Journal of Network and Systems Management (2023) 31:34
https://doi.org/10.1007/s10922-022-09718-9

High Precision Latency Forwarding for Wide Area Networks Through Intelligent In-Packet Header Processing (gLBF)

Toerless Eckert[1] · **Alexander Clemm**[1] · **Stewart Bryant**[2]

Received: 28 February 2022 / Revised: 20 December 2022 / Accepted: 26 December 2022

Abstract
This paper presents guaranteed Latency Based Forwarding (gLBF), a solution that allows for the delivery of packets with end-to-end latency guarantees and that provides per-hop bounded latency with zero jitter. gLBF combines the benefits of earlier algorithms, Urgency-Based Scheduling and Cyclic Queuing and Forwarding, while avoiding their respective downsides. Specifically, gLBF does not need to maintain per-flow state at forwarding devices and does not require strict clock synchronization across the network. As a result, gLBF results in network deployments of significantly reduced complexity and lower cost than previous solutions.

Keywords Deterministic latency · Synchronous guarantees · LBF · TSN · UBS · SLOs · NewIP

1 Introduction

A growing number of applications require communication services that are able to forward traffic with guaranteed or deterministic latency bounds. Such services have been evolving for decades, mostly in limited physical domain networks such as inside vehicles or industrial machinery on factory floors. For example, such

✉ Toerless Eckert
toerless.eckert@futurewei.com

Alexander Clemm
alex@futurewei.com

Stewart Bryant
sb@stewartbryant.com

[1] Futurewei Technologies, Inc., 2220 Central Expressway, Santa Clara, CA 95050, USA

[2] Institute for Communication Systems, University of Surrey, 7XH, Alexander Fleming Rd, Guildford, UK

Published online: 04 February 2023

industrial machinery may involve remote controllers that steer precision assembly robotics requiring very precise timing. It may even involve collaborating robots that require very precise synchronization to be coordinated [1].

Solutions for deterministic bounded maximum latency guarantees in packet forwarding networks were defined in the latter part of the 1990s via the IETF "Guaranteed Services" (GS) standard [2], but never adopted in significant deployments. Instead, deterministic latency services first saw wider deployments in Ethernet networks via vendor proprietary solutions that where replaced over time with several IEEE "Time Sensitive Networking" (TSN) standards including "Cyclic Queuing and Forwarding" (CQF) [3] and later "Asynchronous Traffic Shaping" based on the "Urgency Based Scheduling" (UBS) algorithm [1]. Both are precursors to the work presented here.

With TSN enjoying increasing adoption at Layer 2, there is interest in revisiting deterministic service also in the context of IP and for use across wide-area networks beyond the typically more limited physical scope of TSN networks. This resulted in the formation of the DetNet working group in the IETF. Many core use cases build around high-speed wide-area IP networks [4], which has led to a re-investigation of the algorithmic requirements to best support these services in such networks. However, while DetNet lays out overall architecture and framework for corresponding network technology, actual solutions have not yet been defined.

This paper presents a novel algorithm called "guaranteed Latency Based Forwarding" (gLBF) for deterministic latency that can be delivered at networks with large scale (tens of thousands of flows), high speeds (link speeds $\geq$ 100 Gbps), and across wide areas (radius $\geq$ 10 km), in conjunction with IP and Multi-Protocol Label Switching (MPLS) [5] networks. Its benefits could also be useful for other networks. gLBF was first presented at [6]. This paper builds on and extends that earlier paper, discussing additional aspects such gLBF's ramifications on networking device hardware, clock synchronization, as well as gLBF support by network protocols. Additional validation results are also presented.

gLBF eliminates important problems of prior algorithms: the need to maintain per-hop, per-flow state and the need for network-wide clock synchronization, both of which contribute substantially to the cost of hardware and to the complexity of deployments. gLBF provides tightly bounded jitter, which the authors consider crucial for many applications, and it provides the same simple latency calculus as UBS, therefore providing the same latencies as UBS, as well as its support of different guaranteed latencies for different flows across the same paths and compatibility with UBS Path Computation Engines (PCE) and resource reservation systems.

gLBF requires more advanced queuing mechanisms than UBS requires, such as "PIFO" (Push-In-First-Out") queues. As a result, it may take more time for implementations of in high-speed, low-cost forwarding hardware to become available. Therefore, a mechanism is also described to support at least a subset of gLBF for which queuing mechanisms equivalent to those required for UBS are sufficient.

The rest of the paper is organized as follows: Sect. 2 discusses the current state of the art: Sect. 2.1 introduces a model for the network architecture in which gLBF

and prior bounded latency mechanisms operate and Sect. 2.2 introduces the relevant prior art latency control mechanisms. Section 3 contributes an analysis of their problems for the purpose of bounded latency, Sect. 4 describes the main contribution of this paper, gLBF and its different implementation options, Sect. 5 describes the authors experimental validation of gLBF. Section 6 discusses future work of interest and Sect. 7 concludes the paper.

2 Preliminaries

2.1 Common System Model

Notwithstanding other options to operationalize the control and management of latency for traffic in networks, we consider Fig. 1 as the common reference system model. This central controller approach is common today for control and management of managed traffic both in wide-area networks such as converged IP/MPLS backbone and aggregation networks, as well as in campus or factory-floor networks using IEEE/TSN solutions. In IP/MPLS networks, this model often evolved from the use of a Path Computation Engine (PCE) [7] to optimize path computation. See [8], sect, 4. for motivation why centralized methods have benefits in this type of problems.

The traffic of interest is controlled via an Admission Controller (AC) that tracks all flows and their parameters that are guaranteed to be delivered across the network with deterministic latency within the constraints of the specified traffic engineering parameters (i.e., rate and burst size). For the purpose of managing latency, the PCE is optional and can support the AC function by calculating the optimum path through the network for each individual traffic flow.

Traffic enters the network via the network ingress or first-hop router, also called the Provider Edge (PE) router. The PE is responsible for ensuring that traffic does not

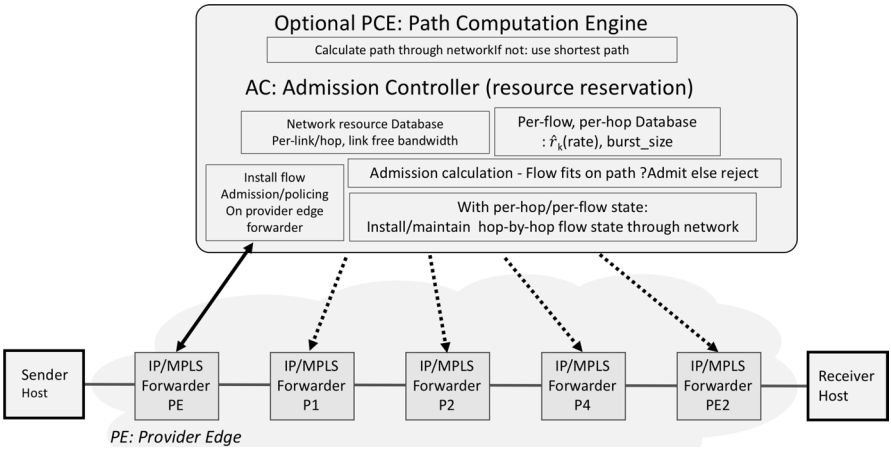

Fig. 1 Common reference system model

exceed the bounds the AC determines that it should have. This means that the ingress PE will have per-flow policing and adjust traffic flows entering the network as needed. This requires the maintaining of ingress state to keep track of traffic rate and burst sizes. Similarly, the PCE may instruct the PE to cause the traffic flow to be routed across a particular path. Traffic is then routed across zero or more Provider (Core) Routers (P routers) towards an egress PE. In "stateless" solutions, P routers will not have per-flow state. Per-flow state may exist purely to steer traffic and/or to perform per-flow shaping in support of bounded latency. The latter type of state has in the past been considered to be most expensive in high-speed networks.

The evolution of networks into Edge-PE routers and Core-P routers occurred in the early 2000'th, when more advanced network services such as Virtual Private Network (VPN) services were introduced into networks, see for example [9] (Sect. 4.3). Reducing service functionality on P routers allows for them to be faster at lower cost than PE routers. Limiting the provisioning of service specific functionality on P routers reduces the operational complexity of the network by only having to configure a service on one instead of N routers.

Note that the P/PE distinction can also occur at interface or linecard level. In networks where every router is a PE-router because it has customer facing interfaces, such as in networks where all PE-routers are connected in a ring, there can still be faster, less-expensive core (ring) facing interfaces/linecards. When traffic passes through multiple hops in the ring, it still logically passes through P-router functionality as it only touches edge functionality on the first and the last hop of the network.

2.2 Precursors and Related Technologies

This section explains and reviews technologies to the extent that they contributed to work presented in this paper. A broader review and comparison of deterministic latency options can be found in [10]. A mathematical overview of the algorithms can be found in [11].

2.2.1 Urgency Based Scheduling

Urgency Based Scheduling [UBS] is the deterministic latency algorithm adopted for IEEE Time Sensitive Networking (TSN) Asynchronous Traffic Shaping (TSN-ATS) [12]. UBS relies on per-hop, per-flow processing. It is an improved mechanism compared to "Guaranteed Services" (GS) [2]. UBS relies on so-called interleaved-regulators compared to per-flow shaping and scheduling in GS, reducing the required queuing hardware cost/complexity.

Because gLBF uses the same latency calculus as UBS, and its explanation requires understanding it, we describe the UBS calculus here.

UBS traffic flows (i) are characterized by a burstiness $b(i)$ [e.g. bits] and a leak rate $r(i)$ [e.g. bits/sec]. These parameters define a constraint for the cumulative amount of data [e.g. bits] $w(i, d)$ over any period d called the leaky bucket constraint:

$$w(i,d) \leq b(i) + d * r(i) \tag{1}$$

Both N and the priority of a flow can be configured differently on every hop of the path. To support different guaranteed latencies for different flows across the same interface, UBS uses a small number of strict priority queues $p = 1 \dots N$ on an outgoing interface, for example $N = 8$. The buffer size required for each queue p is $\sum_{i \in flows_p} b_i$, where $flows_p$ is the set of flows that should use priority queue p. The guaranteed maximum $latency(p)$ of a packet of $priority(p)$ through a forwarder hop is roughly (missing some fixed offset):

$$latency(p) = \sum_{j \in flows(q), q \leq p} b(j)/r_i \tag{2}$$

r_i is the serving rate of the interface, e.g. [bps]. This allows the provision of finer grained end-to-end differences in latency across different flows than N, for example by assigning a flow to p and $p + 1$ on different hops of its path.

Admission control in UBS needs to also ensure that the total amount of traffic does not exceed the link serving rate:

$$r_i \geq \sum_{j \in flows_i} flowrate_j \tag{3}$$

The per-hop calculations required for each packet of a flow to determine the earliest time it can be forwarded so the flow does not violate its leaky bucket constraint is the same as in a per-flow shaper. Instead of using a per-flow FIFO, calculating the head-of-FIFO earliest departure time, and then performing appropriate scheduling across all FIFO queues with a head that is ready to be sent, UBS puts packets of all flows received from the same input interface k and destined for the same priority p into the same $FIFO(k, p)$. It then only calculates the earliest departure time for the head of each such $FIFO(k, p)$ based on the leaky bucket constraints of that flow and schedules packets across those $FIFOs(k, p)$ according to strict priority across p and between different k of the same p based on earliest departure time. These $FIFO(k, p)$ are called "Interleaved Regulators" because they interleave multiple flows and regulate the rate of their traffic. Interleaved regulators are the core novel implementation simplification that UBS contributes to the problem of deterministic latency based forwarding.

2.2.2 Cyclic Queuing and Forwarding

In IEEE TSN Cyclic Queuing and Forwarding [3] deterministic traffic packets are sent via periodic time cycles. Received packets are assigned to a cycle purely by their receive time being within the cycle time. This requires strict time synchronization across forwarders and speed-of-light link propagation time must be known and packets only be sent if their last bit will arrive on the receiving forwarder before the cycle time ends there. The higher the link propagation latency, the lower the percentage of time during a cycle that packets can be sent. Typical CQF networks can therefore only span few kilometers in diameter.

2.2.3 Tagged Cyclic Queuing and Forwarding (TQCF)

Qiang et al. [13] (updated by Liu and Dang [14]) follows the model of CQF but attaches a cycle tag to the packet, therefore removing the size limitation of networks to support wide-area IP or MPLS networks.

CQF identifies the cycle to which a packet belongs by the time of arrival of the packet on the router. By contrast TCQF uses a packet tag to indicate the cycle. TQF can therefore support arbitrary link latencies. When using four or more cycles, it can also support links with jitter.

Informally, this means that packets are transmitted with a cycle identifier, analogous to the identification of a time slot in time-division multiplexing technology. On every hop, packets for one cycle are switched to the next hop mapped onto a corresponding "matching" cycle. Routers are periodically cycling through a set of tags used to identify the cycle of the packets, in which each packet is assigned a given tag. Packets of a given tag that are received from an incoming interfaces are forwarded using a packet with a matching tag, cycled through at the outgoing interface.

Link propagation latency has to be known but can be arbitrary large, and receivers can still appropriately map received packets into the desired cycle buffer.

The downsides of TCQF are that it still requires clock synchronization, although its Mean Time Interval Error (MTIE) can be one or two orders of magnitude (factor 10..100) larger than required for CQF. This reduces the overall cost of clock synchronization significantly; however, it does not eliminate it completely. Likewise, TCQF does not have an easy option to support different priorities/latencies at the same time as UBS does. The biggest issue though is the selection of the cycle time (as in CQF): If it is too large, it creates too much undesirable latency end-to-end. If it is too small, the total number of flows that can be supported is too small - because you need to be able to put one packet of each flow into a cycle for simple designs.

2.2.4 Latency Based Forwarding

Despite its similar name, Latency Based Forwarding (LBF) [15] started from a different set of goals than gLBF. LBF is based on the model that applications would indicate their Service Level Objective (SLO) for end-to-end latency in each data packet, specifying a desired [min..max] latency range. The network would then, without any form of upfront resource reservation, attempt to meet those demands against competing demands from other packets by matching QoS actions against an implied latency budget. LBF is part of a larger effort to examine how better network packet header information can support better network services. Like gLBF, LBF can be supported by data plane protocols that provide support for ancillary data, as will be explained in Sect. 4.6. The ancillary data is used to carry parameters that indicate the desired latency range and that are used to measure and record the latency that is incurred by each packet.

The resulting forwarding algorithm tracks the actual latency of a packet hop-by-hop, predicting the remaining path latency through controller or routing protocols, and ultimately calculating on each hop a local queuing budget for the packet based

on SLO, latency incurred, and remaining path information. The packet is then prioritized through the use of intelligent queuing, such that packets that have already been delayed on prior hops would be subjected to less queuing delay on this hop, all other requirements being equal. This approach opens up a range of new deployment options, such as hiding latency differences across different paths in a wide area network by forcing packets to be delayed on every hop by the minimum required latency.

LBF's goal is thus slightly different from that of other mechanisms: rather than attempting to make latency deterministic (without regard for the end-to-end latency that is achieved, and ultimately the responsibility of properly engineered paths that are controlled e.g. by a PCE), the goal is to meet a defined end-to-end-latency (without mandating determinism on any particular hop or relying on a controller). Integration of LBF with congestion control mechanisms from TCP and other transport layer protocols is future research work.

Because of the flexibility and complexity of the forwarding algorithms of LBF, there has so far been no calculus for guaranteeing latency for traffic of guaranteed bandwidth in conjunction with LBF forwarding. Upon examination of the UBS calculus and comparing the likely UBS forwarding plane implementations with the ones proposed for LBF, it became clear that a variation of LBF as presented in this paper can close this gap. This is the origin of the work presented in this paper.

3 Problem Analysis

The previously described bounded latency mechanism have issues in the face of constraints such as those of network equipment design, network deployment/operations and applications [16]. Some of these issues resulted in the design and implementation of TCQF [13], Sect. 2.2.3. gLBF builds on the experience with these prior mechanisms, combining their benefits and eliminating their drawbacks.

This section summarizes these problems and contributes the analysis for industrial control loop timing behavior to it.

3.1 Per-Hop, Per-Flow State

GS and UBS require (interleaved) regulator ("shaper") state (and corresponding memory to maintain it) for every flow on every hop where flows merge, which in typical topologies is every hop. While such state is common in e.g. lower speed (10 Gbps) in-building switches, it is uncommon and expensive to build at low-cost and for large number of flows in routers with 100 Gbps speed per interface or higher that are common in wide area networks. When, as in GS/UBS, it is desired to not introduce any queuing latency in the absence of competing traffic, this so called "regulator" state is unavoidable to compensate for uncontrollable latency on the prior hop that happens because of traffic from multiple interfaces colliding on the same outgoing interface.

3.2 Per-Hop, Per-Flow Steering State Issues

Per-hop, per-flow state has to be signaled from a resource reservation system, such as the AC in the reference model, whenever a flow is added/deleted/changed and often also, when it needs to change its path because of network topology changes including link/node and other component failure or recovery events.

This creates highly undesirable churn/stress for intermediate hop routers for large scale networks with large number of flows. This experience was gained in IP/MPLS networks. As a result, "Segment Routing" (SR-TE) [17] was developed. For traffic steering, SR-TE replaces the need for per-hop, per-flow state from prior solutions, especially Resource Reservation Protocol extensions for Traffic Engineering (RSVP-TE) [18]. It has become accepted practice in wide-area networks deployed today, specifically with the architectural goal to eliminate per-hop state and the need to signal it.

3.3 Dynamic Application Per-Hop, Per-Flow State Issues

Traffic steering for unicast traffic is typically done independent of individual user traffic. This is done to optimize the overall network goodput by spreading traffic out across all available paths. When traffic steering is done to enable specific user traffic performance, of individual service-instances, it is usually provisioned on a case-by-case basis, making it expensive to manage.

In IP-Multicast with the most commonly deployed protocol is "Protocol Independent Multicast Sparse-Mode" (PIM-SM) [19], the network is operating with per-hop, per-flow state to steer and replicate the flows multicast traffic. This state is typically not explicitly provisioned, but automatically and dynamically created whenever sender and receivers of a multicast application are started.

For example, when a user changes TV programs with IPTV, the multicast tree for the old TV channel is pruned back from this receiver and the tree for the new IP multicast tree is extended to this receiver. When this happens for applications such as video surveillance applications across service provider wide-area networks, the actions of such a user immediately impact the hop-by-hop PIM-SM control plane. At scale, this churn of changing multicast trees creates a highly undesirable load on the networks control plane, which typically is not built for such highly-dynamic activities. Even worse, this ability of users to dynamically change hop-by-hop state also opens up the network to severe Denial-of-Service attacks (DoS).

For this reason, as with unicast traffic steering, per-hop, per-flow stateless solutions where developed. This lead to the "Bit Indexed Explicit Replication" (BIER) [20] in which the packet header contains a bit-string where every bit addresses a target receiver, avoiding the establishment of per-hop, per-flow state.

While currently applications that require bounded latency services need to have those services provisioned statically, it is highly desirable to allow for the dynamic instantiation of such services in the future, analogous to the way this

occurs for IP multicast applications. That, in turn, can be facilitated if the need to maintain per-hop, per-flow state is avoided.

3.4 Per-Hop, Per-Flow Shaper State Cost

RSVP-TE for traffic steering only requires per-hop, per-flow steering state, which is easily realized in hardware and also required for non-traffic engineering routing, such as simple IPv4/IPv6 routing. Per-hop, per-flow shaper state for bounded latency on the other hand is more hardware expensive especially at high speed, and routers typically have far less of such state than steering state.

GS requires scheduling across all flows (per-flow shaper). UBS requires shaping (called interleaved regulation there) only across O(#interfaces * #priorities) interleaved regulators. This makes UBS hardware less expensive than GS for switches with a small number of interfaces, such as in industrial Ethernet switches where commonly #interfaces ≤ 24.

For aggregation routers in wide area networks, there can typically be many hundreds of interfaces, making UBS's cost advantage over GS relatively smaller compared to that industrial settings which typically have fewer interfaces. As a result, the hardware cost reduction that UBS enables compared with the cost of GS in WAN deployments is likely an order of magnitude less compared to industrial Ethernet switches.

Shaper state is more expensive to implement because, unlike steering state, it is not simply read-only data and can therefore not be arbitrarily cached. Instead, the process of shaping traffic requires a lookup(state)-read-calculate-write loop for every packet that is subjected to shaping. Even when the total amount of traffic that needs to be subjected to shaping is only a small percentage of the overall traffic (as is often the case in many service provider networks), the cost of writing to state-table memory does rise with the speed of the router and its interfaces. Therefore, per-hop, per-flow shaper state becomes increasingly an issue as the speed of router interfaces grows. Edge routers, for example, typically have slower interfaces than core routers, therefore per-hop, per-flow operations are ideally confined to just those edge-routers per our reference model.

3.5 Clock Synchronization

IEEE1588 is today's standard for high-accuracy network-wide clock synchronization and it has been adopted through various profiles to various market requirements. However, clock synchronization is also a complex component in systems. Solutions that eliminate or reduce the need for clock synchronization can therefore contribute significantly to simplification and/or cost reduction.

Clock synchronization can be undesirable for the operational complexity, which includes often cumbersome aspects such the as offline determination of asymmetric link propagation latencies, especially in wide area networks. In addition, problems are associated with the fact that clock synchronization in IEEE-1588 operates on a tree that needs to be configured, as well as the fact that the accuracy of

synchronization across links that are not part of that tree but that might be part of a redundant network are not determined automatically.

Clock synchronization can also be undesirable because of the additional hardware cost required for media interface chips to insert time stamps from a local oscillator, as well as the hardware cost of that oscillator. Oscillator cost depends on the accuracy that is required. CQF in particular requires clock accuracy that is proportional to the interface speeds. As a result, clock synchronization that is needed to support CQF becomes more expensive the faster the network links are. Today, clock synchronization is common and accepted in some networks, especially in slower (10 Gbps and below) networks and in short-range networks, such as industrial networks in manufacturing plants, but it is uncommon in wide-area networks beyond mobile fronthaul links.

TSN-ATS was in part developed to support a bounded latency solution for TSN that does not require clock synchronization. GS, too, does not require this. Nevertheless, not requiring clock synchronization for the bounded latency algorithm itself is not sufficient to eliminate the need for clock synchronization within the network devices or applications that require bounded latency. This is especially true for many applications when they use GS or TSN-ATS, as will be explained in the following sections.

3.6 In-Time and On-Time Jitter

Path latency is roughly composed of queuing latency and non-queuing latency, such as the propagation latency of links (speed of light) and other, mostly fixed non-queuing propagation latencies in forwarders. The difference between the maximum and minimum latency that can be incurred is called jitter. Of particular concern to be considered is queuing jitter. In GS and UBS, queuing latency is zero in the absence of competing traffic. In the presence of the maximum permissible amount of competing traffic, queuing jitter is at maximum, resulting in the guaranteed (maximum) bounded latency. This means that queuing jitter in GS and UBS is the maximum of any bounded latency solutions. It has also been called “in-time” delivery, because packets will never be too late, but they can be as early as possible.

In (T)CQF mechanisms, queuing jitter is to the largest extend independent of the amount of competing traffic, and tightly bounded by the time of a cycle. This is also called (near) “on-time” delivery because packet will never arrive significantly earlier than their predetermined bounded latency. The benefits of this are described in detail in the following two points.

3.7 Jitter and Network Size Dependent Playout Buffers

Because the amount of competing traffic may change instantaneously, deterministic applications must be able to deal with packets arriving with maximum latency, even if all prior packets arrived with minimum latency. Many important type of applications can effectively only process data synchronously.

The most simple example of such applications is streaming media where data for e.g.: video frames has to be played out at fixed periodic intervals such as 50 (Europe) or 60 (USA) times per second. Any data arriving too early has to be buffered in playout buffers. When the network provides only in-time bounded latency, the jitter and hence the size of the receiver client devices playout buffer will depend on the size of the network (path), and hence those client devices have to predict the maximum network size/jitter they are prepared to support.

This playout buffer sizing issue may not be a problem for today's memory heavy consumer devices which are capable of buffering tenths of seconds of streaming media. However, it was an issue for early Set Top Boxes (STB) built for digital cable TV in the past. When these STBs where enhanced to support TV over IP (IPTV), their playout buffer were sometimes too small for the jitter of the IP network path and the hardware had to be redesigned. Today, playout buffer sizing due to in-time network services may be a problem primarily for IoT devices, especially in cases when their HW resources are constrained for cost, size, heat or energy consumption reasons.

3.8 Clock Synchronization Because of Jitter

Consider a typical industrial control loop as shown in Fig. 2. A Programmable Logic Controller (PLC) controls operations of an Actor. It does so by using measurement data from a Sensor and calculating through some algorithm appropriate new command data for the Actor. In the common setup shown on the left hand side, the PLC sends a request for measurement data to the Sensor, which immediately measures the requested data and sends the response data packet back to the PLC. After taking this data into account in the algorithm, and determining if and when the actor needs new command data, the PLC sends that newly calculated command data to the Actor, which is expected to act on it immediately upon receipt.

As long as network propagation latencies are on-time (known and fixed), the PLC can exactly control the time at which measurements are taken by the sensor and the time at which action data is acted upon by the Actor. Sensor and Actor can use very

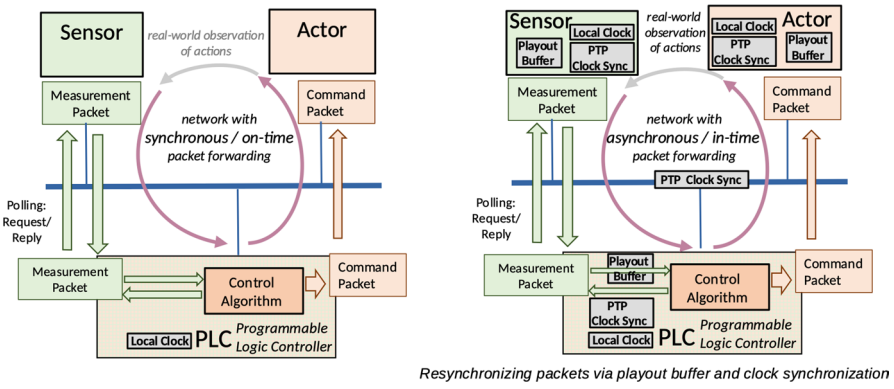

Fig. 2 Industrial Control Loop and the impact of jitter

simple and inexpensive designs, just having to react to arriving packets immediately and not having to consider timings.

As soon as significant jitter in packet transmission occurs, for example with an in-time bounded latency solution such as TSN-ATS, the PLC loses this exact control ability. As a result, additional mechanisms are required. This increases the complexity of the solution.

In the commonly used approach shown on the right-hand side of the picture, PLC, Sensors and Actors implement a clock synchronization protocol, such as PTP. Packets are sent with a sending timestamp and delayed on receipt for further processing until they have reached the bounded latency of the network path. This approach creates an on-time transmission experience using a clock-synchronization driven playout buffer, at the expense (quite literally) of adding complexity and hardware cost.

This example shows how a bounded latency mechanism in the network that has jitter (in-time) may be able to operate without clock synchronization itself, but where that very same solution then introduces the need for clock synchronization in the network and application devices.

3.9 Problem Summary

Per-hop, per-flow state of asynchronous deterministic latency solutions such as GS and UBS is highly undesirable in wide-area networks, solutions such as TCQF can provide bounded latency without this problem. TCQF still requires clock synchronization across the network. Eliminating the need for clock synchronization in the network such as required for (T)CQF is highly desirable, but for wide area networks it is not sufficient to do so at the cost of making jitter become order of magnitude larger and depending on network size. In summary, what is needed is a mechanism that provides deterministic latency without needing to maintain per-hop per-flow state (reducing associated complexity and hardware cost) and without the need for clock synchronization to provide tightly bounded jitter. This is exactly what gLBF provides.

4 gLBF

4.1 gLBF Concept

The root problem for deterministic latency guarantees is to avoid multi-hop packet bunching which happens when packets from multiple flows pass through the same interface(s). On an intermediate hop, a packet P1 from one flow may end up behind a long interface queue and be delayed accordingly, but the next packet P2 from the same flow may arrive when the queue has already cleared and pass through the interface without any queuing delay. As a result, on the next hop, P1 and P2 are bunched together. If P1 is not a single packet, but perhaps a burst of multiple packets, bunching it up with P2 may result in a burst size that exceeds what would

be permissible as the flow's maximum burst size, potentially resulting in collisions and packet drops. This is also called "(micro)burst-accumulation" and it gets worse across multiple hops. As summarized in [21]:

"... computing worst case delays in FIFO networks is a very difficult exercise [22]. Even in feed-forward topologies, finding proven bounds that are close to the worst case is difficult: the determination of the worst case is NP-hard [23] and can be addressed by linear programming [24]."

For this reason, all methods to operationalize deterministic latency guarantees are employing some means to undo this packet bunching so that simple accumulative per-hop arithmetic can be used to calculate the end-to-end latency from per-hop latencies.

In gLBF, each hop measures the packets queuing latency through its interface, calculates the difference between this queuing latency and the guaranteed latency for the hop, signals this remaining latency value in the packet to the next hop, and that next hop then delays the packet by this value. This creates virtually zero-jitter guaranteed latency forwarding (quasi-synchronous) and operates without per-flow state and without network wide clock synchronization—both are not required because of the in-packet field for the remaining per-hop latency. gLBF also allows to use the same latency calculus as UBS and provides therefore the same guaranteed latency.

In comparison, GS and UBS do require per-hop, per-flow state because they do measure each flow's burstiness on every hop and delay a packet when it would otherwise be processed too early (clogged) with respect to the prior packets of the same flow. This also introduces worst-case jitter: zero in the absence of competing traffic, and maximum in the presence of maximum competing traffic.

Recording and signaling the delay that is expected of the next node as part of a packet requires use of a data plane protocol that supports ancillary data. Examples include IPv6 with extension headers, New IP, or newly proposed variants of MPLS, as will be explained further in Sect. 4.6.

4.2 gLBF Mechanism

Referring to the designators in the gLBF part of Fig. 3, gLBF operates as follows.

We assume that at X2, all packets comply with their flows' leaky bucket condition because of prior operations. The maximum delay of a packet through Forwarder 2 FIFO is *MAX_FIFO*. This can be calculated from UBS calculus or simply as the latency of a bit through the full FIFO. *MAX_LINK* is the maximum serialization time of the largest permissible packet through link L1.

At X3, Forwarder 1 measures the packets delay through its FIFO as *FIFO_latency*, (<= *MAX_FIFO*). It also calculates *packet_serialization_delay* from the packet size and link L1 speed. It finally calculates $glbf\_delay = MAX\_FIFO + MAX\_LINK - FIFO\_latency - packet\_serialization\_delay$. This value is signaled as a new packet header field in the packet to Forwarder 2. Forwarder 2 delays the packet by this value before passing it to point Y2.

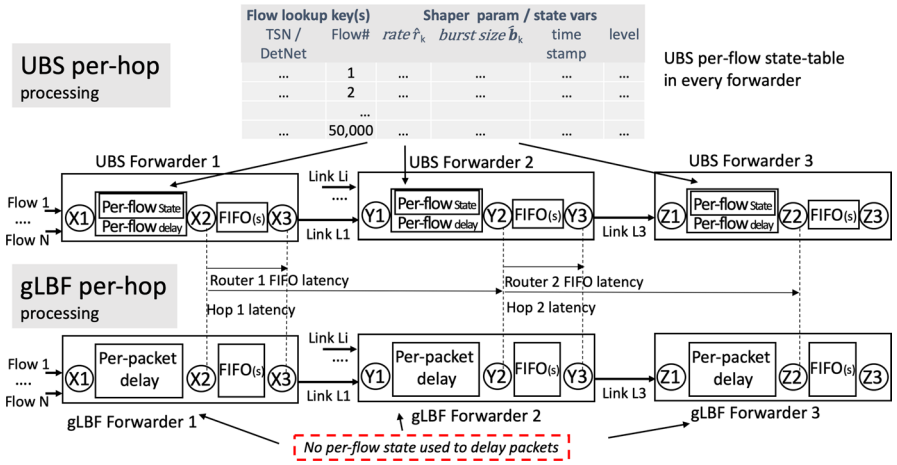

Fig. 3 UBS and gLBF per-hop processing

The result of this gLBF mechanism is that all packets will arrive at Y2 at a time $Hop1\_Latency = MAX\_FIFO + MAX\_LINK$ later than their arrival time at X2. Therefore they will all comply with their leaky bucket condition at Y2 if they did so at X2. This condition likewise applies to any further gLBF hops. When a forwarder receives packets from an untrusted sender and/or without the glbf_delay parameter, it needs to perform per-flow shaping so packets conform to their leaky bucket conditions before entering the gLBF network.

With a constant delay for every gLBF hop, all packets are forwarded with their maximum bounded latency as calculated by e.g. UBS calculus. Unlike UBS, gLBF will also ensure that packets are delivered with very tightly bounded jitter, which is determined solely by the latency through the last hop Forwarders FIFO, whereas the jitter incurred on any other hop across the gLBF network is compensated for by the gLBF delay operations. If the actual receiver application also understands gLBF, it can also implement the gLBF delay stage and effectively process packets with zero path jitter after that stage.

4.3 ASIC-Based gLBF

Sequential delay and FIFO stages will incur performance issues or additional delay in real-world ASIC implementations. The reason for this is that in the worst case, all packets may need to be passed at exactly the same time across X2, for example because each packet came from a different original sender and was sent at the same time. Both stages can be merged into a single Push-In-First-Out (PIFO) queue, in which the rank of each packet is its desired time of release from the delay stage (e.g. Y2 for Forwarder 2). This is depicted in Fig. 4.

When the packet arrives at time $T1$ at Forwarder 2, Y1, it calculates $rank = T1 + glbf\_delay$ and enqueues the packet into the PIFO. When the packet is dequeued at Y3, time $T3$, it calculates

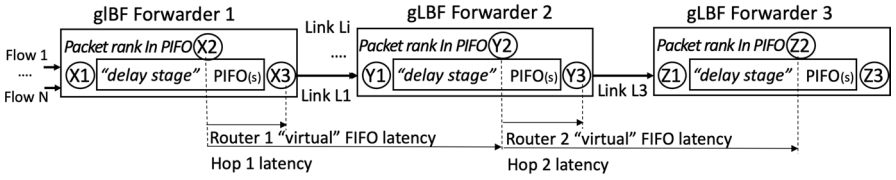

Fig. 4 gLBF with PIFO(s)

$glbf\_delay = MAX\_FIFO + MAX\_LINK - (T3 - rank) - packet\_serialization\_delay$ $(T3 - rank)$ is the time that the packet had to wait longer in the PIFO than its desired *gLBF_delay*, aka: the equivalent of *FIFO_latency*.

4.4 Multi-priority gLBF

gLBF can be expanded to support multiple priorities and therefore latencies in the same fashion as UBS does by using a separate PIFO for each priority and dequeueing packets from the highest priority PIFO whose head of queue packet is ready, e.g. ($rank < current\_time$). Each PIFOs has its own *MAX_FIFO* parameter and may have a different *MAX_LINK* parameter for purposes of gLBF calculations.

For per-flow stateless operations with multiple priorities, each gLBF hop needs to learn the packets priority from a packet header. This can be a single priority parameter for all hops, or a sequence of priority values, one per-hop, to allow different priorities across different hops, as UBS does as well.

In Segment Routing for IPv6 (SRv6) network technology, a sequence of (128 bit) IPv6 addresses is used to steer packets through the network. Therefore a similar type of header, requiring for example 4 bit per hop for up to 16 priorities per hop, would be a relatively insignificant impact to overall packet header size.

4.5 FIFO Based gLBF

100 Gbps or faster PIFO are subject of research for high-speed ASIC/FPGA, but not available in shipping products. We describe how to utilize single stage FIFO, using the design and calculus experience from UBS single stage FIFO queuing.

In UBS, every (IIF,P) combination of Incoming InterFace (IIF) and Priority (P) has its own interleaved regulator (IR), which is a FIFO where packets are dequeued using per-flow (regulator) state on the head of queue packet. Only packets for the same (IIF,P) can share the same IR because only their arrival order is also their departure order. Likewise, in gLBF, each PIFO for a particular priority P can be replaced by a set of (IIF,P) FIFO combined with the logic to dequeue packets based on the earliest rank of the head of those FIFOs.

(IIF,P) FIFOs cannot support different priorities for a packet on different hops, because the FIFO into which a packet needs to get enqueued is the priority of the packet on the receive-side hop of the forwarder (up to point 2), but the dequeuing of the packet is based on the packets priority of the send-side hop of the forwarder (up to 3). Only when those two priorities of a packet are the same can it stay in the

same single-stage FIFO, or else dequeue order is not the same as enqueue order. A single packet header priority parameter can support this model. Single priority deployments require no priority header field at all and just a FIFO per IIF to avoid PIFO implementation.

This per-path priority limitation can be resolved by a combination of (IIF,Input-Prio,Output-Prio) FIFOs and according dequeuing logic, but this would result in a logic more complex than UBS, and is therefore just one out of likely multiple options for implementing full PIFO queuing logic in longer-term hardware designs.

4.6 Network Protocols for gLBF

In summary, the following per-packet ancillary data parameters are required in support of gLBF.

- For basic gLBF, the gLBF_delay parameter is required. It indicates the delay that needs to be inserted at the next gLBF forwarder before the packet can be queued. gLBF_delay is computed at each node by subtracting the local queueing delay that was incurred from the targeted hop_delay. It is rewritten by each gLBF forwarder.
- For multi-priority gLBF, in addition to gLBF_delay, a per-hop sequence of priorities is required. In addition, an index into this sequence is required which is updated on every hop to indicate the priority index that is to be used by the next hop.
- For a simpler per-path priority based gLBF variation, such as when using FIFO to implement multi-priority gLBF, a single priority parameter is required instead of a per-hop sequence of priorities.

While these parameters are algorithmically simple, operationalizing them into packet headers of network protocols incurs several additional decisions.

- The time resolution of gLBF_delay need to be determined. The resolution determines the accuracy of the on-time delivery. Resolution could for example be in nsec, as in our validation in Sect. 5 below. Resolution errors are per-hop additive.
- The size of gLBF_delay needs to be determined. With a nsec resolution, a 24-bit sized gLBF_delay parameter would allow to indicate delays up to 16 msec. For most applications requiring control-loops that operate in the tenths of msec or lower, this would be sufficiently large.
- An encoding needs to be determined to indicate whether or not gLBF and any of its enhancements is to be performed on the packet. With fixed parameter encodings, this is usually done by designating some “do-not-use” value, such a a all-bits-1 value for gLBF_delay to indicate that gLBF is not to be applied on the packet. However, we prefer a variable encoding where the absence of the gLBF_delay indicates that gLBF is not to be performed.

Other ancillary data parameters may be useful to be carried in conjunction with those for gLBF itself, such as a parameter indicating the end-to-end SLO as defined by LBF.

With the variety of encoding decisions and options, it is not clear to the authors that all networks would benefit from a single encoding. For example, low-bitrate, low-power radio networks might benefit from different choices of resolution and size of gLBF_delay than 100Gbps++ WAN networks.

One core issue deciding about the details of packet header encoding is how to maximize the agility of adoption of such a novel network technology. Traditionally, network packet header encoding was optimized for high-cost, low-flexibility ASIC implementations, leading rather inflexibly headers that most often are fixed-size without multiple optional parameters. In contrast, software based solutions such as those found in web-applications achieve higher agility by more flexible encodings, employing mechanisms such as optional and variable length parameters and/or deployment specific templates of parameters.

Currently, the latest of network forwarding hardware is moving more and more towards a programmable infrastructure, so this might be a good time to explore those more flexible encoding options to achieve higher agility and easier adopt novel processing mechanisms of packets such as gLBF to different type of networks and use-cases easier.

The authors therefore think is most beneficial if the target network protocol allows to carry parameters with flexible encoding.

One approach is to include gLBF parameters in an established encoding approach such as an IPv6 extension headers [25]. “Segment Routing Header” (SRH, [26]) is a widely used example of such a header, in which parameters are encoded as TLVs and in which there is also a sequence of per-loose-hop header fields with a pointer to the next active one. Another possible candidate for such protocol is New IP [27–30], which supports to encode ancillary data as metadata) in a so-called flexible packet contract. Likewise, there are newly proposed variants of MPLS that would allow for the encoding and processing of metadata as part of a label stack such as [31].

5 Validation

To validate gLBF, we developed a simple, purpose-built time-discrete simulation tool in Perl [32] to understand and diagnose gLBF behavior with as little as possible software coding. Our test setup uses the topology of Fig. 5. The target behavior of gLBF is to deliver the same per-latency for every packet of a flows across a hop (in the diagram, from reference point X2 to reference point Y2). Therefore we considered it to be sufficient to simulate gLBF primarily over one network fragment with two adjacent hops for its validation, complemented with a small number of additional nodes to generate competing traffic.

Composite multi-hop behavior across multiple network fragments can easily be extrapolated by calculating the respective fragment latencies. End-to-end latency is determined by the sum of node latencies as well as link propagation latecy. Conversely, this means that latencies for each hop can be determined by

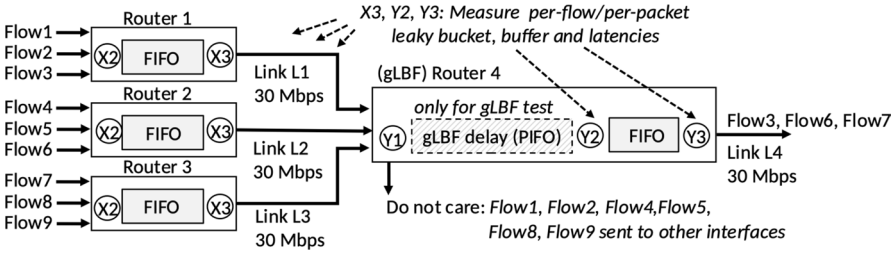

Fig. 5 Validation test setup

considering the targetec end-to-end latency, then assigning latencies for each hop based on factors such as the number of hops on the path as well as link latencies that are incurred across the path.

This tool operates at simulated 1 nsec clock to allow bit-accurate measurements for link speeds up to 1 Gbps. Test runs cover 1 second worth of traffic. In the following, we summarize validation of the most conclusive test scenario we tested.

In a first set of test runs (that we not detail here), we ran some brute-force search for a combination of flow parameters that would maximize the problem that gLBF intends to solve: the micro-burst accumulation across a single hop that leads to the aforementioned NP-hard calculations across multiple hops. Once those parameters where determined, we validated and quantify that gLBF solves these problems through the introduction of the gLBF latency step.

Router 1, Router 2, and Router 3 are set up to each receive three rate-controlled traffic flows, each with a bitrate of 10Mbps, burst sizes of 3 packets, and slightly different packet sizes per flow: Router 1 = (900, 1000, 1100) bytes, Router 2 = (930, 1030, 1130) bytes and Router 3 = (1370, 1170, 970) bytes. These are values that maximize burst accumulation in the absence of gLBF. All nine flows arrive on Router 4, but only Flow 3 (from Router 1), Flow 6 (from Router 2) and Flow 7 (from Router 3) are forwarded across Router 4 FIFO to L4, and are therefore subject to the gLBF processing for the output interface towards L4. The remaining flows are routed by Router 4 to other interfaces that would also employ gLBF, but that are not simulated.

We use three routers (1, 2, 3) to feed Router 4 because, if the output interface towards Link 4 from Router 4 were fed by just a single interface (of the same speed) on input, there would be no further burst accumulation on Router 4 in the absence of gLBF. In other words, even though our focus is on validating gLBF between Y1 and Y2, the test setup is chosen so that further burst accumulation would occur between Y2 and Y3 in the absence of gLBF. In result, we are interested in the simulation of three (gLBF) hops:

(1) from Router 1 point X2 for Link L1 to Router 4 point Y2 for Link L4,
(2) from Router 2 point X2 for Link L2 to Router 4 point Y2 for Link L4, and
(3) from Router 3 point X2 for L2 to Router 4 point Y2 for Link L4.

Upon origination (into Router 1,2,3), all three flows stay within their leaky bucket constraint, which is why we also show traffic entering into point X2 in Router 1, 2, 3:

$$w(i,d) \leq b(i) + d * r(i) \tag{4}$$

where $b(i)$ is the burst size of flow i, $r(i)$ is the rate of flow i (10 Mbps), d is a period of time, and $w(i, d)$ is the maximum permitted total amount of data of flow i across period d.

To recognize bust-accumulation, the validation tool measures compliance of a flows packets with the flows leaky bucket condition using the leaky bucket algorithm of [1] applied to the timestamp every packet of the flow is observed with:

```
// Initialization
for(i:I) { state[i].timestamp := 0;
           state[i].level := b(i); }

receive(p, i, t) { // packet p of flow i arrives at t
  state[i].timestamp = dt = t - state[i].timestamp;
  state[i].level += dt * state[i].rate;
  if(state[i].level > if(state[i].bsize)
    state[i].level = state[i].bsize;
  state[i].level -= 8 * p.length;
  if(state[i].level < 0)
    state[i].levelerrors++;
```

When a flow exceeds its leaky bucket condition, *level* turns negative. Packets of the flow during this period could use more buffer space than allocated in a real system and cause higher latencies on the following hop FIFO.

In our first set of validation runs without gLBF, we observe that flows stay within their calculated latencies across their Router 1, 2, 3 FIFOs, measured between X2 and X3. For example, Fig. 6 shows the latency of a test run period of circa 0.1 seconds worth of packets from Flow 3 across the Router 1 FIFO from X2 to X3. The picture visualizes how the three back-to-back packet bursts of Flow 3 cause periodic increases in those packets' latencies. When intermixed with the same type of three-packet bursts

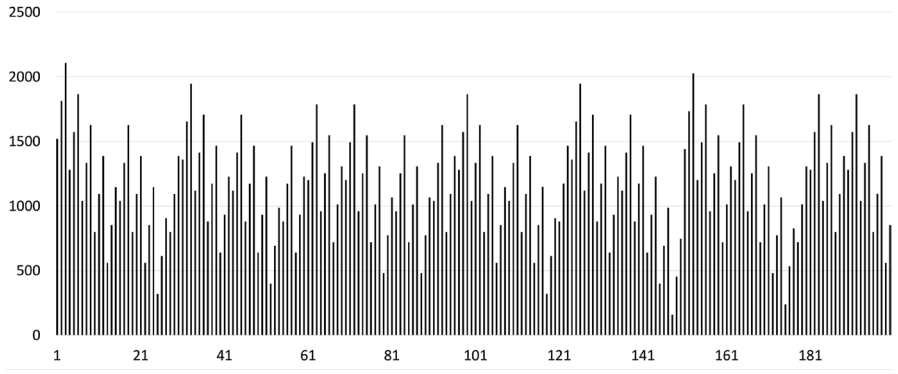

Fig. 6 Latency (X3 - X2) of Flow 3

of Flows 1 and 2, but with slightly different packet sizes, slightly different inter-packet arrivals result. This is due to to a variation of latency of individual packets up to the maximum latency, which can be calculated from the burst sizes of the aggregate according to the following formula (all flows are configured to deliver 10 Mbps each).

The maximum latency *MAX_FIFO* in our validation is the time it takes to serialize the sum of the bursts of all flows except for one packet of the flow in question, because that packets own latency is not accounted against *MAX_FIFO* but the serialization latency for the packet itself. For packets from Flow this this is 2.108 msec:

$$(3 * 900 + 3 * 1000 + 3 * 1100 - 1100) * 8/30,000,000 = 2.108\ msec \quad (5)$$

Even though these flows do maintain their calculated latencies through Router 1 FIFO, they do (as expected) exceed their permitted burst levels after the FIFO as observed at the X3 point (or equally at the Y1 point) due to the nature of the FIFO queuing.

Figure 7 shows the issues (with regard to their impact on the level) that result for packets or Flow 3. They likewise exist for the other flows. These issues are a major contributing factor to the fact that the computation of latency incurred at subsequent hops becomes an NP-hard problem. .

To validate these resulting latency issues, the setup forwards Flows 3, 6 and 7 in Router 4 to link L4. We use flows from different incoming interfaces because if we would simply forward Flows 1, 2, 3 and send them out over link L4, we would observe no other timing on Y3 than we did on X3: No latency excesses would occur, just level errors.

The maximum calculated latency for a packet from Flow 3 via Router 4 L4 FIFO is as follows:

$$(3 * 1100 - 1100 + 3 * 1130 + 3 * 970) * 8/10,000,000 = 2.268 msec \quad (6)$$

Figure 8 shows the validation tool observed latency (Y3 - Y1) (without gLBF between Y2 and Y1), aka: the latency through Router 4 FIFO to link L4 for all 1138

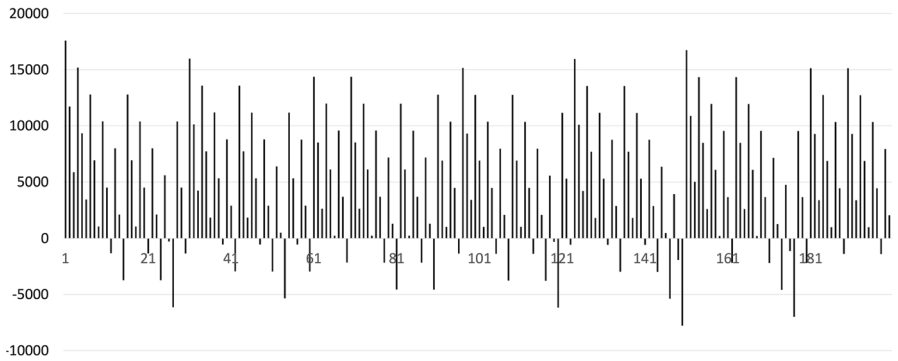

Fig. 7 Burst Level violations of Flow 3 at point X3

packets of Flow 3. It shows that 16 packets or 1.5% exceed the calculated latency, proving the existance of the burst accumulation problem, which gLBF aims to solve.

Likewise, the total amounts of buffer space calculated for Router 4 L4 FIFO is $3 * 1100 + 3 * 1130 + 3 * 970 = 9600$ *bytes*, but the validation shows that the maximum buffer utilization was $11,540$ *bytes*, therefore also proving that not only the latency, but also buffer utilization exceeds easily calculated bounds.

Our second type of validation runs shows the behavior of the same setup when, in contrast to before, gLBF is activated in Routers 1, 2, 3, 4. For Routers 1, 2, 3 this means we do activate the calculation of *glbf_delay* at X3 (and insertion into the packet header) and for Router 4 we activate the gLBF delay component reacting on it. Ultimately, we do want to see that the latency and buffer utilization on the following hop, Router 4 (Y3 - Y2) stay within the target bounds.

Figure 9 shows again the latency of 0.1 seconds worth of Flow 3 through Router 1 FIFO, from X2 to X3, but this time with gLBF_latency also depicted. The black lines are the latency through the FIFO, the grey stacked lines are the gLBF latency calculated at X3 and then inserted into the packet header. As can be seen, the FIFO latency is always lower or equal to the calculated worst case latency of 2.108 msec, so the calculated glbf_latency is larger or equal to 0, hence Router 4 can correctly apply the *gLBF_latency* delay.

As a result of enabling gLBF_delay on Router 4, we observe a fixed latency after gLBF_delay (Y2 - X2) for all packet: 2.4 msec for Flow 3, 2.47 msec for Flow 6, and 2.81 msec for Flow 7. These hop latencies are the calculated maximum FIFO queue latencies for Router 1, 2 and 3 plus the propagation latency of the observed flows largest packet, e.g for Flow 3 the above calculated 2.108 msec FIFO latency and 0.292 msec serialization latency of the Flow 3 1100 byte packet.

The constant latency across all packets of a Flow at Y2 validates the zero jitter that gLBF achieves across a $FIFO + gLBF\_delay$ hop with the same bounded latencies as UBS, completely eliminating any level violations at Y2 that where seen before gLBF was not enabled.

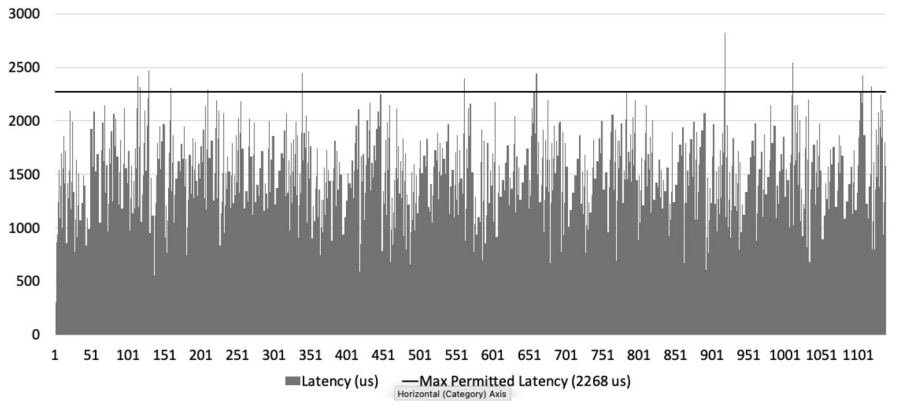

Fig. 8 Latency of Flow 3 through Router 4 FIFO to L4 without gLBF

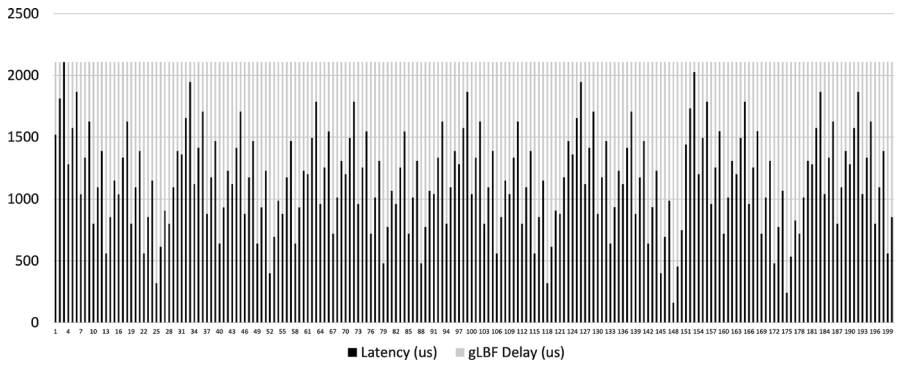

Fig. 9 Latency of Flow 3 through FIFO Router 1, with gLBF

Finally, we observe again the behavior of Flow through Router L4 FIFO, after gLBF_latency is applied, to validate that no excess buffer usage or latency occurs. Figure 10 shows that, unlike in Fig. 8, no excess latency or buffer utilization can be observed. Likewise, the maximum buffer utilization observed is 8630 bytes (out of the theoretical maximum of 9600 bytes for the three flows burst sizes) and a maximum FIFO latency of 2.25 msec for Flow 3.

Note that our tests do not account for the speed of light propagation latency of Link 1, because it does not impact the gLBF algorithm. If there is significant propagation latency, as there would be in any target wide area network deployment, then the physically observed latencies between the X and Y measurement points would simply be larger by that propagation latency than the latencies validated here, but no other impact to the effectiveness of gLBF would be observed.

Even though this validation is only anecdotally through the most simple two FIFO-hop burst aggregation experiment, (X3-X2) and (Y3-Y2)), the details of the

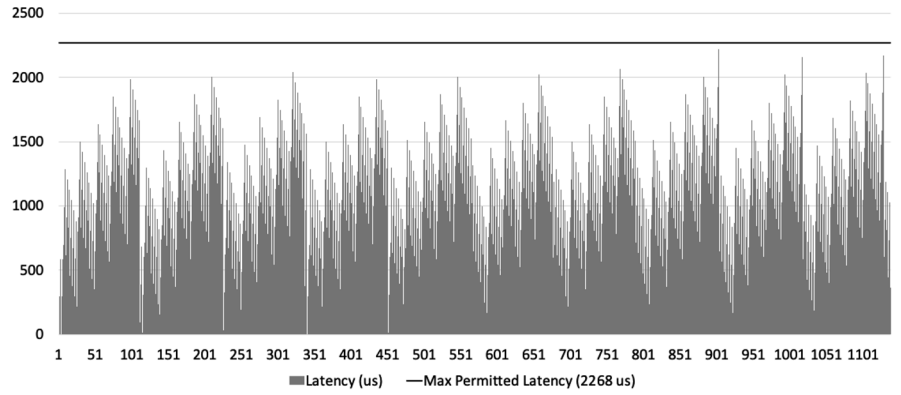

Fig. 10 Latency of Flow 3 through FIFO Router 4, with gLBF

validation do hopefully provide sufficient initial experimental evidence for what the authors think to be theoretically easy to grasp:

gLBF delay of packets reconstitutes packet flows to their prior-hop inter-packet spacing. If on that prior hop, packets of flows stay within their leaky bucket condition, then they too will do so after gLBF delay on the following hop. Therefore the non-exponential calculation of per-hop FIFO latency as done in rate-controlled service disciplines such as UBS can be reused in gLBF, but without the systematic challenges introduced by per-flow shaper/interleaved regulators or cyclic queuing as in prior mechanisms.

gLBF achieves per-hop zero-jitter forwarding relative to the nodes local clock, whereas (T)CQF would introduce jitter in the order of a cycle (e.g. 100usec), and UBS would introduce per-hop jitter in the order of the per-hop bounded latency.

6 Outlook

gLBF as described here does not leverage the mechanisms introduced with LBF to meet a specified end-to-end latency. However, the original concepts of LBF are still applicable even in the context of gLBF. Specifically, LBF's concept of determining a hop's latency budget and its elasticity that allows it to compensate for unforeseen network conditions complements gLBF's ability to provide hard guarantees for specific hop latencies while preventing burst accumulation. We plan to investigate and expand on this aspect as part of our futher work.

Another aspect deserving further investigation concerns the exact behavior of gLBF under differences in clock speed across consecutive hops. The authors believe that differences in clock speeds across hops may result in the need to overestimate per-hop latency and buffer utilization, limiting as a result link utilization in proportion to the relative differences in clock speed. However, experimental and theoretical validation of this are the subject of future work.

7 Conclusions

This paper has described a guaranteed latency based forwarding system that is suitable for both current and future high-speed wide-area networks with interface speeds into the Tbps range. Unlike prior solutions, gLFB does not require per hop state nor does it require network time synchronization. It is capable of multiplexing a variety of services and targeted at network service providers who require more capable and less complex mechanisms to support bounded latency traffic compared to mechanisms that are available today, offering substantial advantages in terms of hardware cost, scalability, operational simplicity and service performance. The fact that gLBF eliminates the need for highly-precise clock synchronization makes it attractive especially in deployments involving low-end IoT devices, which would otherwise struggle to support such services. Many current and even more so future applications will benefit from, or even require, tightly bounded jitter for bounded latency traffic.

gLBF is a new mechanism that enables predictable, virtually-synchronous per-hop-latency, enabling high-precision communication services that provide end-to-end latency guarantees as required by many time-sensitive networking applications. gLBF combines the time-synchronization free multi-priority bounded benefits of UBS with the low-jitter and per-flow-stateless operation of (T)CQF, allowing low-cost implementation and low operational complexity.

The principle of gLBF can be used to compensate for the variability of not only of the queuing part of forwarding and of the (packet size dependent) link serialization latency, but for any variable latency that can be predicted or measured. For example, in complex routers latency through ingress line card or fabric may be variable and/or line cards may not have a common time-base. The adoption of gLBF across these components can be used to create virtually synchronous latency across such complex routers.

As a next step, we plan to submit gLBF for standardization in the IETF in order to prepare it for deployment at scale.

Author Contributions The authors co-wrote the manuscript and jointly developed the concepts. TE was the lead author for most of the sections. AC did an editorial scrub across the paper and contributed several text portions including 2.2.4. and portions of 1, 7. SB contributed section 4.6. Each author reviewed the paper prior to submitting. Test code was developed by TE; test cases were run jointly by TE and AC.

Funding Performed as part of our work for our employer, Futurewei Technologies (Stewart Bryant was a consultant with Futurewei at the time).

Code Availability Per the references, available at https://github.com/network2030/glbf-validation.

Declarations

Conflict of interest The authors declare that they have no competing interest.

Consent to participate Yes.

Consent for publication Yes.

IPR Disclosure A patent was filed with the USPTO on the gLBF mechanism that is described in the paper.

References

1. Specht, J., Samii, S.: Urgency-based scheduler for time-sensitive switched ethernet networks. In: 2016 28th Euromicro Conference on Real-Time Systems (ECRTS) (2016)
2. Shenker, S., Partridge, C., Guerin, R.: Specification of guaranteed quality of service. Internet Requests for Comments. IETF RFC 2212 (1997)
3. 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. https://standards.ieee.org/ieee/802.1Qch/6072/ (2017)
4. Grossman, E.: Deterministic networking use cases. RFC Editor (2019). https://doi.org/10.17487/RFC8578. https://www.rfc-editor.org/info/rfc8578
5. Viswanathan, A., Rosen, E.C., Callon, R.: Multiprotocol label switching architecture. RFC Editor (2001). https://doi.org/10.17487/RFC3031. https://www.rfc-editor.org/info/rfc3031

6. Eckert, T., Clemm, A., Bryant, S.: glbf: per-flow stateless packet forwarding with guaranteed latency and near-synchronous jitter. In: 2021 17th International Conference on Network and Service Management (CNSM), pp. 578–584 (2021). https://doi.org/10.23919/CNSM52442.2021.9615538. https://dl.ifip.org/db/conf/cnsm/cnsm2021/1570754857.pdf
7. Farrel, A., Zhao, Q., Li, Z., Zhou, C.: An architecture for use of PCE and the PCE Communication Protocol (PCEP) in a network with central control. Internet Requests for Comments. IETF RFC 8283 (2017)
8. Vasseur, J., Farrel, A., Ash, G.: A Path Computation Element (PCE)-Based Architecture. RFC Editor (2006). https://doi.org/10.17487/RFC4655. https://www.rfc-editor.org/info/rfc4655
9. Carugi, M., McDysan, D.: Service requirements for layer 3 Provider Provisioned Virtual Private Networks (PPVPNs). RFC Editor (2005). https://doi.org/10.17487/RFC4031. https://www.rfc-editor.org/info/rfc4031
10. Nasrallah, A., Balasubramanian, V., Thyagaturu, A., Reisslein, M., ElBakoury, H.: TSN algorithms for large scale networks: a survey and conceptual comparison. arXiv (2019). arXiv:1905.08478
11. Finn, N., Boudec, J.-Y.L., Mohammadpour, E., Zhang, J., Varga, B.: Deterministic Networking (DetNet) bounded latency. RFC Editor (2022). https://doi.org/10.17487/RFC9320. https://www.rfc-editor.org/info/rfc9320
12. Specht, J.: IEEE Time-Sensitive Networking (TSN) Task Group: IEEE Std P802.1Qcr: Bridges and Bridged Networks Amendment: Asynchronous Traffic Shaping. https://standards.ieee.org/ieee/802.1Qcr/7420/ (2020)
13. Qiang, L., Geng, X., Liu, B., Eckert, T., Geng, L., Li, G.: Large-Scale Deterministic IP Network. Internet-Draft draft-qiang-detnet-large-scale-detnet-05, Internet Engineering Task Force (September 2019). Work in Progress. https://datatracker.ietf.org/doc/html/draft-qiang-detnet-large-scale-detnet-05
14. Liu, B., Dang, J.: A queuing mechanism with multiple cyclic buffers. Internet-Draft draft-dang-queuing-with-multiple-cyclic-buffers-00, Internet Engineering Task Force (February 2021). Work in Progress. https://datatracker.ietf.org/doc/html/draft-dang-queuing-with-multiple-cyclic-buffers-00
15. Clemm, A., Eckert, T.: High-precision latency forwarding over packet-programmable networks. 2020 IEEE/IFIP Network Operations and Management Symposium (NOMS) (2020)
16. Eckert, T., Bryant, S.: Problems with existing DetNet bounded latency queuing mechanisms. Internet-Draft draft-eckert-detnet-bounded-latency-problems-00, Internet Engineering Task Force (July 2021). Work in Progress. https://datatracker.ietf.org/doc/html/draft-eckert-detnet-bounded-latency-problems-00
17. Filsfils, C., Previdi, S., Ginsberg, L., Decraene, B., Litkowski, S., Shakir, R.: Segment routing architecture. RFC Editor (2018). https://doi.org/10.17487/RFC8402. https://www.rfc-editor.org/info/rfc8402
18. Awduche, D.O., Berger, L., Gan, D.-H., Li, T., Srinivasan, D.V., Swallow, G.: RSVP-TE: extensions to RSVP for LSP tunnels. RFC Editor (2001). https://doi.org/10.17487/RFC3209. https://www.rfc-editor.org/info/rfc3209
19. Fenner, B., Handley, M.J., Holbrook, H., Kouvelas, I., Parekh, R., Zhang, Z.J., Zheng, L.: Protocol Independent Multicast—Sparse Mode (PIM-SM): Protocol Specification (Revised). RFC Editor (2016). https://doi.org/10.17487/RFC7761. https://www.rfc-editor.org/info/rfc7761
20. Wijnands, I., Rosen, E.C., Dolganow, A., Przygienda, T., Aldrin, S.: Multicast using Bit Index Explicit Replication (BIER). RFC Editor (2017). https://doi.org/10.17487/RFC8279. https://www.rfc-editor.org/info/rfc8279
21. Boudec, J.L.: A theory of traffic regulators for deterministic networks with application to interleaved regulators. CoRR (2018) arXiv:1801.08477
22. Boyer, M., Fraboul, C.: Tightening end to end delay upper bound for AFDX network calculus with rate latency fifo servers using network calculus. In: 2008 IEEE International Workshop on Factory Communication Systems, pp. 11–20 (2008). https://doi.org/10.1109/WFCS.2008.4638728
23. Bouillard, A., Jouhet, L., Thierry, E.: Tight performance bounds in the worst-case analysis of feed-forward networks. In: 2010 Proceedings IEEE INFOCOM, pp. 1–9 (2010). https://doi.org/10.1109/INFCOM.2010.5461912
24. Bouillard, A., Stea, G.: Exact worst-case delay in fifo-multiplexing feed-forward networks. IEEE/ACM Trans. Netw. **23**(5), 1387–1400 (2015). https://doi.org/10.1109/TNET.2014.2332071
25. Deering, D.S.E., Hinden, B.: Internet Protocol, Version 6 (IPv6) Specification. RFC Editor (2017). https://doi.org/10.17487/RFC8200. https://www.rfc-editor.org/info/rfc8200

26. Filsfils, C., Dukes, D., Previdi, S., Leddy, J., Matsushima, S., Voyer, D.: IPv6 Segment Routing Header (SRH). RFC Editor (2020). https://doi.org/10.17487/RFC8754. https://www.rfc-editor.org/info/rfc8754
27. Li, R., Makhijani, K., Dong, L.: New ip: a data packet framework to evolve the internet: invited paper. In: 2020 IEEE 21st International Conference on High Performance Switching and Routing (HPSR), pp. 1–8 (2020). https://doi.org/10.1109/HPSR48589.2020.9098996
28. Li, R., Chunduri, U.S., Clemm, A., Dong, L.: New IP: enabling the next wave of networking innovation. In: Boucadair, M., Jacquenet, C. (eds.) Design Innovation and Network Architecture for the Future Internet, pp. 1–42. IGI Global, Hershey (2021)
29. Dong, L., Mak, K., Li, R.: Qualitative communication via network coding and new ip: invited paper. In: 2020 IEEE 21st International Conference on High Performance Switching and Routing (HPSR), pp. 1–5 (2020). https://doi.org/10.1109/HPSR48589.2020.9098976
30. François, J., Clemm, A., Maintenant, V., Tabor, S.: Bpp over p4: exploring frontiers and limits in programmable packet processing. In: GLOBECOM 2020—2020 IEEE Global Communications Conference, pp. 1–6 (2020). https://doi.org/10.1109/GLOBECOM42002.2020.9322572
31. Bryant, S., Clemm, A.: Token cell routing: a new sub-ip layer protocol. In: 2021 17th International Conference on Network and Service Management (CNSM), pp. 153–159 (2021). https://doi.org/10.23919/CNSM52442.2021.9615569
32. Eckert, T.: Simple gLBF validation tool (2021). https://github.com/network2030/glbf-validation

Toerless Eckert is a Distinguished Engineer at Futurewei in Santa Clara, California. He has more than 30 years experience working for research organizations and network equipment vendors in roles from planning, architecture, development, deployment and standardization for technologies including IP, Multicast, MPLS, multimedia solutions and secure and autonomous networks.

Alexander Clemm is a Distinguished Engineer at Futurewei in Santa Clara, California. He has been involved in networking software and management technology throughout his career, most recently in the areas of high-precision networks and future networking services, green networking, network analytics, intent-based networking, service assurance, and telemetry.

Stewart Bryant is a Visiting Professor at the University of Surrey with an international reputation for the invention, specification and standardization of new telecommunications technologies. He has filed over 80 patents, written over 30 Internet RFCs (specifications) and held a number of leadership positions in technical organizations including Routing Area Director for the IETF.