# A Delay-based Approach Using Fuzzy Logic to Improve TCP Error Detection in Ad Hoc Networks

Ruy de Oliveira, Torsten Braun
Institute of Computer Science and Applied Mathematics
University of Bern
Neubrueckstrasse 10, CH-3012, Bern, Switzerland
Email: oliveira,braun@iam.unibe.ch

*Abstract*— **In recent years, a great deal of effort has been devoted to make the TCP protocol more resilient to the random packet losses inherent in the wireless channels in ad hoc networks. In this paper, we investigate the use of fuzzy logic theory for assisting the TCP error detection mechanism in such networks. An elementary fuzzy logic engine is presented as an intelligent technique for discriminating packet loss due to congestion from packet loss by wireless induced errors. The architecture of the proposed fuzzy-based error detection mechanism is also introduced and discussed. The full approach, for inferring the internal state of the network, relies on Round Trip Time (RTT) measurements only. Hence, this is an end-to-end scheme which requires only end nodes cooperation. Preliminary simulation evaluations show how viable this approach may be.**

## I. Introduction

The self-organizing mobile ad hoc networks have become more and more topical in the research community over the past few years. These networks are remarkable for not depending on any fixed infrastructure to communicate. The main challenge for such networks is the development of robust protocols able to cope with the high probability of topology changes and wireless impairments in place. In particular, ad hoc networks pose some tough challenges to the Transmission Control Protocol (TCP) since it was not designed to work in such highly dynamic environments. Rather, TCP was designed to work in wired networks where a packet loss can safely be associated to network congestion. So, regular TCP assumes that any packet loss is the result of congestion and appropriately responds to it by slowing down its sending rate.

In wireless mobile ad hoc scenarios, however, such losses may occur not only by congestion but also due to both the typically high Bit Error Rate (BER) of the wireless channels and link interruptions (by mobility). Channel errors induce TCP to mistakenly reduce its transmission rate (by halving its congestion window - cwnd). Mobility induced losses (link interruption) may lead TCP to incredibly long periods of inactivity due to its "exponential backoff mechanism". In both cases the TCP end-to-end throughput will be impaired. These problems may be solved by providing the TCP error detection mechanism with the real cause of every packet loss.

Existing approaches may be classified into two classes: Network oriented [1], [2], [3] and end-to-end [4], [5], [6], [7] solutions. In the former, the end nodes rely on explicit message notifications from inside the network to detect congestion or link interruptions. The main drawbacks here are high dependence on lower layer protocols to carry the messages and necessity of changes in the intermediate nodes, which may not only delay deployment but also pose security concerns as such nodes need full access to the packet header. End-to-end approaches, on the other hand, do not need any explicit cooperation of the intermediate nodes and may be somewhat independent of lower layer protocols. This makes such approaches easier to deploy since changes are limited to the end nodes. Our approach is end-to-end based as follows.

The key idea of our approach is to monitor permanently the TCP flow and record useful data to infer the current state of the network when packet losses are perceived. Actually, Round Trip Time (RTT) measurements are used as indicator of the internal state of the network. The rationale here is that TCP already relies in this parameter for computing its fundamental retransmission timeout (RTO) timer, and such measurements may be really valuable to reflect the condition inside the network [4], [5], [6]. Nevertheless, RTT measurements are not that trivial to be evaluated as they contain imprecision and uncertainties under certain conditions, which calls for an elaborate tool in order to extract the useful data.

We make use of Fuzzy Logic theory [8], [9], [10], for distinguishing between bit error and congestion induced losses, using RTT values as input variables. By using fuzzy logic, the continuous and imprecise behavior of the information can be handled without the necessity of arbitrary rigid boundaries. Besides, it is low processing demanding. This renders fuzzy logic quite suitable for evaluating RTT values where imprecision and uncertainties are effectively present and the processing requirements (at the end nodes) must be as low as possible. In this paper, which is an extension of [7], we focus on our approach for enhancing the TCP error detection mechanism using fuzzy logic as its main part. TCP recovery strategy is left for future work.

The remainder of this paper is organized as follows. The next section characterizes RTT patterns in ad hoc networks and exhibits the simulation scenario in place. In section III we explain the proposed fuzzy-based error detection mechanism. Section IV introduces fuzzy logic. Section V describes our fuzzy engine for distinguishing congestion from channel error losses. Section VI presents the performance evaluation of our fuzzy engine, and section VII concludes the paper.

## II. RTT Patterns

We assume in this paper that the packet size of the connection and the strategy of 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 protocols ensure symmetric links, which is required here.

Taking the two assumptions above into consideration, RTT values might be really useful for distinguishing the effects of congestion from bit error effects. We have shown in [5] that both the growth in the number of hops from sender to receiver and congestion conditions may impose similar behaviors to RTTs. Thus, sender and receiver should exchange information to detect the number of hops between them. We propose to use the Time To Live field (TTL) in the IP header, as explained in the next section. We focus here, however, on the discrimination between congestion and channel error induced losses having a fixed number of hops end-to-end.

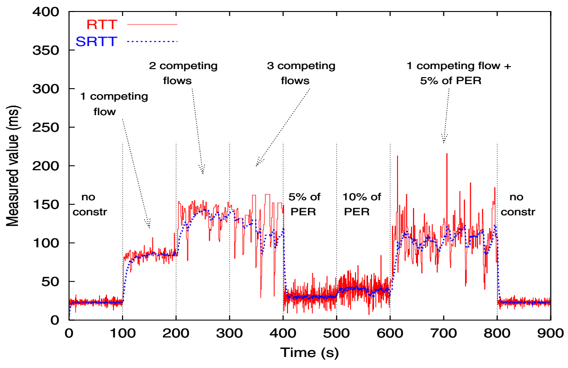
### A. Simulation Environment

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, the maximum congestion window is set to 8, and both the main and the background flows are generated by FTP applications over TCP. A uniform distribution function is used as error pattern for generating the different BER levels, and the channel bandwidth is 2 Mbps.

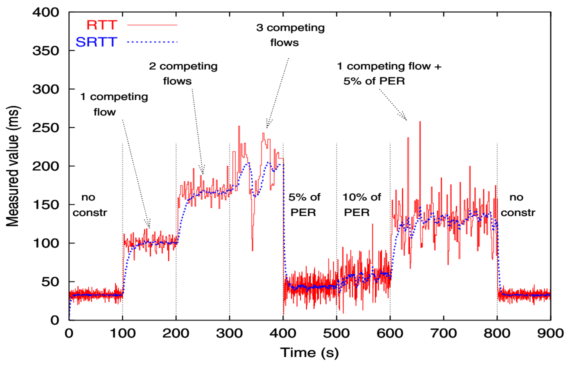
### B. RTT measurements

Fig. 1(a) depicts RTT and SRTT (Smoothed or average RTT) values on a typical 2-hop ad hoc network under varying levels of congestion and distinct bit error rates, starting at 100 seconds. From 100 to 400 seconds there are three stages of congestion, i.e., 1-3 competing flows. From 400 to 600 seconds, there is no congestion, but only two levels of bit error rate, i.e., 5 and 10% of Packet Error Rate (PER) which is the unit used here for expressing bit error constraints. From 600 to 800 seconds there is just one competing flow and a PER of 5%. In the last 100 seconds, the channel is free of any constraint again. Fig. 1(b) shows the same situation for a 3-hop scenario.

Fig. 1 shows clearly that under operative congestion conditions (minimal number of incoming ACKs) and without mobility, RTT mean values (roughly given by SRTT) should suffice to indicate congestion inside the network. The bit error constraint increases RTTs as well, but at a far lower degree. The only possibility for mistakes would be a very high level of bit error rate being misdetected as congestion,



(a) 2-hop scenario.



(b) 3-hop scenario.

Fig. 1. RTT characteristics under congestion and wireless losses.

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. Furthermore, typical wireless environments do not have so lossy channels.

To make the distinction of congestion from wireless induced losses even more robust, the RTT variance can be used to address the remote possibility mentioned above. Fig. 1 shows that the RTT variance (magnitude of the oscillations) increases under bit error conditions. It is worth to compare the case in which there is just one competing flow (100-200) with the instant of highest PER (500-600). Note that for both scenarios (2 and 3-hop) the channel error constraint induces higher RTT variance. For lower congestion levels, which could be achieved with other forms of competing traffic, the distinction would be even better as the variance under congestion would be smaller.

Figs. 1(a) and 1(b) show that in the last part of the simulation under constraint, (600-800), both RTT mean and variance grow. The key point here is that congestion detection has priority over bit error detection, as the TCP sender must slow down anyway under congestion, regardless of simultaneous bit error losses. Thus, if congestion is perceived, the detection algorithm may ignore the bit error detection.

## III. Fuzzy-based Error Detection

The proposed Fuzzy-based Error Detection Mechanism (FEDM) only relies on fuzzy logic for discriminating losses due to congestion from losses by channel errors. Losses by link interruptions are identified differently, as shown in [5], since no ACK is received in such conditions. Additionally, the designed fuzzy engine has to be adjusted in accordance with the number of hops traversed by the connection, which requires that the FEDM monitors permanently the number of hops end-to-end, as explained in the following.

Fig. 2 depicts the general architecture of our approach. The NH (Number of Hops) and RR (RTT increase Rate) blocks are needed for detecting the number of hops in the end-to-end connection and steep increases in the RTT measurements, respectively. C, U and B represent the signaling flags for Congestion, Uncertain, and Bit error, respectively.

The NH block keeps track of the number of hops in the TCP session so that the Improved Error Detector (IED) can set the Fuzzy engine parameters properly. This is needed because such parameters change according to the number of hops in the end-to-end connection, i.e., the more hops the higher delay range. At this stage, the parameters for every number of hops have to be determined in advance by simulation or experiment. Having distinct settings for every connection length, in terms of hops, solves the problem that arises by the similarity of RTT behaviors (steep increase) when either congestion starts or the number of hops between the end nodes increase [5].

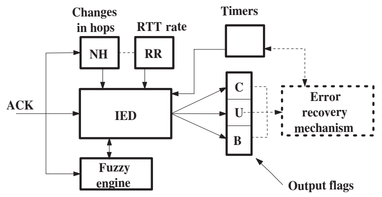


Fig. 2. Fuzzy-based Error Detection Mechanism.

We propose to use the Time To Live (TTL) field within the IP header for identifying the mentioned number of hops, mainly because it is standard in current implementations. This demands a simple interoperation between transport and network layer protocols and the use of either an IP or TCP option inside the packet header. Upon packet receipts, the NH block may use the current TTL along with the TTL sent to compute the exact number of hops crossed by the connection.

The RR block monitors the rate at which the RTT changes. The idea here is to set a threshold for triggering detections of persistently high rate increase in the RTTs faster than the fuzzy engine might do, thereby avoiding network collapse. Specifically, this mechanism detects a predefined number $n$ of subsequent increases in RTT, in which every increment is larger than a given degree $\alpha$. Based on our preliminary simulation evaluations, shown in Fig. 1, n=2 and $\alpha$ = 20% perform well with a RTT granularity of 100 ms (mimimum interval between successive samples). When triggered, this mechanism notifies the IED which can set the congestion flag (C) if no extra hop is detected simultaneously. NH and RR blocks work together as the RTT persistently high rate growth has to be detected only when no increase in hops is identified.

The three blocks around the IED provide it with enough information about the state of the wireless channel, so that it may take decisions more accurately. The outcome of the decision is set in the output flags. The flag Uncertain (U) refers to the undefined output of the fuzzy engine which may not always provide a conclusive output, as discussed below.

Losses by link interruptions will be detected by the timeout timer along with the state flags. Timeout with Congestion (C) call for retransmission and slowdown, while timeout plus bit error (B) ask for retransmission only. Timeout with uncertain (U) should also call for retransmission and slowdown considering a conservative approach. Observe that the fast retransmit mechanism is also in place for speeding up retransmissions before timeouts may occur. The exact actions to be taken depend on the error recovery mechanism strategy in use. In any case, whenever an action is taken under the uncertain (U) condition, the current TCP state variables such as congestion window and slow start threshold should be saved for possible recovery upon effective condition detection.

## IV. Fuzzy Logic

Fuzzy logic is a superset of conventional (boolean) logic that has been extended to handle the concept of partial truth. It was first introduced by L. Zadeh in the 1960s [8] as a means to model the uncertainty of natural language, and has been widely used for supporting intelligent systems. A key feature of Fuzzy logic is to handle uncertainties and non-linearities, existing in physical systems, similarly to the reasoning conducted by human beings, which makes it very attractive for decision making systems. A fuzzy logic system comprises basically three elements: A fuzzifier, an inference method (rules and reasoning) and a defuzzifier. Their roles are as follows.

### A. *Fuzzifier (toward Fuzzy Sets)*

A fuzzifier is responsible for mapping discrete (also called crisp) input data into proper values in the fuzzy logic space. This is done by using membership functions (fuzzy sets) which may provide smooth transitions from false to true (0 to 1). Mathematically, a membership function associates each element $\mu X(x)$ in the universe of discourse U with a number in the interval [0,1] as shown in (1):

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

Therefore, a fuzzifier maps crisp data $x \in$ U into a fuzzy set $X \in$ U, and $\mu X(x)$ gives the degree of membership of $x$ to the fuzzy set $X$, i.e., a real number in the range [0,1]. Where 1 denotes full membership and 0 denotes no membership. So, fuzzy sets are indeed an extension of the classical sets in which only full membership or no membership exist. Fuzzy sets, on the other hand, allow partial membership.

### *B. Fuzzy Rules and Fuzzy Reasoning*

Fuzzy systems perform reasoning on the input data by following a predefined inference method (section IV-D) and fuzzy rules. The amount of rules depends on both the number of inputs and membership functions associated to each input. The general form of the $l_{th}$ fuzzy rule in the rulebase is:

$$R^l \quad : \quad \textbf{if } (x_1 \; is \; F_1^l) \text{ and } (x_2 \; is \; F_2^l) \text{ and} \ldots (x_p \; is \; F_p^l) \\ \textbf{then } (y \; is \; G^l) \tag{2}$$

Where $F_k^l$ and $G^l$ are fuzzy sets associated with the input and output fuzzy variables $x_k$ and $y$, respectively, being $k = 1, ..., p$. As an example of (2), we could have: if (temperature is high) and (humidity is high) then (room is hot).

### *C. Defuzzifier*

Once the input data have been numerically processed by fuzzy reasoning, they are converted back to crisp values. This task is performed by the deffuzifier which combines together mathematically the result of each rule into a single crisp value. There are several methods for doing so, and we use here the most widely used algorithm called gravity-of-mass (GOM) [9], [10], which computes in the simplest case the weighted average over all rule outputs, as explained below.

### *D. Min-max Inference Method as an Example*

For further clarification, we briefly describe here, via an example, a variant of the most widely used inference method labeled min-max. Fig. 3 illustrates the method considering two inputs and two rules only. There are two fuzzy sets labeled ”low” and ”medium” (linguistic variables) for both the input (x1, x2) and output (y) membership functions. The crisp input values are mapped into the membership functions (fuzzification) and assessed according to the rules in place.

Each rule (see Fig. 3) is applied to the involved membership functions in x1 and x2 and the minimum (min) of them is mapped into the associated output membership function in y (low or medium). The output of each rule is aggregated (max) into the deffuzifier which gives the final crisp value that will indicate, in this example, whether the outcome is to be assigned to ”low” or ”medium”. As mentioned earlier, several schemes for defuzzification exist, and for this simplified sort of output membership function (single value in y), the gravity-of-mass method gives the weighted average over all output values in y.

## V. Proposed Fuzzy Logic Engine

The behavior of the RTT measurements presented in section II 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 computationally inexpensive. Thus, a fuzzy logic engine has been designed for handling RTT measurements toward effective packet loss discrimination. The fuzzy engine design follows the work presented in [10] where packet delay behavior is used for detecting wireless links.

### *A. Fuzzy Engine Input*

The input variables of the fuzzy engine are defined as the RTT mean $t$ in (3) and the RTT variance $\delta_t$ in (4), being i=1, 2, ... , maxSamples. Since the maximum value of RTT is always limited by either the timeout timer or the fast retransmit mechanism, it is reasonable to assume that it is limited to [0, Tmax] without loosing generality.

Equations (3) and (4) imply that the universe of discourse of the fuzzy input variables $t$ and $\delta_t$ should be [0, Tmax]. A fuzzifier has to map the crisp values $t$ and $\delta_t$ into fuzzy data.

$$t = \frac{1}{n}\sum_{i=1}^{n} t_i \tag{3}$$

$$\delta_t = \frac{1}{n}\sum_{i=1}^{n} (t_i - t)^2 \tag{4}$$

For computing simplicity and better control of the spread of the curves [10], we use in this paper Gaussian membership functions for the input memberships (Fig. 4(a)). The universe of the fuzzy input variables $t$ and $\delta_t$ are divided into three fuzzy sets as shown in Fig. 4(a). Observe that here we have one more fuzzy set than in the example of Fig. 3. The fuzzy linguistic variables used are S (Small), M (Medium) and L (Large). Thus, the input values have to be mapped to these fuzzy sets, as illustrated in the example of Fig. 3. Note that the more fuzzy sets, the more accurate may be the input discrimination. The number of fuzzy rules and complexity increase though.

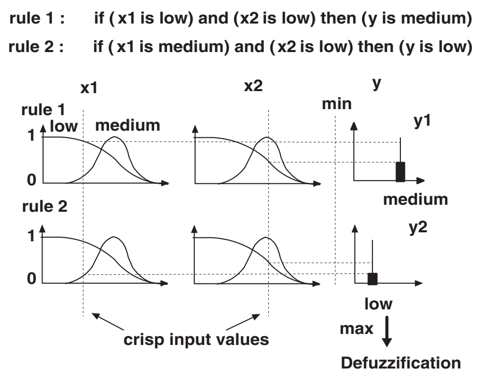
### *B. Fuzzy Engine Output*

The output of each fuzzy rule in our fuzzy engine is assigned to a corresponding output fuzzy set. The output of the fuzzy engine might take several distinct forms. For instance, there might be two fuzzy output sets, one for bit error detection and another for congestion detection. Another possibility should be to have both detections in just a single fuzzy output. The universe of the fuzzy output has to be set accordingly. In this paper, we use just a single fuzzy output $\phi$ (Fig. 4(b)).

Hence, the output of the fuzzy engine is set as the discrimination of congestion from bit error effects, and the universe of the fuzzy output variable is split into three singleton (single value) fuzzy sets as depicted in Fig. 4(b). This means that each rule output will be placed at either 0 or 0.5 or 1 in the universe of the single output $\phi$.

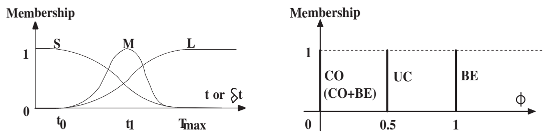
The corresponding fuzzy linguistic variables are CO (Congestion), UC (Uncertain) and BE (Bit Error). As congestion has priority over bit error, the CO variable covers also conditions of simultaneous congestion and bit error constraints. The specific fuzzy rules are shown in table I.

Table I shows, for instance, that small (S) ”RTT mean” and large (L) ”RTT variance” indicate clearly that the measured flow is facing bit error constraint (BE). Likewise, large (L) ”RTT mean” suggest congestion (CO), regardless of the variance value. The other rules are similarly set up according to the RTT evaluation discussed in section II. The fuzzy reasoning is done on the basis of the min-max method explained as an example in section IV-D, and the gravity-of-mass is the deffuzifier in place.



rule 1 : if ( x1 is low) and (x2 is low) then (y is medium)
rule 2 : if ( x1 is medium) and (x2 is low) then (y is low)
x1
x2
y
min
rule 1
low
medium
1
0
y1
medium
rule 2
1
0
y2
low
max
crisp input values
Defuzzification

Fig. 3. An example of the min-max inference method.



(a) Input.

(b) Output.

Fig. 4. Fuzzy memberhip functions.

TABLE I

FUZZY RULES OUTPUT ($\phi$)

| var \ mean | S | M | L |
|---|---|---|---|
| **S** | BE | CO | CO |
| **M** | BE | UN | CO |
| **L** | BE | BE | CO |

## VI. PERFORMANCE EVALUATION OF THE FUZZY ENGINE

We conducted simulations using the ns-2 simulator for collecting representative data to feed our fuzzy engine. The simulation setup is the same described in section II, and we considered only the 3-hop scenario since the procedure is the same for all the others cases.

We ran similar conditions presented in section II except that we considered only two extra FTP flows for disturbing the wireless channel. The rationale here is that three flows generate so many losses that it becomes difficult to have representative results with the short runs performed here. Unless otherwise mentioned, each interval under constraint lasts 100 seconds.

In order to find out the proper parameters to set up the fuzzy engine (Fig. 4(a)), the most critical conditions were taken into account. As stated in section II, low congestion level might be misdetected as bit error and vice-versa. Hence, the case in which the connection is facing only one competing flow and the one with 10% of PER were taken as boundaries.

Due to lack of space we do not show in detail here how much of overlapping exist between the RTT mean and variance computed under congestion and channel error constraints from the measurements of Fig. 1(b). We have shown in [7] that RTT mean and variance distributions for this scenario contain very small and medium overlapping areas, respectively. In fact, RTT mean can be distinguished quite easily since its spread is really low, while RTT variance spreads much more as discussed in section II and shown in Fig. 1.

One can see roughly in Fig. 1(b) that the interval (500-600) contains a RTT mean around 50 ms and practically no overlapping with the values in the interval (100-200). So the parameters t0, t1 and tmax for RTT mean in Fig. 4(a) were set to 40, 50, 60, respectively. Analogously, the RTT variance distribution (refer to [7]) shows that the variations from congestion and channel errors overlap mostly in the range 0-200 ms, and so the parameters t0, t1 and tmax for RTT variance in Fig. 4(a) were set to 50, 100, 150, respectively. These settings ensure that the reasoning for transitions between the different conditions inside the network take place smoothly.

The universe of the input variables is 500 ms (maximum RTT measured). And the ranges assigned to the output parameters CO, UC and BE are: [0,0.3), [0.3,0.7], (0.7,1], respectively, i.e., neighborhood of 0.0, 0.5 and 1.0 in Fig. 4(b). Recall that the fuzzy engine will provide an output value in the range [0,1], and the ranges above will define if such a value represents congestion, uncertain, or bit error.

Fig. 5 shows the fuzzy engine performance concerning the number of correct detections over three distinct conditions, namely under congestion, bit errors (10% of PER) and the combination of both (5% of PER + 1 competing flow). For each of the three conditions we ran three different RTT sampling rates, namely 20, 40 and 60 RTTs per sample. In these evaluations we enabled the TCP timestamp option, so that each incoming ACK carried its experienced RTT. So the RTTs depicted in Fig. 5 correspond to the incoming ACKs. The purpose here was to determine not only the 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 sampled ACKs. The results could certainly be fine tuned by using more elaborate membership functions and different settings, and more accuracy from the input values could be achieved by including more fuzzy sets into the fuzzy engine.

The results depicted in Fig. 5 make it clear that the fuzzy engine may provide accurate results as long as a reasonable number of RTTs is taken in each sampling for computing the mean and variance from (4). Nevertheless, abrupt changes toward congestion might lead the network to collapse if the engine does not detect that in advance. Because of that we have proposed the RR block in Fig. 2 to make sure that, in such cases, congestion will always be detected before the first packet loss is perceived by the TCP sender.

Hence, we simulated conditions in which the channel was initially facing some level of PER (1, 3 and 5%) and suddenly a heavy congestion started. The $n$ and $\alpha$ parameters of the RR block were set to 2 and 20%, respectively. The results are depicted in Fig. 6 where the delays are normalized to the time the regular TCP takes to detect the first lost packet. The main outcome is that even without the RR block, the Fuzzy-based

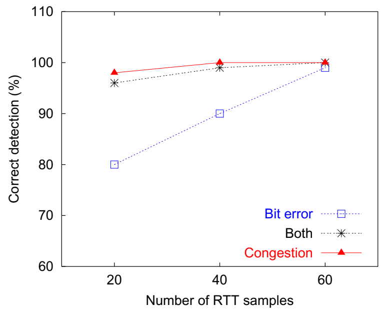


Fig. 5. Correctness of the fuzzy engine.

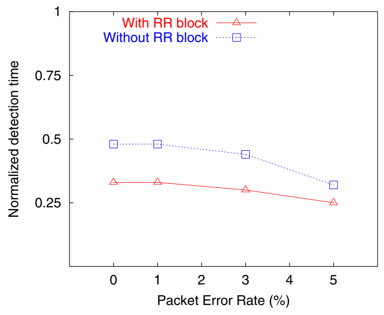


Fig. 6. FEDM (Fig. 2) under abrupt congestion.

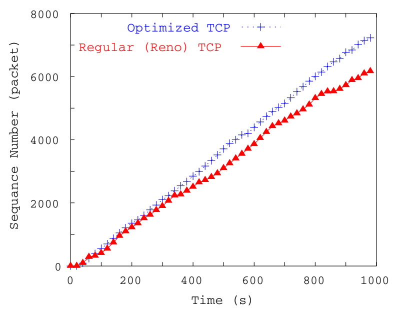


Fig. 7. Modified TCP performance, PER=10%.

Error Detection Mechanism (FEDM) in Fig. 2 was able to detect incipient congestion quicker than the regular TCP. The improvements from the RR block were not so significant and at higher PERs the detections occurred faster.

To substantiate our discussions, we extended the simulator code so it did not slow down in the event of packet loss due to medium error. We simulated a condition in which the only constraint in the wireless channel was 10% of PER, and the run lasted 1000 seconds. The sequence numbers of the successfully transmitted packets are shown in Fig. 7, in which the modified TCP clearly outperforms the well-known TCP Reno by getting more transmissions over the same interval.

In short, the results have shown that the fuzzy engine may indeed distinguish congestion from channel error conditions, and consequently assist the TCP error detection. However, improvements are certainly possible as the model here studied is rather modest. For instance, independent fuzzy outputs for each of the evaluated conditions (congestion and medium error) could provide more flexibility in adjusting the engine. The membership functions can be optimized by using advanced learning/training techniques such as ANFIS [15], and self-adaptive setting models can render our approach very robust.

## VII. Conclusions

We have introduced and evaluated a fuzzy logic engine for supporting TCP error detection mechanism in ad hoc networks. The architecture of the enhanced error detector has been explained, and its primary features discussed.

The main conclusion is that efficiency can be obtained provided that the input data are taken precisely enough to reflect the actual changes inside the network. This implies that a minimum number of ACKs is needed to ensure efficiency in the results. This may render this algorithm a bit slow for detecting transitions between distinct channel states. On the other hand, the proposed mechanism will be quite robust in dealing with steady stead scenarios, where abrupt changes are not too frequent, for instance a lossy channel. We proposed supporting schemes for accelerating our mechanism, which may boost the performance of our model as a whole.

Therefore, the overall evaluation is positive in the sense that we applied an intelligent algorithm for inferring statistically the internal state of the network, and the outcome was surprisingly accurate. However, different scenarios and more elaborate inference models have to be checked to render our proposal even more generic. Its integration with the error recovery mechanism is also to be assessed in future work.

## Acknowledgments

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.

## References

[1] 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.

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

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

[4] 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.

[5] 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.

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

[7] R. Oliveira and T. Braun, *A Fuzzy Logic Engine to Assist TCP Error Detection in Wireless Mobile Ad Hoc Networks*, New2an 2004, February 2004.

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

[9] *Fuzzy Logic Toolbox User's Guide, Version 2.1.2*, MathWorks, July 2002.

[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 RFC 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] J. Jang, *ANFIS: Adaptive-Network-Based Fuzzy Inference System*, IEEE Transactions on Systems, Man, and Cybernetics, Vol. 23, Nr. 3, pp. 665-685, 1993.