DiffQ: Differential Backlog Congestion Control for Wireless Multi-hop Networks

Ajit Warrier, Sangtae Ha, Prashant Wason and Injong Rhee
Dept of Computer Science
North Carolina State University
acwarrie,sha2,pwason,rhee@ncsu.edu

ABSTRACT

Congestion control in wireless multi-hop networks is challenging because of two reasons. First, broadcast is an inherent feature of wireless networks and motivates many creative protocols including opportunistic routing and network coding. These protocols enable the use of many diverse, yet dynamically changing routing paths. Congestion control for these protocols using traditional end-to-end protocols such as TCP may result in too conservative rate control. Second, the wireless medium is shared among neighboring nodes; thus bandwidth must be allocated fairly among neighboring flows that do not necessarily share the same link. There have been no practical solutions for congestion control for these networks. Inspired by existing theoretical solutions of cross-layer optimization, we develop a protocol, called DiffQ, for congestion control in wireless multi-hop networks. DiffQ can support congestion control for network flows that use either single-path or opportunistic multi-path routing. The protocol is currently implemented in Linux 2.6 series and tested in a network of 46 IEEE 802.11 wireless nodes. It is observed that DiffQ greatly improves the efficiency and fairness of existing transport protocols that use application-level multi-path routing and single-path routing.

1. INTRODUCTION

Providing congestion control for wireless multi-hop networks is challenging. The major challenge comes from the liberal use of multi-paths in wireless networks – especially so with the recent proposal of schemes such as opportunistic routing [4] and network coding [24], which leverage the broadcast nature of wireless networks to improve network capacity. In these protocols, packets of a single flow simply "flows" through many paths toward the destination just like a river flowing through valleys. They make possibly dynamic and local changes in routing paths adapting to dynamically varying wireless link condition. Routing is also non-deterministic as it occurs opportunistically through intermediate routers that *happen to* receive packets through overhearing.

Unfortunately, there has been little research on congestion control for non-deterministic multi-path routing for wireless multi-hop networks. Most existing work in multi-path routing pertains to wired networks (e.g., [25, 41]). Applying such algorithms directly to flows taking non-deterministic routing paths is difficult – since the path traversed by a packet is non-deterministic, it is not feasible to allocate traffic to specific paths. Moreover, most congestion control enhancement proposals for wireless networks (see [30]) pertain to single path routing and rely on end-to-end congestion control after gathering feedback on path characteristics from the networks such as link estimation [3, 6, 39], ECN [12, 35] or AQM marking [41] or losses [43, 32, 40]. Applying these protocols to opportunistic multi-path routing is not effective because of a potentially large number of paths that a single flow may take. As the number of possible paths gets large, receiving feedback from all (possibly temporary) congestion incidents from various parts of the network can cause feedback implosion commonly seen in reliable multicast [10]. Protocols using TCP as end-point congestion control must react to all these uncorrelated congestion signals from the network which limits the transmission rate to a very low rate, causing low resource utilization. Techniques to aggregate the feedback inside the network such as reporting only the maximum delays (e.g., ATP [3]) or minimum link rate (e.g., EXACT [6]) lead to too conservative rate control because the end-point rate control protocols must react only to the worst case condition in the network.

Another challenge comes from the wireless medium being a shared medium over the air. The common form of media access control (MAC) is CSMA such as IEEE 802.11 where a radio transmission can affect a geographically scoped region instead of a specific receiver. In such a network, contention occurs not only among those flows that share the same links or routers, but also among neighboring flows that do not necessarily share the links. Directly applying TCP to these networks causes severe unfairness in resource usage among competing flows and this problem has been well-documented in a number of papers (see [17]).

Congestion control over multi-path routing have also been studied in cross-layer optimization. Many existing cross-layer optimization frameworks are highly theoret-

ical with unrealistic assumptions about wireless interference models and have too strong limitations to be applied directly to real networks. For instance, to achieve the optimal performance requires solving an NP-hard problem in general interference models. As a result, none of the existing solutions have been implemented in real systems.

In this paper, we propose a new congestion control protocol for general purpose wireless multi-hop networks. This protocol is designed with the following informally defined goals: (1) it must support traffic carried by non-deterministic multi-path routing over possibly diverse and many paths as well as single path routing. This means congestion control must be scalable to the number of paths being used, (2) congestion control is a service for diverse applications so that it should not require any changes in application operations such as reliability (e.g., coding) and application-level routing (e.g., opportunistic multi-path or single path), and (3) it must improve efficiency in resource usage to achieve high throughput and fairness in resource sharing among concurrent flows.

In developing our solutions, we borrow heavily from existing theoretical work. Our solution is adapted from differential backlog based backpressure, conveniently called *differential backlog*. The technique was first applied to wireless multi-hop networks by Tassiulas and Ephremides [26] and later used in several follow-up studies [33, 34, 36]. We call our adapted version of differential backlog as *DiffQ*. DiffQ implements router-assisted rate control, source rate control and MAC scheduling over an IEEE 802.11 driver. DiffQ is currently implemented in the Linux kernel and supports various transport protocols including UDP, TCP (without its congestion control), as well as application-level routing supported by MORE and ExOR.

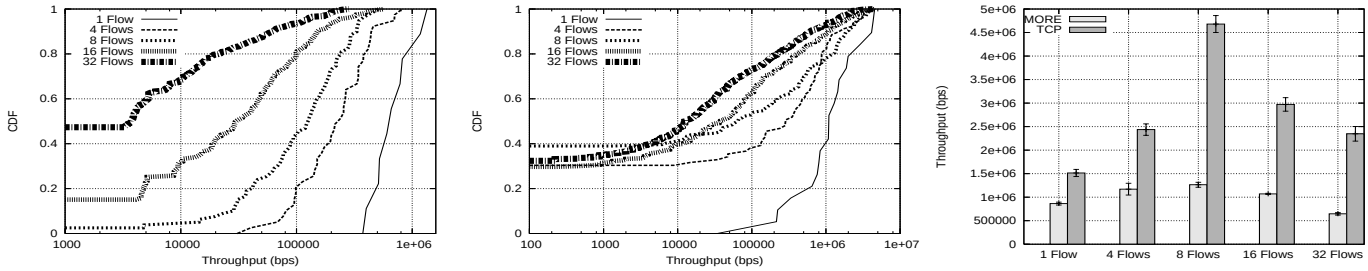
We conduct the evaluation of DiffQ in a testbed of 46 IEEE 802.11b nodes deployed over a 100,000 sq ft building. Our experiments show that DiffQ significantly improves the performance of MORE by 2 to 3 times when tested under 32 concurrent flows of MORE. This gain is possible because the congestion control mechanism of DiffQ avoids congestion collapse and ensures fair sharing of bandwidth among concurrently running MORE flows. The results demonstrate that DiffQ effectively handles multi-path routing used in opportunistic routing and network coding in wireless multi-hop networks. We also compare the performance of DiffQ augmented TCP-SACK and UDP with that of commonly used single-path congestion control algorithms such as TCP-SACK [31], TCP-FeW [32], and TFRC/ECN [13, 12]. TCP-FeW and TFRC/ECN are designed specifically for wireless multi-hop networks; TCP-FeW solves the capacity over-estimation problem of TCP [16] in wireless networks and TFRC is augmented with ECN [12] to remove TCP's dependence on using packet losses as congestion indications. TFRC/ECN uses AQM marking as congestion indications and ignores packet losses. Our experiments show that the DiffQ augmented protocols show superior fairness properties than these protocols while achieving comparable average throughput.

The rest of the paper is organized as follows. Section 2 motivates for our work, Section 3 presents related work, Section 4 and 5 describe the design of DiffQ and implementation details of DiffQ into Linux respectively, and Section 6 discusses the experimental results.

2. MOTIVATIONS

This section examines two key congestion-related performance problems observed in wireless multi-hop networks. The following results are derived from a testbed network of 46 indoor IEEE 802.11 nodes deployed over a three story building of 100,000 sq. ft. space (more detailed description in Section 6). Tests are conducted between random pairs of source and destination nodes The maximum number of hops in our testbed is around 6 or 7 hops. MORE is representative of the multi-path schemes which can benefit from using DiffQ. In each experiment, we increase the number of concurrently running MORE flows. We repeat each experiment 20 times. Figure 1 (a) shows the CDF of per-flow throughput of MORE as we increase the number of flows in each run. We do not employ any congestion control for this experiment.As the number of flows increase, we observe that the number of starved flows increases quickly. With 32 flows, over 50% of flows are completely starved. We also run TCP in the same setup and compare the result in Figure 1(b). TCP also experiences much starvation with about 30 to 40% of flows starving. Figure 1 (c) compares their average per-run total throughput. From these graphs, we find that the problems of these two protocols under high traffic load are quite different. For MORE, since it does not use any congestion control, under higher load, the total throughput of MORE substantially reduces and eventually under 32 flows, not many flows get much utility out of the network; from its CDF of the 32 flow case, we see that over 90% of flows are getting less than 100 kbps. This is a clear sign of congestion collapse. On the other hand, the maximum throughput that TCP flows achieve is almost invariant of the number of concurrent flows running. This shows that TCP experiences extremely unfair bandwidth sharing among its competing flows; while a few flows get a disproportionally larger amount of throughput, many flows are starving.

Figure 2 further illustrates the fairness problem of TCP. In this test, we run three flows as illustrated in Figure 2 (a) where one multi-hop flow (flow 1) starts first and the other two one-hop flows join later. Figure 2 (b) shows the instantaneous throughput of each



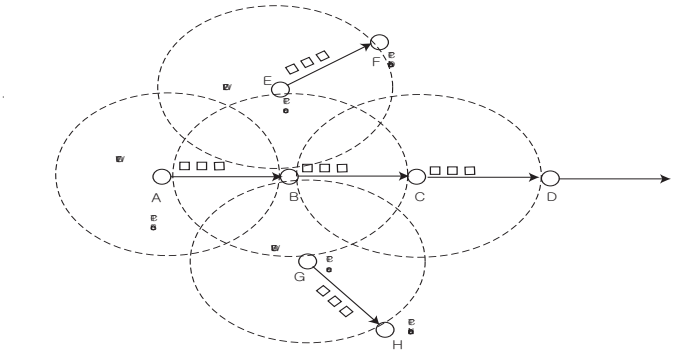
(a) CDF of per-flow average throughput of MORE

(b) CDF of per-flow average throughput of TCP

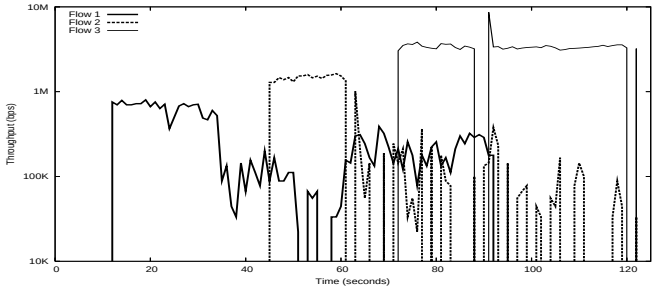
(c) Average per-run total throughput

Figure 1: Congestion problems observed in wireless multi-hop networks. Many concurrent MORE flows running without congestion control experience congestion collapse while concurrent TCP flows experience the fairness problem.

flow over time. As soon as the third flow joins, the first flow quickly starves off. The following explains the reason. While the second and third flows get interference from only the first flow, the first flow does so from the other two flows as well as its own flow being forwarded in different nodes. As a result, the channel accesses of node B interfere with those of four other nodes, A, C, E and G. Thus, the first flow gets less channel accesses especially at node B than the other two flows. This causes congestion at node B. In the mean time, flows 2 and 3 do not get any losses because their destinations are not under any interference. This enables the TCP senders of flows 2 and 3 to increase their rates according to their AIMD algorithm while the TCP sender of flow 1 reduces its rate drastically. This gives the other two flows more chances for channel access and their TCP senders further increase their rates as they interpret the increased channel accesses as increase in available bandwidth. The situation escalates to the eventual starvation of the first flow. This problem is unique in wireless multi-hop networks because in wired networks, when a flow in a congested link receives congestion indications, all the other flows competing in the same link receive a similar amount of congestion indications. In wireless networks, although several flows share the same resource, it is possible that only a subset of those flows get congestion indications depending on their topology. This unfairness problem does not disappear by replacing the congestion indications from losses to delays, ECNs or explicit rate estimation at the link as in ATP [3], EXACT [6], WXCP [39] and ECN[12]. It requires coordinated rate control among competing nodes within the same interference range.



(a) A test scenario; dotted circles represent interference ranges.



(b) Instantaneous throughput of three TCP flows

Figure 2: The fairness problem of TCP in multi-hop wireless networks. We test three flows – one flow over three hops started at time 0 and the two other flows over one-hop started at seconds 45 and 70 respectively. We find that the throughput of flow 1 quickly reduces to zero as flows 2 and 3 join due to the fairness problem.

3. RELATED WORK

3.1 Congestion control for wireless multi-hop networks

Lochert et al. [30] gives an extensive overview on proposals for improving TCP performance in wireless multi-hop networks. Many involve retaining the end-point congestion control of TCP with slight modifications in router functionality to give notifications about link failures (e.g., [19, 29]) and congestion states (e.g., [12, 40, 43]) or to use other congestion metrics such as delays

(e.g., [14, 3]) or explicit rate calculation (e.g., [6, 39]) measured inside routers. Some (e.g., [15, 32]) fix the problems of TCP over-estimation of path bandwidth by reducing the source rate of TCP.

Since all these solutions rely on end-point congestion control, when applied with multi-path routing, they have potential problems with feedback implosion [10] or being too conservative by adapting only to the worst case path condition. Furthermore, these solutions ignore the fairness issues arising from radio interference. Recently, there have been several proposals to fix the fairness problems. Some of them can be applied only to mesh networks or sensor networks where traffic patterns always follow many-to-one transmissions involving one or more gateways or sink nodes (e.g., [17, 37, 20, 38]).

3.2 Congestion control for multi-path routing

Most congestion control studies for multi-path routing are on wired networks with the perspective of load balancing [28, 41, 25]. These solutions apply congestion control at the end-points based on feedback from the routers such as marking or losses that indicate prices of sending a certain rate to a path. Based on the feedback, end-points decide the amount of traffic to send through each path. These techniques cannot be applied to non-deterministic multi-paths where end-points do not have any prior information on which path a packet would traverse at the time of transmission. There are also some multi-path congestion control algorithms for wireless networks (e.g., [18]) but they use multiple paths only for fault-tolerance against link failures.

3.3 Cross-layer optimization

Our solution is inspired by theoretical cross-layer optimization approaches, especially differential backlog backpressure, first proposed for wireless multi-hop networks by [26]. There have been many follow-up studies (e.g., [34, 7, 36, 27, 33]) that use the framework to jointly optimize various components of the protocol stack including scheduling, routing, congestion control, multi-receiver diversity and coding. However, these studies use interference models such as the one-hop interference model to avoid solving an NP-hard problem for the optimal solution. But the interference in real wireless networks commonly affect a multi-hop neighborhood. More important, most of them require the network to be sufficiently loaded to offer the optimization in order to measure queue size differences. In these solutions, when the network is lightly loaded, packets are sent almost everywhere randomly to "create congestion". This incurs high transport delays (see [42]). Since all these solutions optimize for maximizing a function of throughput, the high delays do not affect their optimality. But in practice, delay is an important issue for Internet applications (e.g. web, instant messaging) whose requirement is not real time, but fast response time is a key for their success. Because of combination of the above factors, we do not find any implementation of these solutions in real systems. Recently, Neely and Urgaonkar [34] combine opportunistic routing, and Radunovic et al. [36] combine network coding, into the cross-layer optimization framework. They also propose some practical adaptations of their optimal solutions. Both solutions require a reliable link-level signal plane that transports acknowledgment for each broadcast transmission. Providing reliable link-level acknowledgments for broadcast incurs high overhead in each packet transmission. These solutions are also not completely free from the problems cited above for differential-backlog cross layer optimization. For instance, [34] does not solve packet reordering problems due to the use of diverse paths, and both solutions, to a varying degree, still allow packets to reach many parts of the network that may not lead to destinations just to "create congestion". In general, these cross-layer optimization solutions require changes in existing application-level routing schemes. From the systems perspective, it is of more utility for a congestion control service module to support many application-level protocols without requiring their modifications.

4. DIFFQ DESIGN

In DiffQ, each node maintains a queue for each destination of the flows whose packets it forwards. At the reception of a packet, it can be delivered to the application if the node is the destination or placed into its destination queue in the FIFO order for forwarding to next hops. The queue information and states in a node are soft-state – a destination queue may disappear when there is no packet for that destination.

Each node keeps track of the sizes of destination queues located at its neighboring nodes. When sending a packet, a node piggybacks the queue size of its destination queue where that packet has been queued. This information is overheard by its neighbors. If the next hop is the destination of a packet, then the size of the corresponding queue in that next hop is set to zero. The queue size information is used in a unique way to apply backpressure-based congestion control. Below we first describe congestion control for single-path flows and then for multi-path flows.

4.1 Congestion control for single path routing

We assume that routing is determined by the underlying routing layer and each node knows its next hop forwarding nodes for destinations of received packets. The per-hop congestion control algorithm of DiffQ works as follows. Let us denote $Q_i(d)$ to be the destination queue for destination d in node i. For each destination queue $Q_i(d)$, it maintains the following information every time a new packet is received or overheard from a neighbor

destined to destination d. Suppose that j is the next hop node toward the destination d from node i according to the underlying routing layer.

$$QD_i(d) = |Q_i(d)| - |Q_j(d)| \quad (1)$$

where $|Q_i|$ is the size of queue Q_i. We call $QD_i(d)$ the *queue differential* or *differential backlog* of destination d at node i. For the head-of-line (HOL) packet of each destination queue, we assign a priority which is used for resolving MAC contention in IEEE 802.11 when it is transmitted. At each time that a new packet needs to be transmitted, node i evaluates the priority of the HOL packet of each queue based on its queue differential – the larger the queue differential, the higher priority the packet gets. Since we can only support a finite number of priority levels, we quantize the queue differential value. For simplicity, we use a linear quantization by dividing the queue differential by a fixed interval that is set by dividing the maximum queue size by the number of supported priority levels. Node i chooses the HOL packet of the highest priority among all HOL packets in its destination queues, for transmission next time. Ties are broken arbitrarily. The priority of the HOL packet is used to resolve the channel access. We modified the IEEE 802.11 MadWiFi driver to support prioritized access among competing nodes by adjusting the contention window sizes and AIFS. More details are given in Section 5. It ensures higher chances for channel access to the node with a higher priority packet for transmission. The pseudo code for the source and forwarder are given in *Source Rate Control()* and *Forwarder Algorithm()* respectively.

A source node regulates its flow rate based on its own queue size. Its queue size increases or reduces based on the function of backpressure coming from the network based on our DiffQ scheduling. The rate control must reduce its rate when the queue increases and increase its rate when the queue decreases. As the particular choice of rate control has significant effect on the utility of the system in terms of throughput, it must be carefully designed. In our study, we evaluate AIMD or logarithmic adjustment (which optimizes for the sum of log of per-flow throughput) as used in [8] or [21]. Our pseudo-code of DiffQ below describes a version of AIMD.

Algorithm *Source Rate Control*()
1. F = Destination of flow originating at this node
2. qlen $\leftarrow$$|Q_i(F)|$;
3. **if** qlen $>$ QUEUE_THRESH
4. rate = rate/β;
5. **else**
6. rate = rate + α;

Algorithm *Forwarder Algorithm*()
1. Δ $\leftarrow$Number of priority levels supported by MAC;
2. D $\leftarrow$Maximum per-destination queue size;
3.
4. **Flow Scheduling**
5. F $\leftarrow$argmax$_d$ $QD_i(d)$;
6. P $\leftarrow$HOL packet of $Q_i(F)$;
7. P.priority $\leftarrow$MAX($\lceil \frac{QD_i(F)}{D} \Delta \rceil$, 0); P.qlen $\leftarrow$$|Q_i(F)|$;
8. Transmit P;
9.
10. **On receiving packet** P **from local application**
11. Encapsulate P with DiffQ header;
12. **if** P is the first packet
13. Create flow entry for P's destination;
14. F $\leftarrow$Destination of P;
15. Enqueue P into $Q_i(F)$;
16.
17. **On reception of packet** P **from node** j
18. F $\leftarrow$Destination of P;
19. **if** F is this node
20. Decapsulate DiffQ Header;
21. Send it up to the application;
22. **else**
23. **if** No flow entry exists for F
24. Create flow entry for F;
25. **if** node j is the routing next-hop for F
26. $QD_i(F)$ $\leftarrow$$|Q_i(F)| - |Q_j(F)|$;
27. **else**
28. Enqueue P into $Q_i(F)$;

We now provide the rationale for the above algorithm. DiffQ looks a lot like a scheduling algorithm. But it also has a unique way of applying backpressure. Suppose that a flow f is forwarded through a chain of nodes X, Y, Z and so on in that order, and suppose the size of the destination queue of flow f at node Z is reducing as somehow Z can forward the packets of f fast to its next hop. Then it will cause the queue differential at node Y to increase. This has effect of increasing the forwarding rate at node Y because the channel access priority increases with the queue differential. As Y gets more prioritized access, X will be waiting and its queue builds up. After Y's queue gets depleted, then again X's priority increases because its queue differential against Y's queue is rising, and then X will have a higher chance to the channel next. For the opposite case, suppose that Z's next hop is congested so Z cannot forward its packets. Then Z's queue will build up while increasing its priority. In the mean time, Y's queue differential will reduce because Z's queue is increasing and allows less chance accesses for Y. Consequently, Y's queue builds up. This backpressure will propagate to the source if Z's congestion does not resolve soon enough.

Through backpressure, the size of a destination queue in a node reflects the aggregated condition of all paths from that node to the destination: small queues represent good path conditions on the paths as packets can leave the queue fast and indicate that the paths may be

able to support additional load. Thus, when the destination queue size of the next hop node gets smaller, the priority of the flow being forwarded to that node gets increased, ultimately increasing the flow rate to that node. On the other hand, large queues represent bad path conditions from that node to the destination. This reduces the queue differential of the preceding hop node which reduces its transmission. This condition propagates until the backpressure reaches the source unless the situation improves soon.

The differential backlog scheme improves fairness among neighboring flows although those flows do not share the same routers or links. When a flow is continually denied of transmission at a node in a network path due to contention from its neighbors, then its queue size at that node will increase. This has an effect of increasing the queue differential for that flow at that node, and thus its channel access priority. Consider the scenario in Figure 2. As the one-hop flows send at a high rate, the first flow continues to be denied of channel access. In DiffQ, the destination queue of node B will increase and eventually, get a higher priority than the nodes with single-hop flows (E and G). Thus, B will be allowed to forward the packets of flow 1. DiffQ ensures the nodes with more congestion to have a higher channel access priority so that they can relieve the congestion faster before the backpressure reaches the source. This feature allows DiffQ to enforce fairness among competing flows in wireless multi-hop networks.

DiffQ is different from the queue occupancy based technique used in [22, 20] which sets the contention window size inversely proportional to the queue size, thus giving a higher priority to a node with a larger queue. While it also provides more congested nodes to send packets fast, the difference comes when an area is congested and many neighboring nodes are congested together. In that situation, the queue occupancy scheme allows the node with the largest queue to send packets first whether or not its next hop is congested or not. Thus, it is not clear whether those transmissions will relieve congestion. With DiffQ, those nodes with the largest queue differential make the first transmissions, allowing transmissions to occur always to the direction where congestion can be relieved. This difference is illustrated in Figure 3 where node E has a largest queue. According to the queue occupancy scheme, E will get the highest channel access priority. But in this case, E's transmission only adds congestion already occurring in node F. But in DiffQ, nodes B and F get a channel access first as they have the largest queue differential values within their sensing ranges. This relieves congestion from the locations where it first happens and also backpressure quickly propagates back to the up-stream nodes.

4.2 Congestion control for multi-path routing

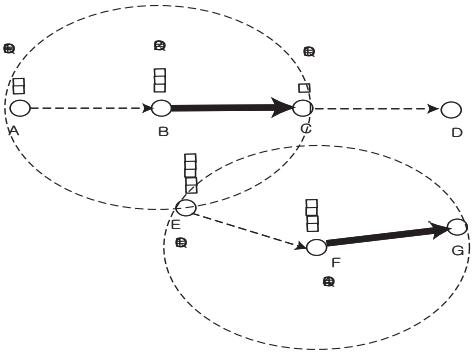


Figure 3: The operation of DiffQ. The thick lines represent channel accesses. According to DiffQ, nodes B and F get the channel access first as they have the largest queue differential values within their sensing ranges.

For the single-path algorithm, we have the information about the next hop router for each packet from the routing layer. For multi-path routing, especially opportunistic routing, each packet does not know the next forwarding node(s); this information is available only after the reception of that packet by the neighboring nodes. One or more neighboring nodes receiving that packet may become the next forwarding routers.

Existing opportunistic routing protocols use source-based routing where each packet contains information about the set of candidate forwarding nodes in the network. This means that at each node i, we can narrow down the possible set of forwarding routers by taking the intersection of the candidate forwarding router list in a forwarded packet p and the one-hop neighbors of that node. Let us call that intersection F_{ip}. When evaluating the priority of an HOL packet p with destination d, each node i computes F_{ip} and then computes the queue differential of that packet with respect to a neighbor $j \in F_{ip}$. The queue differential value, $QD_i(d)$, of p at node i is set to be the minimum of such queue differential values. We use that value for setting the priority of that packet at node i. The following defines this operation.

$$QD_i(d) = \min_{j \in F_{ip}} \{|Q_i(d)| - |Q_i(d)|\} \qquad (2)$$

Node i schedules for transmission the packet with the highest priority (i.e., the maximum queue differential) among all HOL packets and the priority of that packet is used by the MAC layer to schedule the transmission. Each node schedules its transmission based on the worst case next hop router. If there is a queue buildup at any next hop in F_{ip}, then the priority of packet p gets reduced and thus that packet will get a lower channel access priority, which effectively reduces the transmis-

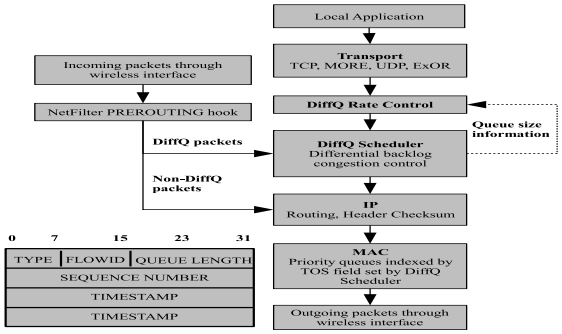


Figure 4: DiffQ Architecture and Packet Header Format - DiffQ sits on top of IP and provides congestion control services to upper layer transport modules. It also controls the MAC priority of packets for scheduling and performs source rate controls for the transport flows (for support of TCP, it disables TCP's congestion control.

sion rate of the flow packet p belongs to. This approach is conservative because the packet may not be received by that worst case next hop.

5. IMPLEMENTATION DETAILS

5.1 Overall architecture

DiffQ has been implemented as a kernel module for the 2.6 series of the Linux kernel (2.6.18 onwards). DiffQ performs queuing and scheduling on every packet being transmitted or received in the Linux networking stack. DiffQ is primarily implemented on top of IP except for prioritized channel access (link layer) and source rate control (transport layer). DiffQ uses the routing support of IP for single path routing while using source routing for multi-path routing in which case, DiffQ bypasses IP routing.

To implement the functions of DiffQ over IP, we need various mechanisms (1) to capture, process and re-inject all IP packets before or after routing in the kernel, (2) to control the MAC priority of each packet transmitted by MAC, and (3) to provide DiffQ information to the transport layer (e.g., queue size information for source rate control). DiffQ uses the Linux Netfilter module to implement the first mechanism, and uses the TOS field of IP header to specify the priority of each packet. The MadWiFi driver is modified to read this field of each packet to get the priority information of the packet and put the packet into appropriate priority queues. The driver outputs packets into the air interface in the order of priority using the queues.

5.2 Interface into IP Layer

We utilize the Netfilter mechanisms of Linux to capture IP packets for DiffQ processing. The Netfilter API provides various hooks throughout the networking stack where packets can be captured from the stack.

We use the PREROUTING hook to trap packets received from a network interface. If the packet has a DiffQ header, then this packet is part of a DiffQ flow and hence we enqueue the packet into its destination queue. If the header is not present then the packet is not a DiffQ flow and is returned back to the stack – this enables our system to handle other IP traffic.

5.3 DiffQ header format

Each DiffQ packet is appended with a DiffQ header containing various fields used by the DiffQ algorithm. We attach the header at the end of data so that the IP layer treats the packet just like an IP packet. This allows the transport header not to be modified since we remove the DiffQ header before passing it to transport layer on the receiver side. The IP header is modified to reflect the new size of the packet. Since we change the size of the packet, we re-compute the checksum before injection to the network stack and delivery to the transport.

The DiffQ header has the following fields (illustrated in Figure 4): The type field contains the type of the flow that this packet belongs to, e.g. we use this field to recognize whether a flow is a source-routed flow like MORE or not. The flowid is the source assigned id for the current flow. Qlen is the length of the packet queue of the destination of the flow. This field is updated by each transmitting node according to the DiffQ algorithm. Timestamp is the timestamp at the source node when this packet was transmitted. Sequence number is used for testing purpose to trace packets.

5.4 DiffQ scheduler

The DiffQ scheduler schedules the queued packets for transmission. The scheduler works as follows: Given that some packets can be transmitted, the scheduler looks into its destination queues and schedules the destination with the highest queue differential. The HOL packet for that destination is then dequeued and re-injected back into the IP stack. Once re-injected, the packet traverses the remaining part of the IP stack to the MAC layer and finally over the air. The MAC layer priority of the packet is loaded into the TOS field of IP header to inform the MAC of the priority to use while transmitting the packet.

5.5 MAC prioritization extension

The packet scheduled by the DiffQ scheduler needs to be transmitted over the air by the MAC with a priority (from the IP TOS field) commensurate with its queue differential; higher the queue differential, higher the priority. We use a linear quantization to allocate packets to one of the priority levels based on the queue

Table 1: 802.11e configuration parameters

Parameter	0	1	2	3	4	5	6	7
AIFS	7	6	6	6	3	3	3	2
CWMin	5	5	5	5	4	3	2	1
CWMax	10	9	8	7	5	4	3	3

differential values as discussed in Section 4. DiffQ uses a modified MADWiFi-NG driver where 8 MAC priority levels have been implemented. This is an extension of the IEEE 802.11e scheme with 4 priority levels. The driver has code that checks the TOS field and puts the packet in the appropriate hardware queue. Most legacy applications do not use IP TOS and hence TOS field is 0 by default. The parameters for various queues (CWmin, CWmax, AIFS) are modified (with values shown in Table 1) to implement MAC priority levels. All non-DiffQ traffic is always sent at priority 0 whose value settings correspond to the default settings of IEEE 802.11b.

6. EXPERIMENTAL RESULTS

6.1 Experimental Setup

Our testbed consists of 46 wireless nodes distributed over three floors in a building of about 100,000 sq ft space. Nodes are PCs with 128MB RAM and 266 MHz processor. Each node is equipped with two Atheros-based 802.11 a/b/g wireless interfaces (AR 5213/5112) connected to omni-directional antennas. All wireless interfaces are configured to operate in the ad-hoc mode with RTS/CTS disabled, transmission power set to 19 dBm and PHY rate set to 11 Mbps. We use an implementation of the OLSR [23] routing protocol which uses ETX [9] to generate routes. We compare the following single path congestion control algorithms:

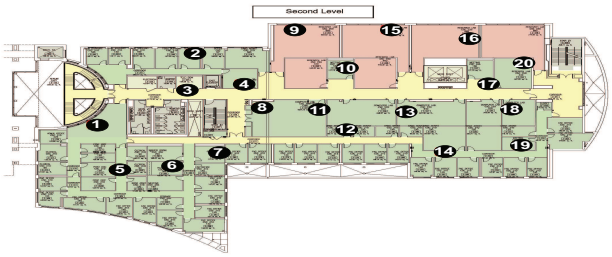


Figure 5: One floor of our testbed with 20 nodes. The full testbed has 46 nodes over three floors.

TCP: We use TCP-SACK [31] as the baseline.
TCP-FeW: TCP-FeW [32] is a variant of TCP proposed for wireless networks. It addresses the congestion window overestimation problem of TCP-Reno style algorithms by reducing their window growth rate. We implement the algorithm in Linux kernel (it is available only in NS-2).
TFRC-ECN: We implement TFRC-ECN where we switch off TFRC's [13] response to packet losses. Instead, intermediate nodes monitor the interface queue size and set the ECN bit in the IP header whenever the queue size exceeds 75% of the maximum queue size. Marked packets within an RTT are treated as a congestion event.
DiffQ-TCP: We implement a TCP-compatible version of DiffQ, where we retain TCP's reliability algorithm but switch off TCP's loss-based congestion control scheme. Instead, the TCP source transmits packets at the rate dictated by the DiffQ source rate control.
DiffQ-UDP: We augment UDP with the DiffQ source rate control. It does not provide any reliability and so its performance is compared to unreliable TFRC-ECN.

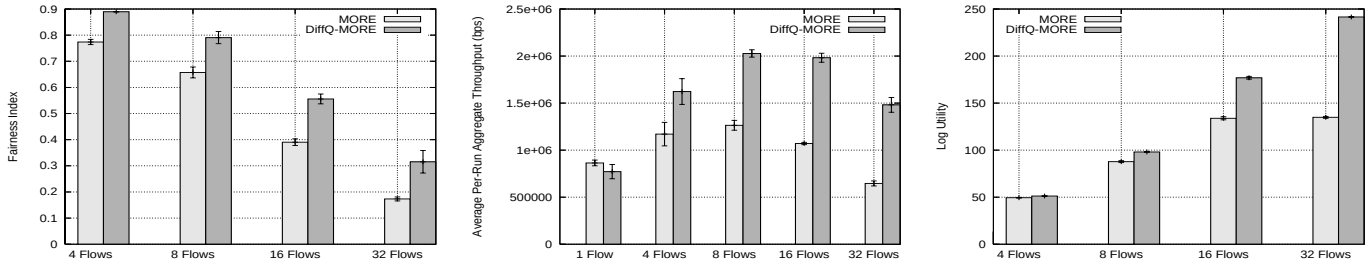
To demonstrate DiffQ's effectiveness in supporting non-deterministic multi-path routing schemes, we use the following algorithms.
MORE: MORE [5] combines network coding with opportunistic routing. We obtained its source code from the authors and ported it over native Linux. It is originally implemented on top of Click [1].
DiffQ-MORE: We apply DiffQ congestion control to MORE as described in Section 4.2. In this implementation the MORE source transmits encoded packets at the DiffQ regulated source rate.

MORE is implemented in the application layer, and MORE packets are delivered to the application at every hop for coding. This particular implementation strategy incurs significantly more performance overheads (context switching, system calls and coding) than TCP implemented in the native Linux kernel. Consequently, the performance of MORE is lower than TCP as shown in Table 2. In this paper, we do not study how much performance enhancement MORE makes over single-path routing. Instead, we focus on evaluating congestion control performance when MORE is combined with DiffQ. Since our purpose is to implement congestion control for opportunistic routing like MORE, the current user-space implementation of MORE is sufficient to serve our purpose.

Table 2: Throughput of MORE and TCP

	1-hop	2-hop	3-hop	4-hop
MORE (Mbps)	1.87	1.25	0.94	0.62
TCP (Mbps)	4.25	2.29	1.45	1.25

We construct scenarios with varying number of concurrent flows. The source and destination of each flow is chosen randomly. For a given number of flows, we construct 20 scenarios with different configurations of sources and destinations. The packet size for all the above algorithms is set to 1400 bytes. MORE and DiffQ-MORE use a batch size of 32 packets. In the



(a) Average per-run fairness index (b) Average per-run aggregate throughput (c) Average per-run log utility

Figure 6: Performance metrics for MORE and DiffQ-MORE.

experiment, we set α and β factors of the DiffQ source rate control to be 0.01 and 0.5 respectively.

The duration of each run is 1 minute. At the end of a run, the throughput of each flow is recorded. We report the mean and 95% confidence intervals of the following metrics. For N competing flows and average per-flow throughput r_i, $1 \leq r \leq N$),

Fairness index per run [2] is given by $(\sum_{i=1}^{N} r_i)^2/(N \sum_{i=1}^{N} r_i^2)$. A fairness index closer to 1 indicates almost equal bandwidth shares among N flows within a run.

Log utility per run is given by $\sum_{i=1}^{N} log(r_i)$. It measures the degree of proportional fairness [11]. This is a utility function commonly used to measure whether a congestion control algorithm achieves high efficiency as well as fairness.

Aggregate throughput per run is $\sum_{i=1}^{N} r_i$. Although the aggregate throughput of a run may be high, the fairness index and log utility for that run may be low, indicating unfair bandwidth allocation among concurrently running flows.

6.2 Multi-path congestion control

The original MORE source [5] broadcasts as fast as it can toward the destination. We saw earlier in Section 2, how this can rapidly lead to congestion collapse, with MORE starving almost 50% of its flows. DiffQ-MORE, however restricts the source rate based on its multi-path backpressure algorithm. The effect of congestion control can be readily seen in Figure 7. The experimental setup is the same as in Section 2.

When there is little or no congestion in the network, e.g. for the case with 1,4 flows, MORE and DiffQ-MORE show similar performance. In fact, the single flow performance of MORE is slightly better than that of DiffQ-MORE. This can be seen in Figure 6 (b) which shows the average per-run aggregate throughput. We attribute this to the fact that DiffQ is conservative in congestion control, it uses the *minimum* queue differential among all its MORE forwarders to decide its outgoing packet priority. This results in some reduction in

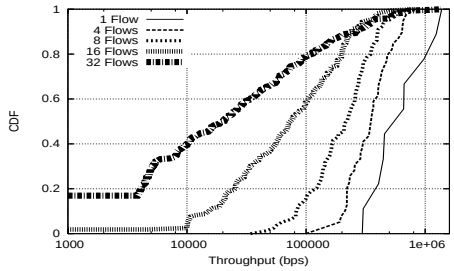


Figure 7: CDF of per-flow average throughput for DiffQ-MORE.

capacity. As the number of flows in the system increase, MORE experiences a dramatic reduction in both average throughput as well as fairness, seen in Figure 6, DiffQ is however able to maintain consistent performance, starving only 18% of the flows in the 32 flow scenario.

Figure 8 shows the normalized throughput difference of DiffQ-MORE and MORE for the 32 flows scenario – for each flow we calculate the difference of throughputs of DiffQ-MORE and MORE and divide it with the throughput of MORE. The x-axis shows the ETX of that path. A positive value indicates a higher throughput for DiffQ-MORE. As the path ETX increases, the throughput difference also increases. This is because MORE selects several forwarders for long lossy paths, and it is precisely such paths which face high contention and congestion. DiffQ-MORE effectively controls congestion among all forwarders for such paths, and hence delivers more throughput.

6.3 Single-path congestion control

We run the same setup as in Section 2 with various single-path algorithms. Figure 9 shows the CDF of per-flow average throughput of DiffQ-TCP. It clearly shows that DiffQ-TCP gets a far less number of starved flows than TCP (see Figure 1 for comparison). We also consider the 3-flow unfair scenario for TCP that we presented in Figure 2. Figure 10 shows the performance

over DiffQ for the same scenario. The routing path for flow 1 is 20→13→12→7→6→5, flow 2 is 4→8→10 and flow 3 is 18→19, on nodes in the Figure 5. Even after all three flows have started, they co-exist and share the medium fairly without starving any one flow. This is due to the DiffQ scheduling based on differential queues.

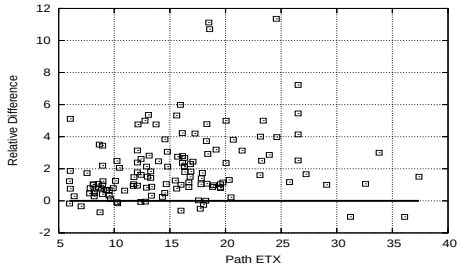


Figure 8: Relative throughput difference between DiffQ-MORE and MORE with respect to path ETX.

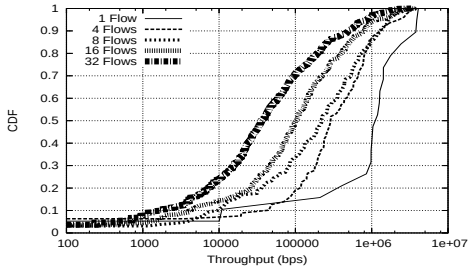


Figure 9: The CDF of per-flow average throughput of DiffQ-TCP. It shows that DiffQ-TCP do not incur much flow starvation.

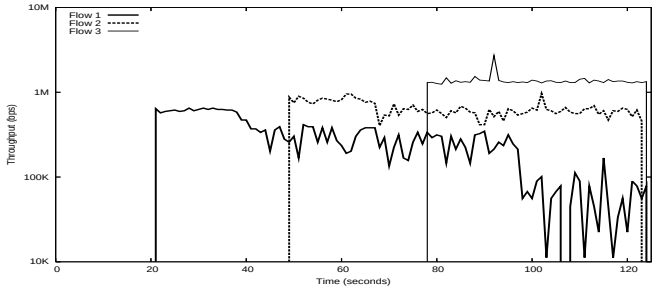


Figure 10: The instantaneous throughput of three DiffQ-TCP flows for the same scenario described in Figure 2.

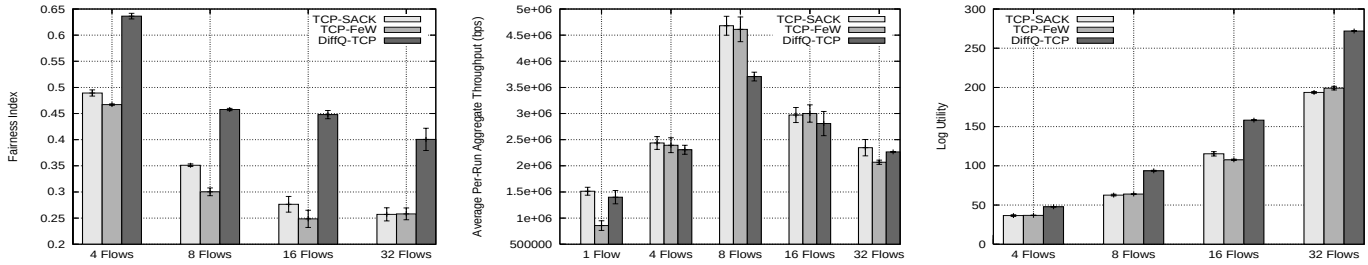
Figure 11 (b) compares the average per-run aggregate throughput of various reliable transfer algorithms under different numbers of concurrent flows. We find that the average throughput of DiffQ-TCP is slightly less than TCP especially under 8 flows, but overall, it achieves comparable average performance to that of TCP and TCP-FeW. On the other hand, the fairness of DiffQ-TCP is much higher. Figures 11 (a) and (c) compare the fairness index and log utility of these protocols respectively. The fairness index is about two times higher for DiffQ-TCP and the log utility is about 20% better. These results are manifested in their median throughput. Table 3 shows the ratio of average median per-flow throughput of DiffQ-TCP over that of TCP.

Table 3: Ratio of average per-run median throughput of DiffQ-TCP over TCP-SACK

Number of Flows	4	8	16	32
Median Ratio: $\frac{DiffQ-TCP}{TCP-SACK}$	1.96	3.45	4.35	8.18

Figure 12 shows a scatter plot comparing flow-by-flow the performance of TCP and DiffQ. For all runs involving 32 concurrent flows, for each source and destination pair, we plot the per-flow throughput of TCP and DiffQ-TCP on a scatter plot. Points on the straight line ($y = x$) indicate that both algorithms got the same throughput over the same path and points on top of the line indicates DiffQ-TCP achieves more throughput than TCP. We can see that for a fraction of flows starved for TCP getting no or less than 1 kbps throughput, DiffQ-TCP achieves about 10 to 100 kbps. Further, TCP has a small fraction of flows around 1 to 5 Mbps but at the cost of a significant fraction of flows around low throughput areas. On the other hand, DiffQ-TCP distributes more evenly around the middle section of plot (around 50 to 500 kbps).

We also find that the performance problem of TCP cannot be fixed by solving only the over-estimation problem of TCP window control [15]. This can be seen from the performance of TCP-FeW which shows about similar performance as TCP. This implies that the solutions for TCP performance problems require much more coordinations of different techniques. On the same note, we find that the unfairness problem of TCP is not completely due to use of packet losses for congestion indications. This can be seen from the performance of TFRC-ECN in Figure 13. We also plot the performance of DiffQ-UDP for comparison. DiffQ-UDP achieves still much better fairness than TFRC-ECN. In this case, DiffQ-UDP also achieves up to 20% higher average throughput. These results imply that even though we remove the effect of losses as congestion indications, TCP-like end-to-end protocols still show significant unfairness to a good fraction of flows. Thus, use of differential backlog using prioritized channel access improves fairness as well as efficiency. This can be visualized from Figure 14 which shows the CDF of per-flow throughput of all the single path algorithms from a particular run involving 32 flows. The figure shows that many flows of TCP and TCP-FeW are starving up to 60% of flows while TFRC-ECN can significantly reduce the number



(a) Average per-run fairness index (b) Average per-run aggregate throughput (c) Average per-run log utility

Figure 11: Performance metrics for reliable single-path congestion control algorithms.

of starved flows, but more than 40% of flows are achieving less than 1 kbps throughput. Since TFRC does not react to packet losses, this indicates that much of TCP starvation is due to heavy losses causing TCP connections to timeout. We also find that these paths where TCP flows are starving can support more than 500 kbps when run without any competing flows.

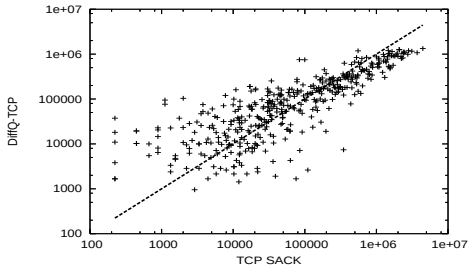


Figure 12: Scatter plot of per-flow throughputs for the 32 flow scenarios.

6.4 Contribution of rate control, scheduling and MAC prioritization

At this point, a pertinent question about DiffQ would be: how much does each DiffQ component (source rate control, differential backlog based scheduling, and MAC prioritization) contribute to the overall fairness and efficiency ? To answer this we construct two variants of DiffQ-UDP: a) *DiffQ-UDP Only RC* – we retain only source rate control and switch off the scheduling and MAC priority components of DiffQ. Flows are scheduled based on a simple round-robin scheduling scheme, and packets are transmitted with priority 0. b) *DiffQ-UDP RC + Scheduling* – we retain both source rate control *and* scheduling, but transmit all packets at priority 0.

We run both variants along with the full DiffQ-UDP algorithm for a single experiment consisting of 16 flows. In Figure 15 we plot the number of queue overflows on intermediate nodes with respect to the path length of the flow (in hops). DiffQ-UDP incurred only about 2500 queue drops over a 2 minute experiment duration, 70% of which occurred along nodes on the 8-hop path. However, both variants of DiffQ-UDP suffer more than 70,000 queue drops over the same period. Source rate control alone is effective only for the 1-hop and 2-hop flows which incur 0 queue drops. The large number of queue drops observed for *DiffQ-UDP RC + Scheduling*, albeit somewhat lower than *DiffQ-UDP RC* indicate that over multi-hop paths, just scheduling flows with higher differential backlog is not enough to control congestion. MAC prioritization is key to flush out the packets for the flows that are scheduled by the scheduling algorithm. This is reflected in the log utilities for this experiment in the same figure.

7. CONCLUSION

DiffQ is a flexible and scalable congestion control algorithm for wireless multi-hop networks. It does not put any restriction on the traffic patterns of flows and can be applied to general ad hoc networks. We implemented DiffQ in the Linux kernel for support of UDP and TCP as well as opportunistic routing such as MORE.

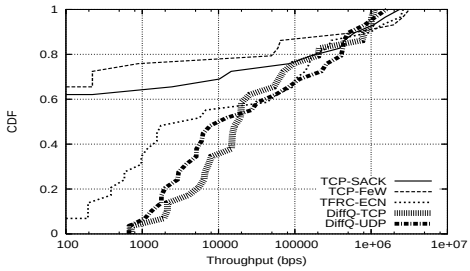
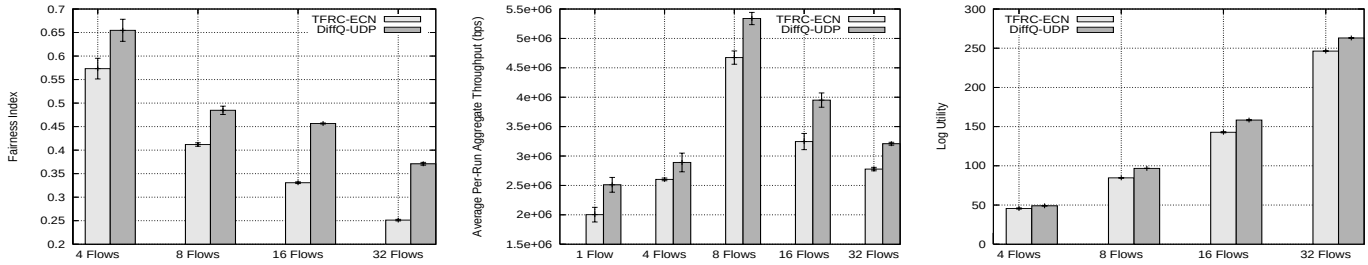


Figure 14: Distribution of throughput of each flow when run together.

8. REFERENCES

[1] http://www.read.cs.ucla.edu/click/.
[2] Jain, r. the art of computer systems performance analysis, first ed. wiley, 1991.
[3] I. Aad and C. Castelluccia. Differentiation mechanisms for ieee 802.11. In *INFOCOM'01*.
[4] V. Anantharaman, K. Sundaresan, H.-Y. Hsieh, and R. Sivakumar. Atp: A reliable transport protocol for ad hoc networks. *IEEE Transactions on Mobile Computing*, 4(6):588–603, 2005.



(a) Average per-run fairness index (b) Average per-run aggregate throughput (c) Average per-run log utility

Figure 13: Performance metrics for unreliable single-path congestion control algorithms.

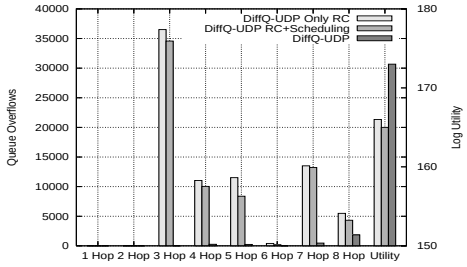


Figure 15: Number of queue overflows in a 2 minute experiment with variants of DiffQ-UDP.

[5] S. Biswas and R. Morris. Exor: opportunistic multi-hop routing for wireless networks. *SIGCOMM Comput. Commun. Rev.*, 35(4):133–144, 2005.

[6] S. Chachulski, M. Jennings, S. Katti, and D. Katabi. Trading structure for randomness in wireless opportunistic routing. In *SIGCOMM '07*.

[7] K. Chen, K. Nahrstedt, and N. Vaidya. The utility of explicit rate-based flow control in mobile ad hoc networks. In *WCNC'04*.

[8] L. Chen, S. H. Low, M. Chiang, and J. C. Doyle. Cross-layer congestion control, routing and scheduling design in ad hoc wireless networks. In *INFOCOM'06*.

[9] D. S. J. D. Couto, D. Aguayo, J. Bicket, and R. Morris. A high-throughput path metric for multi-hop wireless routing. In *MobiCom '03: Proceedings of the 9th annual international conference on Mobile computing and networking*, pages 134–146, New York, NY, USA, 2003. ACM.

[10] D. DeLucia and K. Obraczka. Multicast feedback suppression using representatives. In *INFOCOM'97*.

[11] S. Floyd. Tcp and explicit congestion notification. *ACM Computer Communication Review*, 24(5):10–23, 1994.

[12] S. Floyd, M. Handley, J. Padhye, and J. Widmer. Equation-based congestion control for unicast applications. In *SIGCOMM'00*.

[13] Z. Fu, B. Greenstein, X. Meng, and S. Lu. Design and implementation of a tcpfriendly transport protocol for adhoc wireless networks. In *ICNP'02*.

[14] Z. Fu, P. Zerfos, k Xu, H. Luo, S. Lu, L. Zhang, and M. Geda. On tcp performance in multihop wireless networks.

[15] Z. Fu, P. Zerfos, H. Luo, S. Lu, L. Zhang, and M. Gerla. The impact of multihop wireless channel on tcp throughput and loss. In *INFOCOM'03*.

[16] V. Gambiroza, B. Sadeghi, and E. W. Knightly. End-to-end performance and fairness in multihop wireless backhaul networks. In *MobiCom '04*.

[17] G. Holland and N. Vaidya. Analysis of tcp performance over mobile ad hoc networks. In *MobiCom '99*.

[18] B. Hull, K. Jamieson, and H. Balakrishnan. Mitigating congestion in wireless sensor networks. In *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 134–147, New York, NY, USA, 2004. ACM.

[19] P. Jacquet, P. Mhlethaler, T. Clausen, A. Laouiti, A. Qayyum, and L. Viennot. Optimized link state routing protocol. In *INMIC'01*.

[20] C. Jin, D. X. Wei, and S. H. Low. Fast tcp: Motivation, architecture, algorithms, performance. In *INFOCOM'04*.

[21] S. Katti, H. Rahul, W. Hu, D. Katabi, M. Médard, and J. Crowcroft. Xors in the air: practical wireless network coding. *SIGCOMM Comput. Commun. Rev.*, 36(4):243–254, 2006.

[22] F. P. Kelly, A. Maulloo, and D. Tan. Rate control in communication networks: shadow prices, proportional fairness and stability. *Journal of the Operational Research Society*, pages 237–252, 1998.

[23] P. Key, L. Massoulie, and D. Towsley. Path selection and multipath congestion control. In *INFOCOM'07*.

[24] H. Lim, K. Xu, and M. Gerla. Tcp performance over multipath routing in mobile ad hoc networks. In *ICC'03*.

[25] X. Lin and N. B. Shroff. The impact of imperfect scheduling on cross-layer congestion control in wireless networks. *IEEE/ACM Trans. Netw.*, 14(2):302–315, 2006.

[26] X. Lin and N. B. Shroff. Utility maximization for communication networks with multi-path routing. *IEEE Transactions on Automatic Control*, 51(5):766–781, 2006.

[27] J. Liu and S. Singh. ATCP: TCP for mobile ad hoc networks. *IEEE J–SAC*, 19(7):1300–1315, 2001.

[28] C. Lochert, B. Scheuermann, and M. Mauve. A survey on congestion control for mobile ad hoc networks: Research articles. *Wirel. Commun. Mob. Comput.*, 7(5):655–676, 2007.

[29] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. Tcp selective acknowledgement options.

[30] K. Nahm, A. Helmy, and C.-C. J. Kuo. Tcp over multihop 802.11 networks: issues and performance enhancement. In *MobiHoc '05: Proceedings of the 6th ACM international symposium on Mobile ad hoc networking and computing*, pages 277–287, New York, NY, USA, 2005. ACM.

[31] M. J. Neely. Energy optimal control for time varying wireless networks. In *INFOCOM'05*.

[32] M. J. Neely. Optimal backpressure routing for wireless networks with multi-receiver diversity. In *CISS'06 (invited paper)*.

[33] B. Radunovic, C. Gkantsidis, P. Key, P. Rodriguez, and W. Hu. An optimization framework for practical multipath routing in wireless mesh networks. In *MSR-TR-2007-81, June 2007*.

[34] S. Rangwala, R. Gummadi, R. Govindan, and K. Psounis. Interference-aware fair rate control in wireless sensor networks. *SIGCOMM Comput. Commun. Rev.*, 36(4):63–74, 2006.

[35] A. Raniwala, P. De, S. Sharma, S.Krishnan, and T. Chiueh. End-to-end flow fairness over ieee 802.11-based wireless mesh networks. In *INFOCOM'07*.

[36] I. Rhee, L. Xu, and S. Ha. Cubic for fast long-distance networks, ietf, internet draft, 2007.

[37] Y. Su and T. Gross. Wxcp: Explicit congestion control for wireless multi-hop networks. In *IWQoS'05*.

[38] L. Tassiulas and A. Ephremides. Stability properties of constrained queueing systems and scheduling for maximum throughput in multihop radio networks. *IEEE Transactions on Automatic Control*, 37(12):1936–1949, 1992.

[39] O. Tickoo, V. Subramanian, S. Kalyanaraman, and K. K. Ramakrishnan. Lt-tcp: End-to-end framework to improve tcp performance over networks with lossy channels. In *IwQoS'05*.

[40] C. Wang, B. Li, Y. Hou, K. Sohraby, and Y. Lin. Lred: a robust active queue management scheme based on packet loss rate. In *INFOCOM'04*.

[41] W.-H. Wang, M. Palaniswami, and S. H. Low. Optimal flow control and routing in multi-path networks. *Perform. Eval.*, 52(2-3):119–132, 2003.

[42] A. Warrier, L. Le, and I. Rhee. Cross-layer optimization made practical. In *Broadnets'07 (Invited Paper)*.

[43] K. Xu, M. Gerla, L. Qi, and Y. Shu. Enhancing tcp fairness in ad hoc wireless networks using neighborhood red. In *MobiCom '03: Proceedings of the 9th annual international conference on Mobile computing and networking*, pages 16–28, New York, NY, USA, 2003. ACM.