available throughput fairly among the flows. This is why NC-TCP flows get fair throughput from the bottleneck link. Figure 4b shows that NC-TCP also able to utilize the available throughput in the absence of other flows. It also shows that when three NC-TCP shares a bottleneck link, they get nearly fair amount of throughput from the bottleneck link. The throughput also does not fluctuates as much as TCP Inigo which results in better QoE for adaptive video streaming.

RTT-fairness: The throughput allocation of NC-TCP does not depend on the RTTs of the flows. To evaluate the RTT-fairness of NC-TCP, we have modified the topology in Figure 2 such that the three different flows have different RTTs: 80ms, 100ms and 120ms. Figure 6 the fairness index for that topology. It shows that NC-TCP is fair to flows with different RTTs. NC-TCP is also able to contain the queuing delay in the

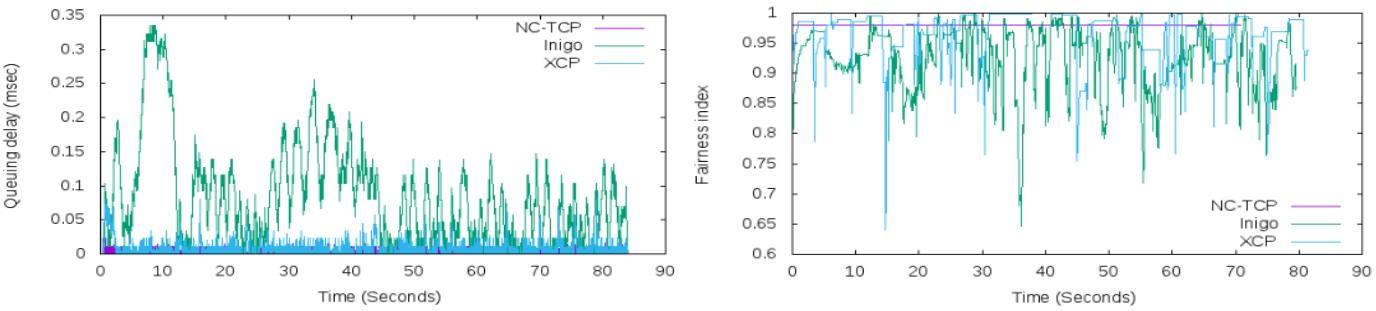


Fig. 7. Comparison among NC-TCP, XCP and TCP Inigo in an interactive VDN with multiple bottleneck links

presence of different RTT flows. The queuing delay produced in this case is same as Figure 3. This is why even though the feedback-delays are different for each flow, they receive smooth feedback throughput. On the other hand, XCP and TCP Inigo exhibit spikes in queuing delay. This is why they also exhibit variability in fairness index.

Multiple bottleneck links: Figure 5 shows a topology with multiple bottleneck links and the flows. Here the links connecting two routers are the bottleneck links. The links connecting a router and a host are the access links. The throughput of bottleneck and access links are 660kbps and 1024kbps respectively. The propagation delays of the links are shown in the figure. We also have introduced 2ms and 3ms delay variance on the access links and bottleneck links respectively. We conducted the same experiment as before on this topology. Here four flows are started simultaneously on the source nodes. We let the experiment run for 80 seconds. Figure 7 compared performance of NC-TCP to that of TCP Inigo and XCP. The figures shows that both NC-TCP and XCP is very effective in containing queuing delay whereas Inigo results in very high queuing delay in the presence of multiple bottlenecks. NC-TCP also outperforms XCP and Inigo in fairness index.

Differential fairness: Network-centric congestion control has a major advantage in this regard. As a router is aware of the throughput requirement of each flow passing through it, it can apply differential fairness in throughput allocation. NC-TCP option has *expected throughput* field which would be used by the senders to specify their throughput requirements. Bottleneck router can apply weighted fairness based on the *expected throughput* in allocating throughput to each flow.

Visual Interruption: Visual interruption is the situation where video player on a receiver enters into a buffering mode. Such interruption degrades the QoE significantly. Table I shows the number of visual interruptions experienced for different topologies. It shows that both NC-TCP and XCP performs significantly well compared to TCP Inigo.

TABLE I. Number of visual interruptions

	Single bottleneck	Multiple bottleneck
TCP Inigo	2	20
XCP	0	0
NC-TCP	0	0

VIII. Conclusions

This paper presents a router-centric TCP (NC-TCP) for interactive video delivery network (VDN). Our experimental results show that NC-TCP produces very smooth and small queuing delay and fair throughput in an interactive VDN. Currently NC-TCP has been implemented in Linux Kernel to be executed on a CPU. Implementation of NC-TCP on a programmable switch will be needed in order for real world deployment.

References

[1] X. Zhang, Y. Xu, H. Hu, Y. Liu, Z. Guo, and Y. Wang, "Profiling skype video calls: Rate control and video quality," in IEEE INFOCOM, 2012, pp. 621–629.
[2] L. A. Grieco and S. Mascolo, "Performance evaluation and comparison of Westwood+, New Reno, and Vegas TCP congestion control," ACM SIGCOMM CCR, vol. 34, no. 2, 2004, pp. 25–38.
[3] L. S. Brakmo and L. L. Peterson, "TCP Vegas: End to end congestion avoidance on a global Internet," IEEE Journal on selected Areas in communications, vol. 13, no. 8, 1995, pp. 1465–1480.
[4] D. Rossi, C. Testa, S. Valenti, and L. Muscariello, "LEDBAT: The New BitTorrent Congestion Control Protocol." in IEEE ICCCN, 2010.
[5] G. Carofiglio, L. Muscariello, D. Rossi, and S. Valenti, "The quest for LEDBAT fairness," in IEEE GLOBECOM, 2010, pp. 1–6.
[6] D. A. Hayes and G. Armitage, "Revisiting TCP congestion control using delay gradients," in IFIP Networking. Springer, 2011, pp. 328–341.
[7] A. G. Shewmaker, C. Maltzahn, K. Obraczka, S. Brandt, and J. Bent, "Tcp inigo: Ambidextrous congestion control," in IEEE ICCCN, 2016.
[8] H. Lundin, S. Holmer, and H. Alvestrand, "A google congestion control algorithm for real-time communication on the world wide web," IETF Informational Draft, 2012.
[9] L. De Cicco, G. Carlucci, and S. Mascolo, "Experimental investigation of the google congestion control for real-time flows," in ACM SIGCOMM workshop on Future human-centric multimedia networking, 2013, pp. 21–26.
[10] P. Bosshart, G. Gibb, H.-S. Kim, G. Varghese, N. McKeown, M. Izzard, F. Mujica, and M. Horowitz, "Forwarding metamorphosis: Fast programmable match-action processing in hardware for SDN," in ACM SIGCOMM CCR, vol. 43, no. 4, 2013, pp. 99–110.
[11] D. Katabi, M. Handley, C. Rohrs, and M.-l. I. Tellabs, "Internet Congestion Control for Future High Bandwidth-Delay Product," in ACM SIGCOMM, 2002.
[12] N. K. Sharma, A. Kaufmann, T. Anderson, C. Kim, A. Krishnamurthy, J. Nelson, and S. Peter, "Evaluating the Power of Flexible Packet Processing for Network Resource Allocation," in USENIX NSDI, 2017.
[13] S. McQuistin, C. Perkins, and M. Fayed, "TCP Hollywood: An unordered, time-lined, TCP for networked multimedia applications," in IFIP Networking, 2016, pp. 422–430.
[14] N. Dukkipati, "Rate Control Protocol (RCP): Congestion control to make flows complete quickly," Ph.D. dissertation, Stanford University, 2007.
[15] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: rapid prototyping for software-defined networks," in Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks, 2010.