Enhancing TCP Fairness in Ad Hoc Wireless Networks Using Neighborhood RED [*] [†]

Kaixin Xu, Mario Gerla
UCLA Computer Science Department
Los Angeles, CA 90095, USA
{xkx, gerla}@cs.ucla.edu

Lantao Qi, Yantai Shu
Department of Computer Science
Tianjin University, Tianjin, 300072, China
{ltqi, ytshu}@tju.edu.cn

ABSTRACT

Significant TCP unfairness in ad hoc wireless networks has been reported during the past several years. This unfairness results from the nature of the shared wireless medium and location dependency. If we view a node and its interfering nodes to form a "neighborhood", the aggregate of local queues at these nodes represents the distributed queue for this neighborhood. However, this queue is not a FIFO queue. Flows sharing the queue have different, dynamically changing priorities determined by the topology and traffic patterns. Thus, they get different feedback in terms of packet loss rate and packet delay when congestion occurs. In wired networks, the Randomly Early Detection (RED) scheme was found to improve TCP fairness. In this paper, we show that the RED scheme does not work when running on individual queues in wireless nodes. We then propose a Neighborhood RED (NRED) scheme, which extends the RED concept to the distributed neighborhood queue. Simulation studies confirm that the NRED scheme can improve TCP unfairness substantially in ad hoc networks. Moreover, the NRED scheme acts at the network level, without MAC protocol modifications. This considerably simplifies its deployment.

Categories and Subject Descriptors

C.2.0 [**Computer-Communication Networks**]: General—*data communications*

General Terms

Algorithms, Design

[*]This work was supported in part by Office of Naval Research (ONR) "MINUTEMAN" project under contract N00014-01-C-0016 and TRW under a Graduate Student Fellowship.

[†]This research was supported in part by the National Natural Science Foundation of China (NSFC) under grant No. 90104015.

MobiCom'03, September 14–19, 2003, San Diego, California, USA.

Keywords

TCP Fairness, Ad Hoc Network, Neighborhood RED, NRED

1. INTRODUCTION

In this paper we address TCP performance within a multihop wireless ad hoc network. This has been an area of active research recently, and progress has been reported in several directions. Three different types of challenges are posed to TCP design by such networks. First, as the topology changes, the path is interrupted and TCP goes into repeated, exponentially increasing time-outs with severe performance impact. Efficient retransmission strategies have been proposed to overcome such problems [4, 5, 9]. The second problem has to do with the fact that TCP performance in ad hoc multihop environment depends critically on the congestion window in use. If the window grows too large, there are too many packets (and ACKs) on the path, all competing for the same medium. Congestion builds up and causes "wastage" of the broadcast medium with consequent throughput degradation [7]. The third problem is significant TCP unfairness which has been revealed and reported through both simulations and testbed measurements recently [17, 18, 19].

This paper focuses on the third problem, namely, enhancing TCP fairness in ad hoc networks. Previous work on this topic mostly dealt with the underlying factors causing TCP unfairness. So far, no successful attempts on TCP fairness restoration have been reported. Many specific factors have been identified as the triggers of TCP unfairness, such as: channel capture, hidden and exposed terminal conditions, and the binary exponential backoff of IEEE 802.11 MAC etc [18, 19]. Most of the factors can be traced back to the unfairness of the IEEE 802.11 MAC protocol. However, the "greedy" behavior of TCP and its poor interaction with the MAC layer further exacerbate the unfairness situation [18].

In this paper we argue that two unique features of ad hoc wireless networks are the key to understand unfair TCP behaviors. One is the spatial reuse constraint; the other is the location dependency. The former implies that space is also a kind of shared resource. TCP flows, which do not even traverse common nodes, may still compete for "shared space" and thus interfere with each other. The latter, location dependency, triggers various problems mentioned above, which are often recognized as the primary reasons for TCP unfairness. TCP flows with different relative positions in the bottleneck may get different perception of the bottleneck situation in terms of packet delay and packet loss rate. Since

getting correct feedback of the bottleneck is critical to the fairness of TCP congestion control, limited information of the bottleneck situation causes significant unfairness (e.g. some flows experience more packet loss and thus tend to reduce their congestion window more frequently than others).

If we view a node and its interfering neighbors to form a neighborhood, the local queues at these nodes can be considered to form a distributed queue for this neighborhood. This distributed queue is *not* a FIFO queue. Flows sharing this queue have different and dynamic priorities determined by the topology and traffic patterns due to channel capture, hidden and exposed terminal situations etc. Thus, they get different feedback in terms of packet loss rate and packet delay when congestion happens. The uneven feedback makes TCP congestion control diverge from the fair share. Similar situations may occur in wired networks when a buffer is full and drop tail queue management scheme is used. The Randomly Early Detection (RED) [6] scheme can improve TCP fairness under such situations by keeping the queue size relatively small and dropping or marking packets roughly proportional to the bandwidth share of each flow through the gateway.

In this paper, we propose a Neighborhood RED (NRED) scheme, which extends the original RED scheme to operate on the distributed neighborhood queue. As RED does, each node keeps estimating the size of its neighborhood queue. Once the queue size exceeds a certain threshold, a drop probability is computed by using the algorithm from the original RED scheme. Since a neighborhood queue is the aggregate of local queues at neighboring nodes, this drop probability is then propagated to neighboring nodes for cooperative packet drops. Each neighbor node computes its local drop probability based on its channel bandwidth usage and drops packets accordingly. The overall drop probability will realize the calculated drop probability on the whole neighborhood queue. Thus, the NRED scheme is basically a distributed RED suitable for ad hoc wireless networks.

The rest of the paper is organized as follows. We briefly review previous work in section 2 and describe in short related protocols as well as the simulation environment in section 3. Section 4 presents the concept of the neighborhood of a node and its distributed queue. We then give the detailed algorithms of the NRED scheme in section 5. Section 6 gives verification of our queue size estimation algorithm and guidelines for setting configurable parameters. The usefulness of NRED is also evaluated under simple but fundamental scenarios in this section. Further performance evaluations under more general scenarios are performed in section 7. Some related issues and future work are discussed in section 8 and we conclude the paper in section 9.

2. RELATED WORK

Recently, several techniques have been proposed to improve TCP performance in ad hoc networks. Most of these techniques address mobility, link breakages and routing algorithm failures. Schemes such as ELFN [9], TCP-F [4], Fixed-RTO [5], and TCP-DOOR [22] belong to this category. Together, this work gives reasonable understanding on mobility related TCP inefficiencies. There is, however, another important problem area in wireless ad hoc networks, namely TCP unfairness. This area has received less attention in the past, although the problem is significant. As shown in section 3, 6 and 7, substantial unfairness and even flow starvation exists. This unfair behavior may seriously delay or even lock out a critical application.

Some efforts have addressed the TCP fairness issue in ad hoc networks. In [8, 17], Tang and Gerla *et al* investigated TCP fairness over different MAC protocols, namely CSMA, FAMA, MACAW and IEEE 802.11. In all the investigated scenarios, IEEE 802.11 always came on top in terms of both throughput and fairness. However, even IEEE 802.11 could not achieve acceptable fairness in the ring and grid topologies with TCP congestion window size allowed to grow larger than 1 packet. A simple MAC layer technique was proposed by the authors. An additional yield time was used to restrain the node that used the channel last. It shows improved fairness under the ring topology.

In [19, 20], Xu *et al* investigated TCP fairness over IEEE 802.11 MAC in ad hoc wireless networks. Their work provides a good understanding of the underlying reasons that trigger TCP unfairness. No remedy, however, is proposed in that work. In [18], Gerla *et al* investigated TCP unfairness on a path across a wired and multihop wireless network. Again, they found that the problem resides in the wireless segment. More precisely, they identified hidden and exposed terminals and the interaction of IEEE MAC and TCP congestion control as the key factors that prevent TCP from stabilizing at fair-share.

Most of the prior work is focused on channel and MAC protocol features in an attempt to identify the factors triggering TCP unfairness. However, so far, no complete solution to this problem has yet been reported. In this paper, we attack the problem at the network layer. We explore the relationship between TCP unfairness and early network congestion. RED was helpful in detecting congestion in wired networks and in enhancing fairness. We wish to extend the RED scheme into mobile multihop ad hoc networks. Such an extension is not trivial as ad hoc wireless networks have very unique features such as location dependency.

3. RELATED PROTOCOLS AND SIMULATION PLATFORM

3.1 Random Early Detection (RED)

The proposed NRED scheme is an extension of the RED [6] scheme developed for the wired Internet. RED is an active queue management scheme for congestion avoidance. It monitors the average queue size at each buffer. Once the average queue size exceeds a predefined threshold, it will start dropping or marking packets with given probability. RED has two independent algorithms. One is for computing the average queue size and the other is to calculate the drop probability as a function of average queue size. Suppose current queue size is q, then the average queue size is computed as $avg = (1 - w_q) * avg + w_q * q$, where w_q is called "queue weight" for smoothing the average queue size and tolerating burstness. Upon a packet arrival, the drop probability is calculated according to the minimum threshold min_{th} and maximum threshold max_{th} as $p_b = max_p(avg - min_{th})/(max_{th} - min_{th})$ and $p_a = p_b/(1 - count * p_b)$. Here max_p is the maximum packet drop probability and *count* is the number of packets arrived since last packet drop. When the average queue size is larger than max_{th}, the drop probability is set to 1. RED was shown to improve network performance by avoiding heavy

congestion, eliminating global synchronization etc. It also improves fairness by dropping packets proportional to a connection's share of buffers and thus bandwidth.

3.2 Simulation Environment

This paper and the evaluation of the proposed NRED scheme rely heavily on simulation experiments. Unless explicitly mentioned, all the simulation experiments in this paper use the configurations described here. The simulation platform we used is the QualNet simulator [15], which is the successor of the previous GloMoSim simulation library [21]. Most configuration parameters of the protocol stack in our simulations use the default values. The channel bandwidth is 2Mbps, channel propagation model is the two-ray ground reflection model [16], transmission range is 367m. IEEE 802.11 MAC DCF is adopted. TCP NewReno is used with Maximum Segment Size set to 512 Bytes. The buffer size at each node is 66 packets. Static routing is used in most scenarios. In most of the simulations, TCP connections start at 10s and end at 130s. The simulation time is 150s.

4. TCP UNFAIRNESS AND RED IN AD HOC NETWORKS

4.1 Why RED does not Work?

We first tested whether the original RED scheme can help improving TCP unfairness in ad hoc networks. The simulation scenario is shown in Figure 1. This scenario is very similar to those scenarios used in the wired networks for testing RED scheme. However, it is a wireless specific scenario. The 3 FTP/TCP connections in Figure 1 do not share any common node. However they still contend with each other for the medium. The implementation of RED used in our experiments follows the algorithm presented in [6]. The minimum and maximum queue size thresholds (min_{th} and max_{th}) are set to 5 packets and 15 packets. The queue weight (e.g. w_q) is 0.002. We vary the maximum drop probability max_p from 0.02 to 0.2.

In all the simulations, we found that FTP 2 is always starved. Typical results are given in Figure 2, where the throughput of the 3 connections at the end of simulation is plotted. As a contrast, we also show the throughput when the default "drop tail" queue management scheme is used. Apparently, no visible advantage in terms of improving TCP unfairness can be found when RED is adopted. However one interesting point we observe is that RED still manages to improve the TCP throughput in ad hoc networks. Since this paper mainly focuses on TCP fairness, further discussion of this point is deferred to a follow up study.

Detailed analysis of the results shows that there are 3 major factors that impede RED from improving TCP unfairness in the ad hoc environment. First, a TCP connection which is penalized in channel contention may experience a queue buildup. However, dropping packets of the penalized flow may actually increase the unfairness. Second, congestion does not happen in a single node, but in an entire area involving multiple nodes. The queue at any single node cannot completely reflect the network congestion state. Third, since multiple nodes are involved in the congestion, they should coordinate their packet drops, rather than act independently. Thus, an appropriate RED scheme for ad hoc wireless networks should consider an aggregate of multiple queues in the bottleneck.

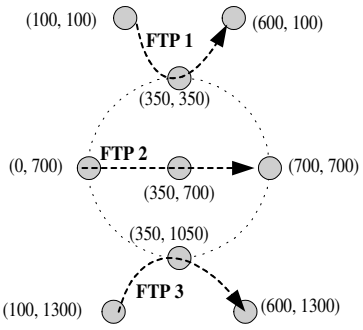

Figure 1: A wireless specific scenario for testing TCP unfairness with RED.

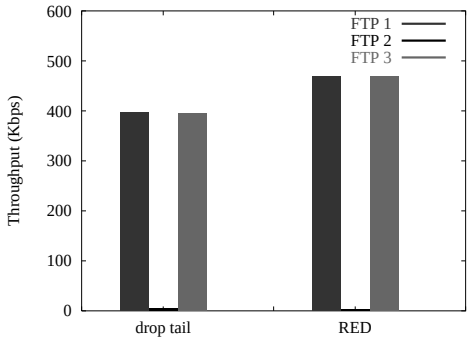

Figure 2: Overall throughput of flows at the end of simulation with RED's max_p equal to 0.06.

4.2 Neighborhood and Its Distributed Queue

In ad hoc wireless networks, there is no pre-defined link between any two nodes. Instead, nodes, which share the same space, compete for the channel under the control of the MAC protocol. Thus, the congestion in an ad hoc network cannot be traced to a specific node, but to the entire space around it. We refer to this space as the "neighborhood" of a node. A more formal definition follows.

Neighborhood: *A node's neighborhood consists of the node itself and the nodes which can interfere with this node's signals.*

Apparently, a node's neighborhood includes both its 1-hop neighbors and 2-hop neighbors. Interference between a node and its 1-hop neighbors is straightforward. For 2-hop neighbors, they may also interfere with the node when they are transmitting to any 1-hop neighbor since collisions may happen at the 1-hop neighbor. Thus 2-hop neighbors indirectly interfere with the node. If we consider the fact that interference range is usually much larger than the transmission range, nodes more than 2-hops away from a node may also be involved in its neighborhood. In this paper, we simplify the problem by ignoring such situations.

In the RED scheme, the early congestion is detected by monitoring the queue size of each outgoing buffer. Similarly, the queue size of a neighborhood reflects the degree of local network congestion. But, where is the queue of a node's neighborhood? In the wired net, there is at least one buffer for each outgoing interface. The queue size can be measured by counting the number of packets in the buffer. However,

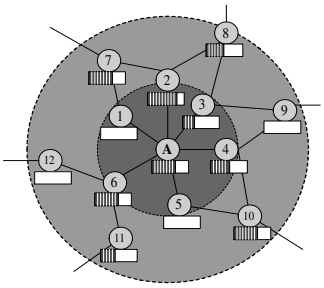

Figure 3: Illustration of a node's neighborhood and its distributed queue.

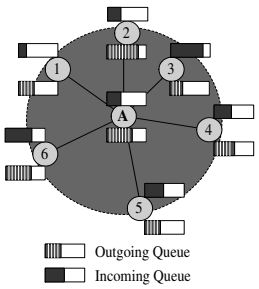

Figure 4: Simplified express of the distributed neighborhood queue.

a node's neighborhood includes multiple nodes each with its local queue. The neighborhood queue should account for all those packets whose transmissions will affect the channel usage within this neighborhood. We view all nodes participating in a node's neighborhood form a distributed queue of that neighborhood with packets scattered among these nodes. The concepts of a node's neighborhood and its distributed queue are illustrated in Figure 3.

In Figure 3, not all packets in the local queue of each node can be considered as packets in the neighborhood queue. Let us examine in detail which packet should be counted. Clearly, the packets in the queues of node A and its 1-hop neighbors are in the distributed queue of node A's neighborhood. Packets in the queues of node A's 2-hop neighbors may or may not be in this distributed queue. More precisely, only the packets directed to a 1-hop neighbor of node A should be considered as a contribution to node A's neighborhood queue size.

4.3 A Simplified Neighborhood Queue Model

The distributed queue illustrated in Figure 3 has 2-hop neighbors involved. Only some packets in the queues of the 2-hop neighbors should be counted. This distributed queue model is not easy to implement and evaluate since it is difficult to get information of 2-hop neighbors without introducing significant communication overhead. We simplify this model to the one illustrated in Figure 4, which only includes 1-hop neighbors. We move those packets in the 2-hop neighbors that are directed to a 1-hop neighbor, to the corresponding 1-hop neighbor. Thus, now each node has two queues. One is its original queue of outgoing packets. The other one is an incoming queue that represents the packets from 2-hop neighbors. The original queue at a node is now referred to as the outgoing queue.

We treat the distributed queue of a neighborhood in an ad hoc network the same way as we would on a single link queue in the wired net and apply RED to it - after proper modifications. This is the basic idea behind the proposed Neighborhood RED scheme. Here, we point out several unique characteristics of the distributed neighborhood queue.

(i) A neighborhood queue consists of multiple queues located at the neighboring nodes that are part of the same spatial reuse constraint set.

(ii) The distributed queue is not a FIFO queue due to location dependency. Instead, multiple sub-queues forming it have different relative priorities in terms of acquiring the wireless channel due to various factors including MAC unfairness, channel capture, hidden and exposed terminal etc.

(iii) The priority of a sub-queue may change dynamically due to topology or traffic pattern changes .

With the concepts of a node's neighborhood and its distributed queue, we can now model TCP unfairness from a transport layer view. As suggested in the previous subsection, multiple TCP flows actually share a distributed queue with different but dynamic priorities. This feature will make TCP diverge from fair-share since they get different feedback in terms of packet drop rate and delay from the congested neighborhood.

5. NEIGHBORHOOD RANDOM EARLY DETECTION

Recognizing the underlying reasons why TCP cannot converge to the fair share, the proposed solution for reestablishing TCP fairness is how to feed the contending flows with the same congestion feedback from the bottleneck (e.g. packet drop probability and packet delay proportional to the share of bandwidth used by each TCP flow). Some form of TCP unfairness, although by far not as dramatic as in the multihop case, manifests itself also in the wired Internet when drop tail queue management scheme is used. The RED active queue management scheme solves that problem by keeping the queue size relatively small and dropping or marking packets of a flow proportionally to its buffer occupancy and thus bandwidth share. This has prompted us to apply a RED-like scheme to the distributed neighborhood queue, which we call Neighborhood Random Early Detection (NRED). To do so, we need to solve 3 problems. 1) How to detect the early congestion of a neighborhood? More precisely, how to compute the average queue size of the distributed neighborhood queue? 2) When and how does a node inform its neighbors about the congestion? 3) How do the neighbor nodes calculate their local drop probabilities so that they add up to the targeted overall drop probability? Correspondingly, three sub-schemes are then proposed, namely Neighborhood Congestion Detection (NCD), Neighborhood Congestion Notification (NCN), and Distributed Neighborhood Packet Drop (DNPD), which are explained in the next subsections.

5.1 Neighborhood Congestion Detection

A direct way to monitor the neighborhood queue size is to let every node broadcast a control packet throughout its

neighborhood to announce its queue size (and destinations of queued packets) upon each packet arrival or departure. By this method, a node can count its neighborhood queue size precisely. However, in a mobile ad hoc wireless network the topology and traffic pattern may continually change. Even if there is no mobility, queue size changes are frequent. A lot of control overhead will be caused by this propagation of queue size information. It is counterproductive to monitor congestion by triggering a lot of extra traffic overhead, which actually worsens the congestion.

Instead of actively advertising queue size information, we opt for a passive measurement technique. Moreover, instead of measuring queue size, we choose an alternate measure related to queue size - namely, channel utilization - which is much easier to monitor than "neighborhood queue size". Naturally, there is a relationship between channel utilization and the size of both outgoing and incoming queues. When these queues are busy, channel utilization around the node is more likely to increase. Now, the trick is to figure out how to measure and account for the various components of channel utilization. To this end, let us carefully examine node A's neighborhood queue shown in Figure 4. When a packet in any outgoing queue is transmitted, node A will detect the medium as busy. If a packet is received to any incoming queue, node A can also learn this through the CTS packet (we assume IEEE 802.11 MAC layer). These two measurements can derive inputs needed for NRED implementation.

More precisely, a node will monitor five different radio states 1) Transmitting, 2) Receiving, 3) Carrier sensing busy, 4) Virtual carrier sensing busy (e.g. deferral to RTS, CTS etc.), and 5) Idle (i.e., no activity on the channel). These radio states can be divided into 3 categories. States 1) and 2) are the contribution of the current node to the total channel utilization within its neighborhood. States 3) and 4) are the contribution of the node's neighbors to the channel utilization. We will assume state 5 means empty queue. This assumption is slightly optimistic, since an idle channel state may also mean all nodes are in backoff stage. In the future, we will improve the scheme by considering more channel statistics such as the collision rate.

By monitoring the five radio states, a node can now estimate 3 channel utilization ratios, namely total channel utilization ratio (U_{busy}), transmitting ratio (U_{tx}) and receiving ratio (U_{rx}). Accordingly, a node constantly monitors the channel condition and records the time period in each of the five radio states. Let T_{tx}, T_{rx}, T_{cs}, T_{vcs} and T_{idle} represent the time period spent at each state during last time period $T_{interval}$. Then the three utilization ratios are defined as:

$$U_{busy} = \frac{T_{interval} - T_{idle}}{T_{interval}} \quad (1)$$

$$U_{tx} = \frac{T_{tx}}{T_{interval}} \quad (2)$$

$$U_{rx} = \frac{T_{rx}}{T_{interval}} \quad (3)$$

Here, $T_{interval} = T_{tx} + T_{rx} + T_{cs} + T_{vcs} + T_{idle}$. U_{busy} reflects the size of the neighborhood queue. U_{tx} and U_{rx} reflect the channel bandwidth usage of the outgoing queue and incoming queue at current node. Later, the local drop probability will be calculated proportional to a node's channel bandwidth usage. When the total channel utilization ratio (U_{busy}) exceeds a certain threshold, we define the neighborhood to be in early congestion.

To facilitate the implementation of the RED algorithm, which is based on packet queue lengths rather than channel utilization, we translate the channel utilization into an index of the queue size. Assume the channel bandwidth is W bps and the average packet size is C bits. We translate the U_{busy} to the queue size index as $q = \frac{U_{busy} * W}{C}$. The variable q is not dimensionally correct, as it is expressed in pkts/sec rather than packets as the queue length should be. However, q is an index that tracks the average queue length, and allows us to manipulate channel utilizations using the conventional RED algorithms. The average packet size C is a constant. Its actual value is irrelevant, as it is only a scaling factor that affects the choice of the values for minimum and maximum threshold (min_{th} and max_{th}). In the rest of the paper, we may still use the term "number of packets" for ease of explanation, where we actually mean this "queue size index".

We translate the U_{tx} and U_{rx} into indices q_{tx} and q_{rx} using the same equation as U_{busy}. After this transformation, we now can apply the original RED scheme. We start by computing the average queue size as $avg = (1 - w_q) * avg + w_q * q$. The initial value of avg is 0. Similarly, we can also get avg_{tx} and avg_{rx} using q_{tx} and q_{rx}. avg_{tx} and avg_{rx} are the average queue size of the outgoing queue and incoming queue (see Figure 4) at current node, which will be used in next two subsections.

The estimation algorithm is independent of the other components of the NRED scheme. Unlike the original RED scheme, which samples queue size on each packet arrival, our scheme samples utilization, and thus "queue size index", periodically. The accuracy of the estimation is controlled by the time interval $T_{interval}$.

5.2 Neighborhood Congestion Notification

Under NRED, a node checks the estimated average queue size avg periodically and compares it with a minimum threshold min_{th}. If queue is larger than threshold, early congestion is detected. Then the node calculates a drop probability p_b based on the average queue size and broadcasts it to its neighbors. Here, we present the algorithm for calculating p_b using pseudocode. The algorithm is based on the original scheme of RED [6].

Algorithm 5.1: CalculatePb()

comment: Procedure to calculate Drop Probability p_b

Saved Variables:
avg: average queue size
Fixed Parameters:
min_{th}: minimum threshold for queue
max_{th}: maximum threshold for queue
max_p: maximum value for p_b
T_{NCN}: time interval for performing this action

for each $\mathcal{T}_{\mathcal{NCN}}$
  $avg \leftarrow estimatedQueueSize()$
  if $min_{th} \leq avg < max_{th}$
    $p_b \leftarrow max_p * (avg - min_{th})/(max_{th} - min_{th})$
    $normalizeP_b \leftarrow p_b/avg$
  else if $max_{th} \leq avg$
    $p_b \leftarrow 1$
    $normalizedP_b \leftarrow 1$

In the above pseudocode, function $estimatedQueueSize()$ gets the estimated neighborhood queue size from channel utilization as discussed in the previous subsection. We also define normalized p_b referred as $normalizedP_b$ by dividing p_b by average queue size avg. This $normalizedP_b$ is the probability which will be broadcasted to neighbor nodes. Based on it, they will compute their local drop probabilities.

If p_b is larger than 0, the neighborhood of this node is in early congestion. In principle, the node should immediately notify the neighbors. However, to avoid "overreaction", we put several conditions before a node broadcasts a congestion notification. These conditions include

(i) The calculated p_b is larger than 0.

(ii) Current node must be on the path of one or more flows. If no traffic goes through it, it should not take any action.

(iii) Current node is suffering in channel contention. We assume mobile nodes are cooperative, yet selfish. Thus, the suffering nodes must speak out to ask neighbors for cooperation. This is realized by comparing the $(avg_{tx} + avg_{rx})$ (see subsection 5.1) with a certain threshold. If $(avg_{tx} + avg_{rx})$ is small, it means current node is suffering in terms of channel contention.

(iv) Current node didn't receive any congestion notification packet in the past interval which contained a larger $normalizedP_b$. Otherwise, that neighbor node's neighborhood is more congested. No need for this node to broadcast a notification since congestion situation may change after some packets are dropped.

Only when all above conditions are met, will a node broadcast a Neighborhood Congestion Notification (NCN) packet to inform neighbors about its congestion situation. The NCN packet must contain enough information for a neighbor node to calculate its local drop probability. In our implementation, the NCN packet includes 3 fields as $<packetType$, $normalizedP_b$, $lifeTime>$. The $packetType$ field indicates that this packet is a NCN packet. $normalizedP_b$ is used for neighbors to calculate their local drop probabilities. The $lifetime$ field indicates the effective duration of this congestion notification. In the NRED scheme, no explicit packet is needed to notify neighbors that congestion has been cleared. A node will simply stop dropping packets after $lifetime$ period. This effective period should be at least twice the NCN broadcast interval T_{NCN} to ensure that a new NCN packet will reach the neighbor nodes if congestion persists.

Once a node receives a NCN packet, it will record the $normalizedP_b$ and $lifetime$ fields of the NCN packet. If multiple NCN packets from different nodes are received, only the packet with the largest $normalizedP_b$ is stored. Both fields are updated when a new NCN packet from the same node as the last stored NCN packet is received. The stored $normalizedP_b$ will be cleared to 0 after $lifetime$ period, if no new NCN packet comes.

From our experiments, we found that broadcast is unreliable when congestion builds up. Thus, it is very likely that the node that is abusing the channel may not receive the NCN packet successfully. To improve the chance that the NCN packet can reach every neighbor successfully, the sender randomly selects a neighbor and "unicasts" the packet to it. Assuming that nodes run in promiscuous mode, all the neighbors can overhear this packet. By using this simple technique, the propagation of NCN packets is much more reliable than with conventional broadcast with more than 20% improvement.

5.3 Distributed Neighborhood Packet Drop

In this sub-scheme, we explain how neighboring nodes cooperatively drop packets to realized the expected drop probability p_b over the distributed neighborhood queue. The key is to calculate a node's local share of this overall drop probability according to its channel bandwidth usage, which has been translated into equivalent queue size for our computational convenience. Suppose a node has received a NCN packet with $normalizedP_b$ larger than 0. Then the local share of p_b at this node should be proportional to its contribution to the average neighborhood queue size avg, which is given as $(avg_{tx} + avg_{rx})$. Thus the share of p_b at this node can be calculated as $p_b * (avg_{tx} + avg_{rx})/avg$. Recall that $normalizedP_b = p_b/avg$, so the local drop probability is calculated as $normalizedP_b * (avg_{tx} + avg_{rx})$.

In our simplified neighborhood queue model, there are two queues at each node, the outgoing queue and incoming queue. Packet drop probabilities will be computed and implemented separately on the two queues. The detailed actions performed on the outgoing queue upon a packet arrival from upper layer is given in pseudocode as below.

Algorithm 5.2: RANDOMDROP()

comment: Actions performed at the outgoing queue

Saved Variables:
$count_{tx}$: *outgoing pkts arrived since last drop*
avg_{tx}: *average outgoing queue size*
Other Parameters:
p_a: *current packet dropping probability*

```
for each packet arrival
  count_tx ← count_tx + 1
  if normalizedP_b < 1
    p_b ← normalizedP_b * avg_tx
    p_a ← p_b/(1 − count_tx * p_b)
   else  p_a ← 1
  if p_a > 0
    aRandomNumber ← random([0, 1])
    if aRandomNumber ≤ p_a
      drop the arriving pkt
      count_tx ← 0
   else  count_tx ← −1
```

In the above pseudocode, $random([0, 1])$ is a function which generates a random number between 0 and 1. When $normalizedP_b$ is equal to 1, it means the average queue size avg has exceeds the maximum threshold. p_a should be 1 no matter what the value $count_{tx}$ is since the drop probability on the neighborhood queue is 1 according to the original RED scheme.

The actions performed at the incoming queue are the same as in the pseudocode above, except that avg_{tx} and $count_{tx}$ are replaced by avg_{rx} and $count_{rx}$. It may be a little confusing that we drop packets from the incoming queue, as the original RED only drops outgoing packets. Referring back to the neighborhood queue model in Figure 3 and its simplified expression in Figure 4, we recall that the incom-

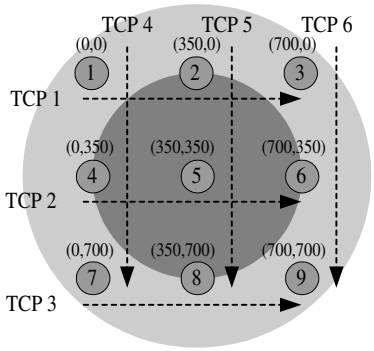

Figure 5: Scenario for verifying queue size estimation algorithm.

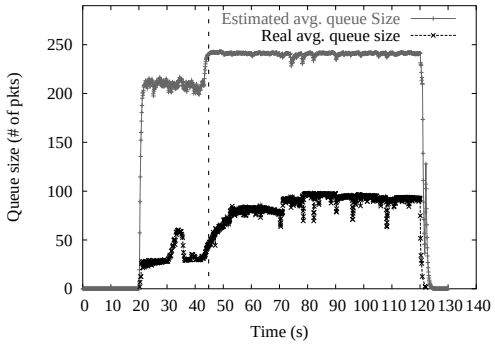

Figure 6: Estimated average queue size and the real average queue size of Node 5's neighborhood under FTP/TCP connections.

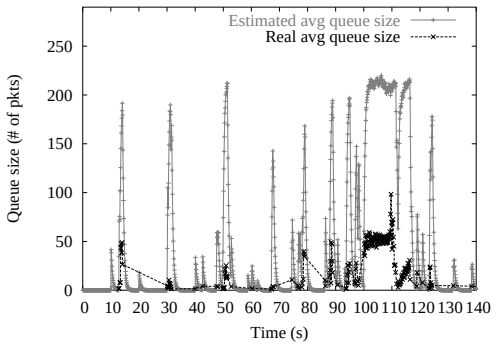

Figure 7: Estimated average queue size and the real average queue size of Node 5's neighborhood under HTTP/TCP connections.

ing queue at a 1-hop neighbor is actually moved from the 2-hop neighbors. Thus, dropping packets in the incoming queue represents the packet drops at corresponding 2-hop neighbors. Since there is no real incoming queue, the actions are triggered each time a packet is received. Dropping incoming packets at the 1-hop neighbors wastes some bandwidth since those packets have consumed bandwidth. To avoid this, a better way would be also propagate the congestion notification to 2-hop neighbors and ask them drop packets correspondingly. However, as we mentioned in section 4.3, coordinating 2-hop neighbors requires non-trivial overhead. First, all 1-hop neighbors must re-broadcast the NCN packets to 2-hop neighbors, much more bandwidth is consumed by such broadcasts. Second, to properly drop packets, a 2-hop neighbor has to know a complete list of 1-hop neighbors of the congested node since it only needs to drop packets intended to the congested area. Propagation of such information will also bring much overhead. Thus, in our NRED scheme, we propose to drop incoming packets at 1-hop neighbors rather than asking 2-hop neighbors to drop them.

6. VERIFICATION AND PARAMETER TUNING

In the proposed NRED scheme, there are several parameters which may affect the performance. In this section, we try to determine their optimal values. Moreover, our scheme for estimating the average queue size of the neighborhood queue is realized by estimating the channel utilization. In this section, we also want to verify that it can indeed approximate the real average queue size. Also in this section, performance of NRED in simple scenarios is investigated.

6.1 Verification of Queue Size Estimation

Recall that NRED is based on channel utilization rather than queue size. In fact, we believe that channel utilization is a more appropriate measure than queue size for our problem. Nevertheless, we are still interested to verify that channel estimation is a good approximation to the real queue size.

We did a series of simulation experiments to verify and validate our queue estimation algorithm. Also, at the same time, we determined the optimal values of parameters related to the estimation. Major parameters are $T_{interval}$ and w_q (see subsection 5.1). The topology of the experiments is given in Figure 5. 9 wireless nodes are involved in the scenario with their coordinates as given in the figure. 6 TCP connections are established. The first TCP connection starts at 20s. The rest of them start one after the other 10s apart. All connections end at 120s and the simulation time is 150s. We record in real time the estimated queue size of Node 5's neighborhood queue. We also record the queue size changes of the local queue at each node and calculate the corresponding real average queue size of Node 5's neighborhood. By comparing the estimated average queue size and the real value, we can then verify the accuracy of our estimation algorithm.

To find the optimal values of $T_{interval}$ and w_q, experiments with different values of $T_{interval}$ and w_q were performed using the same scenario described above. The values of $T_{interval}$ ranged from 1ms to 1s and w_q ranged from 0.02 to 0.8. 20 scenarios were generated using different random seeds, although the figures are not reported in this paper. The conclusion drawn from the experiments is that the optimal values of $T_{interval}$ lie between 100ms and 1s. If it is too short (e.g. shorter than the transmission time of a packet), the estimation is too rugged. If it is too long, it cannot react to channel changes promptly. In practice, this parameter can be set based on the frequency of topology and traffic pattern changes. A smaller value is preferred if the mobility speed is high. The estimation results are not very

sensitive to w_q. The best values are found from 0.1 to 0.4. Larger w_q has better adaption to sudden channel utilization changes caused by mobility or new flows. Smaller values give more smoothly estimation. In our future simulations, we set $T_{interval}$ as 100ms and w_q as 0.2.

Figure 6 and Figure 7 plotted the typical simulation results with $T_{interval}$ and w_q as 100ms and 0.2 respectively. In the simulation of Figure 6, the 6 TCP connections are FTP flows. Since FTP tends to utilize all bandwidth, the neighborhood queue quickly builds up and keeps in a high level during the rest of time. We clearly observe a limitation of our estimation algorithm. After the real average queue size exceeds a certain threshold (e.g. after passed the dotted time line in Figure 6), the estimation cannot reflect future increase of the queue size. Except for this limitation, the estimation algorithm approximately matches the real values. Note, the absolute magnitude of the estimated queue size is meaningless, since we translate the channel utilization to number of packets by dividing it by a fixed packet size. The absolute value only affects the values of the minimum threshold (min_{th}) and maximum threshold (max_{th}).

Since in the FTP case, queue size does not change much. To further verify that our estimation algorithm adaptive to frequent queue size changes, we replaced the FTP flows with HTTP flows. The simulation results are shown in Figure 7. When HTTP traffic is used, the average queue size is small and changes frequently. We observe that our estimation still matches the real values quite well.

Two problems must be justified here. First, channel utilization ratio is past information. Queue size is current state. That means we are estimating the current state based on historical information. Second, when the queue size is too large and channel is fully utilized, further increase of the queue size has no any effect to the channel utilization ratio as observed in Figure 6. Thus, our estimation is bounded to a upper threshold of the queue size. The first problem is not an issue for NRED scheme, as only the average queue size is needed, not the exact current queue size. The original RED scheme also uses a "low pass filter" to average the queue size. For the second problem, it is true that our scheme has this limitation. However, the purpose of NRED is to detect early congestion and keep running the network below heavy congestion by keep the queue size small. Under such conditions, our estimation algorithm should be precise enough.

6.2 Parameter Tuning with Basic Scenarios

In previous subsection, we have verified our queue size estimation algorithm and have tuned related parameters. In this subsection, we focus on another important part of NRED scheme, the calculation of packet drop probability. Once early congestion is detected, the suffering node will notify its neighbor nodes for cooperative packet drops to relieve congestion. The related configuration parameters here are the maximum packet drop probability max_p, minimum threshold min_{th} and maximum threshold max_{th}. We choose the values of min_{th} and max_{th} based on results in the previous subsection. We observe that when there is heavy congestion, the estimated queue size goes around 240 packets. Thus, we choose max_{th} as 240. To get smooth drop probability, we set min_{th} as 100. We then use simulation experiments to decide the optimal values of max_p.

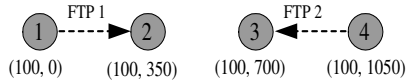

Figure 8: The hidden terminal scenario, where Node 2 is hidden by transmission from node 4 to node 3 and Node 3 is hidden by transmission from node 1 to node 2.

Figure 9: The exposed terminal scenario, where node 2 is exposed to transmissions from node 3 to node 4.

Another purpose of this subsection is to identify how well the NRED scheme can solve the TCP unfairness problem under simple, yet fundamental, scenarios. Besides the scenario in Figure 1 in section 4, another two basic scenarios used are the hidden terminal and exposed terminal topologies as shown in Figure 8 and 9. These two scenarios are first reported in the early MACAW research [3] as basic topologies causing unfairness at the MAC layer. They later are also identified as the major situations where significant TCP unfairness is observed [18, 19]. Thus, any solution for improving TCP fairness should be examined on these basic scenarios.

All the TCP flows start at the same time and last 120 seconds. The simulation time is 150 seconds. The values of max_p tested here is ranging from 0.01 to 0.3. Typical metrics we selected are 1) The overall throughput of each TCP flow. 2) The fairness index and 3) The instantaneous throughput of each flow. The overall throughput is calculated by the TCP receiver at the end of simulation. The fairness index we adopted here is the MaxMin fairness index [2, 10]. Assuming the overall throughput of the two flows are X_1 and X_2, the fairness index is defined as $F(X_1, X_2) = \frac{(X_1+X_2)^2}{2(X_1^2+X_2^2)}$. The MaxMin fairness index is bounded between 0 and 1. The higher the fairness index, the better the fairness. The instantaneous throughput is defined as $X(t) = \frac{D_t}{\Delta_t}$, where D_t denotes the data successfully received during time period $[t \rightarrow t + \Delta_t]$. Δ_t in our simulations is 1 second.

The fairness index and the overall throughput of the two flows under different values of max_p are given in Figure 10 (hidden terminal scenario) and Figure 11 (exposed terminal scenario). In each figure, the upper diagram is the fairness index and the lower one is the aggregated throughput, which is the sum of the overall throughput of the two flows. Before NRED is applied, the two scenarios show very significant unfairness consistent to results reported in the literature [18, 19]. In the hidden terminal scenario, one of the two flows usually achieves much higher throughput (e.g. more than 700Kbps) than the other one (e.g. below 100Kbps). Which one wins the channel is not deterministic. Usually the one which starts slightly earlier wins the channel since its congestion window has chance to grow larger. The unfairness is even more severe in the exposed terminal scenario, where flow 2 always captures the channel and drives the throughput of flow 1 to nearly zero. Detailed analysis of why hidden

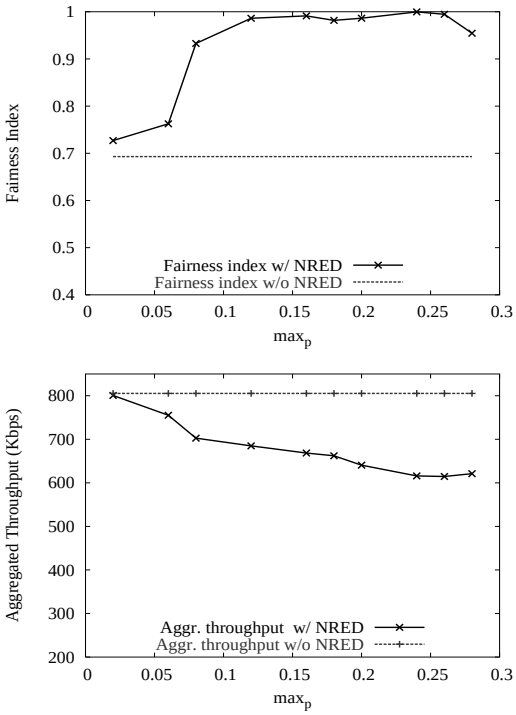

Figure 10: Performance of NRED scheme under the hidden terminal situation with various max_p values

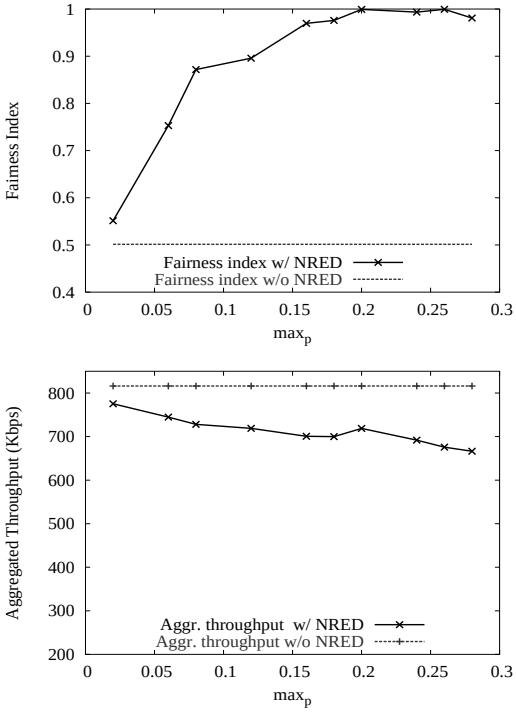

Figure 11: Performance of NRED scheme under the exposed terminal situation with various max_p values

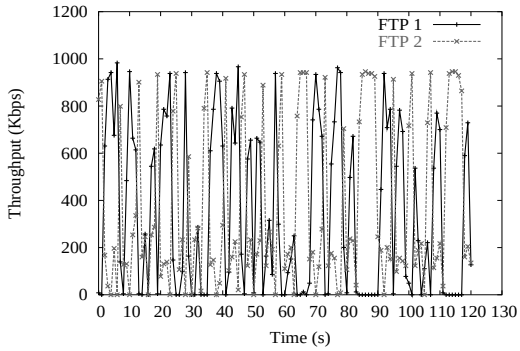

Figure 12: Instantaneous throughput of the two TCP connections under the exposed terminal situation when NRED is applied

and exposed terminal situations cause significant unfairness can be found in [3, 18, 19].

After NRED is applied, we observe that the fairness indices under the both scenarios are improved quickly along with the increase of max_p. For the hidden terminal scenario, the fairness index is close to 1 (the highest value) after max_p is larger than 0.1. For the exposed terminal scenario, fairness index is also above 0.95 when max_p is larger than 0.14. In general, fairness index is increased along with the increase of max_p. This is because larger max_p will punish the flows overusing channel quickly.

Along with the increase of max_p, the aggregated throughput of the two flows is decreased. The throughput loss comes from two reasons. First, before a packet is dropped by NRED, it may have used the channel. Dropping such packets certainly wastes some bandwidth. Second, the NRED scheme tends to keep the wireless channel slightly underutilized. Thus, a small fraction of bandwidth is also sacrificed. To achieve good fairness without too much throughput loss, from above simulation experiments, we conclude that the optimal values of max_p are between 0.1 and 0.2. In later experiments, we set max_p as 0.14.

To further demonstrate how fairly the two flows share the channel when NRED scheme is applied, we plot the instantaneous throughput of the two flows in the exposed terminal scenario with max_p as 0.14 in Figure 12. Results of the hidden terminal scenario are very similar. From Figure 12 we can see that the two flows interlace very well with no any flow continuously uses the channel for long period. Note, only one of the flows can transmit at any time, which explains the up and down of instantaneous throughput.

We then further investigate how the NRED performs under the scenario in Figure 1 in section 4, where the original RED scheme does not help much in terms of improving fairness. The overall throughput of the 3 flows with and without NRED scheme is shown in Figure 13. The fairness is very clear. The starved flow 2 now gains good throughput. Figure 14 further gives the instantaneous throughput of the 3 flows. We can see that short-term fairness is also good (Note, flow 1 and flow 3 can transmit at the same time. But when flow 2 is transmitting, both of them should be quiet.).

Through Figure 13 we also observe that the aggregated throughput of the 3 flows is decreased by 42% when the fairness is improved. This is an interesting point of the

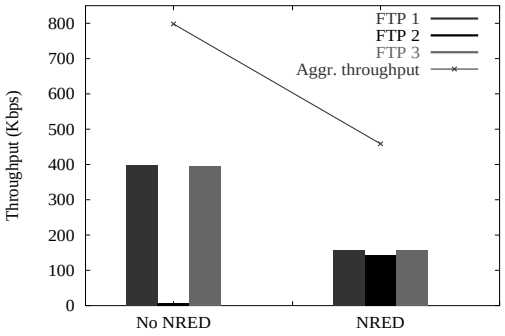

Figure 13: Overall throughput of the 3 flows in the scenario of Figure 1 with and without NRED.

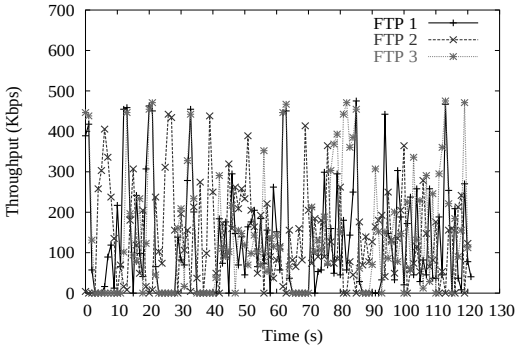

Figure 14: Instantaneous throughput of the 3 flows in the scenario of Figure 1 with and without NRED.

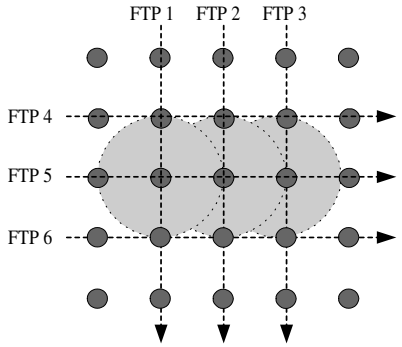

Figure 15: Scenario of the multiple congested neighborhood topology.

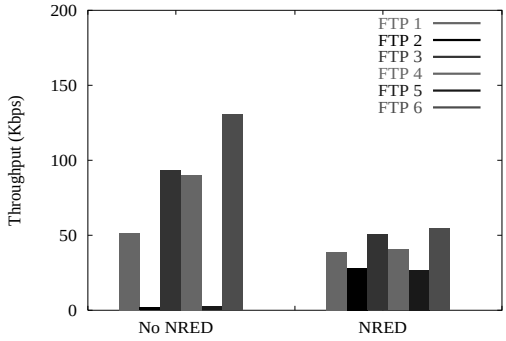

Figure 16: Overall throughput of each flow with and without NRED.

tradeoff between fairness and throughput. Only half the total throughput is achieved during the period of flow 2's transmissions due to the spatial reuse constraint. Thus, to achieve the best network-wide throughput, we should totally starve flow 2. However, this is extremely unfair to flow 2. To achieve fairness among the 3 flows, decreasing the aggregated throughput is unavoidable. More discussion about trade off between fairness and overall network throughput can be found in [12].

7. PERFORMANCE EVALUATION OF NRED

In this section, we further evaluate the proposed NRED scheme under more realistic scenarios such as considering multiple bottlenecks as well as mobility. The major configuration parameters of NRED are set as $w_q = 0.2$, $min_{th} =$ 100 packets, $max_{th} = 240$ packets and $max_p = 0.14$.

7.1 Multiple Congested Neighborhood

In the previous section, most investigated scenarios only contain one bottleneck neighborhood. However, in the ad hoc network, it is highly possible that a TCP flow will traverse several congested neighborhoods, although the degree of congestion may be different. In this experiment, we evaluate how NRED performs under such situations. The experiment scenario is illustrated in Figure 15. The grid topology is used and 6 FTP/TCP connections are established as in the figure. The vertical and horizontal distance between neighboring nodes is 350m. Each TCP connection traverses 4 hops. In this topology, several bottleneck neighborhoods may be present at the same time. In the figure, 3 possible bottlenecks of TCP 5 are shown as gray shaded circles. All 6 TCP connections start at 10s and finish at 130s. The overall throughput of each flow is given in Figure 16.

As we can observe from Figure 16, TCP 2 and TCP 5 are starved originally but gain visible throughput when NRED is applied. The improvement of fairness is obvious. However, TCP 2 and TCP 5 will always achieve less throughput since they have interfering flows on both sides.

7.2 Performance under Mobility

In this experiment, we consider the adaptation of NRED to mobility. Two parameters of NRED scheme control its adaptability to topology changes. The first one is the channel estimation interval $T_{interval}$ which controls how quickly the algorithm can react to sudden channel utilization changes. The second parameter is the $lifetime$ of a NCN packet. Once a node moves out from a congested neighborhood, it will not receive a new NCN packet. But it will continue to mark or drop packets until the $lifetime$ of the old NCN packet expires.

The scenario of this experiment is illustrated in Figure 17. 5 nodes are involved and two FTP/TCP connections are used as shown in the figure. Only node 5 is moving — up and down between positions (200, 600) and (200, 400). Its initial position is (200, 600). The mobility speed is 10m/s. The node stays at each position for around 20s and then continues to move.

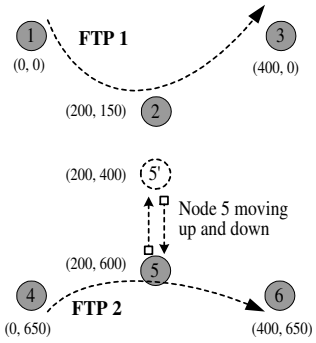

Figure 17: Scenario for evaluating NRED under mobility.

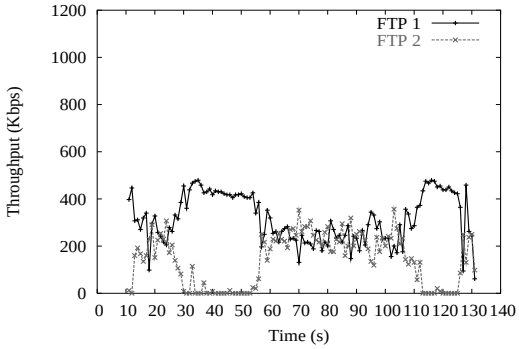

Figure 18: Instantaneous throughput dynamics under mobility without NRED.

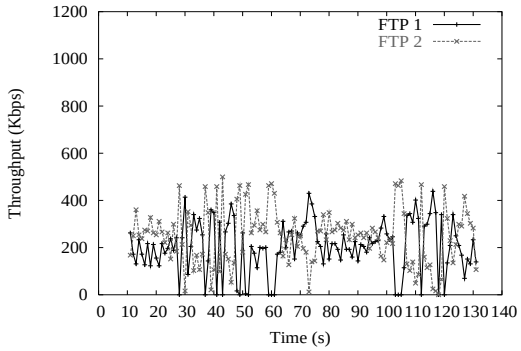

Figure 19: Instantaneous throughput dynamics under mobility with NRED.

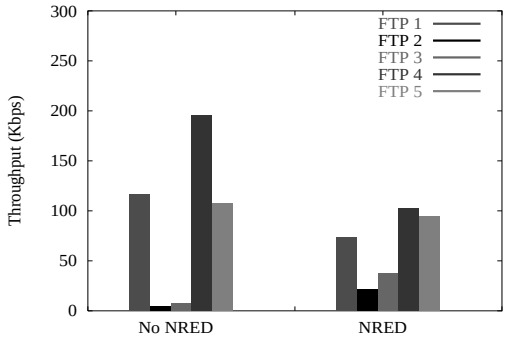

Figure 20: Overall throughput of each TCP connections with and without NRED in the more realistic scenario with random topology, random traffic.

Figure 18 (without NRED) and 19 (with NRED) show the dynamics of the two connections by plotting the instantaneous throughput of each flow. From Figure 18, we observe that when node 5 moves down, the two connections are out of interference with each. Then they both can use the channel well. However, when node 5 moves up, the two connections start interfering with each other. The FTP 1 tends to capture the channel because when node 5 moves up, its distance to node 4 and node 6 is larger. Thus, the transmissions among them are not as robust as the transmissions of FTP 1. Thus it has disadvantages in competing the channel.

When the NRED scheme is applied, node 5 now can detect the congestion after moving close to node 2. From Figure 19, we clearly observe that now the two flows can share the channel fairly when they are close enough to interfere with each other. In the experiment of Figure 19, the estimation interval $T_{interval}$ is set to 100ms and the $lifetime$ of a NCN packet is 2 seconds. The scenario of this experiment is very simple and artificial. However, it clearly demonstrates that the NRED scheme is indeed can adapt to mobility. We didn't evaluate our scheme under random mobility since such situations may distract the main point of the paper. For example, link breaks (path failures) caused by mobility will also result in very low throughput of some TCP flows. However, such low throughput should not be recognized as unfairness (at least not the unfairness issues targeted in this paper).

7.3 More Realistic Scenario

In all simulations above, the topology is simple and specifically selected. The well controlled scenarios allow us to clearly demonstrate the problems and the usefulness of the NRED scheme. However, it is also interesting to take a look at the performance of NRED under more realistic scenarios. In this experiment, both random topology and random traffic are used. More precisely, totally 50 nodes are randomly deployed in a 1000m by 1000m field and 5 FTP/TCP connections are randomly selected. The TCP connections start and end at the same time and last 120 seconds. AODV routing protocol [14] is used here and nodes are not mobile. Different random seeds are used for multiple simulation runs. The results of a typical simulation run are plotted in Figure 20. Other simulations give similar results, but not similar numbers of each specific flow since the traffic pattern is changed due to change of the random seed.

From Figure 20 we can observe that NRED scheme is still able to improve fairness in general, especially reflected by throughput of flow 2 and flow 3. In conclusion, we would like to point out that good fairness does not mean the same throughput for all the participating TCP connections. Two important factors must be considered. First, TCP throughput is highly affected by the number of hops from senders to receivers. It is a well known bias that TCP flows with shorter RTT are usually favored. This bias is strengthened even more in ad hoc wireless networks. The maximum achiev-

able throughput of a 2-hop flow is only 1/2 of that of a 1-hop flow. Similarly, a 3-hop flow can only achieve at most 1/3. This feature is due to the spatial reuse constraint, which has been discussed in detail in [7] and [**?**]. The second factor is that interference is location dependent. Even two flows may contend with each other in a certain location, they are facing different interfering constraints from other locations. In other words, we have the multiple bottleneck effect. It is thus reasonable that flows with fewer interfering flows (i.e., less constraining bottlenecks) achieve higher throughput, which should not be recognized as unfairness.

To sum up, the investigation of TCP fairness in arbitrary topologies and scenarios is extremely complex, especially when nodes are mobile. The NRED method appears to have general applicability in above preliminary experiments. The proper evaluation of NRED in general, multi-bottleneck scenarios requires the definition of new performance measures such as a wireless specific fairness index suitable for multihop ad hoc networks and will be subject of future studies.

8. DISCUSSION AND FUTURE WORK

TCP fairness in multihop wireless networks is closely related to the underlying MAC protocols. Fairness of the MAC protocol has been an active research area in the past several years. Several schemes for improving the fairness of MAC protocols have been proposed in the literature [1, 13]. Although little work has been done to investigate TCP fairness under such fair MAC protocols, we believe improving fairness of the MAC protocol will certainly improve TCP fairness as well.

Naturally, a major problem of MAC layer fairness solutions is implementation difficulty. As the IEEE 802.11 standard has been widely accepted by the industry and IEEE 802.11 wireless radios are becoming the de facto standards for ad hoc testbeds, it is clearly desirable to solve TCP unfairness problems at the network or transport layers. Another important consideration is incremental deployment. Usually, an ad hoc network requires that all involved wireless nodes have a consistent MAC protocol. Thus, any modification of the MAC protocol requires update at all nodes. The NRED scheme proposed in this paper is a network layer solution that can be incrementally deployed. In the worst case, a NCN packet arriving at a non-NRED node will be dropped silently. And as long as some nodes capturing the channel recognize the NCN packet, the unfairness can be improved to some degree.

If we view TCP connections as general traffic flows, TCP fairness is essentially a multihop fair scheduling problem as discussed in [12]. Thus fair scheduling at the network layer for ad hoc networks is quite close to our work in this paper. However, fair scheduling schemes tend to require topology information, in most cases even the global information is needed, for scheduling decisions, which requires additional overhead and is not robust to mobility. Besides, multihop fair scheduling itself is still an active research area without any satisfactory practical solution. In this paper, we utilize the features of TCP congestion control to achieve fairness by reacting to early neighborhood congestion. Compared to fair scheduling, our scheme doesn't need any topology information. It is totally localized with little overhead. We only focus on the bottleneck neighborhood and only the congested nodes need to broadcast a small NCN control packet. In general, fair scheduling will be used for QoS flows, while our scheme is suitable for fairness among best-effort flows. For example, although fair scheduling schemes and QoS schemes probably can solve all problems that RED can, RED gateways still find their important positions in the wired networks due to their simplicity and low overhead.

In current design of NRED scheme, packets of the aggressive TCP flows are randomly dropped at the congested neighborhood. This is not maximally efficient for overall network throughput since those packets have consumed some bandwidth before they reach the congested area. This is also one reason that the aggregated throughput of TCP flows decreases in our experiments when the NRED scheme is applied. An alternative choice would be to explicitly notify the TCP sender to freeze or reduce its congestion window. For example, nodes in the congested neighborhood can mark the Explicit Congestion Notification (ECN) bit of the ongoing TCP packets instead of dropping them. However, such schemes require support from TCP senders and receivers. Directly dropping some packets is the simplest way to notify traffic sources.

In this paper, we mostly evaluated NRED scheme under relatively long-lived TCP flows such as FTP connections transferring a medium or large size file. This is because TCP unfairness issues are more serious to such TCP traffic. If a TCP connection finishes its transfer in seconds, NRED may not have enough time to detect the network congestion and perform proper actions. However, for short-lived TCP flows, even if it captures the channel, it will not hurt other flows too much since it ends very quickly anyway.

Although NRED scheme is targeting improving the fairness of TCP traffic, similar unfairness issues may also happen to non-TCP traffic such as multimedia streaming. The NRED scheme requires the transport protocol to be responsive to network congestion indicated by packet loss. Many TCP friendly or congestion controlled transport protocols have been proposed for ad hoc networks recently. The NRED scheme will also work well with such transport protocols.

Our future research directions include: investing new algorithms for estimating queue size, comparing different choices for congestion notification, investigating fairness among TCP flows and multimedia streaming as well developing an analytical model for neighborhood queue.

9. CONCLUSION

TCP performance is critical to the broad acceptance of multihop wireless networks. In this paper, we proposed a scheme called Neighborhood RED, which is an extension of the RED originally developed in the wired network to ad hoc wireless networks. By detecting early congestion and dropping packets proportionally to a flow's channel bandwidth usage, the NRED scheme is able to improve TCP fairness. The major contributions of this work are the concept of a distributed neighborhood queue (without which the RED scheme does not work) and the design of a network layer solution that does not require MAC modification.

10. ACKNOWLEDGMENTS

We thank our shepherd, Dr. David Maltz (CMU), for helping revise and improve this paper. We also thank the anonymous MobiCom reviewers for providing helpful comments and recommendations.

11. REFERENCES

[1] B. Bensaou, Y. Wang, and C. C. Ko. Fair medium access in 802.11 based wireless ad-hoc networks. *Proceedings of ACM MobiHoc'00*, Aug. 2000.

[2] D. Bertsekas and R. Gallager. *Data Networks*. Prentice Hall, second edition, 1996.

[3] V. Bharghavan, A. Demers, S. Shenker, and L. Zhang. MACAW: A media access protocol for wireless LANs. *Proceedings of the SIGCOMM'94*, Aug. 1994.

[4] K. Chandran, S. Raghunathan, S. Venkatesan, and R. Prakash. A feedback-based scheme for improving TCP performance in ad hoc wireless neworks. *IEEE Personal Communications Magazine*, 8(1), Feb. 2001.

[5] T. D. Dyer and R. V. Boppana. A comparison of TCP performance over three routing protocols for mobile ad hoc networks. *Proceedings of ACM MobiHoc'01*, Oct. 2001.

[6] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, 1(4), Aug. 1993.

[7] Z. Fu, P. Zerfos, H. Luo, S. Lu, L. Zhang, and M. Gerla. The impact of multihop wireless channel on TCP throughput and loss. *IEEE INFOCOM'03*, Mar. 2003.

[8] M. Gerla, R. Bagrodia, L. Zhang, K. Tang, and L. Wang. TCP over wireless multihop protocols: Simulation and experiments. *Proceedings of IEEE ICC'99*, June 1999.

[9] G. Holland and N. H. Vaidya. Analysis of TCP performance over mobile ad hoc networks. *Proceedings of ACM MobiCom'99*, Aug. 1999.

[10] R. Jain, D. M. Chiu, and W. Hawe. A quantitative measure of fairness and discrimination for resource allocation in shared systems". *DEC Technical Report DEC-TR-301*, 1984.

[11] J. Li, C. Blake, D. Couto, H. Lee, and R. Morris. Capacity of ad hoc wireless networks. *Proceeding of ACM MobiCom'01*, Jul. 2001.

[12] H. Luo, S. Lu, and V. Bharghavan. A new model for packet scheduling in multihop wireless networks. *ACM Mobicom'00*, Aug. 2000.

[13] T. Nandagopal, T. Kim, X. Gao, and V. Bharghavan. Achieving MAC layer fairness in wireless packet networks. *Proceedings of ACM MobiCom'00*, Aug. 2000.

[14] C. E. Perkins and E. M. Royer. Ad-hoc on-demand distance vector routing. *Proceedings of IEEE WMCSA'99*, Feb. 1999.

[15] QualNet. Network simulator. *Available at http://www.qualnet.com*, 2003.

[16] T. Rappaport. *Wireless Communications: Principles and Practice*. Prentice Hall, New Jersey, 1996.

[17] K. Tang and M. Gerla. Fair sharing of MAC under TCP in wireless ad hoc networks. *Proceedings of IEEE MMT'99*, Oct. 1999.

[18] K. Xu, S. Bae, S. Lee, and M. Gerla. TCP behavior across multihop wireless networks and the wired internet. *Proceedings of ACM WoWMoM'02*, Sep. 2002.

[19] S. Xu and T. Saadawi. Does the IEEE 802.11 MAC protocol work well in multihop wireless ad hoc networks? *IEEE Communications Magazine*, 39(6), Jun. 2001.

[20] S. Xu and T. Saadawi. Revealing TCP unfairness behavior in 802.11 based wireless multi-hop networks. *Proceedings of IEEE PIMRC'01*, Oct. 2001.

[21] X. Zeng, R. Bagrodia, and M. Gerla. GloMoSim: a library for parallel simulation of large-scale wireless networks. *Proceedings of PADS'98*, May 1998.

[22] Y. Zhang and F. Wang. Improving TCP performance over mobile ad-hoc networks with out-of-order detection and response. *Proceedings of ACM MobiHoc'02*, June 2002.