ASAP: Preventing Starvation in Backpressure Forwarding

Laura Poplawski Ma, Samuel Nelson, Gregory Lauer, Stephen Zabele
Raytheon BBN Technologies, Cambridge MA, USA
{laura.ma, samuel.nelson, greg.lauer, steve.zabele}@raytheon.com

***Abstract*—Backpressure forwarding maximizes throughput in dynamic networks. It does not, however, provide any latency guarantees and can result in indefinitely long queuing delays. ASAP (Anti-Starvation with Artificial Packets) solves this starvation problem by: (1) dynamically detecting starvation based on outbound capacity estimates, (2) using virtual packets to convert queuing delay into queue-based gradient increases, and (3) non-linearly add virtual packets after starvation is detected to rapidly respond. ASAP was implemented into a backpressure forwarding system called IRON. Experimental results show that ASAP effectively detects and rapidly addresses starvation when it is occurring, and does no harm when starvation is not occurring.**

I. Introduction

Backpressure forwarding is an approach to dynamic networks that maximizes network throughput [1]. Founded in queuing theory and stochastic network optimization, backpressure forwarding makes local decisions based on queue depths to decide when and where to send packets. Each node maintains a separate queue for each destination, and the queue depths are shared only with immediate neighbors. The node computes the difference (known as the *gradient*) between each local queue depth and the depth of the corresponding queue at each neighbor. Whenever a link is available to transmit a packet, the node chooses a packet from the destination-based queue with the largest gradient. The benefits of this approach include maximizing throughput, vastly simplifying routing decisions, and decreasing global or wide-spread control traffic imposed by traditional routing algorithms.

While backpressure maximizes throughput, it does not attempt to minimize latency. In fact, it is often the case that latency suffers due to slow gradient build up and long indirect paths. This is most evident when there are competing flows of different rates traveling through common nodes. High rate flows can more rapidly increase and maintain the gradient between neighboring nodes, causing packets from low rate flows to be starved or served much more slowly. While gradient pressure from long-lasting flows will eventually build, latency can be severely impacted and the final packets of flows may be starved indefinitely [2]. This is particularly problematic with TCP flows, since acknowledgments must be received before additional packets are sent, so long queues cannot accumulate.

This material is based upon work supported by DARPA under Contract No. HR0011-15-C-0097. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of DARPA. Distribution Statement "A" (Approved for Public Release, Distribution Unlimited). Approved for Public Release, cleared by DARPA on May 2, 2018, DISTAR Case No. 29390.

Addressing starvation in backpressure forwarding is challenging because the optimization problem being solved is maximizing *network throughput*, and starvation of individual flows is not considered if it does not affect network throughput. Since starvation is manifested as increased queuing delay, including queuing delay into the backpressure algorithm is a natural approach, and has been considered [3], [2], [4]. Always incorporating delay into the gradient works well when the network can support all admitted traffic. However, the use of a delay term has not been previously addressed when backpressure is paired with admission control based on queue depths (i.e., when the offered traffic load cannot all be supported). We have shown that simply incorporating the delay into the gradient computation can unfairly bias the rate of flows when using admission control based on queue depths, such as the log-based admission control introduced in [5]. (This is illustrated in detail in Section II).

We address the starvation problem in the presence of admission control with the following insight: *gradient pressure, based only on queue depth, should rapidly increase if a starvation condition is detected.* To this end, we have developed the Anti-Starvation with Artificial Packets (ASAP) algorithm, which detects starvation and responds by adding virtual packets to the queue in a non-linear fashion based on queuing delay. Virtual packets are included in the queue depths like any other packet, but they are dequeued only when no real packets exist for a destination, and they are never transmitted. We have implemented ASAP into the IRON system [6], developed as part of a DARPA-funded effort, which utilizes backpressure forwarding with admission control to provide a highly survivable and robust network. There are three novel aspects of ASAP that allow it to effectively address starvation without disrupting queue-based admission control:

1) A capacity-based dynamic starvation detection mechanism prevents ASAP from inflating the gradient when queue delay is a response to network conditions rather than to backpressure-induced starvation
2) The addition of virtual packets during a detected starvation condition, instead of directly adding a delay term to the gradient computation, allows the gradient to gracefully increase to counter starvation, without requiring additional control traffic between nodes, and decrease when the starvation condition passes, preserving the inflated gradient even when the queue does not instantaneously contain packets for the starved destination
3) A non-linear function maps long queue delays during

a starvation condition to a quantity of virtual packets added to the queue, allowing for rapid response

The remainder of this paper is as follows. Section II discusses the starvation problem in depth, including existing work. Section III presents the ASAP algorithm, and discusses in detail the three major components. Section IV presents results from the IRON system running ASAP on real hardware, showing that it effectively detects and responds to starvation when starvation is truly occurring, and does no harm when starvation is not occurring. Finally, Section V concludes and presents future directions.

II. Starvation in Backpressure Routing

Due to the focus on network throughput, backpressure forwarding algorithms based only on queue depths can result in large latencies and starvation of flows, including the *last packet* problem [2]. There have been recent advances in starvation avoidance, many of which incorporate queue delay directly into the gradient computation, either in place of queue depths or in conjunction with queue depths.

[2] replaces queue depth with a function of head-of-line queue delay, but only operates with predetermined routes. [3] uses a delay-based Lyapunov function for scheduling, but only applies to single hop networks. [4] uses a combination of delay and queue depth for dynamic multi-hop backpressure forwarding and proves that starvation (including last-packet starvation) cannot occur. However, their result assumes the admitted traffic is within the feasible region for the network, and thus does not take admission control into account. Furthermore, without any discussion of admission control, it cannot address utility maximization. We discuss an example in Section II-A where the algorithm from [4] results in poor network utility under the admission control algorithm from [5].

In this work, we address anti-starvation in the presence of queue-based admission control by including starvation detection into our anti-starvation algorithm, and including a delay term that is based on the degree of starvation. The backpressure gradients for scheduling/forwarding and queue-based admission control will simply use the queue depth *unless* a starvation condition is present for the queue in question. This allows our algorithm to interact correctly with existing admission control algorithms. Furthermore, we significantly simplify the implementation by accounting for head-of-line queuing delay via the addition of virtual packets to the queues, rather than attempting to exchange delay measurements (along with all values necessary to detect starvation) with neighbors. This also allows different nodes to independently determine whether or not a destination is being starved and the relative importance of anti-starvation vs other flow utility functions, and it automatically preserves the starvation-avoidance state (with a natural decay mechanism) after the queue is empty, and thus works well for intermittent flows as well as last packets.

A. Case for Non-linear Delay Term

Simply adding a term linearly proportional to the delay into the gradient computation can result in unfair admission between flows when the latency of those flows are rightfully different. To illustrate this, consider the "Y-shaped" topology in Figure 1. In this scenario, there are three high-rate flows

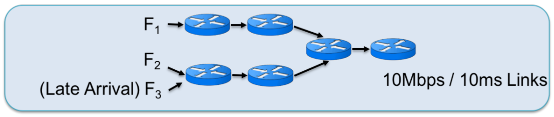


Fig. 1: Y-shaped topology with three flows, resulting in asymmetric latency due to asymmetric queue delay.

all destined to the right-most node. During the first half of the scenario, flows F_1 and F_2 start. Half-way through, F_3 starts.

We can compute the expected latency of the flows as follows. Let $L(f)$ be the latency for a flow f, and Q be the set of all queues on the path f traverses. Let $d(q)$ be the queue depth and $r(q)$ be the drain rate of a queue $q \in Q$. Let $l(q, f)$ be the link transmission delay of a packet in f that has just left queue q. In this scenario, each flow traverses three queues (the end queue is not considered) and three links. The latency of a flow f can be computed as:

$$L(f) = \sum_{q \in Q} \left(\frac{d(q)}{r(q)} + l(q, f) \right)$$

Backpressure forwarding will ensure all queues have the same depth, which in this experiment is 20KB (corresponding to roughly 20 packets) in a fully loaded, stabilized network. [1] Prior to F_3 starting, both F_1 and F_2 have the same expected end-to-end latency. Experimental results from running this experiment in IRON without ASAP show that each flow is able to achieve a throughput of 4.2Mbps after accounting for overhead. Hence the drain rate of the first two queues on the path is 4.2Mbps, and the drain rate of the bottleneck queue is 8.4Mbps. The expected end-to-end latency of these flows is therefore 125ms when all links have 10ms propagation delay.

After F_3 starts, experimental results show that all three flows have the same throughput of around 2.8Mbps, which is expected since the flows have the same priority. Hence the first two queues servicing F_1 have a drain rate of 2.8Mbps and the first two queues servicing the other flows have a drain rate of 5.6Mbps. The bottleneck flow has a drain rate of 8.4Mbps. Plugging these values into the equation, we expect the end-to-end latency of F_1 to be 160ms and the end-to-end latency of both F_2 and F_3 to be 100ms. These expected results match the experimental results, shown in Figure 2.

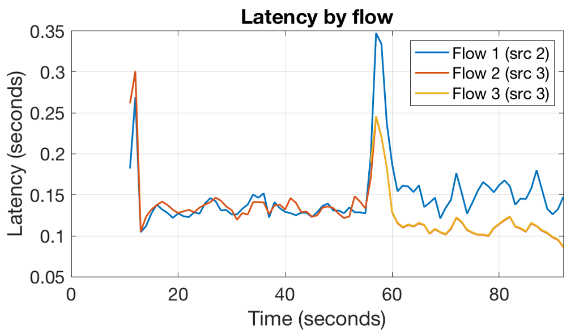


Fig. 2: Experimental results showing latency asymmetry, which would lead to unfair prioritization if using a linear delay term.

[1]We reduce the queue depths in our backpressure forwarding network packets by including placeholder bits to artificially increase the advertised queue depths. Similar techniques are described in previous work such as [7].

Thus, if the queuing delay were incorporated into admission control computation when starvation is not occurring, F_1 would have an unfair disadvantage, since F_1 is expected to have a higher per-packet delay, which would lead to a correspondingly higher value passed into the admission control function, and thus a lower admission rate. Section IV-C shows goodput results for this example when using a linear delay term.

For this reason, it is important to first detect whether or not starvation is occurring before responding to it. Delay should not be considered in the gradient unless there is a high probability that starvation is truly occurring, or natural asymmetry in latency can result in unfair treatment of flows.

III. Anti-Starvation Algorithm

ASAP effectively solves the starvation problem in backpressure forwarding by dynamically detecting starvation and adding virtual packets to artificially increase the queue depths. These virtual packets are dequeued last to ensure they do not remove available capacity from ongoing flows. Starvation is both detected and responded to by *head-of-line queue delay*. We define this delay as the amount of time the packet at the head of the queue has been at the head of the queue. Note that this is different than the amount of time the packet at the head of the queue has been in the queue. With this definition, the sum of head-of-line queue delay over approximately one queue-depth worth of packets is a measure of the total queue delay amortized over the individual packets.

Starvation detection is based on monitoring the head-of-line queue delay. When this delay surpasses a threshold, the starvation reaction component of ASAP is triggered, and virtual packets will be incrementally added to increase the gradient, with the number of virtual packets computed based on the continually increasing delay for the packet. When the packet is sent, no additional virtual packets will be added until/unless the next packet surpasses the threshold, which is less likely because the virtual packets remain in the queue.

A. Starvation Detection

A primary goal of ASAP is to do no harm when starvation is not occurring. Therefore, ASAP does not attempt to take action until starvation is detected. In order to be effect at detecting starvation under a wide range of network characteristics, ASAP computes a starvation threshold dynamically using estimated capacity, number of local queues, and maximum expected packet size.

In particular, ASAP estimates the maximum expected head-of-line queue delay for a packet. This is the serialization time of the packet onto the link times the number of queues the backpressure router must cycle through (since it's possible for multiple queues to use the same link as a next hop).

Let M be the max expected packet size in bits, Q be the number of queues the backpressure router is servicing (one per destination), and c be the expected capacity in bits/sec. We note that IRON provides capacity estimates in real-time to ASAP and specifies a maximum packet size of 1500 bytes. Let α be a "safety factor" which forces ASAP to be cautious before taking anti-starvation action, buffering any temporary queue processing slow downs or ordering issues. This is set to 50 in the ASAP system, which was based on experimental results. We can then compute $s(M,c)$ as the starvation threshold in milliseconds (note the 1000 multiplier) as follows:

$$s_\alpha(M,Q,c) = 1000 \cdot \alpha \cdot Q \cdot \frac{M}{c}$$

Once the head-of-line queue delay for a packet has crossed this threshold, ASAP will trigger the starvation reaction algorithm. To provide an general idea of this threshold, in the triangle topology discussed in Section IV, with 10Mbps links, the threshold is around 100ms.

B. Starvation Reaction

Once starvation is detected, ASAP responds by using virtual packets to artificially increase the gradient. At the heart of this algorithm is a function that converts queue delay into virtual packets (more correctly, virtual bytes). There are three major benefits to adding virtual packets in this way: (1) the delay-to-bytes function provides fine-grained control of the gradient and allows for rapid starvation relief, (2) by adding virtual packets to the queue, rather than including a delay term in the gradient computation, ASAP dramatically simplifies the implementation and retains the low overhead "queue depth only" spirit of backpressure forwarding, and (3) virtual packets allow the artificially-increased gradient to remain in place even after a starved packets is transmitted, allowing future packets for the same destination to be transmitted more quickly.

ASAP uses a non-linear delay-to-bytes function, which rapidly increases as the delay increases. This increase ensures that starvation is alleviated quickly, since many messages, such as TCP control messages, must be delivered in a timely fashion for the protocol to function properly. Let d be the head-of-line queue delay, and $f(d)$ be the virtual bytes to add to the queue. Let a be a scaling constant. ASAP defines $f(d)$ as follows, though we conjecture that this function ought to remain flexible, and can even be defined differently at different nodes, in order to allow an operationally-appropriate trade-off between anti-starvation and throughput-optimality.

$$f(d) = \begin{cases} 0, & \text{for } d < s_\alpha(M,Q,c) \\ Min(P, a \cdot d^2), & \text{for } d \geq s_\alpha(M,Q,c) \end{cases}$$

Note that d is in ms and a is in (bytes / ms^2). This allows the result of $f(d)$ to be in bytes. The a coefficient allows for scaling; in the ASAP system we set $a = 2$ based on experimental results. These bytes are not added all at once; instead, they are incrementally added as d increases. Every ASAP action interval (5 ms in our experiments), $f(d)$ is computed, any virtual bytes already added since the most recent dequeue are subtracted from $f(d)$, and ASAP adds the difference as new virtual bytes. In other words, at any point in time, a total of $f(d)$ will have been added to address the starved packet in question.

P is a cap on the number of virtual bytes to add, based on all computed local gradients. This is to prevent unbounded growth, which is particularly important if the threshold is high causing a large amount of virtual bytes to be immediately added when the threshold is crossed. P is computed such that it will make the starved gradient slightly higher (5% in our experiments) than all other local gradients, which is the maximum needed to unstarve the packet.

Once a queue is built up with virtual packets, these packets remain in the queue until they become the BPF-defined best choice for transmission. In other words, the queue depth will remain inflated until it becomes the highest gradient. At that point, virtual packets will be dequeued until that queue depth no longer has the largest gradient. If the system reaches an equilibrium state, these virtual packets cause the queue depth to stabilize at exactly the threshold where packets are not sent, but where any additional packet added to the queue will be sent quickly. Therefore, not only is the original starvation detected by ASAP starvation detection sent rapidly, but any later packets for the same flow will also be transmitted rapidly.

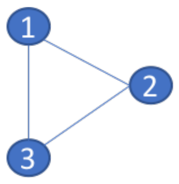
IV. Evaluation

The primary goal of our evaluation is to show that ASAP rapidly addresses starvation and does no harm in cases where starvation is not occurring. We first discuss the methodology used to evaluate ASAP and then present results.

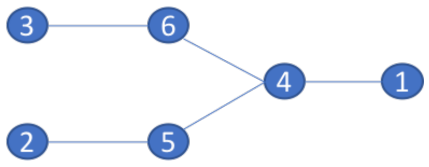
ASAP is implemented within the larger IRON system, and experiments are run on a local testbed. The testbed setup contains a physical machine for each IRON node, all connected to a highspeed switch. The IRON system framework can emulate link delays and capacities. IRON is a backpressure forwarding implementation capable of transporting data at hundreds of megabits per second [6]. IRON is written in C++ and runs in user-space; the ASAP module resides in IRON and continually monitors the available link capacity, queue depths, and head-of-line queuing delays, and directly inserts virtual packets into the monitored queues as appropriate.

We present results from running experiments with and without ASAP to highlight ASAP's ability to address starvation. As an additional point of comparison, we also implemented a linear delay-term approach, which, as we discuss in Section II, interacts poorly with queue-based admission control. This approach tracks the head-of-line queuing delay and adds that directly to the gradient computation.

The primary metric used for evaluation is goodput over time. All graphs contain goodput on the y-axis and time into the scenario on the x-axis, and were generated using the TRPR tool [8]. Starvation is easy to visualize using these graphs, as the goodput of a starved flow will go to zero. These graphs also help visualize the capacity used by different flows.



(a) 3 node triangle topology



(b) Y-shaped topology

Fig. 3: Topologies used for evaluation scenarios

There are three scenarios used to highlight ASAP. The first two use the triangle topology shown in Figure 3a and the third uses the Y-shaped topology shown in Figure 3b.

A. *Scenario 1 - Starvation of Small Flow*

The first scenario operates over the triangle topology (Figure 3a), with all links set to a capacity of 10Mbps and delay of 10ms. There are numerous low-priority flows competing with a single high-priority, low-rate flow. The experiment lasts for around 90 seconds, with traffic described in the table below:

TABLE I: Scenario 1 Flow Definitions

Flow Group	Time	(Src,Dst)	Priority	Volume
1 (20 flows)	10-90	(1,2)	Low	5Mbps
2 (1 flow)	15-50	(1,3)	High	5Kbps
3 (20 flows)	50-90	(1,3)	Low	5Mbps

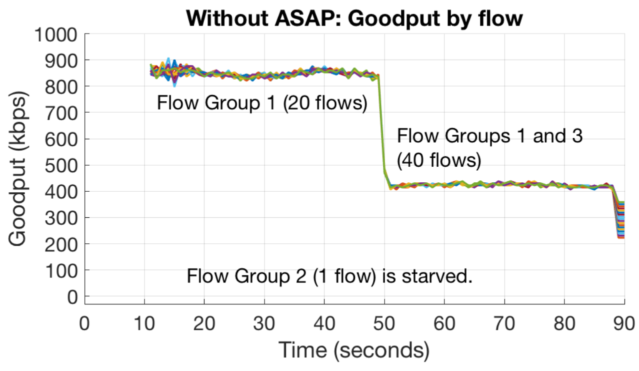
No flows occur for the first 10 seconds to allow the system to start up. Flow Group 1 is a set of low priority, high-rate flows that are active the entire time (other than start up) and have the potential to starve the low-rate flow in Flow Group 2, since the queue at node 1 for destination node 2 will be perpetually deeper than the queue at node 1 for destination node 3. Note that even though Flow Group 1 is destined to 2, backpressure routing will cause *all paths* to 2 to be utilized, optimizing network throughput. Therefore, node 3 will be forwarding traffic from node 1 destined to node 2. After 50 seconds, we start Flow Group 3 destined to node 3 to evaluate whether ASAP's built-up virtual packets give unfair preference once the period of starvation is over.

Figure 4a shows the goodput captured by nodes 2 and 3 **without ASAP running**. As desired, all flows in Flow Group 1 get equal priority and equal share of the available capacity less IRON overhead (recall there is actually 20Mbps of capacity available from 1 to 2, since there are two 10Mbps paths). When Flow Group 3 starts, all flows from Flow Groups 1 and 3 get equal share of capacity, again as desired. However, the goodput graph shows that the flow in Flow Group 2 is being completely starved.

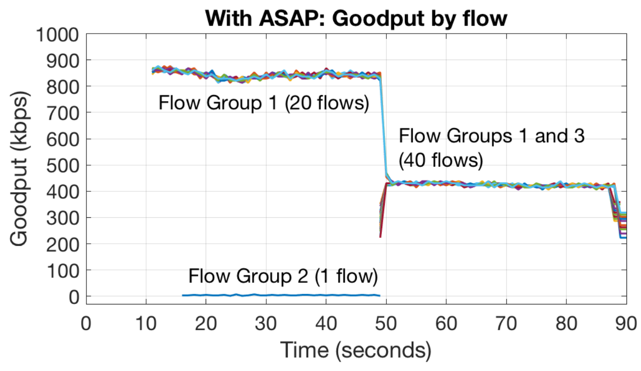
Figure 4b shows the goodput at these nodes **with ASAP running**. The goodput now clearly shows the low-rate flow in Flow Group 2 operating for all 35 seconds during which it was active. Therefore, ASAP is properly detecting starvation and rapidly addressing it. Furthermore, the goodput after Flow Group 2 ends matches the goodput without ASAP. This highlights an important point: ASAP is not harming Flow Groups 1 or 3 after Flow Group 2 has ended. This indicates that ASAP is not inflating the gradient for extended periods of time after the vulnerable flow has ended, and hence not doing harm to other flows in its attempt to address starvation.

B. *Scenario 2 - Starvation of TCP Traffic*

The second scenario also operates over the triangle topology (Figure 3a), with all links set to a capacity of 10Mbps and delay of 10ms. Long lasting TCP flows are sent between the nodes in a circular pattern as described in the traffic table below. The experiment runs for about 50 seconds with the first 10 seconds left for initialization.

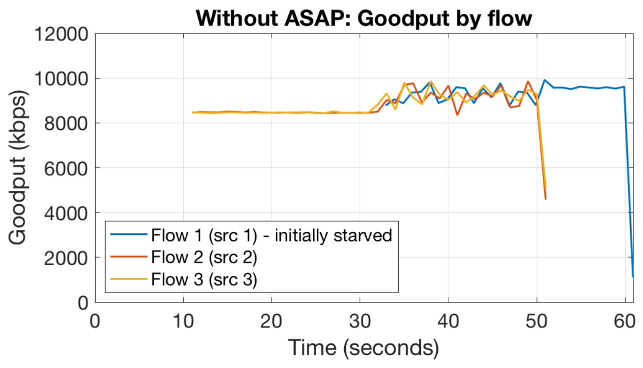


(a) Without ASAP, Flow Group 2 is completely starved.

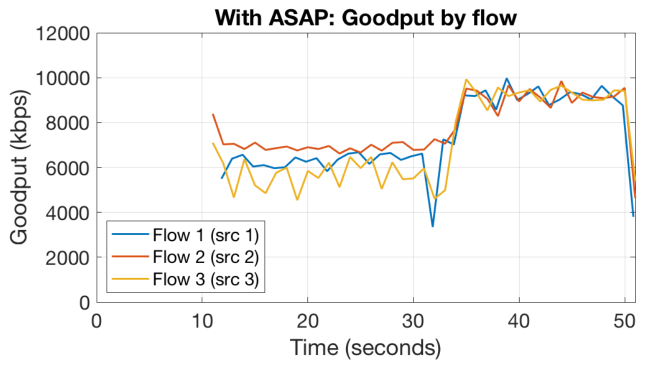


(b) ASAP prevents starvation of small flows.

Fig. 4: Scenario 1 experimental evaluation, including goodput observed at both destination nodes.



(a) Without ASAP, TCP ACKs for Flow 1 are starved when the 1 → 2 link is down.



(b) ASAP eliminates starvation.

Fig. 5: Scenario 2 experimental evaluation, including goodput observed at all three destination nodes.

TABLE II: Scenario 2 Flow Definitions

Flow	Time	(Src,Dst)	Priority	Volume
1	10-50	(1,2)	Medium	Max
2	10-50	(2,3)	Medium	Max
3	10-50	(3,1)	Medium	Max

Until around 30 seconds into the run (20 seconds after the flows are started), the link between nodes 1 and 2 is down. After 30 seconds, that link comes up. While the link is down, the flow between 1 and 2 must traverse node 3. Also, the TCP ACKs must traverse the path 2-3-1. These ACKs are competing with data traffic flowing from 2 to 3, and therefore are at risk of being starved at node 2, since the queue to destination 3 is inherently deeper than the queue to destination 1.

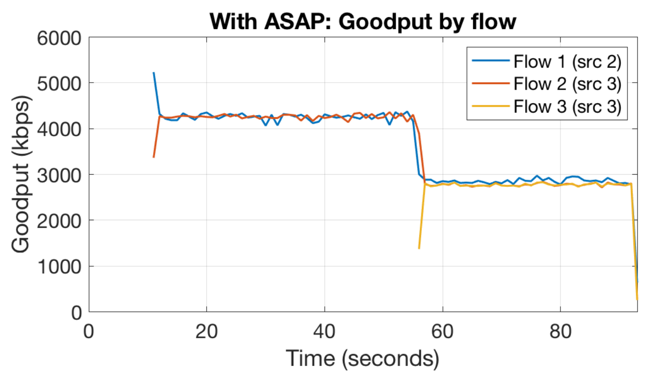
Figure 5a shows the goodput captured by all nodes **without ASAP running**. When the link between nodes 1 and 2 is down, flow 1 is starved. This is due to the competition of the low-rate TCP ACKs (which use significantly less capacity than data) with the data components of the other flows.

Figure 5b shows the goodput captured by all nodes **with ASAP running**. ASAP is able to detect the starvation of the flow 1 TCP ACKs, increase the gradient appropriately using virtual packets, and eliminate the starvation.

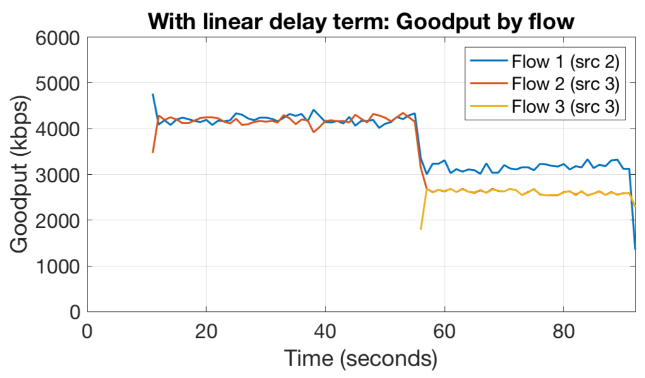
C. Scenario 3 - No Starvation in Y-Shaped Topology

The previous two scenarios shows that ASAP rapidly detects and addresses starvation when it is occurring. This third scenario shows that ASAP does no harm when starvation is not occurring. Moreover, we compare ASAP to the linear delay-term addition approach discussed in Section II-A. This approach considers the delay term and queue depth in linear combination to compute the gradient. We have assigned a weight of 2 for the delay term, which is necessary to ensure starvation is eliminated when it does occur. This scenario shows that ASAP does no harm to admission control, whereas the linear delay-term addition does. All links are set to a capacity of 10Mbps and delay of 10ms, and the experiments run for 100 seconds.

This scenario operates over a Y-shaped topology (Figure 3b). At the start of the experiment, two flows start at nodes 2 and 3, both destined for node 1. Half way through the experiment, an additional flow is added starting at node 3. The end-to-end latency characteristics of this experiment are detailed in Section II-A, and highlight why the direct delay term addition results in unfair goodput between the flows. The traffic is outlined in the following table:



(a) With ASAP, goodput is equal for equal priority flows



(b) When using a linear delay term for BPF, the asymmetric latencies described in Section II-A cause unfair goodput.

Fig. 6: Scenario 3 experimental evaluation, showing goodput observed at node 1 with ASAP and with BPF including a linear delay term.

TABLE III: Scenario 3 Flow Definitions

Flow	Time	(Src,Dst)	Priority	Volume
1	10-100	(3,1)	Medium	10Mbps
2	10-100	(2,1)	Medium	10Mbps
3	55-100	(3,1)	Medium	10Mbps

In this topology, starvation is not occurring. All packets are being drained at steady rates according to the backpressure algorithms. Figure 6a shows that all flows achieve equal goodput values when running ASAP. Running this experiment without ASAP results in identical goodput results. This shows that the starvation detection mechanism is not triggering when starvation is not occurring. ASAP is continually monitoring the queues but is not taking action.

Figure 6b shows that using a linear delay addition term directly in the gradient computation does not account for expected asymmetries in queue delay, resulting in unfair goodput rates between flows. ASAP does not have this problem, since it does not attempt to affect flows unless starvation is detected.

The results of these experiments show that ASAP is able to effectively detect and address starvation without causing harm when starvation is not occurring.

V. Conclusions and Future Directions

ASAP is a new approach to avoiding starvation in backpressure networks with admission control. ASAP adds virtual packets to artificially inflate the queue when starvation is detected. The amount of virtual bytes is determined based on a non-linear function of the head-of-line delay.

The use of virtual packets makes ASAP an inherently simple extension to backpressure forwarding, where nodes still only exchange queue depth information with neighbors. Furthermore, the use of virtual packets is a graceful mechanism for maintaining the artificially-increased queue depths long enough to avoid future starvation of the same flow, while still allowing quick reactions to changes in network conditions. By determining the virtual bytes using a non-linear function and only when starvation is occurring, ASAP avoids disrupting existing admission control mechanisms based on queue depth.

One promising direction for further study is to investigate other functions, including different functions across different nodes, both for starvation detection and for converting head-of-line delay into a number of virtual bytes. A further direction is on extensions to add virtual bytes based on head-of-line delay for all queues, as well as just for queues experiencing starvation, as a means to reduce overall queuing delay.

References

[1] L. Tassiulas and A. Ephremides, "Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks," *IEEE/ACM Trans. Autom. Control*, vol. 37, no. 12, pp. 1936–1948, Dec. 1992.

[2] B. Ji, C. Joo, and N. Shroff, "Delay-based back-pressure scheduling in multihop wireless networks," *IEEE/ACM Trans. Netw.*, vol. 21, no. 5, pp. 1539–1552, Oct. 2013.

[3] M. Neely, "Delay-based network utility maximization," *IEEE/ACM Trans. Netw.*, vol. 21, no. 1, pp. 41–54, Feb. 2013.

[4] S. Vargaftik, I. Keslassy, and A. Orda, "No packet left behind: Avoiding starvation in dynamic topologies," *IEEE/ACM Trans. Netw.*, vol. 25, no. 4, pp. 2571–2584, Jun. 2017.

[5] M. Neely, E. Modiano, and C.-P. Li, "Fairness and optimal stochastic control for heterogeneous networks," *IEEE/ACM Trans. Netw.*, vol. 16, no. 2, pp. 396–409, Apr. 2008.

[6] Raytheon BBN Technologies, "Iron internal bbn technical report," 2018.

[7] L. Huang and M. J. Neely, "Delay reduction via lagrange multipliers in stochastic network optimization," *IEEE Transactions on Automatic Control*, vol. 56, no. 4, pp. 842–857, Apr. 2011.

[8] U.S. Naval Research Lab (NRL), "Trpr user's guide version 2.1b5," Accessed April 2018, https://downloads.pf.itd.nrl.navy.mil/docs/proteantools/trpr.html.