A Fuzzy Logic Engine to Assist TCP Error Detection in Wireless Mobile Ad Hoc Networks

Ruy de Oliveira, Torsten Braun

Abstract—**In this paper, we introduce and evaluate an elementary fuzzy logic engine designed to assist TCP error detection mechanism in wireless mobile ad hoc networks. Our fuzzy logic engine aims at distinguishing packet loss due to congestion from packet loss by wireless channel induced errors, which is of great importance for TCP performance in such environments. The key idea behind fuzzy logic is its robustness in dealing with data encompassing uncertainties and non-linearities. We evaluate in this paper two end-to-end metrics, namely packet delay and its variance, which are inherently uncertain and non-linear but efficient in providing useful information for the desired discrimination. By using this approach, a TCP sender does not need to rely on any specific cooperation from the intermediate nodes, which is quite appealing in terms of deployment. The preliminary evaluation results show that this approach can provide accurate results, but improvements in terms of responsiveness might be needed.**

Keywords— **Ad hoc networks, Round Trip Time, Congestion Control, Fuzzy logic, End nodes.**

I. Introduction

WIRELESS mobile ad hoc networks have become more and more investigated in the research community over the past few years. These networks are challenging since they do not rely on any fixed infrastructure to establish communication. In ad hoc networks, nodes communicate exclusively through wireless connections and can move freely. Every node is required to forward data for the others, thereby acting as both host and nodes. Since there is no centralized point of coordination, these networks are labeled self-organizing networks, where topology changes may occur quite dynamically and unpredictably.

Ad hoc networks pose some tough challenges to the Transmission Control Protocol (TCP) [1] because it was not designed to work in such highly dynamic and unpredictable environments [2]. Rather, TCP was designed to work in wired networks where packet loss can safely be associated to network congestion. Hence, regular TCP relies on the assumption that any packet loss is a result of congestion inside the network, and appropriately reacts to that by slowing down its transmission rate.

This assumption is not valid for ad hoc networks though, where packet loss may also be related to both the inherently lossy wireless medium and link interruptions by mobility. The typically high bit error rate (BER) of wireless channels induce packet losses that are wrongly detected by TCP as congestion . In such cases, TCP sender improperly reduces its sending rate by halving its transmission rate. Concerning the link interruptions by mobility, they may lead TCP to long interval of idle state if sequential timeouts take place, which depends naturally on the length of the interruption and current round trip time [1], [2]. In both cases, the end-to-end throughput may be severely impaired. Thus, the TCP error detection mechanism really has to be adjusted to this challenging framework.

In general, proposed solutions are either end-to-end [3], [4], [5] or network oriented [6], [7], [8]. In the former, only the end nodes need to be changed, which is quite encouraging in terms of deployment. In the latter, the intermediated nodes are required to cooperate with the end nodes by providing them with explicit signaling messages about the internal state of the network. Our approach is end-to-end oriented. The idea here is to monitor the regular TCP flow and record useful data to infer the current state of the network when lost packets are perceived. In this way, not only deployment is facilitated but also security concerns are avoided since only the end nodes have full access to the packet header.

A robust TCP sender needs to have dedicated actions for each of the constraints found in ad hoc networks, namely congestion, channel error, and link interruption. Considering end-to-end approaches, link interruptions will always be detected by the TCP timeout timer. The exact protocol response will depend on the current internal state of the network though. This means that the sender needs to be permanently aware of the ongoing congestion and channel error conditions inside the network. With these issues in mind, we propose here a basic fuzzy logic engine for distinguishing between bit error and congestion induced losses using Round Trip Time (RTT) values as input variables. We do not address in this paper packet losses associated to link interruptions. The rationale for using RTT as indicator of the network internal state is that TCP already relies on such a parameter for computing its fundamental timeout interval. Besides, as we will show later, RTT measurements suffer distinct impacts under congestion and bit error constraints.

Fuzzy logic [9], [10] is a powerful tool for decision making processes involving information characterized by imprecision and uncertainties. It was first introduced by L. Zadeh in the 1960's as a means to model the uncertainty of natural language. Fuzzy logic the-

The work presented in this paper was supported (in part) by the National Competence Center in Research on Mobile Information and Communication Systems (NCCR-MICS), a center supported by the Swiss National Science Foundation under grant number 5005-67322.

Both authors are with the Institute of Computer Science and Applied Mathematics, University of Bern, Bern, Switzerland. E-mails: {oliveira|braun@iam.unibe.ch}

ory combines human expertise and uncertainties into an efficient set of decision making rules. By using fuzzy logic, the continuous and imprecise behavior of the information can be handled without the necessity of arbitrary rigid boundaries. A great advantage of this theory is that it is by far less computational intensive than other similar intelligent theories. Therefore, fuzzy logic is quite suitable for evaluating RTT measurements where imprecision and uncertainties are effectively present and the processing requirements (at the end nodes) must be as low as possible.

II. RTT Patterns in Multihop Wireless Networks

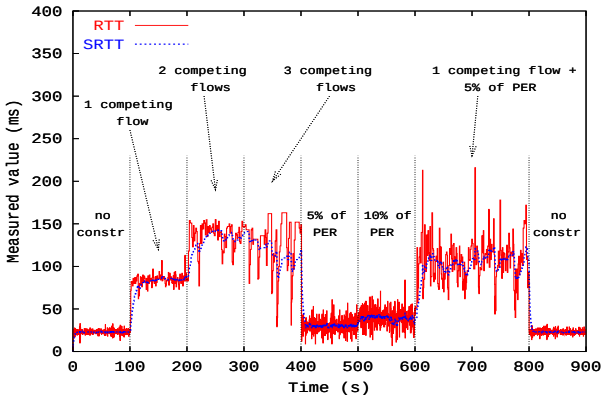
In ad hoc networks, RTT values reflect the impact of the following distinct factors: Congestion, wireless channel errors, changes in the number of hops crossed by the connection between sender and receiver, packet size, and lower layer protocol mechanisms such as MAC layer retransmission strategy [2], [11].

We assume in this work that the packet size of the connection and the lower layer protocols are fixed. Both assumptions should, however, not be too restrictive for future ad hoc networks. Small packet sizes have been claimed as a proper way for avoiding packet collisions in such environments [11], and so their use would not be costly. Concerning the lower layer protocols, IEEE 802.11 MAC protocol [12] is already standardized and both AODV [13] and DSR [14] routing protocols are the most prominent frameworks to be standardized in the near future. So, it is reasonable to consider that these two routing protocols will be effectively present in future ad hoc networks. This is important because both AODV and DSR ensure symmetric links, which is also a requirement of our initial approach.

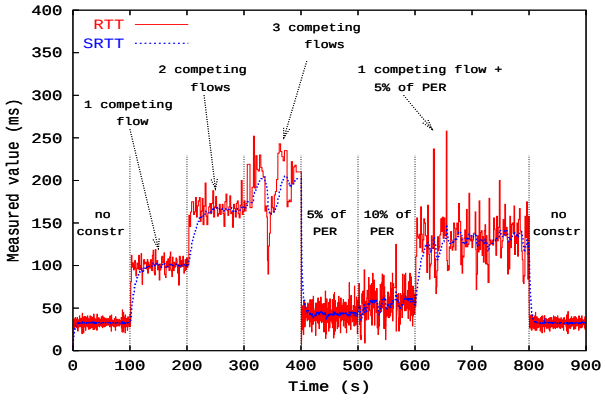
Therefore, taking the two assumptions above into consideration (fixed packet size and conventional lower layer protocols), RTT values might be really useful for distinguishing the effect of the other three factors, namely congestion, bit error and link interruptions.

In [4] we have shown that both the growth in the number of hops between sender and receiver and congestion conditions may impose similar behaviors to RTT measurements. Thus, sender and receiver should exchange information to detect the exact number of hops in place. We do not address this issue in this work, although it can be easily implemented using the Time To Live field in the IP header [1] at the end nodes. Actually, we focus here on the discrimination of packet losses by congestion from packet losses by channel error, having a fixed number of hops end-to-end.

Fig. 1 shows the result of two simulation runs, using ns-2 simulator. AODV and IEEE 802.11 are the routing and MAC layer protocols in place, respectively. The packet size is set to 1000 bytes, and both the main flow and the background flow are generated by FTP applications over TCP. A uniform distribution function is used as wireless channel error pattern for generating the different levels of BERs, and the wireless bandwidth is 2 Mbps. We use hereafter Packet Error Rate (PER) instead of BER to represent the channel error because PER is the metric in fact simulated.



(a) 2-hop scenario



(b) 3-hop scenario

Fig. 1. RTT characteristic under congestion and wireless losses

Fig. 1a depicts RTT and SRTT (Smoothed or average RTT) measurements on a typical 2-hop ad hoc network under varying levels of congestion and distinct packet error rates, starting at 100 seconds. Between 100 and 400 seconds there are three stages of congestion, i.e., 1-3 competing flows. Between 400 and 600 seconds, there is no congestion, but only two levels of packet error rate, i.e., 5 and 10%. At last, between 600 and 800 seconds there is only one competing flow and a packet error rate of 5%. In the last 100 seconds the channel is free of any constraint again. Fig. 1b illustrates the same conditions but for a 3-hop scenario.

Figs. 1a and 1b show clearly that under operative congestion conditions (having at least a minimum flow of ACKs) and without mobility, RTT mean values (roughly given by SRTT) should suffice to indicate congestion inside the network. The channel error constraint increases RTT as well, but at a far lower degree. The only possibility of mistake, although rare,

would be a very high level of packet error rate being misdetected as congestion, since high PERs could increase the RTTs considerably. This possibility is quite low, however, because at such a high level of PERs practically no packet gets through. Besides, typical wireless environments do not have so lossy channels.

To make the distinction of congestion from wireless induced losses even more robust, RTT variance can be used to address the remote possibility mentioned above, as indicated in table I. The values in this table are computed over each interval using equations (2) and (3) presented in section III. Table I shows how much the RTT variance increases under channel error conditions. It is worth to compare the case in which there is just one competing flow (100-200) with the instant of the highest PER (500-600). One can notice here that for both scenarios the channel error constraint induces higher RTT variance, namely 215 and 537 against 97 and 135 for both scenarios, respectively. For lower congestion level, which could be achieved with other forms of competing traffic, the discrimination would be even better since the variance under congestion would be smaller.

TABLE I
RTT Mean and Variance

	2-hop		3-hop	
Interval	mean	variance	mean	variance
0-100 (1 flow)	22	7	32	25
100-200 (2 flows)	82	97	99	135
200-300 (3 flows)	136	304	162	246
300-400 (4 flows)	128	820	189	980
400-500 (5% PER)	32	176	47	646
500-600 (10% PER)	47	215	60	537
600-800 (2 flows + 5% PER)	115	840	136	863

In the last part of the simulation under constraint (600-800) depicted in Figs. 1a and 1b, there is only one competing flow and a packet error rate of 5%. It can be seen that both the RTT mean (roughly given by SRTT) and variance grow (higher magnitude of the oscillations). Table I supports this observation as well. The key point here is to note that the congestion detection has priority over the channel error detection, as TCP sender has to slow down anyway under congestion regardless of simultaneous channel error losses. In other words, if congestion is perceived, then the detection algorithm can ignore the channel error detection.

III. A Fuzzy engine for distinguishing between losses by congestion and channel error

The behavior of the RTT measurements presented in the last section suggests that an intelligent mechanism may be able to distinguish congestion from channel error induced losses inside the network. In particular, such a mechanism should be simple and flexible in terms of easy adaptation to this highly dynamic scenario, and also computational inexpensive. With these aspects in mind, we have designed a basic fuzzy logic engine for handling both the RTT mean and variance values toward an effective packet loss discrimination.

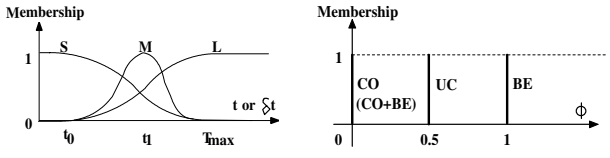
Fuzzy logic is a superset of conventional (Boolean) logic that has been extended to handle the concept of partial truth. Unlike classical sets of Boolean logic, in which there are only the elements 0 and 1 (false and true), fuzzy sets can have values in between. In Fuzzy logic, input values (i.e., discrete values) are mapped into membership functions that have smooth transition from 0 to 1. As an example, let us say that a given input is assigned the value 0.5 in the fuzzy space. This means that the degree of truth for this input value, determined by the fuzzy process, is 50%. In other others, fuzzy logic systems perform the fuzzyfication of input data (called crisp value) into membership functions in the fuzzy space.

Let U be the universe of discourse which the input data belong to, then an element x in U is mapped by the membership function $\mu X(x)$ into the range [0,1] within the fuzzy space, as shown in equation (1). This equation shows that membership functions define the degree of membership assigned to the input data, where 1 denotes full membership and 0 denotes no membership at all. Thus, fuzzy sets are indeed an extension of the classical sets in which only 0 (false) and 1 (true) degrees of membership exist.

$$\mu X : U \rightarrow [0, 1] \tag{1}$$

By avoiding sharp transition from false to true, fuzzy logic provides better reasoning on data that have overlapping values. This is done by carefully adjusting the shape of the membership functions and setting the involved fuzzy rules properly, as addressed below.

A typical fuzzy engine performs the following procedure: First it maps the discrete input values into the predefined membership functions (fuzzification), then it applies predefined fuzzy rules to the mapped inputs (inference process), and lastly the engine combines the result of each individual rule into a discrete output value (defuzzification). The fuzzy rules are mostly defined on the basis of the observed features of the input data. Human expertise may be taken into account as well. The key point of any fuzzy logic engine is to define really representative membership functions.



(a) Input (b) Output

Fig. 2. Fuzzy engine membership functions

In this paper, RTT mean and variance were computed through equations (2) and (3), respectively.

Where n refers to the number of samples used in each computation, and t_i is the value of each sampled RTT. We have classified the input data observed in Fig. 1 into three fuzzy sets, where each fuzzy set is represented by a distinct membership function. Specifically, the input data have been split into Small (S), Medium (M), and Large (L) values. Fig. 2a depicts the input membership functions which are identical for both RTT mean (t) and variance (δt). We have chosen Gaussian membership functions simply because this is the most widely used in the literature. Nevertheless, different membership functions might surely be tried. Besides, more precision in the results might be achieved by splitting the input data into a larger number of fuzzy sets.

$$t = \frac{1}{n}\sum_{i=1}^{n} t_i \tag{2}$$

$$\delta_t = \frac{1}{n}\sum_{i=1}^{n}(t_i - t)^2 \tag{3}$$

Accordingly, the fuzzy rules were established as shown in table II, where each rule is obtained from the generic equation (4) below. The number of rules depend on both the amount of input variables and the amount of fuzzy sets associated with each input variables. In our engine we have nine rules resulting from the combination of two inputs (t and δt) each having three fuzzy sets. Equation (4) shows how the n_{th} rule is obtained for an engine containing p input variables. Where F_k^n and G^n are fuzzy sets associated to the input and output fuzzy variables x_k and y, respectively, being k=1,...,p. For instance, in table II we have: if (variance is Small) and (mean is Large) then (output is Channel). Alternatively, other example could be: if (temperature is High) and (humidity is High) then (room is hot).

$$\begin{aligned} R^n \quad : \quad & \textbf{if } (x_1 \; is \; F_1^n) \textbf{ and } (x_2 \; is \; F_2^n) \textbf{ and} \\ & \ldots (x_p \; is \; F_p^n) \\ & \textbf{then } (y \; is \; G^n) \end{aligned} \tag{4}$$

The fuzzy rule outputs (ϕ) were assigned to the output memberships shown in Fig. 2b and table II. As already stated, the process by which the value of each fuzzy rule output is combined into the output memberships is called defuzzification. There are a number of ways for doing so, and we have chosen the most widely used method called gravity-of-mass [10].

TABLE II
FUZZY RULES OUTPUT (ϕ)

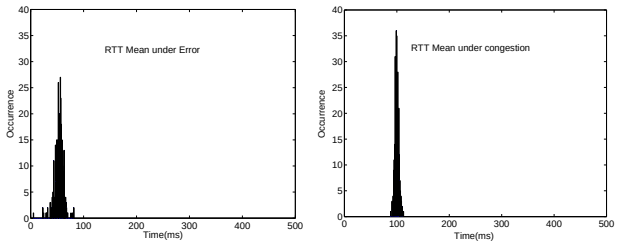
var \ mean	Small	Medium	Large
Small	Channel	Congestion	Congestion
Medium	Channel	Uncertain	Congestion
Large	Channel	Channel	Congestion

The fuzzy outputs (ϕ) were split into Channel error (BE), Congestion (CO), and Uncertain (UN), as depicted in table II. This table shows that Large value of *RTT mean* indicates that the connection is facing congestion regardless of the *RTT variance*. Similarly, Small value of *RTT mean* points out that any packet loss in such a condition has to be assigned to channel error regardless of the variance value. Medium value for both *RTT mean and variance* is a non conclusive output since this might be caused by either congestion or channel error conditions, as already explained. Under this output, the fuzzy engine needs to go on with its evaluation until obtain an effective output.

In our evaluation, we have set our engine based on the values found in the simulation shown in Fig. 1. It is worth to note that, even though the scenario simulated is just a particular one, the results are general. What is important in these simulations is the range in which the measured values have to be related to congestion or channel errors. This will not change in presence of more competing flows or different levels of packet error rate. On the other side, the fuzzy settings will change in accordance with the number of hops in place. More elaborate fuzzy engines could allow these settings to be done adaptively, but we do not address this issue in this work.

IV. Fuzzy Engine Performance

The evaluation of the proposed engine has been conducted by providing it with RTT mean and variance values obtained by simulations. The engine output is then compared to the actual reason of the detected packet loss. The simulation setup is the same discussed in the previous section.



(a) Channel error (10%) (b) Congestion

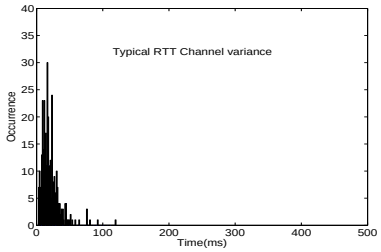
Fig. 3. RTT mean distribution

The simulation results, after applying equations (2) and (3), are depicted as histograms in Figs. 3 and 4. It is important to note that only the most critical conditions have been taken into consideration. As stated in section II, it might happen that low congestion condition is misdetected as lossy channel. Because of that, we took the case in which the main connection is facing only one competing flow and the one in which only 10% of PER constraints the medium. Additionally, longer runs of 500 seconds were simulated for each condition. These conditions are used as boundaries

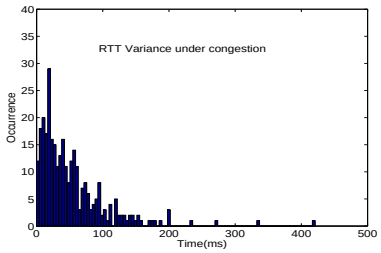
for setting the fuzzy engine parameters.

Figs. 3a and 3b depict the RTT mean under the critical conditions mentioned above. One can see here that even though the level of PER is relatively high, it is still possible to distinguish both conditions by simply comparing their RTT mean values. Fig. 3a shows that the experienced RTT mean values for the high PER are concentrated around 50 ms. So, we set the parameters to, t1, and tmax for RTT mean in Fig. 2a to 40, 50 and 60, respectively.

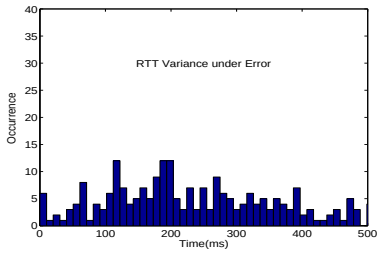
In Fig. 4, we have the variance distribution for a channel without any constraint and for a channel facing either congestion or channel error. Comparing Fig. 4b with 4c, one can see that channel error constraints impose indeed higher spread to the RTT variance. Nonetheless, there is an overlapping area roughly in the range (50,200), which has to be taken into consideration for setting the fuzzy membership functions associated with the RTT variance.



(a) No constraints



(b) One competing flow



(c) 10% of PER

Fig. 4. RTT variance distribution in a 3-hop multihop network

In reality, the range of this overlapping area would not be too wide if we had a more precise comparison by simulating lower level of congestion. The smaller the congestion level, the closer the histogram of Fig. 4c from the one in Fig. 4a. We set the parameters to, t1, and tmax for RTT variance in Fig. 2a to 50, 100 and 150, respectively. The universe of discourse of the input variables is set to 500 ms, which means that no RTT greater than 500 ms is found in our evaluations.

The ranges assigned to the output parameters CO, UC and BE (Fig. 2b) are set to [0,0.3), [0.3,0.7], and (0.7,1], respectively. These values are strategically established to reflect the neighborhood of 0.0, 0.5, and 1.0 in Fig. 2b.

Fig. 5 shows the fuzzy engine performance concerning the number of correct detections over the simulated scenarios. For each scenario we ran three different RTT sampling rates, namely 20, 40 and 60 RTTs per sample. The purpose here was to determine not only the engine accuracy itself but also the tradeoff between accuracy and detection delay. The results show that for any of the three RTT sampling rate, congestion is detected over 98% in all cases, while bit error constraints detection decreases considerably for lower number of RTTs. The results could certainly be fine tuned by using either more elaborate membership functions or different settings. Besides, more accuracy from the input values could be achieved by including more fuzzy sets (membership functions) into the fuzzy engine. These are desired goals for future work.

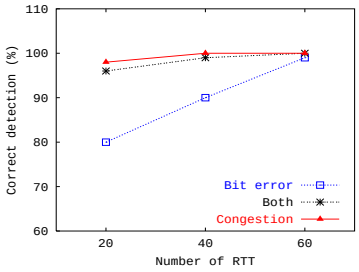


Fig. 5. Fuzzy engine performance

The results depicted in Fig. 5 make it clear that the fuzzy engine may provide accurate result as long as a reasonable number of RTTs is taken in each sampling for computing the mean and variance from equations (2) and (3). This may render the fuzzy engine a bit slow for detecting abrupt transitions in the network internal state since the engine has to await a certain number of incoming acks to trigger its detection. On the other hand, this approach will be quite efficient in making inference over systems under reasonable stability in which RTT mean and variance do not change neither too often nor too abruptly. Additionally, special mechanisms such as the "history discount" proposed in [15], in which the old values of the measurements are discarded when necessary, may speed up our engine, but we have not evaluated that in this work.

Acknowledgments

The authors would like to thank the anonymous reviewers for their valuable comments and suggestions

that helped us to improve considerably the quality of this paper.

V. Conclusions

We have designed a simple fuzzy logic engine to infer the network internal state on the basis of RTT measurements. The approach is end-to-end oriented since no explicit intermediate node cooperation is needed and inexpensive in terms of processing demand.

Our main conclusion is that the designed fuzzy engine may really provide useful input to TCP error detection mechanism. A TCP sender relying on such inputs will certainly outperform standard TCP in lossy environments, where the enhanced TCP will simply retransmit instead of slowing down as standard TCP does. Under congestion, there is nothing that TCP can do to improve its performance since the communication channel is impaired. Thus, as far as TCP friendly behavior is concerned, the enhanced TCP should slow down as the standard TCP does. And finally, when facing link interruptions, the sender will timeout and should start probing the network for connectivity in order to avoid long idle periods.

The primary aspect to be enhanced in this approach is its reaction time during transition of states such as congestion to channel error and vice versa. Therefore, at the moment our approach demands reasonable network stability to provide optimal performance. Additionally, more elaborate engines may be designed toward optimization. In particular, adaptive models for setting the fuzzy engine parameters automatically are of interest. These are open issues to be addressed in future work.

References

[1] W. Stevens, *TCP/IP Illustrated* Vol. 1. Addison Wesley, 1994.

[2] R. Oliveira and T. Braun, *TCP in Wireless Mobile Ad Hoc Networks*, Technical Report IAM-02-003. Inst. of Computer Science, University of Berne, July 2002.

[3] Z. Fu, B. Greenstein, X. Meng, S. Lu. *Design and Implementation of a TCP-Friendly Transport Protocol for Ad Hoc Wireless Networks*, ICNP'02, November 2002.

[4] R. Oliveira and T. Braun, M. Heissenbuettel, *An Edge-based Approach for Improving TCP in Wireless Mobile Ad Hoc Networks*, ASTC/DASD 2003, pp. 172-177, March/April 2003.

[5] J. Liu, I. Matta and M. Crovella. *End-to-End Inference of Loss Nature in a Hybrid Wired/Wireless Environment*, WiOpt'03, March 2003.

[6] K. Chandran, S. Raghunathan, S. Venkatesan, R. Prakash. *A Feedback Based Scheme For Improving TCP Performance In Ad-Hoc Wireless Networks* In Proceedings of International Conference on Distributed Computing Systems-ICDCS'98. pp. 472-479, 1997.

[7] J. Liu, S. Singh. *ATCP: TCP for Mobile Ad Hoc Networks* IEEE Journal on selected areas in communications, pp. 1300-1315, July 2001.

[8] G. Holland and N. Vaidya. *Analysis of TCP performance over mobile ad hoc networks*, Mobicom'99, August 1999.

[9] L. A. Zadeh, *Fuzzy logic = computing with words*, IEEE Transactions on Fuzzy Systems, Vol. 4, No 2, pp. 104-111, 1996.

[10] L. Cheng and I. Marsic, *Fuzzy Reasoning for Wireless Awareness*, International Journal of Wireless Information Networks, Vol. 8, Issue 1, Jan. 2001, pp. 15-26.

[11] S. Xu and T. Saadawi. *Does the IEEE 802.11 MAC protocol work well in multihop wireless ad hoc networks?*, IEEE Communications Magazine, Vol 39, No. 6, pp. 30 -137, June 2001.

[12] The Institute of Electrical and Electronics Engineers, *inc. IEEE std 802.11 - Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications*, The Institute of Electrical and Electronics Engineers, inc.,1999 edition.

[13] C. Perkins and E. Belding-Royer, S. Das, *Ad hoc On-Demand Distance Vector routing*, IETF RCF 3561, July 2003.

[14] D. Johnson and D. Maltz. *Dynamic source routing in ad hoc wireless networks*, In T. Imielinski and H. Korth, editors, Mobile computing, Kluwer Academic. 1996.

[15] M. Handley, S. Floyd, J. Padhye, J. Widmer, *TCP Friendly Rate Control (TFRC): Protocol Specification*, IETF RFC 3448, January 2003.