Congestion Management for Ethernet-based Lossless DataCenter Networks

Pedro Javier Garcia[*1], Jesus Escudero-Sahuquillo[†1], Francisco J. Quiles [‡1] and Jose Duato[§2]

[1]Department of Computing Systems, University of Castilla-La Mancha, Spain

[2]Department of Computing Engineering, Technical University of Valencia, Spain

February 4, 2019

DCN: 1-19-0012-00-ICne

Abstract

This paper describes congestion phenomena in lossless data center networks and its negative consequences. It explores proposed solutions, analyzing their pros and cons to determine which are suited to the requirements of modern data centers. Conclusions identify important issues that should be addressed in the future.

Purpose

This contribution to IEEE 802 Nendica is intended to stimulate discussion on enhancements to the IEEE 802 Nendica Report on The Lossless Network for Data Centers. The content is proposed for incorporation into a revised Nendica Report.

[*]pedrojavier.garcia@uclm.es

[†]jesus.escudero@uclm.es

[‡]francisco.quiles@uclm.es

[§]jduato@disca.upv.es

Contents

1 Introduction

Nowadays, more and more latency-sensitive applications such as deep learning, real-time big-data analysis, cloud HPC, high frequency trading, etc. are being deployed in data centers (DCs). Thanks to the advances in interconnect technologies, the server has cut its latency by using RDMA technologies instead of traditional mechanisms. Among them, RDMA over Converged Ethernet v2 (RoCEv2) [1] is the mainstream. But RoCEv2 needs a lossless network because of its poor performance retransmission mechanism (see Section 3). Specifically, if one packet of a message is dropped, all the packets of that message will be retransmitted, thus wasting network bandwidth. If the retransmission mechanism is improved, for example, by implementing a TCP-like retransmission mechanism, the RoCEv2 protocol will become too complicated. As a consequence, the latency will increase. Retransmission also consumes additional bandwidth by transmitting again packets that were already transmitted.

In order to provide the lossless property, current Ethernet-based DC networks (DCNs) rely on Priority-based Flow Control (PFC) [2] to implement flow control (see Section 3.1). However, although PFC prevents packets from being discarded, it introduces other problems such as congestion propagation and the derived HOL blocking [3] (see Section 3.2), that may degrade dramatically the network performance.

In general, congestion appears when several flows arriving to a switch through different ingress ports contend persistently for being forwarded through the same egress port. If the egress port cannot forward these packets at the requested aggregated rate, packets belonging to the contending flows will accumulate in the storage queues of the ingress ports, making them congested and (in lossless networks) eventually activating the flow control mechanism (PFC or others), that will then propagate congestion upstream through backpressure. This congestion propagation may end up clogging any number of internal network paths (which is called a “congestion tree” [4], see section 2), then slowing down the advance of packets through the network. Note that even flows not contributing to congestion will be affected if they share queues with contributor flows, as the latter may produce HOL blocking to the former. In summary, congestion propagation may prevent the entire network from achieving high-throughput and low-latency.

The problems derived from congestion must be mandatorily addressed because this phenomenon is likely to appear in DCNs, due to the nature of the services that they provide and to the traffic patterns that they generate in the network. Specifically, DCNs are likely to support “many-to-one” communications, hence they are prone to suffer the incast problems [5], that, in general, happen when many sources send packets simultaneously to the same destination. In these cases, the packets sent from all these sources will tend to saturate the queues of the switches where the packet flows meet. This may happen, for instance, when the leaf worker-nodes in a parallel-application hierarchy return their answers to the upper aggregators. Note that, in lossy networks, the queues oversubscription would be solved by packet dropping, but in lossless networks packets will accumulate at the queues, that will eventually propagate congestion by flow control, generating a congestion tree. Nevertheless, and depending on the traffic pattern derived from the specific applications supported by the system, congestion may appear also due to contention between packet flows addressed to different destinations, which is usually called “in-network” congestion, as it is always derived from contention to access a port connected to a switch, i.e. to the inner network.

In order to deal with congestion and their negative consequences, Explicit Congestion Notifications (ECN) [6] may be deployed combined with PFC, so that congestion is notified to the sources in order to reduce their rate of packet injection. However, because of the long control loop delay of ECN, congestion propagation may still occur from time to time. Moreover, like other closed-loop mechanisms, ECN frequently introduces oscillations in the sources injection rates that lead to poor link bandwidth utilization and degraded performance. Another flaw of this type of mechanisms is that they may end up sending congestion notifications at a wrong rate to the congestion sources, or sending notifications to the wrong sources, as they are not well suited to properly address some congestion situations, especially in-network ones (see Section 5). Consequently, well-founded concerns exist regarding the large-scale use of PFC and ECN in DCNs.

For all these reasons, new congestion management solutions are strongly needed that, besides being packet lossless, should be also performance lossless. That means that the new solutions should guarantee that the DCN does not drop packets because of congestion and also keep high throughput and low latency. Furthermore, it should eliminate the HOL Blocking problem, be easy to configure, as well as being scalable and traffic-pattern independent. Such a solution would be critical towards achieving scalable high-performance DCNs.

This white paper describes in depth the congestion phenomenon in lossless DCNs, as well as its negative consequences, and explores which solutions proposed to deal with it are really suited to the requirements of modern DC systems, analyzing their pros and cons and pointing to the most important issues that should be addressed in the future.

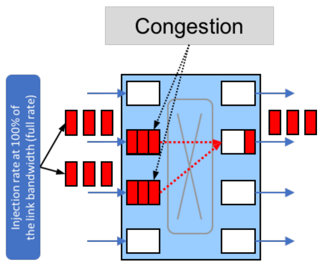
2 Congestion Dynamics in Datacenter Networks

Congestion may actually appear in both lossy or lossless Datacenter networks. However, in general in lossy networks a congested queue will lead to packet dropping, so that congestion cannot spread from that queue to other queues in the network (although discarding packets may eventually introduce other problems that may end up being even more dangerous that congestion, see Section 3). By contrast, as mentioned above, in lossless networks packets are not discarded, then the backpressure of the flow-control mechanisms tends to propagate congestion from the original congested queue to other queues. This forms “congestion trees” whose “root” is the point where congestion originates and whose “branches” are the paths where congestion spreads through. Similarly, the endnodes, or the switches that they are connected to, can be considered as the “leaves” of the tree.

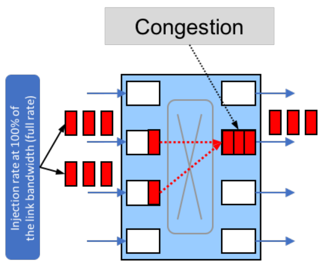
Congestion trees may evolve (i.e. appear, grow and disappear) according to different, and sometimes complex, dynamics. The specific way a congestion tree will evolve depends on several factors. Some of these factors, like the network topology, have an obvious impact in the way the congestion tree can evolve. By contrast, the impact of other factors, in particular the switch architecture or the traffic patterns present in the network, may be more difficult to analyze. In the following subsections the diverse dynamics of congestion trees are studied from the point of view of their appearance, spreading and vanishing, analyzing the main factors that influence these dynamics.

2.1 Appearance of congestion

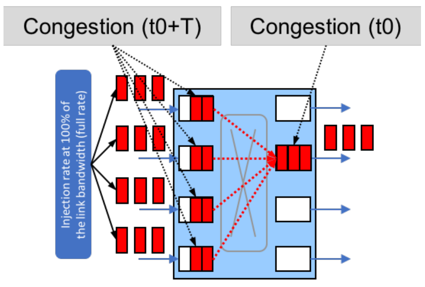
For the sake of an accurate and early detection of congestion appearance, it is important to know how and where congestion may start to be noticeable. In that sense, the architecture of the switch is an essential factor. For instance, in Output-Queued (OQ) switches congestion can appear only at the egress queues since there are no ingress queues. Similarly, in Input-Queued (IQ) switches congestion can appear only at the ingress queues since there are no egress queues. However, in switches with both ingress and egress queues, as it is the case in Combined Input- and Output-Queued (CIOQ) switches, congestion can actually appear either at the ingress or the egress side. In this case, the factor that determines where congestion appears is the switch speedup, together with the number of flows contributing to congestion and their arrival rate.



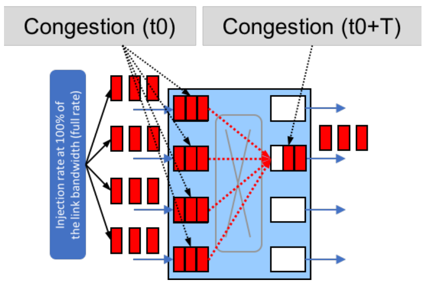
(a) Speedup = 1.



(b) Speedup = 2.



(c) Speedup = 2.



(d) Speedup = 1.5.

Figure 1: Congestion appearance at ingress or egress queues in CIOQ switches with different speedup values.

For instance, Figure 1.a) shows an example where congestion appears at the ingress side. In this case, switches have no internal speedup (i.e. speedup=1, the ingress-egress transmission speed equals link speed), hence if two (or more) incoming flows that arrive to ingress ports at a high rate[1] persistently request the same egress port, packets will be forwarded to that egress port more slowly than they arrive to the ingress ports, then the ingress queues will become congested. It is worth pointing out that in this case the requested egress queue will never become congested, as packets can be forwarded from this queue at the same speed as they arrive to it. By contrast, in Figure 1.b) a switch speedup of two is assumed (i.e. speedup=2, ingress-egress transmission speed is double than link speed), hence if two incoming flows request the same egress port, the internal speedup is enough to forward immediately all the packets of these flows from the receiving ingress ports to the requested egress port, even if the flows arrive to the ingress ports at maximum rate, so that ingress queues will not fill up. On the contrary, in this case packets will be forwarded from the requested egress queue more slowly than they arrive to it, so this queue will become congested. Note that in the last case, if the same situation persists, the ingress queues may be later blocked if the congested egress queue is able to push some type of backpressure over the ingress queues (see Section 3.1). If so, the ingress queues will eventually fill up and become congested.

Congestion may appear also simultaneously at both ingress and egress queues. For instance, in Figure 1.c) the assumed internal speedup is again two, but in this case four flows request the same egress port. If the flows arrive to the ingress ports at maximum rate (i.e. at a rate equal to link speed), both the ingress queues receiving the flows and the requested egress queue will fill up, although the latter will fill at a faster pace than the former. Specifically, as the internal speedup is two and there are four arrival flows, one packet is forwarded from any ingress queue while two packets arrive to that queue, thus one packet accumulates at each ingress queue in this time interval. In the same interval, four packets arrive to the egress queue and two are forwarded from that queue to the downstream switch, thus two packets accumulate at the egress queue in this interval. Note that if the internal speedup were higher than two, the egress queue would become congested also earlier than the ingress queues, but at a faster pace than in the example. On the contrary, in the same traffic scenario, if the number of incoming flows were higher than four, or the internal speedup were lower than two (like in Figure 1.d), where speedup=1.5), the ingress queues would become congested earlier than the egress queue.

Other combinations of internal speedups, number of incoming flows and arrival rates would produce other congestion appearance situations, although they would be variations (mainly regarding the pace at which the queues fill) of the situations analyzed above. It is worth pointing out that the previous analysis is valid for any queue size, as this factor is actually relevant regarding the activation of backpressure upon congestion appearance, but not in congestion generation. Moreover, note that the number of ports in commercial switches has grown in the last years, so increasing the probability of several incoming flows requesting the same egress queue, although maybe not persistently, which must be taken

[1]Note this rate may be as high as link speed, but the same situation will happen even for lower rates.

into account regarding congestion detection. Nevertheless, it must be noted also that both the number of incoming flows and their arrival rate depend mostly on the traffic pattern generated by the applications supported by the system, but not on switch architecture.

Finally, it is important to mention that in all the previous examples, the flows contending for the egress port may be addressed either to the same destination (that may be directly connected to that egress port, or beyond) or to different ones, so any of the situations analyzed may correspond to the generation of either incast or in-network congestion. In any case, if congestion spreads to other (upstream) switches, we will refer to that contended egress port as the root of the generated congestion tree, regardless of whether congestion appears at the ingress or egress sides of the switch, or at both.

2.2 Growth of Congestion Trees

As mentioned above, congestion trees may evolve in very different ways from the initial point of congestion appearance. Indeed, a congestion tree may grow from root to leaves, but also from leaves to root. On the other hand, roots may stay stable or "move" away from or towards the leaves. Similarly, several independent trees may merge into a larger one, or, on the contrary, overlap without merging. In this section we will analyze several examples to illustrate these different ways that congestion trees may grow, as this is relevant regarding how the congestion should be notified and managed once detected.

For instance, Figure 2 shows what is probably the simplest and most comprehensible case of congestion tree generation and evolution. As can be seen, two flows are injected (we assume that they are injected at the maximum speed allowed by the links), then they cross separately several switches, and finally they meet at a switch where they contend to access an egress port. The internal switch speedup assumed in the example is 1.5, hence packets from the two incoming flows will accumulate at both the egress queue of the contended egress port and at the ingress queues of the ports where the flows arrive to[2]. If the flows persist in time and the network is lossless (as we will assume hereafter in all the examples of this section), sooner or later the flow control will be activated, propagating congestion from the ingress queues of that switch to the upstream neighbor switches. Hence, the queues at these switches will become blocked and, if the flow persist, eventually congested, then activating upstream flow control and so on. In this way, the backpressure of flow control will propagate congestion along the paths followed by the two flows, in the opposite sense, i.e. from the root (located at the only contended egress port in the network) upstream along the branches of the tree, so that the closer a queue to the root, the earlier that queue will become congested. Note that, if the flows persist enough, congestion propagation may reach eventually the sources, that then will become the leaves of the tree.

However, congestion may appear first at queues closer to the sources than to the root. This situation is depicted in Figure 3, where we assume again an internal switch speedup of 1.5 and that the injection rate of the flows is 100% of link speed. We assume also

[2]It can be calculated that in this situation the egress queue will fill faster than the ingress queues, but this is not relevant for the example.

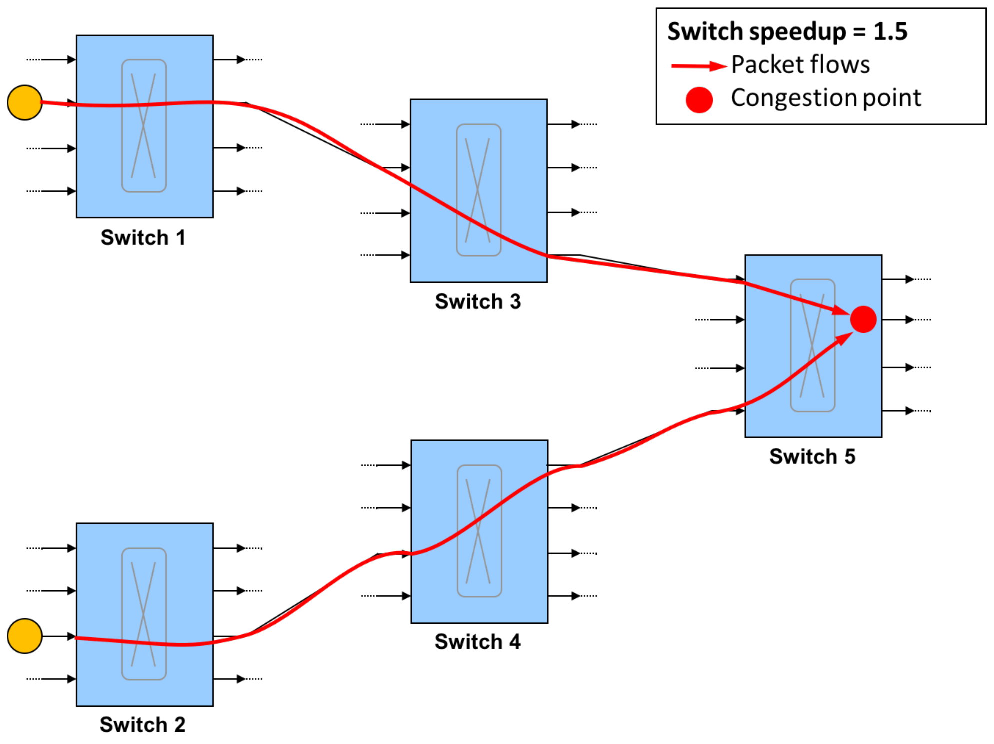


Figure 2: Congestion tree growing from root to leaves

that the injection of all the flows from their respective sources start simultaneously. Note that congestion appears (either at the ingress or the egress queues) first in the switches where the flows meet first, hence in this case congestion appears in several switches before appearing in the one where the root will be finally located. For instance, following the "upper" part of the tree, congestion appear first "locally" in switch 1, next in switch 5, and finally in switch 7, where the two main branches of the final tree meet. This can be considered as a tree growing from leaves to root, or as several "temporary" trees that eventually converge, merge into a larger one, and become subtrees of the new tree. Note also that each subtree has its own particular root, but new roots appear as the flows meet with more flows downstream, which can be seen as the root "moving" downstream.

Note that a similar effect may occur if new flows appear in the network (which is not unlikely to happen in DC networks, although this depends ultimately on the application). For instance, the example shown in Figure 4 illustrates one of these situations. Again, we assume a switch speedup of 1.5 and a flow injection rate of 100% of the link speed. Initially, as shown in the leftmost side of Figure 4, four flows meet (i.e. request the same egress port) at the switch directly connected to their sources (i.e. Switch 1), then they continue advancing along the same path. Hence, a congestion root appears at the contended egress port of that switch, that is the only one affected by this situation. However, a new flow (drawn as a dashed line) is injected later at other switch and meets the "merged" previous

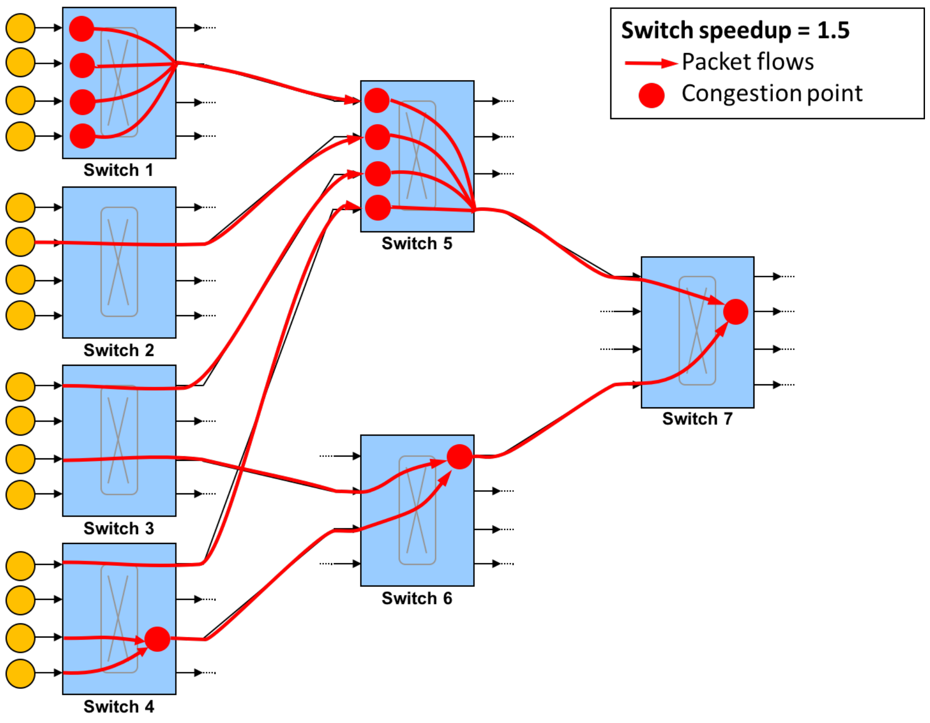


Figure 3: Congestion tree growing from leaves to root, or congestion trees merging.

flows in a downstream switch (all this is depicted in the rightmost side of Figure 4.b). Like in the previous example, a new, larger tree appears whose root is located downstream with respect to the root that appeared first. However, this case cannot be considered as several congestion trees converging and merging, but actually as a new flow expanding an existing congestion tree, which eventually "moves" the root downstream.

It is very important to point out that, in the previous examples, we do not restrict the final destinations of the flows but only the paths that they follow in the network portion depicted in the figures. This means that actually these examples may depict either incast or in-network congestion. In the case of incast, all the flows in the examples would have the same destination, so that they will reach the destination immediately after crossing the root, or they will continue advancing along the same path until reaching destination. In the case of in-network congestion, the flows in the examples would have different destinations, so that they would diverge after meeting at the final root of the tree. An example of in-network congestion derived from congestion trees is depicted in Figure 5, where the flows drawn as solid lines have a different destination from the flows depicted in dashed lines. Note that this example is actually an extended variant of the example shown in Figure 3, and shows that congestion trees may converge and merge into a larger one even if the flows do not have the same final destination. Note also that, in contrast with the main tree, the different subtrees indeed can be considered separately as cases of incast congestion, therefore this example shows also that incast and in-network congestion may

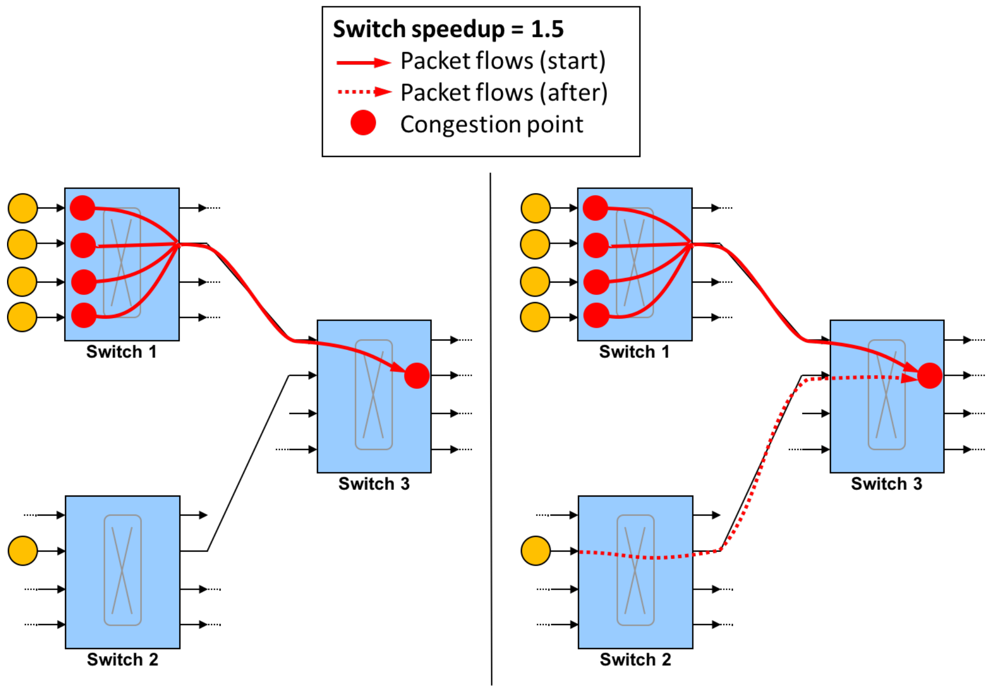


Figure 4: Congestion tree whose root moves "*downstream*".

coexist, which is relevant regarding the countermeasures that should be taken to deal with these situations.

On the other hand, it is possible that two independent congestion trees meet and overlap without merging into a larger tree. This situation is shown in the example of Figure 6, where again we assume switches with an internal speedup of 1.5 and flows injected at 100% of link bandwidth. In the figure, flows belonging to different trees are depicted as different types on lines (solid and dashed, respectively). As can be seen, the two trees meet and overlap partially in the link connecting two intermediate switches (actually, also in the queues located at both extremes of that link). Note that a new local in-network congestion point appears at the switch where the congestion trees meet. However, the two trees do not actually merge because each tree has its own, different root, and none of these roots is located at a point "common" to the two trees. Note also that in this case again incast and in-network congestion coexist. Note finally that, as mentioned in Section 1, ECN-like congestion management mechanisms are not too suited to solve in-network congestion properly, so they may be in trouble in this situation, as their reaction may be wrong or innaccurate (more insights in that sense can be found in Section 5).

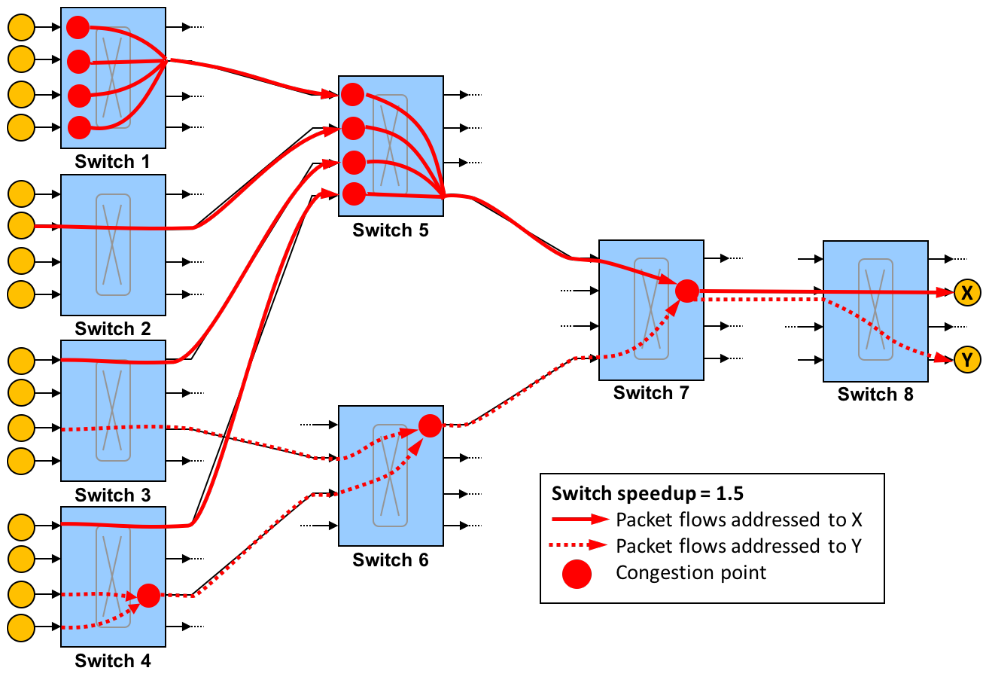


Figure 5: In-network congestion situation generated by two congestion trees converging.

2.3 Congestion Trees Vanishing

Similarly to their growth, congestion trees may vanish in different ways, for instance from leaves to root, or from root to leaves. Congestion trees may vanish due to the sources ceasing or (significantly) reducing the injection of packets, but also due to re-routing of some contributor flows, which may happen, for instance, if congestion-aware load-balancing techniques are used (although this may also create new congestion trees, see Section 4).

It is not difficult to imagine a congestion tree vanishing from leaves to root. For instance, in any of the examples of the previous section, the congestion trees will vanish from leaves to root if all the sources stop injecting packets simultaneously. Indeed, in this situation, as packets advance (i.e. are drained) in the downstream sense, the congested queues that will empty first are those ingress queues that receive packets directly from the sources, next the egress queues receiving packets from these ingress queues will empty, and so on. Finally, the congestion tree will disapear when the queues closest to the root become empty.

However, as mentioned above, congestion trees may vanish also from root to leaves. This case is depicted in Figure 7. Note that the initial tree (leftmost side of Figure 7) is basically the same whose growth is analyzed in Figure 4. At a given moment (rightmost side of Figure 7), only the source of the packet flow drawn in blue stops injecting that flow, hence that flow vanishes and does not contend with the remaining flows to access the

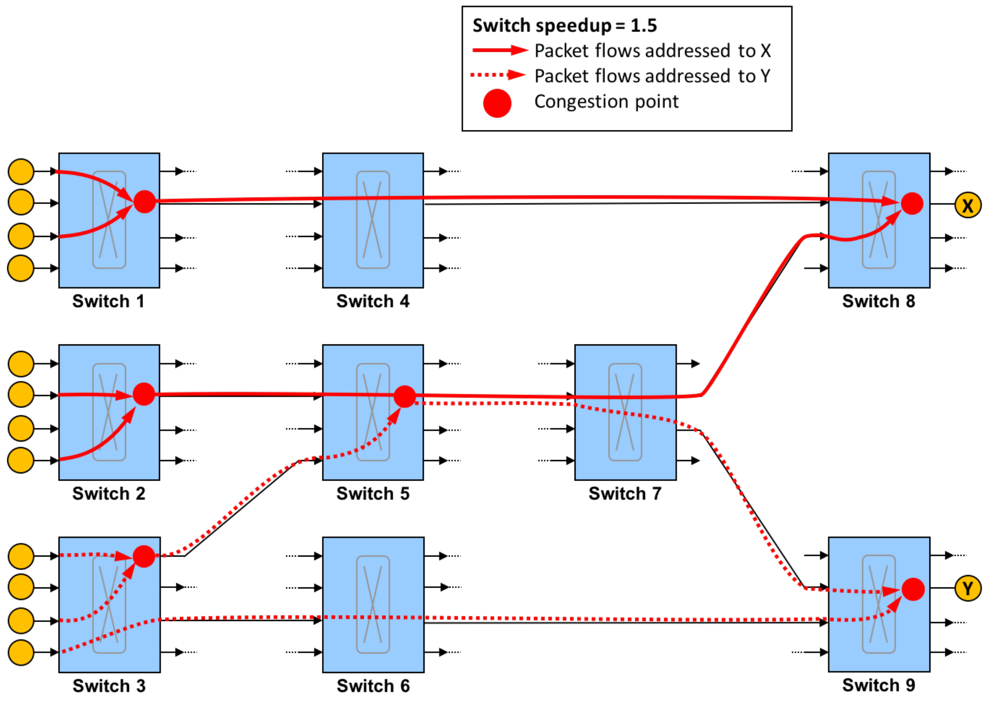


Figure 6: Two independent congestion trees overlapping but not merging.

egress port at switch 3[3]. Therefore, the root of the tree "moves" from that switch to switch 1. It can be seen that, if later the sources of the remaining flows stop injecting them (or the flows are re-routed), the tree would disapear completely, in this case from the original root towards the leaves,

In general, the different ways the congestion trees may vanish should be taken into account by mechanisms that allocate resources where the flows belonging to the tree are isolated, since the deallocation of these resources should be activated upon congestion vanishing (see Section 6).

3 Lossless Transmission and Switch Architecture

Although in the previous sections we have introduced the advantages of lossless networks over lossy ones, as well as the risks and problems derived from lossless transmission, it is worth a deeper analysis in that sense in order to efficiently address such problems, as well as to define the possible implications of lossless transmission on the switch architecture.

[3]Note that the same situation would occur if the flow drawn in blue is re-routed so that it does not reach switch 3.

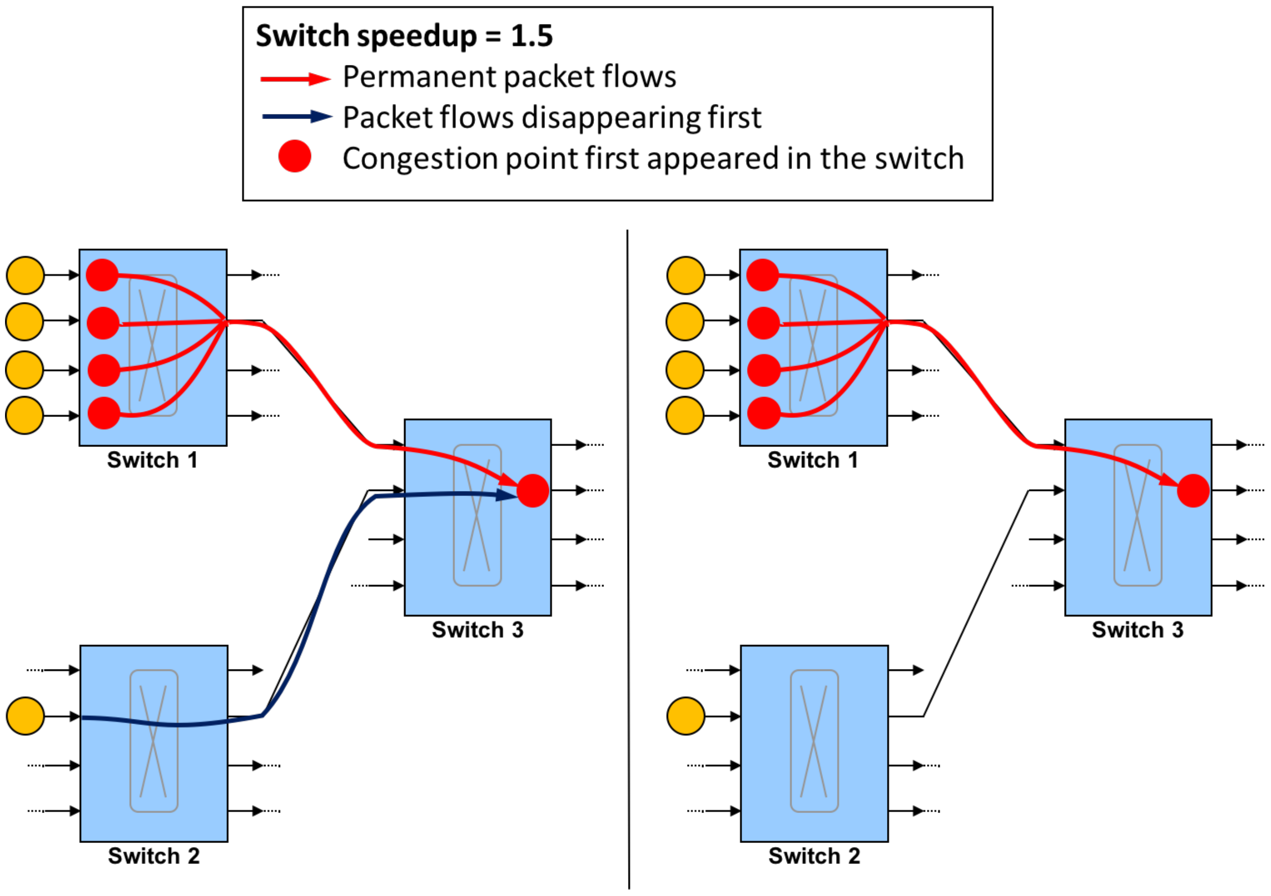


Figure 7: Vanishing of a congestion tree, its root moving "*upstream*"

As mentioned above, the need for lossless transmission in DCNs comes from the high requirements regarding latency of the applications currently supported by these systems. These requirements cannot be satisfied in lossy networks, mainly due to the prohibitive cost in terms of time of packet retransmission upon packet dropping. Although the exact latency introduced by the retransmission mechanism depends on the specific protocol used, in general this overhead is not acceptable. For instance, RoCEv2, the current trend in DCNs, requires the complete retransmission of a message upon the dropping of any of the packets that compose this message, which obviously introduces a huge latency overhead. Note that RoCEv2 is UDP-based, but even if it were enhanced with a more elaborated TCP-like retransmission mechanism (what, on the other hand, would introduce unaffordable complexity), the retransmission overhead would be still significant: the round-trip-time in modern DCNs may be below one millisecond, while the TCP minimal retransmission timeout (RTO) is several orders of magnitude above. In addition to the possible latency overhead, packet dropping may introduce a certain degree of unfairness, especially under bursty traffic, if the discarded packets were injected only from a subset of all the sources contributing to congestion, while the packets injected from other contributor sources may be not discarded. Moreover, note that retransmissions introduce additional load in the network, which consumes bandwidth and may also increase congestion probability. Summing up, lossless transmission seems mandatory in current DCNs, the most natural way to provide it being flow control.

3.1 Lossless Flow-Control Basics

In high-performance networks, lossless flow control is generally provided by using one of two main approaches [7]: credit-based flow control and Stop&Go (also known as Xon-Xoff). Both operate as a transmission protocol between a sender queue in a port and a receiver queue at the next (i.e. downstream) port, so that queue overflow (and so packet loss) is prevented.

In the case of credit-based flow control (see Figure 8) each sender has a number of credits (1) that represents the number of data units that can still be stored at the receiver. The number of credits is decremented (2) whenever a data unit is sent at the sender to the receiver buffer, until the number of credits becomes zero. Therefore, new data units can be transmitted only if there are available credits. On the other hand, whenever a data unit is forwarded downstream from the receiver, the corresponding queue slot is released, then a credit is sent back to the sender (3), that then increments its number of credits (4). This type of flow control is implemented, for instance, in InfiniBand-based networks.

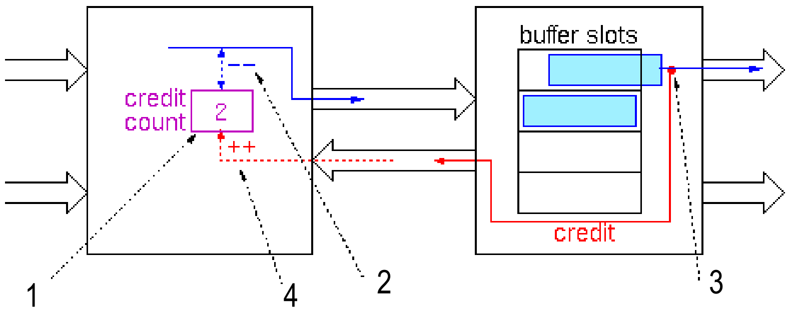
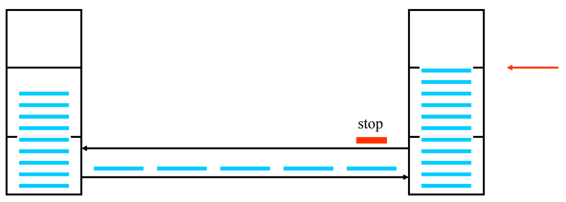
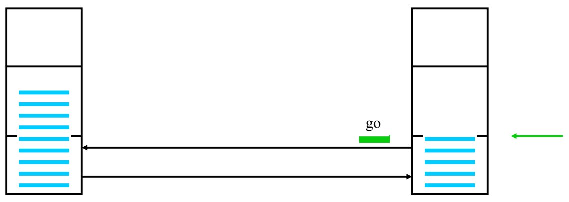


Figure 8: Credit-based flow control. 1: remaining credits, 2: decrease credit counter when sending, 3: release queue slot, 4: send information to increase the credit counter.

On the other hand, in the case of Stop&Go (or Xon-Xoff, or similar) mechanisms, two thresholds are defined (Stop and Go) at every queue, as shown in Figure 9. When the occupancy level of a receiver queue reaches the stop threshold then a signal or control message (Stop notification) is sent upstream to the sender to stop the transmission. Once the receiver queue occupancy decreases below the second (Go) threshold, then another signal or control message (Go notification) is sent upstream to resume the transmission of the data. This is the type of basic approach followed by the Priority-based Flow Control



(a) Stop threshold reached



(b) Go threshold reached

Figure 9: Stop&Go flow control.

(PFC) defined in IEEE 802.1 for Ethernet-based networks, but this approach has been also followed by other similar mechanisms that have been implemented in other interconnect technologies such as Myrinet or Advanced Switching. PFC can actually select which specific class (or priority) of packets will be stopped at the sender queue upon reception of the "Stop" notification, which may be useful in some cases but does not prevent the problems derived from lossless transmission (see next Section).

Note that PFC is defined assuming that the receiver queue is located at an ingress port of a switch and the sender queue is located at the immediate upstream egress port, i.e. the receiver and sender queues are located at different switches (or at a switch and at an endnode) and connected through a link. Hence, PFC is actually an "inter-switch" protocol, so that ingress queues rule regarding the allowance or prevention of transmission from other switches, but leave uncertain the coordination of "intra-switch", ingress-egress transmission. This uncertainty is relevant because, as explained in Section 2.1, congestion may appear inside a switch first at the egress queues, i.e. the egress queues may fill up faster than the ingress queues (this can be seen, for instance, in Figures 1.b) and 1.c). Therefore, it may happen that the congested egress queue fills completely, o almost completely, before the occupancy of the ingress queues reach the level required to activate PFC. If this happens, and no countermeasures are taken, packets requesting that egress port may end up being discarded. Hence, some mechanism is required to coordinate the intra-switch packet forwarding from ingress to egress queues, so that this transmission is also lossless.

In that sense, a possible solution is complementing PFC with Virtual Input Queuing (VIQ), an approach to coordinate the available buffer space at the egress ports with the transmission requests from the ingress queues. Specifically, through VIQ, the egress ports inform the ingress queues of its buffer availability in order to prevent ingress-egress transmissions ending up in packet losses. VIQ can be seen as having, at every egress port, a queue specifically dedicated to each ingress port. A switch architecture implementing VIQ can be seen in Figure 10. Note that, in addition to preventing internal switch packet losses, VIQ allows fair schedulings regarding the packets leaving the switch. However, it is worth clarifying that other solutions could be applied to prevent internal packet losses, hence hereafter we will assume that some mechanism is used for that purpose, without restricting the specific mechanism.

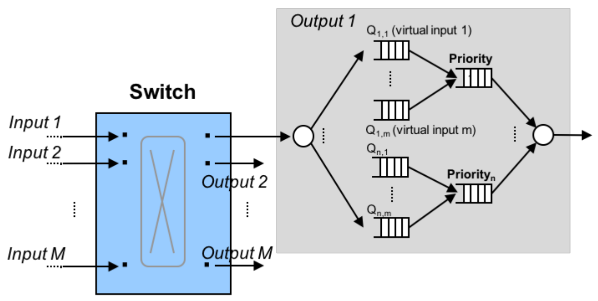


Figure 10: Switch architecture with Virtual Input Queues

3.2 Head-of-Line blocking

Regardless the way it is provided, lossless flow-control presents the problem of propagating congestion, as it has been mentioned in previous sections. However, congestion propagation is not a problem by itself. For instance, consider the situation depicted in Figure 2, where a congestion tree grows from root to leaves due to congestion propagation: if the only flows in the depicted portion of the network are those contributing to the tree, that portion of the network is working at the maximum possible efficiency, despite the presence of a congestion tree. Indeed, in this example links are used at the maximum speed that the traffic pattern allows. Therefore, there must be other reason for the performance degradation that is usually observed in the networks upon congestion propagation. This reason is the Head-Of-Line (HOL) blocking derived from the congestion trees.

The HOL blocking is a well-known phenomenon that, in general, happens when the packet at the head of a queue blocks, then preventing other packets stored behind from advancing, even if they request available queues to be forwarded. In congestion situations, the flow control indeed will tend to block packets at the head of the queues placed along the paths followed by the flows contributing to congestion (i.e. along the congestion tree branches). If these queues also store packets belonging to flows not contributing to the tree, these packets are likely to suffer HOL blocking. The overall effect is that the flows not contributing to the congestion tree, but sharing queues with contributor flows, will end up advancing at the same pace as the latter. This leads to a strong underutilization of network resources and so to a strong network performance degradation.

Figure 11 shows a congestion situation where HOL blocking appears at several switches. A switch speedup of one is assumed, as well as the use of some type of flow control mechanism. Packet injection rate is assumed to be the maximum rate allowed by the link and by the flow control mechanism. Note that in the figure, each link has been labelled with its final percentage of utilization once the network reaches stable state. Specifically, three flows addressed to the same destination (X) are injected from different sources, this incast situation forming a congestion tree. The bandwidth of the link connected to the incast destination is divided equally into the contributor flows, so that, due to flow control, they end up being injected at 33% of the link bandwidth. As can be seen, there are

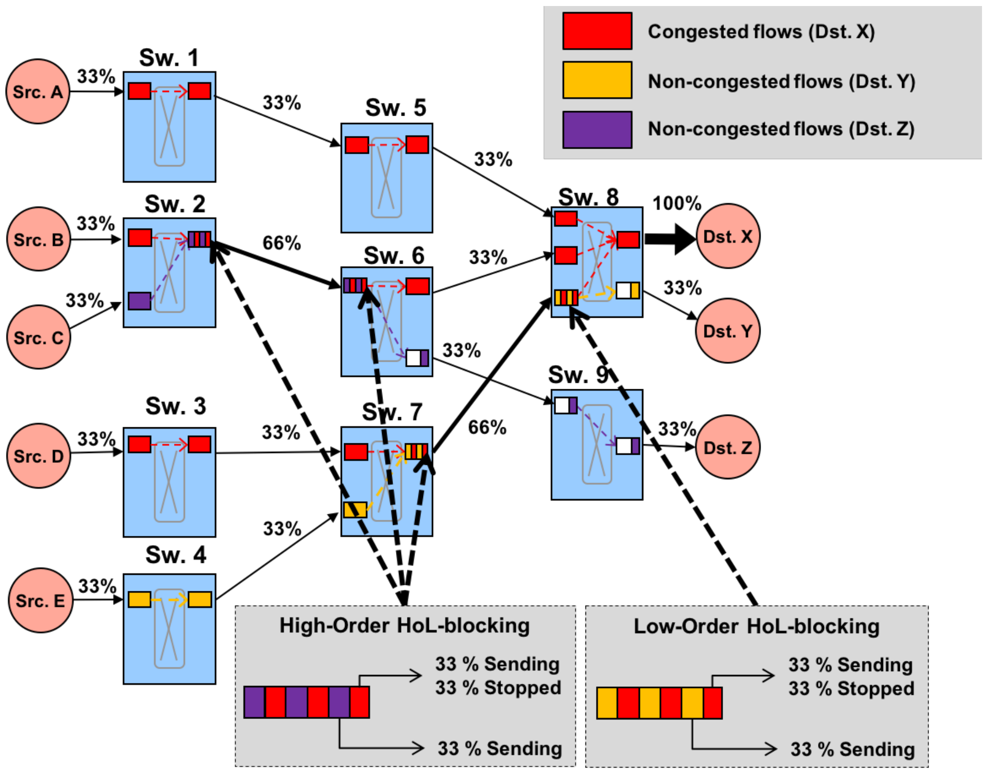


Figure 11: Network performance degradation due to HOL blocking

other flows in the network that are injected from other sources, and are addressed to destinations different from X (Y and Z respectively), so that they do not contribute to the congestion tree. However, as can be seen, the latter flows interact (i.e. share queues) with the congested flows at several points of the network, then the latter flows produce HOL blocking to the former ones. As a consequence, all the flows end up being injected at the same rate as the contributor ones, despite the fact that the links connected to Y and Z are not saturated. On the other hand, note that, at the switch where the root is located, the HOL blocking appears due to the direct contention of the congested flows to access the root of the tree, but in other (upstream) switches the HOL blocking appears due to congestion propagation through flow control. In general, the HOL blocking produced in the switch where congestion originates is called "low-order" HOL blocking, while the one produced at upstream switches is called "high-order" HOL blocking.

4 Reducing In-Network Congestion

Taking into account the problems derived from congestion trees, a natural approach to deal with them is trying to prevent their formation. This approach is actually a particular case of the generic "proactive" congestion management, that in general tries to solve congestion before it appears. Although many strategies have been proposed to prevent the formation of congestion trees, load balancing is probably one of the most popular. Basically, load-balancing techniques try to reduce the oversubscription of some links (so the contention to access their connected egress ports) by using several alternative paths between a given source and a given destination.

In order to be really effective against congestion, load-balancing techniques should be aware of network status regarding congestion and make decisions based on this information. However, this is not the case of some techniques that distribute flows obliviously among several paths. For instance, Equal Cost Multi-Path (ECMP) spreads packets across the available paths based on a hash of their flow identity field. Obviously, ECMP (like similar techniques) cannot guarantee at all that the formation of congestion trees is prevented, as it acts "blindly" with respect to the current status of the paths where the packets are spread across. Indeed, some of these available paths may be already congested, or about to be congested, then ECMP would contribute to expand/create a congestion tree when selecting those paths. Moreover, as the path selection is based on the packets flow identifier, an intense flow will be routed always through the same path but not scattered, also increasing the probability of expanding/creating congestion trees. Summing up, using ECMP-like techniques to deal with congestion may be completely counterproductive.

Therefore, congestion-aware load-balancing techniques are much more likely to prevent the formation of congestion trees. For instance, Load-Aware Packet Spraying (LPS) distributes packets across the available paths based on the congestion level of these paths. Note that, in contrast with ECMP-like techniques, LPS may route packets belonging to the same flow through different paths. This is positive as, in general, the packets will avoid the congested (or about to be congested) routes, that then wil reduce their congestion level. Moreover, in contrast with ECMP, the packets belonging to intense flows can be scattered with the purpose of preventing these flows from contributing strongly to a congestion situation. Overall, all this may help to reduce the probability of congestion situations inside the network. Note, however, that this requires a re-ordering mechanism of the packets at destination, which may introduce a certain complexity. In addition, note that congestion awareness also introduces complexity, as some congestion detection/measuring mechanism is required, as well as some storage structures at the sources to gather the congestion information, and the corresponding logic to consult these structures and make routing decisions.

However, the main drawback of the congestion-aware load-balacing techniques is not the introduced complexity, but that they cannot solve some congestion situations by themselves. Specifically, in the case of incast-derived (i.e. created by an intense "many-to-one" communication pattern) congestion trees, spraying the packets will not prevent these trees from appearing, as the contributing packets will meet unavoidably at some switch or switches. Indeed, even if the packets contributing to the incast situation are sprayed so that they follow completely different routes, they sure will end up meeting at the common end of these routes, i.e. at the egress port connected to the common destination. Therefore, in incast-derived congestion situations, a congestion tree will form in one way or another

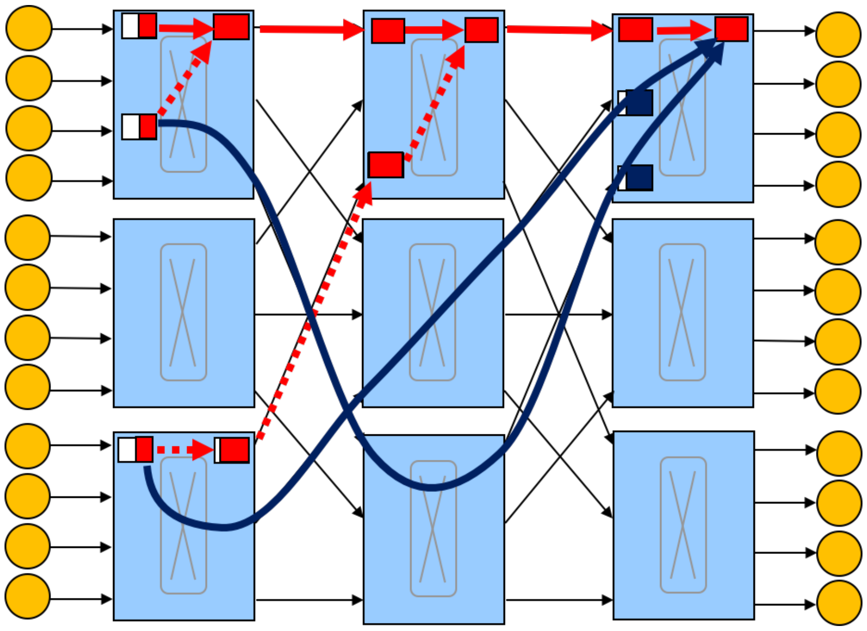


Figure 12: Load balancing moving downstream the root of a tree, but not preventing congestion.

despite the load-balancing mechanism, that, on the other hand, is likely to move the root downstream and change the shape of the branches as contributor packets are sprayed. This effect can be seen in Figure 12, where the original flows are shown in red while re-routed flows are shown in blue.

Nevertheless, packets belonging to flows not contributing to the congestion root should be routed by the mechanism so that they avoid the congested paths (i.e. the branches of the tree), then reducing in-network congestion, and also preventing the HOL blocking that contributor packets could produce in the queues along the congested paths. However, note that if the shape of the branches of the tree changes frequently, it may be difficult to keep track of the congestion status across the network, so that congestion information at the sources may be outdated, and packets may end up being routed wrongly, through congested paths. Note that something similar would happen when congestion trees grow fast, so that the sources are not able to detect congestion and react in time to avoid the formation of the trees. Another problem is that, especially under strong congestion situations, spraying contributor packets may propagate congestion across more paths than if traffic were not balanced, so making increasingly difficult, or even impossible, to avoid the (more numerous) branches of the tree.

In summary, congestion-aware load-balancing techniques may help to delay the appearance of congestion or to reduce in-network congestion, but they cannot prevent the formation of congestion trees derived from incast, end even they may be counterproductive under strong or sudden congestion situations that may lead to a wrong or late reaction.

5 Reducing Incast Congestion

The only way to prevent or remove congestion trees derived from incast is that the sources of packet flows contributing to those trees adjust in some way their packet injection rate so that they do not end up saturating the queues at the switches where they meet. This basic approach has been followed by several techniques that, on the other hand, differ in several (and even many) aspects. However, these techniques can be grouped into two main categories, usually known as end-to-end (e2e) injection throttling and destination scheduling.

End-to-end injection throttling is in general based on detecting congestion roots, mainly as the ports where there are oversubscribed queues that are not blocked due to flow control (although other methods are possible). When a congestion root is detected, then the packets crossing that root are marked as congested, although depending on the specific e2e technique, the percentage of marked packets may vary. Upon reception of marked packets, the receivers send congestion notifications back to their sources. When sources receive congestion notifications, they react reducing the rate of (or even ceasing) the injection of packets, then allowing the draining of the flows that contribute to the tree, that will eventually vanish. This general strategy has been followed by several techniques suitable for Ethernet-based networks like Explicit Congestion Notification (ECN) and Quantized Congestion Notification (QCN) [8], but also for other techniques suitable for other technologies, like the InfiniBand FECN/BECN (Forward Explicit Congestion Notification/Backwards Explicit Congestion Notification) mechanism [9]. The main difference between these mechanisms is the specific way that the sources reduce their injection once notified of congestion, but in general the more notifications receive the sources, the more is reduced the injection rate.

Although in some "light" incast scenarios this type of mechanisms may be useful, in other cases they may be not able to solve the incast congestion situation by themselves. In general, their main drawback is the delay between congestion detection and the reaction at the sources, as in the meanwhile the incast-derived congestion tree may have appeared and grown significantly due to the activation of flow control[4] (see Section 3.1). Although the sources will eventually react, the congestion tree will affect for a while to several paths in the network. Moreover, the draining of the congested packets will take a significant time, as all of them are addressed to the same destination. Note that the detection-reaction delay is higher in modern, large DC networks, where the round-trip-time of the packets increases as the routes consist of more hops. Note also that, the higher the link bandwidth, the faster the congestion notifications will reach the sources, but also the faster the congested packets will be injected into the network prior to congestion notification. As a consequence of all these problems, these mechanisms are prone to introduce oscillations in the network performance (a common problem of many closed-loop control mechanisms), as the flows may end up being repeatedly throttled and released.

[4]Note that, in lossy networks, the congestion tree would not grow, but the detection-reaction delay would lead to huge packet losses.

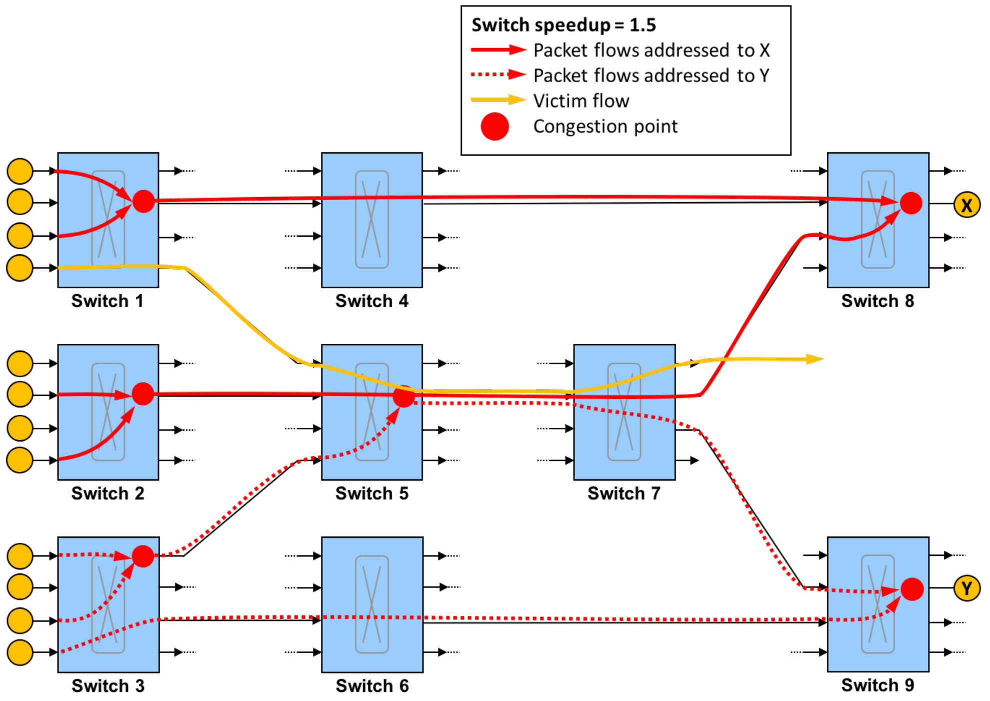


Figure 13: In-network "false" root leading to mark "innocent" packets as congested.

Moreover, in general injection throttling mechanisms are not well suited to deal with in-network congestion. For instance, if two or more congestion trees, derived from incast at different destinations (i.e. trees with different roots), overlap at some network points and create a new in-network congestion point (like in Figure 6), the mechanism is likely to deal with these situation detecting (wrongly) this in-network point as a congestion root, then all the packets crossing that point will be marked as congested. As a consequence, the mechanism may end up marking more packets than necessary of the flows contributing to the incast situation, then more congestion notifications than necessary may be sent to their sources, that eventually may throttle injection more than actually required. Obviously, this is especially inconvenient if one of the trees is not as intense as the other one. Even worse, if a flow not contributing to any incast-derived congestion tree crosses that "false" root (i.e. it partially overlaps with the contributor flows at that point, as can be seen in Figure 13), the mechanism may mark packets of that flow as congested, then their source will be wrongly requested to reduce its injection rate. Note that in this case this "innocent" flow will be doubly penalized, as its packets are likely to suffer also HOL blocking when meeting the contributor flows. In summary, while injection throttling mechanisms may help in some cases (not always) to eliminate incast congestion, in-network congestion may lead these mechanisms to an inaccurate or wrong reaction.

On the other hand, destination scheduling is based on some type of "negotiation" between source and destination, so that the source regulates its packet injection rate according to the availability of resources at destination. Traditional techniques following this strategy are usually based on request/grant communication between source and destination prior to packet injection. This is usually known as "pulling" packets from the sources, in contrast with the sources "pushing" packets without any prior negotiation, so being unaware of destination status. The most obvious drawback of packet pulling is the delay introduced by the request/grant mechanism, as a request message must travel form source to destination, then a grant message is sent back from destination to source, i.e. this exchange of messages introduces a round-trip delay before any transmission of packets can start. Moreover, note that this delay may make the feedback information (implemented through the grants) obsolete by the time it reaches the sources, so that a congestion situation may have appeared, or be about to appear, after the grant is sent from the destination.

In order to solve these problems, a less strict strategy is combining both packet pushing and pulling, which is known as Push and Pull Hybrid (PPH). Basically, the sources are by default allowed to push packets without prior negotiation. However, the sources monitor the congestion level in the path towards destination so that they may switch to pull transmission if necessary, i.e. so that congestion trees does not appear. Under heavy loads, the sources may eventually end up transmitting only in pull mode, especially to avoid the formation of incast-derived congestion trees. Note that the PPH pull mode would introduce still a round-trip delay, but as pull is used only in heavy-load scenarios, that increase packet latency, the impact of this delay may be not as relevant as in light-load scenarios. However, like in the "exclusive pull" policy, the round-trip delay of PPH in pull mode may still spoil the validity (update) of the grants. Moreover, like in congestion-aware load-balancing, it may be difficult for the sources to gather and keep track of congestion information, especially when congestion trees grow fast, so that the switching from push to pull mode may happen too late to avoid the formation of incast-derived congestion trees.

In summary, both injection-throttling and destination-scheduling techniques present the common problem of the high, round-trip delays due to their required end-to-end communication. In general, the reaction of these mechanisms is too slow to appropriately deal with congestion trees that grow fast.

6 Queue-Based Congestion Isolation

6.1 Congestion Prevention vs Congestion Isolation

In contrast with the strategies explained in the previous sections, that prevent, delay ot remove the formation of congestion trees, other techniques focus on preventing HOL blocking without necessarily attacking congestion, i.e they let the congestion trees exist and try to eliminate or reduce the impact its main negative effect. Indeed, as discussed in Section 3.2, network performance degradation under congestion situations is mainly due to the low- or high-order HOL blocking produced by the flows contributing to congestion over

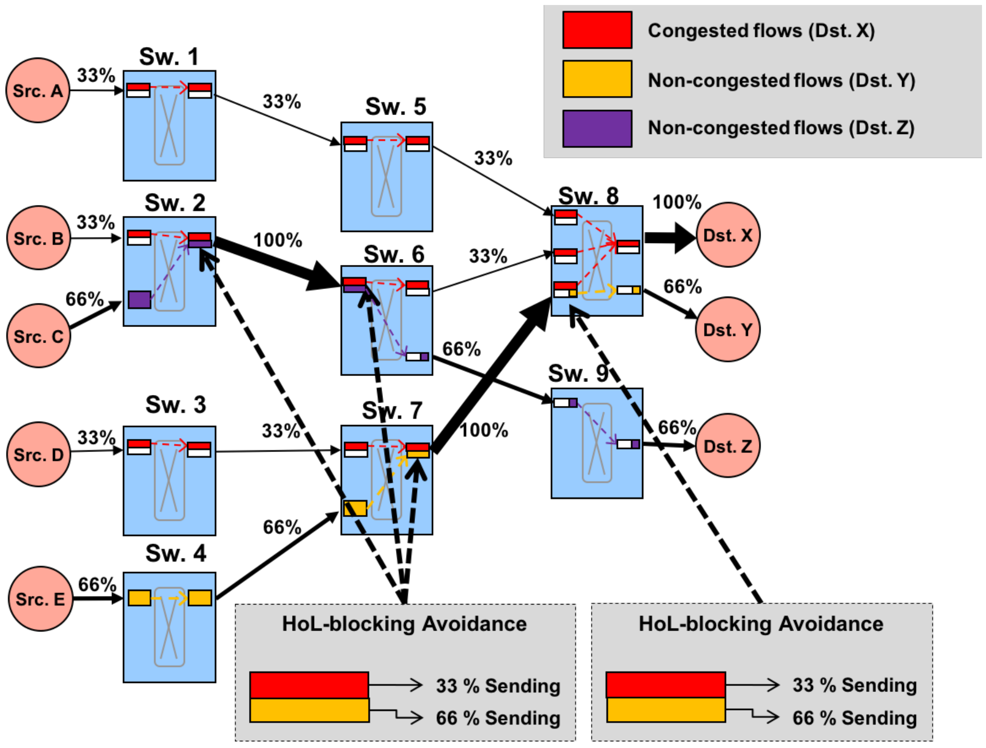


Figure 14: Isolating contributor flows eliminates HOL blocking

other flows, so that if HOL blocking were prevented, congestion trees could exist without actually being a problem on their own.

Although this basic approach have been followed by very different techniques, in general a common advantage of all of them is that they deal with HOL blocking fast and locally, as they attack the HOL blocking when and where it may appear. Therefore, they can complement some techniques explained above that react slowly to congestion (see Section 7). Another common issue of all these techniques is that they configure several queues per port to separately store packets belonging to different flows, hence reducing or eliminating the interaction among these flows and so the HOL blocking they can produce each other. Hence, the ultimate goal of these techniques is to isolate in separate queues the flows that contribute to congestion, which are the ones producing significant HOL blocking. This can be seen in Figure 15, that shows the same situation depicted in Figure 11 but now with several queues configured per port to separate the flows addressed to different destinations. As can be seen, the separation of flows into different queues leads to the elimination of HOL blocking, so to an improvement of the performance of the network, as the utilization of several links increases with respect to the situation that the flows share queues.

Note that the HOL blocking would be ideally and completely eliminated if there were, at every port, a queue specifically dedicated to each destination [10], but this is obviously unaffordable as the number of queues in the network would grow quadratically with network size. Hence, other alternatives have been proposed in order to implement this approach affordably (in terms of the number of required queues), that can be classified according to their policy to manage the queues, either statically or dynamically. The techniques based on static policies configure a reduced number of queues per port and map packets to these queues according to some fixed criteria that, in general, tries to separate the different flows as much as possible into the available queues. Among these techniques, it is worth mentioning Virtual Output Queues (VOQsw) [11], Dynamically Allocated Multiqueues (DAMQs) [12], Destination-Based Buffer Management (DBBM) [13], Dynamic Switch Buffer Management (DSBM) [14], Output-Based Queue-Assignment (OBQA) [15], Flow-to-Service-Level (Flow2SL) [16], vFTree [17], etc. In general these techniques are popular and straightforward to implement, differing in the number of required queues and in the criteria followed to map packets to queues, as well as in the interconnect technologies where they can be implemented.

However, none of these "static-queuing" techniques detects actually congestion, neither incast nor in-network, so that they map packets to queues obliviously, unaware of the state of network congestion. Therefore their mapping criteria are not actually oriented to separate congested flows from non-congested ones, but to blindly separate any flow from as many others as possible, which reduces their efficiency as the queues may be not used optimally to reduce HOL blocking. By contrast, the techniques based on dynamic criteria to map packets to queues can completely isolate the contributor flows if these flows can be identified, then mapped to exclusive queues, as explained in the next section.

6.2 Basics of Dynamic Isolation of Congestion

The key idea behind the dynamic isolation of congestion is to detect congestion roots at a switch (which can be done through several methods, but mainly based on the occupancy level of the standard queue) in order to identify packets contributing to that root, then isolating them in special queues that are specially and dynamically assigned for that purpose. On the contrary, packets belonging to flows not contributing to any root are all stored in standard queues without suffering significant HoL-blocking, even if they belong to different cold flows, as theoretically none of these flows will be blocked. In this way, one additional, special queue per port is enough to prevent the low- and high- order HOL blocking derived from a single congestion root. In addition, the special queues can be dynamically assigned/released in order to be used only when required (i.e. when hot flows appear/disappear at some port), thereby optimizing even more the use of the queues available at the ports.

Note that the first reaction of these type of mechanisms upon detection of congestion is local and immediate, so that countermeasures are taken early without need for waiting to feedback messages from other switches or endnodes. However, when the occupancy of the special queue fills over a certain level, a notification is sent upstream so that a special queue is assigned at the receiver(s) upstream port(s) to store packets contributing to the detected congestion root. If the newly-assigned special queue subsequently gets also congested, it will notify upstream and so on. In this way, the notifications will propagate upstream so that special queues will be assigned along the branches of the congestion tree to effectively isolate the contributor packets (and so preventing the HOL blocking that they could cause). Note that this requires that notifications must indicate some type of information to locate the root, or to identify the contributor flows. Moreover, some structures are needed to gather this information, as well as some logic to decide if incoming packets must be stored in the special or in the standard queue (i.e. to decide if they belong to a contributor flow or not), which may introduce a certain complexity to the implementation of these mechanisms. It is worth clarifying that the special and standard queues must be independent regarding flow control, i.e a special queue can block only the immediate upstream special queue, the same being valid also for standard queues[5].

Another essential aspects of these mechanisms is the policy followed to release the queues. Indeed, In order to be actually efficient, the special queues should be released upon congestion vanishing, so that they could be reused if new congestion trees appear. This is actually as important as the assignment of the special queues, since a too-early or too-late release may spoil the efficiency of the mechanism.In general the special queues should be released only when it is clear that congestion has vanished at the port where they are placed, which can be detected by monitoring the occupancy level of the own queue.

This general procedure has been followed by several techniques such as Regional Explicit Congestion Notification (RECN) [18, 19], Flow-Based Implicit Congestion Management (FBICM), Distributed-Routing-Based Congestion Management (DRBCM) [20] and Dynamic Virtual Lanes (DVL). However, only DVL has been specifically proposed for Ethernet-Based DCNs, the former techniques focusing on HPC systems built from other interconnect technologies. Figure 15 shows an example of how DVL operates once the mechanism has detected congestion, assigned the special queues, and propagated congestion information. Note that the special queues are called Congested-Flow Queues (CFQ) while the standard queues are called Non-Congested Flow Queues (nCFQs). Note also that the congestion notification, called CIP (Congestion Isolation Packet), would have been sent actually prior to the assignment of CFQ at switch A. It is worth pointing out that, in the example, the assignment of CFQs in both switches A and B prevents that two non-congested flows suffer HOL blocking, despite the fact that one of them (shown in black in the figure) does not reach the switch where congestion originates and is detected.

In general, Dynamic Congestion Isolation is a very efficient strategy, as the impact of congestion trees in network performance is almost eliminated through a reduced set of queues. However, it may happen that the number of congestion trees concurrently active in a port is higher than the number of queues available to store the flows contributing to the different trees. If the mechanism runs-out of special queues, congested packets will end up being stored at the standard queues, producing HOL blocking to non-congested packets. Note that even if only one congestion tree is not assigned with special queues, this may be enough to degrade network performance. Hence, it is convenient to complement these

[5]However, note that, as standard queues store only non-congested packet, the probability of these queues activating flow control significantly decreases.

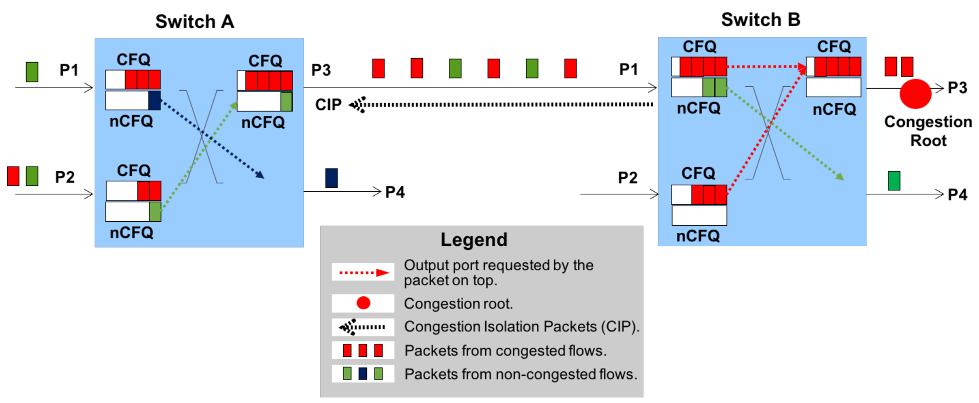

Switch A
Switch B
CFQ
nCFQ
P1
P2
P3
P4
CIP
Congestion Root
Legend
Output port requested by the packet on top.
Congestion root.
Congestion Isolation Packets (CIP).
Packets from congested flows.
Packets from non-congested flows.

Figure 15: DVL operation

mechanisms with others able to prevent or remove congestion trees once appear, in order to reduce the number of concurrent trees in the network so that the number of special queues is enough to isolate all them (more insights on this can be found in Section 7.

7 Combined Behavior of Congestion Management Mechanisms

In previous sections we have analyzed independently the pros and cons of several approaches to deal with congestion, but actually several techniques can be combined in order to complement each other so that their respective drawbacks are solved while the respective benefits are kept.

For instance, in Section 4 we have seen that load balancing may have problems dealing with incast congestion, while in Section 5 we have seen that injection throttling or destination scheduling may react inappropriately in cases of in-network congestion. Hence, it is possible to combine bot approaches, depending on whether incast or in-network congestion is monitored. Basically, the network could work in load-balancing mode (using LPS, for instance) until incast congestion appears, then switches to destination-scheduling mode (using PPH, for instance). In general this combination would lead to LPS solving light congestion situations while PPH solving heavy congestion situations. Moreover, note that PPH would prevent LPS from changing constantly the routes of congested flows contributing to incast situations.

However, as we have explained previously, both load balancing, injection throttling and destination scheduling are intended to prevent or remove congestion trees, but in practice they react slowly to congestion, since the countermeasures are taken at the sources, which in general introduces a round-trip delay. By contrast, congestion isolation techniques such as DVL react fast and locally, isolating the flows contributing to congestion, but they may run out of special queues (CFQS) when several congestion trees appear in the network. Hence, combining DVL with LPS, ECN or PPH would be positive for all the combined techniques. On the one hand, DVL would deal with congestion during the transient period required to activate the reaction of LPS, ECN or PPH at the sources. In addition, the isolation of the congestion trees by DVL would help the feedback information required by LPS, ECN or PPH to reach the sources faster. On the other hand, once the sources react through LPS, ECN or PPH, the congestion trees will be drained, so that DVL will be able to release the CFQs assigned to those trees, then these CFQs will be ready to be assigned to newly-appearing congestion trees.

Note, however, that LPS may cause that the branches of incast-derived congestion trees change their shape constantly without actually disappearing. In these cases, DVL may have problems to keep track of the evolution of the tree, then CFQs may end up being repeatedly assigned and released at several points in the network. Therefore, a fine tuning of the LPS reaction would be required, ideally based on the feedback information that could be drawn from the own DVL control structures. Alternatively, LPS+PPH could be used in combination with DVL, so that PPH reduces the instability introduced by LPS and both are benefited from the fast and local reaction of DVL. For the same or similar reasons, DVL could be also combined with ECN. Indeed, a combination of a congestion isolation mechanism with an injection throttling one has been already proposed in [21], where the mutual benefits of this combination are tested and evaluated positively.

In summary, the combination of different approaches to deal with congestion in DC networks seems a strategy that is worth to explore in order to exploit optimally the positive synergies of different techniques working together.

8 Conclusions

Lossless networks suitable for data centers are currently required due to the high requirements of the applications supported by these systems nowadays (and likely in the future). Specifically, applications are becoming extremely sensitive to latency, which is not compatible with lossy networks due to the high delay introduced by retransmission mechanisms upon packet losses. However, lossless networks are prone to suffer congestion situations, mainly due to the backpressure of PFC flow control, that tends to propagate congestion throughout the network, generating HOL blocking that dramatically spoils network performance. In addition, incast situations derived from the usual many-to-one traffic patterns that can be found in data centers, may increase the probability of congestion appearing. The classical solution is to complement PFC with ECN to eliminate the congestion trees, but the slow reaction of this mechanism, which is even slower in modern, larger DC networks, allows congestion trees to grow and stay for longer. Hence, alternative approaches must be considered to deal with congestion in modern DC networks. Ideally, these approaches should be designed and used taking into account the dynamics of congestion trees, that have been described in depth in this work. From the analysis of the different alternative approaches to deal with congestion, we can conclude that all of them present advantages and disadvantages. Among the most important disadvantages, all the strategies requiring feedback information to produce a reaction at the sources (load balancing, injection throttling, destination scheduling) react too slowly to cope with fast-growing congestion trees. Another relevant drawback is that some of these strategies are suited to deal with incast congestion but not with in-network congestion, or vice versa. On the other hand, congestion isolation is very effective in reducing the impact of congestion but may be unable to cope with several simultaneous congestion trees. Hence, a smart combination of different types of techniques may be necessary to avoid the performance degradation of lossless DC networks under congestion situations, so that the techniques complement each other, solving the respective flaws while keeping the respective benefits.

References

[1] *InfiniBand^{TM} Architecture Specification Release 1.2.1 Annex A17: RoCEv2*, InfiniBand Trade Association, September 2014.

[2] H. B. et al., "Proposal for priority based flow control. http://www.ieee802.org/1/files/public/docs2008/new-dcb-pelissier-pfc-proposal-0308.pdf."

[3] M. Karol, M. Hluchyj, and S. Morgan, "Input Versus Output Queueing on a Space-Division Packet Switch," *IEEE Transactions on Communications*, vol. 35, no. 12, pp. 1347–1356, Dec. 1987.

[4] P. J. García, J. Flich, J. Duato, I. Johnson, F. J. Quiles, and F. Naven, "Dynamic Evolution of Congestion Trees: Analysis and Impact on Switch Architecture," in *High Performance Embedded Architectures and Compilers*, ser. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, Nov. 2005, pp. 266–285. [Online]. Available: https://link.springer.com/chapter/10.1007/11587514_18

[5] T. Colombo, "Optimising the data-collection time of a large-scale data-acquisition system," Ph.D. dissertation, University of Heidelberg, 2018.

[6] K. R. et al., "The addition of explicit congestion notification (ecn) to ip https://tools.ietf.org/html/rfc3168."

[7] T. M. Pinkston and J. Duato, "Appendix F: Interconnection Networks," in *Computer Architecture: A Quantitative Approach*, Elsevier, Ed. Morgan Kaufmann, 2012. [Online]. Available: http://booksite.elsevier.com/9780123838728/references/appendix_f.pdf

[8] A. Kabbani, M. Alizadeh, M. Yasuda, R. Pan, and B. Prabhakar, "AF-QCN: approximate fairness with quantized congestion notification for multi-tenanted data centers," in *IEEE 18th Annual Symposium on High Performance Interconnects, HOTI 2010, Google Campus, Mountain View, California, USA, August 18-20, 2010*, 2010, pp. 58–65. [Online]. Available: https://doi.org/10.1109/HOTI.2010.26

[9] G. Pfister, M. Gusat, W. Denzel, D. Craddock, N. Ni, W. Rooney, T. Engbersen, R. Luijten, R. Krishnamurthy, and J. Duato, “Solving Hot Spot Contention Using InfiniBand Architecture Congestion Control,” in *Proc. of Int. Workshop HPI-DC*, 2005.

[10] W. Dally, P. Carvey, and L. Dennison, “Architecture of the Avici terabit switch/router,” in *Proc. of 6th Hot Interconnects*, 1998, pp. 41–50.

[11] T. Anderson, S. Owicki, J. Saxe, and C. Thacker, “High-Speed Switch Scheduling for Local-Area Networks,” *ACM Transactions on Computer Systems*, vol. 11, no. 4, pp. 319–352, November 1993.

[12] Y. Tamir and G. L. Frazier, “Dynamically-Allocated Multi-Queue Buffers for VLSI Communication Switches,” *IEEE Trans. Computers*, vol. 41, no. 6, pp. 725–737, Jun. 1992.

[13] T. Nachiondo, J. Flich, and J. Duato, “Buffer Management Strategies to Reduce HoL Blocking,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 21, pp. 739–753, 2010.

[14] W. Olesinski, H. Eberle, and N. Gura, “Scalable Alternatives to Virtual Output Queuing,” in *Proceedings of IEEE International Conference on Communications, ICC 2009, Dresden, Germany, 14-18 June 2009*, 2009, pp. 1–6.

[15] J. Escudero-Sahuquillo, P. J. García, F. J. Quiles, J. Flich, and J. Duato, “OBQA: Smart and cost-efficient queue scheme for Head-of-Line blocking elimination in fat-trees,” *J. Parallel Distrib. Comput.*, vol. 71, no. 11, pp. 1460–1472, 2011.

[16] J. Escudero-Sahuquillo, P. J. García, F. J. Quiles, S. Reinemo, T. Skeie, O. Lysne, and J. Duato, “A new proposal to deal with congestion in InfiniBand-based fat-trees,” *J. Parallel Distrib. Comput.*, vol. 74, no. 1, pp. 1802–1819, 2014.

[17] W. L. Guay, B. Bogdanski, S.-A. Reinemo, O. Lysne, and T. Skeie, “vFtree - A Fat-tree Routing Algorithm using Virtual Lanes to Alleviate Congestion,” in *Proceedings of the 25th IEEE International Parallel & Distributed Processing Symposium*, Y. Xin, Ed. IEEE Computer Society Press, 2011, pp. 197–208.

[18] P. J. García, F. J. Quiles, J. Flich, J. Duato, I. Johnson, and F. Naven, “Efficient, Scalable Congestion Management for Interconnection Networks,” *IEEE Micro*, vol. 26, no. 5, pp. 52–66, 2006.

[19] G. Mora, P. J. García, J. Flich, and J. Duato, “RECN-IQ: A cost-effective input-queued switch architecture with congestion management,” in *2007 International Conference on Parallel Processing (ICPP 2007), September 10-14, 2007, Xi-An, China*, 2007, p. 74. [Online]. Available: https://doi.org/10.1109/ICPP.2007.71

[20] J. Escudero-Sahuquillo, P. J. García, F. J. Quiles, J. Flich, and J. Duato, "An effective and feasible congestion management technique for high-performance mins with tag-based distributed routing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 24, no. 10, pp. 1918–1929, 2013. [Online]. Available: http://dx.doi.org/10.1109/TPDS.2012.303

[21] J. Escudero-Sahuquillo, E. G. Gran, P. J. García, J. Flich, T. Skeie, O. Lysne, F. J. Quiles, and J. Duato, "Efficient and Cost-Effective Hybrid Congestion Control for HPC Interconnection Networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 1, pp. 107–119, 2015.