Workgroup: DETNET
Internet-Draft: draft-eckert-detnet-tcqf-05
Published: 6 January 2024
Intended Status: Standards Track
Expires: 9 July 2024
Authors: T. Eckert, Ed.
Futurewei Technologies USA
Y. Li, Ed.
Huawei Technologies
S. Bryant
University of Surrey ICS
A. G. Malis
Malis Consulting
J.-d. Ryoo
ETRI
P. Liu
China Mobile
G. Li
Huawei Technologies
S. Ren
Huawei Technologies
F. Yang
Huawei Technologies

Deterministic Networking (DetNet) Data Plane - Tagged Cyclic Queuing and Forwarding (TCQF) for bounded latency with low jitter in large scale DetNets

Abstract

This memo specifies a forwarding method for bounded latency and bounded jitter for Deterministic Networks and is a variant of the IEEE TSN Cyclic Queuing and Forwarding (CQF) method. Tagged CQF (TCQF) supports more than 2 cycles and indicates the cycle number via an existing or new packet header field called the tag to replace the cycle mapping in CQF which is based purely on synchronized reception clock.

This memo standardizes TCQF as a mechanism independent of the tagging method used. It also specifies tagging via the (1) the existing MPLS packet Traffic Class (TC) field for MPLS packets, (2) the IP/IPv6 DSCP field for IP/IPv6 packets, and (3) a new TCQF Option header for IPv6 packets.

Target benefits of TCQF include low end-to-end jitter, ease of high-speed hardware implementation, optional ability to support large number of flow in large networks via DiffServ style aggregation by applying TCQF to the DetNet aggregate instead of each DetNet flow individually, and support of wide-area DetNet networks with arbitrary link latencies and latency variations as well as low accuracy clock synchronization.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute

working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 9 July 2024.

Copyright Notice

Table of Contents

1. Introduction
  1.1. Terminology
2. Overview (informative)
  2.1. Cyclic Queuing and Forwarding (CQF)
  2.2. Benefits of CQF with higher speed links
  2.3. Challenges of CQF with higher latency links
  2.4. Review of CQF benefits and challenges for DetNet
  2.5. Tagged CQF
    2.5.1. CQF with more than two buffers
    2.5.2. From CQF with multiple buffers to TCQF
  2.6. Summary of TCQF benefits and goals for DetNet
3. Using TCQF in the DetNet Architecture and MPLS forwarding plane (informative)
4. TCQF per-flow stateless forwarding (normative)
  4.1. Configuration Data model and tag processing for MPLS TC tags
  4.2. Packet processing
  4.3. TCQF for MPLS with TC tagging
  4.4. TCQF for IP/IPv6 with DSCP tagging
  4.5. TCQF for IPv6 with IPv6 Option tagging
    4.5.1. TCQF Option Format
    4.5.2. TCQF Option Processing

4.5.3. Encapsulation of TCQF Option for Deterministic IP (DIP) data plane
4.6. TCQF Pseudocode (normative)
5. TCQF Per-flow Ingress forwarding (normative)
5.1. Ingress Flows Configuration Data Model
5.2. Ingress Flows Pseudocode
6. Implementation, Deployment, Operations and Validation considerations (informative)
6.1. High-Speed Implementation
6.2. Controller plane computation of cycle mappings
6.3. Link speed and bandwidth sharing
6.4. Controller-plane considerations
6.5. Validation
7. Security Considerations
8. IANA Considerations
9. Acknowledgement
10. Contributors
11. Changelog
12. References
12.1. Normative References
12.2. Informative References
Appendix A. CSQF
Appendix B. TCQF with multiple priorities
Appendix C. TSN Multiple Buffer CQF
Authors' Addresses

1. Introduction

1.1. Terminology

CQF Cyclic Queuing and Forwarding. A queuing mechanism defined by annex T of [IEEE802.1Q].

DT Dead Time. A term from CQF indicating the time during each cycle in which no frames can be sent because the the receiving node could not receive it into the desired cycle buffer.

TCQF Tagged Cyclic Queuing and Forwarding. The mechanism specified in this memo.

2. Overview (informative)

2.1. Cyclic Queuing and Forwarding (CQF)

Cyclic Queuing and Forwarding (CQF) is a bounded (guaranteed) per-hop latency forwarding mechanism standardized for use in ethernet switched networks by the IEEE TSN working group originally via [IEEE802.1Qch] (802.1 Qch), which later became annex T of [IEEE802.1Q]. See also [RFC9320], Section 6.6.

CQF is not a separate forwarding mechanism, but it is simple a profile of the IEEE Time Aware Shaper (TAS) standard, [IEEE802.1Qbv], which introduce Time-Gated Queues.

CQF uses a two-queue based forwarding mechanism on every switch along a path between a sender and receiver. One queue is used to receive and store frames destined toward a particular outgoing interface on the switch, the other queue is used simultaneously to send frames to the same outgoing interface. At every cycle time T_c interval these two queues are swapped, or in terms of Time-Gated Queus, one is closed for sending, the other is opened for sending. This operation is synchronized across all switches in the network by network wide synchronized clocks, so that all queues open and close at the same time.

For a path of h hops, the end-to-end latency bound is between (h-1) * T_c + DT and (h+1) * T_c. DT is the so-called dead time at the end of a cycle during which no frames can be transmitted from the sending queue to ensure that the last byte of the last frame will be received earlier than the end of the same cycle on the receiving switch.

A core contributor to DT is the (physical) link between the sending and receiving switch. DT needs to be larger than the latency of this link, including physical propagation latency (speed of light), possible error correction latencies, and interface serialization latency.

T_c needs to be choosen carefully: The larger it is, the higher the bounded latency. The smaller it is, the fewer bytes (and hence frames) will fit into a cycle.

To admit flows into a CQF network, the ingress switch uses per-flow Time-Gated Queues. In the most simple case, such a gate is configured to admit a maximum amount of bytes from the flow into every cycle. More advanced admission control can be performed for bursty flows. For example N bursty flows f_i = 0...(N-1) could share admitted bandwidth by each having their burst admitted in different cycles c_i = c % N + c_i, where c is a continuous increasing cycle number.

2.2. Benefits of CQF with higher speed links

The typical CQF deployments in manufacturing networks with 1Gbps links uses no less than hundreds of microseconds as a cycle interval. In a network with a small diameter, say less than 8 hops, it is sufficiently good to provide an end-to-end latency bound in the order of several milliseconds.

With the increasing of link speed from 100Mbps to 1Gbps, 10Gbps, 100Gbps or even higher in larger networks, either more bytes can be transmitted within the same cycle interval or the smaller cycle interval is required to transmit the same amount of bytes in a cycle as that in low speed networks. Likewise, the serialization latency reduces with higher speed links and DT reduces. This overall makes CQF for higher speed networks more attractive than for lower speed networks.

Figure 1 shows a simple calculation on the number of bytes that can be transmitted in a cycle with different cycle intervals and link speeds. A minimum of 1500 bytes is labeled with * as a baseline because a typical maximum Ethernet frame is 1500 bytes and a selected cycle interval should at least allow one such frame size to be transmitted unless otherwise specified.

TBD: These numbers probbly need to be adjusted to reflect reducing DT based on serialization latency.

Cycle Time	Bytes Transmitted in a Cycle				
	Link Speed				
(us)	100Mbps	1Gbps	10Gbps	100Gbps	
1	12.5	125	1250	12500*	
1.2	15	150	1500*	15000	
2	25	250	2500	25000	
4	50	500	5000	50000	
10	125	1250	12500	125000	
12	150	1500*	15000	150000	
120	1500*	15000	150000	1500000	

Figure 1: Bytes transmitted within one cycle interval

When the link speed is at 10Gbps, the cycle interval could be as small as 1.2 us if a 1500 byte frame needs to be transmitted in one cycle interval, and with 100Gbps links even 1 usec cycle time allows for 8 frames of 1500 byte each. These are not accurate calculations because there are certainly other factors to determine the cycle interval. However, it shows that as the link speed increases, cycle

interval can be greatly reduced in practice while satisfying the minimum amount of data transmitted in a single cycle. The end-to-end latency bound when applying CQF is determined by cycle interval and number of hops. That is to say, CQFs with a smaller cycle interval have the potential to meet more strict end-to-end latency requirements in higher link speed networks or meet the same end-to-end latency requirement in networks with much larger network diameter (number of hops).

Industry automation has some typical application period requirement, e.g. 100 us to 2 ms for isochronous traffic, 500 us to 1 ms for cyclic-synchronous and 2 to 20 ms for cyclic-asynchronous traffic. The network cycle interval is usually a fraction of the application period. When the cycle interval is in the order of tens of microseconds, CQF can be used to meet the most strict end-to-end latency requirements. For instance, if we assume the number of hops is 24, when cycle interval is set to 10us, the end-to-end latency bound can be around (24+1)*10 = 250 us which has the potential to meet the latency bound requirement for isochronous traffic.

In summary a higher speed network makes the shorter cycle interval feasible because sufficiently large traffic volume can be transmitted within one cycle interval. A shorter cycle interval further offers shorter end-to-end latency and jitter bounds which provide CQF with the potentials to meet more strict latency requirements in wider deployments while preserving its simplicity of latency calculation and provisioning. Therefore there is a strong motivation to leverage CQF and at the same time to make cycle interval as short as possible.

2.3. Challenges of CQF with higher latency links

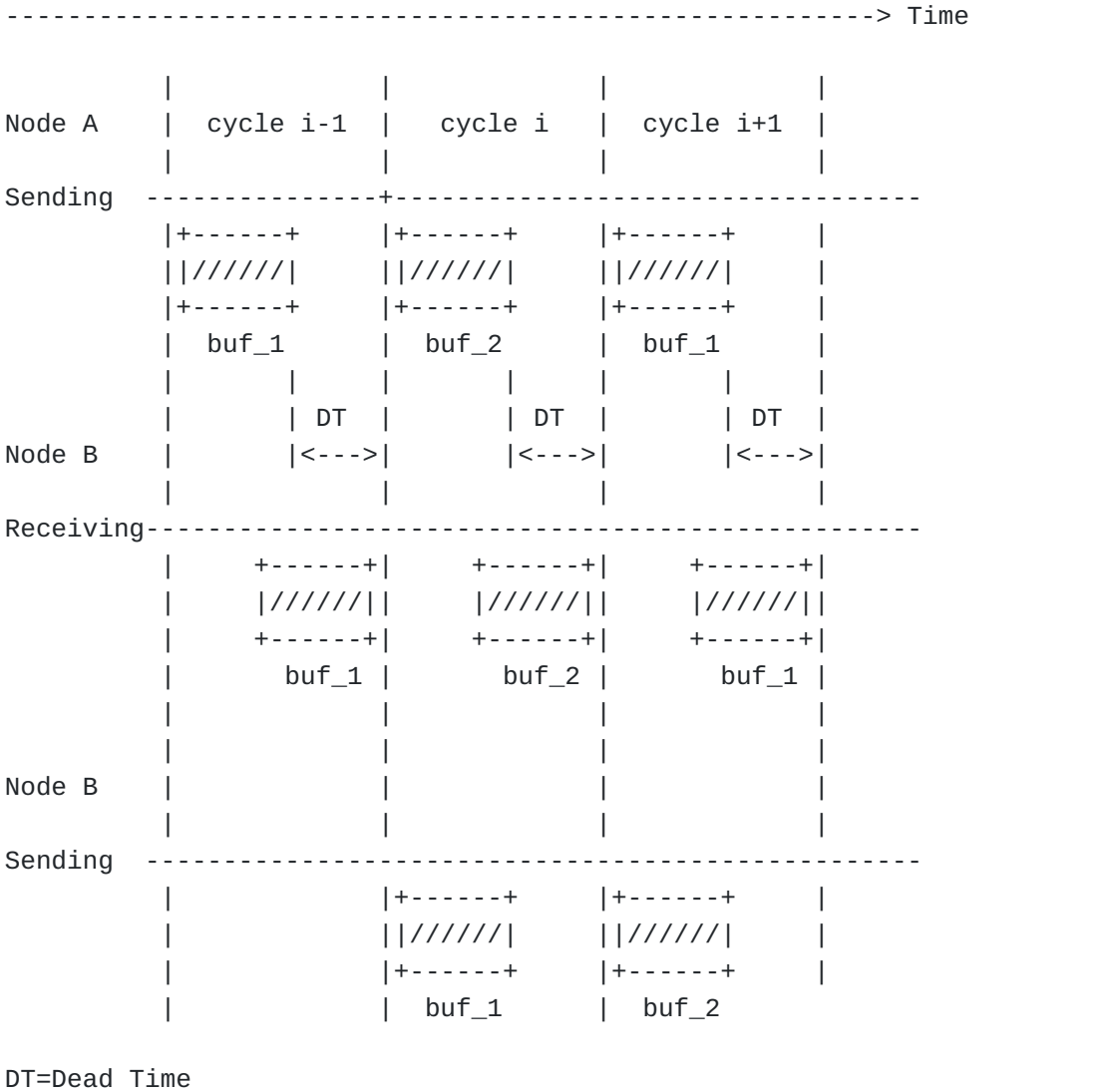
Unlike the original targets for IEEE TSN work, DetNet not only targets to support IETF forwarding planes (IP, MPLS,...), but also wide-area networks with therefore longer physcial propagation latencies.

As shown in Figure 2 for fundamental (two buffer) CQF, the last byte sent by node A in cycle (i-1) has to be ready for sending at node B before the start of cycle i. To realize it, DT or dead time is imposed. It is a time interval usually at the end of a cycle so that a node should not send the scheduled CQF packets.

Dead time is at least the sum of the maximum propagation delay to the next node, the maximum processing delay at the next node and the maximum other time variations. Therefore either the longer propagation or longer processing delay makes dead time larger. Packets from DetNet service is likely to be propagated over long links in the wider area. It takes around 5us per kilometer to

propagate, i.e. 0.5ms every hundred kilometers. Hence the dead time can be as large as milliseconds or tens of milliseconds in case of hundred kilometers of longer links and larger processing delays. That would make the dead time eat up most of the cycle interval when cycle interval is short (e.g., at the same order or one order higher of magnitude in time as dead time). Then the useful time in a cycle will be much reduced. In some extreme cases, when the link is long and the cycle interval is set to extremely short, the first packet sent in a cycle by a node will not be possibly received in the same cycle interval at the next node. That makes the useful time in a cycle reaches zero in two buffer CQF. Then two buffer CQF will be no longer suitable.

In result of these considerations, reasonable limits for the size of TSN CQF networks are in the order of at most few Km per hop, beyond which DT exceeds common cycle times and possible through of CQF traffic is hence 0.



DT=Dead Time

Figure 2: Fundamental Two Buffer CQF

2.4. Review of CQF benefits and challenges for DetNet

In review, CQF has a range of benefits for DetNet.

1. It provides bounded latency.

2. It provided tightly bounded jitter.

3. It has a very simple and easily standardized calculus for its bounded latency and jitter.

4. It has very simple per-hop forwarding machinery (cyclic queues) easily supportable in high-speed network equipment.

5. Like Diffserv forwarding, it does not use per-hop,per-flow state in the forwarding plane and therefore does not require per-hop,per-flow signaling with the DetNet controller-plane, allowing it to scale to large number of flows.

6. The faster the links are, the lower the per-hop latency impact of the cyclic queuing mechanism.

The core limitation of CQF, which TCQF intends to solve, lies in its use of arrival time clock to determine the cycle into which the packet is to be placed, see [I-D.eckert-detnet-bounded-latency-problems] for more details.

1. Cycles times should be as short as feasible to support lower end to end latency (Section 2.2).

2. When networks have longer links, or links with higher propagation jitter as in Metro and WAN, this increases the dead time, and hence reduces the possible utilization or need to increase cycle times.

3. When shorter cycle times are feasible because of higher speed links, this would require an increase in clock-synchronization accuracy.

2.5. Tagged CQF

Tagging of CQF packets with cycle identifiers can be used to solve the dilemma aforementioned with minor changes to the fundamental two buffer CQF. This section introduces this mechanism with multipl buffers and CQF cycle identification in the packet header. Note that we are also now using the term packet (as used for IP, MPLS and other IETF forwarding planes) and buffers for packets, as opposed to frames as used by IEEE.

2.5.1. CQF with more than two buffers

CQF can use more than two buffers to minimize the dead time and increase the useful time in a cycle so as to support long link delay. Figure 3 shows how a three buffer CQF works in a rotating manner in general. Node A sends packets in cycle (i-1). The time interval over which node B receives these packet spans two cycles, cycle (i-1) and cycle i. Hence a method is needed to make node B send them all at once in cycle (i+1) in order to ensure packets in a single cycle from the previous node always being sent out in one cycle at the current node.

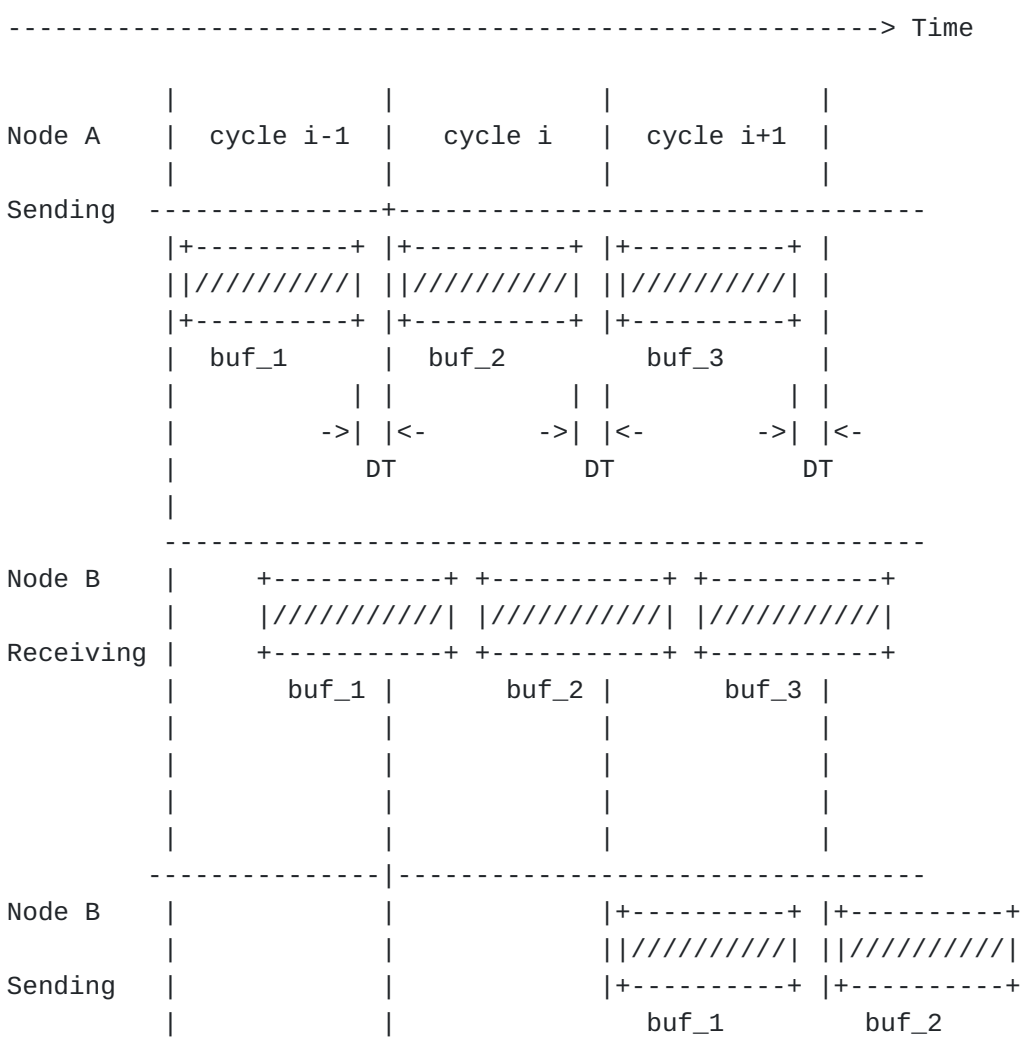


Figure 3: Three Buffer CQF

More than three buffers will be required when the receiving interval at node B for packets sent in a single cycle interval from node A spans over more than two cycle interval boundaries. This can happen

when the time variance (jitter) including propagation, processing, regulation, clock synchronization variance (so called Maximum Time Interval Error - MTIE) and other factors between two neighbouring DetNet nodes can become larger than a single cycle tim.

2.5.2. From CQF with multiple buffers to TCQF

Note that due to the variance in time, the receiving interval at the downstream node can be much larger than one cycle interval in which the upstream node transmits. When time variance is large and cycle interval and dead time are set small, the possible receiving time of the last few packets from node A’s cycle (i-1) at node B can overlap with the possible receiving time of the first few packets from node A’s cycle i in different rounds of buffer rotations. Hence, when the buffer number is larger than two, if the receiving side still uses the traditional CQF implicit time borderline to demarcate the receiving packets from the consecutive cycles of the upstream node, it may cause the ambiguity in identifying the right sending cycle at the upstream node and further affect the correctness of the decision of which output buffer to put the received packets at the current node.

Figure 4 shows such an ambiguity when time based cycle demarcation is used. The packet sent by node A in its cycle (i-1) can be received at any time in the receiving interval indicated as “receiving window for A’s buf_1” in Figure 4. The receiving window refers to the time interval between the earliest time that the first packet sent in a given cycle from an upstream node is processed and enqueued in an output buffer and the latest time that the last packet of the cycle is processed and enqueued in an output buffer. Network operators may configure the size of the receiving window, taking the time variance of their networks into account. It can be seen that the spanning time period of receiving window is longer than the cycle interval. This is because there is a large time variance experienced between A and B, e.g. varying processing time for different packets in different cycles. It does not mean the receiving interval for every cycle always constantly span over such a large receiving window. The receiving window time interval indeed is determined by the worst case time variance value and that should be used for regular time cycle demarcation.

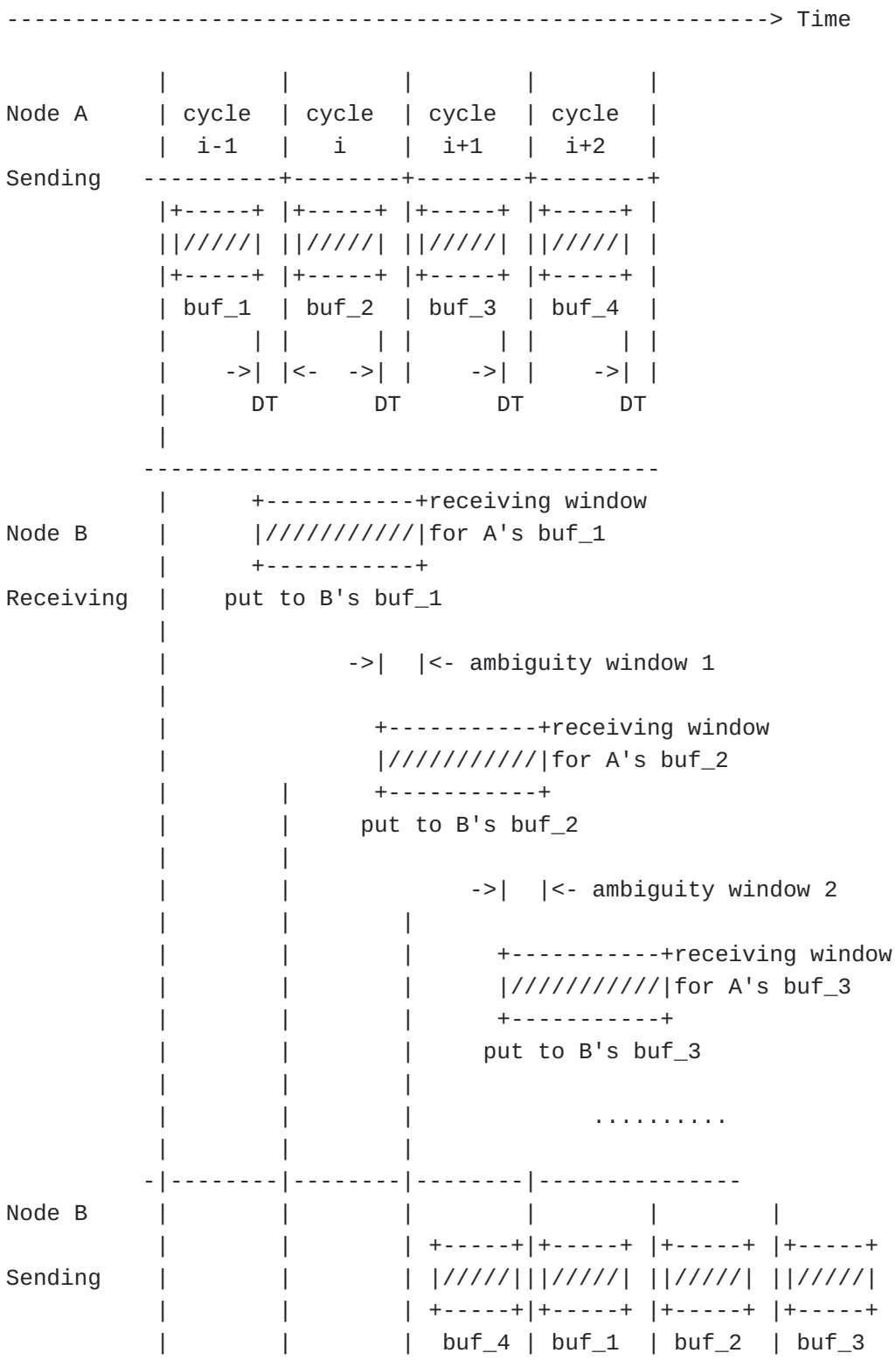


Figure 4: Three Buffer ambiguity

When a packet is received in ambiguity window 1 in Figure 4, node B is not able to use the receiving time to determine which buffer is

the correct one to put the packet in because it cannot tell if the packet is sent from cycle (i-1) or cycle i on node A. If node B puts the packet to the wrong output buffer, the packet may experience the unexpected delay. At the same time, the packet occupying the non-designated buffer may break the contracts between the end hosts and DetNet networks and then cause the unpredictable consequences.

It has been noted that the DT can be greatly increased to beat the time variance in order to make the receiving windows do not overlap so as to remove such ambiguity. However, it is not always practical and usually not desired because large DT will eat useful cycle time and bring the low utilization issue as illustrated in Section 2.3. Therefore, it would be desired to keep DT as small as possible and at the same time identify the cycle interval correctly.

With tagged CQF, the sending router A encodes the sending cycle identification in some existing or new packet header field as specified later in this document. This allows the receiving router B to determine the correct output port cycle buffer to place the data packet into. Except for the need for the operator to pre-configure this mapping on router B, based on the above described latency and jitter of the link (and processing between the sending and receiving router, tagging does not change the fundamental mechanism and benefits of CQF. makes no change from the fundamental CQF.

Compared to CQF with multiple buffers, Tagged CQF allows to operate with clock synchronization at significantly reduced accuracy requirements than CQF. In CQF, the MTIE is an addend determing DT and should hence typically be less than 1% of the cycle time. In TCQF it is an addent in the permitted receive window and can hence be for example as large as the cycle time, and such 100 times larger. A network using TCQF with 100Gbps interfaces can hence can hence use the same or less expensive clock synchronization setup than a CQF network with 1Gbps interfaces. In addition, when conditions of the network connections change, the mappings can dynamically changed from network operations.

CQF with multiple buffers but without tagging has been proposed to IEEE TSN in [multipleCQF], but has not been adopted. Instead of relying on a cycle tag in a packet header, it still relies solely on the arrival time of packet, and can hence not equally resolve arrival time ambiguities as TCQF can, because it does not know the cycle from which the packet was sent.

2.6. Summary of TCQF benefits and goals for DetNet

TCQF inherits the benefits of CQF for DetNet as outlined in Section 2.4, and byusing a configurable number of three or more

cycles, and signaling the cycle as part of a packet header, it resolves these problems as follows.

1. With three cycles, TCQF can support arbitrary latency links at arbitrary speeds without reduction of utilization because of longer links or higher link speeds (same cycle time, same clock accuracy, only change in lengths and speeds).

2. With four or more cycles, TCQF can also eliminate Dead Time caused by variation of clock synchronization inaccuracies (MTIE) as well as jitter caused by link propagation and processing variation. The sum of cycles times needs to be larger than the total jitter to achieve this.

Prior documents describing the concept of TCQF (without using that name) include [I-D.qiang-detnet-large-scale-detnet] and [I-D.dang-queuing-with-multiple-cyclic-buffers]. TCQF does not depend on other elements of [RFC8655], so it can also be used stand alone in otherwise non-deterministic IP/IPv6 or MPLS networks to achieve bounded latency and low jitter.

TCQF is likely especially beneficial when networks are architected to avoid per-hop, per-flow state even for traffic steering, which is the case for networks using SR-MPLS [RFC8402] for traffic steering of MPLS unicast traffic, SRv6 [RFC8986] for traffic steeering of IPv6 unicast traffic and/or BIER-TE [I-D.ietf-bier-te-arch] for tree engineering of MPLS multicast traffic by using the TC and/or DSCP header fields of BIER packets according to [RFC8296].

In these networks, it is specifically undesirable to require per-flow signaling to non-edge forwarders (such as P-LSR in MPLS networks) solely for DetNet QoS because such per-flow state is unnecessary for traffic steering and would only be required for the bounded latency QoS mechanism and require likely even more complex hardware and manageability support than what was previously required for per-hop steering state (such as in RSVP-TE, [RFC4875]). Note that the DetNet architecture [RFC8655] does not include full support for this DiffServ model, which is why this memo describes how to use TCQF with the DetNet architecture per-hop, per-flow processing as well as without it.

3. Using TCQF in the DetNet Architecture and MPLS forwarding plane (informative)

This section gives an overview of how the operations of TCQF relates to the DetNet architecture. We first revisit QoS with DetNet in the absence of TCQF using an MPLS network as an example.

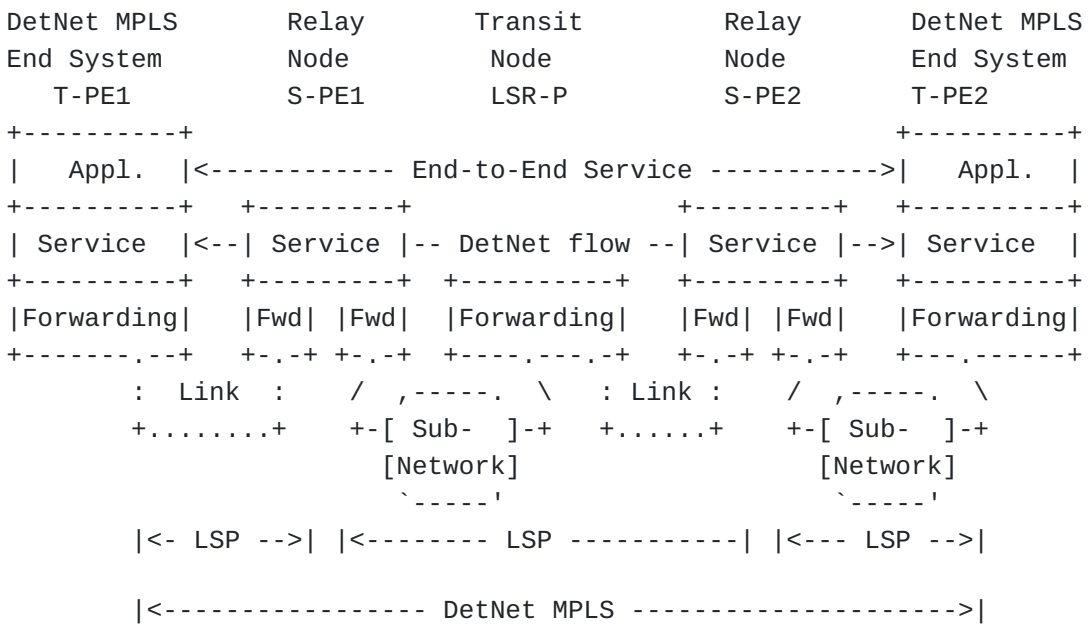


Figure 5: A DetNet MPLS Network

The above Figure 5, is copied from [RFC8964], Figure 2, and only enhanced by numbering the nodes to be able to better refer to them in the following text.

Assume a DetNet flow is sent from T-PE1 to T-PE2 across S-PE1, LSR, S-PE2. In general, bounded latency QoS processing is then required on the outgoing interface of T-PE1 towards S-PE1, and any further outgoing interface along the path. When T-PE1 and S-PE2 know that their next-hop is a service LSR, their DetNet flow label stack may simply have the DetNet flows Service Label (S-Label) as its Top of Stack (ToS) LSE, explicitly indicating one DetNet flow.

On S-PE1, the next-hop LSR is not DetNet aware, which is why S-PE1 would need to send a label stack where the S-Label is followed by a Forwarding Label (F-Label), and LSR-P would need to perform bounded latency based QoS on that F-Label.

For bounded latency QoS mechanisms relying on per-flow regulator state (aka: per-flow packet scheduling), such as in [TSN-ATS], this requires the use of a per-detnet flow F-Labels across the network from S-PE1 to S-PE2. These could for for example be assigned/managed through RSVP-TE [RFC3209] enhanced as necessary with QoS parameters matching the underlying bounded latency mechanism (such as [TSN-ATS]).

With TCQF, a sequence of LSR and DetNet service node implements TCQF with MPLS TC, ideally from T-PE1 (ingress) to T-PE2 (egress). The ingress node needs to perform per-DetNet-flow per-packet

"shaping"/"regulating" to assign each packet of a flow to a particular TCQF cycle. This is specified in Section 5.

All LSR/Service nodes after the ingress node only have to map a received TCQF tagged DetNet packet to the configured cycle on the output interface, not requiring any per-DetNet-flow QoS state. These LSR/Service nodes do therefore also not require per-flow interactions with the controller plane for the purpose of bounded latency.

Per-flow state therefore is only required on nodes that are DetNet service nodes, or when explicit, per-DetNet flow steering state is desired, instead of ingress steering through e.g.: SR-MPLS.

Operating TCQF per-flow stateless across a service node, such as S-PE1, S-PE2 in the picture is only one option. It is of course equally feasible to Have one TCQF domain from T-PE1 to S-PE2, start a new TCQF domain there, running for example up to S-PE2 and start another one to T-PE2.

A service node must act as an egress/ingress edge of a TCQF domain if it needs to perform operations that do change the timing of packets other than the type of latency that can be considered in configuration of TCQF (see Section 6.2).

For example, if T-PE1 is ingress for a TCQF domain, and T-PE2 is the egress, S-PE1 could perform the DetNet Packet Replication Function (PRF) without having to be a TQCF edge node as long as it does not introduce latencies not included in the TCQF setup and the controller plane reserves resources for the multitude of flows created by the replication taking the allocation of resources in the TCQF cycles into account.

Likewise, S-PE2 could perform the Packet Elimination Function without being a TCQF edge node as this most likely does not introduce any non-TCQF acceptable latency - and the controller plane accordingly reserves only for one flow the resources on the S-PE2->T-PE2 leg.

If on the other hand, S-PE2 was to perform the Packet Reordering Function (PRF), this could create large peaks of packets when out-of-order packets are released together. A PRF would either have to take care of shaping out those bursts for the traffic of a flow to again conform to the admitted CIR/PIR, or else the service node would have to be a TCQF egress/ingress, performing that shaping itself as an ingress function.

4. TCQF per-flow stateless forwarding (normative)

4.1. Configuration Data model and tag processing for MPLS TC tags

The following data model summarizes the configuration parameters as required for TCQF and discussed in further sections. 'tcqf' includes the parameters independent of the tagging on an interface. 'tcqf_*' describes the parameters for interfaces using MPLS TC and IP DSCP tagging.

```
# Encapsulation agnostic data
tcqf
+-- uint16 cycles
+-- uint16 cycle_time
+-- uint32 cycle_clock_offset
+-- if_config[oif] # Outgoing InterFace
    +-- uint32 cycle_clock_offset
    +-- cycle_map[iif] # Incoming InterFace
        +--uint8 oif_cycle[iif_cycle]
```

Figure 6: Encapsulation independent TCQF Configuration Data Model

4.2. Packet processing

This section explains the TCQF packet processing and through it, introduces the semantic of the objects in Figure 6

tcqf contains the router wide configuration of TCQF parameters, independent of the specific tagging mechanism on any interface. Any interface can have a different tagging method. This document uses the term router when it is irrelevant whether forwarding is for IP or MPLS packet, and the term Label Switched Router (LSR) to indicate MPLS is used, or IP router to indicate IP or IPv6 are used - independent of the specific encapsulation used for IP or MPLS to carry the cycle identification.

The model represents a single TQCF domain, which is a set of interfaces acting both as ingress (iif) and egress (oif) interfaces, capable to forward TCQF packets amongst each other. A router may have multiple TCQF domains each with a set of interfaces disjoint from those of any other TCQF domain.

tcqf.cycles is the number of cycles used across all interfaces in the TCQF domain. routers MUST support 3 and 4 cycles. The maximum number of supportable cycles depends on the encapsulation. For example, to support interfaces with MPLS TC tagging, 7 or fewer cycles MUST be used across all interfaces in the CQF domain. See Section 4.3.

The unit of tcqf.cycle_time is micro-seconds. routers MUST support configuration of cycle-times of 20,50,100,200,500,1000,2000 usec.

Cycles start at an offset of tcqf.cycle_clock_offset in units of nsec as follows. Let clock1 be a timestamp of the local reference clock for TCQF, at which cycle 1 starts, then:

tcqf.cycle_clock_offset = (clock1 mod (tcqf.cycle_time * tcqf.cycles))

The local reference clock of the router is expected to be synchronized with the neighboring LSR/router in TCQF domain. tcqf.cycle_clock_offset can be configurable by the operator, or it can be read-only. In either case will the operator be able to configure working TCQF forwarding through appropriately calculated cycle mapping.

tcqf.if_config[oif] is optional per-interface configuration of TCQF parameters. tcqf.if_config[oif].cycle_clock_offset may be different from tcqf.cycle_clock_offset, for example, when interfaces are on line cards with independently synchronized clocks, or when non-uniform ingress-to-egress propagation latency over a complex router/LSR fabric makes it beneficial to allow per-egress interface or line card configuration of cycle_clock_offset. It may be configurable or read-only.

The value of -1 for tcqf.if_config[oif].cycle_clock_offset is used to indicate that the domain wide tcqf.cycle_clock_offset is to be used for oif. This is the only permitted negative number for this parameter.

When a packet is received from iif with a cycle value of iif_cycle and the packet is routed towards oif, then the cycle value (and buffer) to use on oif is tcqf.if_config[oif].cycle_map[iif].oif_cycle[iif_cycle]. This is called the cycle mapping and is must be configurable. This cycle mapping always happens when the packet is received with a cycle tag on an interface in a TCQF domain and forwarded to another interface in the same TCQF domain.

This encapsulation independent data model only defines how to map from a received packets cycle to a sending interface cycle buffer and hence sent packet cycle. It does not specify how the cycle identifier is encoded in the received or sent packet. This is amended by the specification in the following sections.

This data model does therefore also not determine whether interfaces use IP/IPv6, MPLS or any other encapsulation. This is determined by the configuration of the DetNet domain. A mixed use of MPLS and IP/

IPv6 interfaces is possible with this data model, but at the time of writing this document not supported by DetNet.

4.3. TCQF for MPLS with TC tagging

This section describes operation of TCQF for MPLS packets using the Traffic Class (TC) field of MPLS label to carry the cycle-id. To support this encapsulation, the TCQF Data Model as defined in Figure 6 is expanded as follows.

```
# MPLS TC tagging specific data
tcqf_tc[oif]
+--uint8 tc[oif_cycle]
```

Figure 7: TCQF Configuration Data for MPLS TC

tcqf_tc[oif].tc[oif_cycle] defines how to map from the internal cycle number oif_cycle to an MPLS TC value on interface oif. tcqf_tc[oif] MUST be configured, when oif uses MPLS. This oif_cycle <=> tc mapping is not only used to map from internal cycle number to MPLS TC tag when sending packets, but also to map from MPLS TC tag to the internal cycle number when receiving packets.

In the terminology of [RFC3270], TCQF QoS as defined here, is TC-Inferred-PSC LSP (E-LSP) behavior: Packets are determined to belong to the TCQF PSC solely based on the TC of the received packet.

The internal cycle number SHOULD be assigned from the Top of Stack (ToS) MPLS label TC bits before any other label stack operations happens. On the egress side, the TC value of the ToS MPLS label SHOULD be assigned from the internal cycle number after any label stack processing.

With this order of processing, TCQF can support forwarding of packets with any label stack operations such as label swap in the case of LDP or RSVP-TE created LSP, Penultimate Hop Popping (PHP), or no label changes from SID hop-by-hop forwarding and/or SID/label pop as in the case of SR-MPLS traffic steering.

4.4. TCQF for IP/IPv6 with DSCP tagging

This section describes operation of TCQF for IP/IPv6 packets using the Differentiated Services Code Point (DSCP) field of IP/IPv6 packets to carry the cycle-id. To support this encapsulation, the TCQF Data Model as defined in Figure 6 is expanded as follows.

```
# IP/IPv6 DSCP tagging specific data
tcqf_dscp[oif]
+--uint8 dscp[oif_cycle]
```

Figure 8: TCQF Configuration Data for IP/IPv6 DSCP

tcqf_dscp[oif].dscp[oif_cycle] defines how to map from the internal cycle number oif_cycle to an IP/IPv6 DSCP value on interface oif. tcqf_dscp[oif] MUST be configured, when oif uses DSCP tagging of IP/IPv6 packets for TCQF. This oif_cycle <=> idscp mapping is not only used to map from internal cycle number to the DSCP tag when sending packets, but also to map from IP/IPv6 DSCP to the internal cycle number when receiving packets.

As how DetNet domains are currently assumed to be single administrative network operator domains, this document does not ask for standardization of the DSCP to use with TCQF. Instead, deployments wanting to use TCQF with IP/IPv6 encapsulation and DSCP tagging need to assign within their domain DSCP from the xxxx11 "EXP/LU" Codepoint space according to [RFC2474], Section 6. This allows up to 16 DSCP for intradomain use and hence up to 16 cycle identifiers.

4.5. TCQF for IPv6 with IPv6 Option tagging

This section describes operation of TCQF for IPv6 packets without having to rely on DSCP by defining a new IPv6 option for DetNet. This option is to be placed in the IPv6 HbH (Hop-by-Hop) Options or DOH (Destination Option Header) header. To support this encapsulation, the TCQF Data Model as defined in Figure 6 is expanded as follows.

```
# IPv6 TCQF Option tagging specific data
tcqf_ipv6oh[oif]
+--uint8 ipv6oh[oif_cycle]
```

Figure 9: TCQF Configuration Data for IPv6 TCQF Option Header

4.5.1. TCQF Option Format

The TCQF Option helps the receiving port to identify in which time cycle interval the packet is sent from the upstream router. It can be used to determine the output port cycle buffer to enqueue the packet.

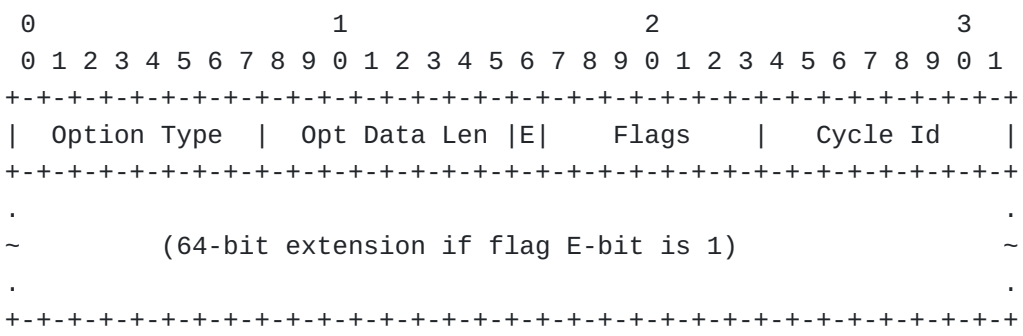


Figure 10: TCQF Option Format

TCQF-Option fields:

*Option Type: 8-bit identifier of the type of option. Value TBD by IANA. If the processing IPv6 node does not recognize the Option Type it must discard the packet and return an ICMPv6 message (the highest-order 2 bits = 10). The Option Data of this option may change en route to the packet's final destination (the third-highest-order bit=1).

*Opt Data Len: 8-bit length of the option data.

*Flags: 8-bit field to indicate what TCQF Option information follows. The leftmost bit is called E-bit. When E-bit set to 1, there is a 64-bit extension in length after Cycle Id.

*Cycle Id: 8-bit field to indicate the time cycle ID at output port of the upstream node when the packet is sent out. This is the packet header field name for the data model ipv6oh[oif_cycle] element.

*64-bit extension: This field contains values required for a possible additional options, such as timestamp. This field exists only when E-bit in Flags field is set to one. [Editor's Note: Text will be modified or added as specific uses for this field are identified]

4.5.2. TCQF Option Processing

A packet carrying the TCQF Option with Cycle Id does not change the fundamental cyclic queuing and forwarding behaviors of TCQF over the encapsulation independent forwarding behavior described above (Section 4.2).

Compared to DSCP it does not introduce a limited number of cycle-ids, and eliminates the possible operation consideration to use multiple DSCP for effectively a single per-hop forwarding behavior, which otherwise would be a novel aspect that could cause issues for example with diagnostics or other operational standards. It also

allows easier extensions with other potentially beneficial DetNet features in the same Option header.

As part of the packet processing of Section 4.2, the Cycle ID field of the option heade is rewritting from tcqf.ipv6oh[oif_cycle], in the same way as DSCP wold be rewritten from tcqf.dscp[oif_cycle].

4.5.3. Encapsulation of TCQF Option for Deterministic IP (DIP) data plane

When used in IPv6 ([RFC8200]) networks, the TCQF Option can be placed in an HbH extension header or Destination Option Header (DOH).

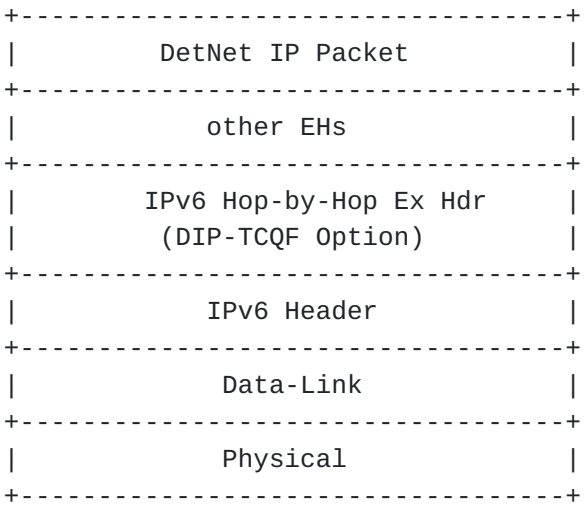


Figure 11: TCQF Option Encapsulated in HbH for Deterministic IP data plane

Figure 11 shows the encapsulation of TCQF option in HbH extension header for deterministic IP (DIP data plane. When every DetNet forwarding node along the path is provisioned to use TCQF as the queuing mechanism, this option should be placed here. If a router does not support this option, it discards the packet and returns an ICMP message.

In some deployments the path selection is indicated using IPv6 routing header (RH) by specifying a set of nodes that must be traversed by the packet along its path to the destination. When such a source routing mechanism is used, TCQF Option is placed in DOH (Destination Option Header) as shown in Figure 12 for Deterministic IP data plane. Then the TCQF Option will be processed by the specified in-path routers.

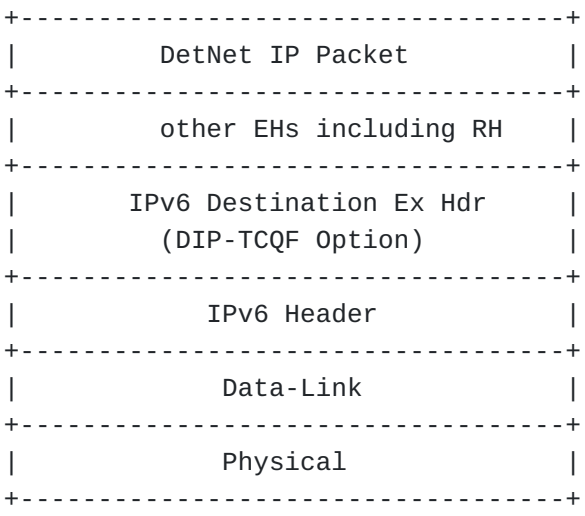


Figure 12: TCQF Option Encapsulated in DOH for Deterministic IP data plane

(TBD: Should and how TCQF Option be used in SRv6 ?)

4.6. TCQF Pseudocode (normative)

The following pseudocode restates the forwarding behavior of Section 4 in an algorithmic fashion as pseudocode. It uses the objects of the TCQF configuration data model defined in Section 4.1.

```
void receive(pak) {
  // Receive side TCQF - retrieve cycle of received packet
  // from packet internal header
  iif = pak.context.iif
  if (tcqf.if_config[iif]) { // TCQF enabled on iif
    if (tcqf_tc[iif]) {      // MPLS TCQF enabled on iif
      tc = pak.mpls_header.lse[tos].tc
      pak.context.tcqf_cycle = map_tc2cycle( tc, tcqf_tc[iif])
    } else
    if (tcqf_ipv6oh[iif]) {    // IPv6 Option Header used on iif
      cycle_id = pak.ipv6_header.tcqf_oh[cycle_id]
      pak.context.tcqf_cycle =
        map_ipv6oh2cycle( cycle_id, tcqf_ipv6oh[iif])
    } else
    if (tcqf_dscp[iif]) {      // IP DSCP TCQF used on iif
      dscp = pak.ip_header.dscp
      pak.context.tcqf_cycle = map_dscp2cycle( dscp, tcqf_dscp[iif])
    } else // ... other encaps
  }
  forward(pak);
}

// ... Forwarding including any label stack operations

void forward(pak) {
  oif = pak.context.oif = forward_process(pak)

  if(ingres_flow_enqueue(pak))
    return // ingress packets are only enqueued here.

  if(pak.context.tcqf_cycle) // non TCQF packets cycle is 0
    if(tcqf.if_config[oif]) {    // TCQF enabled on OIF
      // Map tcqf_cycle iif to oif - encap agnostic
      cycle = pak.context.tcqf_cycle
            = map_cycle(cycle,
              tcqf.if_config[oif].cycle_map[[iif])

      // MPLS TC-TCQF
      if(tcqf.tc[oif]) {
        pak.mpls_header.lse[tos].tc = map_cycle2tc(cycle, tcqf_tc[oif])
      } else
      if (tcqf_ipv6oh[oif]) { // IPv6 Option Header used on iif
        pak.ipv6_header.tcqf_oh[cycle_id] =
          map_cycle2ipv6oh(cycle, tcqf_ipv6oh[oif])
      } else
      // IP DSCP TCQF enabled on iif
      if (tcqf_dscp[oif]) {
        pak.ip_header.dscp = map_cycle2dscp(cycle, tcqf_dscp[oif])
      } // else...  other future encap/tagging options for TCQF
```

```
      tcqf_enqueue(pak, oif.cycleq[cycle,iif])  // [3]
      return
    } else {
      // Forwarding of egress TCQF packets [1]
    }
  }
  // ... non TCQF OIF forwarding [2]
}

// Started when TCQF is enabled on an interface
// dequeues packets from oif.cycleq
// independent of encapsulation
void send_tcqf(oif) {
  cycle = 1
  cc =  tcqf.cycle_time *
        tcqf.cycle_time
  o =   tcqf.cycle_clock_offset
  nextcyclestart = floor(tnow / cc) * cc + cc + o

  while(1) {
    ingress_flow_2_tcqf(oif,cycle) // [5]
    wait_until(tnow >= nextcyclestart); // wait until next cycle
    nextcyclestart += tcqf.cycle_time
    forall(iif) {
      forall(pak = tcqf_dequeue(oif.cycleq[cycle,iif]) {
        schedule to send pak on oif before nextcyclestart; // [4]
      }
    }
    cycle = (cycle + 1) mod tcqf.cycles + 1
  }
}
```

Figure 13: TCQF Pseudocode

Processing of ingress TCQF packets is performed via ingres_flow_enqueue(pak) and ingress_flow_2_tcqf(oif,cycle) as explained in Section 5.2.

Packets in a cycle buffer can be sent almost arbitrarily within the time period of the cycle. They also do not need to be sent as soon as possible, as long as all will be sent within that period. There is no need to send them in the order of their arrival except that packets from the same ingres flow that end up in the same cycle must not be reordered across any number of tcqf hops. The pseudocode describes this by using a queue oif.cycleq[cycle,iif] ([3]) for all packets from the same iif. The pseudocode describes the oterwise arbitrary scheduling of all packets within the cycle time via the statement shown in [4].

Ingress packets are passed from their ingress queues to the next cycle queue via [5].

Processing of egres TCQF packets is out-of-scope. It can performed by any non-TCQF packet forwarding mechanism such as some strict priority queuing in step [2], and packets could accordingly be marked with an according packet header traffic class indicator for such a traffic class in step [1].

5. TCQF Per-flow Ingress forwarding (normative)

Ingress flows in the context of this text are packets of flows that enter the router from a non-TCQF interface and need to be forwarded to an interface with TCQF.

In the most simple case, these packets are sent by the source and the router is the first-hop router. In another case, the routers ingress interface connects to a hop where the previous router(s) did perform a different bounded latency forwarding mechanism than TCQF.

5.1. Ingress Flows Configuration Data Model

```
# Extends above defined tcqf
tcqf
...
| Ingress Flows, see below (TBD:
+-- iflow[flowid]
    +-- uint32 csize # in bits
```

Figure 14: TCQF Ingress Configuration Data Model

The data model shown in Figure 14 expands the tcqf data model from Figure 6. For every DetNet flow for which this router is the TCQF ingress, the controller plane has to specify a maximum number of bits called csize (cycle size) that are permitted to go into each individual cycle.

Note, that iflow[flowid].csize is not specific to the sending interface because it is a property of the DetNet flow.

5.2. Ingress Flows Pseudocode

When a TCQF ingress is received, it first has to be enqueued into a per-flow queue. This is necessary because the permitted burst size for the flow may be larger than what can fit into a single cycle, or even into the number of cycles used in the network.

```
bool ingres_flow_enqueue(pak) {
  if(!pak.context.tcqf_cycle &&
      flowid = match_detnetflow(pak)) {
    police(pak) // according to RFC9016 5.5
    enqueue(pak, flowq[oif][flowid])
    return true
  }
  return false
}
```

Figure 15: TCQF Ingress Enqueue Pseudocode

ingres_flow_enqueue(pak) as shown in Figure 15 performs this enqueuing of the packet. Its position in the DetNet/TCQF forwarding code is shown in Figure 13.

police(pak): If the router is not only the TCQF ingress router, but also the first-hop router from the source, ingres_flow_enqueue(pak) will also be the place where policing of the flows packet according to the Traffic Specification of the flow would happen - to ensure that packets violating the Traffic Specification will not be forwarded, or be forwarded with lower priority (e.g.: as best effort). This policing and resulting forwarding action is not specific to TCQF and therefore out of scope for this text. See [RFC9016], section 5.5.

```
void ingress_flow_2_tcqf(oif, cycle) {
  foreach flowid in flowq[oif][*] {
    free = tcqf.iflow[flowid].csize
    q = flowq[oif][flowid]
    while(notempty(q) &&
          (l = head(q).size) <= free) {
      pak = dequeue(q)
      free -= l
      tcqf_enqueue(pak, oif.cycleq[cycle,internal])
    }
  }
}
```

Figure 16: TCQF Ingress Pseudocode

ingress_flow_2_tcqf(oif, cycle) as shown in Figure 16 transfers ingress DetNet flow packets from their per-flow queue into the queue of the cycle that will be sent next. The position of ingress_flow_2_tcqf() in the DetNet/TCQF forwarding code is shown in Figure 13.

6. Implementation, Deployment, Operations and Validation considerations (informative)

6.1. High-Speed Implementation

High-speed implementations with programmable forwarding planes of TCQF packet forwarding require Time-Gated Queues for the cycle queues, such as introduced by [IEEE802.1Qbv] and also employed in CQF [IEEE802.1Qch].

Compared to CQF, the accuracy of clock synchronization across the nodes is reduced as explained in Section 6.2 below.

High-speed forwarding for ingress packets as specified in Section 5 above would require to pass packets first into a per-flow queue and then re-queue them into a cycle queue. This is not ideal for high speed implementations. The pseudocode for ingres_flow_enqueue() and ingress_flow_2_tcqf(), like the rest of the pseudocode in this document is only meant to serve as the most compact and hopefully most easy to read specification of the desired externally observable behavior of TCQF - but not as a guidance for implementation, especially not for high-speed forwarding planes.

High-speed forward could be implemented with single-enqueueing into cycle queues as follows:

Let B[f] be the maximum amount of data that the router would need to buffer for ingress flow f at any point in time. This can be

calculated from the flows Traffic Specification. For example, when using the parameters of [RFC9016], section 5.5.

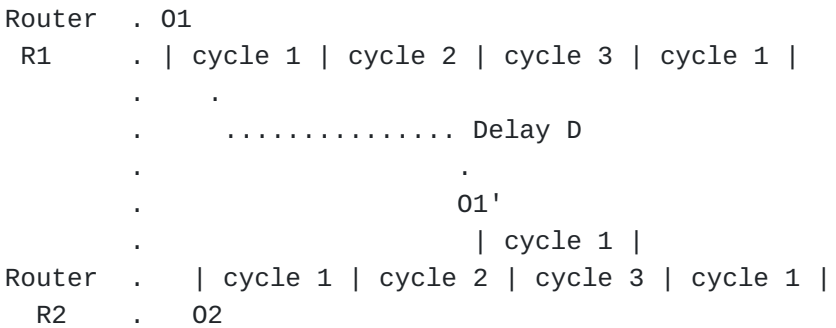
```
B[f] <= MaxPacketsPerInterval*MaxPayloadSize*8

maxcycles = max( ceil( B[f] / tcqf.iflow[f].csize) | f)
```

Maxcycles is the maximum number of cycles required so that packets from all ingress flows can be directly enqueued into maxcycles queues. The router would then not cycle across tcqf.cycles number of queues, but across maxcycles number of queues, but still cycling across tcqf.cycles number of cycle tags.

Calculation of B[f] and in result maxcycles may further be refined (lowered) by additionally known constraints such as the bitrates of the ingress interface(s) and TCQF output interface(s).

6.2. Controller plane computation of cycle mappings

The cycle mapping is computed by the controller plane by taking at minimum the link, interface serialization and node internal forwarding latencies as well as the cycle_clock_offsets into account.




```
CT  = cycle_time
C   = cycles
CC  = CT * C
O1  = cycle_clock_offset router R1, interface towards R2
O2  = cycle_clock_offset router R2, output interface of interest
O1' = O1 + D
```

Figure 17: Calculation reference

Consider in Figure 17 that Router R1 sends packets via C = 3 cycles with a cycle_clock offset of O1 towards Router R2. These packets arrive at R2 with a cycle_clock offset of O1' which includes through D all latencies incurred between releasing a packet on R1 from the cycle buffer until it can be put into a cycle buffer on R2: serialization delay on R1, link delay, non_CQF delays in R1 and R2,

especially forwarding in R2, potentially across an internal fabric to the output interface with the sending cycle buffers.

```
A = ( ceil( ( O1' - O2 ) / CT) + C + 1) mod CC
map(i) = (i - 1 + A) mod C + 1
```

Figure 18: Calculating cycle mapping

Figure 18 shows a formula to calculate the cycle mapping between R1 and R2, using the first available cycle on R2. In the example of Figure 17 with CT = 1, (O1' - O2) =~ 1.8, A will be 0, resulting in map(1) to be 1, map(2) to be 2 and map(3) to be 3.

The offset "C" for the calculation of A is included so that a negative (O1 - O2) will still lead to a positive A.

In general, D will be variable [Dmin...Dmax], for example because of differences in serialization latency between min and max size packets, variable link latency because of temperature based length variations, link-layer variability (radio links) or in-router processing variability. In addition, D also needs to account for the drift between the synchronized clocks for R1 and R2. This is called the Maximum Time Interval Error (MTIE).

Let A(d) be A where O1' is calculated with D = d. To account for the variability of latency and clock synchronization, map(i) has to be calculated with A(Dmax), and the controller plane needs to ensure that that A(Dmin)...A(Dmax) does cover at most (C - 1) cycles.

If it does cover C cycles, then C and/or CT are chosen too small, and the controller plane needs to use larger numbers for either.

This (C - 1) limitation is based on the understanding that there is only one buffer for each cycle, so a cycle cannot receive packets when it is sending packets. While this could be changed by using double buffers, this would create additional implementation complexity and not solve the limitation for all cases, because the number of cycles to cover [Dmin...Dmax] could also be (C + 1) or larger, in which case a tag of 1...C would not suffice.

6.3. Link speed and bandwidth sharing

TCQF hops along a path do not need to have the same bitrate, they just need to use the same cycle time. The controller plane has to then be able to take the TCQF capacity of each hop into account when admitting flows based on their Traffic Specification and TCQF csize.

TCQF does not require to be allocated 100% of the link bitrate. When TCQF has to share a link with other traffic classes, queuing just

has to be set up to ensure that all data of a TCQF cycle buffer can be sent within the TCQF cycle time. For example by making the TCQF cycle queues the highest priority queues and then limiting their capacity through admission control to leave time for other queues to be served as well.

6.4. Controller-plane considerations

TCQF is applicable to both centralized as well as decentralized/ distributed controller-plane models. From the perspective of the controller plane. If the controller-plane is centralized, then it is logically very simple to perform admission control for any additional flow by checking that there is sufficient bandwidth for the amount of bits required for the flow on every cycle along the intended path. Likewise, path computation can be done to determine on which non-shortest path those resources are available.

More efficient use of resources can be achieved by considering that flows with low bit rates would not need bits reserved in every cycle, but only in every N'th cyce. This requires different gates on ingres to admit packets from such flows than shown in this document and more complex admission control that attempts for example to interleave multiple flows across different set of cycles to as best as possible utilize all cycles. This is the same complexity as possible in TSN technologies. Beside the admission control and different ingres policing, such enhancements have no impact on the per-hop TCQF forwarding and can thus potentially be added incrementally.

Decentralized or distributed controller planes including on-path, per-flow signaling, such as one using the mechanisms of RSVP-TE, [RFC3209] is equally feasible with TCQF. In this case one of the potential benefits of TCQ is not leveraged, which is the complete removal of per-hop,per-flow awarenes on each router. Nevertheless, the controller-plane only introduces the need for this state maintenance into the control-plane of each router, but does not change the TCQF forwarding plane, but maintains its per-hop, per-flow non-stateful nature and resulting performance/cost benefits.

6.5. Validation

[LDN] describes an accurate simualtion based validation of TCQF and provides further details on the mathematical models.

[CENI] is a report summary of a 100Gbps link speed commercial router validation implementation of TCQF deployed and measured in a research testbed with a range of up to 2000km across China, operated by the China Environment for Network Innovations (CENI). The report also provides a reference to a more detailled version of the report.

Note that both reports are in chinese. TCQF is called DIP in these reports.

7. Security Considerations

TBD.

8. IANA Considerations

This document defines a new TCQF-Variant Option for the “Destination Options and Hop-by-Hop Options” under the “Internet Protocol Version 6 (IPv6) Parameters” registry [IPV6-PARMS] with the suggested values in Figure 19.

Hexa	act	chg	rest	Description	Reference
0xB1	10	1	10001	TCQF Option	this document

Figure 19: TCQF Option Code in Destination Options and Hop-by-Hop Options

9. Acknowledgement

Many thanks for review by David Black (DetNet techadvisor).

10. Contributors

The following co-authors have contributed to this document.

Xiaoliang Zheng Huawei Email: zhengxiaoliang@huawei.com

11. Changelog

[RFC-editor: please remove]

Initial draft name: draft-eckert-detnet-mpls-tc-tcqf

00

Initial version

01

Added new co-author.

Changed Data Model to "Configuration Data Model",

and changed syntax from YANG tree to a non-YANG tree, removed empty section targeted for YANG model. Reason: the configuration parameters that we need to specify the forwarding behavior is only a subset of what likely would be a good YANG model, and any work to define such a YANG model not necessary to specify the algorithm would be scope creep for this specification. Better done in a separate YANG document. Example additional YANG aspects for such a document are how to map parameters to configuration/operational space, what additional operational/monitoring parameter to support and how to map the YANG objects required into various pre-existing YANG trees.

Improved text in forwarding section, simplified sentences, used simplified configuration data model.

02

Refresh

03

Added ingress processing, and further implementation considerations.

New draft name: draft-eckert-detnet-tcqf

00

Added text for DSCP based tagging of IP/IPv6 packets, therefore changing the original, MPLS-only centric scope of the document, necessitating a change in name and title.

This was triggered by the observation of David Black at the IETF114 DetNet meeting that with DetNet domains being single administrative domains, it is not necessary to have standardized (cross administrative domain) DSCP for the tagging of IP/IP6 packets for TCQF. Instead it is sufficient to use EXP/LU DSCP code space and assignment of these is a local matter of a domain as is that of TC values when MPLS is used. Standardized DSCP in the other hand would have required likely work/oversight by TSVWG.

In any case, the authors feel that with this insight, there is no need to constrain single-domain definition of TCQF to only MPLS, but instead both MPLS and IP/IPv6 tagging can be easily specified in this one draft.

01

Added new co-author.

02

Attempt to resolve issues from https://github.com/toerless/detnet/issues/1.

*Review from David Black, refine queueing/scheduling of pseudocode/explanation to highlight the non-sequential requirements.

*Comment from Lou Berger re. applicability of controller-plane resulting in new section about controller-plane.

*Reference to CENI chinese validation deployment.

03

Merged draft with draft-yizhou-detnet-ipv6-options-for-cqf-variant-02.

Changed specification to be independent of encapsulation/forwarding plane and moved MPLS and IP/DSCP (from old TCQF draft) and IPv6 with extension header into separate seconds.

Human translation of CENI report, uploaded CENI report with permission from CENI onto web page accessible from outside chinese firewall.

04

Added appendix sections on comparison with CSQF and multi class TCQF

05

Refresh.

12. References

12.1. Normative References

[RFC2474] Nichols, K., Blake, S., Baker, F., and D. Black, "Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers", RFC 2474, DOI 10.17487/RFC2474, December 1998, <https://www.rfc-editor.org/rfc/rfc2474>.

[RFC3270] Le Faucheur, F., Ed., Wu, L., Davie, B., Davari, S., Vaananen, P., Krishnan, R., Cheval, P., and J. Heinanen, "Multi-Protocol Label Switching (MPLS) Support of

Differentiated Services", RFC 3270, DOI 10.17487/RFC3270, May 2002, <https://www.rfc-editor.org/rfc/rfc3270>.

[RFC8200] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", STD 86, RFC 8200, DOI 10.17487/RFC8200, July 2017, <https://www.rfc-editor.org/rfc/rfc8200>.

[RFC8655] Finn, N., Thubert, P., Varga, B., and J. Farkas, "Deterministic Networking Architecture", RFC 8655, DOI 10.17487/RFC8655, October 2019, <https://www.rfc-editor.org/rfc/rfc8655>.

[RFC8964] Varga, B., Ed., Farkas, J., Berger, L., Malis, A., Bryant, S., and J. Korhonen, "Deterministic Networking (DetNet) Data Plane: MPLS", RFC 8964, DOI 10.17487/RFC8964, January 2021, <https://www.rfc-editor.org/rfc/rfc8964>.

12.2. Informative References

[CENI] China Environment for Network Innovations (CENI), "CENI DIP Networking Test Report", 2020, <https://raw.githubusercontent.com/network2030/publications/main/CENI_DIP_Networking_Test_Report.pdf>. Translated with permission from chinese version at: https://ceni.org.cn/406.html

[I-D.chen-detnet-sr-based-bounded-latency] Chen, M., Geng, X., Li, Z., Joung, J., and J. Ryoo, "Segment Routing (SR) Based Bounded Latency", Work in Progress, Internet-Draft, draft-chen-detnet-sr-based-bounded-latency-03, 7 July 2023, <https://datatracker.ietf.org/doc/html/draft-chen-detnet-sr-based-bounded-latency-03>.

[I-D.dang-queuing-with-multiple-cyclic-buffers] Liu, B. and J. Dang, "A Queuing Mechanism with Multiple Cyclic Buffers", Work in Progress, Internet-Draft, draft-dang-queuing-with-multiple-cyclic-buffers-00, 22 February 2021, <https://datatracker.ietf.org/doc/html/draft-dang-queuing-with-multiple-cyclic-buffers-00>.

[I-D.eckert-detnet-bounded-latency-problems] Eckert, T. T. and S. Bryant, "Problems with existing DetNet bounded latency queuing mechanisms", Work in Progress, Internet-Draft, draft-eckert-detnet-bounded-latency-problems-00, 12 July

2021, <https://datatracker.ietf.org/doc/html/draft-eckert-detnet-bounded-latency-problems-00>.

[I-D.eckert-detnet-flow-interleaving]
Eckert, T. T., "Deterministic Networking (DetNet) Data Plane - Flow interleaving for scaling detnet data planes with minimal end-to-end latency and large number of flows.", Work in Progress, Internet-Draft, draft-eckert-detnet-flow-interleaving-01, 5 January 2024, <https://datatracker.ietf.org/doc/html/draft-eckert-detnet-flow-interleaving-01>.

[I-D.ietf-bier-te-arch] Eckert, T. T., Menth, M., and G. Cauchie, "Tree Engineering for Bit Index Explicit Replication (BIER-TE)", Work in Progress, Internet-Draft, draft-ietf-bier-te-arch-13, 25 April 2022, <https://datatracker.ietf.org/doc/html/draft-ietf-bier-te-arch-13>.

[I-D.qiang-detnet-large-scale-detnet]
Qiang, L., Geng, X., Liu, B., Eckert, T. T., Geng, L., and G. Li, "Large-Scale Deterministic IP Network", Work in Progress, Internet-Draft, draft-qiang-detnet-large-scale-detnet-05, 2 September 2019, <https://datatracker.ietf.org/doc/html/draft-qiang-detnet-large-scale-detnet-05>.

[IEEE802.1Q] IEEE 802.1 Working Group, "IEEE Standard for Local and Metropolitan Area Network — Bridges and Bridged Networks (IEEE Std 802.1Q)", doi 10.1109/ieeestd.2018.8403927, 2018, <https://doi.org/10.1109/ieeestd.2018.8403927>.

[IEEE802.1Qbv] IEEE Time-Sensitive Networking (TSN) Task Group., "IEEE Standard for Local and metropolitan area networks -- Bridges and Bridged Networks - Amendment 25: Enhancements for Scheduled Traffic", 2015.

[IEEE802.1Qch] IEEE Time-Sensitive Networking (TSN) Task Group., "IEEE Std 802.1Qch-2017: IEEE Standard for Local and Metropolitan Area Networks - Bridges and Bridged Networks - Amendment 29: Cyclic Queuing and Forwarding", 2017.

[IPV6-PARMS] "Internet Protocol Version 6 (IPv6) Parameters", IANA , n.d., <https://www.iana.org/assignments/ipv6-parameters/ipv6-parameters.xhtml>.

[LDN] Liu, B., Ren, S., Wang, C., Angilella, V., Medagliani, P., Martin, S., and J. Leguay, "Towards Large-Scale Deterministic IP Networks", IEEE 2021 IFIP Networking Conference (IFIP Networking), doi 10.23919/

IFIPNetworking52078.2021.9472798, 2021, <https://dl.ifip.org/db/conf/networking/networking2021/1570696888.pdf>.

[multipleCQF] Finn, N., "Multiple Cyclic Queuing and Forwarding", October 2021, <https://www.ieee802.org/1/files/public/docs2021/new-finn-multiple-CQF-0921-v02.pdf>.

[RFC3209] Awduche, D., Berger, L., Gan, D., Li, T., Srinivasan, V., and G. Swallow, "RSVP-TE: Extensions to RSVP for LSP Tunnels", RFC 3209, DOI 10.17487/RFC3209, December 2001, <https://www.rfc-editor.org/rfc/rfc3209>.

[RFC4875] Aggarwal, R., Ed., Papadimitriou, D., Ed., and S. Yasukawa, Ed., "Extensions to Resource Reservation Protocol - Traffic Engineering (RSVP-TE) for Point-to-Multipoint TE Label Switched Paths (LSPs)", RFC 4875, DOI 10.17487/RFC4875, May 2007, <https://www.rfc-editor.org/rfc/rfc4875>.

[RFC8296] Wijnands, IJ., Ed., Rosen, E., Ed., Dolganow, A., Tantsura, J., Aldrin, S., and I. Meilik, "Encapsulation for Bit Index Explicit Replication (BIER) in MPLS and Non-MPLS Networks", RFC 8296, DOI 10.17487/RFC8296, January 2018, <https://www.rfc-editor.org/rfc/rfc8296>.

[RFC8402] Filsfils, C., Ed., Previdi, S., Ed., Ginsberg, L., Decraene, B., Litkowski, S., and R. Shakir, "Segment Routing Architecture", RFC 8402, DOI 10.17487/RFC8402, July 2018, <https://www.rfc-editor.org/rfc/rfc8402>.

[RFC8938] Varga, B., Ed., Farkas, J., Berger, L., Malis, A., and S. Bryant, "Deterministic Networking (DetNet) Data Plane Framework", RFC 8938, DOI 10.17487/RFC8938, November 2020, <https://www.rfc-editor.org/rfc/rfc8938>.

[RFC8986] Filsfils, C., Ed., Camarillo, P., Ed., Leddy, J., Voyer, D., Matsushima, S., and Z. Li, "Segment Routing over IPv6 (SRv6) Network Programming", RFC 8986, DOI 10.17487/RFC8986, February 2021, <https://www.rfc-editor.org/rfc/rfc8986>.

[RFC9016] Varga, B., Farkas, J., Cummings, R., Jiang, Y., and D. Fedyk, "Flow and Service Information Model for Deterministic Networking (DetNet)", RFC 9016, DOI 10.17487/RFC9016, March 2021, <https://www.rfc-editor.org/rfc/rfc9016>.

[RFC9320] Finn, N., Le Boudec, J.-Y., Mohammadpour, E., Zhang, J., and B. Varga, "Deterministic Networking (DetNet) Bounded

Latency", RFC 9320, DOI 10.17487/RFC9320, November 2022, <https://www.rfc-editor.org/rfc/rfc9320>.

[TSN-ATS] Specht, J., "P802.1Qcr - Bridges and Bridged Networks Amendment: Asynchronous Traffic Shaping", IEEE , 9 July 2020, <https://1.ieee802.org/tsn/802-1qcr/>.

Appendix A. CSQF

[I-D.chen-detnet-sr-based-bounded-latency] (CSQF) describes a variation of the cyclic queuing mechanism in which the cycle identifier is not mapped by a mapping table in each node (as in TCQF), instead the packet header carries the sequence of cycles for every cyclic queuing hop. In the draft, this is proposed specifically for networks using Segment Routing and can therefore allocate for N cycles N SIDs, each one for a different cycle to allow indicating in a SID sequence header for each hop, which cycle to use.

The core new functionality enabled with eliminating the cycle mapping table on the routers and moving the sequence of cycles into the header is the ability to utilize in a flexible fashion more than a fixed number of cycles, independently on each hop.

Assume a minimum of N (e.g.: N = 3) cycles would be required in a particular deployment with TCQF. If CSQF is then set up with e.g.: N + 4 = 7 cycles, then it would be possible for the controller-plane to delay packets of a flow on every hop by 1,2,3 or 4 more cycles than necessary at minimum. This can lead to an easier ability to achieve higher utilization in the face of controller-plane operations that manages large number of flows in large scale DetNets, and does not allocate to every flow bandwidth in every cycle. This naturally leads to uneven utilization of cycles and the problem of managing distribution of traffic load across cycles.

[I-D.eckert-detnet-flow-interleaving] discusses this overall advanced controller-plane traffic management and how different queuing options can be used in such a setup. It also describes the necessary ingress processing to allow forwarding traffic flows only in such well engineered specific cycles.

While such advanced cycle engineering may look at first quite complex, it should simply be compared to the mechanisms that already are standard in service provider networks to manage bandwidth/ capacity by engineering per-flow paths across topologies with non-equal cost paths. In that overall complex problem space, managing distribution of traffic across cycles is but a minor extension.

Note that TCQF can of course also benefit from such advanced cycle engineering at the controller plane, albeit less flexibly than CSQF.

Given how CQSF and TCQF share all the forwarding behavior except for where the cycle Identifier is retrieved from and how it is mapped, it would also be a very useeful consideration to consider both approaches options of a single target standard. It seems unlikely though, that an implementation that can support TCQF could not support CSQF - or vice versa.

Appendix B. TCQF with multiple priorities

TSN CQF [IEEE802.1Qch] does permit to establish multiple independent cyclic queuing instances and therefore create more flexbility.

Consider likewise, that in DetNet, there are separate packet headers for a packet priority and a cycle identifier. For each priority, a separate instance of TCQF is established, and the priority decides which instance of CQF the packet gets processed by, whereas the cycle identifier determines the cycle within the TCQF instance.

Consider for example a setup with 4 priorities 1..4. The cycle time for the highest priority 1 is C. The cycle time for priority 2 is 2 * C, for priority 3 3 * C and for priority 4 4 * C. In queuing, strict priority queuing is used, packets from a priority 1 cycle queue will always be sent over those from priorities 2...4, and so on. In result, a flow can now be given one out of 4 priorities, each with an increasing per-hop latency: C (prio 1), 2C (prio 2), 3C (prio 3), 4C (prio 4). This does of course also require for admission control to not allow full utilization of the capacity of cycles in each class. In a simple static splitting of capacity across classes, each cycle of of each priority could for example be allowed to be utilized up to 25%.

This multi-priority "extension" to TCQF is in this version of the document only mentioned as an appendix, because it is not clear if this degree of flexibility is desired in a first-generation target standard for TCQF. Given how both priority and cycle identifiers are needed, this mechanism would certainly require for both MPLS and IP/IPv6 a new extension header, such as the one proposed in this document to carry the Cycle IDentifierm and then the priority could be indicated by the IP header DSCP.

Appendix C. TSN Multiple Buffer CQF

CQF with multiple buffers but without tagging has been proposed to IEEE TSN in [multipleCQF], but has not been adopted. Instead of relying on a cycle tag in a packet header as proposed in this memo, it still relies solely on the arrival time of packet, and can hence not equally resolve arrival time ambiguities as TCQF can, because it does not know the cycle from which the packet was sent, or the cycle for which it is intended.

Consider that multiple buffer CQF is like TCQF, except the cycle id is missing from the packet that is sent. Upon arrival at the receiving router, the sending cycle ID has to be determined solely by the time the packet is received (reception timestamp) because this time is an indicator of the sending timestamp and hence the sending cycle. The sum of MTIE, processing variation link propagation latency and other variations from layer 1 and layer 2 processing (forward error correction, retransmissions) is the erorr of the sending time that the receiving router can determine. As soon as this error is so large, that the receiving router can not unambiguously determine a sending cycle, the mechanism does not work anymore. The receiving router can also not simply assume for a packet to be sent by one of the possible cycles, because when this is not the actual sending cycle, then such an assumption will cause possible overruns of cycle buffers and hence failure of admission control and pckets drop or congestion. In result, multiple buffer CQF without carrying a target cycle in a packet header seems not feasible to actually solve the issue or real propagation latency variation in transmission, or the perceived variation in propagation due to jitter in clocks between adjacend nodes.

https://www.iana.org/assignments/ipv6-parameters/ipv6-parameters.xhtml

Authors' Addresses

Toerless Eckert (editor)
Futurewei Technologies USA
2220 Central Expressway
Santa Clara, CA 95050
United States of America

Email: tte@cs.fau.de

Yizhou Li (editor)
Huawei Technologies
Nanjing
China

Email: liyizhou@huawei.com

Stewart Bryant
University of Surrey ICS
United Kingdom

Email: s.bryant@surrey.ac.uk

Andrew G. Malis
Malis Consulting
United States of America

Email: agmalis@gmail.com

Jeong-dong Ryoo
ETRI
South Korea

Email: ryoo@etri.re.kr

Peng Liu
China Mobile
China

Email: liupengyjy@chinamobile.com

Guangpeng Li
Huawei Technologies
Beijing
China

Email: liguangpeng@huawei.com

Shoushou Ren
Huawei Technologies
Beijing
China

Email: renshoushou@huawei.com

Fan Yang
Huawei Technologies
Beijing
China

Email: shirley.yangfan@huawei.com