Enrichment of TCP Fairness in MANETs using FSD Algorithm

Varun Manchikalapudi[(1)], Sk.Khadar Babu[(2)]

Abstract: A MANET is an autonomous collection of mobile users that communicate over relatively bandwidth constrained wireless links. Since the nodes are mobile, the network topology may change rapidly and unpredictably over time. The network is decentralized, where all network activity including discovering the topology and delivering messages must be executed by the nodes themselves, i.e., routing functionality will be incorporated into mobile nodes. As it is decentralized, the problem of fair distribution of available bandwidth among traffic flows or aggregates remains an essential issue in computer networks. Unlike wired networks, some unique characteristics of mobile ad hoc networks seriously deteriorate TCP performance. These characteristics include the unpredictable wireless channels due to fading and interference, the vulnerable shared media access due to random access collision, the hidden terminal problem and the exposed terminal problem, and the frequent route breakages due to node mobility. Undoubtedly, all of these pose great challenges on TCP to provide reliable end-to-end communications in mobile ad hoc networks. This paper focuses on fair distribution of bandwidth over TCP networks preventing packet losses and implementing congestion control for reducing bit error rates.

Index terms- MANET, Bandwidth, Fairness, Efficiency.

1 INTRODUCTION

In this paper we address TCP performance within a multi hop 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. The second problem has to do with the fact that TCP performance in ad hoc multi hop 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.

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 back off of IEEE 802.11 MAC etc.

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. 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.

In this paper, we propose a Fair share distribution (FSD) 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

- *Varun Manchikalapudi is Research Scholar in School of Computing Science & Engineering in VIT University, Vellore, Tamil Nadu, India. E-mail: varunmanchikalapudi@gmail.com*
- *Sk.Khadar Babu is currently working as Sr. Assistant Professor in School of Advanced Sciences in VIT University, Vellore, Tamil Nadu, India. E-mail:khadar.babu36@gmail.com*

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 FSD scheme is basically a distributed RED suitable for ad hoc wireless networks. [1]

The robustness of today's Internet depends heavily on the TCP congestion control mechanism. However, as more and more UDP applications (e.g. packet audio/video applications) are deployed on the Internet, people cannot rely on end users to incorporate proper congestion control. Router mechanisms must be provided to protect responsive flows from non-responsive ones, and prevent "Internet meltdown". Several methods have been proposed for the management of shared resources on the Internet, active queue management is one of the major approaches.

Traffic on the Internet tends to fluctuate and to be greedy. Ideally, a router queue management algorithm should allow temporary burst traffic, and penalize flows that persistently overuse bandwidth. Also, the algorithm should prevent high delay by restricting the queue length, avoid underutilization by allowing temporary queuing, and allocate resource fairly among different types of traffic [1]. In practice, most of the routers being deployed use simplistic Drop Tail algorithm, which is simple to implement with minimal computation overhead, but provides unsatisfactory performance.

To attack this problem, many queue management algorithms are proposed, such as Random Early Drop (RED) [3] , BLUE, Stochastic Fair BLUE (SFB), and CHOKE (Choose and Keep for responsive flows, Choose and Kill for unresponsive flows). Most of the algorithms claim that they can provide fair sharing among different flows without imposing too much deployment complexity. Most of the proposals focus on only one aspect of the problem (whether it is fairness, deployment complexity, or computational overhead), or fix the imperfections of previous algorithms, and their simulations setting are different from each other. These all make it difficult to evaluate, and to choose one to use under certain traffic load.[3]
RED-RED [3] was designed with the objectives to (1) minimize packet loss and queuing delay, (2) avoid global synchronization of sources, (3) maintain high link utilization, and (4) remove biases against burst sources. The basic idea behind RED queue management is to detect incipient congestion early and to convey congestion notification to the end-hosts, allowing them to reduce their transmission rates before queues in the network overflow and packets are dropped.

To do this, RED maintains an exponentially-weighted moving average (EWMA) of the queue length which it uses to detect congestion. When the average queue length exceeds a minimum threshold (min^{th}), packets are randomly dropped or marked with an explicit congestion notification (ECN) bit [2]. When the average queue length exceeds a maximum threshold (max^{th}), all packets are dropped or marked.
While RED is certainly an improvement over traditional drop tail queues, it has several shortcomings.

One of the fundamental problems with RED is that they rely on queue length as an estimator of congestion. While the presence of a persistent queue indicates congestion, its length gives very little information as to the severity of congestion. That is, the number of competing connections sharing the link. In a busy period, a single source transmitting at a rate greater than the bottleneck link capacity can cause a queue to build up just as easily as a large number of sources can. Since the RED algorithm relies on queue lengths, it has an inherent problem in determining the severity of congestion.

As a result, RED requires a wide range of parameters to operate correctly under different congestion scenarios. While RED can achieve an ideal operating point, it can only do so when it has a sufficient amount of buffer space and is correctly parameterized [3].

RED represents a class of queue management mechanisms that does not keep the state of each flow. That is, they put the data from the all the flows into one queue, and focus on their overall performance.

2 RELATED WORK AND THEIR IMPACT

We briefly introduce how the previous research work has overcome TCP unfairness. Some efforts have addressed the TCP fairness issue in ad hoc networks. Tang and Gerla et al investigated scenarios TCP fairness over different MAC protocols. IEEE802.11 always came on top in terms of both throughput and fairness, but it could not achieve acceptable fairness in the ring and grid topologies.

A simple MAC layer technique, an additional yield time was used to restrain the node that used the channel last, is proposed. Xu et al investigated TCP fairness over 802.11 MAC in ad hoc networks. Their work provides good understanding of the underlying reasons that trigger TCP unfairness, but no remedy is proposed in that work. Gerla et al investigated TCP

unfairness on a path across a wired and multi-hop wireless network. They identified hidden and exposed terminals and the interaction of IEEE MAC and TCP congestion control as the key factors that prevent TCP form stabilizing at fair-share.

Instead of focusing on channel and MAC protocol features in an attempt to identify the factors triggering TCP unfairness, [1] and modify RED scheme to solve the TCP unfairness problem.[1] proposed Neighbor RED which extends the RED concept to distributed neighbor-hood queue. Zhenghua Fu et al proposed link RED and adaptive pacing which made link drop probability sufficient to stabilize the large TCP window size. [2]

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, TCP-F, Fixed-RTO and TCP-DOOR 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. 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 Tang and Gerla et al investigation, TCP fairness over different MAC protocols, namely CSMA, AMA, MACAW and IEEE 802.11 is done. 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. Xu et al investigated TCP fairness over IEEE802.11 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. Gerla et al investigated TCP unfairness on a path across a wired and multi hop 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. [1].

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 multi hop ad hoc networks. Such an extension is not trivial as ad hoc wireless networks have very unique features such as location dependency [1].

3 PROTOTYPE IMPLEMENTATION

Perceiving the hidden reasons why TCP can't converge to the fair share, the proposed solution for restoring 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 corresponding to the offer of transfer speed utilized by each TCP flow). Some type of TCP unfairness, albeit by a wide margin not as sensational as in the multi hop case, shows itself likewise in the wired Internet when drop tail line administration plan is utilized. The RED dynamic line administration plan tackles that issue by keeping the line estimate generally little and dropping or stamping parcels of a stream relatively to its cushion inhabitance and along these lines transfer speed offer. This has incited us to apply a RED-like plan to the circulated neighborhood queue, which we call Neighborhood Random Early Detection. 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?

Neighborhood Congestion Detection

An immediate approach to monitor the neighborhood queue size is to let each node broadcast a control packet all through its neighborhood to report its queue size (and destinations of lined packets) upon every packet entry or takeoff. By this strategy, a hub can number its neighborhood line measure absolutely. In any case, in a portable impromptu remote system the topology and activity example might consistently change. Regardless of the possibility that there is no versatility, line size changes are incessant.

A considerable measure of control overhead will be created by this engendering of line size data. It is counterproductive to screen blockage by setting off a

considerable measure of additional movement overhead, which really exacerbates the clogging.

Rather than effectively promoting line size data, we select a latent estimation system. Also, as opposed to measuring line size, we pick a substitute measure identified with line size-to be specific, channel use – which is much less demanding to screen than "neighborhood queue size". Normally, there is a relationship between channel usage and the extent of both active and approaching lines.

At the point when these lines are occupied, channel use around the hub is more inclined to increment. Presently, the trap is to make sense of how to gauge and record for the different segments of channel use. To this end, let us precisely analyze node A's neighborhood line demonstrated in Figure 4. At the point when a bundle in any cordial line is transmitted, node A will recognize the medium as occupied. In the event that a parcel is gotten to any approaching line, hub A can likewise realize this through the CTS bundle (we accept IEEE 802.11 MAC layer). These two estimations can infer inputs required for FSD usage.

FSD develops the first RED plan. Every hub continues assessing the span of its neighborhood line (distributed queue). When the queue size surpasses a certain threshold, a overall drop probability is processed by the calculation of RED. This general drop probability is then proliferated to neighboring nodes for cooperative packet drops.

Distributed Queue of a Node-the outgoing queue of the node itself, 1-hop neighbors' outgoing queues, 2-hop neighbors' packets which are directed to a 1-hop neighbor of node A. Simplified Model is 2-hop neighborhood distributed queue model is not easy to implement and evaluate a lot of control packet overhead. The packets in the 2-hop neighbors directed to a 1-hop neighbor are moved to the 1-hop neighbor. Outgoing queue has the original queue at a node Incoming queue has the packets from 2-hop neighbors as in figure 1.

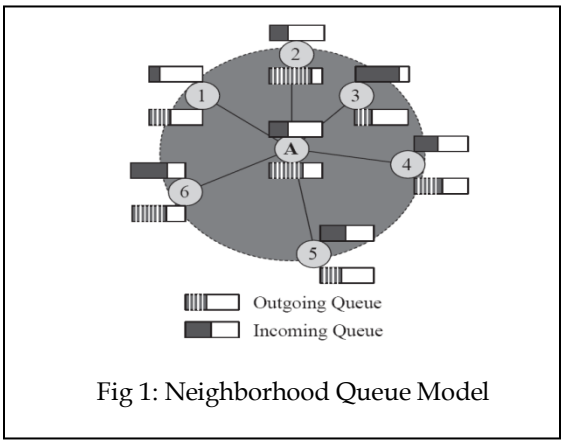

Fig 1: Neighborhood Queue Model

Neighborhood Congestion Detection (NCD): A node monitors five different radio state Transmitting (T_{tx}), Receiving (T_{rx}), Carrier sensing busy (T_{cs}), Virtual carrier sending busy (T_{vcs}), Idle (T_{idle}). By monitoring the five radio states, a node can now estimate 3 channel utilization ratio

Total channel utilization

$U_{busy} = (T_{interval} - T_{idle}) / T_{interval}$

Transmitting ratio $U_{tx} = T_{tx} / T_{interval}$

Receiving ratio $U_{rx} = T_{rx} / T_{interval}$

$T_{interval} = T_{tx} + T_{rx} + T_{cs} + T_{vcs} + T_{idle}$

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.

To facilitate the implementation of the RED algorithm, the channel utilization is translated into an index of the queue size

The queue size index $q = U_{busy} * W/C$

W: channel bandwidth, C: the average packet size

Now the original RED scheme can be applied

The average queue size, avg = (1-wq)*avg + wq*q

If the queue size exceeds a certain threshold, the neighborhood is in congestion.

Drop probability

$Pb = Maxp^* (Avg - Min^{th}) / (Max^{th} - Min^{th})$

Normalized Pb = Pb/avg

Current node A broadcasts Drop probability to 1-hop neighbors.

The broadcast message → drop probability + life time

Neighborhood nodes choose the largest drop probability, if they receive multiple NCN.

4 RESULTS

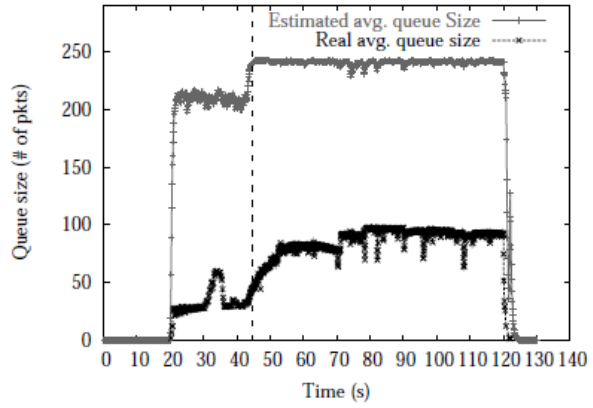

Fig 2: Verification of Queue size

In figure 2, estimated average queue size and the real average queue size of Node 5's neighborhood under FTP/TCP connections.

It is observed that when there is heavy congestion, the estimated queue size goes around 240 packets. Thus, we choose maxth as 240. To get smooth drop probability, we set minth as 100. Simulation experiments are used to decide the optimal values of maxp.

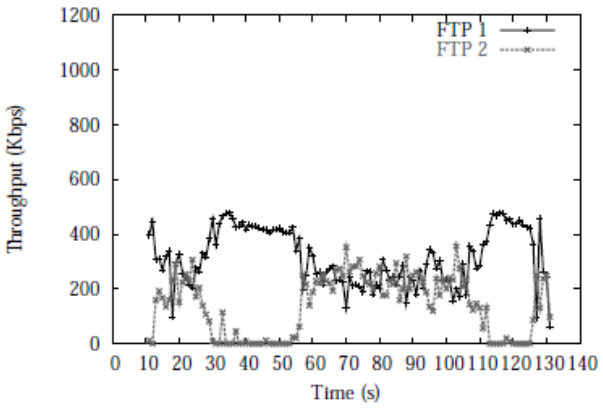

Fig 3: Instantaneous throughput dynamics under mobility without FSD

In Figure 3, the disadvantage of competing the channel is shown.

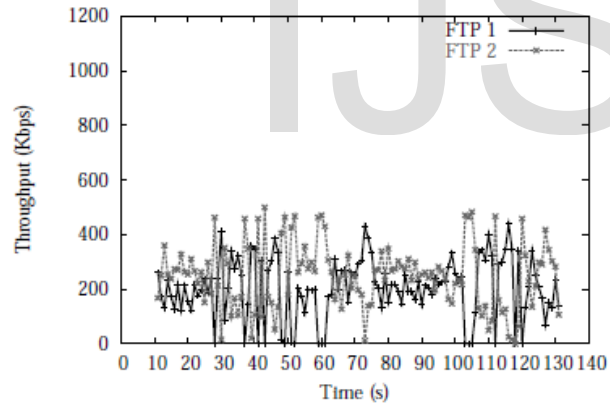

Fig 4: Instantaneous throughput dynamics under mobility with FSD

In figure 4, it is shown that the two flows can share the channel fairly when they are close enough to interfere with each other.
It clearly demonstrates that the FSD scheme is indeed can adapt to mobility.

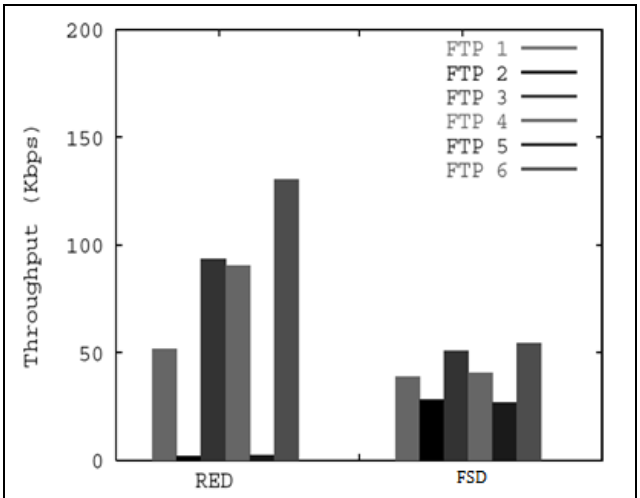

Fig 5: Overall throughput of each TCP connections with and without FSD.

In figure 5, we can observe that FSD scheme is still able to improve fairness in general, especially reflected by throughput of flow 2 and flow 3. First, TCP throughput is highly affected by the number of hops from senders to receivers.

5 CONCLUSION

TCP performance is critical to the broad acceptance of multi hop wireless networks. In this paper, we proposed a scheme called FSD, which is an extension of the RED originally developed in the ad hoc wireless networks. By detecting early congestion and dropping packets proportionally to a flow's channel bandwidth usage, the FSD scheme is able to improve TCP fairness.

REFERENCES:

[1] Enhancing TCP Fairness in Ad Hoc Wireless Networks Using Neighborhood RED by Kaixin Xu, Mario Gerla, Lantao Qi, Yantai Shu
[2] Network Coding for Improving the Fairness of Long-Hop TCP Flows in a Multi-Hop Wireless Network by Yong Oh Lee, Manish Kumar Singh
[3] Evaluation of Queue Management Algorithms by Ningning Hu, Liu Ren, Jichuan Chang Computer Networks.