# Understanding Congestion Control in Multi-hop Wireless Mesh Networks

Sumit Rangwala Apoorva Jindal Ki-Young Jang Konstantinos Psounis
Ramesh Govindan
University of Southern California
{srangwal, apoorvaj, kjang, kpsounis, ramesh}@usc.edu

## ABSTRACT

Complex interference in static multi-hop wireless mesh networks can adversely affect transport protocol performance. Since TCP does not explicitly account for this, starvation and unfairness can result from the use of TCP over such networks. In this paper, we explore mechanisms for achieving fair and efficient congestion control for multi-hop wireless mesh networks. First, we design an AIMD-based rate-control protocol called Wireless Control Protocol (WCP) which recognizes that wireless congestion is a neighborhood phenomenon, not a node-local one, and appropriately reacts to such congestion. Second, we design a distributed rate controller that estimates the available capacity within each neighborhood, and divides this capacity to contending flows, a scheme we call Wireless Control Protocol with Capacity estimation (WCPCap). Using analysis, simulations, and real deployments, we find that our designs yield rates that are both fair and efficient, and achieve near optimal goodputs for all the topologies that we study. WCP achieves this level of performance while being extremely easy to implement. Moreover, WCPCap achieves the max-min rates for our topologies, while still being distributed and amenable to real implementation.

## Categories and Subject Descriptors

C.2.1 [**Computer Communication Networks**]: Wireless communication

## General Terms

Design, Experimentation

## Keywords

Congestion Control, Multi-hop, Mesh, Wireless, WCP, WCPCap

## 1. INTRODUCTION

Static multi-hop wireless mesh networks, constructed using off-the-shelf omnidirectional 802.11 radios, promise flexible edge connectivity to the Internet, enabling low-cost community networking in densely populated urban settings [2]. They can also be rapidly deployed to provide a communications backbone where none exists, such as in a disaster recovery scenario.

However, their widespread adoption has been limited by significant technical challenges. Finding high-quality routing paths was an early challenge addressed by the research community [13]. However, that alone is not sufficient to ensure good performance in mesh networks, where transport protocols like TCP can perform poorly because of complex interference among neighboring nodes. In particular, TCP does not explicitly account for the fact that congestion in a mesh network is a neighborhood phenomenon. Consider the topology of Figure 1, in which links connect nodes which can exchange packets with each other, perhaps with asymmetric reception rates. In this topology, it is easy to show in simulation and actual experiments that the TCP connection in the middle is almost completely starved (gets extremely low throughput), since it reacts more aggressively to congestion than the two outer flows. As an aside, we note that research on TCP for last-hop wireless networks [8, 7] does not address this problem.

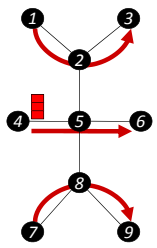


**Figure 1: Stack topology**

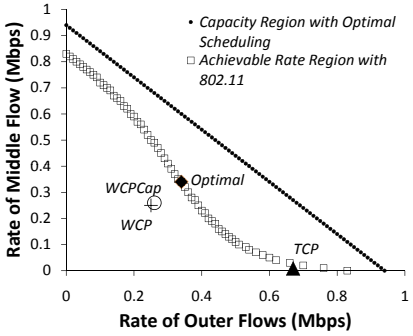


**Figure 2: The achievable rate region**

To understand the properties of a desirable solution to this problem, consider Figure 2. The y-axis plots the rate achieved by the middle flow, and the x-axis for the outer two flows (by symmetry, these flows will achieve approximately the same rate for any scheme) of Figure 1. Now, with a perfect MAC scheduler that has the same overhead as 802.11, it is intuitively clear that the rates achievable lie on or below the straight line shown in the figure (since an optimal scheduler would either schedule the two outer flows simultaneously or the flow in the middle). With 802.11, there is some loss of throughput due to contention, and the corresponding *achievable-rate region* bounds the rates achievable by the flows on this topology (in Section 4.1, we describe a methodology to compute the achievable-rate region). TCP achieves rates that lie at one corner of this plot. We contend that, for this topology, a desirable solution is one that gets us close to the max-min fair rate allocation


*MobiCom'08,* September 14–19, 2008, San Francisco, California, USA.

point, which corresponds to the intersection of the 45° line and the 802.11 achievable-rate curve.

In this paper, we explore mechanisms for achieving such a solution in wireless mesh networks. Three considerations inform our choice of mechanisms. First, we do not make any changes to the widely-used 802.11 MAC. It may well be that such changes can improve the performance of our mechanisms, but we have deliberately limited the scope of our work to enable a clearer understanding of congestion control. Second, our approach is *clean-slate*. We conduct our explorations in the context of a *rate-based* protocol that incorporates some of TCP's essential features (such as ECN, and SACK), yet allows us to explore more natural implementations of the mechanisms for improving fairness and efficiency that we study in this paper. However, our work makes no value judgement on whether a clean-slate transport protocol is *necessary* for mesh networks; it may be possible to retrofit our mechanisms into TCP. Finally, we restrict our explorations to plausibly implementable mechanisms, in contrast to other work that has explored theoretical methods for optimizing (separately or jointly) scheduling and rate assignment in wireless networks [16, 35, 46, 39].

**Contributions.** We make two contributions in this paper. First, we design an AIMD-based rate-control protocol called WCP which explicitly reacts to congestion within a wireless neighborhood (Section 3.1). Specifically, we correctly identify the precise set of nodes within the vicinity of a congested node that needs to reduce its rates. Signaling these nodes can be implemented using a lightweight *congestion sharing* mechanism. More interestingly, we find that congestion sharing alone is not enough, and that, to achieve fairness, sources also need to clock their rate adaptations at the time-scale of the highest RTTs of flows going through the congested region. This can be implemented using a local mechanism for *RTT sharing*. Figure 2 shows that, for the topology of Figure 1, WCP avoids starving the middle flow (we discuss methodology and more detailed experimental results in Sections 4 and 5).

Our second contribution is the design of a distributed rate controller that estimates the available capacity within each neighborhood, and apportions this capacity to contending flows. This scheme, which we call WCPCap (Section 3.2), has the property that it uses local information and can *plausibly* be implemented in a distributed fashion. Techniques that perform congestion control by estimating capacity in wired networks have been proposed before, *e.g.,* [29], but wireless capacity estimation is significantly harder. WCPCap is the first attempt in that direction that does not rely on heuristics, but instead uses a precise analytical methodology to accurately estimate the available capacity. Figure 2 shows that, for the topology of Figure 1, WCPCap achieves a max-min fair rate.

Using analysis, simulations, and real deployments, we find that our designs yield rates that are both fair and efficient. WCP is fairer than TCP, while WCPCap is max-min fair in all the topologies we study and achieves goodputs within 15% of the optimal. WCP achieves consistently good performance in the topologies we study while being extremely easy to implement. In fact, our experiments using five flows in a 14-node testbed show that, while TCP starves one or two of these flows in each run, WCP assigns fair rates to all the flows. Finally, in addition to good throughput performance, WCPCap exhibits low end-to-end delay and fast convergence.

## 2. RELATED WORK

Extensive research has been done to understand the shortcoming and to improve the performance of TCP in wireless networks [48, 19, 10, 23, 52, 30, 36, 51]. We briefly discuss broad classes of research pertinent to our work while referring interested reader to [37] for a more comprehensive survey of congestion control in wireless networks.

Early work on improving TCP performance in wireless networks focused on distinguishing between packet loss due to wireless corruption from loss due to congestion, in the context of last-hop wireless [8, 7] or wireless wide-area networks [45]. In contrast, we address congestion control for multi-hop wireless networks.

More recent work, however, has addressed congestion control for mobile ad-hoc wireless networks. One class of work, exemplified by TCP-F [10], TCP-ELFN [23], TCP-BuS [30], ATCP [36], and EPLN/BEAD [52], concentrates on improving TCP's *throughput* by freezing TCP's congestion control algorithm during link-failure induced losses, especially when route changes occur. Individual pieces of work differ in the manner in which these losses are identified and notified to the sender and in their details of freezing TCP. For example, TCP-ELFN [23] explicitly notifies the TCP sender of routing failure causing the sender to enter a standby mode. The sender re-enters the normal TCP mode on route restoration, identified using periodic probe messages. Unlike WCP, these proposals do not explicitly recognize and account for congestion within a neighborhood. As a result, they would exhibit the same shortcomings of TCP as discussed in Section 1.

Another class of work related to WCP includes schemes like COPAS [12], LRED [19], and ATP [48], which discuss TCP performance issues even in ad-hoc networks with no link-failure induced losses. COPAS [12] proposes a route selection scheme that attempts to find disjoint paths for different flows by assigning weights to links proportional to the average number of backoffs on the link. LRED [19] uses a exponential weighted moving average of the number of retransmissions at the MAC layer as a measure of congestion while marking packets in a manner similar to RED [18]. ATP [48], like WCP, is a rate-based congestion control scheme that involves explicit rate feedback to the sources from the network. In ATP, a flow receives the maximum of the weighted average of the sum of the queueing and transmission delay at any node traversed by the flow. ATP uses the inverse of this delay as the sending rate of a sender. Even though these schemes do not recognize the need of congestion detection and signaling over a neighborhood, their congestion metric *implicitly* takes some degree of neighborhood congestion into account. However, congestion in wireless networks exhibits strong location dependency [51] *i.e.,* different nodes in a congested neighborhood *locally* perceive different degrees of congestion. In the above schemes, flows traversing different nodes in a single congested neighborhood would receive varying levels of congestion notification. In contrast, WCP explicitly shares congestion within a neighborhood, ensuring that each flow in a single congested neighborhood gets its fair share of the bottleneck bandwidth.

Three other pieces of work, however, have recognized the importance of explicitly detecting and signaling congestion over a neighborhood. NRED [51] identifies a subset of flows which share channel capacity with flows passing through a congested node. But, it identifies only a subset of contending flows: it misses flows that traverse two hop neighbors of a node without traversing its one hop neighbors (for example, the flow traversing $7 \rightarrow 9$ in Fig. 3, Section 3). Moreover, the mechanism to regulate the traffic rates on these flows is quite a bit more complex than ours (it involves estimating a neighborhood queue size, and using RED [18]-style marking on packets in this queue). Finally, unlike WCP, NRED requires RTS/CTS, is intimately tied to a particular queue management technique (RED), might require special hardware for channel monitoring, and has not been tested in a real implementation. EWCCP [49] correctly identifies the set of flows which share channel capacity with flows passing through a congested node. EWCCP is designed to be proportionally-fair, and its design as well as its proof

of correctness assumes that the achievable rate region of 802.11 is convex. As Figure 2 shows, however, this is not necessarily true. Moreover, EWCCP [49] has also not been tested in a real implementation. Finally, our own IFRC [43] is an interference-aware fair rate control scheme designed for many-to-one communication, e.g. when a number of sensors send measurements towards a common sink. IFRC's design takes advantage of the tree-structured topology and many-to-one traffic pattern and cannot be used in a general, many-to-many communication setting.

As a final note, our AIMD-based scheme WCP borrows heavily from TCP's essential features such as ECN, SACK, and round-trip time estimation [25,17], and uses some well established approaches from the active queue management literature [18,33] to detect congestion at a node.

An alternative to AIMD based schemes are schemes in which intermediate routers send explicit and precise feedback to the sources. XCP [29] and RCP [15] are examples of such schemes for wired networks. Such schemes cannot be directly extended to multi-hop wireless networks, since the available capacity at a wireless link depends on the link rates at the neighboring edges, and ignoring this dependence will overestimate the available capacity and lead to performance degradation [40] and eventually to instability. Variants of XCP for wireless multi-hop networks, like WXCP [47] and XCP-b [4], use heuristics based on measuring indirect quantities like queue sizes and the number of link layer retransmissions, to reduce the overestimation in the available capacity. If, instead, one can directly estimate the exact capacity of a link as a function of the link rates at the neighboring edges, then an accurate XCP-like scheme can be implemented for wireless multi-hop networks.

In 802.11-scheduled multi-hop networks, the complex interference among nodes makes it very hard to estimate the capacity of a link. Results have been known either for multi-hop networks that use perfect MAC schedulers [26,32], or for single-hop 802.11-scheduled networks under saturation traffic conditions [9,44]. We have recently developed an analytical methodology which characterizes the achievable rate region of 802.11-scheduled multi-hop networks [28,27]. Our second scheme, WCPCap, uses this prior work of ours to find the supportable per-flow rate in a neighborhood. Further, it uses a novel, decentralized mechanism that relies on message exchanges within local neighborhoods only, to calculate the end-to-end flow rates.

Related to WCPCap is an interesting line of work that has explored theoretical methods for jointly optimizing scheduling and rate assignment in wireless networks [16,35,46,39]. Unlike this body of work, we restrict the scheduler to be 802.11. Moreover, we restrict our explorations to plausibly implementable rate-control mechanisms, whereas this line of research yields schemes that require a centralized implementation to optimize both scheduling and rate assignment. While optimized rate assignment (congestion control) can be done in a distributed fashion by using back-pressure techniques [50], it still requires every node in the network to maintain separate queues for each possible network destination. More recent practical studies of the problem have not been able to relax [5] this requirement. Recently, Li *et al.* [34] have explored theoretical methods to set up centralized optimization problems for 802.11-scheduled multi-hop networks to find rate allocations achieving a given objective, like max-min fairness. However, they do not discuss how to achieve these allocations through distributed rate control schemes.

Finally, there has been a growing interest in industry [6] and academia [31] in using multiple radios per node, in an effort to mitigate or nullify the complex interference found in multi-hop wireless networks. This line of work is orthogonal to our efforts. We believe that in dense deployments our work will be relevant even if multiple radios are used, since the large number of channels required to completely avoid interference, as well as the complexity associated with their scheduling, would be prohibitively expensive.

# 3. DESIGN

In this section, we first discuss the design and implementation of WCP, an AIMD-based rate-control protocol that incorporates many of the features of TCP, but differs significantly in its congestion control algorithms. We then describe WCPCap which incorporates wireless capacity estimation in order to assign fair and efficient rates to flows.

## 3.1 WCP

WCP is a rate-based congestion control protocol for static multi-hop wireless mesh networks which use the 802.11 MAC. In WCP, for every flow, the source maintains a rate $r$ which represents the long term sending rate for the flow. WCP is AIMD-based, so that the source additively increases $r$ on every ack reception and multiplicatively decreases $r$ upon receiving a congestion notification from routers (intermediate forwarding nodes). Routers signal congestion by setting a congestion bit in the packet header of ongoing packets. Unlike existing congestion control techniques, WCP has novel algorithms for detecting and signaling congestion at the intermediate routers, as well as for adapting rates at sources in response to congestion signals.

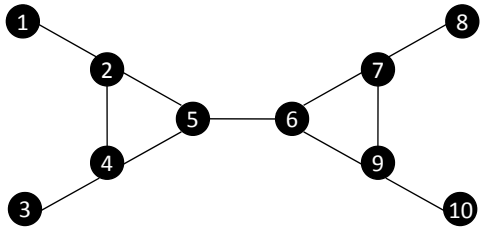


**Figure 3: Congestion neighborhood**

**Congestion in Multi-hop Wireless Networks.** The central observation underlying the design of WCP is that the nature of congestion in a wireless network is qualitatively different from that in a wired network. In a wireless network, since neighboring nodes share the wireless channel, the available transmission capacity at a node can depend on traffic between its neighbors.

More precisely, congestion in wireless networks is defined not with respect to a node, but with respect to transmissions from a node to its neighbor. In what follows, we use the term *link* to denote a one-hop sender-receiver pair. (We use the terms sender and receiver to denote one-hop transmissions, and source and destination to denote the endpoints of a flow). Thus, in Figure 3, we say that a transmission from 5 to 6 is along the link $5 \rightarrow 6$. Consider the following example. When 5 is transmitting to node 6 it shares the wireless channel with any transmission from node 7, say a transmission from node 7 to node 9, as that transmission can collide with a transmission from node 5 to node 6. However, when node 5 is transmitting to node 2 it *does not* share capacity with, for example, a transmission from node 7 to node 9. Thus, congestion in wireless networks is defined not with respect to a node $i$, but with respect to a link $i \rightarrow j$.

What, then, are the set of links ($L_{i \rightarrow j}$) that share capacity with a given link ($i \rightarrow j$)? Consider link $5 \rightarrow 6$ in Figure 3. Clearly, all outgoing links from node 5 and node 6 share capacity with link

$5 \rightarrow 6$. Moreover, every outgoing link from a one-hop neighbor of node 5 shares capacity with link $5 \rightarrow 6$ because any transmission from a neighbor of 5, say node 2, can be sensed by node 5 and would prevent node 5 from capturing the channel while node 2 is transmitting. Additionally, any incoming link to any neighbor of node 5, say $1 \rightarrow 2$, also shares capacity with link $5 \rightarrow 6$ as the link-layer acknowledgement from node 2 to node 1 would also prevent node 5 from capturing the channel for transmission. Similarly, any outgoing link from a neighbor of node 6 shares capacity with link $5 \rightarrow 6$ as any transmission along the outgoing link of neighbor of 6 can collide with transmission along $5 \rightarrow 6$. Finally, any incoming link into a neighbor of node 6, say $8 \rightarrow 7$, also shares capacity with $5 \rightarrow 6$ as the link-layer acknowledgement from node 7 to node 8 can collide with transmissions along $5 \rightarrow 6$.

Thus, $L_{i \rightarrow j}$, for a mesh network using an 802.11 MAC is defined as:

> the set of all incoming and outgoing links of $i$, $j$, all neighbors of $i$, and all neighbors of $j$.

Note that $L_{i \rightarrow j}$ includes $i \rightarrow j$. Moreover, this relationship is symmetric. If a link $i \rightarrow j$ belongs to $L_{k \rightarrow l}$, $k \rightarrow l$ also belongs to $L_{i \rightarrow j}$. Furthermore, this definition is valid even when RTS-CTS is used. However, there is an important limitation in our definition. If a node is outside another node's transmission range, but within its interference range, WCP cannot account for the reduction in channel capacity as a result of the latter's transmissions.

**Congestion Detection and Sharing.** In WCP, one key idea is *congestion sharing*: if link $i \rightarrow j$ is congested, it shares this information with all links in $L_{i \rightarrow j}$. Packets traversing those links (as well as link $i \rightarrow j$ itself) are marked with an explicit congestion notification, so that sources can appropriately adapt the rates of the corresponding flows. We now describe how routers detect congestion, and how they share their congestion state.

Congestion detection in WCP is deliberately simple. A router detects congestion on its outgoing link using a simple thresholding scheme. It maintains an exponentially weighted moving average (EWMA) of the queue size for every *outgoing link*. A link is congested when its average queue size is greater than a congestion threshold $K$. Various other congestion detection techniques have been explored in wireless networks: channel utilization [51], average number of retransmissions [20], mean time to recover loss [42], among others. We choose queue size as a measure of congestion for two reasons: it has been shown to work sufficiently well in wireless networks [43, 24]; and is a more natural choice for detecting congestion per link compared to, say, channel utilization which measures the level of traffic around a node. Also, more sophisticated queue management schemes are possible (*e.g.,* RED or AVQ), but they are beyond the scope of this paper.

When a router $i$ detects that $i \rightarrow j$ is congested, it needs to share this information with nodes at the transmitting ends of links in $L_{i \rightarrow j}$ (henceforth referred to as nodes in $L_{i \rightarrow j}$). These nodes are within two hops of the congested link. We omit a detailed description of the protocol that implements this congestion sharing. However, we will note that this protocol can be implemented efficiently. Congestion state information can be piggybacked on outgoing packets, which neighbors can snoop. Furthermore, this information needs to be transmitted only when a link is congested; moreover, if more than one incoming link at a node is congested, it suffices for the node to select (and inform its neighbors) of only one of these.

Finally, when a node in $L_{i \rightarrow j}$ detects that link $i \rightarrow j$ is congested, it marks all outgoing packets on that link with an explicit congestion indicator (a single bit).

**Rate Adaptation.** In WCP sources perform rate adaptation. While the principles behind our AIMD rate adaptation algorithms are relatively standard, our contribution is to correctly determine the timescales at which these operations are performed. The novel aspect of our contribution is that these timescales are determined by the RTTs of flows traversing a congested neighborhood; without our innovations (described below), flows do not get a fair share of the channel, and sometimes react too aggressively to congestion.

A source $S$ in WCP maintains a rate $r_m$ for every flow $f_m$ originating at $S$. It linearly increases the rate $r_m$ every $t_{ai}$ seconds, where $t_{ai}$ is the control interval for additive increase:

$$r_m = r_m + \alpha$$

where $\alpha$ is a constant. The choice of $t_{ai}$ is an important design parameter in WCP. In the above equation the rate of change of $r_m$, $dr_m/dt$ is $\alpha/t_{ai}$. Intuitively, for stable operation, $dr_m/dt$ should be dependent on feedback delay of the network. Using the weighted average round-trip time, $rtt_m^{avg}$, of the flow seems an obvious choice for $t_{ai}$ as it satisfies the above requirement. But consider three flows $1 \rightarrow 3$, $4 \rightarrow 6$, and $7 \rightarrow 9$ in Figure 1. Packets of flow $1 \rightarrow 3$ share the wireless channel with nodes 1 through 6 while packets from flow $4 \rightarrow 6$ share wireless channel with all the nodes in the figure. As the rate of all the flows in the network increases, flow $4 \rightarrow 6$ experiences more contention as compared to flow $1 \rightarrow 3$ and the average RTT of flow $4 \rightarrow 6$ increases much faster than the average RTT of flow $1 \rightarrow 3$. Thus, even if these flows were to begin with the same rate, their rates would diverge with the choice of $t_{ai} = rtt_m^{avg}$. For fairness, all flows sharing a congested neighborhood should have similar $dr_m/dt$.

To enable fairness, WCP introduces the notion of a *shared RTT*. Denote by $rtt_{i \rightarrow j}^{avg}$ the average RTT of all the flows traversing the link $i \rightarrow j$ (the average RTT of each flow is computed by the source, and included in the packet header) *i.e.,*

$$rtt_{i \rightarrow j}^{avg} = \sum_{\forall m \in F_{i \rightarrow j}} \frac{rtt_m^{avg}}{|F_{i \rightarrow j}|}$$

where $F_{i \rightarrow j}$ is the set of flows traversing link $i \rightarrow j$. These "link RTTs" are disseminated to all nodes in $L_{i \rightarrow j}$ in a manner similar to congestion sharing (as described above), and can also be efficiently implemented (details of which we omit). For link $i \rightarrow j$, node i computes the shared RTT $rtt_{i \rightarrow j}^{Smax\_avg}$ as the maximum RTT among all links in $L_{i \rightarrow j}$ *i.e.,*

$$rtt_{i \rightarrow j}^{Smax\_avg} = \max_{\forall k \rightarrow l \in L_{i \rightarrow j}} (rtt_{k \rightarrow l}^{avg})$$

In words, this quantity measures the largest average RTT across the set of links that $i \rightarrow j$ shares the channel with. Why this particular choice of timescale? Previous work has shown that the average RTT of flows is a reasonable control interval for making congestion control decisions [29]. Our definition conservatively chooses the largest control interval in the neighborhood.

For all flows traversing link $i \rightarrow j$, the router includes $rtt_{i \rightarrow j}^{Smax\_avg}$ in every packet header only if it exceeds the current value of that field in the header. The source uses this value for $t_{ai}$: thus, $t_{ai}$ is the largest shared RTT across all the links that the flow traverses. This value of control interval $t_{ai}$ ensures that all flows going through the same congested region increase their rates at the same timescale. If a flow traverses multiple congested regions, its rate increase is clocked by the neighborhood with the largest shared RTT.

Upon receiving a packet with a congestion notification bit set, a source reduces the rate $r_m$ as $r_m = r_m/2$ and waits for a control interval $t_{md}$ before reacting again to any congestion notification from

the routers. $t_{md}$ must be long enough so that the source has had time to observe the effect of its rate reduction. Moreover, for fairness, flows that traverse the congested region must all react at roughly the same timescale. To ensure this, WCP also computes a quantity for each link that we term the *shared instantaneous RTT*, denoted by $rtt_{i\to j}^{Smax\_inst}$. This is computed in exactly the same way as the shared RTT, described above, except that the instantaneous RTT is used, rather than the average RTT. The former is a more accurate indicator of the current level of congestion in the network and is a more conservative choice of the timescale required to observe the effect of a rate change. As before, routers insert this shared instantaneous RTT into the packet header only if it exceeds the current value. Sources set $t_{md}$ to be the value of this field in the packet header that triggered the multiplicative decrease. If a flow traverses multiple congested regions, its multiplicative decreases are clocked by the neighborhood with the largest shared instantaneous RTT.

Finally, we describe how the source uses the value $r_m$. WCP aims to assign fair *goodputs*. Naively sending packets at the rate $r_m$ assigns fair throughputs, but packet losses due to channel error or interference can result in unequal goodputs. Instead, WCP sends packets at a rate $r_m/p_m$, when $p_m$ is the empirically observed packet loss rate over the connection. Intuitively, this *goodput correction* heuristic sends more packets for flows traversing lossy paths, equalizing flow goodputs. Other rate-based protocols [41] use more sophisticated loss rate computation techniques to perform similar goodput correction. As we show in Section 4, our approach works extremely well for WCP. In that section, we also quantify the impact of turning off this "correction".

**Implementation.** We could have retrofitted congestion and RTT sharing in TCP. But, the complexity of current TCP implementations, and the fact that TCP performs error recovery, congestion control and flow control using a single window-based mechanism, made this retrofit conceptually more complex. Given that our goal was to understand the issues underlying congestion in mesh networks, incremental deployability was not paramount. So, at the cost of some additional packet header overhead, we decided to explore a clean-slate approach. Our implementation uses a rate-based protocol for congestion control (as described above), but uses an implementation of TCP SACK for error recovery, a window-based flow control mechanism exactly like TCP, and the same RTO estimation as in TCP. In a later section, we show that our implementation does not bias the results in any way: if we remove our sharing innovations from WCP, its performance is comparable to TCP. There is no fundamental reason to consider a clean-slate design of TCP: our choice was dictated by convenience.

## 3.2 WCPCap

An alternative to WCP is a protocol in which the network sends explicit and precise feedback to the sources. In order to do this, it is important to be able to estimate the available capacity within a congested region, a non-trivial task. In this section, we describe WCPCap, a protocol that provides explicit feedback (in much the same way that XCP [29] and RCP [15] do for wired networks). An important goal in designing WCPCap is to explore the feasibility of capacity estimation using only local information, thereby making it amenable to distributed implementation.

**Determining the Achievable Link-Rate Region.** At the core of WCPCap is a technique to determine whether a given set of rates is *achievable* in an 802.11 network; using this technique, WCPCap estimates the available capacity and distributes this fairly among relevant flows. This technique is presented in detail in our prior work [28, 27]. For completeness, we describe the main idea of the analytical methodology here, assuming IEEE 802.11 scheduling with RTS/CTS in the network.

The precise goal of the technique is as follows. Given a link $i \to j$, and a set of candidate aggregate rates $r_{l\to m}$ over links $l \to m$ belonging to $L_{i\to j}$ (link $i \to j$ belongs to this set), we seek a decision procedure that will enable us to determine if these rates are achievable. The decision process assumes that the channel loss rates (losses not due to collisions) of links in $L_{i\to j}$, and the interference graph between links in $L_{i\to j}$ are known. Channel losses are assumed to be independent Bernoulli random variables. The interference model neglects some physical layer phenomena like the capture effect [11] (where a receiver can correctly decode data despite of interference), situations where transmitters send data despite of interference and carrier sensing, and situations where remote links in isolation do not interfere with the link under study, but their aggregate effect may cause loses on the link [14]. The interested reader is referred to [27] for a detailed description of all the assumptions, an extensive evaluation of their effect on the accuracy of the model, and a discussion on how to remove them.

The decision process first determines, for each link, the expected service time in terms of (a) the collision probability at the receiver and (b) the idle time perceived by the transmitter of that link. It does this by solving an absorbing Markov Chain which tracks the evolution of the state of the transmitter during a packet service [28, 27]. Given these service times, the given set of link-rates is achievable only if

$$\sum_{e\in O_v} \lambda_e E[S_e] \leq U, \forall v \in V$$

where $V$ is the set of all nodes, $O_v$ is the set of outgoing links from a node $v \in V$, $\lambda_e$ is the packet arrival rate at link $e$, $E[S_e]$ is the expected service time of a packet at link $e$, and the *utilization factor* $U$ is a fraction between 0 and 1 and reflects the desired utilization of the channel. In practice, $U$ is usually set to less than 1 to keep the system stable. Otherwise, small non-idealities can drive the network beyond the capacity region. Furthermore, this summation is taken over all outgoing links because these links share the same queue. This incorporates the following head of line blocking effect. Consider the queue at node 5 in Figure 3. A packet destined for link $5 \to 2$ can get trapped behind a packet destined for link $5 \to 6$ which is not scheduled due to a simultaneous transmission at link $7 \to 9$.

The key challenge then, is to determine the collision and the idle time probabilities, made difficult because these values for a link depend on the rates at the neighboring links. We use the following procedure: the sub-graph formed by the set of links in $L_{i\to j}$ is *decomposed* into a number of two-link topologies and the collision and idle probabilities for each two-link topology is derived. The net probabilitity is found by appropriately *combining* the individual probabilities from each two-link topology. We now describe these steps.

*Two-link topologies.* There can exist four types of two-link topologies in an 802.11 network [22, 21]. To describe these, we use the following notation: let $l_1$ and $l_2$ denote the two links under consideration and let $T_{l_j}$ and $R_{l_j}$, $j = 1, 2$, denote the transmitter and the receiver of the respective links We use the Stack topology introduced in Figure 1 to give examples of each category.

*Coordinated Stations*: in which $T_{l_1}$ and $T_{l_2}$ can hear each other. For example, links $4 \to 5$ and $5 \to 6$ in the Stack topology form a coordinated station. *Near Hidden Links:* in which $T_{l_1}$ and $T_{l_2}$ cannot hear each other, but there is a link between $T_{l_1}$ and $R_{l_2}$ as well as one between $T_{l_2}$ and $R_{l_1}$. For example, links $4 \to 5$ and $6 \to 5$ in the Stack topology form near hidden links. *Asymmetric*

*Topology:* in which $T_{l_1}$ and $T_{l_2}$ as well as $T_{l_1}$ and $R_{l_2}$ cannot hear each other, but $T_{l_2}$ and $R_{l_1}$ are within each other's range. Thus $T_{l_2}$ is aware of the channel state as it can hear the CTS from $R_{l_1}$, but $T_{l_1}$ is unaware of the channel state as it can hear neither the RTS nor the CTS from the transmission on $l_2$. For example, link $4 \rightarrow 5$ forms an asymmetric topology with link $2 \rightarrow 3$. *Far Hidden Links*: in which only $R_{l_1}$ and $R_{l_2}$ are within each others' range. For example, links $4 \rightarrow 5$ and $1 \rightarrow 2$ form a far hidden link topology.

The first step in our procedure is to decompose a general topology into its constituent two-link topologies. This is easily achieved by evaluating how each link in $L_{i \rightarrow j}$ interferes with $i \rightarrow j$, based on the definitions stated in the previous paragraph.

*Finding collision and idle probabilities for two-link topologies.* The next step is to find the collision and the idle probabilities for each two-link topology.

In multi-hop topologies with RTS/CTS enabled, both RTS and DATA packets can be lost due to collision. For coordinated stations and near hidden links, an RTS collision takes place if the two links start transmitting at the same time. For near hidden links, an RTS collision can also take place if a node starts transmitting an RTS while an RTS transmission is ongoing on the other link. For asymmetric topologies where transmitters have an incomplete view of the channel, and for far hidden links, the receiver of the link will not send back a CTS whenever there is a transmission ongoing at the other link. By similar arguments, it is easy to see that, for DATA packets, collisions cannot happen in coordinated stations or near hidden links, but can happen for asymmetric topologies and far hidden links.

Finally, the idle probability for each link can be derived based on the following observation. The channel around the transmitter of a link is busy if there is a transmission ongoing at any one of the following links: links which form a coordinated station, a near hidden link or an asymmetric topology with the link under consideration having a complete view of the channel.

We will describe the basic derivation steps for these probabilities when we discuss how to combine the individual probabilities from each two-edge topology. However, for brevity, we will omit the analytical formulas and their complete derivations. The interested reader is referred to [28, 27] for details.

*Combining collision and idle probabilities.* The final step is to combine the probabilities obtained from each two-link topology. This step must account for the dependence between the neighboring links. For example, in the Stack topology links $2 \rightarrow 3$ and $2 \rightarrow 1$ which are both in $L_{4 \rightarrow 5}$, cannot be scheduled simultaneously because they have the same transmitter, and hence the individual probabilities from these two two-link topologies cannot be combined independently. To understand when individual probabilities can be combined independently and when they cannot, we consider two cases.

The first case corresponds to the situations where probabilities can be independently combined. The RTS collision probability due to coordinated station and near hidden links, and the DATA collision probability due to asymmetric topologies and far hidden links belong to this case. In all these situations, the corresponding computation depends on the probability of two (or more) transmitters starting transmission at the same time. The probability of this event in turn depends on the probability of the backoff counters at these links being equal to a specific value. Let $p^e_{w_0}$ denote the probability that the backoff counter at link $e$ is equal to 0. And let $\lambda_e$ and $E[S_e]$ denote the packet arrival rate and expected packet service time at link $e$. Then, one can show that

$$\lambda_e E[S_e] \frac{2}{W_m + 1} \leq p^e_{w_0} \leq \lambda_e E[S_e] \frac{2}{W_0 + 1},$$

where $W_0$ is the minimum backoff window value and $W_m$ is the maximum backoff window value. These two bounds can be used to compute the required probabilities with high accuracy.

The second case is more involved. The corresponding computation depends on the probability of the event that there is no ongoing transmission among a set of links. The RTS collision probability due to asymmetric and far hidden links and the idle probability belong to this category. For example, for link $4 \rightarrow 5$ in the Stack topology, to find the RTS collision probability, one needs to find the probability that there is no ongoing transmission on links $1 \rightarrow 2, 2 \rightarrow 3, 3 \rightarrow 2, 2 \rightarrow 1, 7 \rightarrow 8, 8 \rightarrow 9, 9 \rightarrow 8$ and $8 \rightarrow 7$. These probabilities cannot be independently combined. Here are two examples of dependencies between these links which require careful handling. If two links interfere with each other, like links $1 \rightarrow 2$ and $2 \rightarrow 3$, then they cannot be simultaneously scheduled. If two links do not interfere with each other, like $2 \rightarrow 3$ and $8 \rightarrow 9$, then they can be indepedently scheduled given that none of the links which interfere with both (links $4 \rightarrow 5, 5 \rightarrow 6, 6 \rightarrow 5$ and $5 \rightarrow 4$) are transmitting. In general, let $X_e$ denote the event that there is a transmission going on at edge $e$, $N$ denote a set of edges, $N_s$ denote a subset of $N$, and $S_{N_s}$ denote the set of edges which interfere with all the edges in $N_s$. Then, basic probability dictates that

$$P(\cup_{e_n \in N} X_{e_n}) = \sum_{e_i \in N} P(X_{e_i}) - \sum_{e_i, e_j \in N} P(X_{e_i} \cap X_{e_j})$$

$$+ \ldots + (-1)^{|N|-1} P(\cap_{e_i \in N} X_{e_i}),$$

and to compute the individual terms in this expression note that $P(\cap_{e_i \in N_s} X_{e_i}) = 0$ if any two edges in $N_s$ interfere with each other, and it equals

$$\left( \prod_{e_i \in N_s} P(X_{e_i}) \right) / \left( 1 - \sum_{e_k \in S_{N_s}} P(X_{e_k}) \right)^{|N_s|-1}$$

otherwise. Finally, note that $P(X_e) = K_e \lambda_e T_s$ where $K_e$ is the average number of times a packet is transmitted (including retransmissions) and $T_s$ is the average packet transmission time.

We have briefly summarized the intuition behind the derivation of the collision and idle probabilities of each link when RTS/CTS is used. We omit a discussion of the derivation in the absence of RTS/CTS: a similar reasoning applies, but we only need to consider DATA packet collisions.

As we have said before, once the collision and idle probabilities are known at each link, we can compute the service times and determine whether a given set of rates is achievable.

**Estimating Available Bandwidth.** WCPCap uses the achievable rate computation technique to estimate achievable bandwidth and give precise rate feedback to sources. Conceptually, each router maintains, for each outgoing link $i \rightarrow j$, a rate $R_{i \rightarrow j}$ which denotes the maximum rate allowable for a flow passing through the link. However, a flow traversing $i \rightarrow j$ is actually only allowed to transmit at the minimum (denoted $R^{min}_{i \rightarrow j}$) of all rates $R_{k \rightarrow l}$ such that $k \rightarrow l$ belongs to $L_{i \rightarrow j}$ (intuitively, at the most constraining rate over all links that share channel capacity with $i \rightarrow j$). The rate feedback is carried in the packet header. When a packet traverses $i \rightarrow j$, the router sets the feedback field to $R^{min}_{i \rightarrow j}$ if $R^{min}_{i \rightarrow j}$ is lower than the current value of the field. This feedback rate is eventually delivered to the source in an end-to-end acknowledgement packet, and the

source uses this value to set its rate. Thus the source sets its rate to the smallest allowable rate in the wireless neighborhoods that it traverses.

$R_{i\to j}$ for each link is updated every $k \cdot rtt_{i\to j}^{Smax\_avg}$, where $rtt_{i\to j}^{Smax\_avg}$ is the shared RTT defined in Section 3.1 and $k$ is a parameter which trades-off the response time to dynamics for lower overhead. The duration between two successive updates of $R_{i\to j}$ is referred to as an epoch. During each epoch, transmitter $i$ measures $x_{i\to j}$, the actual data rate over link $i \to j$ and $n_{i\to j}$, the number of flows traversing link $i \to j$. Using $x_{k\to l}$ and $n_{k\to l}$ for all $k \to l$ in $L_{i\to j}$ transmitter $i$ computes the new value of $R_{i\to j}$ (denoted by $R_{i\to j}^{new}$) to be used in the next time epoch, and broadcasts $x_{i\to j}$, $n_{i\to j}$, and $R_{i\to j}^{new}$ to all nodes in $L_{i\to j}$.

We now describe how $R_{i\to j}^{new}$ is determined (Figure 4). Note that the transmitter $i$ has $x_{k\to l}$ and $n_{k\to l}$ for all links $k \to l$ in $L_{i\to j}$. It uses this information, and the methodology described above, to determine the maximum value of $\delta$ such that the rate vector $\vec{x}$ shown in Figure 4 is achievable . ($\delta$ can have a negative value if the current rates in the neighborhood are not achievable.) Then, node $i$ sets $R_{i\to j}^{new}$ to $R_{i\to j} + \rho\delta$ if $\delta$ is positive, else $R_{i\to j}^{new}$ is set to $R_{i\to j} + \delta$. We use a scaling factor $\rho$ while increasing the rate to avoid big jumps, analogous to similar scaling factors in XCP and RCP. On the other hand, we remain conservative while decreasing the rate. Each node independently computes $R_{k\to l}$ for its links. These computations do not need to be synchronized, and nodes use the most recent information from their neighbors for the computation.

The computational overhead of the WCPCap algorithm is very low. To determine $R_{i\to j}^{new}$, we perform a binary search to find the maximum value of $\delta$ such that the rate vector $\vec{x}$ is achievable. Each iteration decomposes $L_{i\to j}$ into two-link topologies, computes collision and idle probabilities for each two-link topology, and combines the results. Overall, the algorithm requires a logarithmic number of iterations whose complexity is polynomial in $|L_{i\to j}|$. In practical topologies the cardinality of $L_{i\to j}$ is small. For example, in our experiments (run on 3.06GHz Linux boxes) determining $R_{i\to j}^{new}$ takes as much time as it takes to send a data packet. Since each epoch consists of about 30 data packet transmissions and a single $R_{i\to j}^{new}$ computation, the computational overhead per epoch is very low.

Finally, we note that, if naively designed, WCPCap can impose significant communication overhead. For each link $i \to j$, the following information needs to be transmitted to all nodes in $L_{i\to j}$ once every epoch: the maximum RTT across the flows passing through the link, the actual data rate at the link, the number of flows passing through the link and $R_{i\to j}$. There are ways to optimize this, by quantizing the information or reducing the frequency of updates, but we have left these to future work. Instead, in our simulations, we assume that all the relevant information is available at each node without cost, since our goal has been to understand whether available bandwidth estimation using only local information is plausibly implementable in wireless networks.

# 4. SIMULATION RESULTS

In this section we evaluate the performance of WCP and WCPCap in simulation, and in the next we report on results from real-world experiments of WCP.

## 4.1 Methodology

We have implemented WCP and WCPCap using the Qualnet simulator [3] version 3.9.5. Our WCP implementation closely follows the description of the protocol in Section 3.1. Our WCPCap implementation, on the other hand, does not simulate the exchange

Every $k \cdot rtt_{i\to j}^{Smax\_avg}$ sec
  Find max $\delta$ such that
  $\vec{x} \leftarrow (\ x_{k\to l} + n_{k\to l}\delta \quad \text{for } k \to l \in L_{i\to j}\ )$
  is achievable
    $R_{i\to j}^{new} \leftarrow \begin{cases} R_{i\to j} + \rho\delta & \delta > 0 \\ R_{i\to j} + \delta & \delta \le 0 \end{cases}$
  Broadcast $R_{i\to j}^{new}$, $x_{i\to j}$ and $n_{i\to j}$
    to all links in $L_{i\to j}$

**Figure 4: Pseudo-code for rate controller at link $i \to j$**

of control messages at the end of each epoch; rather, this control information is made available to the relevant simulated nodes through a central repository. This ignores the control message overhead in WCPCap, so our simulation results overestimate WCPCap performance. This is consistent with our goal, which has been to explore the feasibility of a wireless capacity estimation technique.

All our simulations are conducted using an unmodified 802.11b MAC (DCF). We use default parameters for 802.11b in Qualnet unless stated otherwise. Auto-rate adaption at the MAC layer is turned-off and the rate is fixed at 11Mbps. Most of our simulations are conducted with zero channel losses (we report on one set of simulations with non-zero channel losses), although packet losses due to collisions do occur. However, we adjusted the carrier sensing threshold to reduce interference range to equal transmission range in order to generate interesting topologies to study.

On this set of topologies (described below), we run bulk transfer flows for 200 seconds for WCP, WCPCap, and TCP. Our TCP uses SACK with ECN, but with Nagle's algorithm and the delayed ACK mechanism turned off; WCP implements this feature set. (We have also evaluated TCP-Reno on our topologies. The results are qualitatively similar.) Congestion detection for all three schemes uses the average queue size thresholding technique discussed in Section 3.1. Other parameters used during the runs are given in Table 1. We discuss the choice of parameter $U$ later, but our choice of $\alpha$ is conservative, ensuring small rate increases over the range of timescales we see in our topologies. This choice of $\alpha$ also works in our real-world experiments, but more experimentation is necessary to determine a robust choice of $\alpha$. For each topology we show results averaged over 10 runs.

| Parameter | Value |
|---|---|
| Congestion Threshold($K$) | 4 |
| EWMA Weight ($w_q$) | 0.02 |
| Router Buffer size | 64 packets |
| Packet Size | 512 bytes |
| Additive Increase Factor ($\alpha$) | 0.1 |
| Utilization Factor ($U$) | 0.7 |
| WCPCap epoch duration constant ($k$) | 10 |
| WCPCap scale factor ($\rho$) | 0.1 |

**Table 1: Parameters used in simulations**

We measure the goodput achieved by each flow in a given topology by TCP, WCP, and WCPCap, and compare these goodputs with the optimal max-min rate allocations for each topology. To compute these allocations, we observe that the methodology in Section 3.2 can be applied to a complete topology to characterize the achievable rate region for a collection of flows. Intuitively, we can view the service times and arrival rates on links, together with flow conservation constraints, as implicitly defining the achievable rate region for the topology. (Essentially, this is how we derive the achievable rate region in Figure 2). The max-min allocations can

then be found by searching along the boundary of the achievable rate region. Using this methodology, we are also able to find the congested links in a given topology: we simply simulate the optimal max-min rate allocations, and identify congested links as those whose queues are nearly fully utilized.

To understand the performance of WCP and WCPCap, we examine four topologies, with associated flows, as shown in Figures 1, 5, 6, and 7. Nodes connected by a solid line can hear each others' transmissions. (Since, in our simulations, we equalize interference range and transmission range, only nodes that can hear each others' transmissions share channel capacity with each other.) Arrows represent flows in the network. Congested links (determined using the methodology described above) are indicated with a symbol depicting a queue. Each of these four topologies has qualitatively different congestion characteristics, as we discuss below.

**Stack** (Figure 1) consists of a single congested region. $4 \rightarrow 5$ is the congested link, and all other links in the topology belong to $L_{4\rightarrow5}$. **Diamond** (Figure 5) contains two intersecting congested regions. $1 \rightarrow 2$ and $7 \rightarrow 8$ are both congested links. $L_{1\rightarrow2}$ includes all outgoing links from nodes 1 to 6 and $L_{7\rightarrow8}$ includes all outgoing link from node 4 to 9. **Half-Diamond** (Figure 6) contains two overlapping congested regions. $4 \rightarrow 5$ and $7 \rightarrow 8$ are congested, and $L_{7\rightarrow8}$ is a subset of $L_{4\rightarrow5}$. **Chain-Cross** (Figure 7) contains two congested regions, with four flows traversing one region, and two flows the other. $1 \rightarrow 2$ is a congested link, but $6 \rightarrow 7$ does not belong to $L_{1\rightarrow2}$. $4 \rightarrow 5$ and $4 \rightarrow 3$ are also congested, and $L_{4\rightarrow5}$ does include $6 \rightarrow 7$.

Finally, since WCP uses a rate-based implementation, it is important to ensure that its *baseline* performance is comparable to that of TCP. To validate this, we ran TCP and WCP on a chain of 15 nodes. WCP gets 20% *less* throughput on this topology; it is less aggressive than TCP. We also disabled RTT and congestion sharing in WCP, and ran this on all our topologies. In general, this stripped-down version of WCP gets qualitatively the same performance as TCP. For example, Figure 8 shows the goodputs achieved by each flow for the Stack topology. As expected, WCP without congestion and RTT sharing starves the middle flow, just as TCP does, although to a lesser extent since its rate increases are less aggressive than that of TCP.

## 4.2 Performance of WCP and WCPCap

We now discuss the performance of WCP and WCPCap for each of our topologies. In what follows, we use the notation $f_{i\rightarrow j}$ to denote a flow from node $i$ to node $j$.

**Stack** (Figure 9). The optimal (max-min) achievable rates for this topology are 300 kbps for all the flows. TCP, as described earlier, starves the middle flows ($f_{4\rightarrow6}$). Intuitively, in TCP, flows traversing links that experience more congestion ($4 \rightarrow 5$) react more aggressively to congestion, leading to lower throughput. WCP identifies the single congestion region in this topology ($L_{4\rightarrow5}$) and shares the rate equally among all the flows assigning about 250 kbps to all the flows. WCPCap, with its more precise rate feedback, assigns slightly higher rates to all the flows while still being max-min fair. Intuitively, one would expect the performance difference between WCPCap and WCP to be higher than what it is, since the latter's AIMD mechanism is more conservative than one which can directly estimate capacity. However, as we discuss below, WCPCap also needs to be conservative, setting its utilization factor $U$ to 0.7. Finally, WCP is within 20% and WCPCap is within 15% of the optimal achievable rate for this topology.

**Diamond** (Figure 10). The optimal achievable rates for this topology are 325 kbps for all the flows. TCP starves flows traversing the congested links in this topology. By contrast, WCPCap, assigns 300 kbps to all the flows achieving max-min fairness. WCP, however, assigns $f_{4\rightarrow6}$ approximately half the rate assigned to the other flows. This topology consists of two congested regions ($L_{1\rightarrow2}$ and $L_{7\rightarrow8}$) and $f_{4\rightarrow6}$ traverses both congested regions while the other two flows traverse only one. Roughly speaking, $f_{4\rightarrow6}$ receives congestion notification twice as often as the other flows, and therefore reacts more aggressively. Thus, WCP is *not* max-min fair. WCP appears to assign rates to flows in inverse proportion to the number of congested regions traversed.

**Half-Diamond** (Figure 11). The optimal max-min rates for this topology are 315 kbps for $f_{4\rightarrow6}$ and $f_{7\rightarrow9}$, and 335 kbps for $f_{1\rightarrow3}$; the asymmetry in this topology permits $f_{1\rightarrow3}$ to achieve a slightly higher rate. Relative to other topologies, TCP performs fairly well for this topology. WCPCap assigns max-min fair rates to all the flows, and is within 14% of the optimal. WCP assigns comparable rates to $f_{4\rightarrow6}$ and $f_{7\rightarrow9}$ as they traverse both congested regions $L_{4\rightarrow5}$ and $L_{7\rightarrow8}$. $f_{1\rightarrow3}$ achieves a higher rate as it traverses only one congested region ($L_{4\rightarrow5}$) but its rate is not twice the rate of the other flow . We conjecture that WCP achieves a form of fairness in which the rate allocations depend not only on the number of congested regions a flow passes through, but also the "intensity" of congestion in those regions. Understanding the exact nature of fairness achieved by WCP is left to future work.

**Chain-Cross** (Figure 12). The optimal rates for this topology are 420 kbps for $f_{6\rightarrow7}$ and 255 kbps for all other flows. TCP starves the flows traversing the most congested link $1\rightarrow 2$. WCPCap achieves a max-min allocation which is within 15% of the optimal. WCP achieves rates that depend inversely on the number of congested regions traversed, with $f_{1\rightarrow7}$ achieving lower goodput and $f_{1\rightarrow2}$, $f_{10\rightarrow11}$, $f_{8\rightarrow9}$ achieving equal rates. WCP is able to utilize available network capacity efficiently; $f_{6\rightarrow7}$ does not traverse $L_{1\rightarrow2}$ and gets higher goodput.

## 4.3 Discussion

**Impact of Physical Losses.** Thus far, we have assumed perfect wireless links in our simulations (losses do occur in our simulations due to collisions, however). Figure 13 shows the performance of WCP and WCPCap for the Stack with a loss rate of 10% on each link. The results are qualitatively similar to Figure 9. As expected, the goodputs drop by about 10% for WCP and WCPCap, as do the optimal rates. We have conducted similar experiments for the other topologies, but omit their results for brevity. We also illustrate the efficacy of goodput correction in dealing with packet loss (Section 3.1). Figure 14 shows the goodputs achieved in the Stack topology with 10% loss on all links, when goodput correction is disabled. Goodputs are no longer fair.

**In-Network Rate Adaptation.** The WCP AIMD rate control algorithms described in Section 3.1 are implemented at the source. These algorithms can also be implemented per flow within the network. Although this approach has scaling implications, it is still interesting to consider. It can reduce the feedback delay in control decisions, and a router can avoid reacting to congestion when the rate of a flow traversing it is lower than the rates of flows traversing the congested link. Indeed, in our simulations, such a scheme performs uniformly better than WCP implemented at the source. For example, for Chain-Cross (Figure 15) $f_{1\rightarrow7}$ gets the same rate as other flows in $L_{1\rightarrow2}$ improving overall fairness while preserving the higher goodput allocated to $f_{6\rightarrow7}$.

**Choice of $U$ in WCPCap** . Recall that $U$ denotes the maximum allowed utilization per queue in WCPCap. In simulations, we set its value to 0.7. We now justify this choice. The analysis described in

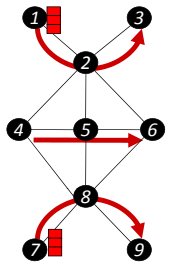


**Figure 5: Diamond topology**

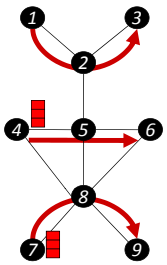


**Figure 6: Half-Diamond topology**

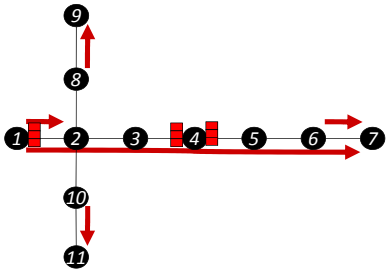


**Figure 7: Chain-Cross topology**

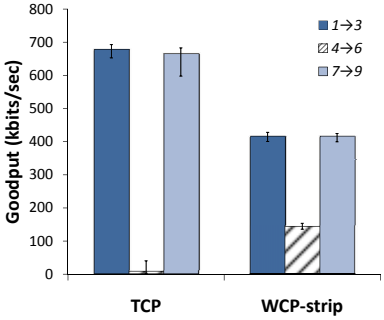


**Figure 8: WCP without congestion and RTT sharing, Stack**

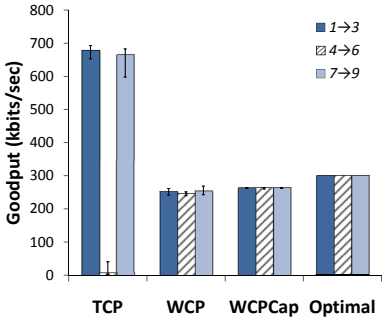


**Figure 9: WCP and WCPCap, Stack**

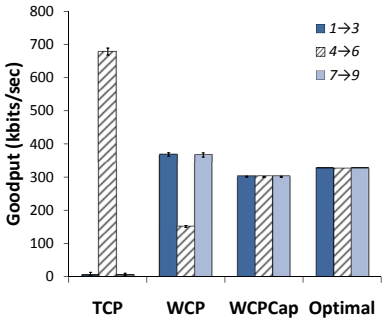


**Figure 10: WCP and WCPCap, Diamond**

Section 3.2 derives the achievable rate region without losses and hence assumes infinite buffer sizes and infinite MAC retransmit limits. Assuming no losses, operating very close to the capacity region will result in a large delays. However, in practice both the buffer sizes and MAC retransmit limits are finite. Hence, these large delays can result in significant losses. For this reason, we operate the network well within the capacity region; the parameter $U$ controls how far the network is from the boundary of the capacity region. To understand how to set its value, we plot the end-to-end delay of the middle flow, $f_{4\rightarrow6}$, in the Stack topology (which passes through the congested link $4 \rightarrow 5$) (Figure 16). We vary $U$ from 0 to 1 and assume very large buffer sizes and MAC retransmit limits. Ideally one would set $U$ near the knee of the curve, which is around 0.8. We choose a conservative estimate and set it to 0.7.

**RTS/CTS and WCP.** Our simulation results have used RTS/CTS so far. In Section 3.1, we asserted that our definition of congestion in a wireless network is insensitive to the use of RTS/CTS. Indeed, WCP without RTS/CTS (Figure 17) performs just as well as (and gets higher goodputs than) WCP with it (Figure 9). Other topologies show similar results, except for Half-Diamond (Figure 18). Without RTS/CTS, $1 \rightarrow 2$ and $7 \rightarrow 8$ become the most congested links in Half-Diamond, changing the dynamics of congestion in this topology. Qualitatively, this topology starts to resemble the Diamond, with two overlapping congested regions. A lower goodput for $f_{7\rightarrow9}$ than $f_{1\rightarrow3}$ results from additional links $4 \rightarrow 8$ and $6 \rightarrow 8$ in $L_{7\rightarrow8}$ which reduces the capacity in the region.

**RTS/CTS and WCPCap.** Disabling RTS/CTS has deeper implications for WCPCap. Consider the chain-cross topology of Figure 7. Without RTS/CTS, increasing the rate of flow $6 \rightarrow 7$ will cause more DATA collisions on link $4 \rightarrow 5$ as 4 is unaware of an ongoing transmission on link $6 \rightarrow 7$. As a result, more retransmissions occur on the link $4 \rightarrow 5$, and the total data rate (including retransmissions) on this link increases. Note that an increase in data rate on link $4 \rightarrow 5$ will cause more DATA collisions on link $2 \rightarrow 3$ as 2 is unaware of any ongoing transmission on link $4 \rightarrow 5$. This will reduce the capacity of link $2 \rightarrow 3$. Hence, the effective capacity of edge $2 \rightarrow 3$ decreases even though none of the flows passing through its neighborhood has changed its rate. Hence, finding the residual capacity at a link without rate information from non-neighboring links will overestimate the residual capacity. (By contrast, when RTS/CTS is used, the much smaller RTS packets collide, resulting in a less pronounced overestimation.) We can avoid overestimating the residual capacity by using a smaller value of $U$ (Section 3.2). For example, reducing it to 0.6 from 0.7 avoids any overestimation of capacity for the chain-cross topology. We use this value of $U$ for generating results for WCPCap without RTS/CTS (Figure 17 and 18). However, the choice of $U$ now depends to a greater extent on topology, and complete topology information is needed to compute it.

**Delay and Convergence.** Since WCPCap keeps the network within the achievable rate region, it is able to maintain smaller queues than WCP. Hence, WCPCap has smaller average end-to-end delay than WCP (Figure 19). The one exception is the Chain-Cross: since the throughput of flows $1 \rightarrow 7$ and $6 \rightarrow 7$ is much higher in WCPCap than WCP, the total traffic over $6 \rightarrow 7$ is much higher for WCPCap (Figure 12). This results in a higher delay for these two flows. Finally, WCPCap converges quickly; for all our topologies, it converges to within 10% of the final rate in less than 10 seconds. We have omitted these results for brevity.

**WCP Performance under Network Dynamics.** In the simulations above, all flows start and end at the same time. Figure 20 shows the instantaneous sending rate $r_m$ (Section 3) of all the flows in the Chain-Cross topology for a simulation where this is not the case. All flows start at $0s$ and end at $200s$ except for $f_{1\rightarrow7}$ which starts at $25s$ and ends at $100s$. In addition to being fair while all flows are active, WCP assigns fair rates to all the flows before the

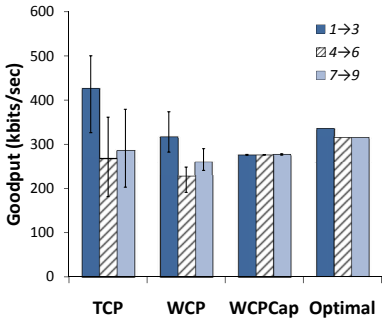


**Figure 11: WCP and WCPCap, Half-Diamond**

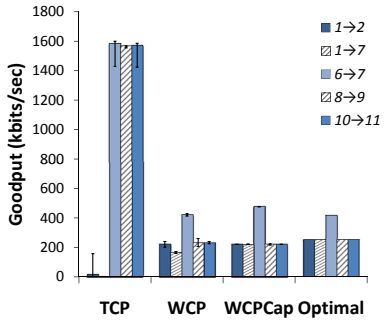


**Figure 12: WCP and WCPCap, Chain-Cross**

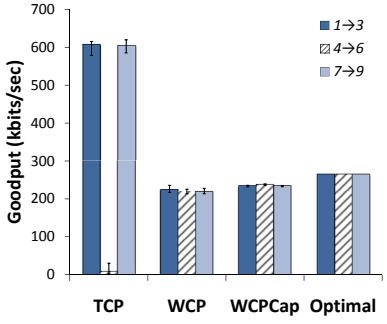


**Figure 13: WCP and WCPCap over lossy links, Stack**

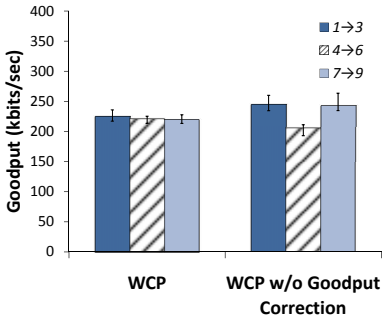


**Figure 14: WCP without goodput correction, Stack**

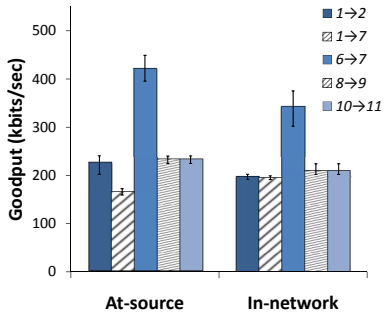


**Figure 15: WCP with in-network rate adaptation, Chain-Cross**

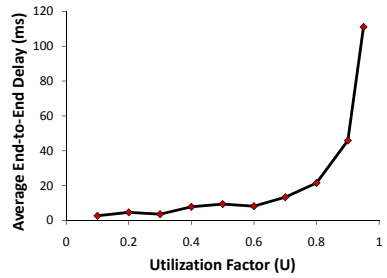


**Figure 16: Delay as a function of $U$ for $f_{4\to6}$, Stack**

arrival and after the departure of $f_{1\to7}$. A more extensive evaluation of WCP dynamics, such as its robustness to routing changes, is left to future work. Also left to future work is an evaluation of WCPCap's performance under network dynamics: our current simulation of WCPCap does not simulate control message exchange (Section 4.1) and it would be premature at this stage to understand its dynamics.

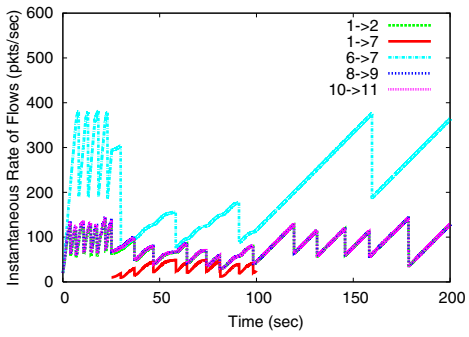


**Figure 20: WCP with delayed flow arrival**

## 4.4 Summary

WCPCap achieves max-min fair rate allocation for all topologies we study, while WCP allocates rates that appear to depend inversely on the number of congested neighborhoods traversed by a flow and the intensity of congestion in those regions. WCPCap is within 15% of the optimal for all topologies. For the Stack topology (where the max-min rate coincides with the proportionally fair rate), WCP is within 20% of the optimal. In addition, WCPCap exhibits low delay and fast convergence.

However, while WCP is implementable (indeed, we describe results from an implementation in the next section), some challenges need to be addressed before the same can be said of WCPCap: the potentially high overhead of control information exchange, the sensitivity of the choice of $U$ to the topology, and the ability to estimate the amount of interference from external wireless networks so that the collision probabilities can be correctly computed. None of these challenges are insurmountable, and we plan to address these as part of future work.

# 5. EXPERIMENTS

We have implemented WCP, and, in this section, report its performance on a real-world testbed. We first validate our simulations by recreating the Stack topology and showing that our experimental results are qualitatively similar to those obtained in simulation. We then demonstrate that WCP performs as expected on a 14 node topology running five flows in a real-world setting.

Our experiments use an ICOP eBox-3854, a mini-PC running Click [38] and Linux 2.6.20. Each node is equipped with a Senao NMP-8602 wireless card running the madwifi driver [1] and an omni-directional antenna. Wireless cards are operated in 802.11b monitor (promiscuous) mode at a fixed transmission rate of 11Mbps with 18dBm transmission power. RTS/CTS is disabled for the experiments. We empirically determined, at the beginning of each experiment, that the packet loss rate on each link was less than 10%.

On these nodes, we run *exactly* the same code as in our simulator by wrapping it within appropriate user-level elements in Click. Furthermore, all experimental parameters are exactly the same as in the simulation (Table 1), with one exception: we use receiver buffer sizes of 2048 packets so that flows are not receiver-limited. For re-

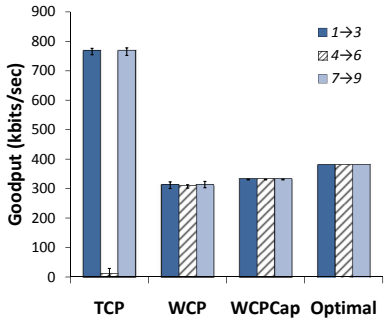


**Figure 17: WCP and WCPCap with no RTS/CTS, Stack**

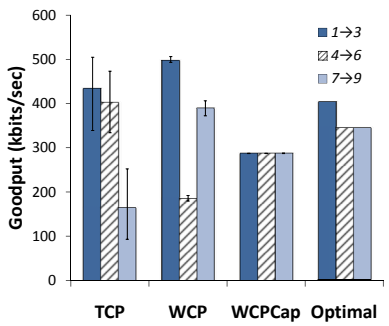


**Figure 18: WCP and WCPCap with no RTS/CTS, Half-Diamond**

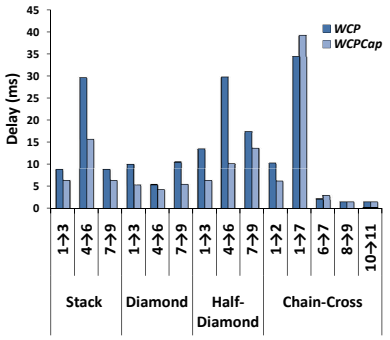


**Figure 19: Average end-to-end delay with WCP and WCPCap**

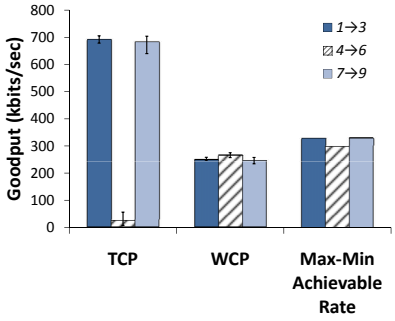


**Figure 21: Results from Stack experimental topology**

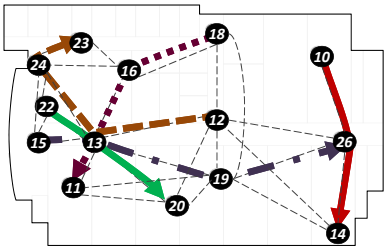


**Figure 22: Arbitrary experimental topology**

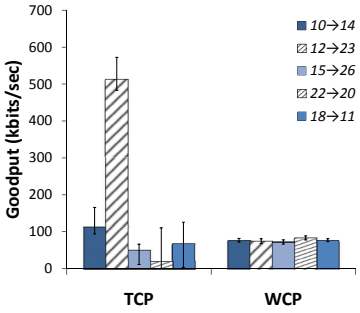


**Figure 23: Results from arbitrary topology**

peatability, all experiments were performed between midnight and 8am. All our experiments ran for 500 seconds and we show results averaged over five runs.

We re-created the Stack topology by carefully placing nine nodes across three floors, and by removing the antennas on some nodes. Figure 21 shows that the experimental results are similar to the simulation results (Figure 9). Furthermore, WCP achieves goodputs within 20% of an empirically determined maximum achievable rate. (We do not use our theory for determining optimal rates, because we cannot accurately estimate the amount of interference from external wireless networks.) We determine this by sending CBR flows at increasing rate till the goodput of flow $f_{4\to6}$ is less than 90% of the goodput of the other two flows.

Finally, to examine the performance of WCP in a real-world setting, we created an arbitrary topology of 14 nodes by placing them on one floor of our office building (Figure 22). To create a multi-hop topology, we covered antennas of nodes with aluminium foil. On this topology, we ran five flows as shown. Figure 23 shows the end-to-end goodput achieved by the flows. TCP starves $f_{15\to26}$, $f_{22\to20}$ or $f_{18\to11}$ during different runs. By contrast, WCP is able to consistently assign fair goodputs to all five flows in each run of the experiment!

## 6. CONCLUSIONS AND FUTURE WORK

Congestion control has vexed networking researchers for nearly three decades. Congestion control in wireless mesh networks is, if anything, harder than in wired networks. In this paper, we have taken significant steps towards understanding congestion control for mesh networks. Our main contributions include: the first near-optimal implementation of fair and efficient rate control for mesh networks; a plausibly implementable available capacity estimation technique that gives near-optimal max-min fair rates for the topologies we study; and, insights into the impact of various factors (e.g., RTS/CTS, whether rate control is implemented within the network or at the source) on performance.

Much work remains. First, we plan to understand the kind of fairness achieved by WCP. Second, we intend to investigate efficient implementations of WCPCap. Finally, we intend to explore how to account for the loss of capacity caused by interference, the impact of mobility on WCP and WCPCap's performance, how to support short-lived flows, and whether modifications to the 802.11 MAC layer can improve performance. We are also interested in the more general question of whether there exists a family of rate control schemes that can operate closer to the boundary of the achievable rate region than either WCP or WCPCap.

## 7. ACKNOWLEDGEMENTS

We would like to thank our shepherd Lili Qiu and the anonymous reviewers for their suggestions. Our thanks to Horia Vlad Balan for his help with real-world experiments.

## 8. REFERENCES

[1] MadWifi. http://madwifi.org/.
[2] MIT Roofnet. http://pdos.csail.mit.edu/roofnet/.
[3] Qualnet. http://www.scalable-networks.com/products/.
[4] F. Abrantes and M. Ricardo. A simulation study of xcp-b performance in wireless multi-hop networks. In *Proc. of Q2SWinet*, 2007.
[5] U. Akyol, M. Andrews, P. Gupta, J. Hobby, I. Saniee, and A. Stolyar. Joint scheduling and congestion control in mobile ad hoc networks. In *Proc. of IEEE INFOCOM*, 2008.

[6] P. Bahl, A. Adya, J. Padhye, and A. Wolman. Reconsidering Wireless Systems with Multiple Radios. *ACM SIGCOMM Computer Communications Review*, 2004.

[7] A. Bakre and B. Badrinath. I-TCP: indirect TCP for mobile hosts. In *Proc. of IEEE ICDCS*, 1995.

[8] H. Balakrishnan, S. Seshan, and R. H. Katz. Improving reliable transport and handoff performance in cellular wireless networks. *Wireless Networks*, 1995.

[9] G. Bianchi. Performance Analysis of the IEEE 802.11 Distributed Coordination Function. *IEEE Journal on Selected Areas in Communications*, 2000.

[10] K. Chandran, S. Raghunathan, S. Venkatesan, and R. Prakash. A feedback based scheme for improving tcp performance in ad-hoc wireless networks. In *Proc. of IEEE ICDCS*, 1998.

[11] H. Chang, V. Misra, and D. Rubenstein. A general model and analysis of physical layer capture in 802.11 networks. In *Proceedings of IEEE INFOCOM*, 2006.

[12] C. Cordeiro, S. Das, and D. Agrawal. Copas: dynamic contention-balancing to enhance the performance of tcp over multi-hop wireless networks. In *Proc. of IEEE ICCCN*, 2002.

[13] D. S. J. D. Couto, D. Aguayo, J. Bicket, and R. Morris. A high-throughput path metric for multi-hop wireless routing. In *Proc. of ACM MobiCom*, 2003.

[14] S. M. Das, D. Koutsonikolas, Y. C. Hu, and D. Peroulis. Characterizing multi-way interference in wireless mesh networks. In *Proceedings of ACM WinTECH Workshop*, 2006.

[15] N. Dukkipati, M. Kobayashi, R. Zhang-Shen, and N. McKeown. Processor Sharing Flows in the Internet. In *Proc. of IWQoS*, 2005.

[16] A. Eryilmaz and R. Srikant. Fair Resource Allocation in Wireless Networks using Queue-length based Scheduling and Congestion Control. In *Proc. of IEEE INFOCOM*, 2005.

[17] K. Fall and S. Floyd. Simulation-based comparisons of Tahoe, Reno and SACK TCP. *ACM SIGCOMM Comput. Commun. Rev.*, 1996.

[18] S. Floyd and V. Jacobson. Random Early Detection gateways for Congestion Avoidance. *IEEE/ACM Transactions on Networking*, 1993.

[19] Z. Fu, H. Luo, P. Zerfos, S. Lu, L. Zhang, and M. Gerla. The impact of multihop wireless channel on tcp performance. In *IEEE Transactions on Mobile Computing*, 2005.

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

[21] M. Garetto, T. Salonidis, and E. Knightly. Modeling Per-flow Throughput and Capturing Starvation in CSMA Multi-hop Wireless Networks. In *Proc. of IEEE INFOCOM*, 2006.

[22] M. Garetto, J. Shi, and E. Knightly. Modeling Media Access in Embedded Two-Flow Topologies of Multi-hop Wireless Networks. In *Proc. of ACM MobiHoc*, 2005.

[23] G. Holland and N. Vaidya. Analysis of TCP performance over mobile ad hoc networks. In *Proc. of ACM MobiCom*, 1999.

[24] B. Hull, K. Jamieson, and H. Balakrishnan. Mitigating congestion in wireless sensor networks. In *Proc. of ACM SenSys*, 2004.

[25] V. Jacobson. Congestion avoidance and control. In *Proc. of ACM SIGCOMM*, 1988.

[26] K. Jain, J. Padhye, V. N. Padmanabhan, and L. Qiu. Impact of interference on multi-hop wireless network performance. In *Proc. of ACM MobiCom*, 2003.

[27] A. Jindal and K. Psounis. Characterizing the Achievable Rate Region of Wireless Multi-hop Networks with 802.11 Scheduling. *USC Technical Report CENG-2007-12, submitted to IEEE/ACM Transactions on Networking*, 2007. `http://tinyurl.com/5heujd`.

[28] A. Jindal and K. Psounis. Achievable Rate Region and Optimality of Multi-hop Wireless 802.11-Scheduled Networks. In *Proc. of the Information Theory and Applications Workshop (ITA)*, 2008.

[29] D. Katabi, M. Handley, and C. Rohrs. Congestion control for high bandwidth-delay product networks. In *Proc. of ACM SIGCOMM*, 2002.

[30] D. Kim, C.-K. Toh, and Y. Choi. TCP-BuS: improving TCP performance in wireless ad hoc networks. *IEEE International Conference on Communications*, 2000.

[31] M. Kodialam and T. Nandagopal. Characterizing the capacity region in multi-radio multi-channel wireless mesh networks. In *Proc. of ACM MobiCom*, 2005.

[32] V. Kumar, M. M.V, S. Parthasarathy, and A. Srinivasan. Algorithmic Aspects of Capacity in Wireless Networks. In *Proc. of ACM SIGMETRICS*, 2005.

[33] S. Kunniyur and R. Srikant. Analysis and Design of an Adaptive Virtual Queue (AVQ) Algorithm for Active Queue Management. In *Proc. of ACM SIGCOMM*, 2001.

[34] Y. Li, L. Qiu, Y. Zhang, R. Mahajan, and E. Rozner. Predictable Performance Optimization for Wireless Networks. In *Proc. of ACM SIGCOMM*, 2008.

[35] X. Lin and N. B. Shroff. Joint Rate Control and Scheduling in Multihop Wireless Networks. In *Proc. of IEEE Conference on Decision and Control*, 2004.

[36] J. Liu and S. Singh. Atcp: Tcp for mobile ad hoc networks. *IEEE Journal on Selected Areas in Communications*, 2001.

[37] C. Lochert, B. Scheuermann, and M. Mauve. A survey on congestion control for mobile ad hoc networks: Research Articles. *Wirel. Commun. Mob. Comput.*, 2007.

[38] R. Morris, E. Kohler, J. Jannotti, and M. F. Kaashoek. The Click modular router. *SIGOPS Oper. Syst. Rev.*, 1999.

[39] M. Neely and E. Modiano. Capacity and Delay Tradeoffs for Ad-Hoc Mobile Networks. *IEEE Transactions on Information Theory*, 2005.

[40] G. Nychis, Sardesai, and S. Seshan. Analysis of XCP in a Wireless Environment. *Carnegie Mellon University*, 2006.

[41] J. Padhye, J. Kurose, D. Towsley, and R. Koodli. A model based TCP-friendly rate control protocol. In *Proc. of NOSSDAV*, 1999.

[42] J. Paek and R. Govindan. RCRT: rate-controlled reliable transport for wireless sensor networks. In *Proc. of ACM SenSys*, 2007.

[43] S. Rangwala, R. Gummadi, R. Govindan, and K. Psounis. Interference-aware fair rate control in wireless sensor networks. In *Proc. of ACM SIGCOMM*, 2006.

[44] G. Sharma, A. Ganesh, and P. Key. Performance Analysis of Contention Based Medium Access Control Protocols. In *Proc. of IEEE INFOCOM*, 2006.

[45] P. Sinha, T. Nandagopal, N. Venkitaraman, R. Sivakumar, and V. Bharghavan. WTCP: a reliable transport protocol for wireless wide-area networks. *Wireless Networks*, 2002.

[46] A. L. Stolyar. Maximizing queueing network utility subject to stability: greedy primal-dual algorithm. *Queueing Systems*, 2005.

[47] Y. Su and T. Gross. WXCP: Explicit Congestion Control for Wireless Multi-Hop Networks. In *Proc. of IWQoS*, 2005.

[48] K. Sundaresan, V. Anantharaman, H.-Y. Hsieh, and R. Sivakumar. ATP: A Reliable Transport Protocol for Ad Hoc Networks. *IEEE Transactions on Mobile Computing*, 2005.

[49] K. Tan, F. Jiang, Q. Zhang, and X. Shen. Congestion Control in Multihop Wireless Networks. *IEEE Transactions on Vehicular Technology*, 2006.

[50] L. Tassiulas and A. Ephremides. Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks. *IEEE Transactions on Automatic Control*, 1992.

[51] K. Xu, M. Gerla, L. Qi, and Y. Shu. Enhancing TCP fairness in ad hoc wireless networks using neighborhood RED. In *Proc. of ACM MobiCom*, 2003.

[52] X. Yu. Improving TCP performance over mobile ad hoc networks by exploiting cross-layer information awareness. In *Proc. of ACM MobiCom*, 2004.