DetNet
Internet-Draft
Intended status: Informational
Expires: December 27, 2019

N. Finn
Huawei Technologies Co. Ltd
J-Y. Le Boudec
E. Mohammadpour
EPFL
J. Zhang
Huawei Technologies Co. Ltd
B. Varga
J. Farkas
Ericsson

June 25, 2019

DetNet Bounded Latency
draft-finn-detnet-bounded-latency-04

Abstract

This document presents a timing model for Deterministic Networking (DetNet), so that existing and future standards can achieve the DetNet quality of service features of bounded latency and zero congestion loss. It defines requirements for resource reservation protocols or servers. It calls out queuing mechanisms, defined in other documents, that can provide the DetNet quality of service.

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 December 27, 2019.

Table of Contents

1. Introduction

The ability for IETF Deterministic Networking (DetNet) or IEEE 802.1 Time-Sensitive Networking (TSN, [IEEE8021TSN]) to provide the DetNet services of bounded latency and zero congestion loss depends upon A)

configuring and allocating network resources for the exclusive use of DetNet/TSN flows; B) identifying, in the data plane, the resources to be utilized by any given packet, and C) the detailed behavior of those resources, especially transmission queue selection, so that latency bounds can be reliably assured. Thus, DetNet is an example of an IntServ Guaranteed Quality of Service [RFC2212]

As explained in [I-D.ietf-detnet-architecture], DetNet flows are characterized by 1) a maximum bandwidth, guaranteed either by the transmitter or by strict input metering; and 2) a requirement for a guaranteed worst-case end-to-end latency. That latency guarantee, in turn, provides the opportunity for the network to supply enough buffer space to guarantee zero congestion loss.

To be of use to the applications identified in [RFC8578], it must be possible to calculate, before the transmission of a DetNet flow commences, both the worst-case end-to-end network latency, and the amount of buffer space required at each hop to ensure against congestion loss.

This document references specific queuing mechanisms, defined in other documents, that can be used to control packet transmission at each output port and achieve the DetNet qualities of service. This document presents a timing model for sources, destinations, and the DetNet transit nodes that relay packets that is applicable to all of those referenced queuing mechanisms.

Using the model presented in this document, it should be possible for an implementor, user, or standards development organization to select a particular set of queuing mechanisms for each device in a DetNet network, and to select a resource reservation algorithm for that network, so that those elements can work together to provide the DetNet service.

This document does not specify any resource reservation protocol or server. It does not describe all of the requirements for that protocol or server. It does describe requirements for such resource reservation methods, and for queuing mechanisms that, if met, will enable them to work together.

2. Terminology and Definitions

This document uses the terms defined in [I-D.ietf-detnet-architecture].

3. DetNet bounded latency model

3.1. Flow creation

This document assumes that following paradigm is used for provisioning DetNet flows:

1. Perform any configuration required by the DetNet transit nodes in the network for the classes of service to be offered, including one or more classes of DetNet service. This configuration is done beforehand, and not tied to any particular flow.

2. Characterize the new DetNet flow, particularly in terms of required bandwidth.

3. Establish the path that the DetNet flow will take through the network from the source to the destination(s). This can be a point-to-point or a point-to-multipoint path.

4. Select one of the DetNet classes of service for the DetNet flow.

5. Compute the worst-case end-to-end latency for the DetNet flow, using one of the methods, below (Section 3.1.1, Section 3.1.2). In the process, determine whether sufficient resources are available for that flow to guarantee the required latency and to provide zero congestion loss.

6. Assuming that the resources are available, commit those resources to the flow. This may or may not require adjusting the parameters that control the filtering and/or queuing mechanisms at each hop along the flow's path.

This paradigm can be implemented using peer-to-peer protocols or using a central server. In some situations, a lack of resources can require backtracking and recursing through this list.

Issues such as un-provisioning a DetNet flow in favor of another when resources are scarce are not considered, here. Also not addressed is the question of how to choose the path to be taken by a DetNet flow.

3.1.1. Static flow latency calculation

The static problem:

> Given a network and a set of DetNet flows, compute an end-to-end latency bound (if computable) for each flow, and compute the resources, particularly buffer space, required in each DetNet transit node to achieve zero congestion loss.

In this calculation, all of the DetNet flows are known before the calculation commences. This problem is of interest to relatively static networks, or static parts of larger networks. It gives the best possible worst-case behavior. The calculations can be extended to provide global optimizations, such as altering the path of one DetNet flow in order to make resources available to another DetNet flow with tighter constraints.

The static flow calculation is not limited only to static networks; the entire calculation for all flows can be repeated each time a new DetNet flow is created or deleted. If some already-established flow would be pushed beyond its latency requirements by the new flow, then the new flow can be refused, or some other suitable action taken.

This calculation may be more difficult to perform than that of the dynamic calculation (Section 3.1.2), because the flows passing through one port on a DetNet transit node affect each others' latency. The effects can even be circular, from Flow A to B to C and back to A. On the other hand, the static calculation can often accommodate queuing methods, such as transmission selection by strict priority, that are unsuitable for the dynamic calculation.

3.1.2. Dynamic flow latency calculation

The dynamic problem:

> Given a network whose maximum capacity for DetNet flows is bounded by a set of static configuration parameters applied to the DetNet transit nodes, and given just one DetNet flow, compute the worst-case end-to-end latency that can be experienced by that flow, no matter what other DetNet flows (within the network's configured parameters) might be created or deleted in the future. Also, compute the resources, particularly buffer space, required in each DetNet transit node to achieve zero congestion loss.

This calculation is dynamic, in the sense that flows can be added or deleted at any time, with a minimum of computation effort, and without affecting the guarantees already given to other flows.

The choice of queuing methods is critical to the applicability of the dynamic calculation. Some queuing methods (e.g. CQF, Section 6.6) make it easy to configure bounds on the network's capacity, and to make independent calculations for each flow. Other queuing methods (e.g., transmission selection by strict priority), make this calculation impossible, because the worst case for one flow cannot be computed without complete knowledge of all other flows. Other queuing methods (e.g. the credit-based shaper defined in [IEEE8021Q] section 8.6.8.2) can be used for dynamic flow creation, but yield

poorer latency and buffer space guarantees than when that same queuing method is used for static flow creation (Section 3.1.1).

3.2. Relay node model

A model for the operation of a DetNet transit node is required, in order to define the latency and buffer calculations. In Figure 1 we see a breakdown of the per-hop latency experienced by a packet passing through a DetNet transit node, in terms that are suitable for computing both hop-by-hop latency and per-hop buffer requirements.

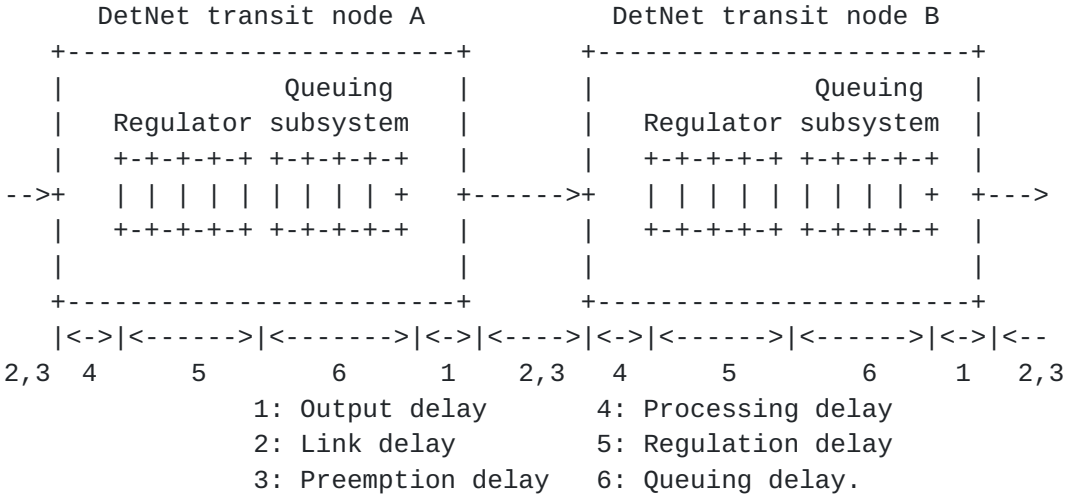

Figure 1: Timing model for DetNet or TSN

In Figure 1, we see two DetNet transit nodes (typically, bridges or routers), with a wired link between them. In this model, the only queues we deal with explicitly are attached to the output port; other queues are modeled as variations in the other delay times. (E.g., an input queue could be modeled as either a variation in the link delay [2] or the processing delay [4].) There are six delays that a packet can experience from hop to hop.

1. Output delay
   The time taken from the selection of a packet for output from a queue to the transmission of the first bit of the packet on the physical link. If the queue is directly attached to the physical port, output delay can be a constant. But, in many implementations, the queuing mechanism in a forwarding ASIC is separated from a multi-port MAC/PHY, in a second ASIC, by a multiplexed connection. This causes variations in the output delay that are hard for the forwarding node to predict or control.

2. Link delay

The time taken from the transmission of the first bit of the packet to the reception of the last bit, assuming that the transmission is not suspended by a preemption event. This delay has two components, the first-bit-out to first-bit-in delay and the first-bit-in to last-bit-in delay that varies with packet size. The former is typically measured by the Precision Time Protocol and is constant (see [I-D.ietf-detnet-architecture]). However, a virtual "link" could exhibit a variable link delay.

3. Preemption delay
   If the packet is interrupted in order to transmit another packet or packets, (e.g. [IEEE8023] clause 99 frame preemption) an arbitrary delay can result.

4. Processing delay
   This delay covers the time from the reception of the last bit of the packet to the time the packet is enqueued in the regulator (Queuing subsystem, if there is no regulation). This delay can be variable, and depends on the details of the operation of the forwarding node.

5. Regulator delay
   This is the time spent from the insertion of the last bit of a packet into a regulation queue until the time the packet is declared eligible according to its regulation constraints. We assume that this time can be calculated based on the details of regulation policy. If there is no regulation, this time is zero.

6. Queuing subsystem delay
   This is the time spent for a packet from being declared eligible until being selected for output on the next link. We assume that this time is calculable based on the details of the queuing mechanism. If there is no regulation, this time is from the insertion of the packet into a queue until it is selected for output on the next link.

Not shown in Figure 1 are the other output queues that we presume are also attached to that same output port as the queue shown, and against which this shown queue competes for transmission opportunities.

The initial and final measurement point in this analysis (that is, the definition of a "hop") is the point at which a packet is selected for output. In general, any queue selection method that is suitable for use in a DetNet network includes a detailed specification as to exactly when packets are selected for transmission. Any variations in any of the delay times 1-4 result in a need for additional buffers in the queue. If all delays 1-4 are constant, then any variation in

the time at which packets are inserted into a queue depends entirely on the timing of packet selection in the previous node. If the delays 1-4 are not constant, then additional buffers are required in the queue to absorb these variations. Thus:

- Variations in output delay (1) require buffers to absorb that variation in the next hop, so the output delay variations of the previous hop (on each input port) must be known in order to calculate the buffer space required on this hop.

- Variations in processing delay (4) require additional output buffers in the queues of that same DetNet transit node. Depending on the details of the queueing subsystem delay (6) calculations, these variations need not be visible outside the DetNet transit node.

4. Computing End-to-end Latency Bounds

4.1. Non-queuing delay bound

End-to-end latency bounds can be computed using the delay model in Section 3.2. Here it is important to be aware that for several queuing mechanisms, the worst-case end-to-end delay is less than the sum of the per-hop worst-case delays. An end-to-end latency bound for one DetNet flow can be computed as

```
end_to_end_latency_bound = non_queuing_latency + queuing_latency
```

The two terms in the above formula are computed as follows. First, at the h-th hop along the path of this DetNet flow, obtain an upper bound per-hop_non_queuing_latency[h] on the sum of delays 1,2,3,4 of Figure 1. These upper-bounds are expected to depend on the specific technology of the DetNet transit node at the h-th hop but not on the T-SPEC of this DetNet flow. Then set non_queuing_latency = the sum of per-hop_non_queuing_latency[h] over all hops h.

4.2. Queuing delay bound

Second, compute queuing_latency as an upper bound to the sum of the queuing delays along the path. The value of queuing_latency depends on the T-SPEC of this flow and possibly of other flows in the network, as well as the specifics of the queuing mechanisms deployed along the path of this flow.

For several queuing mechanisms, queuing_latency is less than the sum of upper bounds on the queuing delays (5,6) at every hop. This occurs with (1) per-flow queuing, and (2) per-class queuing with

regulators, as explained in Section 4.2.1, Section 4.2.2, and Section 6.

For other queuing mechanisms the only available value of queuing_latency is the sum of the per-hop queuing delay bounds. In such cases, the computation of per-hop queuing delay bounds must account for the fact that the T-SPEC of a DetNet flow is no longer satisfied at the ingress of a hop, since burstiness increases as one flow traverses one DetNet transit node.

4.2.1. Per-flow queuing mechanisms

With such mechanisms, each flow uses a separate queue inside every node. The service for each queue is abstracted with a guaranteed rate and a delay. For every flow the per-node delay bound as well as end-to-end delay bound can be computed from the traffic specification of this flow at its source and from the values of rates and latencies at all nodes along its path. Details of calculation for IntServ are described in Section 6.5.

4.2.2. Per-class queuing mechanisms

With such mechanisms, the flows that have the same class share the same queue. A practical example is the credit-based shaper defined in section 8.6.8.2 of [IEEE8021Q]. One key issue in this context is how to deal with the burstiness cascade: individual flows that share a resource dedicated to a class may see their burstiness increase, which may in turn cause increased burstiness to other flows downstream of this resource. Computing latency upper bounds for such cases is difficult, and in some conditions impossible [charny2000delay][bennett2002delay]. Also, when bounds are obtained, they depend on the complete configuration, and must be recomputed when one flow is added. (The dynamic calculation, Section 3.1.2.)

A solution to deal with this issue is to reshape the flows at every hop. This can be done with per-flow regulators (e.g. leaky bucket shapers), but this requires per-flow queuing and defeats the purpose of per-class queuing. An alternative is the interleaved regulator, which reshapes individual flows without per-flow queuing ([Specht2016UBS], [IEEE8021Qcr]). With an interleaved regulator, the packet at the head of the queue is regulated based on its (flow) regulation constraints; it is released at the earliest time at which this is possible without violating the constraint. One key feature of per-flow or interleaved regulator is that, it does not increase worst-case latency bounds [le_boudec_theory_2018]. Specifically, when an interleaved regulator is appended to a FIFO subsystem, it does not increase the worst-case delay of the latter.

Figure 2 shows an example of a network with 5 nodes, per-class queuing mechanism and interleaved regulators as in Figure 1. An end-to-end delay bound for flow f, traversing nodes 1 to 5, is calculated as follows:

end_to_end_latency_bound_of_flow_f = C12 + C23 + C34 + S4

In the above formula, Cij is a bound on the aggregate response time of queuing subsystem in node i and interleaved regulator of node j, and S4 is a bound on the response time of the queuing subsystem in node 4 for flow f. In fact, using the delay definitions in Section 3.2, Cij is a bound on sum of the delays 1,2,3,6 of node i and 4,5 of node j. Similarly, S4 is a bound on sum of the delays 1,2,3,6 of node 4. A practical example of queuing model and delay calculation is presented Section 6.4.

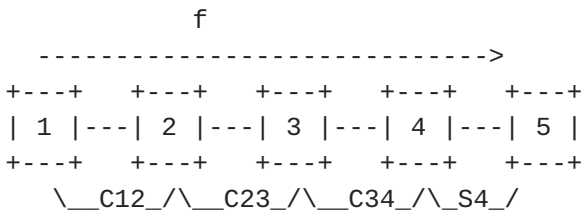

Figure 2: End-to-end latency computation example

REMARK: The end-to-end delay bound calculation provided here gives a much better upper bound in comparison with end-to-end delay bound computation by adding the delay bounds of each node in the path of a flow [TSNwithATS].

4.3. Ingress considerations

A sender can be a DetNet node which uses exactly the same queuing methods as its adjacent DetNet transit node, so that the latency and buffer calculations at the first hop are indistinguishable from those at a later hop within the DetNet domain. On the other hand, the sender may be DetNet unaware, in which case some conditioning of the flow may be necessary at the ingress DetNet transit node.

This ingress conditioning typically consists of a FIFO with an output regulator that is compatible with the queuing employed by the DetNet transit node on its output port(s). For some queuing methods, simply requires added extra buffer space in the queuing subsystem. Ingress conditioning requirements for different queuing methods are mentioned in the sections, below, describing those queuing methods.

4.4. Interspersed non-DetNet transit nodes

It is sometimes desirable to build a network that has both DetNet aware transit nodes and DetNet non-aware transit nodes, and for a DetNet flow to traverse an island of non-DetNet transit nodes, while still allowing the network to offer latency and congestion loss guarantees. This is possible under certain conditions.

In general, when passing through a non-DetNet island, the island causes delay variation in excess of what would be caused by DetNet nodes. That is, the DetNet flow is "lumpier" after traversing the non-DetNet island. DetNet guarantees for latency and buffer requirements can still be calculated and met if and only if the following are true:

1. The latency variation across the non-DetNet island must be bounded and calculable.

2. An ingress conditioning function (Section 4.3) may be required at the re-entry to the DetNet-aware domain. This will, at least, require some extra buffering to accommodate the additional delay variation, and thus further increases the worst-case latency.

The ingress conditioning is exactly the same problem as that of a sender at the edge of the DetNet domain. The requirement for bounds on the latency variation across the non-DetNet island is typically the most difficult to achieve. Without such a bound, it is obvious that DetNet cannot deliver its guarantees, so a non-DetNet island that cannot offer bounded latency variation cannot be used to carry a DetNet flow.

5. Achieving zero congestion loss

When the input rate to an output queue exceeds the output rate for a sufficient length of time, the queue must overflow. This is congestion loss, and this is what deterministic networking seeks to avoid.

5.1. A General Formula

To avoid congestion losses, an upper bound on the backlog present in the regulator and queuing subsystem of Figure 1 must be computed during resource reservation. This bound depends on the set of flows that use these queues, the details of the specific queuing mechanism and an upper bound on the processing delay (4). The queue must contain the packet in transmission plus all other packets that are waiting to be selected for output.

A conservative backlog bound, that applies to all systems, can be derived as follows.

The backlog bound is counted in data units (bytes, or words of multiple bytes) that are relevant for buffer allocation. For every class we need one buffer space for the packet in transmission, plus space for the packets that are waiting to be selected for output. Excluding transmission and preemption times, the packets are waiting in the queue since reception of the last bit, for a duration equal to the processing delay (4) plus the queuing delays (5,6).

Let

- nb_classes be the number of classes of traffic that may use this output port
- total_in_rate be the sum of the line rates of all input ports that send traffic of any class to this output port. The value of total_in_rate is in data units (e.g. bytes) per second.
- nb_input_ports be the number input ports that send traffic of any class to this output port
- max_packet_length be the maximum packet size for packets of any class that may be sent to this output port. This is counted in data units.
- max_delay45 be an upper bound, in seconds, on the sum of the processing delay (4) and the queuing delays (5,6) for a packet of any class at this output port.

Then a bound on the backlog of traffic of all classes in the queue at this output port is

```
backlog_bound = ( nb_classes + nb_input_ports ) *
max_packet_length + total_in_rate* max_delay45
```

6. Queuing techniques

6.1. Queuing data model

Sophisticated queuing mechanisms are available in Layer 3 (L3, see, e.g., [RFC7806] for an overview). In general, we assume that "Layer 3" queues, shapers, meters, etc., are precisely the "regulators" shown in Figure 1. The "queuing subsystems" in this figure are not the province solely of bridges; they are an essential part of any DetNet transit node. As illustrated by numerous implementation examples, some of the "Layer 3" mechanisms described in documents

such as [RFC7806] are often integrated, in an implementation, with the "Layer 2" mechanisms also implemented in the same node. An integrated model is needed in order to successfully predict the interactions among the different queuing mechanisms needed in a network carrying both DetNet flows and non-DetNet flows.

Figure 3 shows the general model for the flow of packets through the queues of a DetNet transit node. Packets are assigned to a class of service. The classes of service are mapped to some number of regulator queues. Only DetNet/TSN packets pass through regulators. Queues compete for the selection of packets to be passed to queues in the queuing subsystem. Packets again are selected for output from the queuing subsystem.

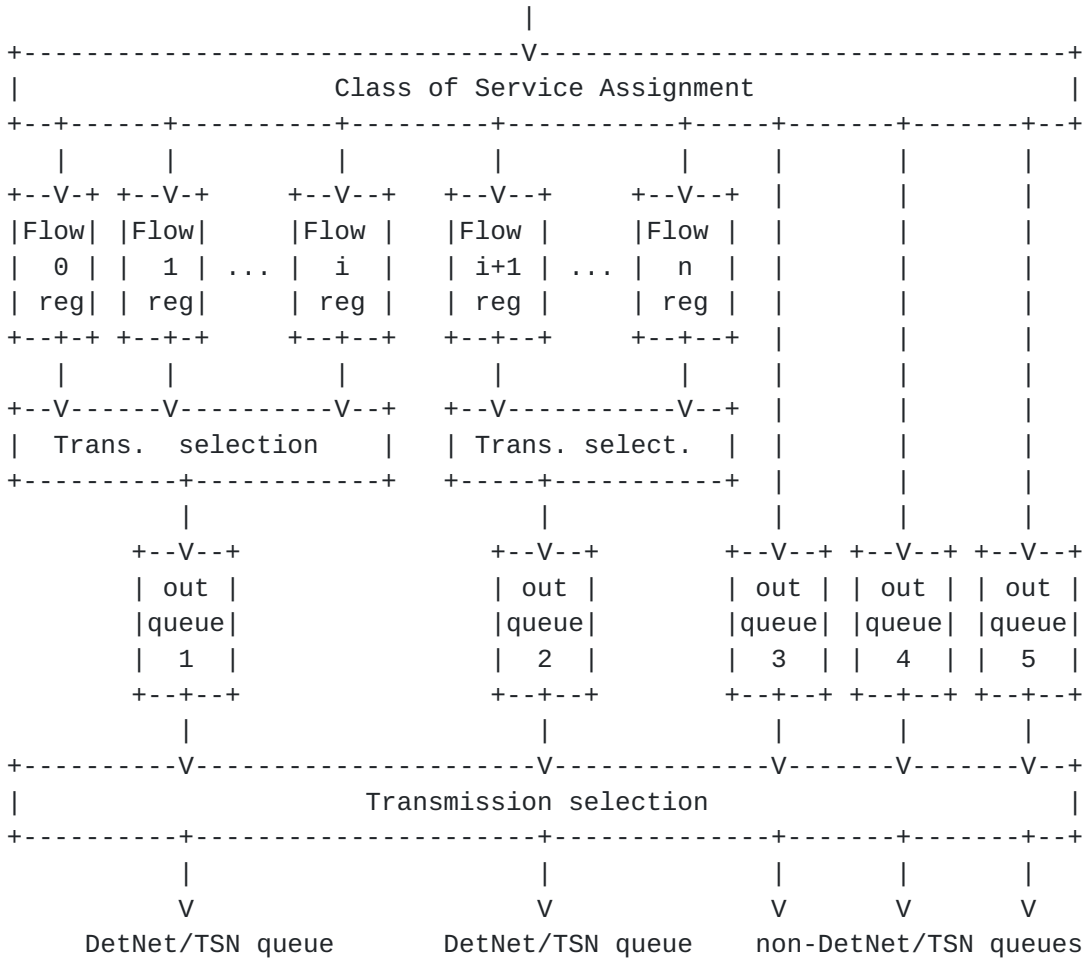

Figure 3: IEEE 802.1Q Queuing Model: Data flow

Some relevant mechanisms are hidden in this figure, and are performed in the queue boxes:

o Discarding packets because a queue is full.

- Discarding packets marked "yellow" by a metering function, in preference to discarding "green" packets.

Ideally, neither of these actions are performed on DetNet packets. Full queues for DetNet packets should occur only when a flow is misbehaving, and the DetNet QoS does not include "yellow" service for packets in excess of committed rate.

The Class of Service Assignment function can be quite complex, even in a bridge [IEEE8021Q], since the introduction of per-stream filtering and policing ([IEEE8021Q] clause 8.6.5.1). In addition to the Layer 2 priority expressed in the 802.1Q VLAN tag, a DetNet transit node can utilize any of the following information to assign a packet to a particular class of service (queue):

- Input port.

- Selector based on a rotating schedule that starts at regular, time-synchronized intervals and has nanosecond precision.

- MAC addresses, VLAN ID, IP addresses, Layer 4 port numbers, DSCP. ([I-D.ietf-detnet-ip], [I-D.ietf-detnet-mpls]) (Work items are expected to add MPC and other indicators.)

- The Class of Service Assignment function can contain metering and policing functions.

- MPLS and/or pseudowire ([RFC6658]) labels.

The "Transmission selection" function decides which queue is to transfer its oldest packet to the output port when a transmission opportunity arises.

6.2. Preemption

In [IEEE8021Q] and [IEEE8023], the transmission of a frame can be interrupted by one or more "express" frames, and then the interrupted frame can continue transmission. This frame preemption is modeled as consisting of two MAC/PHY stacks, one for packets that can be interrupted, and one for packets that can interrupt the interruptible packets. The Class of Service (queue) determines which packets are which. Only one layer of preemption is supported -- a transmitter cannot have more than one interrupted frame in progress. DetNet flows typically pass through the interrupting MAC. Best-effort queues pass through the interruptible MAC, and can thus be preempted.

6.3. Time-scheduled queuing

In [IEEE8021Q], the notion of time-scheduling queue gates is described in section 8.6.8.4. Below every output queue (the lower row of queues in Figure 3) is a gate that permits or denies the queue to present data for transmission selection. The gates are controlled by a rotating schedule that can be locked to a clock that is synchronized with other DetNet transit nodes. The DetNet class of service can be supplied by queuing mechanisms based on time, rather than the regulator model in Figure 3. Generally speacking, this time-aware scheduling can be used as a layer 2 time division multiplexing (TDM) technique.

Consider the static configuration of a deterministic network. To provide end-to-end latency guaranteed service, network nodes can support time-based behavior, which is determined by gate control list (GCL). GCL defines the gate operation, in open or closed state, with associated timing for each traffic class queue. A time slice with gate state "open" is called transmission window. The time-based traffic scheduling must be coordinated among the DetNet transit nodes along the path from sender to receiver, to control the transmission of time-sensitive traffic.

Ideally all network devices are time synchronized and static GCL configurations on all devices along the routed path are coordinated to ensure that length of transmission window fits the assigned frames, and no two time windows for DetNet traffic on the same port overlap. (DetNet flows' windows can overlap with best-effort windows, so that unused DetNet bandwidth is available to best-effort traffic.) The processing delay, link delay and output delay in transmitting are considered in GCL computation. Transmission window for a certain flow may require that a time offset on consecutive hops be selected to reduce queueing delay as much as possible. In this case, TSN/DetNet frames transmit at the assigned transmission window at every node through the routed path, with zero congestion loss and bounded end-to-end latency. Then, the worst-case end-to-end latency of the flow can be derived from GCL configuration. For a TSN or DetNet frame, denote the transmission window on last hop closes at gate_close_time_last_hop. Assuming talker supports scheduled traffic behavior, it starts the transmission at gate_open_time_on_talker. Then worst case end-to-end delay of this flow is bounded by gate_close_time_last_hop - gate_open_time_on_talker + link_delay_last_hop.

It should be noted that scheduled traffic service relies on a synchronized network and coordinated GCL configuration. Synthesis of GCL on multiple nodes in network is a scheduling problem considering all TSN/DetNet flows traversing the network, which is a non-

deterministic polynomial-time hard (NP-hard) problem. Also, at this writing, scheduled traffic service supports no more than eight traffic classes, typically using up to seven priority classes and at least one best effort class.

6.4. Credit-Based Shaper with Asynchronous Traffic Shaping

Consider a network with a set of nodes (DetNet transit nodes and hosts) along with a set of flows between hosts. Hosts are sources or destinations of flows. There are four types of flows, namely, control-data traffic (CDT), class A, class B, and best effort (BE) in decreasing order of priority. Flows of classes A and B are together referred to AVB flows. It is assumed a subset of TSN functions as described next.

It is also assumed that contention occurs only at the output port of a TSN node. Each node output port performs per-class scheduling with eight classes: one for CDT, one for class A traffic, one for class B traffic, and five for BE traffic denoted as BE0-BE4 (according to TSN standard). In addition, each node output port also performs per-flow regulation for AVB flows using an interleaved regulator (IR), called Asynchronous Traffic Shaper (ATS) in TSN. Thus, at each output port of a node, there is one interleaved regulator per-input port and per-class. The detailed picture of scheduling and regulation architecture at a node output port is given by Figure 4. The packets received at a node input port for a given class are enqueued in the respective interleaved regulator at the output port. Then, the packets from all the flows, including CDT and BE flows, are enqueued in a class based FIFO system (CBFS) [TSNwithATS].

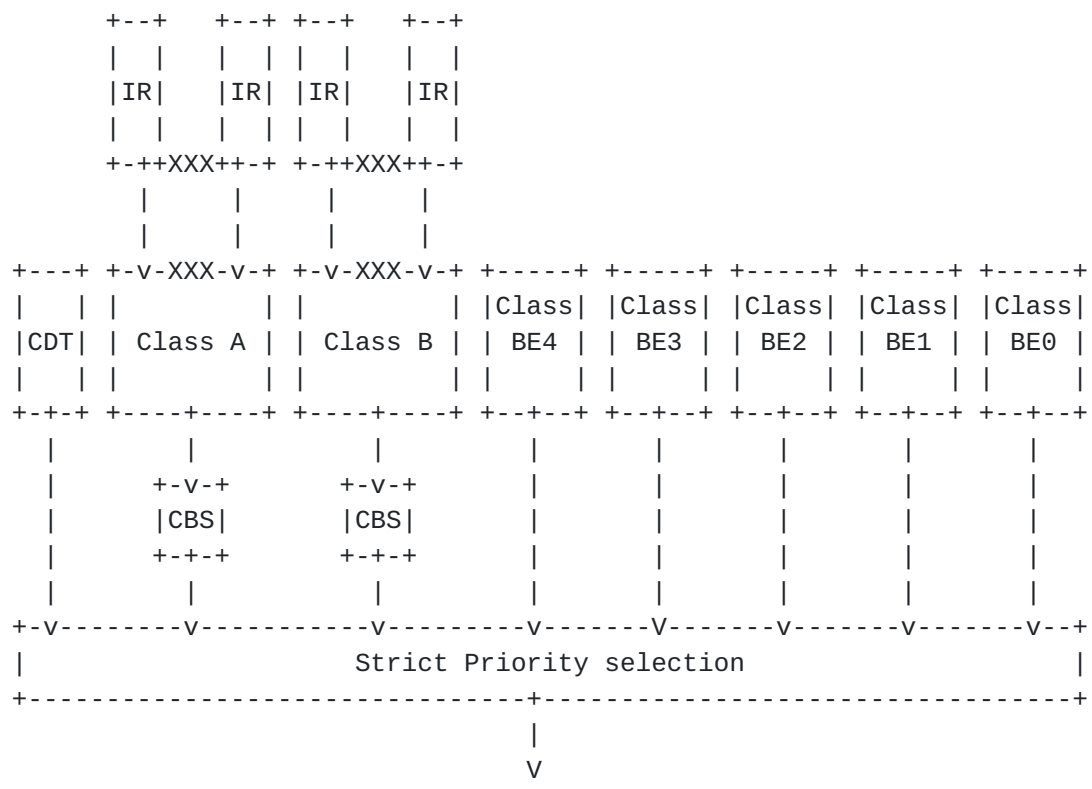

Figure 4: Architecture of a TSN node output port with interleaved regulators (IRs)

The CBFS includes two Credit-Based Shaper (CBS) subsystems, one for each class A and B. The CBS serves a packet from a class according to the available credit for that class. The credit for each class A or B increases based on the idle slope, and decreases based on the send slope, both of which are parameters of the CBS. The CDT and BE0-BE4 flows in the CBFS are served by separate FIFO subsystems. Then, packets from all flows are served by a transmission selection subsystem that serves packets from each class based on its priority. All subsystems are non-preemptive. Guarantees for AVB traffic can be provided only if CDT traffic is bounded; it is assumed that the CDT traffic has leaky bucket arrival curve with two parameters r_h as rate and b_h as bucket size, i.e., the amount of bits entering a node within a time interval t is bounded by r_h t + b_h.

Additionally, it is assumed that the AVB flows are also regulated at their source according to leaky bucket arrival curve. At the source hosts, the traffic satisfies its regulation constraint, i.e. the delay due to interleaved regulator at hosts is ignored.

At each DetNet transit node implementing an interleaved regulator, packets of multiple flows are processed in one FIFO queue; the packet at the head of the queue is regulated based on its leaky bucket

parameters; it is released at the earliest time at which this is possible without violating the constraint. The regulation parameters for a flow (leaky bucket rate and bucket size) are the same at its source and at all DetNet transit nodes along its path. A delay bound of CBFS for an AVB flow f of class A or B can be computed if the following condition holds:

sum of leaky bucket rates of all flows of this class at this node <= R, where R is given below for every class.

If the condition holds, the delay bound is:

$$d_f = T + (b_t - L_{min_f})/R - L_{min_f}/c$$

where L_{min_f} is the minimum packet length of flow f; c is the output link transmission rate; b_t is the sum of the b term (bucket size) for all the flows having the same class as flow f at this node. Parameters R and T are calculated as follows for class A and class B, separately:

If f is of class A:

$$R = I_A (c - r_h)/ c$$

$$T = L_{nA} + b_h + r_h L_n/c)/(c - r_h)$$

where L_{nA} is the maximum packet length of class B and BE packets; L_n is the maximum packet length of classes A,B, and BE.

If f is of class B:

$$R = I_B (c - r_h)/ c$$

$$T = (L_{BE} + L_A + L_{nA} I_A/(c_h - I_A) + b_h + r_h L_n/c)/(c - r_h)$$

where L_A is the maximum packet length of class A; L_{BE} is the maximum packet length of class BE.

Then, an end-to-end delay bound is calculated by the formula Section 4.2.2, where for Cij:

$$Cij = \max(d_f')$$

where f' is any flow that shares the same CBFS class with flow f at node i and the same interleaved regulator as flow f at node j.

More information of delay analysis in such a DetNet transit node is described in [TSNwithATS].

6.4.1. Flow Admission

The delay calculation requires some information about each node. For each node, it is required to know the idle slope of CBS for each class A and B (I_A and I_B), as well as the transmission rate of the output link (c). Besides, it is necessary to have the information on each class, i.e. maximum packet length of classes A, B, and BE. Moreover, the leaky bucket parameters of CDT (r_h,b_h) should be known. To admit a flow/flows, their delay requirements should be guaranteed not to be violated. As described in Section 3.1, the two problems static and dynamic are addressed separately. In either of the problems, the rate and delay should be guaranteed. Thus,

The static admission control:

The leaky bucket parameters of all flows are known, therefore, for each flow a delay bound can be calculated. The computed delay bound for every flow should not be more than its delay requirement. Moreover, the sum of the rate of each flow (r_f) should not be more than the rate allocated to each class (R). If these two conditions hold, the configuration is declared admissible.

The dynamic admission control:

For dynamic admission control, we allocate to every node and class A or B, static value for rate (R) and maximum burstiness (b_t). In addition, for every node and every class A and B, two counters are maintained:

R_acc is equal to the sum of the leaky-bucket rates of all flows of this class already admitted at this node; At all times, we must have:

$$R_{acc} \leq R, \quad \text{(Eq. 1)}$$

b_acc is equal to the sum of the bucket sizes of all flows of this class already admitted at this node; At all times, we must have:

$$b_{acc} \leq b_t. \quad \text{(Eq. 2)}$$

A new flow is admitted at this node, if Eqs. (1) and (2) continue to be satisfied after adding its leaky bucket rate

and bucket size to R_acc and b_acc. A flow is admitted in the network, if it is admitted at all nodes along its path. When this happens, all variables R_acc and b_acc along its path must be incremented to reflect the addition of the flow. Similarly, when a flow leaves the network, all variables R_acc and b_acc along its path must be decremented to reflect the removal of the flow.

The choice of the static values of R and b_t at all nodes and classes must be done in a prior configuration phase; R controls the bandwidth allocated to this class at this node, b_t affects the delay bound and the buffer requirement. R must satisfy the constraints given in Annex L.1 of [IEEE8021Q].

6.5. IntServ

Integrated service (IntServ) is an architecture that specifies the elements to guarantee quality of service (QoS) on networks. To satisfied guaranteed service, a flow must conform to a traffic specification (T-spec), and reservation is made along a path, only if routers are able to guarantee the required bandwidth and buffer.

Consider the traffic model which conforms to token bucket regulator (r, b), with

- Token bucket depth (b).

- Token bucket rate (r).

The traffic specification can be described as an arrival curve:

$$alpha(t) = b + rt$$

This token bucket regulator requires that, during any time window t, the number of bit for the flow is limited by alpha(t) = b + rt.

If resource reservation on a path is applied, IntServ model of a router can be described as a rate-latency service curve beta(t).

$$beta(t) = max(0, R(t-T))$$

It describes that bits might have to wait up to T before being served with a rate greater or equal to R.

It should be noted that, the guaranteed service rate R is a share of link's bandwidth. The choice of R is related to the specification of flows which will transmit on this node. For example, in strict priority policy, considering a flow with priority j, its share of

bandwidth may be R=c-sum(r_i), i<j, where c is the link bandwidth, r_i is the token bucket rate for the flows with priority higher than j. The choice of T is also related to the specification of all the flows traversing this node. For example, in a generalized processor sharing (GPS) node, T = L / R + L_max/c, where L is the maximum packet size for the flow, L_max is the maximum packet size in the node across all flows. Other choice of R and T are also supported, according to the specific scheduling of the node and flows traversing this node.

As mentioned previously in this section, delay bound and backlog bound can be easily obtained by comparing arrival curve and service curve. Backlog bound, or buffer bound, is the maximum vertical derivation between curves alpha(t) and beta(t), which is v=b+rT. Delay bound is the maximum horizontal derivation between curves alpha(t) and beta(t), which is h = T+b/R. Graphical illustration of the IntServ model is shown in Figure 5.

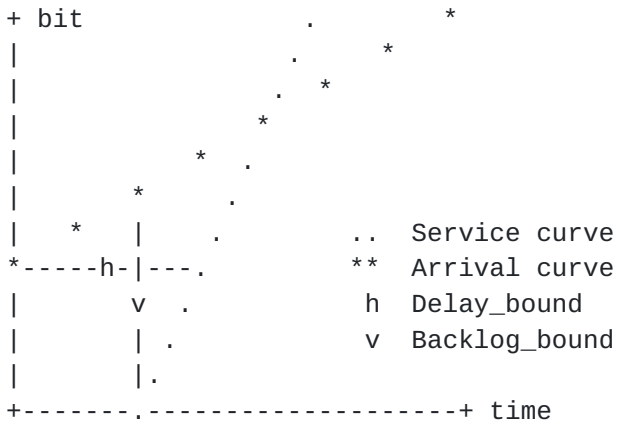

Figure 5: Computation of backlog bound and delay bound. Note that arrival and service curves are not necessary to be linear.

The output bound, or the next-hop arrival curve, is alpha_out(t) = b + rT + rt, where burstiness of the flow is increased by rT, compared with the arrival curve.

We can calculate the end-to-end delay bound for a path including N nodes, among which the i-th node offers service curve beta_i(t),

beta_i(t) = max(0, R_i(t-T_i)), i=1,...,N

By concatenating these IntServ nodes, an end-to-end service curve can be computed as

beta_e2e (t) = max(0, R_e2e(t-T_e2e))

where

$$R_{e2e} = \min(R_1, \ldots, R_N)$$

$$T_{e2e} = T_1 + \ldots + T_N$$

Similarly, delay bound, backlog bound and output bound can be computed by using the original arrival curve alpha(t) and concatenated service curve beta_e2e(t).

6.6. Cyclic Queuing and Forwarding

Annex T of [IEEE8021Q] describes Cyclic Queuing and Forwarding (CQF), which provides bounded latency and zero congestion loss using the time-scheduled gates of [IEEE8021Q] section 8.6.8.4. For a given DetNet class of service, a set of two or three buffers is provided at the output queue layer of Figure 3. A cycle time Tc is configured for each class c, and all of the buffer sets in a class swap buffers simultaneously throughout the DetNet domain at that cycle rate, all in phase.

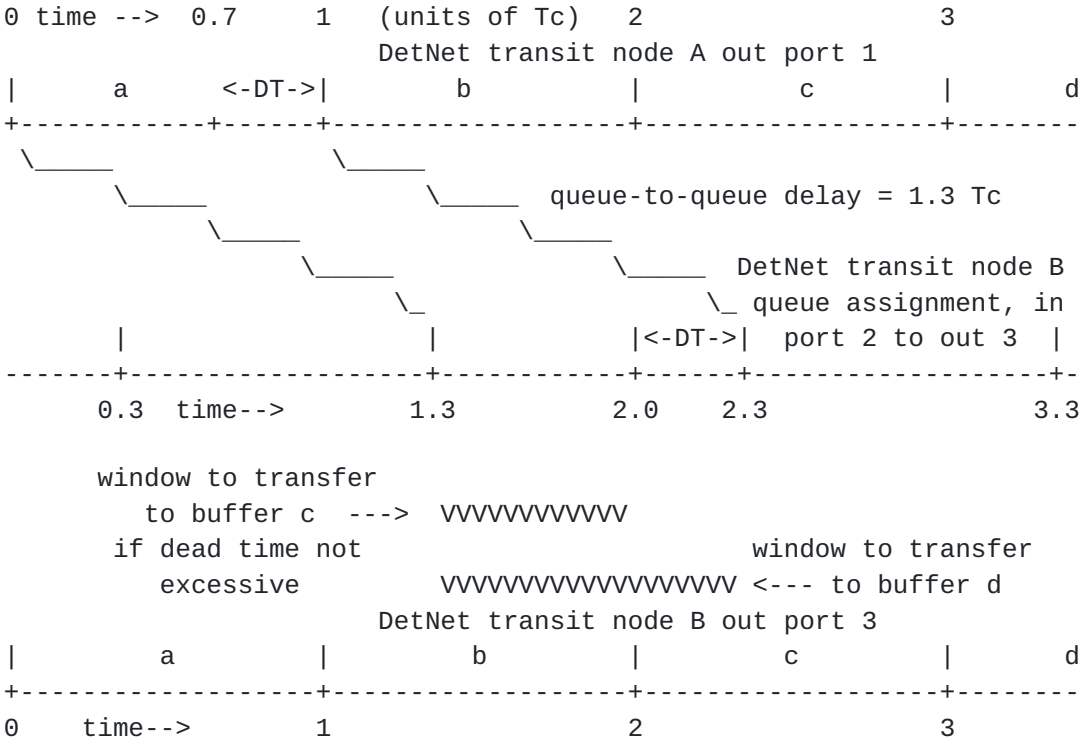

Figure 6: CQF timing diagram

Figure 6 shows two DetNet transit nodes A and B, including three timelines for:

1. The output queues on port 1 in node A.

2. The input gate function ([IEEE8021Q], 8.6.5.1) that assigns packets received on port 1 of transit node B to output queues on port 2 of transit node B.

3. The output queues on port 2 of node B.

In this figure, the output ports on the two nodes are synchronized, and a new buffer starts transmitting at each tick, shown as 0, 1, 2, ... The output times shown for timelines 1 and 3 are the times at which packets are selected for output, which is the start point of the output time (1) of Figure 1. The queue assignments times on timeline 3 take place at the beginning of the queuing delay (6) of Figure 1. Time-based CQF, as described here, does not require any regulator queues. In the shown in the figure, the total time for delays 1 through 6 of Figure 1 is 1.3Tc. Of course, any value is possible.

6.6.1. CQF timing sequence

In general, as shown in Figure 6, the windows for buffer assignment do not align perfectly with the windows for buffer transmission. The input gates (the center timeline in Figure 6) must switch from using one buffer to using another buffer in sync with the (delayed) received data, at times offset by the dead time from the output buffer switching (the bottom timeline in Figure 6).

If the dead time DT in Figure 6 is not excessive, then it is feasible to subtract the dead time from the cycle time Tc, and use the remainder as the input window. In the example in Figure 6, packets from node A buffer a can be transferred from the input port to node B's buffer "c" during the window shown by the upper row "VVVV...". Input must cease by time = 2.0, because that is when transit node B starts transmitting the contents of buffer c. In this case, only two output buffers are in use, one filling and one outputting.

If the dead time is too large (e.g., if the delays placed the middle timeline's switching points at n+0.9, instead of n+0.3), three buffers are used by node B. This case is shown by the lower row "VVVV..." in Figure 6. In this case, node B places the data received from node A buffer a into node B buffer d between the times 1.3 and 2.3 in Figure 6. Buffer b starts outputting at time = 2.0, while buffer d is filling. Thus, three buffers are in use, one filling, one waiting, and one emptying.

6.6.2. CQF latency calculation

The per-hop latency is trivially determined by the wire delay and the queuing delay. Since the wire delay is either absorbed into the queueing delay (dead time is small and two buffers are used) or padded out to a whole cycle time Tc (three buffers are used) the per-hop latency is always an integral number of cycle times Tc, with a latency variation at the output of the final hop of Tc.

Ingress conditioning (Section 4.3) may be required if the source of a DetNet flow does not, itself, employ CQF.

Note that there are no per-flow parameters in the CQF technique. Therefore, there is no requirement for per-hop configuration when a new DetNet flow is added to a network, except perhaps for ingress checks to see that the transmitter does not exceed the contracted bandwidth.

7. References

7.1. Normative References

[I-D.ietf-detnet-architecture]
Finn, N., Thubert, P., Varga, B., and J. Farkas, "Deterministic Networking Architecture", draft-ietf-detnet-architecture-08 (work in progress), September 2018.

[I-D.ietf-detnet-ip]
Varga, B., Farkas, J., Berger, L., Fedyk, D., Malis, A., Bryant, S., and J. Korhonen, "DetNet Data Plane: IP", draft-ietf-detnet-ip-00 (work in progress), May 2019.

[I-D.ietf-detnet-mpls]
Varga, B., Farkas, J., Berger, L., Fedyk, D., Malis, A., Bryant, S., and J. Korhonen, "DetNet Data Plane: MPLS", draft-ietf-detnet-mpls-00 (work in progress), May 2019.

[RFC2212] Shenker, S., Partridge, C., and R. Guerin, "Specification of Guaranteed Quality of Service", RFC 2212, DOI 10.17487/RFC2212, September 1997, <https://www.rfc-editor.org/info/rfc2212>.

[RFC6658] Bryant, S., Ed., Martini, L., Swallow, G., and A. Malis, "Packet Pseudowire Encapsulation over an MPLS PSN", RFC 6658, DOI 10.17487/RFC6658, July 2012, <https://www.rfc-editor.org/info/rfc6658>.

[RFC7806] Baker, F. and R. Pan, "On Queuing, Marking, and Dropping", RFC 7806, DOI 10.17487/RFC7806, April 2016, <https://www.rfc-editor.org/info/rfc7806>.

[RFC8578] Grossman, E., Ed., "Deterministic Networking Use Cases", RFC 8578, DOI 10.17487/RFC8578, May 2019, <https://www.rfc-editor.org/info/rfc8578>.

7.2. Informative References

[bennett2002delay]
J.C.R. Bennett, K. Benson, A. Charny, W.F. Courtney, and J.-Y. Le Boudec, "Delay Jitter Bounds and Packet Scale Rate Guarantee for Expedited Forwarding", <https://dl.acm.org/citation.cfm?id=581870>.

[charny2000delay]
A. Charny and J.-Y. Le Boudec, "Delay Bounds in a Network with Aggregate Scheduling", <https://link.springer.com/chapter/10.1007/3-540-39939-9_1>.

[IEEE8021Q]
IEEE 802.1, "IEEE Std 802.1Q-2018: IEEE Standard for Local and metropolitan area networks - Bridges and Bridged Networks", 2018, <http://ieeexplore.ieee.org/document/8403927>.

[IEEE8021Qcr]
IEEE 802.1, "IEEE P802.1Qcr: IEEE Draft Standard for Local and metropolitan area networks - Bridges and Bridged Networks - Amendment: Asynchronous Traffic Shaping", 2017, <http://www.ieee802.org/1/files/private/cr-drafts/>.

[IEEE8021TSN]
IEEE 802.1, "IEEE 802.1 Time-Sensitive Networking (TSN) Task Group", <http://www.ieee802.org/1/>.

[IEEE8023]
IEEE 802.3, "IEEE Std 802.3-2018: IEEE Standard for Ethernet", 2018, <http://ieeexplore.ieee.org/document/8457469>.

[le_boudec_theory_2018]
J.-Y. Le Boudec, "A Theory of Traffic Regulators for Deterministic Networks with Application to Interleaved Regulators", <http://arxiv.org/abs/1801.08477/>.

[NetCalBook]
Le Boudec, Jean-Yves, and Patrick Thiran, "Network calculus: a theory of deterministic queuing systems for the internet", 2001, <https://arxiv.org/abs/1804.10608/>.

[Specht2016UBS]
J. Specht and S. Samii, "Urgency-Based Scheduler for Time-Sensitive Switched Ethernet Networks", <https://ieeexplore.ieee.org/abstract/document/7557870>.

[TSNwithATS]
E. Mohammadpour, E. Stai, M. Mohiuddin, and J.-Y. Le Boudec, "End-to-end Latency and Backlog Bounds in Time-Sensitive Networking with Credit Based Shapers and Asynchronous Traffic Shaping", <https://arxiv.org/abs/1804.10608/>.

Authors' Addresses

Norman Finn
Huawei Technologies Co. Ltd
3101 Rio Way
Spring Valley, California 91977
US

Phone: +1 925 980 6430
Email: nfinn@nfinnconsulting.com

Jean-Yves Le Boudec
EPFL
IC Station 14
Lausanne EPFL 1015
Switzerland

Email: jean-yves.leboudec@epfl.ch

Ehsan Mohammadpour
EPFL
IC Station 14
Lausanne EPFL 1015
Switzerland

Email: ehsan.mohammadpour@epfl.ch

Jiayi Zhang
Huawei Technologies Co. Ltd
Q22, No.156 Beiqing Road
Beijing 100095
China

Email: zhangjiayi11@huawei.com

Balazs Varga
Ericsson
Konyves Kalman krt. 11/B
Budapest 1097
Hungary

Email: balazs.a.varga@ericsson.com

Janos Farkas
Ericsson
Konyves Kalman krt. 11/B
Budapest 1097
Hungary

Email: janos.farkas@ericsson.com