Converged Networking in the Data Center

Peter P. Waskiewicz Jr.
LAN Access Division, Intel Corp.
peter.p.waskiewicz.jr@intel.com

Abstract

The networking world in Linux has undergone some significant changes in the past two years. With the expansion of multiqueue networking, coupled with the growing abundance of multi-core computers with 10 Gigabit Ethernet, the concept of efficiently converging different network flows becomes a real possibility.

This paper presents the concepts behind network convergence. Using the IEEE 802.1Qaz Priority Grouping and Data Center Bridging concepts to group multiple traffic flows, this paper will demonstrate how different types of traffic, such as storage and LAN traffic, can efficiently coexist on the same physical connection. With the support of multi-core systems and MSI-X, these different traffic flows can achieve latency and throughput comparable to the same traffic types' specialized adapters.

1 Introduction

Ethernet continues to march forward in today's computing environment. It has now reached a point where PCI Express devices running at 10GbE are becoming more common and more affordable. The question is, what do we do with all the bandwidth? Is it too much for today's workloads? Fortunately, the adage of "if you build it, they will come" provides answers to these questions.

Data centers have a host of operational costs and upkeep associated with them. Cooling and power costs are the two main areas that data center managers continue to analyze to reduce cost. The reality is as machines become faster and more energy efficient, the cost to power and cool these machines is also reduced. The next question to ask is, how can we push the envelope of efficiency even more?

Converged Networking, also known as Unified Networking, is designed to increase the efficiency of the data center as a whole. In addition to the general power and cooling costs, other areas of focus are the physical amount of servers and their associated cabling that reside in a typical data center. Servers very often have multiple network connections to various network segments, plus they're usually connected to a SAN: either a Fiber Channel fabric or an iSCSI infrastructure. These multiple network and SAN connections mean large amounts of cabling being laid down to attach a server. Converged Networking takes a 10GbE device that is capable of Data Center Bridging in hardware, and consolidates all of those network connections and SAN connections into a single, physical device and cable. The rest of this paper will illustrate the different aspects of Data Center Bridging, which is the networking feature allowing the coexistence of multiple flows on a single physical port. It will first define and describe the different components of DCB. It then will show how DCB consolidates network connections while keeping traffic segregated, and how this can be done in an efficient manner.

2 Priority Grouping and Bandwidth Control

2.1 Quality of Service

Quality of Service is not a stranger to networking setups today. The QoS layer is composed of three main components: queuing disciplines, or qdiscs (packet schedulers), classifiers (filter engines), and filters [1]. In the Linux kernel, there are many QoS options that can be deployed: One qdisc provides packet-level filtering into different priority-based queues (sch_prio); Another can make bandwidth allocation decisions based on other criteria (sch_htb and sch_cbq). All of these built-in schedulers run in the kernel, as part of the `dev_queue_xmit()` routine in the core networking layer (`qdisc_run()`). While these pieces of the QoS layer can separate traffic flows into different priority queues in the kernel, the priority is isolated to the packet

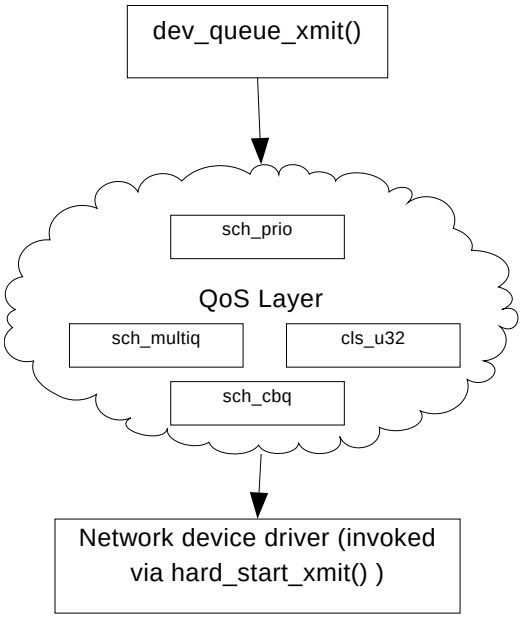

Figure 1: QoS Layer in the Linux kernel

scheduler within the kernel itself. The priorities, along with any bandwidth throttling, are completely isolated to the kernel, and are not propagated to the network. This highlights an issue where these kernel-based priority queues in the qdisc can cause head-of-line-blocking in the network device. For example, if a high priority packet is dequeued from the sch_prio qdisc and sent to the driver, it can still be blocked in the network device by a low priority, bulk data packet that was previously dequeued.

Converged Networking makes use of the QoS layer of the kernel to help identify its network flows. This identification is used by the network driver to decide on which Tx queue to place the outbound packets. Since this model is going to be enforcing a network-wide prioritization of network flows (discussed later), the QoS layer should not enforce any priority when dequeuing packets to the network driver. In other words, Converged Networking will not make use of the sch_prio qdisc. Rather, Converged Networking uses the sch_multiq qdisc, which is a round-robin based queuing discipline. The importance of this is discussed in Section 2.2.

2.2 Priority Tagging

Data Center Bridging (DCB) takes the QoS mechanism into hardware. It also defines the network-wide infrastructure for a QoS policy across all switches and end-stations. This allows bandwidth allocations plus prioritization for specific network flows to be honored across all nodes of a network.

The mechanism used to tag packets for prioritization is the 3-bit priority field of the 802.1P/Q tag. This field offers 8 possible priorities into which traffic can be grouped. When a base network driver implements DCB (assuming the device supports DCB in hardware), the driver is expected to insert the VLAN tag, including the priority, before it posts the packet to the transmit DMA engine. One example of a driver that implements this is ixgbe, the Intel® 10GbE PCI Express driver. Both devices supported by this driver, 82598 and 82599, have DCB capabilities in hardware.

The priority tag in the VLAN header is utilized by both the Linux kernel and network infrastructure. In the kernel, vconfig, used to configure VLANs, can modify the priority tag field. Network switches can be configured to modify their switching policies based on the priority tag. In DCB though, it is not required for DCB packets to belong to a traditional VLAN. All that needs to be configured is the priority tag field, and whatever VLAN that was already in the header is preserved. When no VLAN is present, VLAN group 0 is used, meaning the lack of a VLAN. This mechanism allows non-VLAN networks to work with DCB alongside VLAN networks, while maintaining the priority tags for each network flow. The expectation is that the switches being used are DCB-capable, which will guarantee that network scheduling in the switch fabric will be based on the 802.1P tag found in the VLAN header of the packet.

Certain packet types are not tagged though. All of the inter-switch and inter-router frames being passed through the network are not tagged. DCB uses a protocol, LLDP (Link Layer Discovery Protocol), for its DCBx protocol. These frames are not tagged in a DCB network. LLDP and DCBx are discussed in more detail later in this paper.

2.3 Bandwidth Groups

Once flows are identified by a priority tag, they are allocated bandwidth on the physical link. DCB uses bandwidth groups to multiplex the prioritized flows. Each bandwidth group is given a percentage of the overall bandwidth on the network device. The bandwidth group can further enforce bandwidth sharing within itself among the priority flows already added to it.

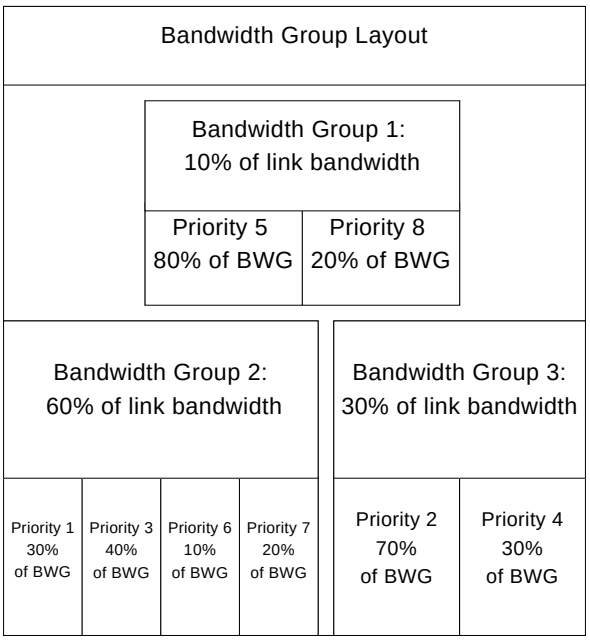

Figure 2: Example Bandwidth Group Layout

Each bandwidth group can be configured to use certain methods of dequeuing packets during a Tx arbitration cycle. The first is group strict priority: It will allow a single priority flow within the bandwidth group to grow its bandwidth consumption to the total of the bandwidth group. This allows a single flow within the group to consume all the bandwidth allocated to the group. This would normally be applied to flows that would run off-hours, and would be in groups that ran on-hours. An example of such a flow is a network backup. The second configuration is link strict priority: This allows any flow from any bandwidth group to grow to the maximum link bandwidth. Obviously this configuration can be dangerous if misconfigured, which could result in the starvation of other flows. However, this mode is necessary to guarantee flows that require maximum bandwidth to get the maximum bandwidth, without needing to reconfigure all bandwidth group layouts [2]. Refer to Figure 2 to see an example Bandwidth Group Layout.

2.4 Using TC filters to identify traffic

Now that all the priority flows are distributed into bandwidth groups, traffic flowing down from userspace must be filtered into the underlying queues. There are a few mechanisms that can be used to filter traffic into different queues.

- **`select_queue`** The network stack in recent kernels (2.6.27 and beyond) has a function pointer called `select_queue()`. It is part of the `net_device` struct, and can be overridden by a network driver if desired. A driver would do this if there is a special need to control the Tx queueing specific to an underlying technology. DCB is one of those cases. However, if a network driver hasn't overridden it (which is normal), then a hash is computed by the core network stack. This hash generates a value which is assigned to `skb->queue_mapping`. The skb is then passed to the driver for transmit. The driver then uses this value to select one of its Tx queues to transmit the packet onto the wire.

- **`tc filters`** The userspace tool, tc, can be used to program filters into the qdisc layer. tc is part of the iproute2 package. The filters can match essentially anything in the skb headers from layer 2 and up. The filters use classifiers, such as u32 matching, to match different pieces of the skbs. Once a filter matches, it has an action part of the filter. Most common for qdiscs such as sch_multiq is the skbedit action, which will allow the tc filter to modify the `skb->queue_mapping` in the skb.

DCB needs to make use of both of these mechanisms to properly filter traffic into the priority flows. First, the network driver must override the `select_queue()` function to return queue 0 for all traffic. DCB requires that all unfiltered traffic (i.e. traffic not matching a tc filter) be placed in priority flow 0. The `select_queue()` call is executed prior to the qdisc tc filter section in the core network stack, so if no filter is matched, then the value of `select_queue()` is retained.

tc filters are then added for each network flow that needs to be filtered into a specific priority flow Tx queue.

3 Priority Flow Control

In a converged network, various traffic types that normally wouldn't be on an Ethernet-based network are now present. Some of these traffic types are not tolerant of packet loss. Fiber Channel is a good example, and is added to a converged network using Fiber Channel over Ethernet [4]. Fiber Channel is not as tolerant of congestion and packet loss as Internet protocols. Therefore, it must have some form of flow control present to

ensure the frames can be paused, prior to some overrun causing dropped frames.

Using traditional Ethernet flow control is a viable option for these traffic flows. However, the point of Converged Networking is to provide separate, independent network pipes to traffic flows, and not allow one pipe to affect another pipe. Link-based flow control would cause all traffic flows to stop. This is not desired for DCB and Converged Networking.

Priority Flow Control (also known as per-priority pause or PFC) was designed to solve this issue by utilizing utilizes a different packet type from the traditional Ethernet pause. It passes a bitmap of all eight priorities to the link partner, indicating which priorities are currently paused. This way an XOFF/XON pair can be sent for each individual priority flow, while all other flows can continue transmitting and receiving data [3].

4 MSI-X and interrupt throttling for latency

4.1 Latency requirements

Each priority flow in a DCB network most likely has different latency considerations for the traffic in that flow. For example, high-availability management traffic require very low latency to operate correctly. On the other hand, bulk data transfers, like an FTP transfer, do not require low latency. Other examples of network traffic that have varying latency requirements include Voice Over IP, computer gaming, web surfing, audio streaming, p2p networks, etc. Each of these traffic types treat latency differently, where it either negatively effects the traffic, or doesn't make much difference whatsoever.

4.2 Interrupt rates vs. latency

The easiest way to affect the latency of a network device's traffic is to change the interrupt rate of the receive flow interrupt. For example, receive processing running at 8,000 interrupts per second will have a much higher latency than a device running at 100,000 interrupts per second. The trade-off is that the more interrupts a device generates, the higher your CPU utilization will be. Interrupt rates should be tuned to meet the target flow's latency considerations, and will vary based on the contents of that flow.

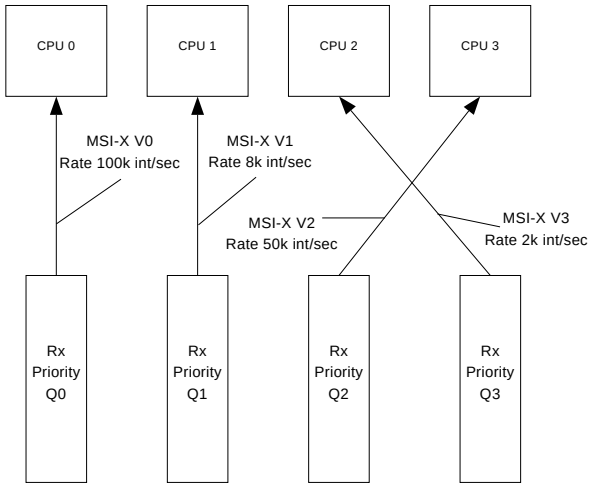

Figure 3: Example MSI-X mapping with variable interrupt rates

4.3 MSI-X interrupts

Each traffic flow in DCB may require a unique latency target, therefore requiring a unique interrupt rate. On devices that only support legacy pin interrupts, this cannot be achieved. Rather, the lowest latency must be chosen, and that interrupt rate must be used for the device. This will cause much more CPU overhead than is required for the other flows in your converged network.

MSI-X interrupts (Messaged Signaled Interrupts, Extended) provide the ability to have separate interrupt vectors for each traffic flow. Each vector can be assigned to a receive queue and transmit queue on the network device. Each of those vectors can then be assigned a different interrupt rate, which allows separate traffic latencies for each flow. Refer to Figure 3 to see a sample MSI-X layout with variable interrupt rates.

For another example, the ixgbe's EITR (Extended Interrupt Throttle Rate) registers control the interrupt rates for each interrupt vector. When the device is in MSI-X mode, the device enables an individual EITR register for each MSI-X vector [5]. The driver can then program each EITR separately, accomplishing the need to have fully independent interrupt rates among flows.

5 Data Center Bridging Exchange Protocol

DCB has a number of parameters that define how the link operates. The priority group configuration, the

bandwidth allocations, and the priority flow control settings are all part of the overall DCB configuration. Since DCB is a network-wide configuration, there needs to be a mechanism between link partners to negotiate these configuration settings. Data Center Bridging Exchange Protocol, or DCBx, is the protocol that defines how DCB configuration parameters are negotiated on a link. This is a very similar mechanism used to negotiate link parameters, such as auto-negotiated speed, or auto-negotiated flow control settings [6].

5.1 LLDP vehicle

DCBx uses the Link Layer Discovery Protocol, or LLDP, to transfer the DCBx configuration frames between link partners. The LLDP frames carry the configuration required to successfully negotiate a DCB link. The protocol also requires that a DCBx negotiation that cannot be resolved (i.e. configuration mismatch) mark the link as failed to negotiate, and disable DCB on the port.

LLDP also carries an application TLV (meaning type, length, and value [6]). This includes information required for applications needing to negotiate parameters with DCBx outside of the stack DCBx parameters. An example is FCoE: FCoE needs to find on which priority it will be resident in DCB. This way, FCoE knows which queues to configure in the base network driver, plus it can make software stack adjustments to properly feed the underlying network driver.

5.2 dcbd userspace tools

Linux has a set of userspace tools that implements the DCBx protocol. These tools also push the DCB configuration parameters into the registered network drivers. These tools are part of the dcbd package, which include the dcbd daemon and the dcbtool command line utility. dcbd runs in the background and listens for rtnetlink events on devices that it is managing.

rtnetlink is a Linux kernel interface that allows userspace tools to send messages to kernel components. It is similar to traditional ioctls. Other implementations of rtnetlink interfaces include network interface control (ifconfig commands), VLAN control (vconfig), and Qdisc manipulation (tc).

dcbd learns about network devices when it starts, and when any new device is brought online by listening to link up events from rtnetlink. dcbtool can be used to display the current DCB configuration of a device, manage the configuration of a device, and also toggle DCB mode on and off on a device. The dcbd userspace tools are available on Source Forge at `http://e1000.sf.net`.

6 DCB support in the Linux kernel

Data Center Bridging is fully supported in the Linux kernel as of 2.6.29. To date, the only driver making use of DCB features is ixgbe, using the DCB support found in the 82598 and 82599 devices. The main piece of DCB that is resident in the kernel is the configuration API used by dcbd utilities. The layer is called dcbnl, and is an rtnetlink interface. The interface has a full complement of get/set commands to configure each parameter of a DCB device. dcbd uses this interface to pull all DCB-related configuration to feed the DCBx negotiation, and in turn uses the interface to reprogram the device with any new DCB configuration returning from DCBx.

The other portion that DCB makes use of in the kernel is the QoS layer, which was previously discussed. The sch_multiq qdisc is the main workhorse, along with the tc filter act_skbedit action mechanism. These two QoS pieces help filter skbs into their proper priority flows in the underlying base driver.

7 Typical DCB deployment model

Now that DCB components have been defined, it's time to take a look at what a DCB deployment model looks like in the data center. The typical deployment today is a composed of two flows, one being storage traffic (FCoE) and the other being all LAN traffic. The LAN traffic can be spread across the other seven priority flows, if the traffic patterns warrant that many prioritized flows in your network. This is a network-dependent setting. Refer to Figure 4 for the order of startup. The steps are numbered.

From here, the FCoE instance will query dcbd through DCBx, using the application TLV, requesting which 802.1P priority it needs to use. Once the priority is provided (default of priority 3), the FCoE tools create tc

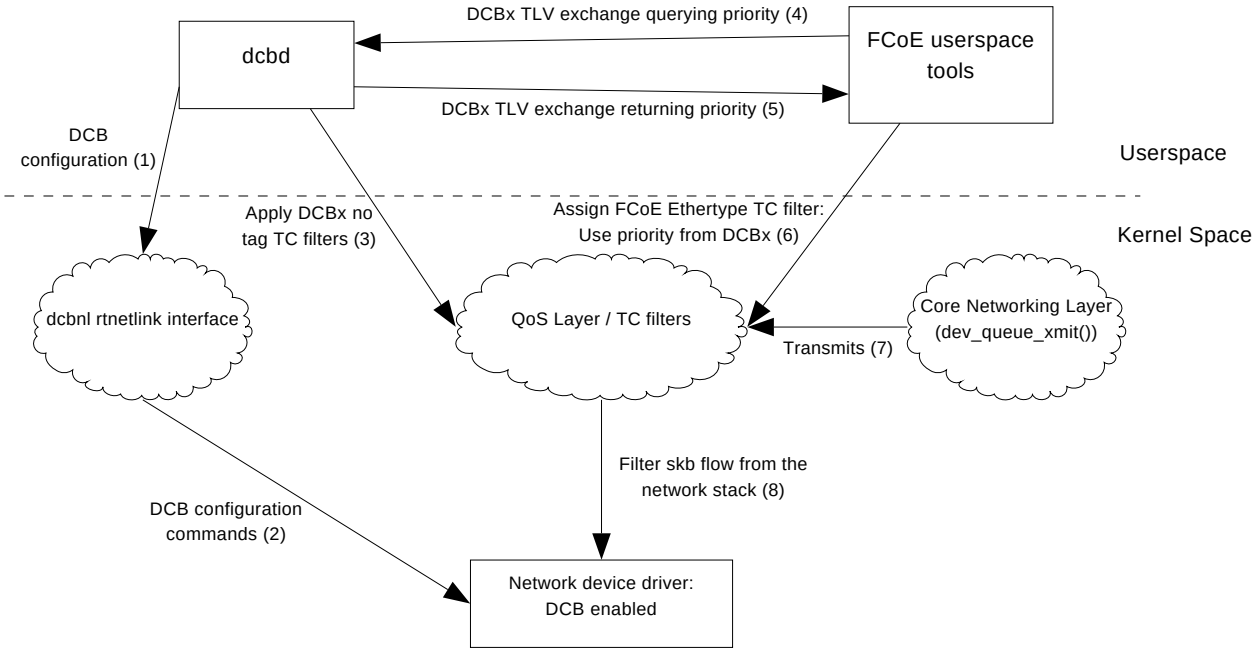

Figure 4: Typical DCB Deployment Model

filters that filter the FCoE ethertype (0x8906) [4]. These filters use the skbedit action to direct the matched flows into flow id 3. The base driver will then use the value of `skb->queue_mapping`, which is set by the skbedit action, to select which transmit queue the base driver allocated for that priority.

On the LAN side, other tc filters can be added by either dcbd or by the system administrator. A typical filter that is added is to match LLDP traffic, and skbedit the `skb->priority` field to be a control frame. That way the base driver can look for that setting, and not set a priority tag on the frame. LLDP frames should not be tagged with priorities within DCB, since they're control traffic.

Once the DCBx negotiation is finished with the switch or link partner, the DCB hardware is ready to use. If all the tc filters are in place, then DCB networking is running.

8 Conclusion

Ethernet networks will continue to push the speed envelope. As more 10GbE (and beyond) devices continue to pour into the market, these Ethernet networks will be cheaper to deploy. With data centers pushing the envelope to lower cost of operation with increased computing power and efficiency, Converged Networking will help realize these goals.

References

[1] Bert Hubert, Thomas Graf, et al. *Linux advanced Routing and Traffic Control* `http://lartc.org`

[2] Manoj Wadekar, Mike Ko, Ravi Shenoy, Mukund Chavan *Priority Groups, Traffic Differentiation over converged link (802.1Qaz)*

[3] Hugh Barrass *Definition for new PAUSE function (802.1Qbb)*

[4] Open-FCoE team *Open-FCoE.org homepage* `http://www.open-fcoe.org`

[5] Intel Corp. *Intel 82598 10GbE Ethernet Controller Datasheet*

[6] Intel Corp., Cisco Systems, Nuova Systems *DCB Capability Exchange Protocol Specification*

Proceedings of the Linux Symposium

July 13th–17th, 2009
Montreal, Quebec
Canada

Conference Organizers

Andrew J. Hutton, *Steamballoon, Inc., Linux Symposium, Thin Lines Mountaineering*

Programme Committee

Andrew J. Hutton, *Steamballoon, Inc., Linux Symposium, Thin Lines Mountaineering*
James Bottomley, *Novell*
Bdale Garbee, *HP*
Dave Jones, *Red Hat*
Dirk Hohndel, *Intel*
Gerrit Huizenga, *IBM*
Alasdair Kergon, *Red Hat*
Matthew Wilson, *rPath*

Proceedings Committee

Robyn Bergeron
Chris Dukes, *workfrog.com*
Jonas Fonseca
John 'Warthog9' Hawley

With thanks to
John W. Lockhart, *Red Hat*