On Delay Performance of Throughput Optimal Back-pressure Routing: Testbed Results

Venkataramana Badarla and Douglas J. Leith
Hamilton Institute, NUI Maynooth, Ireland

***Abstract*—Maximizing network throughput via back-pressure routing is the subject of a considerable body of literature. It is well known that back-pressure routing induces long delays and this has motivated a number of proposals for improving the delay performance. However, to our knowledge, no useful results are available in the literature that characterize the delays induced by packet re-ordering which is due not only to the nature of back-pressure routing but also to the ubiquity of routing loops. Therefore in this paper we provide a systematic evaluation of basic back-pressure routing algorithm and two of its recently proposed variants on an experimental testbed. This provides the first direct comparison of delay performance. Our findings reveal that even in simple network topologies these algorithms induce extensive routing loops with associated high packet delay.**

I. Introduction

Using dynamic multi-path routing, back-pressure routing ([1]-[3] and references therein) offers considerable gains in throughput over conventional single-path routing algorithms. Indeed, it guarantees to achieve the network capacity and so its throughput performance cannot be bettered by any algorithm. While maximizing network throughput, back-pressure routing comes with no guarantee on network delay. Indeed certain features of the back-pressure routing algorithm suggest that long delays will be common. Firstly, back-pressure routing uses queue buildup at nodes to create a "gradient" within the network that guides routing. However, this may come at the cost of increased queuing delay. Secondly, back-pressure routing tends to explore all paths in a network, including paths with loops and "dead-end" paths that cannot lead to the desired destination. Hence, packets generally may not take the shortest path to their destination, thereby leading to additional delay.

Unfortunately, analytic results on the delay performance of these algorithms are difficult to establish and only relatively weak qualitative or asymptotic bounds are available (*e.g.* see [4], [5] and references therein). To our knowledge, there are no useful theoretical, simulation, or experimental results available that characterize the delays induced by packet re-ordering which, as we will see in this paper, is a major factor affecting the performance of current back-pressure routing algorithms due not only to their multi-path nature but also to the ubiquity of routing loops. Further, while back-pressure algorithms offer substantial throughput performance gains, the ease or otherwise with which this potential might be realized in practical networks is currently unclear.

Therefore in this paper we provide a systematic evaluation of basic back-pressure routing algorithm and two of its recently proposed variants on an experimental testbed. This provides the first direct comparison of delay performance. Our findings reveal that even in simple network topologies these algorithms induce extensive routing loops with associated high packet delay.

II. Back-pressure Algorithms

In this section we briefly review the basic back-pressure algorithm and its variants.

A. Basic Back-pressure Algorithm

The basic back-pressure algorithm [2], [3] is detailed in Algorithm 1. Each forwarding node in the network uses per flow FIFO queuing and we let $q_n^f(t)$ denote the number of packets queued for flow f at node n at time t. Roughly speaking, whenever it has a transmission opportunity each node n forwards a packet from the flow f^* to the next hop $m^*(f^*)$ that jointly maximizes the utility

$$U_{n,m}^f(t) = \left(q_n^f(t) - q_m^f(t)\right) R_{n,m}$$

where $R_{n,m}$ is the mean transmit rate of the link from node n to node m.

Algorithm 1 Basic back-pressure algorithm

1: For each flow f and neighbor node m, node n computes utility $U_{n,m}^f(t) = \left(q_n^f(t) - q_m^f(t)\right) R_{n,m}$, $m^*(f) = \arg\max_m U_{n,m}^f$, $f^* = \arg\max_f \left(U_{n,m^*(f)}^f\right)$ (ties broken arbitrarily).
2: If $U_{n,m^*(f^*)}^{f^*} > 0$, then node n schedules flow f^* and forwards $\min(q_n^{f^*}(t), R_{n,m^*(f^*)})$ packets to neighbor $m^*(f^*)$.
3: Otherwise node n takes no action at time t.

Observe that this back-pressure algorithm tends to transmit packets to the neighbor(s) with the smallest queue and highest link rate and intuitively the queue backlogs provide a "gradient" down which packets are routed. However, it commonly occurs that $q_n^f(t)$ and $q_m^f(t)$ differ by only a small amount. The routing "gradient" is then both small and rapidly fluctuating, in which case routing loops can readily be induced. Furthermore, observe that even when a neighbor has no connectivity to the destination of a flow, packets will still be forwarded to this neighbor until such time as a sufficiently large queue backlog has developed to prevent further packets stop being forwarded in that direction. However, all the packets already sent in that direction will never reach the flow destination.

B. Back-pressure with Shadow Queues

In *shadow-queue* based back-pressure routing [6], each node maintains a fictitious queue called a shadow queue, which is just a counter, for each flow. Scheduling and forwarding decisions are made based on shadow queue sizes $\tilde{q}_n^f(t)$ instead of the real queue sizes $q_n^f(t)$. The shadow queues are updated in a similar manner to the real queues but with a shadow packet arrival rate that is slightly higher than the real packet arrival rate. For example, if the real packet arrival rate of flow f is λ^f then the shadow packet arrival rate is $(1+\epsilon)\lambda^f$, $\epsilon > 0$. In order to ensure stability of the shadow queues, the shadow arrival rate $(1+\epsilon)\lambda^f$ must lie in the interior of the network capacity region. Since the shadow queue size always upper bounds the real queue size, it follows that the real queue is also guaranteed to be stable. The advantage of this approach is that build up of the shadow queues can take place to provide a routing "gradient" for the back-pressure algorithm without corresponding build up (and so packet delay) of the real queues, but at the cost of reduced network capacity.

C. Min-Resource Routing

Min-resource routing is also proposed by [6]. Here, the utility function $U_{n,m}^f$ is modified to

$$U_{n,m}^f(t) = \left(q_n^f(t) - q_m^f(t) - M\right) R_{n,m}$$

where $M \geq 0$ is a design parameter. With this change, the differential in queue backlog at node n and neighbor m must exceed M packets before packets for flow f will be forwarded to m (since the utility function must be positive before a link will be used). $M = 0$ recovers the basic back-pressure algorithm. Larger values of M make forwarding decisions less sensitive to small changes in queue occupancy, but this comes at the cost of large queue sizes.

III. Implementation Details and Experimental Setup

We implemented the back-pressure algorithms as a *DynamicRouter* element within the Click router framework in Linux. The Click router [7] framework provides a modular architecture that lies between the network layer and the device driver layer and interacts with the network layer and the interface device driver via the Click elements ToHost/FromHost and ToDevice/FromDevice, respectively.

The back-pressure algorithms require each node to maintain per-flow queues and to exchange messages with its one-hop neighbors in order to share queue size information (required to calculate the utilities $U_{n,m}^f(t)$). We implemented this message passing via special packets called *neighbor-update* packets. A node must also maintain the corresponding queue size information at all of its one-hop neighbors. At each transmission opportunity, the node uses this queue size information to calculate the utilities $U_{n,m}^f(t)$ and schedules transmission of a packet from the flow f^* to neighbor $m^*(f^*)$ that maximizes the utility.

As shown in Fig. 1, the DynamicRouter element has 3 input ports (for receiving packets) and 3 output ports

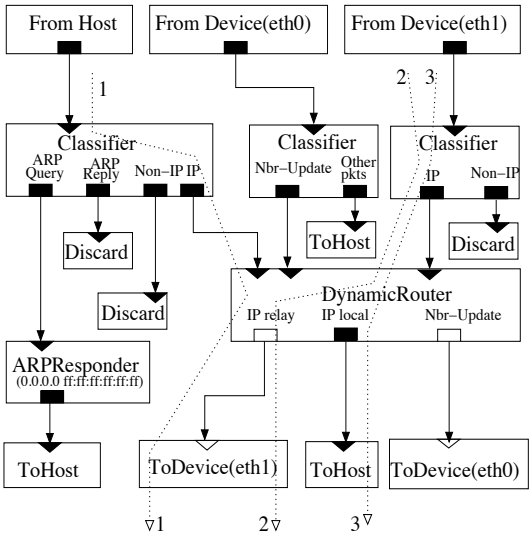
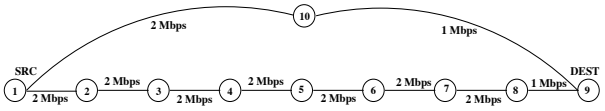
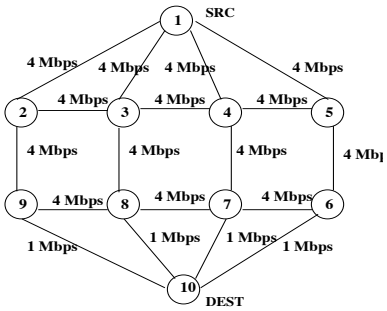


Fig. 1. Packet flow-graph of Click router implementation. Notation: Input (output) ports are represented by triangles (square boxes). A connection between a shaded (empty) input and output port is called a Push (Pull) connection. In a Push (Pull) connection it is the sending element (receiving element) that initiates a packet transfer. The dotted lines show the possible paths for the data packets. The packets of a flow typically follow path 1 at source node, path 2 at intermediate nodes, path 3 at destination node.

(for transmitting packets). It receives locally generated IP packets passed down from the network layer, neighbor-update packets that are received from the neighbors, and IP packets received from neighbors, on its first, second, and third input ports, respectively. Whenever the ToDevice element, which sends the packets to the network card, gets scheduled it sends a request for a packet to the DynamicRouter element. In response, the DynamicRouter uses the back-pressure algorithm to select a packet to send to one of its neighbors and delivers this packet to the ToDevice element. Note that once a forwarding decision is made, DynamicRouter fills the MAC address of the corresponding next-hop node into the destination MAC address field in the ethernet header of the packet. IP packets either generated by the local host (and so passed via the FromHost element) or received from neighboring nodes (and so passed via the FromDevice element) are handed off to DynamicRouter for processing. Packets which are to be forwarded are stored in the appropriate per flow queue. Packets for which the current node is the destination are stored in a re-assembly buffer if they arrive out-of-order, otherwise they are handed over via the second output port (ToHost) to the local host. In order to help the destination to resolve packet re-ordering, source node assign a sequence number to each packet which is stored in a special header *DR header* located between transport and IP header. The DynamicRouter element also runs a timer to periodically generate neighbor-update packets. To avoid interference between neighbor-update packet and data packet transmissions, the neighbor-update packets are transmitted on a separate network interface. Note that as a node can



(a) Network with 2 paths



(b) Network with 4 paths

Fig. 2. Network topologies. SRC and DEST denote flow's source and destination nodes, resp.

learn the MAC addresses of its neighbors from the received neighbor-update packets, there is no explicit ARP request and response messaging required here.

Hardware and Software Configuration: We conducted our experiments on a testbed constructed from 10 Soekris net5501 embedded Linux boxes. These boxes run Linux 2.6.24.7 and have a 433MHz CPU, 256MB RAM, and four 100Mbps ethernet ports. As the link rates in the scenarios (see Fig 2) that we consider are less than the 100Mbps physical rate, we included a delay component within the DynamicRouter element which introduces a specified minimum delay between packets delivered to the ToDevice element, thereby allowing lower link rates to be emulated in a controlled manner. The MTU is taken as 1400 Bytes and the interval between neighbor-update packet transmissions is set to 1 millisecond.

IV. Performance Evaluation

It is important to emphasize that our goal is not to achieve exhaustive testing, which is in any case impossible due to the large number of network topologies, device configurations and traffic mixes that exist in modern networks. Instead we seek to define a small number of benchmark tests that are amenable to systematic, reproducible testing and which exercise the core functionality of the routing algorithms. As we will see, relatively simple network configurations are already sufficient to uncover a number of basic issues with existing back-pressure algorithms.

We consider the following performance metrics.

a) Packet delay This is computed at the sender as the difference between the time $t_s(p)$ when a packet p is transmitted and the time $t_d^{hl}(p)$ when the packet is delivered to the application layer at the receiver. As the source and destination clocks will not be perfectly synchronized, we compute $t_d^{hl}(p)$ as follows. The source node sends t_s in the header of each transmitted packet. The receiver echos this t_s value to the sender in a small 8 byte packet sent via a dedicated ethernet port operating at 100Mbps and connected by a cross-over cable. Since the transmit time of an 8 byte packet at 100Mbps is approximately $0.6\mu s$, the time $t_d^{hl}(p)$ can be accurately estimated as the time when this echo packet arrives at the source and the packet delay is then calculated as $t_d^{hl}(p) - t_s(p)$.

The *packet delay* can break down into a component due to network delay (the time between when a packet leaves the source and when it arrives at the destination) and a component due to reassembly delay (the time between when a packet enters the reassembly queue at the receiver and when it is delivered in order to the application layer).

b) Throughput This is computed at the receiver as X/T where X is the amount of data received over a time period T, where $T = 500s$.

c) Buffer Requirements We measure the average occupancy of every per-flow queue at every node, with packet-level timing granularity. Also the average re-assembly queue size at destination nodes.

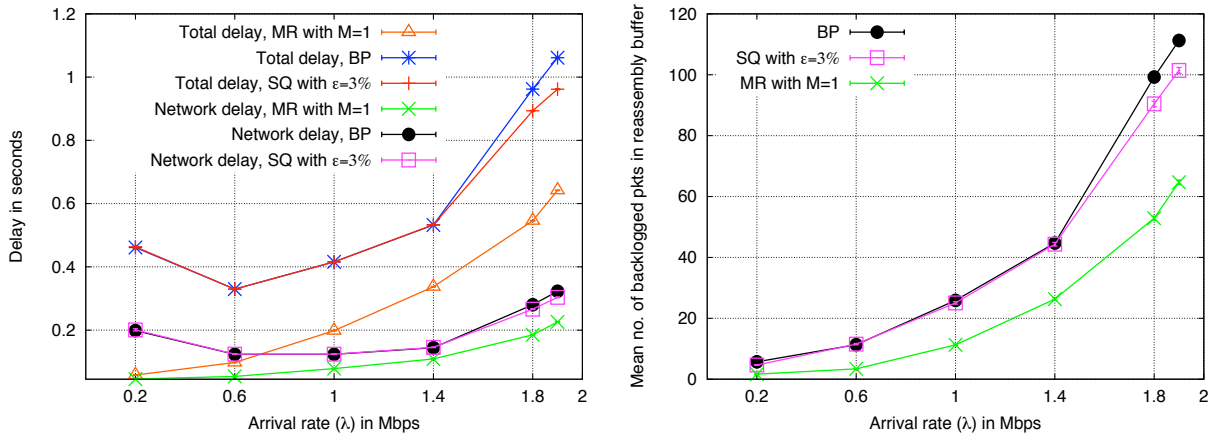
Each experiment is conducted for 5 different runs, with each run being of 500 seconds duration. Results are plotted with 95% confidence intervals marked.

V. Results

In this section we present performance measurements over the topologies shown in Fig. 2 for the three algorithms (i.e., basic back-pressure algorithm (BP), the basic algorithm augmented with shadow queues (SQ), the min-resource algorithm (MR)) discussed in Section II. The topologies are intentionally kept simple, so that we can have a rough understanding on the minimum delay, however they require routing along multiple paths to achieve the network capacity and also they can induce implicit routing loops. The results shown here are for a UDP Poisson traffic, which is generated by $mgen$ utility. Note that we ran shadow-queue and min-resource algorithms for different values of their respective tuning parameters ϵ and M, and presented here the results with the best values for ϵ and M. Selecting a large value for ϵ in shadow-queue based routing leads to improved delay performance, however at the cost of reduced network capacity. Selection of a good value for M in min-resource routing is heavily dependent on the network topology and offered load. A Large value (small value) for M causes poor delay performance at lighter loads (higher loads). Hence there is no unique value for M that best works at all offered loads for a flow. Due to space constraint, the parameter tuning plots are not shown here.

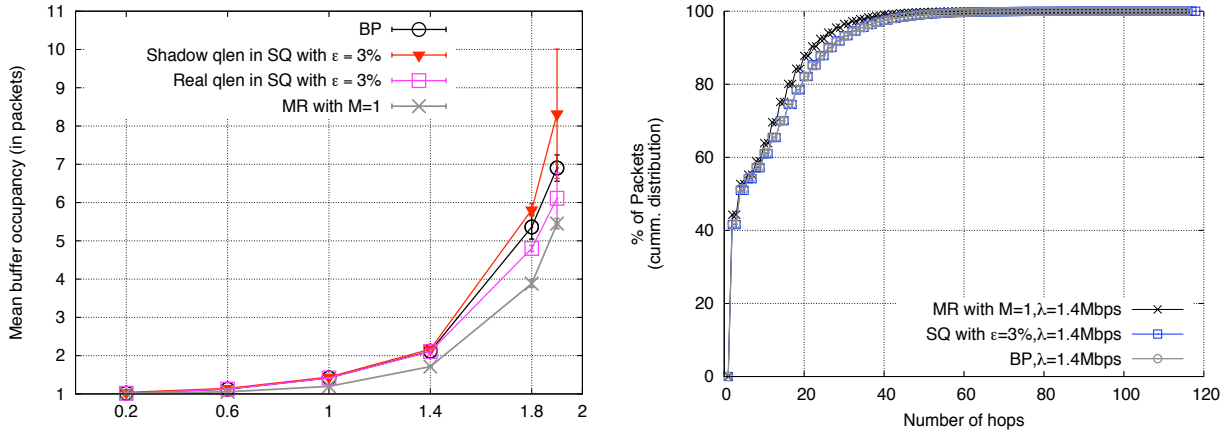
A. First Topology

We begin by considering the network topology shown in Fig. 2(a). Fig. 3(a) shows the mean measured packet delay (and also network delay) vs offered load. We can make number of observations from this results. 1. The total delay is much higher (almost x 3 times) than the network delay at most of the offered loads, which shows the severity (and also the significance) of packet reordering on the delay performance of the algorithms. 2. For both basic back-pressure and shadow-queue algorithms first the delay (both network and total



(a) Mean total and network delays

(b) Mean reassembly queue at receiver



(c) Mean buffer at intermediate nodes

(d) Hop count distrib. at λ=1.4Mbps

Fig. 3. Performance of the algorithms over the topology shown in Fig. 2(a).

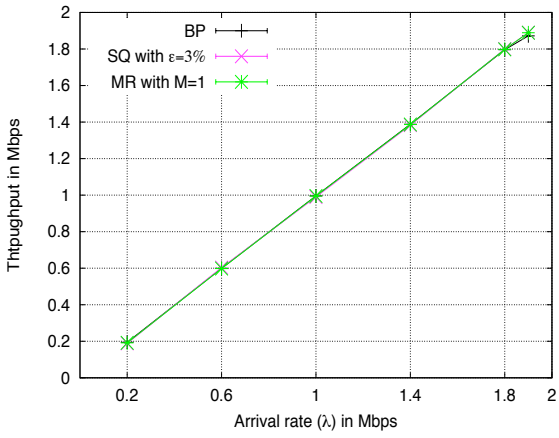
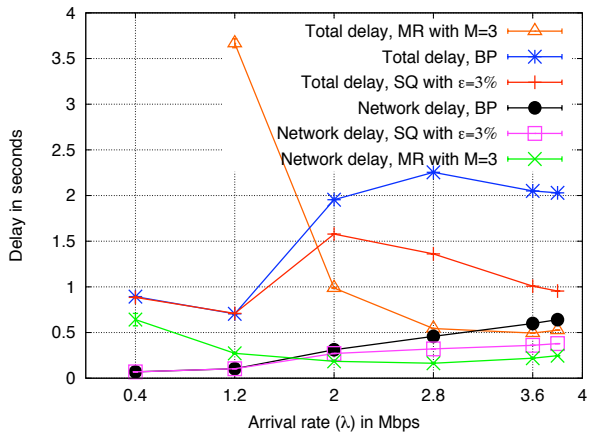


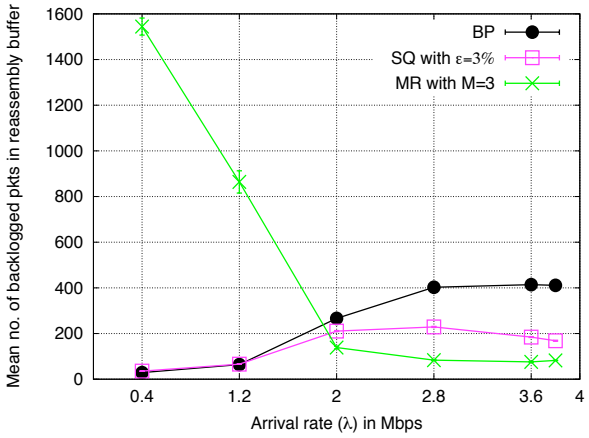
Fig. 4. (a)-(e) and (f)-(i) show the performance of the algorithms over the topology in Fig. 2(a) and Fig. 2(b), respectively.

delay) decreases and then increases with offered load. This is expected as at lighter loads packets are routed where ever excess network capacity is available, thus causing high packet reordering, and at higher loads the increased delays are associated with both reordering and queuing delays. 3. The delay trends of basic back-pressure and shadow-queue routing are almost same at lighter and moderate loads. However, as $\epsilon > 0$ becomes significant at moderate to high loads, shadow-queue routing shows improved delay performance over back-pressure routing. 4. The min-resource routing shows the least delays among the algorithms considered. 5. For all algorithms the delays remain large at high offered loads.

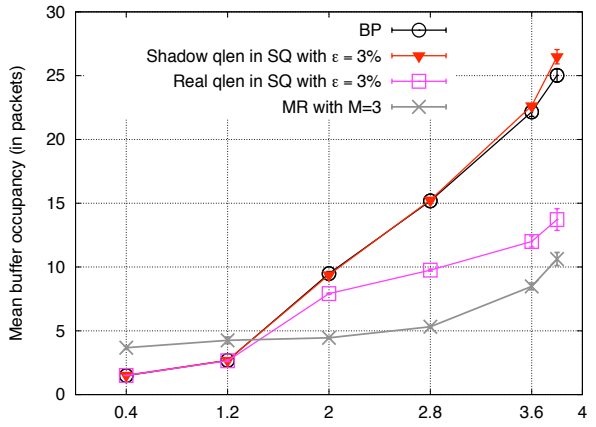
Figs. 3(b) and (c) give insight into the buffer sizing requirements of the algorithms considered. Fig. 3(b) plots the mean number of packets in the reassembly buffer at the receiver. The reassembly buffer is used to ensure in-order delivery of packets to the application layer and so its occupancy is dependent on the amount of packet re-ordering. It can be seen that the reassembly buffer requirements for the basic back-pressure and shadow-queue algorithms are high, with on average more than 100 packets buffered at the destination at an offered load of 1.9Mbps. Fig. 3(c) shows the mean buffer occupancy at forwarding nodes. Observe that, as expected the queue length with the basic back-pressure algorithm and the shadow queue length with shadow-queue algorithm are essentially the same at low to moderate loads, and at moderate to high offered loads



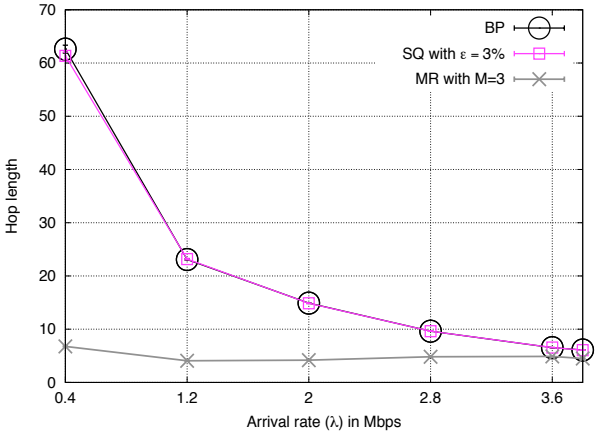
(a) Mean total and network delays, For MR, at λ=0.4Mbps, these delays are 36s and 0.64s, resp.



(b) Mean reassembly queue at receiver



(c) Mean buffer at intermediate nodes



(d) Mean number of hops traversed

Fig. 5. Performance of the algorithms over the topology shown in Fig. 2(b).

(as $\epsilon > 0$ becomes significant) the shadow queues grow faster than the queues in basic back-pressure routing. Also can be observed that, with the shadow-queue algorithm the real queue lengths are smaller than the shadow queue lengths.

We can also understand the severity of packet-reordering via hop count results show in Fig. 3(d), which plots the cumulative distribution of the number of hops traversed by a packet as it travels from src to $dest$. Observe that at λ=1.4Mbps, 15-20% of the packets in all the algorithm traverse more than 20 hops, and the maximum number of hops traversed for all these algorithms is above 100 hops. Also can be observed that both basic back-pressure and shadow-queue routing algorithms show essentially same performance. Min-resource routing shows slightly better performance over the other two algorithms. We can note that even such mild gain in hop count performance leads to significant gain in the delay performance as can be seen in Fig. 3(a).

Fig. 4 shows the measured throughput for all the algorithms. As all the algorithms are throughput optimal, it can be seen that throughput attained is roughly same as the offered load. This result demonstrates the potential for substantial throughput gains via multi-path back-pressure routing.

B. *Second Topology*

We now present measurements for the topology in Fig. 2(b). As expected, the throughput attained is same as that of the offered load, however due to space constraint the plot not show here.

Fig. 5(a) shows the measured total delay and network delay performance vs offered load. We can identify a number of qualitatively different features from the data for the previous topology. 1. Packet delays with basic back-pressure routing is significantly higher for this topology, and it ranges between [1s,2.3s]. 2. For the basic back-pressure and shadow-queue algorithms the delay is no longer increasing with offered load. We can gain insight into this behavior from the re-assembly delays (i.e., *total delay - network delay*). It can be seen that at higher loads (above 2.8Mbps), the reassembly decreases with offered load while the network delay increases. That is, reordering decreases but queue backlogs within the network increase. For the basic back-pressure and shadow-queue algorithms, on balance the decrease in reassembly delay is greater than the increase in network delay and so the

overall delay falls. 3. At low offered loads the delay with the min-resource algorithm is significantly higher than that of the basic back-pressure algorithm. This is true not only for the overall delay but holds for both the reassembly delay component and the network delay component. For example at λ is 0.4Mbps, total, reassembly, network delays are 36s, 35.36s, and 0.64s resp. With regard to the network delay component, we note that the min-resource algorithm *subtracts* amount M from the queue occupancy to obtain the utility function. At low offered loads this leads to an increase in mean queue occupancy and so network delay. With regard to the reassembly delay component, what appears to be happening is that quasi-orphaned packets are common at low offered loads - due, once again, to the need to build up queues larger than M before packets can be forwarded.

Fig. 5(b) shows the mean reassembly queue occupancy at the receiver. Observe that while basic back-pressure and shadow-queue algorithms show high occupancy at high offered loads (ranges between [200,400] packets, min-resource routing shows very high occupancy at lighter loads (i.e., about 1600 packets). Fig. 5(c) presents the corresponding queue occupancy results at forwarding nodes. The only notable differences are 1. In shadow-queue based routing, real queue lengths are much smaller than shadow queue lengths (which reveals the fact that higher the difference, more the gain in delay performance) 2. At low offered loads the mean buffer occupancy with the min-resource algorithm is slightly higher than the other routing algorithms.

Fig. 5(d) shows the mean number of hops traversed vs offered loads. Observe that the number of hops traversed in basic back-pressure and shadow-queue algorithms is decreasing with offered load, and both show exactly same results at all offered loads. This confirms that gains in delay performance for shadow-queue algorithm is only from smaller real queues over basic back-pressure algorithm (as can be witnessed in Fig. 3(h)). We can also note that the packets in min-resource routing traverse through smaller number of hops compared to basic back-pressure and shadow-queue algorithms.

C. Multiple flows

We also have studied the performance of these algorithms for multiple flows (i.e., 3 flows). We observed that delay trends are same of those of single flow studies, however showing much higher delays. The results are not shown due to space issue.

VI. Related Work

In selecting the back-pressure algorithms to study, we have tried to focus on algorithms suited to practical implementation on current hardware. We note that [8] proposes use of a last-in-first-out (LIFO) queuing discipline to improve the delay performance of back-pressure routing. This works by sacrificing initial packets to create back-pressure, however ignoring initial packets is not always/practically possible, and so is not considered here. [9] considers improving delay performance by maintaining separate queues for each loop-free path. However, as the network size grows this results in an exponential growth in the number of queues that must be maintained by forwarding nodes and so we argue that it is not suited to practical implementation. The work in [10] proposes use of queue draining times, instead of queue length in the utility equation of back-pressure routing. However, this approach lacks stability guarantees in general network topologies. In [11], the authors propose use of joint coding and routing to improve the delay performance of the back-pressure routing. Such coding can potentially be combined with any of the back-pressure algorithms considered here, but our focus in this paper is on the routing algorithms themselves.

VII. Conclusions

In this paper we present an experimental performance evaluation of recently proposed low-delay back-pressure routing algorithms, providing the first direct comparison of delay performance. We find that, while lowering delay relative to the original back-pressure routing algorithm, the absolute value of delay nevertheless remains large due to a tendency for the algorithms to induce extensive routing loops. Apart from the basic back-pressure algorithm, the other algorithms considered involve design parameters that significantly affect performance yet lack guidelines as to how they should be chosen. In our tests we found that the appropriate value are strongly dependent on the network and flow configuration. Motivated by these observations, our future work includes enhancing basic back-pressure routing with a machanism to adaptively updating the design parameters for each flow by maintaining a recent history of delay and throughput trends.

References

[1] L. Tassiulas and A. Ephremides, "Stability Properties of Constrained Queueing Systems and Scheduling for Maximum Throughput in Multihop Radio Networks," *IEEE Trans. on Automatic Control*, vol. 37, no. 12, pp. 1936-1949, 1992.

[2] A. L. Stolyar, "Maximizing Queueing Network Utility subject to Stability: Greedy-Primal Dual Algorithm," *Queueing Systems*, vol. 50, no.4, pp. 401-457, 2005.

[3] M. J. Neely et al, "Fairness and Optimal Stochastic Control for Heterogeneous Networks," *in Proc. IEEE Infocom*, 2005.

[4] G. R. Gupta and N. B. Shroff, "Delay Analysis for Multi-Hop Wireless Networks", *in Proc. IEEE Infocom*, 2009.

[5] Long B. Le, K. Jagannathan, and Eytan Modiano, "Delay Analysis of Maximum Weight Scheduling in Wireless Ad hoc Networks," *in Proc. Conference on Information Sciences and Systems*, 2009.

[6] L. Bui, R. Srikant, and Alexander Stolyar, "Novel Architectures and Algorithms for Delay Reduction in Back-pressure Scheduling and Routing", *in Proc IEEE Infocom Mini-Conference* 2009.

[7] "The Click Modular Router Project", http://read.cs.ucla.edu/click/

[8] S. Moeller, A. Sridharan, B. Krishnamachari, and O. Gnawali, "Routing without Routes: The Backpressure Collection Protocol", accepted to *9th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)* 2010.

[9] L. Ying, S. Shakkottai, and A. Reddy. "On Combining Shortest-Path and Back-Pressure Routing Over Multihop Wireless Networks", *in Proc. IEEE Infocom*, pp. 1674-1682, 2009.

[10] V. Subramanian, and D.J. Leith. "Draining Time based Scheduling Algorithms", *in Proc. IEEE Conference on Decision and Control*, pp. 1162-1167, 2007.

[11] V. Badarla, V. Subramanian, and D. J. Leigth, "Low-delay dynamic routing using fountain codes", *IEEE Communications Letters*, vol. 13, no. 7, pp. 552-554, July 2009.