On TCP Performance in Multihop Wireless Networks

Zhenghua Fu, Petros Zerfos, kaixin Xu, Haiyun Luo, Songwu Lu, Lixia Zhang, Mario Gerla
Computer Science Department,
University of California at Los Angeles
{zfu,pzerfos,xkx,hluo,slu,lixia,gerla}@cs.ucla.edu

Abstract

This paper studies TCP performance over static, ad hoc networks that use IEEE 802.11 protocol as the access method. Our study reveals some interesting results. First, there exists an optimal value for TCP congestion window size, at which the TCP throughput is maximized. However, TCP does not operate around this optimal point, and typically grows its window much larger; this leads to decreased throughput and increased packet loss. To better understand this behavior, we further study the characteristics of TCP packet loss. Our results show that, network overload is mainly signified by wireless link contention. As long as the buffer size at each node is reasonable (larger than 10 packets), buffer overflow-induced packet loss is rare and packet drops due to link-layer contention dominate. Link-layer drops offer the first sign for network overload. We further observe that multihop wireless links collectively demonstrate Random Early Detection (RED) like graceful drop behavior, but the current TCP protocol does not adapt well to this built-in grace drop characteristic. We further propose two techniques of link RED and adaptive spacing at the link layer, and simulations show that they can improve TCP throughput by 5% to 30%.

1 Introduction

Wireless ad hoc networks offer convenient infrastructure-free communication over shared, multihop wireless channels. Following a peer-to-peer networking model, they enable a group of networking devices to communicate among one another over error-prone wireless links. Network applications often require reliable data delivery, and must depend on a reliable transport protocol for this purpose. TCP has successfully fulfilled this requirement in the Internet domain, its design has been well tested over the years and a large base of applications already makes use of it. For these reasons, TCP also seems to be the natural choice for users of ad hoc networks that want to communicate reliably with each other and even with the rest of the Internet.

TCP is an adaptive transport protocol that controls its offered load according to the available bandwidth of the underlying network. For wireless ad hoc networks, such as those based on the IEEE 802.11 protocol – the *de facto* access method, this characteristic is of particular importance; the wireless channel is a scarce resource, has limited capacity and is shared by multiple users, thus improving its utilization (through increasing channel spatial reuse and reducing packet loss) is highly desirable.

Earlier research on TCP over ad hoc networks investigated the mobility-induced factors on TCP performance, such as link breakage and routing failures [2, 3]. However, the impact upon TCP of sharing the wireless medium has not yet been sufficiently addressed. The goal of this work is twofold: First, we seek to provide a comprehensive understanding of TCP performance over multihop wireless networks, by studying the behavior of throughput and the evolution of packet loss. Then, based on the findings of this study, we describe an initial design effort to improve TCP performance and achieve higher channel utilization.

The analysis of TCP performance over static, multihop wireless networks, reveals some interesting results. First, there exists an optimal value for TCP congestion control window size, at which the throughput is maximized. However, over ad hoc wireless networks, the standard TCP protocol does not operate around this optimal point, and typically grows its window much larger. As a consequence, the network experiences considerable packet loss, and its throughput decreases. Our findings show that by carefully tuning the protocol parameters, an improvement in the order of 5% to 30% can be readily achieved.

TCP buffer loss and link-layer packet drops also exhibit an interesting pattern, which can prove helpful as an indication of network overload. As long as the buffer at each node/router is not too small (larger than 10 packets), buffer overflow-induced packet loss is rare and packet drops due to link-layer contention dominate. We further observe that

multihop wireless links collectively demonstrate RED[1] -like graceful drop behavior, but the current TCP protocol does not adapt well to this built-in drop characteristic. From the above, we see that contention loss[2] offers the first sign of network overload, and provides a way for TCP to identify its optimal operating window size.

The previously made observations also shed some light on how to improve TCP performance over multihop wireless networks. In this paper, we propose two link layer techniques that improve TCP efficiency: a Link-RED algorithm to tune the wireless link's drop probability, and an adaptive link-layer pacing scheme to increase the spatial reuse of the channel. The goal is to let TCP operate in the *contention avoidance* region.

To summarize, this paper makes the following two important contributions:

- Through a combination of analysis, simulations, and real experiments, an extensive study of TCP performance over multihop wireless networks is presented, and the reasons behind suboptimal behavior are identified.

- Two techniques at the link-layer that improve performance by a factor of 5% to 30% are described and evaluated.

The rest of the paper is organized as follows. Background information, including a brief introduction to the 802.11 MAC protocol, is provided in section 2. A thorough study of the TCP throughput on various topologies and traffic patterns follows in section 3. Section 4 provides more insights on the packet loss of the TCP protocol and the link-layer. In section 5, mechanisms that improve the TCP performance are proposed and evaluated. Discussions of several important issues follow in section 6. Related work is presented in section 7 and the paper concludes in section 8.

2 Background

We consider a static, multihop, wireless ad hoc network, in which every node also acts as a router, forwarding packets originated from other nodes. A single radio channel is shared for wireless transmissions, and only receivers within the transmission range of the sender can receive the packets. The IEEE 802.11 Distributed Coordination Function, the *de facto* access method used in ad hoc networks, serves as the wireless MAC protocol. In 802.11, each packet transmission is preceded by a control handshake of RTS/CTS messages. Upon overhearing the handshake, the nodes in the neighborhood of both the sender and the receiver will defer their transmissions, to ensure the successful completion of the subsequent data packet transmission.

Failures in the transmission of control and data packets are usually caused either by the effect of the Hidden Terminal problem [12, 18], or by channel errors. Specifically, the sender drops the DATA packet after re-sending the RTS message seven times and does not hear a CTS reply from the receiver. DATA packet is also dropped after four retransmissions without receiving an ACK.

The locality of wireless transmissions implies that contention for the shared medium is location dependent. A collision occurs when a receiver is in the reception range of two simultaneously transmitting nodes, thus unable to cleanly receive signal from either of them. In such cases, link-layer *contention loss* happens when packets are dropped due to collisions or failed handshakes. The location-specific nature of contention, coupled with the multi-hop nature of the network, also allows for spatial channel reuse. Specifically, any two transmissions that are not interfering with each other can potentially send data packets over the physical channel simultaneously, which improves aggregate channel utilization. Figure 1 illustrates an example for contention loss and spatial reuse, in which pairs of A-B and E-F may transmit simultaneously, but simultaneous transmissions from pairs of A-B and C-D will collide.

3 TCP throughput study

In this section we study the throughput of TCP over several topologies of ad hoc networks. Since the shared, wireless medium is bandwidth-constrained, we are interested in the conditions under which TCP maximizes the wireless channel utilization. For this purpose, using simulations, analysis and real experiments, we examine TCP throughput with respect to its window size, and an optimal value for this window is identified. This optimal value results in maximum TCP throughput, which increases channel spatial reuse, and minimizes at the same time the link-layer contention. The section concludes with an analysis of the throughput that TCP achieves during the steady state.

3.1 TCP optimal window size – chain topologies

To give insight on the TCP throughput behavior, we begin our study by presenting simulation results on a 7-hop chain, as well as a more general chain topology, consisting of h hops (h-hop). Then, an analysis on TCP optimal congestion window size in the h-hop chain is provided, and a

[1] Random Early Detection (RED) is a congestion control mechanism used in the Internet. It starts to gradually drop incoming packets as the queue size builds up.

[2] The term "contention loss" is used to denote packet drops that are due to link-layer contention, as it will be further described in section 2.

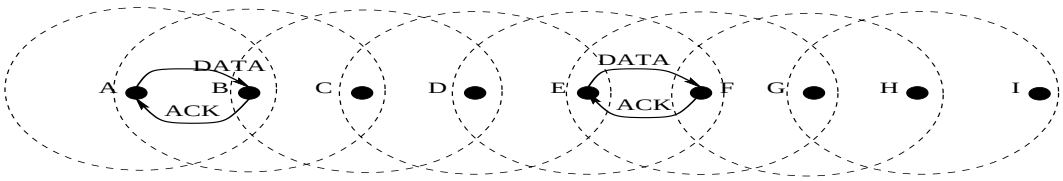

Figure 1. *Spatial reuse and contention.* An example of 8 hop chain. Note that the optimal spatial reuse is achieved when nodes A, D and G transmit simultaneously first and then B, C, and E follow.

set of real experiments verifies the simulation and analytical results.

3.1.1 Simulation results

For our simulation study, we use the *ns-2* simulator with CMU wireless extensions. The parameters are tuned to emulate the AT&T Lucent wireless card, operating at 2Mbps. In the physical layer model of *ns-2*, the effective transmission range of each node is set to 250 meters, and the carrier sensing and interfering range are set to 550 meters. We also ignore the impact of channel errors, since, as shown in [9], the seven retransmissions provided by 802.11 are sufficient in order to recover from typical channel errors. To minimize the significant effect of routing protocol dynamics on TCP [2, 3], we use manually pre-configured source routing, to set up the TCP data path. We choose TCP NewReno for our studies, which is an improved version of TCP Reno that handles more efficiently multiple packet losses in the same window. Comments on other TCP versions will be discussed in section 6. Table 1 shows the default settings used in our simulations.

Our experiments start with a 7-hop chain topology. We vary the maximum TCP window size and observe the corresponding throughput at the given window. To this end, we artificially bound the maximum *allowed* sender window size $MaxWin$ for TCP, in the range of 1 to 32 packets. This is equivalent to enforcing flow control in TCP, where $MaxWin$ plays the role of advertised receiver window size. At each $MaxWin$, we run one TCP flow over such network, for 300 seconds. Figure 2 (left) plots the TCP throughput vs. the $MaxWin$ value. The results show that TCP achieves the best throughput when $MaxWin$ is set to a value of 3 in a 7-hop topology. When $MaxWin$ is in the range of 1∼3, the throughput grows almost *linearly* with $MaxWin$. When $MaxWin \geq 3$, TCP throughput starts to decrease but stabilizes eventually. In figure 2, results for three different packet sizes – 576B, 1024B and 1460B – are also presented; it is clear that TCP achieves its best throughput at a window size of 3, irrespective of the packet size.

We now generalize our simulations to the case of an h-hop chain, by varying the hop count from 3 up to 48. Figure 2 (middle) presents the optimal TCP window size as a function of the hop count. As we observe from the straight line of this figure, which will be proven through the analysis following in the next section, the optimal window size can be well approximated by $\frac{1}{4}h$ in the h-hop chain, as the chain becomes longer. Figure 2 (right) also plots TCP throughput at the optimal window size $\frac{1}{4}h$, for different values of hop count. From the graph, it is clear that TCP optimal throughput tends to decrease, as the length of the chain increases. As a final remark, in all cases, the measured value does not deviate from the predicted one by more than 2 packets.

3.1.2 Analysis

At this point, we provide a simple, yet insightful analysis on the optimal congestion window size W^* of TCP, in an h-hop chain. The analysis makes several ideal assumptions, however it does capture important physics of the TCP performance. One assumption is that the distance between any two *neighboring* nodes in the chain is close to the maximum transmission range of 250 meters. Furthermore, for the purpose of illustrating the analysis, a global scheduler, which perfectly coordinates and synchronizes transmissions at each node, is assumed to be available. In what it follows, we will elaborate on the 8-hop chain of figure 1.

Consider the interference range as 550 meters. It is easily noticeable from the figure that maximal spatial reuse is achieved if the scheduler adopts the following policy: nodes A, E simultaneously transmit first, followed by concurrent transmissions of B, F, then of D and H, and finally, nodes C and G transmit at the same time; then, the same process repeats. In general, all nodes that are 4 hops apart in the chain can send their packets simultaneously, thus maximizing the spatial channel reuse. In essence, this translates to each node being able to transmit one packet every 4 data slots [3], according to this policy. Therefore, if the raw capacity of the shared wireless link is C, the effective bandwidth of the network becomes $C^* = \frac{C}{4}$.

Also note that the 802.11 protocol makes use of the RTS-CTS-DATA-ACK sequence for each data transmission; it will not send the next data packet, until the ACK for the

[3] Each slot is equal to the transmission time of one packet

Wireless Channel Raw Capacity	2M bps
Radio Transmission Range	250 meters
Carrier Sensing Range	550 meters
Buffer Size at Each Intermediate Node/Router	50 packets
Adjacent Node Distance	200 meters
Routing Protocol	Manual Source Routing
TCP Version	TCP NewReno
Default TCP Packet Size	1460 bytes
Default TCP Receiver Window Size	32 packets

Table 1. *Simulation parameters*

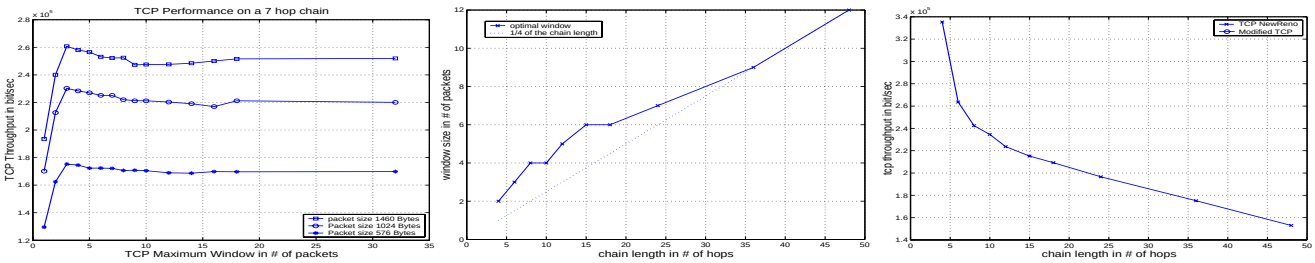

Figure 2. *Best TCP Throughput is achieved when window size is about 3 in an 7-hop Chain* Left: Single TCP throughput for different packet sizes. Middle: TCP optimal window in variable length ad hoc chains. Right: Best throughput vs. hop count.

packet currently under service is received. Essentially, the DATA-ACK loop over each wireless hop (as illustrated in figure 1) has functionality similar to a link-layer *Stop-and-Wait* protocol. However, unlike the wireline Ethernet counterpart, each link can only transmit one packet at a time, and cannot deliver multiple packets back-to-back. Hence, at any time t, each hop can only allow one packet on the fly. If each packet is l bits and the raw link capacity is C, then the one-way RTT is given by: $RTT = h \cdot \frac{l}{C}$. With the optimal window size being W^*, the effective bandwidth becomes $C^* = \frac{W^*}{RTT}$. Since $C^* = \frac{C}{4}$, we obtain the optimal TCP window size (in packets) as:

$$W^* = C^* \cdot RTT \cdot \frac{1}{l} = \frac{C}{4} \cdot h\frac{l}{C} \cdot \frac{1}{l} = \frac{h}{4}$$

At this window size of $\frac{h}{4}$, TCP can achieve best throughput. Of course, the assumption of perfect coordination among neighboring nodes is unrealistic. If the coordination is imperfect, the best window size for TCP that maximizes its throughput may be larger in reality. But the above analysis still provides a good approximation, indicating at what window size TCP achieves the best throughput, at least in a statistical sense. This has been confirmed by the real experiments that follow.

3.1.3 Verification through real experiments

To confirm the results obtained through simulations and analysis, we also performed experiments on a real testbed. We set up a 7-hop ad hoc network of chain topology, using 8 laptops computers, running the Linux kernel, version 2.4.3. Each notebook was equipped with an AT&T Lucent ORiNOCO wireless card, operating at a frequency of 2.422GHz, having a bit rate of 2Mbps and the RTS/CTS option turned on. In this topology, only neighboring nodes are within the transmission range of each other, and manual routing was employed, in order to eliminate the dynamics of routing protocols. The maximum window size of the TCP connection was controlled by adjusting the advertised window size at the end nodes, through the *Maximum_Window_Clamp* sysctl variable of the IPv4 stack.

Figure 3, shows the measured TCP throughput with different advertised window sizes. Each data point in the figure is the average calculated over 5 runs. The measurements match the results previously obtained through simulations, with a slight divergence of about 10%, mainly due to packet losses, caused by the imperfect conditions of the wireless channel.

3.2 More complex scenarios

The results described above show that there exists an optimal value for the TCP window size in the chain topol-

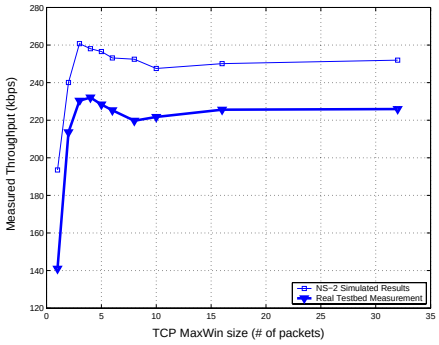

Figure 3. *Real testbed measurements of TCP throughput for different maximum advertised window sizes and a packet size of 1460 bytes.*

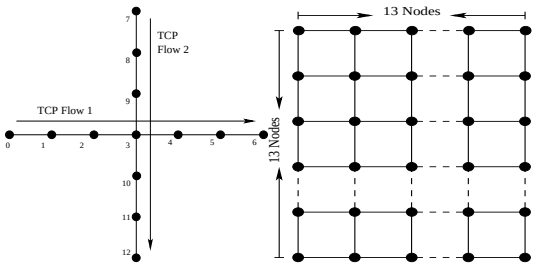

Figure 4. *More complex topologies, used in the simulation*Left: the cross topology, with 13 nodes. The distance between two neighboring nodes is 200 meters, 2 TCP flows run in each direction. Right: the 13x13 grid topology, where distances between adjacent nodes are 200 meters, both horizontally and vertically.

ogy, so that channel reuse is maximized. This results in best channel utilization and consequently maximum TCP throughput. In order to gain an even deeper understanding of the TCP throughput behavior, we expand our study to more complicated scenarios of complex topologies and multiple TCP flows.

3.2.1 Complex topologies

Figure 4 depicts the cross and grid topologies tested in our simulations, using the same parameters as those in section 5. In the cross topology, we run two TCP flows from node 0 to node 6 and from node 7 to node 12, while in the 13 X 13 grid topology, we run 4, 8 and 12 TCP flows, half of them for each direction and spaced out evenly. In the random topology, 200 nodes are placed within a rectangular area of size 1000×2500, and 20 TCP flows run, with the sources and destinations being randomly selected. Due to the randomness of the latter scenario, we repeat the simulation for each $MaxWin$ setting with 10 different randomly generated topologies and TCP flows, compute the average value as the final result.

Cross topology On a 13-node cross topology, we run 2 TCP flows. The optimal window for each flow is 2, but we measured aggregated TCP window to be 12 packets, along with 25% degraded throughput. In fact, we show that the optimal TCP window size in such cross topology $W_{cr}^* = \lceil \frac{h+4}{8} \rceil$, where h is the hop count for each flow. When $h = 6$, as in the 13-node case, $W_{cr}^* \approx 2.$, which matches well with the simulation results. The derivation of this formula is given in the appendix.

Grid topology On the 169-node grid, we ran 4, 8, and 12 TCP flows. In all cases, the measure TCP windows are significantly larger than the optimal, with throughput decreases from 15% up to 30% in 12 flows case. In particular, for the 4 TCP flow case, we computed the optimal window for each flow to be $\lceil \frac{h+5}{10} \rceil$. When $h = 12$, $W_{cr}^* \approx 2$ which also matches our simulation results well. The derivation of the above formula can again be found in the appendix.

Random topology We also run extensive experiments with random topologies. For each random topology, we repeat the simulation with increasing $MaxWin$ limitation of each flow just as in the chain topology. We still observe that there exists an optimal window when the maximum aggregated throughput is achieved. However, we are not able to analytically characterize the optimal window in the random topology.

3.2.2 Multiple flows

In the TCP study of multiple flows, the network capacity is shared by all flows. In theory, if the flows are distributed uniformly in the topology, the aggregate of the optimal TCP window of all these flows approximates the network pipe line capacity, which corresponds to the case of maximum spatial reuse of the underlying network. Thus, we test multiple TCP flows scenarios of 4, 8, and 12 flows over the grid topology, and the results are presented in table 2. The same observation on the existence of an optimal value of the TCP window size also holds for the multiple flows scenario.

3.3 Suboptimal TCP performance in Ad Hoc Network

All our simulations and analytical results confirm that for a given topology and traffic pattern, there exists an optimal window size, at which TCP achieves the best possible spatial channel reuse, and consequently maximizes the channel utilization. However, if we let TCP $MaxWin$ unbounded (i.e., the normal TCP behavior), a common observation for

Topology	Number of flows	Optimal Throughput (Kbps)	Measured Throughput (Kbps)	Optimal Window	Average measured Window
6-hop Chain	6	298	272	2	22
7-hop Chain	3	255	215	2	16
13-node Cross	2	248	203	4	12
169-node Grid	4	287	241	8	14
169-node Grid	8	957	824	8	19
169-node Grid	12	872	690	8	26
200-node Random	20	1,196	1,015	-	-

Table 2. *TCP performance over more complex topologies with multiple flows.* The data for TCP throughput and window are the aggregation of all flows in topology. In all cases, TCP stable performance is suboptimal.

all topologies and scenarios examined before is that TCP experiences a 10% to 30% decrease in its throughput. For example, The TCP throughput difference between the optimal and steady-state in random topology, given in Table 2, is about 18%. This shows that TCP normally operates suboptimally, there is clearly a lot of room for improvement. In order to propose solutions that could bridge the gap between the current and the maximum performance, in Section 4, we identify in details the reasons for this suboptimal behavior. To this end, we investigate the causes of TCP losses and link-layer packet drops, which in turn provide a mechanism for detecting the optimal operating point.

4 TCP Buffer Loss and Link-Layer Drops

The previous illustration of the variation of the window size showed that TCP does not operate around the optimal throughput point, in the steady state. This section brings the reasons behind such suboptimal performance to light, through a careful study of the TCP buffer loss behavior and the link-layer dropping pattern, in multihop, 802.11-based, wireless networks.

4.1 Link-layer drops dominate over buffer loss

The congestion control mechanism of TCP assumes that all packet losses are due to congestion in the network. Indeed, in the wired Internet, TCP packet loss is mainly caused by congestion that is experienced by the network; in this case, buffer overflow drops at the congested link can readily serve as an indication of a congestion incident. In what it follows, we examine whether the same assumption regarding packet loss holds for the environment of ad hoc wireless networks.

For this purpose, we run a series of simulations with different TCP buffer sizes, all of which are larger than 20 packets. The results are quite surprising: packet drops due to buffer overflow at each node are *never* observed, and all packet drops are due to link-layer contentions, in all simulated scenarios. Table 3 shows results for one such scenario, and more specifically the buffer occupancy at each node when the buffer size is set to 30. As it is clear from the table, the average number of packets in the buffer is only about $1\sim2$ packets at each node, and the maximum never exceeds 17 packets, at any time instant. In the same setting, we observe that all TCP drops of 165 packets out of the 12349 transmissions, during a lifetime of 300 seconds, are due to link drops, and none is cause by buffer overflows.

The above result is somewhat intriguing, and requires further explanation. How can the network experience packet loss, whether that be buffer overflows or link drops, if the average number of packets in the buffers is only $1\sim2$? Once again, due to space constraints, we refer to figure 1 in section 2 to provide an insight on this phenomenon. Suppose each node has only 1 packet in the buffer at time t, and all the nodes contend for the shared channel, in order to deliver their packets to their next-hop nodes. Further assume that node D first starts transmitting its buffered packet to E after a random backoff period, by initiating the RTS-CTS-DATA-ACK handshake with node E. After hearing the handshake between D and E, node B will defer until the packet transmission is completed (recall that D is within the carrier-sensing range of B, set to be the interference range of 550 meters, in *ns-2*). If, during this period, node A is unable to detect the ongoing transmission between D and E, and initiates an RTS message to B, the latter will not reply. If A receives no response from B after seven retransmissions of this RTS request, it drops its head-of-line packet in its buffer, according to the IEEE 802.11 standard. What we just described is essentially a by-product of the Hidden Terminal effect. This results in packet drops even when the buffer occupancy is low (only 1 packet in this case). Thus, buffer loss never happens in a multihop wireless network with each node having a sufficient buffer size. In contrast, wireline networks usually do not experience link drops, but suffer only from buffer overflows.

Node ID	#1	#2	#3	#4	#5	#6	#7	#8	#9
Max. Buf	9	11	13	14	16	15	12	10	6
Avg. Buf	0.4	0.8	1	1.9	1.9	1	0.9	0.7	0.3

Table 3. *TCP Buffer Occupancy (in packets): No packet drop due to buffer overflow.*

Using the same figure 1, we are also able to explain why contention loss eventually saturates and stabilizes at a certain dropping point, as the load increases. As far as contention drop is concerned, the *number of non-empty buffers* is the key, while the absolute number of packets in the buffer does not matter much. When all the buffers are non-empty, the network reaches maximum contention. The value of link drop probability is governed by the 802.11 MAC protocol, and not by the number of packets in each buffer – the latter aspect is only related to TCP congestion control and decides on how long the TCP flow will stay in the link loss phase.

4.2 Distributed RED-like Link Drop Behavior

If link-layer contention loss offers the first sign of network overload in many scenarios, the next concern is how this loss occurs as the network overload builds up. Before moving on to the results, a brief discussion on the meaning of "network overload" in the wireless ad hoc context is necessary. In the wireline counterpart, a busy link or a built-up queue at the bottleneck pipe signifies a condition of network overload. However, in a multihop wireless network, network overload is *no longer* a bottleneck link property, but a *shared feature* of multiple links. To make things worst, it cannot be detected within a single hop alone. Informally, we can define it as the state of the network where the incoming traffic is more than the network can handle, while operating at the point of maximum utilization; this is met when the optimal spatial channel reuse is achieved. Having defined this term that will be referred to later on, we proceed to the following results, which show that link drop exhibits a behavior similar to RED [19] over the Internet, in the sense that the drop is graceful as the network load increases.

Figure 5 (left) plots the link drop probability as a function of the TCP window size. We employ the TCP window size in order to characterize the current network load. The graph shows that contention drop probability gradually increases as more packets find their way inside the network. The figure clearly presents graceful link drops, with the increase of the network load. To better understand the underlying mechanics of this link drop behavior, we also experimented with a CBR flow using UDP. From the results of contention loss probability shown in figure 5 (right), a similar gradual link drop behavior is exhibited, in the presence of increasing network load. The figure also depicts the saturation of the contention drop probability, which happens when the load augments beyond a certain threshold.

The above results indicate that each wireless link behaves similarly to RED up to some extent, having a graceful drop behavior in the presence of network overload: it increases its drop probability as the network load grows, and finally saturates at some point. However, there are several key differences between link-layer contention drops and wired RED drops. The latter occur inside the buffer (of the single bottleneck link), but the former happen over multiple contending links, of which they are a collective feature. As illustrated in figure 6, RED drop probability is 0 when its buffer is less than a threshold value $\min\_th$, hits a maximum value when its buffer grows to $\max\_th$, and becomes 1 when the buffer is filled up. However, link drops may happen before the link reaches its capacity due to randomness, though its drop probability is very small. The link drop likelihood gets its largest value when the network load reaches a threshold, and then remains saturated at this point.

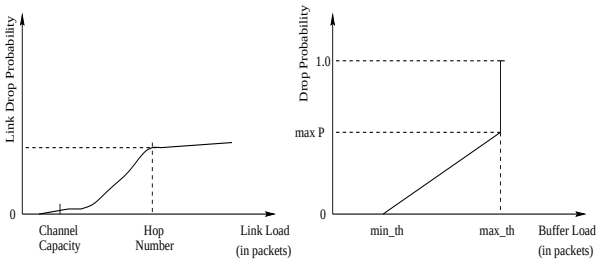

Figure 6. *Comparison between link drop and RED*

4.3 More Complex Topologies

We further run more simulations with multiple flows in topologies of increased complexity, to test the validity of the above observations. Due to space limitations, we do not show the detailed results here, hence the following provides only a concise summary of our findings.

Some among the tests include cross, grid, and random topologies. In all simulated scenarios, buffer overflows are rare if not observed at all, given that the buffer size at each node is reasonably large. Link-layer contention-induced losses happen well before buffer overflows (this is also true for UDP, as shown in the middle graph of figure 5), and contribute to all packet drops. In all cases, RED-like graceful drop is observed as the network overload starts to build up. If a node has more active neighbors, the maximum drop probability increases, but it still follows the RED-like drop pattern.

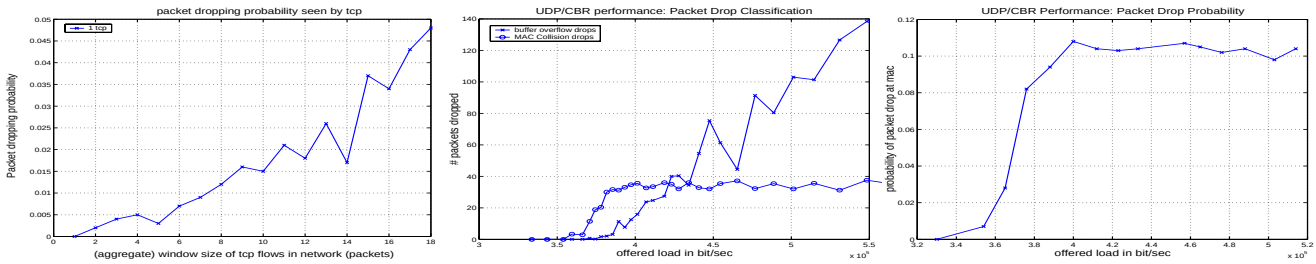

Figure 5. *RED-like Link Drop Behavior: Contention loss preceding buffer-overflow loss is an inherent feature of shared wireless medium.* Left: Contention loss and buffer-overflow loss; Middle: Contention loss as a function of the offered load. Right: Throughput decreases after contention builds up.

4.4 An Analysis of RED-like Drop

We now provide a simple analysis to show why multihop wireless links exhibit RED-like drop feature. For this purpose, we consider a generic, N-node uniformly distributed random topology. For simplicity, assume that each of the N nodes has the same probability to become backlogged (i.e., has packets to deliver). We denote the total number of *backlogged* nodes in the network as m and all of them are ready for transmission. We will show that as m increases, the link drop probability will exhibit a RED like behavior.

The m backlogged nodes in the network will use RTS-CTS-DATA-ACK handshake of 802.11 protocol in data transmission. If m is very large, not each of the m nodes is able to successfully transmit its data packet due to collisions. Among m nodes, let us denote that $c(m)$ nodes are able to successfully initiate RTS request. This happens only if each of these $c(m)$ nodes detect clear channel through its carrier sensing. Note that $c(m) \leq m$ for large m. Due to hidden terminal problem [12, 18, 11], not all of these $c(m)$ nodes may successfully transmit their DATA packets. Let us denote the number of nodes that are able to successfully transmit the DATA packets as $b(m)$. It is easy to see that $b(m) \leq c(m)$.

Using Markov-chain to modeling the retrying process of the 802.11 protocol, we can derive each node's packet loss probability P_l as follows (An outline of the derivation is given in the appendix):

$$P_l(m) = \frac{\frac{b(m)}{m}}{1 - \left(1 - \frac{b(m)}{c(m)}\right)^{r+1}} \cdot \left(1 - \frac{b(m)}{c(m)}\right)^{r} \qquad (1)$$

where $r = 7$ is the maximum retry count for RTS in 802.11 protocol when RTS-CTS handshake fails.

We now use (1) to explain why each node/link has a RED like drop behavior as the network load increases. In the uniformly distributed N node field, let us use B^* denote the maximum number of nodes that can transmit their DATA packets concurrently without collision. At this value, the network achieves maximum channel spatial reuse. Among N nodes, we also denote the maximum number of nodes that can initiate RTS messages, i.e., they perceive clear channel through carrier sensing) as C^*. In the following, we will show that the link drop probability behaves like RED with B^* and C^* as the two threshold values.

First we consider the case when the network is underloaded:

Proposition 4.1 *Denote the maximum number of nodes (that can concurrently transmit DATA in the given topology) as B^*. When the number of backlogged nodes m is smaller than B^*, i.e., $m < B^*$, then packet drop probability $P_l \approx 0$.*

To see why the above is true, since $m \leq B^*$, then all m nodes can statistically transmit simultaneously. Therefore, statistically $b(m) \approx c(m) \approx m$. Then according to (1), the drop probability over each link is $P_l \approx 0$. This shows that, as long as the network is underloaded, the link drop is negligible.

Now let us consider the second case. When the number of backlogged nodes m increases larger than B^*, i.e., the network is overloaded, the following holds:

Proposition 4.2 *When the network is overloaded (i.e., the number of backlogged nodes m is greater than B^*), the link drop probability P_l increases as m increases.*

We still use (1) to see why the above is true. In this case, all m can successfully initiate RTS message but only B^* nodes can transmit their DATA without collisions. That is, $b(m) \approx B^*$ but $c(m) \approx m$. Then, $B^* < m < C^*$. It is easy to see that $P_l(m)$ is an increasing function of m since $\frac{dP_l(m)}{dm} > 0$. This shows that link drop probability increases as the network load (as expressed by m) further increases.

Finally, we look at the third case. As the network load further increases, then link drop probability starts to saturate:

Proposition 4.3 *Once network is heavily loaded in the sense that $m > C^*$, then the link drop probability P_l remains constant statistically.*

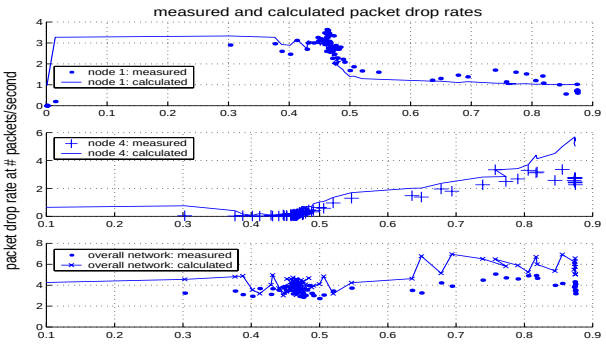

Figure 7. *Link drop probability in a 7-hop chain matches well with our analysis.* We introduce 7 1hop UDP/CBR flows over the chain. As we increase the source rate uniformly for each source, we measure the backlog number $b(m)$ and $c(m)$. Given equation (1), we derived the value of packet drop probability. The dots are the packet drop rate measured in real simulation, and the lines are the derived value. The X-Axis is the drop probability, and the Y-Axis is the average backlog probability of all nodes in the network.

In this case, among the m nodes, only C^* out of $c(m)$ nodes can initiate RTS, and only B^* nodes can transmit DATA packets without collision. Therefore, $c(m) \approx C^*$ and $b(m) \approx B^*$. Then $P_l(m)$ remains constant statistically according to (1).

In the above, we use a simple model to explain why multihop wireless links may exhibit RED-like drops. Though the model seems crude and leaves out many details such as the impact of backoff in the MAC protocol, it provides a good approximation to real results. Figure 7 shows that it matches very well with the simulations in a 7-hop chain example.

5 Improving TCP Performance

The sound understanding of TCP throughput behavior, and the fundamental reasons for packet dropping as illustrated in the preceding two sections of this work, shed light on how to improve the TCP performance, over a multihop wireless network. In what it follows, we propose and evaluate two simple, yet elegant solutions that are based on the aforementioned observations, and enhance performance up to 30%.

5.1 Techniques

The Distributed Link RED (LRED) algorithm The first straightforward way for improving TCP's performance is to reduce the buffer size at each node. Unfortunately, this approach will exhibit the undesirable side effect of the buffer not being able to absorb traffic periods characterized by burstiness of TCP flows.

Therefore, we devise a Link RED (LRED) algorithm that exploits the built-in dropping mechanism of the 802.11 MAC, from which the transportation layer performance can benefit. The main idea behind LRED is to further tune wireless link's drop probability, based on the observed likelihood of link drops. While the wired RED provides a linearly increasing drop curve as the queue exceeds a minimum size min_th, LRED does so as the link drop probability exceeds a minimum threshold. As it was shown in more details in section 4, link-layer packet drops provide an indication of network overload; we take advantage of exactly this observation, in order to control packet dropping –or marking, as will be elaborated later, according to the average number of MAC attempts.

In the LRED algorithm, sketched below in pseudocode, the link layer maintains an average number of the retries of recent packet transmissions. The head-of-line packet is dropped/marked from the buffer with a probability that is based on this average number. At each node, if the average number of retries is small, say less than min_th, which in essence means that the node is seldom hidden, packets in the buffer are not dropped/marked. If it gets larger, then the dropping/marking probability is computed, and the minimum of the calculated value and a maximum bound max_P is used.

A nice feature provisioned by this algorithm, which we mentioned above and also used in our simulations, is its smooth integration with ECN-enabled TCP flows: instead of blindly dropping the packets, we can simply mark them at the link layer, and thus allow ECN-enhanced TCP flows to adapt their offered load without losing any packets. TCP performance is further improved, by paying the moderate cost of a slightly more complex link-layer design.

To summarize, LRED is a simple, yet attractive mechanism that, by monitoring just a single parameter –the average number of retries in the packet transmissions at the link-layer, accomplishes three goals: a) ameliorates the performance of TCP, by increasing its throughput, b) provides an early sign of network overload to the transport layer protocols, which in turn can react accordingly, and c) improves fairness among multiple competing flows, as it will be shown in the evaluation of the next section.

Adaptive Pacing Our second technique seeks to enforce an adaptive pacing approach at the link-layer. In parallel to the LRED algorithm, which prevents the traffic from overloading the network, the goal of adaptive pacing it to improve spatial channel reuse, by distributing the traffic in the network in a more balanced way, while enhancing at the same time the coordination of forwarding nodes along the data path. As a welcome outcome, the network runs around its optimal operating point. The design of this mechanism works in concert with the distributed coordination function

Algorithm 1 L-RED: LinkLayerSend(Packet p)

Require: avg_retry is the average MAC retries for each packet

if $avg\_retry < min\_th$ **then**
  $mark\_prob \leftarrow 0$
  $pacing \leftarrow ON$
else
  $mark\_prob = min\{\frac{avg\_retry - min\_th}{max\_th - min\_th}, max\_P\}$
  set $pacing$ OFF
end if
mark p with $mark\_prob$
MacLayerSend(p, $pacing$)
$retry$ = GetMacRetries()
$avg\_retry = \frac{7}{8} avg\_retry + \frac{1}{8} retry$

Algorithm 2 Adaptive Pacing

Require: *extra_Backoff* $= 0$

if received ACK **then**
  random_Backoff $\leftarrow$ *ran_backoff(cong_win)* {DATA transmission succeeded. Setup the backoff timer}
  if $pacing$ is ON **then**
    extra_Backoff = TX_Time(DATA) + overhead
  end if
  backoff $\leftarrow$ *random_Backoff + extra_Backoff*
  start *backoff_timer*
end if

(DCF) of the 802.11 MAC standard.

In the current 802.11 protocol, a node is constrained from contending for the channel by a random backoff period, plus a single packet transmission time that is announced by its immediate downstream node. However, the exposed receiver problem [14] still happens due to lack of coordination between nodes that are *two* hops away from each other. Adaptive pacing solves this problem, without requiring nontrivial modifications to the 802.11, or a second wireless channel [14]. The underlying idea is to let a node further backoff an additional packet transmission time when necessary, besides its current deferral period (i.e. the random backoff, plus one packet transmission time). This extra backoff interval helps in reducing contention drops caused by exposed receivers, and extends the range of the link-layer coordination from one hop to two hops, along the packet forwarding path.

The algorithm, an outline of which is presented below in pseudocode, works in collaboration with the previously depicted LRED algorithm, and is described as follows: first, it is enabled from within the LRED algorithm, when a node, after sending one packet, finds its average number of retries to be less than a min_th threshold. Then, this node calculates its backoff time as usual, but if pacing was enabled, it also adds to that backoff period an interval equal to the transmission time of the previous data packet (plus the overhead), and triggers its timer.

To conclude, by requiring nodes to backoff for an extra packet sending period, after successfully transmitting a packet, a better coordination among nodes is achieved, and the behavior of an ideal scheduler that increases the end-to-end network throughput is imitated.

5.2 Performance Evaluation

In this section, the efficiency of the two previously proposed techniques is studied. We measure and compare the performance of TCP NewReno over the 802.11 standard and our enhanced link layer, on chain, cross and grid topologies, as well as with single and multiple TCP flows. Initially, the performance is analyzed separately for each mechanism, and then an evaluation for the combined effect of these solutions follows.

L-RED A 7-hop chain is used to evaluate whether the L-RED algorithm is able to stabilize the TCP window around the optimal operating point. Figure 8 plots the distribution of time TCP spent in various window sizes. In our simulations, the maximum allowed window size is set to 32 packets. The graph confirms that the L-RED control is very effective in keeping the TCP window size distribution close to the optimal value of 3, while packet drops due to link-layer contention almost disappear, as a result of reduced load in the network. The figure also shows that by applying L-RED, the window distribution becomes much "thinner" compared with TCP NewReno over the unmodified 802.11. This comes as a consequence of L-RED's packet marking mechanism upon detecting the onset of congestion. The latter is further explained in the right graph of figure 8, which shows that LRED reduces the packet drops at the link layer. As we observed in previous discussions, contention offers the first sign of network congestion, and L-RED catches this sign by monitoring the average number of retransmissions for each packet.

Adaptive Pacing The 7-hop chain topology is again employed for measuring the effectiveness of the adaptive pacing. As already mentioned, the goal of pacing is to improve the spatial reuse of the channel, and force it to operate around the network's optimal operating point. Figure 9 shows the simulation results of TCP NewReno over link layers with and without pacing. With adaptive pacing, TCP is able to achieve as much as 10% improvement (figure 9, right) when the window fluctuates around the optimal $\frac{1}{3}h = 6$. The figure also shows that when pacing is applied, the link layer packet drop decreases (figure 9, middle), and RTT (figure 9, left) is kept small before and around the op-

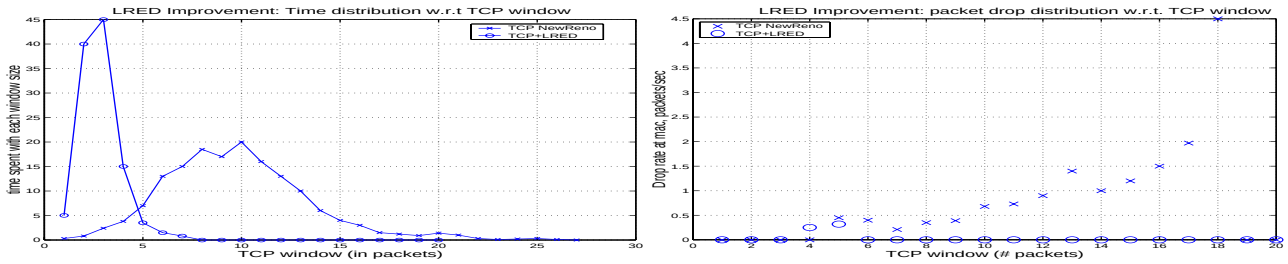

Figure 8. *Window control stabilizes TCP window size around the optimal value.* Left: Time distribution of instantaneous window size becomes narrower and sharper; Right: Contention loss reduces compared with TCP NewReno.

timal window. However, by doing the pacing along, TCP can still overload the network. In the following, we combine these two techniques and evaluate how much performance improvement TCP is gained, in terms of throughput and fairness.

5.2.1 Overall performance evaluation

We now evaluate the overall performance achieved by combining the techniques of our design, over three topologies: chain, cross and grid.

Chain topology The results for chain topologies of various lengths, both for a single flow and six flows, are plotted in figure 10. In all cases, we observe that our LRED+pacing enhanced link layer is able to boost TCP throughput up to 30%. Furthermore, our modifications force TCP to stabilize at a window size close to the optimal value. For chains longer than 15 hops, our techniques are again able to achieve a gain of 10%∼30% in throughput; and as it appears, the longer the chain, the better the throughput improvement. This is because our pacing mechanism helps TCP to optimize spatial channel reuse, and the longer the chain, the more it benefits from that reuse property.

Cross Topology Two flows over a 13-node cross topology were simulated in this experiment. Table 4 presents the throughput and fairness results for both flows. The fairness results are computed using the fairness index $\frac{(\sum_{i=1}^{n} x_i)^2}{n \cdot \sum_{i=1}^{n} x_i^2}$, as defined in [20]. Our design not only increases aggregate throughput, but also improves fairness of both flows. On the other hand, TCP NewReno over the unmodified link layer results in large unfairness; this is mainly, due to the well known capture characteristic of the IEEE 802.11 protocol [7, 8].

Grid Topology Finally, for the grid case, we simulated 2, 4, 8 and 12 flows over a 13×13 grid topology (figure 4). Aggregate throughput and fairness results are summarized in table 6, while more details for the specific case of 4 flows, 2 for each direction, are provided in table 5. Once again, in all cases, we are able to achieve about 5%∼10% throughput increase, while significantly improving the fairness index.

	TCP NewReno	LRED+
flow 1	244 Kbps	166 Kbps
flow 2	0 Kbps	153 Kbps
Aggregate	244 Kbps	319 Kbps
Fairness	0.5	0.9983

Table 4. *Throughput and Fairness Comparison of NewReno(NR) and NewReno+LRED+ PACING in Cross Topology:* Throughput and fairness improve over TCP NewReno.

	TCP NewReno over standard LL	TCP NewReno over LL+LRED+PACING
flow 1	532 Kbps	85512 Kbps
flow 2	126229 Kbps	90459 Kbps
flow 3	115554 Kbps	70334 Kbps
flow 4	1608 Kbps	47946 Kbps
Aggregate	242923	294251
Fairness	0.51	0.95

Table 5. *Throughput and Fairness Comparisons of NewReno and NewReno+LRED+PACING(LRED+) of 4 flows in 13×13 Grid.*

6 Discussions

This section further discusses two important issues of the previous study.

Other TCP variants From the results shown in sections 3 and 4, it seems that TCP Vegas, which gauges the throug-

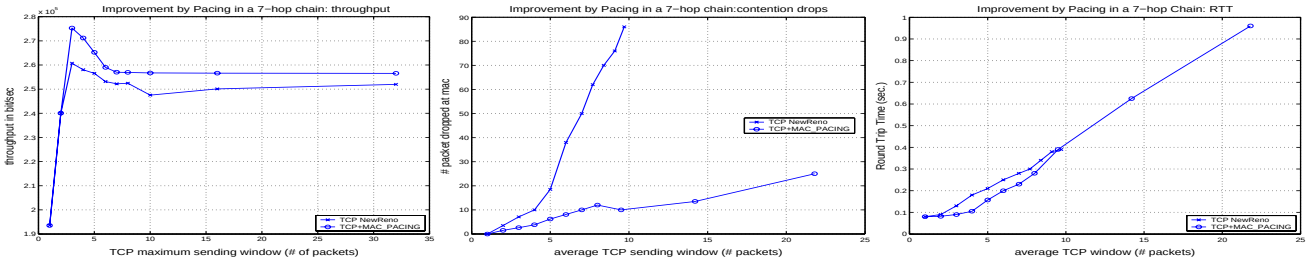

Figure 9. *Pacing increases TCP throughput in a 8-hop chain.* Left: Throughput comparison with TCP NewReno. Middle: Pacing reduces link-layer contention losses. Right: Pacing slightly reduces RTT.

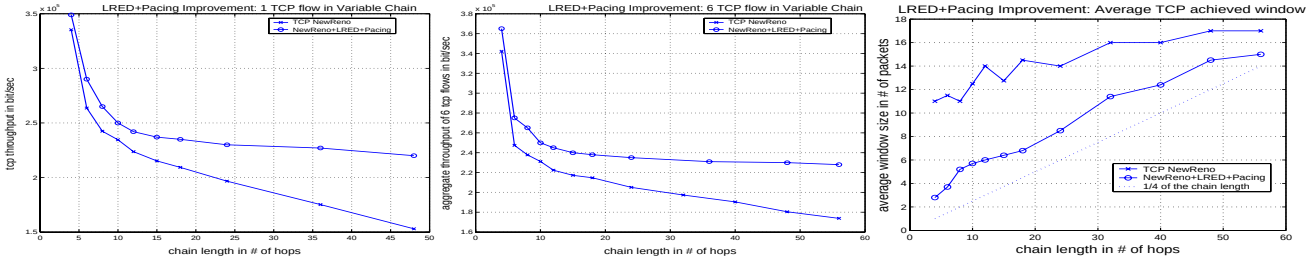

Figure 10. *Throughput improvement over TCP NewReno in a h-hop chain topology ($h = 3, \ldots, 48$).* Left: Single flow; Middle: Aggregate throughput of 6 flows; Right: Average window size closer to the optimal value

	NR Aggregate	NR Fairness	LRED+ Aggregate	LRED+ Fairness
2 flows	203K bps	0.502	252K bps	0.921
4 flows	241K bps	0.508	294K bps	0.952
8 flows	824K bps	0.524	963K bps	0.527
12 flows	690K bps	0.455	880K bps	0.56

Table 6. *Aggregate throughput and fairness comparisons of NewReno(NR) and NewReno+LRED+PACING with 2, 4, 8 and 12 flows in the grid.*

put before increasing its congestion window size, may work better. However, our experiments indicated that TCP Vegas and TCP NewReno perform comparably in short hops ($\leq$ 6). However, Vegas performs 10%$\sim$20% worse than NewReno in longer hops ($\geq$ 9). The main reason is that Vegas keeps its average window size very small (e.g., about 3 packets even in a 16-hop chain), which adversely affects its throughput. Our findings lead to the conclusion that TCP NewReno seems to be the best TCP variant that we could use.

Implications of our results The built-in RED-like link drop property can assist in controlling the network overload, and this is particularly true for long-hop flows. However, this drop behavior has much randomness in it and is also hard to control. In our design part, we would like to reduce it through the adaptive pacing mechanism, and use LRED instead, which has a more predictable behavior.

7 Related Work

TCP over wireless cellular networks has been an active research topic. [1] presents a summary of TCP optimization over such networks, where the *single-hop* wireless link is the first/last hop. The focus is to hide wireless channel errors from TCP. If IEEE 802.11 protocol is used in cellular networks, channel-error induced losses would not be an issue since seven retransmissions are adequate for loss recovery. The focus of this research is on multihop wireless networks, which may incur link-contention-induced packet drops that do not exist in the single-hop cellular networks.

There are several recent studies on TCP performance over ad hoc networks. [2] investigates the effect of mobility-induced link breakage on TCP performance. It studies the impact of routing dynamics of DSR protocol and mobility pattern upon TCP protocol. The authors further proposed an *explicit link failure notification* (ELFN) technique to help TCP differentiate link failure-induced losses from congestion losses. A more recent work [3] examines various performance aspects of TCP-ELFN. Our research takes another dimension: It studies static ad hoc networks, and solely focuses on the effect of multihop, shared wireless medium upon TCP. In fact, we use pre-configured, manual routing in our study to minimize the impact of routing dynamics.

[7, 8] study TCP ACK traffic effect on TCP performance and severe unfairness and capture effect caused by the MAC backoff mechanism. The work was conducted in the context of two MAC protocols: CSMA and FAMA. TCP is observed to have very small throughput when it traverses multiple wireless hops with a window size larger than 1 packet. The authors call for introduction of link-layer ACKs to help reduce packet drops. Our work showed that even though link-layer ACKs are used in 802.11, TCP still suffers packet drops due to link-layer contentions. In fact, all packet drops are due to such contentions, and buffer never overflows. We further show that TCP typically operates in a suboptimal region that does not lead to best utilization of the shared wireless channel.

There are several works that employ throughput control to improve TCP congestion control in wired Internet [6, 5]. This is typically based on the observation that TCP throughput flattens when the buffer on the bottleneck link builds up and the network approaches congestion state, hence increasing TCP window size will only lead to buffer overflows. Tri-S [6] compares the currently measured throughput with the value achieved when the window size is one packet smaller, and TCP Vegas [5] compares the expected throughput and the achieved throughput. The goal is to avoid buffer overflow events on the bottleneck link of the wired network. Our focus is on multihop wireless networks. We use throughput control to avoid injecting excessive packets into the network. The goal is to reduce packet drops due to link contentions (instead of buffer overflows) and increase spatial channel reuse.

Early study [12] on Hidden Terminal problem showed that the performance of CSMA in multihop packet radio network suffers from hidden terminal interference, and the performance could degrade to that of the pure ALOHA protocol. Solutions that use three or four way handshake protocols have been proposed recently [13, 14, 15], which make effort to clear up the channel at both sender and receiver side before DATA can be sent. Some other solutions apply code assignment approach in Frequency-hopping spread-spectrum (FHSS) networks [17] or a receiver initiated collision avoidance technique [16] to increase the efficiency of multiple channel access. In our paper, we take a different approach. We take the occurrence of hidden terminal packet drops as an indication to the network overload. By tuning the packet dropping behavior at the link layer, we seek to improve the throughput of TCP by helping it identify its optimal window.

8 Conclusion

Ad hoc networks hold great promise in pervasive computing and wireless sensor networks. TCP seems to be the natural choice for reliable data delivery over such multihop, error-prone, shared-channel wireless networks. This work systematically studies the impact of shared medium on TCP performance. Our results showed that only when the buffer is small, buffer overflow drops dominate. As the buffer further increases, link-layer drops dominate. The link drop behaves like a RED gateway in terms of graceful drops in the presence of network overload. However, the drop provided by link layer is not sufficient. This motivates us to design a link RED which compensates the dropping probability.

Ongoing work seeks to further tune the LRED design parameter and test the design through more extensive tests.

References

[1] H. Balakrishnan, V. Padmanabhan, S. Seshan, and R. H. Katz. "A Comparison of Mechanisms for Improving TCP Performance over Wireless Links," *IEEE/ACM Transactions on Networking*, Dec. 1997

[2] G. Holland and N. Vaidya, "Analysis of TCP Performance over Mobile Ad Hoc Networks," *ACM MOBICOM* 1999

[3] J. P. Monks, P. Sinha and V. Bharghavan, "Limitations of TCP-ELFN for Ad Hoc Networks," *MOMUC* 2000

[4] A. Aggarwal, S. Savage, and T. Anderson, "Understanding the Performance of TCP Pacing," *IEEE INFOCOM* 2000

[5] L. S. Brakmo, S. W. O'Malley and L. L. Peterson, "TCP Vegas: New Techniques for Congestion Detection and Avoidance," *ACM SIGCOMM* 1994

[6] Z. Wang and J. Crowcroft, "A New Congestion Control Scheme: Slow Start and Search (Tri-S)," *ACM Computer Communication Review*, 1991

[7] M. Gerla, R. Bagrodia, L. Zhang, K. Tang, and L. Wang, "TCP over Wireless Multihop Protocols: Simulation and Experiments," *IEEE ICC* 1999

[8] M. Gerla, K. Tang, and R. Bagrodia, "TCP Performance in Wireless Multihop Networks," *IEEE WMCSA* 1999

[9] D. Eckhardt and P. Steenkiste, "Measurement and Analysis of the Error Characteristics of an In Building Wireless Network," *ACM SIGCOMM* 1996.

[10] C. Barakat, "TCP/IP Modeling and Validation," *IEEE Network*, 15(3), 2001.

[11] V. Bharghavan, "Performance Analysis of a Medium Access Protocol for Wireless Packet Networks," *IEEE Performance and Dependability Symposium* 1998.

[12] F.A. Tobagi and L.Kleinrock, "Part II - the hidden terminal problem in carrier sense multiple-access and the busy-tone solution", *IEEE TOC 23, 1975*

[13] P.Karn, "MACA, A New Channel Access Method for Packet Radio", *Proc. 9th ARRL/CRRL Amateur Radio Computer Networking Conference,*, September, 1990.

[14] Vaduvur Bharghavan, Alan Demers, Scott Shenker, Lixia Zhang, "MACAW: A Media Access Protocol for Wireless LAN's " *In Proc. of the SIGCOMM'94*

[15] C.L.Fullmer and J.J.Garcia-Luna-Aceves, "Solutions to Hidden Terminal Problems in wireless Networks", *In Proc. ACM SIGCOMM 1997*

[16] Asimakis Tzamaloukas and J.J.Garcia-Luna-Aceves, "A Receiver-Initiated Collision-Avoidance Protocol for Multi-Channel Networks", *In Proceedings of INFOCOM 2001.*

[17] E.S.Sousa and J.A.Silvester, "Spreading Code Protocols for Distributed Spread Spectrum Packet Radio Networks", *IEEE Transactions on Communications, 36, March 1988*

[18] D. Allen, "Hidden Terminal Problems in Wireless LAN's", *IEEE 802.11 Working Group paper 802.11/93-xx.*

[19] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking* 1(4):397-413, Aug. 1993

[20] R.Jain, D-M. Chiu and W. Hawe. "A Quantitative Measure of Fairness and Discrimination For Resource Allocation in Shared Conputer Systems." *Technical Report TR-301,* DEC Research Report, September, 1984

[21] Jinyang Li, Charles Blake, Douglas S. J. De Couto, Hu Imm Lee, and Robert Morris, "Capacity of Ad Hoc Wireless Networks", *Proceedings (MobiCom '01)* Rome, Italy, July 2001.

9 Appendix

9.1 A General Model For Packet Loss and Network Load

A node is either in the backoff state when it remains silent, or in the process of DATA sending attempt which include RTS/CTS handshake and DATA transmission. Note an initiation of RTS/CTS doesn't guarantee an eventual success DATA transfer. At the steady state, for a given time slot, we define the probability that a node u initiates with RTS for flow f as CS_f, and the probability that a successful DATA transfer for flow f as B_f. Note they are average value over a long term measurement.

From our previous discussions, a successful DATA transfer of flow f requires the sender to initiate the flow first, in addition, the flow is not hidden by any terminals in the Hidden Area. Let H_f be the probability that a flow f is hidden

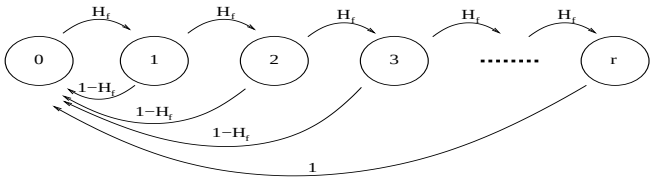

Figure 11. *the diagram of the Markov Chain* for calculating the packet drop probability from the hidden probability.

at node u at the steady state, then $B_f = (1 - H_f) \cdot CS_f$. Therefore, in steady state, we have

$$H_f = 1 - \frac{B_f}{CS_f} \tag{2}$$

The steady state per-flow drop probability We consider the discrete Markov model of retrying process in steady state in Figure 11, where H_f is the long term average hidden probability for each RTS initiation. The states represent the number of times the sender has tried for a packet. Each transition is triggered by RTS initiation. For each initiation, the packet either goes throughput with probability $1 - H_f$ or fails with H_f. We are interested in the probability of the sender has r failed attempts p_r, that is the probability of state r.

By considering state 0, we have $p_0 h = (1 - H_f) \cdot \sum_{i=1}^{r-1} p_i + p_r$, and for each other state, it holds that $p_i = p_{i-1} \cdot H_f = p_0 \cdot h^i \qquad i = 1, 2, \ldots, r$ By the unity condition, we have

$$p_r = \frac{1 - H_f}{1 - H_f^{r+1}} H_f^r$$

Since p_r is probability measure based on number of RTS initiations, to convert it into time slot, we note that the expected time slots for each initiation is $1/CS_f$. And the average packet loss probability L_f for a given time slot in steady state is

$$L_f = p_r \cdot CS_f = \frac{1 - H_f}{1 - H_f^{r+1}} H_f^r \cdot CS_f \tag{3}$$

The steady state aggregated drop probability We have discussed the per-flow packet drop probability. Now we consider the aggregated behavior of all flows within the network. Our goal is to derive the probability that at least one packet loss happen for all the m backlogged nodes in the network, $L(m)$. This probability allows us to connect the link layer behavior with TCP window achieved which has direct impact on the throughput achieved by TCP.

In the following, we first compute CS_f and B_f for each backlogged node based on the global spatial reuse perspective. Then we apply (2) and (3) to derive the steady state aggregated drop probability. To make the problem tractable, we make the following assumptions. We assume that the traffic are distributed within the network in a purely random fashion. Specifically, for m backlogged senders in the network, each node has an equal backlog probability of $\frac{m}{|V|}$, where $|V|$ is the total number of nodes in the network. Further, we assume that the nodes are randomly distributed in the network, that is for a network covering an area of S, the expected space each node occupies is $S/|V|$ on average.

At the network level, the carrier sensing capacity,C^*, is defined as maximum number of concurrent RTS initiation in the network without collision; and the data forwarding capacity, B^*, as maximum number of concurrent successful DATA transmissions. It's easy to see that we always have $C^* \geq B^*$.

Again, we consider the system in steady state. Given the global backlog ρ, the average number of backlogged sender is $m = |V|\bar{\rho}$, where $|V|$ is total nodes in network, and $\bar{\rho} = \frac{1}{|V|} \sum_{i=1}^{|V|} \rho_i$. Since we assume these senders distribute in the network evenly, the expected area covered by each node is S/m. In steady state, at a given time slot, in order for all these nodes to initiate with RTS, the minimum spacing required is S/C^*. Therefore, on average, $c(m) = \lfloor m / \lceil \frac{m}{C^*} \rceil \rfloor$ nodes among total m senders can initiate concurrently. Among them, $b(m) = \lfloor c(m) / \lceil \frac{c(m)}{B^*} \rceil \rfloor$ will succeed in concurrent DATA transmission. Therefore for each flow f the carrier sensing probability is readily given $CS_f(m) = \frac{c(m)}{m}$, and successful data forwarding probability $B_f(m) = \frac{b(m)}{m}$. from equation (2), we have $H_f(m) = 1 - \frac{b(m)}{c(m)}$ for each flow. According to (3), per flow loss probability is

$$L_f(m) = \frac{b(m)/m}{1 - (1 - (b(m)/c(m))^{r+1}} \cdot \Big(1 - \big(\frac{b(m)}{c(m)}\big)\Big)^r \tag{4}$$

For all the backlogged flows, the aggregated loss probability among all $a(m)$ initiated node is given by

$$L(m) = 1 - (1 - L_f(m))^{c(m)} \tag{5}$$

9.2 Optimal window size derivation for the grid and cross topologies

Cross topology We derive the optimal window size $W_{cr}^* = \lceil \frac{h+4}{8} \rceil$ by using the cross topology of Figure 4 (left). Note, as in the derivation of the optimal window size of a chain topology in the paper, we use the interference range as two times the transmission range. Thus two transmissions should be 3 hops away from each other. In the cross topology, each TCP flow traverses h hops and the total number of nodes is $2h + 1$. Look at how packets traverse the 4-hop neighborhood, consisting of nodes #1, #2, #3, #4, #5, #8, #9, #10, #11. Due to shared-medium feature and the large interference range, the eight links among these 9 nodes are

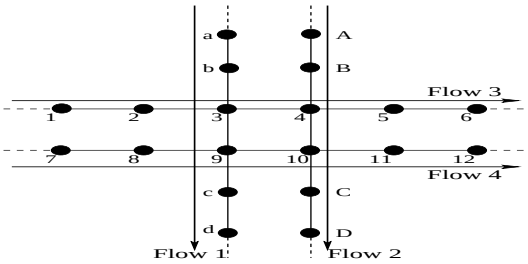

Figure 12. A diagram for TCP optimal window derivation in grid topology with 4 flows.

interfering with each other. None of them can be scheduled at the same time. Therefore, each packet needs 8 slots to traverse this 4-hop neighborhood. It follows the effective bandwidth will be $\frac{C}{8}$ for each flow, where C is the raw link capacity. The time RTT needed to traverse h hops in the cross topology is computed as follows. Since the situation outside the 4-hop region is identical to a chain topology, $RTT = (h-4)\frac{l}{C} + 8\frac{l}{C} = (h+4)\frac{l}{C}$. Then, the optimal window size $W^*_{cr} = \frac{C}{8} \cdot RTT \cdot \frac{1}{l}$. This way, we derive the window size (in packets) as $W^*_{cr} = \lceil \frac{h+4}{8} \rceil$ by rounding up the integer.

Grid topology If each flow traverse h hops in the grid topology, we now derive the optimal window size $\lceil \frac{h+9}{14} \rceil$ by using Figure 12. Note that Flows 1 and 3 form a cross, thus they need 8 slots to deliver a packet out of the intersection area. Now add Flow 2 first. The worst-case estimate shows that we need $8 + 5 = 9$ slots to deliver a packet for each flow over the intersections. However, since $A \rightarrow B$ and $C \rightarrow D$ can be scheduled with the first 8 slots with Flows 1 and 3, we only need 3 extra slots. Similarly, adding Flow 4 also needs 3 extra slots. Therefore, $8 + 3 + 3 = 14$ slots are needed to deliver each packet out of the intersections. Then the effective bandwidth is $\frac{C}{14}$, given raw link capacity C. We can also compute the time needed for a packet to traverse h hops in the grid topology is $RTT = (h-5)\frac{l}{C} + 14\frac{l}{C} = (h+9)\frac{l}{C}$. Finally, the optimal window size $W^*_{gr} = \frac{C}{14} \cdot RTT \cdot \frac{1}{l}$. Hence, the optimal window $W^*_{gr} = \lceil \frac{h+9}{14} \rceil$.