RTS/*CTS*-Induced Congestion in Ad Hoc Wireless LANs

Saikat Ray, Jeffrey B. Carruthers and David Starobinski

***Abstract*— The *RTS*/*CTS* mechanism is widely used in wireless networks in order to avoid packet collisions and, thus, achieve high throughput. In ad hoc networks, however, the current implementation of the *RTS*/*CTS* mechanism may lead to inter-dependencies so that nodes become unable to transmit any packets during long periods of time. This effect manifests itself in the form of congestion where, after a certain point, the network throughput decreases with increasing load instead of maintaining its peak value. In this paper, we describe and analyze this problem in detail and provide a backward-compatible solution, called *RTS Validation*. Our simulations show that this solution leads to a 60% gain in the peak throughput in addition to stabilizing the throughput at high load.**

I. Introduction

The performance of a wireless network critically depends upon the medium access control (MAC) protocol used. Carrier Sense Multiple Access (CSMA) protocol is often chosen because of its simplicity and scalability. However, simple CSMA is susceptible to the hidden node problem [1], especially in so called *ad hoc* networks where a node may communicate directly with every other node in range or using intermediate nodes as relays otherwise [2, 3]. Hidden nodes cause costly packet collisions and thus significantly affect network performance. In order to combat the hidden node problem, a mechanism known as *RTS*/*CTS* handshake is often used. For example, the *RTS*/*CTS* mechanism is supported in the IEEE 802.11 family of standards. The *RTS*/*CTS* mechanism was initially proposed in [4] in a protocol called Multiple-Access with Collision Avoidance (MACA). In [5], the authors proposed a modified version of MACA, MACA for Wireless (MACAW), which includes a MAC level acknowledgment (*ACK*). IEEE 802.11 standard uses a variant of MACAW along with CSMA.

From a network point of view, one of the primary reasons for using the *RTS*/*CTS* mechanism is to avoid network congestion resulting from frequent packet collisions. Figure 1 depicts a conceptual "throughput versus load" curve for a network. In the presence of congestion, the throughput goes to zero as the load is increased beyond a certain value. A properly designed network, on the other hand, maintains the maximum throughput as the load goes to infinity.

The *RTS*/*CTS* mechanism generally works well in infrastructure-based networks, even though it may lead to unfairness in some situations [6]. However, in the general setting of ad hoc networks, the current way of implementing the *RTS*/*CTS* mechanism gives rise to situations where a large number of nodes are unable to transmit any packet. These situations can lead to network-level congestion. Consequently, the throughput of the network goes to zero as the load increases, i.e. it follows the "Congested" curve in Figure 1 instead of following the "Desired" curve. Therefore, the *RTS*/*CTS* mechanism fails to achieve its goal from a network point of view.

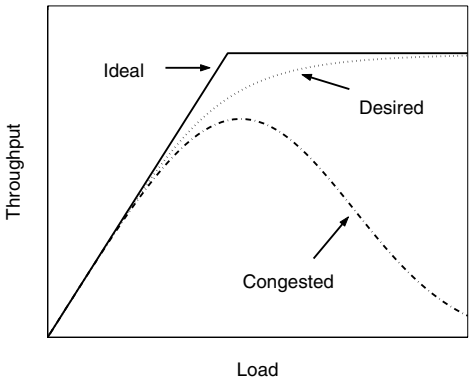


Fig. 1. Typical network throughput.

We note that *RTS*/*CTS*-induced congestion is different from the congestion that arises in the familiar TCP context. The latter occurs due to buffer overflow while the former is related to medium access control, i.e. *RTS*/*CTS*-induced congestion can take place even if an infinite buffer is used in every node.

In this paper, we examine the phenomenon of *RTS*/*CTS*-induced congestion in ad hoc wireless LANs. We first explain the cause of *RTS*/*CTS*-induced congestion, the *blocking problem*, which occurs due to the fact that any node that receives an *RTS* or a *CTS* packet defers its transmissions without using any further information. The blocking problem may force a large number of nodes not to transmit, sometimes creating situations akin to a deadlock, which we describe in Section II. By observing that it is possible for the nodes to use additional information to decide whether to defer or not, we present a backward compatible solution to the blocking problem in Section III, which is called *RTS Validation*. We compare our approach against the standard implementation of *RTS*/*CTS* mechanism in Section IV. Simulation results show that with the standard *RTS*/*CTS* implementation, the network behaves as a congested network and that the *RTS* Validation solves

The authors are with the Department of Electrical and Computer Engineering, Boston University.

This work was supported in part by the National Science Foundation under NSF CAREER grant ANI-0132802 and by a SPRInG award from Boston University.

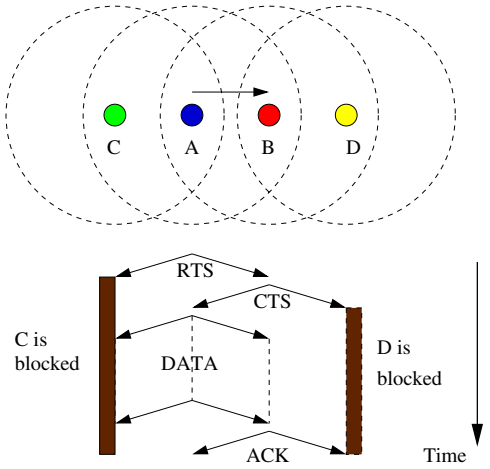


Fig. 2. IEEE 802.11 MAC. The lower half depicts the time-line. The dark bars below node C and D indicates their NAV.

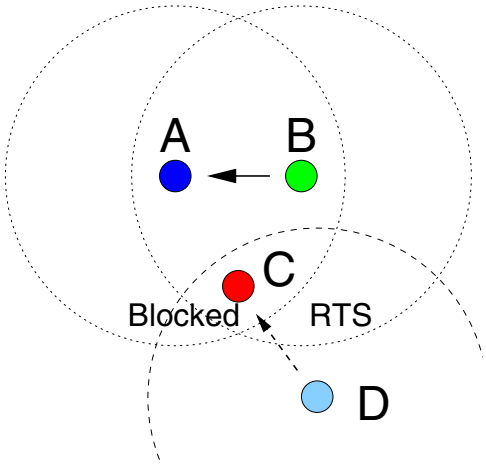


Fig. 3. Blocking Problem. Node C is blocked due to the communication between node A and node B. Therefore, node D does not get any response to the *RTS* packets it sends and enters backoff.

the congestion problem. Finally, we summarize the work in Section V.

II. *RTS/CTS*-Induced Congestion

A. RTS/CTS Mechanism in IEEE 802.11

In this section, as a typical example of the implementation of the *RTS/CTS* mechanism, we briefly describe the DCF mode of the IEEE 802.11 Wireless LAN protocol. The protocol is described in detail in [7].

In the DCF mode, a node may transmit a packet using one of the following two methods: the *basic access* method or the *RTS/CTS* method. In the basic access method, a node transmits a *DATA* packet if it senses the channel to be idle. The receiver, upon receiving an error-free packet, returns an *ACK*. If the transmitting node does not get an *ACK* back, it enters into back-off and retransmits after the back-off period.

The basic access method suffers from the well-known *hidden node* problem [1]. In order to address the issue, IEEE 802.11 supports the *RTS/CTS* access control mode. The *RTS/CTS* access mode is a combination of carrier sensing and a modified version of the *MACAW* protocol proposed in [5]. Figure 2 illustrates the scheme. When a node A wants to send a packet to node B, it initially sends a small packet called *Request-to-Send* (RTS). Upon correctly receiving the *RTS*, node B responds with another small packet called *Clear-to-Send* (*CTS*). After receiving the *CTS*, node A sends the *DATA* packet to node B. If node B receives the *DATA* packet correctly, it sends an *Acknowledgment* (*ACK*) back to node A. Any node that hears an *RTS* or a *CTS* is prohibited from transmitting any signal for a period that is encoded in the *duration* field of the received *RTS* or *CTS*. The duration fields in *RTS* and *CTS* are set such that nodes A and B will be able to complete their communication within the prohibited period. The deferral periods are managed by a data structure called the Network Allocation Vector (NAV). Finally, if a node does not get a response to an *RTS* or a *DATA* packet, it enters into an exponential backoff mode.

B. The Blocking Problem

We define a node to be *blocked* if it is prohibited from transmitting at a given instant. Since only one node is allowed to transmit at any time within the range of a receiver, many nodes in a wireless network may be blocked. Moreover, neighbors of a blocked node are unaware of the fact that this node is blocked. Therefore, a node may wish to initiate a communication with a node that is presently blocked. In that case, when the sender sends an *RTS* packet, the destination does not respond because it is blocked. The sender, however, interprets this to be a channel contention and enters backoff. We refer to this problem as the *blocking problem*.

Figure 3 describes the blocking problem. In this figure, node B transmits a packet to node A. Node C receives both *RTS* and *CTS* packets and therefore remains prohibited from transmitting. While the communication between node B and node A goes on, node D sends an *RTS* to node C. Since node C is blocked, it cannot respond with a *CTS*. Therefore, node D does not get any response and enters into backoff mode.

In [8], the author briefly describes this problem where it is termed as *hidden receiver* and *exposed receiver* problem. However, the blocked node need not be a hidden or an exposed node. For example, in Figure 3, node C receives both *RTS* and *CTS* and is therefore neither a hidden nor an exposed node. Therefore, we prefer to call this problem the *blocking problem*.

Due to exponential backoff of IEEE 802.11 MAC, the *RTS* sender (node D in the example above) quickly enters into a long inhibition period. This under-utilizes the network capacity. However, the current implementation of the *RTS/CTS* mechanism suffers from an even more severe problem, as described next.

C. The False Blocking Problem and its Propagation

In IEEE 802.11 MAC, any node that receives an *RTS* packet is required to inhibit transmission. This rule is designed to

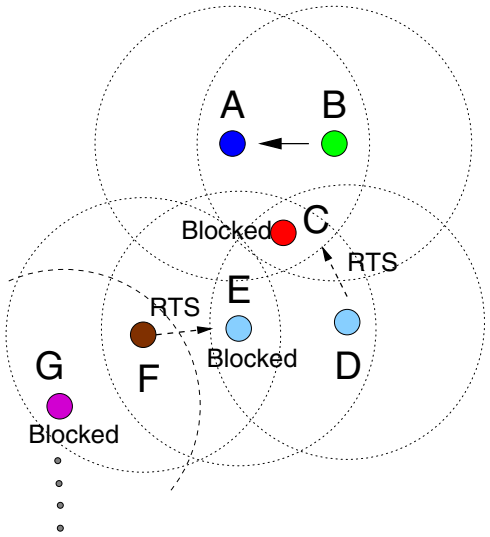


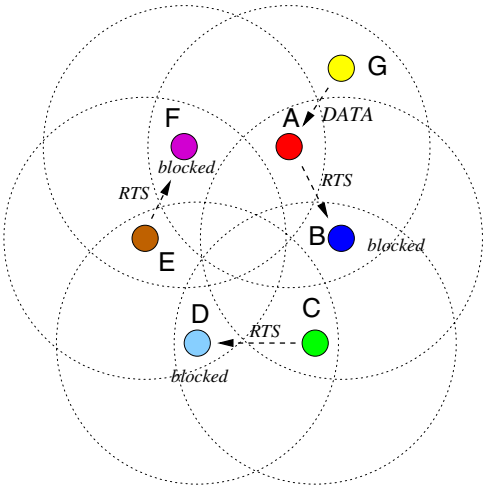
Fig. 4. False blocking problem. Node E is unnecessarily blocked because of node D's *RTS*. Therefore, node F does not get any response to its *RTS*, which in turn blocks node G and so on.

ensure that the *ACK* packets can be received by the sender without any collision. However, due to this rule, a nearby node may get falsely blocked, i.e. it may become prohibited from transmitting even if no other node is actually transmitting. Specifically, an *RTS* packet destined to a blocked node forces every other node that receives the *RTS* to inhibit even though the blocked destination does not respond and thus no *DATA* packet transmission takes place. We call this problem the *false-blocking problem*.

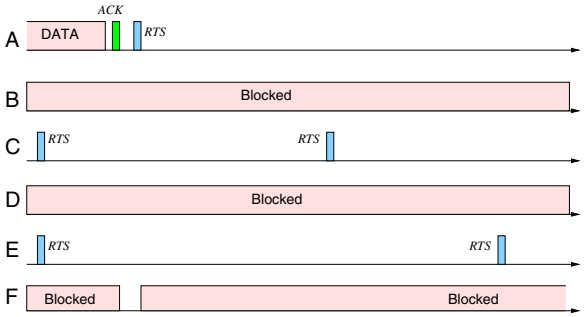
Figure 4 explains the problem. In this figure, node B is sending a packet to node A and therefore node C is blocked. While node C is blocked, node D sends *RTS* packets to node C, but node C does not respond. However, node E receives the *RTS* packets and prohibits itself from transmitting. Node E is therefore false blocked.

The simple blocking problem is localized to the neighbors of the blocked node and thus has a limited impact on the network performance. False blocking, however, may *propagate* through the network, i.e. one node may become false blocked due to a node that itself is false blocked. Therefore, false blocking may affect network performance severely.

Figure 4 shows a scenario of the propagation of false blocking. As in the previous example, node E is false blocked. Now, node F sends an *RTS* packet to node E and does not get any response. Therefore, node F goes into backoff. However, the *RTS* packet sent by node F blocks node G and so on. Note that a blocked node remains blocked for longer than a *DATA* packet transmission time (see Figure 2). Therefore, the likelihood that some other node wishes to communicate to a blocked node is non-negligible, especially at higher network load.



(a) Topology.



(b) Timing.

Fig. 5. Pseudo-Deadlock. Initially node A receives a packet from node G. This transmission creates the sequence of blocked and deferring nodes $\{F, E, D, C\}$. After G's transmission is over, node A transmits an *RTS* packet to node B. However, node C's *RTS* sent to node D blocks node B while node A receives packet from node G, therefore, node A does not get response to its *RTS*. Now, every node tries to transmit to a blocked node and a deadlock occurs.

D. Pseudo-Deadlock

The false blocking problem may not only propagate throughout a network, but it might also give rise to deadlock situations, at least for temporary periods. Once such a deadlock takes place, the throughput of the nodes involved in the deadlock goes down to zero. However, this deadlock is expected to be broken eventually as packets are dropped after a certain number of back-off attempts. Therefore, we refer to this situation as a *pseudo-deadlock*.

The basic cause of a pseudo-deadlock is that the propagation of false blocking takes place along a "circular" path. Figure 5(a) depicts such a situation. In this figure, node A initially receives a packet from node G. Node F is blocked during this time because node A is receiving. So, node E does not get any response to the *RTS* packets it sends to node F. Node E's *RTS* packets, however, force node D into false

blocking. Subsequently, node C does not get any response to the *RTS* packets sent to node D and, therefore, goes into backoff. Node B, however, receives node C's *RTS* packets and therefore gets blocked.

Now, node A, after receiving the packet from node G, wishes to communicate to node B. But, when node A sends an *RTS* to node B, node B is already blocked and so node A does not get any response. Node A's *RTS* blocks node F, though. So, when node E sends its next *RTS*, it again receives no reply. At this moment, in the cycle $\{A, B, C, D, E, F, A\}$, every second node $\{A, C, E\}$ is sending *RTS* to the next node $\{B, D, F\}$, which is already blocked and due to this *RTS*, the previous node gets blocked. As long as the blocking period for a node that receives an *RTS* is greater than the maximum time gap between two *RTS* packets during retries, the nodes cannot come out of this situation and therefore, this is a deadlock. Figure 5(b) shows the timings of each packet.

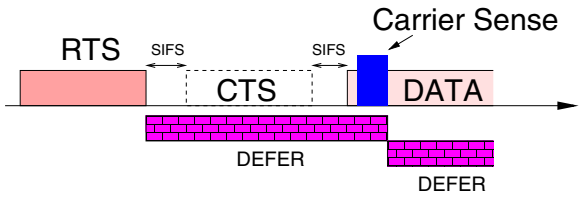
In the usual situation, one of the nodes will drop its packet after a certain number of retries and the deadlock will be broken. However, if each node needs to send several packets to the same destination (that may originate from a higher layer protocol such as TCP), then the deadlock may persist for a long period of time.

False blocking and consequent deadlocks significantly affect the network performance by drastically reducing the throughput. At the network level, this effect manifests itself as congestion where the throughput goes to zero as the load increases. We will present our simulation results in Section IV that corroborate the congestion phenomenon, but first we present a backward-compatible solution to the problem of false blocking in the next section.

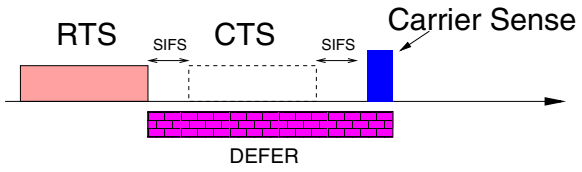
III. Solution to False Blocking: *RTS* Validation

False blocking is a consequence of the fact that every node that receives an *RTS* inhibits itself from transmitting. However, if a node is false blocked, then the corresponding *DATA* packet transmission does not take place while the node defers. Therefore, it follows that if a node assesses the channel to be idle during the expected *DATA* packet transmission period following an *RTS*, then the node must be false blocked.

Our proposed solution, called *RTS Validation*, is based on this observation. A node that uses *RTS* Validation, upon overhearing an *RTS* packet, defers until the corresponding *DATA* packet transmission is expected to begin and then assesses the state of the channel. If the channel is found idle, then it defers no longer, otherwise it continues deferral. Specifically, when a node receives an *RTS* that is not destined for itself, it defers for next *RTS_Defer_Time*. The *RTS_Defer_Time* is set as small as possible so that the *DATA* packet transmission is expected to begin at the end of this period, with allowances for various propagation delays. After this deferral period, the node assesses the channel for next *Clear-Channel Assessment Time* (CCA_Time) while continuing deferring (The CCA_Time is the time required to assess the state of the channel [7]). If the channel is assessed to be busy, the node defers for an additional period so that the total deferral time equals to *Requested_Defer_Time*, the duration of deferral requested by the *RTS*; otherwise it defers no longer. Figure 6 explains the proposed rule.



(a) The node senses busy channel following *RTS*_Defer_Time and therefore continues deferral.



(b) The node assesses idle channel following *RTS*_Defer_Time and therefore defers no longer.

Fig. 6. *RTS* Validation mechanism.

With *RTS* Validation, the nodes that receive an *RTS* destined to a blocked node ignore the *RTS* when the channel is assessed idle. Since *RTS_Defer_Time* and CCA_Time are generally much smaller than the *Requested_Defer_Time*, the likelihood of propagation of false blocking greatly reduces when *RTS* Validation is used. Note, however, that the *RTS* Validation mechanism may decide to defer even if the corresponding *DATA* packet transmission did not begin if it assesses the channel to be busy because of other transmissions.

RTS Validation is a backward-compatible approach in the sense that a node that uses *RTS* Validation (an "intelligent" node) may communicate with a node that does not use *RTS* Validation (standard node), since *RTS* Validation does not require any change in the packet format or the packet exchange protocol. However, in a network mixed with intelligent and standard nodes, the intelligent nodes may be able to transmit more packets since they defer for a much smaller time in the case of false blocking.

Note that there are other ways to combat the false blocking problem. For example, in the MACAW [5] protocol, another packet, called *DATA_Send* (DS), is sent when a node receives *CTS* in response to an *RTS*. Another approach is to use *Negative CTS* (NCTS) as suggested in [8]. These approaches would perform similarly to *RTS* Validation. The *RTS* Validation approach, however, has the important advantage of being backward compatible.

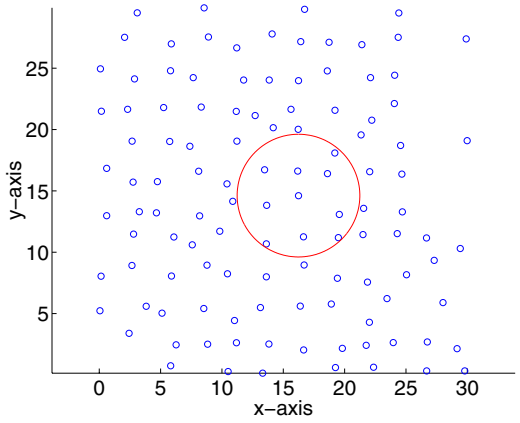


Fig. 7. The network used for the simulation. The circle represents the footprint of the node at the center of the circle. There are 10 nodes per footprint on average.

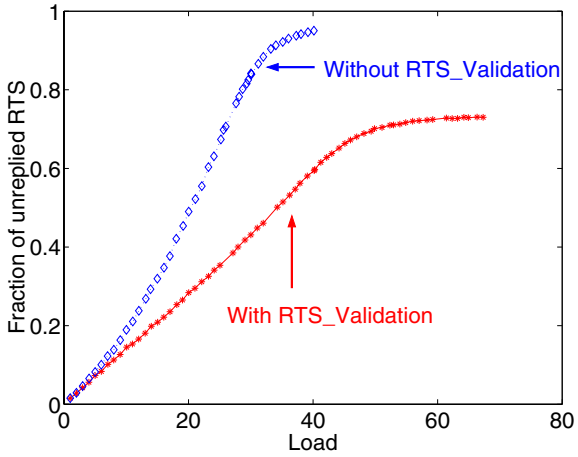


Fig. 8. Blocking: Comparison between the standard and the *RTS* Validation approach. Only successfully received *RTS* packets are counted in this figure.

IV. Simulation

A. Simulation Models

Our MATLAB based simulator [9] simulates a static two-dimensional network with wrap-around topology. Every node transmits omni-directionally with the same power, all transmissions experience the same path loss versus distance profile, and each node has the same antenna gain and receiver sensitivity. Around each node, a circular region exists, called the *footprint*, which defines the transmission range of the node. Nodes within transmission range of each other communicate without any error in the absence of packet collisions; nodes outside transmission range do not interfere. Figure 7 shows the network used and the range of a node. Propagation delay is assumed to be negligible. The 2300 byte size packets are transmitted at a 1-Mbps rate according to Poisson arrival processes independently generated at each node. The common average rate of packet generation per footprint defines the load of the network. DSSS parameters values of the IEEE 802.11 standard are used. The destination of each packet is one hop away, chosen at random. In addition, we assume that a successful *RTS*/*CTS* exchange guarantees a collision-free *DATA* packet transmission so that the effect of packet loss does not obscure the simulation results [10].

B. Results

In this section we compare the performance of the network when every node uses *RTS* Validation against the standard scenario where no node uses *RTS* Validation.

Figure 8 shows the fraction of times a node cannot send *CTS* due to blocking after receiving an *RTS* packet. We observe that the use of *RTS* Validation reduces this fraction considerably. For example, at a load of 30, the fraction is about 0.8 when *RTS* Validation is not used, but reduces to about 0.4 when *RTS* Validation is used. The significant blocking of standard *RTS*/*CTS* manifests itself as congestion when we consider network throughput, as shown next.

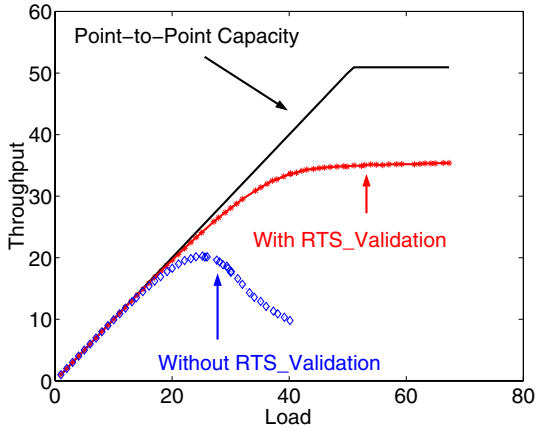


Fig. 9. Throughput: Comparison between the standard and the *RTS* Validation approach.

We define throughput to be the number of packets successfully transmitted per second per footprint on average. Figure 9 shows the throughput of the network as the load of the network increases. Clearly, the throughput of the network with standard *RTS*/*CTS* mechanism goes to zero as the load is increased, which implies that the network behaves like a congested network. When we introduce *RTS* Validation, we observe that the network performance is improved in two aspects. First, it stabilizes the network throughput with the load, i.e. the throughput does not fall after it achieves the peak. Secondly, it increases the peak throughput from approximately 22 packets per second to 35 packets per second per footprint. The point-to-point capacity in this figure, shown as a reference, refers to the number of packets a node can send back-to-back when no other node contends for the channel.

Another important aspect of network performance, the packet delivery time (or simply delay), is also improved. Figure 10 shows the average packet delay as the load increases. We observe that the difference in the two approaches is not

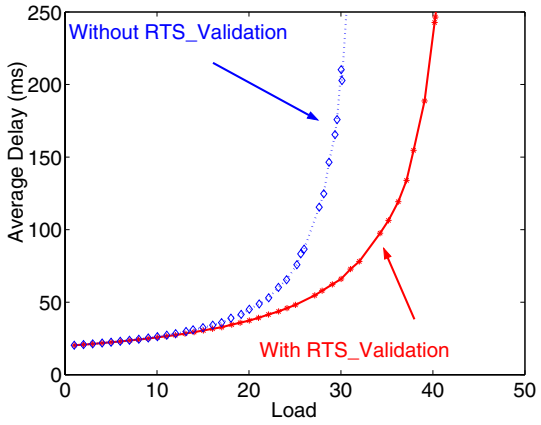


Fig. 10. Delay: Comparison between the standard and the *RTS* Validation approach.

significant up to a load of about 15. After that, however, the network saturates quickly if *RTS* Validation is not used. Specifically, without *RTS* Validation, the network saturates at a load of about 30 packets per second, while with *RTS* Validation, it takes a load of about 40 packets per second to saturate the network. Also, at any load, use of *RTS* Validation yields a lower average delay.

V. Summary

The *RTS*/*CTS* mechanism is widely used in ad hoc networks to avoid collisions caused by hidden nodes. In the current implementation, any node that receives an *RTS* or a *CTS* packet inhibits itself from transmitting without using any further information. In this paper, we have shown that this approach of implementing the *RTS*/*CTS* mechanism leads to false blocking, where a node remains prohibited from transmitting even if no nearby node transmits. Moreover, false blocking may propagate through the network; as a result, a large number of nodes may get false blocked. We have also shown that propagation of false blocking can lead to pseudo-deadlocks. Due to false blocking, the throughput of the network goes to zero as the load of the network is increased beyond a certain value. Therefore, the *RTS*/*CTS* mechanism may congest a network instead of stabilizing it.

We have proposed a simple solution to the false-blocking problem, called *RTS* Validation. A node that uses *RTS* Validation defers for an entire packet transmission period if *DATA* packet transfer begins, but defers only for a short time if no transmission takes place when *DATA* packet transmission is expected. By means of simulation, we have shown that the use of *RTS* Validation improves the network performance in three aspects: it eliminates congestion by stabilizing the network throughput at high load, it increases the peak throughput by 60%, and it significantly reduces the average delay. *RTS* Validation is a backward compatible solution and thus can be implemented incrementally.

References

[1] L. Kleinrock and F. A. Tobagi, "Packet switching in radio channels: Part 2 - the hidden node problem in carrier sense multiple access modes and the busy tone solution," *IEEE Transactions on Communications*, vol. COM-23, no. 12, pp. 1417–1433, 1975.

[2] C K Toh, *Ad Hoc Mobile Wireless Networks: Protocols and Systems*, Prentice Hall, December 2001.

[3] Zygmunt J. Haas, Jing Deng, Panagiotis Papadimitratos, and S Sajama, "Wireless ad hoc networks," in *Wiley Encyclopedia of Telecommunications*, John G. Proakis, Ed. Wiley, December 2002.

[4] Phil Karn, "MACA - a new channel access method for packet radio," in *ARRL/CRRL Amature Radio 9th Computer Networking Conference*, September 22 1990, pp. 134–140.

[5] Vaduvur Bharghavan, Alan Demers, Scott Shenker, and Lixia Zhang, "MACAW: A media access protocol for wireless LANs," in *Proceedings of ACM SIGCOMM '94*. 1994, pp. 212–225, ACM.

[6] V. Kanodia, C. Li, A. Sabharwal, B. Sadeghi, and E. Knightly, "Ordered packet scheduling in wireless ad hoc networks: Mechanism and performance analysis," in *Proceedings of MOBIHOC'02*, EPFL Lausanne, Switzerland, 2002, ACM.

[7] "ANSI/IEEE Std 802.11-1999 Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications," 1999.

[8] Vaduvur Bharghavan, "Performance evaluation of algorithms for wireless medium access," in *IEEE Performance and Dependability Symposium '98*, Raleigh, NC., 1998, IEEE.

[9] "SimEleven: An IEEE 802.11 MATLAB-based simulator.," Available at: `http://netlab1.bu.edu/~saikat`.

[10] Saikat Ray, Jeffrey B. Carruthers, and David Starobinski, "The masked node problem in ad hoc wireless LANs," Preprint.