High-Precision Latency Forwarding over Packet-Programmable Networks

Alexander Clemm, Toerless Eckert
Futurewei
Santa Clara, California/USA
{alex | toerless.eckert}@futurewei.com

***Abstract*—New emerging network applications such as Industry 4.0, haptic applications, and holographic-type communications will require support for stringent end-to-end latency objectives with precisely specified lower and upper bounds. These requirements will push current best-effort Internet technology and QoS mechanisms to their limit. This paper presents a new approach to achieve high-precision latency objectives, carrying latency objectives as part of packet metadata and taking differentiated actions at network nodes depending on a packet's remaining latency budget, its destination, and the latency encountered so far. A Proof-of-Concept has been developed using BPP, a novel programmable networking framework.**

Keywords—High precision networking, latency, SLOs, QoS, LBF, BPP, New IP

I. INTRODUCTION

Many future networking applications will place high demands on service levels that networks must deliver, specifically precise end-to-end latency objectives that must be met [1][2].

For example: (1) Industry 4.0 applications include industrial and robotic automation, in which robotic machinery is controlled remotely. This requires very precise synchronization and spacing of control and telemetry data with near-deterministic latency in which packets not only do not exceed a certain latency but are also not delivered sooner. (2) Haptic applications of the Tactile Internet require end-to-end networking latencies with well-defined upper bounds on the order of 5 ms or less in order to provide the illusion of remotely "touching" something. Without it, the ability to remotely operate machinery on the basis of haptic feedback is lost. (3) Holographic-Type Communications (HTC) and VR applications all require the ability to rapidly adapt streamed contents based on changes in user position or viewing angle. Well-defined network latency is required to guarantee certain levels of user experience.

In many of those cases, simply optimizing a network to minimize latency and then performing measurements to determine what latency is actually delivered is no longer sufficient. Instead, there is a need to be able to quantify a specific target latency. This can mean a latency that is not to be exceeded ("in-time guarantee"). In some cases, it can even mean a specific target latency window ("on-time guarantee"), which specifies a minimum latency in addition to a latency that must not be exceeded. Examples include applications that require fairness (e.g. for gaming or for trading) and/or whose endpoints do not possess sufficient buffer memory to buffer packets for later playout that are being received early. Specifying a target minimum latency can serve as an equalizer for cases where traffic may flow across paths of different lengths.

Current Internet technology is based on the Best Effort principle. QoS mechanisms provide for the ability to prioritize traffic, reserve resources, and apply admission control in order to provide better service levels for applications that need it. However, while these mechanisms allow for optimizations of latency, they do not change the fact that fundamentally end-to-end latency still needs to be accepted as delivered by the network, as opposed to have the network deliver on a given end-to-end latency as required by an application. Also, existing mechanisms that attempt to optimize latency typically focus on constant bit rate (CBR)-type of applications and do not accommodate variable bit rates (VBR) without incurring significant waste in terms of reserved resources, and do not support delivery of packets that is ensured to be no sooner than a given minimum latency.

In order to address this gap, we propose a novel approach called Latency-Based Forwarding (LBF). This approach is based on the idea that for any given packet with a given end-to-end latency objective, a networking device takes a differentiated action that can "slow down" or "speed up" the packet as needed. To deliver this functionality, each node that receives a packet computes a local latency budget based on the end-to-end latency objective, on the latency that was incurred by the packet up to this point, and on the expectation of the remaining path latency and number of hops for the packet's destination. The local queuing / forwarding decision is then based on this budget. Latency metadata, including the latency objective, can be carried as part of the packet itself. In cases where a requested latency is physically impossible to meet, packets can be dropped early to reduce congestion, instead of forwarding them all the way to the destination only to have them dropped there because they are late. In addition, the network can provide exceeded-latency notifications.

The advantages of LBF include the fact that it can be used to support both "in-time" (not to exceed) and "on-time" (i.e. right on target) latency objectives, that it does not require admission control, and that it supports applications with variable bit and packet rates. In addition, LBF is very elastic in

the sense that it is able to react to unforeseen variations in latency, adjusting per-hop latency as needed along the path. Finally, LBF allows stateless forwarding, in which the forwarders do not need to maintain per-flow-state, unlike other solutions that require awareness of per-flow awareness on forwarders, which may impact scalability.

Instead of developing a new custom protocol, we have implemented a Proof-of-Concept of our approach using Big Packet Protocol (BPP) [3], a novel network programming framework that allows to program custom flow behavior using metadata and commands that are carried as part of packets themselves. This allows us to not only prove the viability of LBF, but to demonstrate that the network programming framework can indeed be used to support applications with real-time networking demands without needing to deploy custom software in networking devices or even controllers.

The remainder of this paper is structured as follows: Section 2 provides an overview of related work. Section 3 provides brief background on BPP, as required for the understanding of this paper. Section 4 describes the LBF algorithm in greater detail. Section 5 presents a brief assessment of LBF based on a PoC implementation. Section 6 offers an outlook for further work and concludes the paper.

II. Related work

IETF has defined two complementary QoS architectures in order to facilitate delivery of networking services whose QoS requires certain guarantees. The first QoS architecture, DiffServ, is a multiplexing technique that is used to manage resources such as bandwidth and queuing buffers between different classes of traffic, including IntServ-style admission-controlled traffic as well as other traffic, e.g. traffic that is subject to congestion control. The second QoS architecture, IntServ, includes two services: Controlled Load Service [4] and Guaranteed Service (GS) [5]. The latter is of particular relevance here and provides per-flow fixed bandwidth guarantees based on the concept of reserving resources in advance for a given flow. GS traffic is shaped at the ingress network edge as necessary so the flow does not consume more resources than have been reserved. To support latency guarantees, flows need to be re-shaped on every hop as collisions and resource contention between packets could occur, which in turn might lead to the possibility of loss and unpredictable variations in latency. GS does not support flows to dynamically adjust the bitrate and does not offer mechanisms to slow down packets based on their desired minimum latency. Furthermore, queueing does not prioritize packets on their desired end-to-end latency.

More recently, the IETF DetNet Working Group has proposed the Deterministic Networking Architecture (DetNet) [6]. The DetNet architecture intends to provide per-flow service guarantees in terms of maximum end-to-end latency (called bounded delay in DetNet) and bounded jitter, as well as packet loss ratio and upper bounds on out-of-order packet delivery. The most fundamental limitation of DetNet, similar to GS, is in its targeted scope of CBR reservations, whereas many future applications may have highly variable bitrates. Lower latency bounds, as required for on-time services, are also not directly supported in DetNet.

Time-Sensitive Networking (TSN) [7] provides in effect an Ethernet Layer 2 variation of IntServ with two enhancements: Cyclic queueing allows for traffic shaping and avoids the need for time synchronization. Frame Replication and Elimination for Reliability (FRER) introduces 1:n path protection, sending replicated packets across disjoint paths. Replicas are eliminated on egress and latency is minimized by delivering the first instance that is received. TSN is geared towards short distances and not routing capable. Like IntServ, TSN is furthermore geared towards CBR, not variable-rate traffic and does not support slowdown of packets based on minimum required latency.

III. Background: BPP

Big Packet Protocol (BPP) is part of a new framework for packet-based networking, called New IP, centered around a new data plane protocol and the framework to support it. BPP allows packets to provide guidance to network nodes for how to handle the packet and its flow, carrying metadata and commands for this purpose. This provides network operators and edge applications with greater control of packet and flow behavior, as needed (for example) by high-precision networking applications. (Other aspects of New IP include support for variable addressing schemes and facilities to structure payload, but those aspects are of no further relevance here.) BPP does not commit to a particular "host protocol" and can be used, for example, at Layer 3 with IPv6 or IPv4. This allows BPP to be seamlessly inserted into existing network technology. In the following, those aspects that are needed for to understand this paper are summarized; please refer to [3] for more details.

At BPP's core is the concept of a *BPP Block* that contains metadata and commands for how to handle a packet and its flow. For example, in our case, we use metadata to carry the objectives for minimum and maximum end-to-end latency, and a command to queue the packet depending on the latency budget. (More on that command later.) Commands are restricted in their scope to the packet and its associated flow. This allows to define custom behavior for that particular packet or flow without altering network nodes or affecting behavior of other flows. BPP Blocks can be injected or stripped off at the edge of a network, which allows packet and flow behavior to be programmed from the edge without needing to modify core network and control infrastructure. While BPP's vision is to expand to multi-domain scenarios and end hosts, initially only single network domains are supported. This way, the need for additional mechanisms for authentication, authorization, and tamper-proofing of commands and metadata data carried in BPP Blocks can be avoided.

In some respect, BPP programming is comparable to serverless computing and Lambda functions in the cloud world [8]: Lambda provides a model in which users can request the execution of a function on a set of data without concern for how to manage the underlying virtual compute infrastructure, for how to dimension VMs, or even for basic lifecycle concepts such as when to spin up or release containers. This frees users

up to focus on the needs of their application without introducing a set of second-order problems related to dealing with underlying infrastructure, even when that infrastructure is cloudified and provided as a service. BPP programming introduces similar concepts to the networking world. It allows users to customize behavior and manage their packets and flows without needing to worry about programming SDN control functionality or orchestrating Virtual Network Functions or Service Function Chains. At the same time, BPP ensures that any such customization or programming is restricted to a particular flow, isolated from other users and the network provider infrastructure at large.

IV. Latency-Based Forwarding

LBF is based on a distributed algorithm that is performed on intermediate hops as a packet traverses the network. The fundamental idea is to have intermediate nodes assess whether the packet is on track to meet its latency objective, then take corresponding action. If the packet is ahead and at risk of arriving too early, the packet is slowed down. If the packet is behind, it needs to be sped up. To do so, each node implements the following algorithm (Fig. 1):

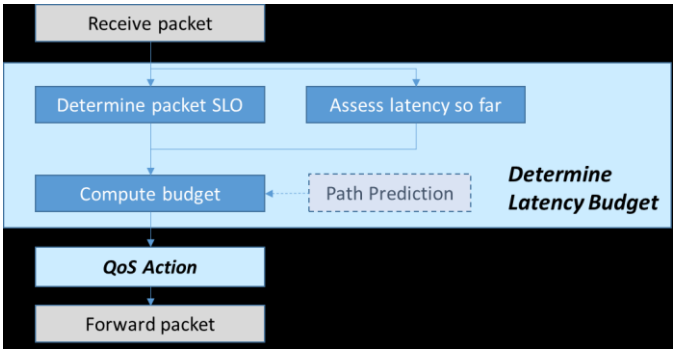

Fig. 1. Latency-Based Forwarding – node algorithm

On receipt of the packet, first the packet's local latency budget is determined. The latency budget concerns the amount of latency the packet can or should spend at the node in order for meet the required end-to-end latency.

Once the local latency budget has been determined, a proper QoS action is chosen: the packet needs to be buffered and scheduled it in a way that matches that budget and that also takes competing traffic into account.

If it is determined that a packet is so far behind that it will not be possible for the packet to arrive at its destination within the required latency window, the packet can be dropped early, instead of each node continuing to forward the packet all the way across the network only to arrive late and be discarded by the receiver. This way, congestion and network load are reduced, which helps increase the chance that service level objectives for other flows are adhered to. Alternatively, the packet could also be marked as exceeding its requested latency to alert receiver as well as monitoring applications.

The following subsections will discuss the algorithm to determine the latency budget and the QoS action taken in greater detail. In addition, the mapping to BPP will be presented.

A. Latency Budget Determination

A packet's local latency budget on a node is in effect determined by three factors: (1) the packet's Service Level Objective (SLO), i.e. the required end-to-end latency, (2) the latency that the packet has encountered up to this point, and (3) information about the path to the destination (e.g., how many more nodes will be encountered and expectation for the latency that will be incurred from here on out).

To determine (1), the SLO could be signaled (e.g. via a controller) or it could be carried as metadata within the packet itself. Since in our case we are using BPP as the underlying framework, the SLO parameters (i.e.: the lower and upper bounds of the end-to-end latency) can simply be carried as metadata parameters inside the BPP Block. Additional optimizations are possible and will be briefly discussed in section IV.D.

In order to determine (2) (the latency that the packet has encountered up to this point), different options exist.

The first option involves adding a time stamp to the packet when it is initially sent, carrying it as metadata inside the BPP Block. The node can then subtract that time from the time at which the packet is received to determine the latency. This requires a network with time synchronization.

A second option combines steps (1) and (2) by not determining latency and end-to-end SLO separately, but by computing a "rest-SLO" for the remainder of the path from the current hop to the destination. This is basically done by subtracting the latency that was incurred so far from the SLO: A node can measure the time that the packet spends being processed and queued within the node; this is referred to as "transit delay" [9]. Also, the link latency between adjacent hops (e.g. from the current node to the next hops) can be known by a node a priori: it could be measured using a separate method, or it could even be provisioned or communicated to the node via a separate control protocol, e.g. IGP extended with a parameter for link latency. The transit delay and the link latency are then subtracted from the latency SLO bounds prior to forwarding the packet. As a result, the rest-SLO that is carried in the packet reflects the required latency from the next hop to the destination.

A third option, which is the option that we chose for our Proof-of-Concept, concerns carrying a separate metadata item as part of the packet that indicates the latency encountered so far. In that case, the link latency and the local transit delay are added at each hop to the packet latency metadata item as the packet is being forwarded. The original end-to-end SLO is not modified and still carried as a whole. This adds a piece of metadata but keeps the original SLO intact. This makes the method more generally applicable for future uses that may involve other service level parameters than simple latency bounds, for which computation of the rest SLO might involve greater complexity than simple arithmetic.

Regarding (3), information of interest about the path from the node to the destination includes the number of remaining

hops as well as the path's minimal latency (consisting of link delays and basic node processing delays without buffering or queuing delays) to the destination. This information can be attached to the destination's Forwarding Information Base (FIB) entry at the node and be provisioned using a controller. It could also be signaled to the node using IGP extensions that include e.g. link latencies. Knowing the expected minimal latency, while useful, is in fact not required and a budget can be determined without it using the number of hops. In that case, budgets need to be determined with a little wider tolerance level to account for any uncertainty.

Using this information, a latency budget can now be computed as follows: Subtract the latency encountered so far from the end-to-end SLO to determine the "rest-SLO", i.e. the target latency of the remainder of the path. From that rest-SLO, subtract the expected minimal latency to the destination. This provides the "remainder budget" that needs to be allocated across the remaining hops. One way to allocate the node's latency budget is to simply divide the remainder budget by the number of remaining hops on the path (including the node itself).

For example, assume that a node receives a packet with an end-to-end SLO of 70ms ±10 ms latency (i.e. minimum latency 60ms, maximum latency 80ms). The latency that was encountered up to this point is 30 ms. The remaining path includes 4 nodes: this node and 3 hops further downstream. The minimal latency towards the destination is 20 ms. In this case, the budget would be determined as follows: The rest SLO is 40 ±10 ms latency. Subtracting from this the 20 ms minimal latency results in a remainder budget of 20 ±10 ms. Dividing this by the number of hops on the path results in a node budget of 5 ±2.5 ms. 5 ms would be the midpoint which would be targeted in the subsequent QoS action.

A second example is depicted in Fig. 2. In that example, the path includes 3 hops, with link latencies of 500 usec and an end-to-end SLO of 7 ±1 ms. The example shows how small differences between latency budget and actual queuing latency achieved by one node (e.g. the first hop 1: 2ms budget vs actual latency of 2.3ms) are automatically adjusted by downstream nodes.

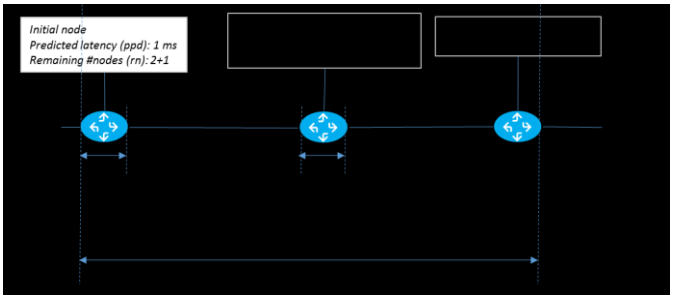

Fig. 2. Latency-Based Forwarding – Example

Of course, variations of the algorithm for how to determine a node's latency budget are possible. For example, one refinement might account for the distance from the destination and err on the side of sending a packet early, especially when the number of remaining hops is large, as it becomes difficult to make up time once it is lost. In the earlier (first) example, instead of targeting 5 ms as local queueing delay, the node might in that case skew the decision towards sending the packet earlier, e.g. 2.5 ms (the lower end of the range). At the same time the packet should not be sent too early, as that will make it difficult to buffer packets long enough at nodes further downstream in case a lower bound on end-to-end latency must be observed. In case the minimal latency from the current node to the destination is not known, a node could make a default assumption based on a multiple of the number of downstream hops, resulting in finer tuning of the latency the closer the destination is approached.

B. QoS Action

After determining the latency budget, a QoS action needs to be taken that will result in the packet being forward with an incurred delay that matches the local latency budget at least in approximation.

In our case, the QoS action that is applied for this purpose takes a two-stage approach. The first stage addresses any delay that needs to be incurred to comply with the budget for the lower delay bound. For this purpose, a Push-In, First Out (PIFO) queue [10] is utilized. PIFO allows to dynamically determine at which position to insert a packet into a queue at the time the packet is enqueued. Using the latency budget, the PIFO queue and the position for the packet that best match the latency budget are selected. This way, the packet will be dequeued no sooner than called for by the latency budget's lower bound. (Certain limits apply. In cases where the lower bound is very high, additional steps may need to be taken.) When the packet is dequeued from the PIFO queue, it enters a FIFO queue in the second stage, for example prioritized according to the budget for the upper delay bound.

Different strategies can be implemented to determine the priority order of packets with different but overlapping latency budgets; a comparison of those strategies will be the subject of a future paper.

It would be possible to apply other actions as well. For example, in order to slow a packet down, the packet could be forwarded over a slow path versus a fast path, or additional buffering (as far as applicable) could be applied.

C. Mapping to BPP

To implement LBF, it would be possible to devise a new custom protocol which carries the end-to-end SLO in dedicated fields. However, this would raise the question how to embed support for that new protocol across networking devices across a given network and how to use it in conjunction with other protocols. Instead, we decided to implement LBF on top of BPP, allowing us to introduce LBF without needing to define a new protocol. This decision also facilitates the integration of LBF with other use cases and functionality enabled by BPP, which could be applied simultaneously to the same packets and flows. One example concerns network-embedded Operational Flow Profiling, which provides users with operational insights into flows and facilitates network-embedded analytics of flow telemetry [11]. As BPP itself is still very new, this decision also provided us with the additional benefit to "exercise" BPP

and assess its feasibility for a use case that features demanding real-time requirements.

LBF can be mapped onto BPP as follows: packets carry their SLO as metadata inside a BPP Block to represent the minimum and maximum end-to-end latency that is acceptable for a packet. A third metadata item is defined to record and carry the latency incurred up to this point in the path. In addition, the BPP Block carries a command to match the queuing delay based on the computation of the local node delay budget. This action has 4 implicit parameters: the extended FIB entry of the destination (including the number of hops and expected latency for the path remainder), as well as the respective references to the three metadata items with the latency SLO and the latency that has been incurred so far.

The BPP pseudocode for LBF is depicted in Fig. 3. This indicates that a node processing the packet is supposed to perform command "lbf-ify" with three parameters. The first two parameters, slo-lb and slo-ub, refer to the SLO's lower and upper bounds of acceptable latency and are included as values. The third parameter points to the metadata item that carries the packet latency incurred up to to this point.

```
Cmd1:
  cond: -;
  action: lbf-ify
    (par.value("slo-lb", slo-lb),
     par.value("slo-ub", slo-ub),
     par.meta("packet-latency" offset 0 len 2));
```

Fig. 3. LBF pseudocode for BPP

The mapping to an actual BPP Block is depicted in Fig. 4. Worth noting is the fact that the block in its entirety requires 16 octets. This is the overhead imposed on packets that receive Latency-Based Forwarding service.

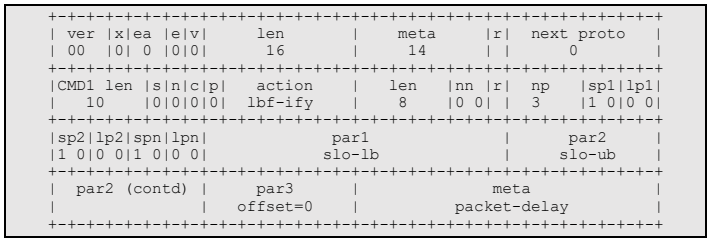

Fig. 4. Mapping LBF to BPP

One important design decision concerns the definition of a single action that combines the logic to determine the latency budget and the subsequent QoS action being taken. Other designs could utilize more generic commands. For example, an alternative design would map the logic to four commands: the first command retrieves the SLO, the second command computes the local latency budget (i.e. acceptable lower and upper bounds for local queuing delay), the third command matches the packet to the queue per the budget (queues include a nil queue to drop the packet as applicable), and the fourth command updates the packet latency metadata, adding next link delay and transit delay prior to forwarding the packet. However, this option would have increased packet overhead significantly. Using a single command facilitates performance optimization and we felt that LBF is a use case that is important enough to warrant introducing a custom command.

D. Optimizations

While the packet overhead of 16 octets for fully customizable SLOs is small, there is potential for further optimization. This stems from the observation that the SLO and the command being applied will be the same for every packet of the flow. As part of future work it is conceivable to cache both SLO and command along the path. The BPP framework actually does provide support for this by means of a set of stateful extensions revolving around the concept of statelets, described in greater detail in [11]. Statelets are cache entries that are associated with a particular flow key, analogous to entries in a flow cache.

For LBF, statelets could be used to cache SLOs and commands applied to packets of the flow. A first packet will include a command to deploy statelets along the flow. Statelets are automatically cleaned up when the flow terminates. Special considerations apply in case path changes occur. In a nutshell, when a node determines that the forwarding decision of a packet is new (it is the first packet it sees of a flow, or different from that of previous packets), the contents of the statelet is copied into a BPP Block along with a command to initialize a corresponding statelet and forwarded to the new hop. As this functionality has not yet been implemented, this remains an item for future improvement.

E. Other Aspects

An important feature of the LBF algorithm is the fact that it is fundamentally "elastic" in nature, in the sense that any unforeseen changes in delay along the path can in many cases be made up or adjusted for by later nodes. This allows the algorithm to applied also in scenarios where it is not supported by every node along the path or in scenarios where link latencies are unknown, as well as in the event of unexpected path changes or variations in link delay. Of course, the "margin of error" gets smaller as the packet nears its destination and there are less hops to make up between any discrepancy between latency experienced up to this point and end-to-end latency objectives. However, an important feature of the algorithm concerns the fact that any variations in latency precision encountered at the nodes are not aggregated along the path. Instead, inaccuracies encountered by earlier nodes will be compensated and corrected by later nodes. The precision with which the end-to-end latency can be hit is gated by the precision with which the last hop can meet its latency budget; it is independent of the actual number of nodes along the path.

In addition to being elastic, the algorithm also does not need to make "greedy" forwarding decisions in which packets of one flow are consistently prioritized higher at the expense of packets of other flows. Instead, packet priority can in effect be varied as needed along the path, just high enough to meet the latency objective. This allows to determine when competing flows can be given precedence because a packet is still well within its latency budget and can afford to be placed lower in its queue, and when a packet needs to be indeed prioritized to

avoid running late. This allows QoS strategies that allow to create a "happy medium" that take the needs of multiple flows into account simultaneously to satisfy their constraints.

As mentioned, the algorithm does allow to accommodate different types of QoS actions and buffering and scheduling strategies, as long as they fulfill the requirement of being cognizant of a local latency budget and adjusting the local decision accordingly – to forward a packet immediately with as minimal delay as possible, to slow it down, or something in between. Variations of the algorithm are possible. For example, the forwarding decision itself could be chosen on the basis of latency, forwarding some packets on slower paths that may experience less utilization than others, relieving network congestion as an additional benefit.

V. PoC and Assessment

To validate the LBF algorithm, we built a lightweight, near-real-time software implementation as a Proof-of-Concept (PoC). The PoC allows quick experimentation with different algorithms in small topologies through implementation and test runs; its code is publicly available [12].

A. PoC Implementation

The validation Proof-of-Concept (PoC) is a simple (ca. 3000 lines of code) but effective tool to validate LBF behavior and quickly innovate and experiment with different algorithms.

The implementation consists of one Linux application for the forwarder and one for the sender. The PoC runs the entire network within a single server. UDP ports are used as link-layer interfaces. UDP payloads represent the BPP packet. It consists of the required parts of IP headers to hop-by-hop forward a packet and the BPP Block to perform LBF processing. Forwarders use separate threads for each ingress and egress interface. An ingress thread receives and routes a packet to an output interface and enqueues it into that interface's LBF queue. An egress thread dequeues and sends packets, emulating outgoing interface speeds (currently 100 Mbps). By assigning threads for each interface to separate CPU cores, the code can as necessarily use busy loops (so called poll-mode) for higher timing accuracy. At the end of each test run (e.g.: 10 seconds), logs for each packet that passed through an interface are written for analysis.

The PoC uses Linux UDP sockets to pass packets. There is a small possibility that Linux introduces scheduling delays up to hundreds of usec [13]. Our implementation compensates for this by calculating LBF latency using the variant in which the BPP Block carries a sender timestamp and computing the latency by subtracting it from the current system time. The system performs at 100 Mbps linerate speed and usec accuracy of LBF operations. Changing the implementation from sockets to DPDK could eliminate the Linux scheduling issues and improve performance to Gbps speeds.

The PoC forwarder architecture is depicted in Fig. 5. The BPP-programable forwarding plane relies on data maintained in the FIB, e.g. the number of hops (tohops) to the destination and the minimum latency of the path to the destination, to compute the local latency budget, e.g. the delay that must be imposed at a minimum (qmin). The population of the data in the FIB could occur using IGP, although in the PoC it is done via simple configuration. After determination of the local latency budget, the packet is put in to the LBF queue, consisting of the two stages as discussed earlier.

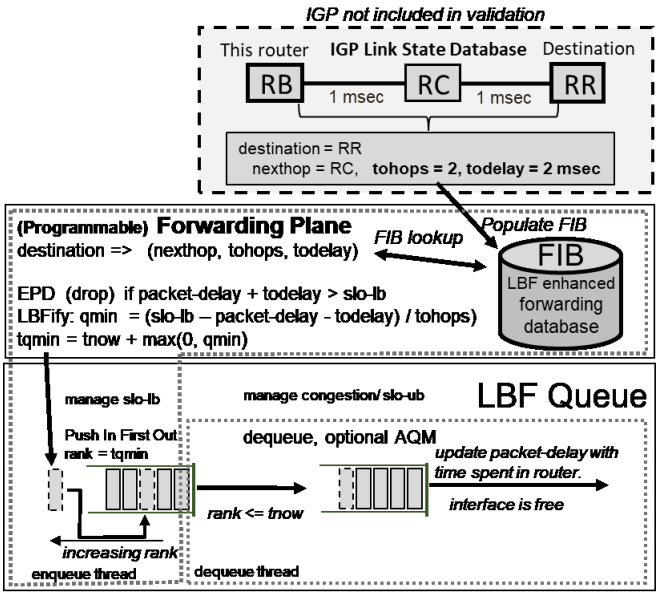

Fig. 5. Forwarder design

B. Test topology

Fig. 6 shows a topology where Flow 1 from Sender 1 passes through 4 LBF queues 1, 2, 3, 4. On each queue, it can compete with traffic from another flow that will be routed by the following forwarder to a different interface than Flow 1, so that for each hop the queue buildup can be controlled separately by setting up the competing traffic flow sender parameters. The receivers are shown with dashed lines because no actual receiver applications are started, as only the per-packet statistics generated by the forwarders are of interest.

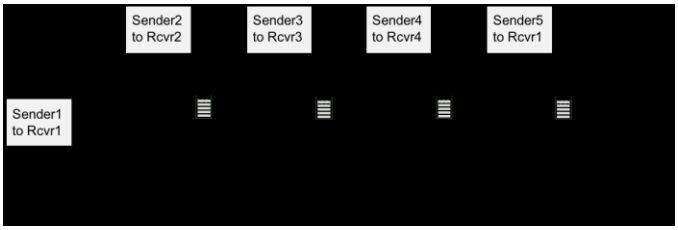

Fig. 6. Multi-hop lmin collision topology

C. Evaluation

In our evaluation, we focus specifically on the impact that the lower end-to-end latency bound (slo-lb) has on the actual end-to-end latency that is observed. To evaluate the impact of slo-lb, we generate traffic that periodically bursts, such that the flow under evaluation plus the competing flows periodically

build up and drain queues so that varying amounts of latency are introduced.

Fig. 7 shows results obtained from our baseline test, in which traffic is best effort, with the lower end-to-end latency bound (slo-lb) set to 0 and the upper end-to-end latency bound (slo-ub) set to infinite. Queue 1, 2, 3, and 4 refer to the latency observed at the queues of the first, second, third, and fourth hops, respectively. The last link to the receiver just adds the link propagation latency (100 usec) to the Queue 4 latency. Burst rate of Flow 1 is 55 Mbps. The burst rates of the competing flows are Flow 2: 57 Mbps, Flow 3: 57 Mbps, Flow 4: 59 Mbps, and Flow 5:70 Mpbs. These rates where simply chosen so that the baseline test would result in useful per-hop latency increase and to avoid inadvertent synchronization that might resul from identical competing flow rates (the validation does currently not support more randomized test flows).

When there is no competing traffic (column 1), the end-to-end latency that is observed on the receiver is just the path latency of 5 * 100 usec = 500 usec and there is no latency variation – the minimum, average, and maximum latency observed for the packets is exactly the same. As competing flows are successively added, they simply add latency and latency variation. For example, competing flow 2 causes the average latency observed at Queue 1 to be 170 usec and the maximum latency grows to 252 usec, resulting in 152 usec of latency variation. The addition of more competing flows causes the averages, maxima, and variations to grow even further. As no end-to-end latency bounds are specified, LBF does not take any adjustive action and latency, once introduced, simply stays until the packet reaches the receiver.

Best Effort Flow 1: latency min/average/max [range] all in [usec]					
	no competing traffic	competing Flow2	competing Flow 2,3	competing Flow 2, 3, 4, 5	competing Flow 2, 3, 4, 5
Queue1	100/100/100 [0]	100/170/252 [152]	100/170/252 [152]	100/170/252 [152]	100/170/252 [152]
Queue2	200/200/200 [0]	200/270/352 [152]	200/304/379 [179]	200/304/379 [179]	200/304/379 [179]
Queue3	300/300/300 [0]	300/370/452 [152]	300/404/479 [179]	300/425/522 [222]	300/425/522 [222]
Queue4	400/400/400 [0]	400/470/552 [152]	400/504/579 [179]	400/525/622 [222]	400/560/685 [285]
Receiver	500/500/500 [0]	500/570/652 [152]	500/604/679 [179]	500/626/722 [222]	500/660/785 [285]

Fig. 7. Multi hop congestion, best effort flow under test (Flow1)

Fig. 8 shows the behavior when an slo-lb of 700 usec is introduced. This is 200 usec more than the path propagation delay, causing LBF to kick in. In the absence of competing traffic, an additional 50 usec of delay is incurred at each hop, causing packets to arrive 200 usec later. However, when competing traffic is added, the additional delay that is introduced causes the latency on further hops to be reduced. This can be easiest seen when only Flow 2 is competing: the latency variation of 137 usec introduced at the first hop is effectively eliminated by the time the packet reaches the receiver, because the reduced buffering budget on the three following hops cause the latency to be reduced. When all four hops have competing traffic, the resulting latency variation is reduced as well, although the effect is less pronounced the later in the path the additional latency is incurred, as there are fewer hops left to compensate for it.

Fig. 9, finally, shows the effect of varying lower bounds in the presence of the same competing traffic. As slo-lb is increased, latency variation decreases. An explanation for that is the fact that a greater lower bound provides more room for compensation of greater variations.

LBF flow slo-lb: 700, slo-ub: infinite: latency min/average/max [range] all in [usec]					
	no competing traffic	competing Flow2	competing Flow 2,3	competing Flow 2, 3, 4, 5	competing Flow 2, 3, 4, 5
Queue1	150/150/150 [0]	150/210/287 [137]	150/210/287 [137]	150/210/287 [137]	150/210/287 [137]
Queue2	300/300/300 [0]	300/339/391 [91]	300/377/457 [157]	300/377/457 [157]	300/377/457 [157]
Queue3	450/450/450 [0]	450/469/495 [45]	450/492/557 [107]	450/522/600 [150]	450/522/600 [150]
Queue4	600/600/600 [0]	600/600/600 [0]	600/608/657 [57]	600/629/700 [100]	600/662/772 [172]
Receiver	700/700/700 [0]	700/700/700 [0]	700/708/757 [57]	700/729/800 [100]	700/762/872 [172]

Fig. 8. Multi-hop congestion, LBF flow under test (Flow 1) slo-lb 700

Competing Flow 2,3,4,5, LBF flow slo-lb increasing, slo-ub: infinite: latency min/average/max [range] all in [usec]					
	LBF flow slo-lb 700	LBF flow slo-lb 800	LBF flow slo-lb 900	LBF flow slo-lb 1000	LBF flow slo-lb 1100
Queue1					
Queue2					
Queue3					
Queue4					
Receiver					

Fig. 9. Multi-hop-congestion, LBF flow under test (Flow1) increasing slo-lb

VI. Summary and Conclusion

One of the challenges that future networking applications are faced with concerns the ability to support high-precision services that adhere to stringent service level objectives. In many cases, this includes quantifiable end-to-end latency objectives that include not just latency that is not be exceeded, but also lower bounds to latency before which packets should not be delivered.

Latency-Based Forwarding is a new approach to ensure packet delivery according to end-to-end latency objectives, based on the idea to make packets "latency-aware" and let networking device take differentiated action to slow down or speed up packets as needed, based on their SLO, latency incurred so far, and expectation of remaining path latency. LBF exhibits interesting properties in that it is elastic and can react to unforeseen variations in latency. LBF also provides an excellent use case for BPP, demonstrating its applicability to a real-time application and obviating the need for development of a custom protocol. It thus complements applications for BPP that have been proposed for other domains, from operational flow profiling for better management insights [11] to task optimization in Mobile Edge Computing [14].

The algorithm presented here, while simple, is in fact very effective. It also demonstrates that the abstraction of signaling the desired packet SLO in the data plane is a rich territory for exploration for high precision communications that goes way beyond the goal of simply reducing "bufferbloat" [15] that has for decades been at the center of attention.

Going forward, we are planning to experiment with additional QoS strategies to manage the local latency incurred by packets inside a device and to mitigate between competing flows with overlapping delay bounds. The impact of different QoS actions applied at the node as part of the algorithm on other aspects, such as fairness, is an interesting topic that we intend to explore in a future paper. We are also looking to expand our experiments to bigger and non-virtual topologies and more varied test traffic, and are looking into the possibility

of developing time-discrete simulations to complement the PoC. Last not least, we plan to expand the underlying BPP framework to improve its programmability and facilitate experimentation with other algorithms.

VII. Acknowledgments

We would like to express our thanks to Asit Chakraborti for his contributions to the validation prototype, to Uma Chunduri and Padma Pillay-Esnault for numerous stimulating technical discussions, and to Lijun Dong, Kiran Makhijani, and Richard Li for comments on an earlier version of the manuscript.

References

[1] "Network 2030 – A Blueprint of Technology, Applications and Markeet Drivers Towards the Year 2030 and Beyond." White Paper, accessible at https://extranet.itu.int/sites/itu-t/focusgroups/net-2030/SitePages/White%20Paper.aspx, ITU-T FG-NET-2030, May 2019.

[2] ITU-T FG-NET2030: "New Services and Capabilities for Network 2030: Description, Technical Gap and Performance Target Analysis". FG-NET2030 document NET2030-O-027, November 2019.

[3] R. Li, A. Clemm, U. Chunduri, L. Dong, K. Makhijani: "A New Framework and Protocol for Future Networking Applications." ACM SIGCOMM Workshop on Networking for Emerging Applications and Technologies (NEAT), Budapest, Hungary, August 2018.

[4] J. Wroclawski: "Specification of the Controlled-Load Network Element Service." RFC 2211, IETF, September 1997.

[5] S. Shenker, C. Partridge, R. Guerin: "Specification of Guaranteed Quality of Service." RFC 2212, IETF, September 1997.

[6] N. Finn, P. Thubert, B. Varga and J. Farkas: "Deterministic Networking Architecture (DetNet)." Internet Draft draft-ietf-detnet-architecture-13, IETF, May 2019.

[7] https://1.ieee802.org/tsn/, last accessed 23 December 2019

[8] T. Lynn, P. Rosati, A. Lejeune, V. Emeakaroha: "A Preliminary Review of Enterprise Serverless Cloud Computing (Function-as-a-Service) Platforms." IEEE 9th Intl. Conference on Cloud Computing Technology and Science, December 2017.

[9] F. Brockners, S. Bhandari, C. Pignataro, H. Gredler, J. Leddy, S. Youell, T. Mizrahi, D. Mozes, P. Lapukhov, R. Chang, D. Bernier, J. Lemon: "Data Fields for In-situ OAM." Internet Draft draft-ietf-ippm-ioam-data-06, IETF, July 2019.

[10] A. Sivaraman, S. Subramanian, M. Alizadeh, S. Chole, S. Chuang, A. Agrawal, H. Galakrishnan, T. Edsall, S. Katti, N. McKeown:" Programmable Packet Scheduling at Line Rate". ACM SIGCOMM, Florianapolis, Brazil, August 2016.

[11] A. Clemm, U. Chunduri: "Network-Programmable Operational Flow Profiling." IEEE Communications Magazine Vol. 57 No. 7, July 2019.

[12] https://github.com/network2030/lbf-poc

[13] A. Toussaint, M. Hawari, T. Clausen: "Chasing Linux Jitter Sources for Unncompressed Video." IEEE CNSM Workshop on High-Precision Networks Operations and Control (HiPNet 2018), Rome, Italy, November 2018.

[14] L. Dong, R. Li, "Distributed Mechanism for Computation Offloading Task Routing in Mobile Edge Cloud Network," IEEE ICNC 2019.

[15] J. Gettys: "Bufferbloat: Dark Buffers in the Internet." IEEE Internet Computing Vol. 15 No.3, May 2011.