Solution Brief

Intel Network and Edge Group

# Impact of Intel® Ethernet 800 Series Network Adapters with ADQ on CDN Performance

**Testing confirms that Intel Ethernet 800 Series Network Adapters with Application Device Queues (ADQ) can improve latency for content delivery network (CDN) workloads.[1]**

## Highlights

- **Intel Ethernet 800 Series Network Adapters feature Application Device Queues (ADQ), an Intel technology designed to prioritize application traffic.**
- **Intel's test results confirm that providers can increase response-time predictability and reduce latency for CDN workloads by adding Intel Ethernet 800 Series Network Adapters with ADQ to a high-performance platform.[1]**

## The challenge of peak capacity

The volume of digital content being created and delivered is exploding. Content owners, communications service providers (CoSPs), cloud service providers (CSPs), and others in this complex ecosystem need to address the approaching peak-capacity gap caused by the overwhelming, ever-increasing amounts of digital content.

Peak capacity is expected to grow by a factor of 10x to 20x over the next few years as the number of concurrent users watching video grows to more than a billion people.[2] Between 2021 and 2026, the cloud content delivery network (CDN) market is expected to grow at a compound annual growth rate (CAGR) of 12.8 percent.[3]

Current CDN infrastructure is unsustainable in the face of this rapid growth in peak capacity, as CDN providers will need to support a growing number of concurrent users, in addition to caching for ultra-high-definition (UHD) media and more immersive and personalized content.

CDNs face other pressures in addition to the looming capacity gap:

- CDN providers find it difficult to meet the quality-of-service (QoS) and quality-of-experience (QoE) requirements from content providers and over-the-top (OTT) content delivery, because existing providers do not have control over carrier networks.
- Growth in live streaming, watch-together apps, and interactive content has added to the importance of maintaining low latency, so viewers watching together across devices and locations have the same (or a very similar) viewing experience. Low-latency protocols, such as Common Media Application Format (CMAF), break streaming content into smaller chunks than did previous protocols. With content broken into more objects, however, the impact of latency issues is multiplied.
- Meeting unprecedented digital content-delivery demands requires compute capacity and powerful ways to move data faster and more predictably to ensure a good customer experience.

## Intel platforms for CDNs

Intel offers platform solutions to help overcome the infrastructure bottlenecks faced by CDNs. The need to encrypt massive quantities of web content, for example, requires tremendous compute power. Intel offers high-performance Intel® Xeon® Scalable processors with vector instructions to accelerate in-line encryption and Intel® Stratix® FPGA accelerators and Intel® QuickAssist Technology (Intel® QAT) to accelerate encryption work and reduce CPU cycles spent on cryptography processing.

For high-priority, network-intensive CDN workloads, Intel Ethernet 800 Series Network Adapters feature Application Device Queues (ADQ), an Intel technology designed to prioritize application traffic to deliver the performance required to meet QoS and QoE requirements.

Network-connectivity performance is measured by:

1. Throughput, how much data can be moved from one location to another in a given amount of time
2. Latency, the average amount of delay in communication, measured in microseconds
3. Application response time predictability, measured by latency variability, or tail latency, which is the slowest application response times (with the largest amount of delay) out of all response times from a system

When more servers are added to support growing CDN demands, tail latency becomes a limiting factor to scalability. Improving the predictability of application workload-response times by improving tail latency can enable more compute platforms to be allocated to deliver a better end-user experience.

This brief presents test results showing performance improvements achieved by including Intel Ethernet 800 Series Network Adapters with ADQ in a high-performance Intel platform running CDN workloads.

## ADQ for network-intensive CDN workloads

Managing the variability in CDN response times can be challenging. Increasing the predictability of response times by improving tail latency provides a better end-user experience. With a higher level of consistency, CDN providers can meet QoS and QoE requirements more easily.

Intel Ethernet 800 Series Network Adapters feature ADQ, an Intel technology designed to prioritize application traffic to deliver the performance required for high-priority, network-intensive content-delivery workloads. By performing application-specific queuing and steering, ADQ can reduce latency, improve response time predictability, and increase throughput.

ADQ acts like Ethernet express lanes for important application data; it dedicates queues and shapes traffic for the transfer of data over Ethernet.

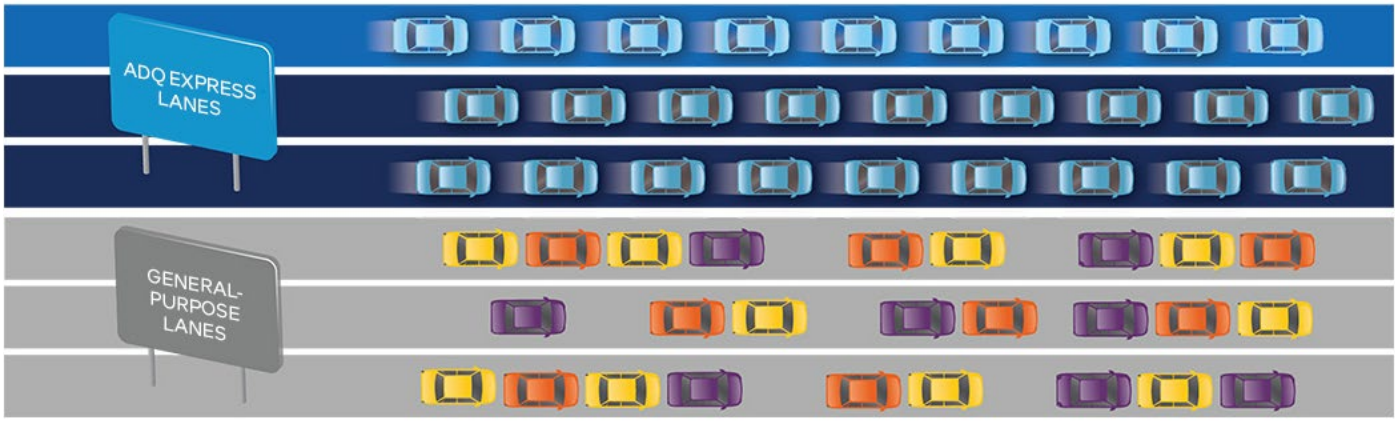


**Figure 1.** ADQ dedicates lanes and shapes traffic for the transfer of data over Ethernet

ADQ version 1.0 was released in 2019, using an application-dependent (triggered) polling approach. ADQ version 2.0 now offers an alternative to application-dependent polling, where polling is triggered independent of the application. ADQ 2.0 enables application-agnostic ADQ acceleration, where the queueing and steering for ADQ workloads is done independently from the application threads. Linux kernel threads are used to handle busy polling on behalf of the application. This enhancement provides ADQ acceleration without application workload changes to align thread to queue.

ADQ 2.0 delivers new and enhanced steering modes that enable the acceleration of workloads running in containers or virtual machines (VMs). With 2.0, ADQ also offers a setup tool (https://pypi.org/project/adqsetup/) that makes ADQ easier to set up and deploy. The script accepts parameters as command-line input or standard JavaScript Object Notation (JSON) format, and then generates and executes commands in the correct order and with correct syntax. The configuration can be saved and deployed across additional servers or clusters, and there is a persistence capability across reboots.

## Test setup

The setup used to test CDN performance included an edge cache node and four test generators (see Figure 2). The edge cache node included dual Intel Xeon Platinum 8360Y processors and four Intel Ethernet Network Adapters E810-CQDA2 (two per NUMA node), in addition to the Intel SSD DC P4510 Series. The test-generator nodes included dual-socket Intel Xeon Platinum and Gold processors, one Intel Ethernet Network Adapter E810-CQDA2, and the Intel SSD DC P4510 Series. The edge cache node ran Apache Traffic Server, and the test generators ran the Vegeta load-testing tool. Systems were connected via a 100 gigabit per second (Gbps) Ethernet switch.

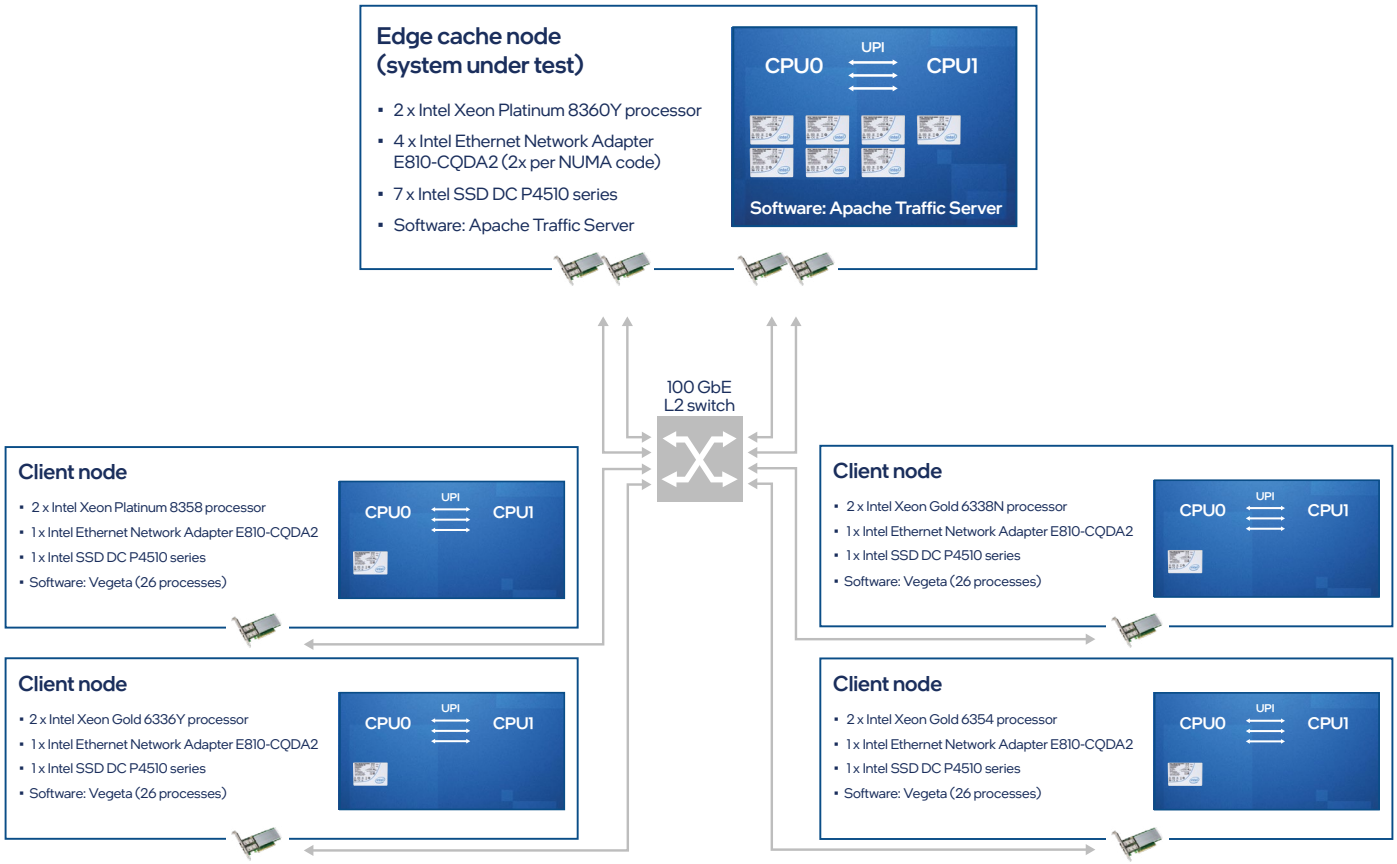


**Figure 2.** Test setup with one edge cache node (system under test) and four test-generator nodes

## Test results

To quantify the predictability and latency improvements that could be achieved by using ADQ, CDN workloads on Apache Traffic Server were run for small 10 KB web objects and 4 MB objects, a size commonly associated with standard video traffic. Normalized results were calculated for mean latency and for latency at the 99th and 99.5th percentiles. For a baseline configuration, Intel tested systems configured without ADQ enabled, then tested the same systems with ADQ enabled.

Results showed significant improvements in predictability and latency with ADQ enabled.[1]

Figure 3 illustrates the results of testing ADQ with CDN workloads on Apache Traffic Server using 10 KB web objects. ADQ achieved a 52-percent improvement in mean latency, an 83-percent improvement in the 99th percentile, and a 69-percent improvement in 99.5th-percentile tail latency, meaning more uniform latency distribution across all users.[1] This level of improvement can be attributed to reductions in the number of interrupts firing and in the number of context switches.

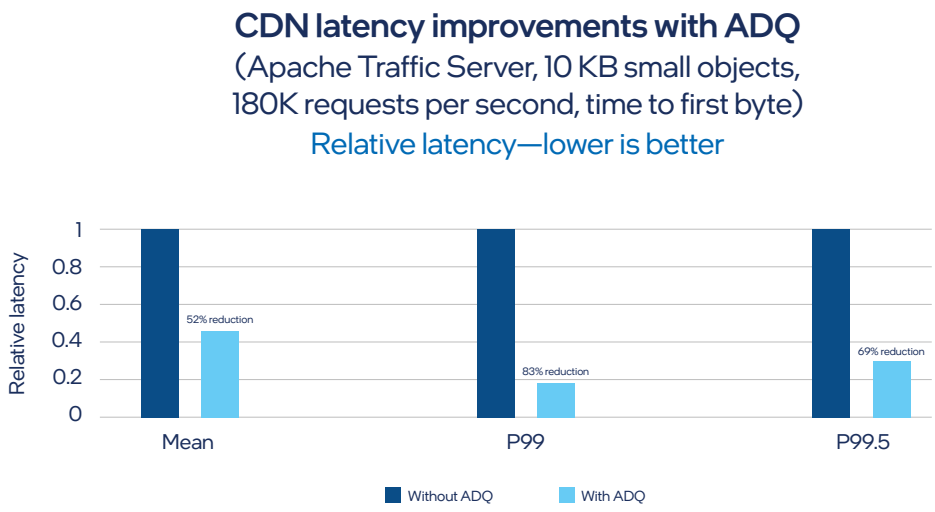


**Figure 3.** Latency improvement on ATS with 10 KB web objects[1]

Tests run with 4 MB objects also showed significant latency reductions with ADQ enabled. As illustrated in Figure 4, mean latency improved by 13 percent, and results for 99th and 99.5th percentile showed 37-percent and 35-percent reductions, respectively.[1]

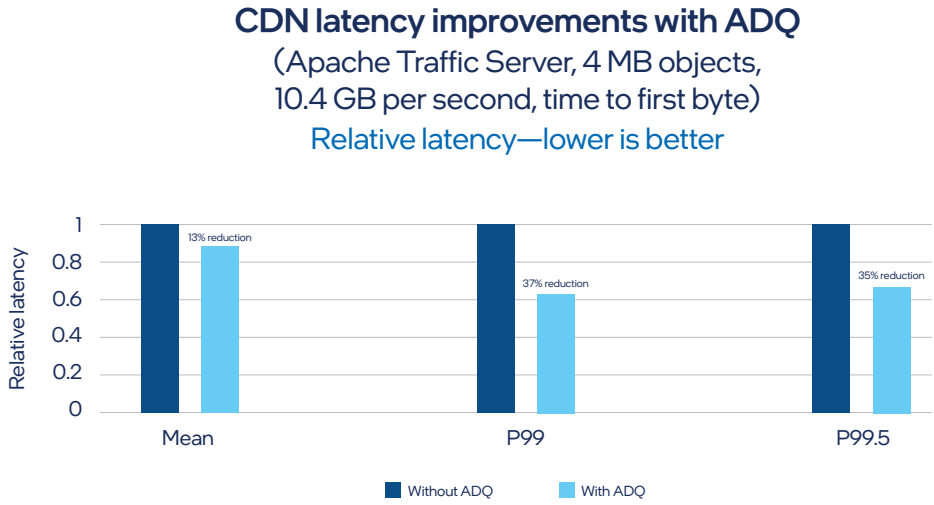


**Figure 4.** Latency improvement on ATS with 4 MB objects[1]

Figures 5 and 6 illustrate the results of testing time to first byte and time to last byte with 10 KB video objects at a rate of 180K requests per second. Tests for time to first byte with ADQ enabled achieved significant improvements, with latency reductions of 83 percent for 99th percentile and 70 percent for 99.5th percentile.[1] Test results for time to last byte showed similar improvements, with latency reductions of 83 percent for 99th percentile and 69 percent for 99.5th percentile with ADQ enabled.[1]

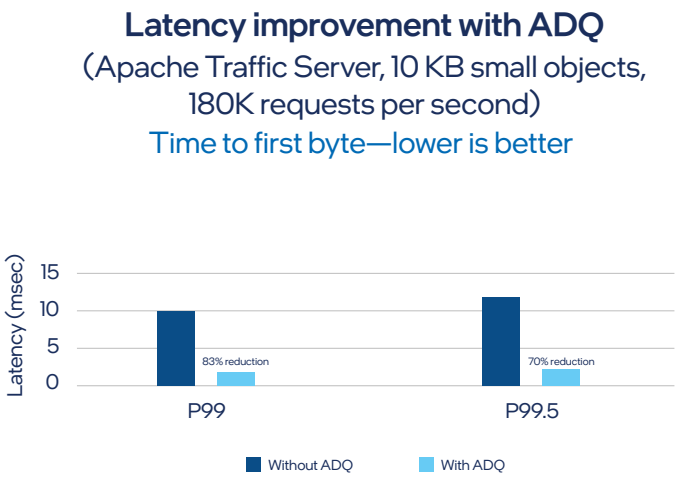


**Figure 5.** Latency improvement on ATS with 10 KB small objects (time to first byte)[1]

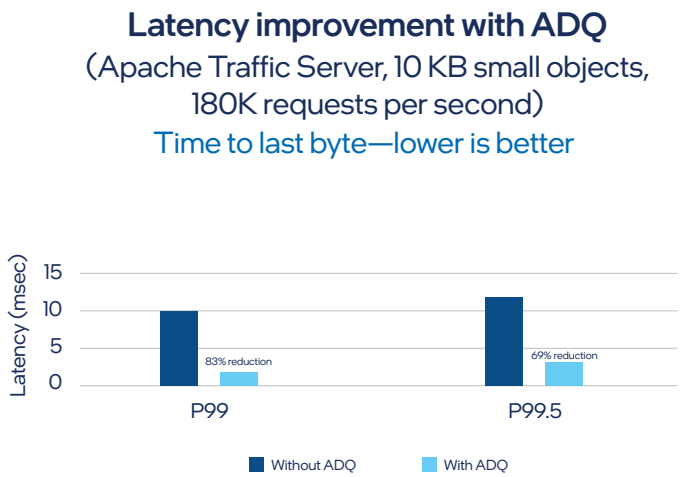


**Figure 6.** Latency improvement on ATS with 10 KB small objects (time to last byte)[1]

## Conclusion

These test results confirm that providers can reduce latency and get better predictability for CDN workloads by adding Intel Ethernet 800 Series Network Adapters with ADQ to a high-performance platform to help overcome typical infrastructure bottlenecks. Better predictability and significant reductions in network latency delivered by Intel Ethernet 800 Series Network Adapters with ADQ can improve the customer experience for streaming media and other content-delivery services as CDN providers deliver increasing amounts of streaming content to end users who have little tolerance for delays or interruptions.

Learn more about ADQ at intel.com/adq and "Intel Ethernet Controller E810 Application Device Queues (ADQ) Configuration Guide."

## Appendix: Configuration details

Testing was conducted by Intel on August 31, 2022.

**Table 1.** System under test configuration

| | Edge node (system under test) | Traffic generators |
|---|---|---|
| **Platform** | M50CYP2SBSTD | NF5280M6/NF5180M6/<br>M50CYP2SBSTD/M50CYP2SBSTD |
| **Number of nodes** | 2 | 2 |
| **Number of sockets** | 2 | 2 |
| **CPU** | 2 x Intel Xeon Platinum 8360Y processor (2.40 GHz) | 2 x Intel Xeon Gold 6338N processor (2.20 GHz)<br>2 x Intel Xeon Gold 6354 processor (3.00 GHz)<br>2 x Intel Xeon Gold 6336Y processor (2.40 GHz)<br>2 x Intel Xeon Platinum 8358 processor (2.60 GHz) |
| **Cores/socket and threads/socket** | 36/72 | 32/64; 18/36; 24/48; 32/64 |
| **Microcode** | 218104448 | 218104448/218104480/<br>218104497/218104626 |
| **Intel Hyper-Threading Technology (Intel HT Technology)** | On | On |
| **Intel Turbo Boost Technology** | On | On |
| **Power management** | Disabled | Disabled |
| **BIOS version** | SE5C620.86B.01.01.0003.2104260124 | AMI 05.00.02/AMI 05.01.01/<br>SE5C6200.86B.0022.D64.2105220049/<br>SE5C620.86B.01.01.0005.2202160810 |
| **System DDR memory configuration: slots/cap/speed** | 16 slots/32 GB/DDR4 3,200 megatransfers per second (MT/s) | 16 slots + 8 slots/32 GB + 253 GB/<br>DDR4 3,200 MHz + LRDIMM 3,200 MHz<br>16 slots/16 GB/DDR4 3,200 MHz<br>8 slots/16 GB/DDR4 3,200 MHz<br>16 slots/32 GB/DDR4 3,200 MHz |
| **Total memory (DDR and Intel® Optane™ persistent memory [PMem])** | 512 GB | 2.5 TB (512 GB DDR4/2 TB Intel Optane LRDIMM)<br>256 GB DDR4<br>128 GB DDR4<br>512 GB DDR4 |
| **PCH** | Intel C620 Series Chipset | 4 x Intel C620 Series Chipset |
| **Operating system (OS)/settings** | CentOS Stream 8.5<br>Stopped and disabled: firewalld | CentOS Stream 8.5<br>Stopped and disabled: firewalld |
| **Version** | Kernel 5.15.16 | Kernel 5.15.16 |
| **BIOS settings** | Default except for the following:<br>Power and Performance > CPU Power and Perf Policy > Performance Advanced > Power & Performance -> CPU C State Control > Package C-State = C0/C1 State | Default |

**Table 2.** Adapter configuration for Intel Ethernet Network Adapter E810-CQDA2

**Baseline adapter configuration**

```
tuned-adm profile latency-performance
x86_energy_perf_policy performance

systemctl stop irqbalance
systemctl stop firewalld

sysctl -w net.core.busy_poll=0
sysctl -w net.core.somaxconn=4096
sysctl -w net.core.netdev_max_backlog=250000
sysctl -w net.ipv4.tcp_max_syn_backlog=250000
sysctl -w net.core.rmem_max=568435456
sysctl -w net.core.wmem_max=568435456
sysctl -w net.ipv4.tcp_mem="764688 1019584 167772160"
sysctl -w net.ipv4.tcp_rmem="10240 87380 125829120"
sysctl -w net.ipv4.tcp_wmem="10240 87380 125829120"
sysctl -w net.ipv4.route.flush=1

ethtool -L ens801f0 combined 36
ethtool -L ens802f0 combined 36
ethtool -L ens785f0 combined 36
ethtool -L ens786f0 combined 36
../scripts/set_irq_affinity -x 0-35,72-107 ens785f0
../scripts/set_irq_affinity -x 0-35,72-107 ens786f0
../scripts/set_irq_affinity -x 36-71,108-143 ens801f0
../scripts/set_irq_affinity -x 36-71,108-143 ens802f0
```

**ADQ 2.0 adapter configuration**

```
tuned-adm profile latency-performance
x86_energy_perf_policy performance

systemctl stop irqbalance
systemctl stop firewalld

sysctl -w net.core.busy_poll=0
sysctl -w net.core.somaxconn=4096
sysctl -w net.core.netdev_max_backlog=250000
sysctl -w net.ipv4.tcp_max_syn_backlog=250000
sysctl -w net.core.rmem_max=568435456
sysctl -w net.core.wmem_max=568435456
sysctl -w net.ipv4.tcp_mem="764688 1019584 167772160"
sysctl -w net.ipv4.tcp_rmem="10240 87380 125829120"
sysctl -w net.ipv4.tcp_wmem="10240 87380 125829120"
sysctl -w net.ipv4.route.flush=1
```

```
ethtool -K ens785f0 hw-tc-offload on
ethtool -K ens786f0 hw-tc-offload on
ethtool -K ens801f0 hw-tc-offload on
ethtool -K ens802f0 hw-tc-offload on

tc qdisc add dev ens785f0 root mqprio num_tc 2 map 0 1 queues 2@0 26@2 hw 1 mode channel
tc qdisc add dev ens785f0 ingress
tc qdisc add dev ens786f0 root mqprio num_tc 2 map 0 1 queues 2@0 26@2 hw 1 mode channel
tc qdisc add dev ens786f0 ingress
tc qdisc add dev ens801f0 root mqprio num_tc 2 map 0 1 queues 2@0 26@2 hw 1 mode channel
tc qdisc add dev ens801f0 ingress
tc qdisc add dev ens802f0 root mqprio num_tc 2 map 0 1 queues 2@0 26@2 hw 1 mode channel
tc qdisc add dev ens802f0 ingress
sleep 5

tc filter add dev ens785f0 protocol ip ingress prio 1 flower dst_ip 192.168.125.125/32 ip_proto tcp
dst_port 8443 skip_sw hw_tc 1
tc filter add dev ens785f0 protocol ip ingress prio 1 flower dst_ip 192.168.125.125/32 ip_proto tcp
dst_port 8080 skip_sw hw_tc 1
tc filter add dev ens785f0 protocol ip ingress prio 1 flower dst_ip 192.168.125.125/32 ip_proto tcp
dst_port 5001 skip_sw hw_tc 1
tc filter add dev ens786f0 protocol ip ingress prio 1 flower dst_ip 192.168.100.105/32 ip_proto tcp
dst_port 8443 skip_sw hw_tc 1
tc filter add dev ens786f0 protocol ip ingress prio 1 flower dst_ip 192.168.100.105/32 ip_proto tcp
dst_port 8080 skip_sw hw_tc 1
tc filter add dev ens786f0 protocol ip ingress prio 1 flower dst_ip 192.168.100.105/32 ip_proto tcp
dst_port 5001 skip_sw hw_tc 1
tc filter add dev ens801f0 protocol ip ingress prio 1 flower dst_ip 192.168.150.125/32 ip_proto tcp
dst_port 8443 skip_sw hw_tc 1
tc filter add dev ens801f0 protocol ip ingress prio 1 flower dst_ip 192.168.150.125/32 ip_proto tcp
dst_port 8080 skip_sw hw_tc 1
tc filter add dev ens801f0 protocol ip ingress prio 1 flower dst_ip 192.168.150.125/32 ip_proto tcp
dst_port 5001 skip_sw hw_tc 1
tc filter add dev ens802f0 protocol ip ingress prio 1 flower dst_ip 192.168.200.235/32 ip_proto tcp
dst_port 8443 skip_sw hw_tc 1
tc filter add dev ens802f0 protocol ip ingress prio 1 flower dst_ip 192.168.200.235/32 ip_proto tcp
dst_port 8080 skip_sw hw_tc 1
tc filter add dev ens802f0 protocol ip ingress prio 1 flower dst_ip 192.168.200.235/32 ip_proto tcp
dst_port 5001 skip_sw hw_tc 1

ethtool --coalesce ens785f0 adaptive-rx off rx-usecs 100
ethtool --coalesce ens785f0 adaptive-tx off tx-usecs 100
ethtool --coalesce ens786f0 adaptive-rx off rx-usecs 100
ethtool --coalesce ens786f0 adaptive-tx off tx-usecs 100
ethtool --coalesce ens801f0 adaptive-rx off rx-usecs 100
ethtool --coalesce ens801f0 adaptive-tx off tx-usecs 100
ethtool --coalesce ens802f0 adaptive-rx off rx-usecs 100
ethtool --coalesce ens802f0 adaptive-tx off tx-usecs 100
```

```
../scripts/set_irq_affinity -X all ens785f0
../scripts/set_xps_rxqs ens785f0
../scripts/set_irq_affinity -X all ens786f0
../scripts/set_xps_rxqs ens786f0
../scripts/set_irq_affinity -X all ens801f0
../scripts/set_xps_rxqs ens801f0
../scripts/set_irq_affinity -X all ens802f0
../scripts/set_xps_rxqs ens802f0

ethtool --set-priv-flags ens785f0 channel-pkt-inspect-optimize off
ethtool --set-priv-flags ens786f0 channel-pkt-inspect-optimize off
ethtool --set-priv-flags ens801f0 channel-pkt-inspect-optimize off
ethtool --set-priv-flags ens802f0 channel-pkt-inspect-optimize off

echo 1 > /sys/class/net/ens785f0/threaded
echo 1 > /sys/class/net/ens786f0/threaded
echo 1 > /sys/class/net/ens801f0/threaded
echo 1 > /sys/class/net/ens802f0/threaded

cgcreate -g cpuset,memory,net_prio:ats
cgset -r net_prio.ifpriomap="ens785f0 1" ats
cgset -r net_prio.ifpriomap="ens786f0 1" ats
cgset -r net_prio.ifpriomap="ens801f0 1" ats
cgset -r net_prio.ifpriomap="ens802f0 1" ats
cgset -r cpuset.mems="0-1" ats
cgset -r cpuset.cpus="8-35,44-71,80-107,116-143" ats

devlink dev param set pci/0000:4b:00.0 name tc1_qps_per_poller value 7 cmode runtime
devlink dev param set pci/0000:4b:00.0 name tc1_poller_timeout value 500 cmode runtime
devlink dev param set pci/0000:4b:00.0 name tc1_inline_fd value true cmode runtime
devlink dev param set pci/0000:31:00.0 name tc1_qps_per_poller value 7 cmode runtime
devlink dev param set pci/0000:31:00.0 name tc1_poller_timeout value 500 cmode runtime
devlink dev param set pci/0000:31:00.0 name tc1_inline_fd value true cmode runtime
devlink dev param set pci/0000:b1:00.0 name tc1_qps_per_poller value 7 cmode runtime
devlink dev param set pci/0000:b1:00.0 name tc1_poller_timeout value 500 cmode runtime
devlink dev param set pci/0000:b1:00.0 name tc1_inline_fd value true cmode runtime
devlink dev param set pci/0000:ca:00.0 name tc1_qps_per_poller value 7 cmode runtime
devlink dev param set pci/0000:ca:00.0 name tc1_poller_timeout value 500 cmode runtime
devlink dev param set pci/0000:ca:00.0 name tc1_inline_fd value true cmode runtime
```

```
CHK () {
    “$@”
    if [ $? -ne 0 ]; then
        echo “Error with ${1}, stopping now!” >&2
        exit 1
    fi
}

CHK () {
    “$@”
    if [ $? -ne 0 ]; then
        echo “Error with ${1}, stopping now!” >&2
        exit 1
    fi
}

    num_threads=$(ps -aef |grep “\[napi\/ens785f0” | wc -l)
    start_pid=$(ps -ae |grep napi\/ens785f0 |head -n 1 |cut -d ? -f 1)

    for (( i = 0; i < $num_threads; i++ ))
    do
        printf -v pid “%d” $((start_pid+i))
        printf -v cpu “%d” $((2*i+1))
        CHK taskset -p -c “0-7” $pid
    done

    num_threads=$(ps -aef |grep “\[napi\/ens786f0” | wc -l)
    start_pid=$(ps -ae |grep napi\/ens786f0 |head -n 1 |cut -d ? -f 1)

    for (( i = 0; i < $num_threads; i++ ))
    do
        printf -v pid “%d” $((start_pid+i))
        printf -v cpu “%d” $((2*i+1))
        CHK taskset -p -c “0-7” $pid
    done

    num_threads=$(ps -aef |grep “\[napi\/ens801f0” | wc -l)
    start_pid=$(ps -ae |grep napi\/ens801f0 |head -n 1 |cut -d ? -f 1)

    for (( i = 0; i < $num_threads; i++ ))
    do
        printf -v pid “%d” $((start_pid+i))
        printf -v cpu “%d” $((2*i+1))
        CHK taskset -p -c “36-43” $pid
```

```
done

num_threads=$(ps -aef |grep "\[napi\/ens802f0" | wc -l)
start_pid=$(ps -ae |grep napi\/ens802f0 |head -n 1 |cut -d ? -f 1)

for (( i = 0; i < $num_threads; i++ ))
do
    printf -v pid "%d" $((start_pid+i))
    printf -v cpu "%d" $((2*i+1))
    CHK taskset -p -c "36-43" $pid
done
```

[1] See backup for workloads and configurations. Results may vary.
[2] Akamai Edge Conference 2019; Ericsson Mobility Report 2020.
[3] Mordor Intelligence. "Cloud CDN Market - Growth, Trends, COVID-19 Impact, and Forecasts (2021 - 2026)." January 2021. researchandmarkets.com/reports/4833397/cloud-cdn-market-growth-trends-covid-19. And: Intricately. "CDN Industry: Trends, Size, And Market Share." July 2020. https://blog.intricately.com/2020-state-of-the-cdn-industry-trends-market-share-customer-size.

Performance varies by use, configuration and other factors. Learn more at www.Intel.com/PerformanceIndex.

Performance results are based on testing as of dates shown in configurations and may not reflect all publicly available updates. See configuration details in this document. No product or component can be absolutely secure.

Your costs and results may vary.

Intel technologies may require enabled hardware, software or service activation.

Intel does not control or audit third-party data. You should consult other sources to evaluate accuracy.

© Intel Corporation. Intel, the Intel logo, and Stratix are trademarks of Intel Corporation or its subsidiaries. Other names and brands may be claimed as the property of others.

Printed in USA 1022/SC/PRW/PDF Please Recycle 348512-001US