A reputation-based approach using collaborative indictment/exculpation for detecting and isolating selfish nodes in MANETs

Lotfi Zaouche, Sofiane Aitarab, Anfel Khireddine, Mawloud Omar, Enrico Natalizio, Abdelmadjid Bouabdallah

A reputation-based approach using collaborative indictment/exculpation for detecting and isolating selfish nodes in MANETs

Lotfi Zaouche[(1)], Sofiane Ait Arab[(2)], Anfel Khireddine[(3)],
Mawloud Omar[(3)], Enrico Natalizio[(1)], Abdelmadjid Bouabdallah[(1)]

[(1)] Heudiasyc Lab - UMR CNRS 7253, Université de Technologie de Compiègne, <name.surname>@hds.utc.fr
[(2)] LISSI Lab - Université de Paris Est Créteil, sofiane.aitarab@gmail.com
[(3)] LIMED Lab – Université de Bejaia, Algérie, {khireddine.anfel@gmail.com, mawloud.omar@gmail.com}

***Abstract*—Collaboration between nodes in Mobile Ad hoc Networks (MANETs) is very important for the proper functioning of the network. This is an assumption that has to be fulfilled in the design of routing protocols. However, this is not always true since some nodes could misbehave in order to have some benefits or simply avoid wasting resources. In this paper we question this assumption that does not take into consideration the bad behavior of nodes involved in the routing protocols. We analyze the characteristics of the existing solutions, and we propose a reputation-based mechanism that isolates selfish nodes based on control packets generated as a result to nodes' observations on the behavior of other nodes. We propose a mathematical framework to increase/decrease the reputation of a node depending on the situation and the observation condition. We show via simulation that our solution achieve remarkable improvements in the delivery rate of packets, more than satisfying results concerning false positive and false negative, and it shows that the overhead caused by our system is negligible.**

Keywords—MANET; selfish behavior; reputation system;

I. INTRODUCTION

In Wireless Sensor Networks (WSN), collaboration between nodes is essential. If a source and destination of data flow are not in line of sight, the information should be transmitted along intermediate nodes, along a path established and maintained by the network. Routing in such conditions becomes a complex task, especially as energy resources are limited, and nodes can legitimately become selfish and refuse to route other nodes' packets to preserve their energy. The need for cooperation between nodes to ensure the functioning of the network conflicts with the individual interest of each node to spend its energy solely for data for which they are the source or the destination. We can identify two types of non-cooperative nodes: faulty/malicious nodes and selfish nodes. Faulty/malicious nodes belong to the class of nodes that are either defective and therefore cannot follow a well-defined protocol, or intentionally malicious, thus trying to attack the system [5].

Although the problem of selfishness is a form of passive attack, it still causing a negative impact on network's performances. Numerous studies have been performed to evaluate the impact of the presence of selfish nodes in an ad hoc network [6, 7, 9]. The non-cooperation of a node implies that packets passing through this node will be lost. The mentioned problem calls for solutions that force the selfish nodes to cooperate in the network and if necessary, excluding them. Such solutions would greatly increase the network performance. In this work, we are interested in the study of the proposed problem of selfishness in mobile ad hoc networks.

Our contribution is summarized in the following points: We propose an improvement of the TWOACK scheme [3], aimed at considerably decreasing the number of control packets. Also, the TWOACK scheme is only able to detect a selfish link, whereas our solution detects the selfish node. We propose reward/punishment model for cooperative/selfish nodes by taking into consideration all nodes that participate in the deliverance of a packet.

The rest of the paper is organized in four sections. In Section II, we present the state of the art of solutions for the problem of selfishness. In Section III we present our solution approach. Section IV is devoted to the presentation of the simulation results. Section V concludes the paper.

II. RELATED WORK

Since the transmission of a message imposes a cost (energy and other resources) to the nodes, a selfish node will need an inducement or reward for transmitting messages from others [8]. There are two types of solutions to encourage selfish nodes in ad hoc mobile network to cooperate: credit-based systems and reputation-based system.

A. Credit-based system

Systems based on credit provide incentives to nodes to ensure network functionality. To achieve this virtual goal, a payment system may be implemented. Nodes are paid to rely other nodes' packets. This kind of system can be implemented using two models: the Packet Purse Model (PPM), and Packet Trade Model (PTM) [2].

In [8], the authors proposed an interesting solution called SPRITE. When a node receives a message, it keeps a receipt of this message, then when it has a fast connection to the Credit Clearance Service (CCS) it reports its receipts of the messages it received/transmitted. CCS then determines the charge and the credit of each node involved in the transmission of the message.

In this type of solution, we are facing new problems such as the centralization/decentralization of the paying authority, false receipts and sometimes the solution needs to address not only software issues but also hardware ones.

B. Reputation-based System

A reputation-based system relies on the observations of nodes to other nodes. Since one observation does not allow a direct and objective measure of malicious nodes, it is necessary that each node maintains a degree of confidence in respect of all the nodes it has observed. The value of this confidence is influenced by observations on the behavior of nodes. In this type of system, the reputation calculation is either performed locally at each node, or by the distribution of reputations stored in the nodes within the network.

TWOACK scheme [3] is based on reputation. A node that transmits/broadcasts a message, is informed that the following node has completed its task by forwarding the message at his turn, by receiving from the two hops next node a special acknowledgment called TWOACK packet. Each node that receives a message must send an acknowledgment to the node two hops back in the message path. The message path is the path that has been given by the routing protocol. To detect a misbehaving node, the source maintains a list of IDs of

messages that he has not received TWOACK packet yet, and each node maintains a unique list of data structure for each transmission link that it uses.

S-TWOACK (Selective-TWOACK) [3] and 2ACK [4] schemes aims at reducing network congestion caused by the large number of TWOACK packets sent. The first is inspired by the principle of the sliding window, acknowledging a certain number of well received messages. The second one acknowledges only a part of them, and includes a certification mechanism for the security of its packets.

In respect of the presented solutions, our proposal consists in optimizing the control packets. We do not generate control packet until something does not work in the message delivery, whereas TWOACK generates control packet during all transmissions. Consequently, the more selfish nodes are discovered, the less control packets are generated.

III. PROPOSED APPROACH

Since it is the less constraining in the architectural design, and there is no need to have special hardware, we choose to work on Credit-based systems to force the nodes to participate with other nodes, in order to keep a good reputation and keep being well served by others.

A good solution should: (i) guarantee the detection of selfish nodes, (ii) penalize the selfish nodes, and avoid, in the routing phase, those excluded because they do not cooperate anymore (iii) be able to know if it is necessary to give a second chance to a node who wants to repent.

A. Assumptions

We assume that the links are bidirectional. We also assume that a certification service public key is set up and used to encrypt the messages circulating in the network, including messages that are unique to our system, and guarantee data integrity.

B. Operating details of our approach

The principle of our approach is quite simple, and relies on multi-hop acknowledgment. Several studies justify and prove that two hops is an efficient number of hops for the acknowledgments [3, 4]. Based on these studies, we choose to make two hops acknowledgment, because it will make the indictment of a node more precise than in the case with higher number of hops. Messages used by the system are described in Table I.

TABLE I. MESSAGES LIST

Message	Definition
2HopAck	Message sent by a node Ni to Ni-2 node.
SelfExculpation	Packet sent to the source by the last node that tried to transmit the message to report the refusal of a node to transmit the message,
Selfish_Detection	Packet sent by a node that does not receive the exculpation of his successor, thus accusing him of being selfish. It must be said here that if for example the message is sent by a node Ni, then the node Ni-1 will not accept to transmit that packet only if Ni+1 has exculpated Ni by sending a 2HopAck packet to him.
SelfishAlert	Packet sent in order to report the detection of a selfish node.

Knowing that we have made the assumption that a certification service is implemented in the network, *2HopAck* and *SelfExculpation* messages will be encrypted to ensure their integrity and authenticity.

Let consider that a node N_s wants to send a message to N_d. N_s builds a path to N_d by using any routing protocol, but avoiding the known selfish nodes. By sending this packet, all nodes that are on its path will wait for an acknowledgement of the destination for an *Ack_Delay* time. Upon the reception of the acknowledgment is received, each node increases the reputation of following nodes in the path. If the timeout *Ack_Delay* expires and no acknowledgment is received, each node N_i involved in the message transfer, send a 2HopAck message, to the node N_{i-2}, who was two hops back in the message path to prove that the node N_{i-1} who was the intermediate is innocent. When the node N_{i-2} receives this message, it will increase the reputation of the nodes N_{i-1} and N_i. When after a delay, *Exculpation_Delay*, the node N_i sees that the node N_{i+1} did not send the *2HopAck* packet to node N_{i-1}, it exculpates itself by sending the previous nodes a package *SelfExculpation* and hold N_{i+1} responsible for the failure of the transmission of the message, and decreases its reputation. Nodes receiving this message start increasing the reputation of nodes that transmitted the message to the node N_i, and since they do not know which node caused the problem, they will penalize the two nodes N_i and N_{i+1}, by decreasing the reputation of the node that seems most selfish. If after a delay, a node N_i does not receive the *2HopAck* packet from node N_{i+2}, and the node N_{i+1} does not proclaim its innocence, then it will be indicted by the node N_i, and will be signaled to the source node. All nodes in the path receiving this indictment will reduce the reputation of node N_{i+1}, and increase the reputation of other intermediate nodes. In addition, when a node receives a packet, it will check the message path, and will increase the reputation of all intermediate nodes from the source to him.

Each node holds a table named *TrustTable* that stores the values of reputation he has for other nodes. Whenever a node obtains an observation about a node, it updates the value of its reputation if it has an entry in the table for this node, if no entry in the table corresponds to this node, then it will create a new entry for this node and save the value inside. Initially, each node gives an initial reputation to neighboring nodes. In addition, each node holds a data table called *PostTable*, which stores the identifier of the messages it transmits, and the path it takes. When the destination sends an acknowledgment to the source to confirm the receipt of a message, all nodes that receive this acknowledgment, in their *PostTable* checks if there is a message that matches the packet acknowledged, then deletes the corresponding line. And nodes do the same when they receive *SelfExculpation* or *SelfishDetection* packets. If a timeout after no acknowledgment is received for a message, the line for this message will be deleted after it has sent the message to the node *2HopAck* two hops back, as explained above to exonerate one hop rear's node.

In order to more quickly detect selfish nodes, it is preferable that the network nodes collaborate by exchanging knowledge on the behavior of other nodes. In order not to clutter the network messages, the nodes transmit the values of the reputation of a node only if it changes by a certain threshold. The value of the threshold should be well studied; a value too small will make the collaboration stronger and therefore more effective, but will increase the load on the network, and will be a waste of energy to the nodes. Also, taking a bigger value will reduce the network load, but it will also reduce nodes collaboration, making the detection of selfish nodes slower.

C. Reward and punishment computation

Formulas' parameters and functions are detailed in Table II.

TABLE II. PARAMETER LIST

Parameter/function	Definition
RSD	Positive value to add to the reputation of a node having transmitted a message.
PNSD	Positive value to subtract of the reputation of a node as a punishment for failing to transmit a message.
APNSD	Positive to subtract also the node that seems to be the selfish node value.
RSM	Positive value to add to the reputation of a node as a reward for reporting the bad behavior of a node.

RST	Positive value to add to the reputation of a node as a bonus for sending 2HopAck packet.
lastBroadcastedTrust$_i$(j)	The last value of reputation had broadcast the node i about the node j.
Ack_Delay	The time that a node has to wait for acknowledgment of the message he collaborated to transmit. After that, the node Ni launches *Exculpation_Delay*.
Exculpation_Delay	The time node N_i must wait for the 2HopAck packet from N_{i+1} toward N_{i-1}. Beyond this period, the node N_i sends a packet *SelfExculpation*.
Others_Exculpation_Delay	The time a node N_i must wait 2HopAck packet from node N_{i+2} exculpating the node. N_{i+1}.
$\alpha_{reputation}$	The initial reputation of each node.
$\Delta_{reputation}$	Change threshold that must wait before broadcasting the new reputation of a node.
Threshold	The threshold for which a node is considered selfish if it goes below it.
WitnessRate	The required rate of nodes accusing a node to be selfish, to be considered as such throughout the network.
MPS(i, j)	A function that returns 1 if the reputation of the node *j* is greater than those of node *i*, and returns 0 otherwise.
Trust$_i$(j)	A function that returns a reputation of node *j* that holds node *i*, which is stored in the *TrustTable*.

Upon receipt of a reputation value of a node, a new reputation value is calculated for this node, taking into account the value received and the value we already had. To avoid defamatory values that distort the reputation of the nodes, we can take into account the values received with a low impact factor. We apply the following formula to calculate a new value of reputation taking into account the values received:

$$Trust_i(j) = \frac{\left(\varphi * Trust_i(j) + \sum_{j=1}^{n} receivedReputation_j\right)}{\varphi + n}$$

Where ϕ is the factor that is given to the reputation we have already calculated previously, *receivedReputaion* is the value of reputation received from any node.

When a node detects that the reputation of another node has fallen below a certain threshold, it broadcasts a packet named *SelfishAlert*, containing its identity and the identity of the accused node. Each node receiving this message will save it. If at a certain time you get a number of accusation for a given node, equal to *WitnessRate*, then this node will be considered selfish by all network nodes. This technique speeds up the process of detecting selfish nodes in the network.

The *WitnessRate* parameter is very important. Indeed, high levels reduce the rate of false accusations caused by defamatory information sent by malicious nodes, but the detection of selfish nodes becomes longer. A smaller rate ensures that the detection of selfish nodes is faster, but false positive rate could increase.

N_i rewards N_{i+1} for sending the message to N_{i+2} as follow:

$$Trust_i(N_{i+1}) = Trust_i(N_{i+1}) + RSD \quad (1)$$

And rewards N_{i+2} for confirming it as follow:

$$Trust_i(N_{i+2}) = Trust_i(N_{i+2}) + RST \quad (2)$$

When N_i receives a message from N_{i-1}, it applies:

$$Trust_i(N_j) = Trust_i(N_j) + RSD, \forall \boldsymbol{N_j} \in \{\boldsymbol{N_1}, \boldsymbol{N_2}, \dots, \boldsymbol{N_{i-1}}\} \quad (3)$$

A node penalizes his successor after it did not send *2HopAck* packet, using the following formula:

$$Trust_i(N_{i+2}) = Trust_i(N_{i+2}) - PNSD \quad (4)$$

When a node receives a *SelfExculpation* packet of a node *k*, it executes:

$$Trust_i(N_j) = Trust_i(N_j) + RSD, \forall \boldsymbol{N_j} \in \{\boldsymbol{N_{i+2}}, \boldsymbol{N_{i+3}}, \dots, \boldsymbol{N_{k-1}}\} \quad (5)$$

$$Trust_i(N_k) = Trust_i(N_k) - \left(1 - Trust_i(N_k)\right) * PNSD - MPS(N_k, N_{k+1}) * APNSD + MPS(N_{k+1}, N_k) * RSM \quad (6)$$

$$Trust_i(N_{k+1}) = Trust_i(N_{k+1}) - \left(1 - Trust_i(N_{k+1})\right) * PNSD - MPS(N_{k+1}, N_k) * APNSD \quad (7)$$

If we receive an acknowledgment of the destination these will be applied to each node N_i path:

$$Trust_i(N_j) = Trust_i(N_j) + RSD, \forall \boldsymbol{N_j} \in \{\boldsymbol{N_{i+1}}, \boldsymbol{N_{i+2}}, \dots, \boldsymbol{N_{d-1}}\} \quad (8)$$

$$Trust_i(N_{i+2}) = Trust_i(N_{i+2}) + RST \quad (9)$$

If no exoneration is sent neither by the node N_k itself nor by the node that follows it, a *Selfish_Detection* packet is sent, and the reputation of node N_k will be reduced by all nodes in the path receiving this message by applying formula (4).

IV. PERFORMANCE EVALUATION

In this section we will describe the simulation environment and we will show the simulation results.

A. The simulation environment

For our simulation campaigns, we implemented TWOACK and our protocol on Java. We simulated an ad hoc network with 50 mobile nodes. Selfish nodes are randomly selected from these 50 nodes with a percentage ranging from 0 to 40% in steps of 5%. These nodes move according to the Random Waypoint model. The maximum speed of a node is 15 m/s and the maximum idle time is 3 s. Network nodes have the same range that is equal to 150 meters. The deployment surface of nodes is 1000 * 1000 m². Each node sends a message according to Poisson distribution with parameter λ=10 ms. Finally, the simulation time is 300 seconds, and we repeat it 50 times each time we increment the selfish nodes rate in the network. To show that we are tolerant to collisions, we assume that the packets loss rate is 5%. As shown in [10], this packet loss rate is sufficient. Table III. summarizes these parameters.

TABLE III. SIMULATION PARAMETERS

Parameter	Value
Number of nodes	50
Rate selfish nodes	0-40 % (step 5%)
Maximum Speed	15 m/s
Maximum idle time	3 s
Radius increased	150 m
Width of the area	1000 m
Length of the area	1000 m
Simulation time	300 s
$\alpha_{reputation}$	0.75
Threshold	0.25
RSD	0.1
PNSD	0.15
APNSD	0.05
RST	0.05
RSM	0.03
$\Delta_{reputation}$	0,2
WitnessRate	5 %

The important thing to show here as an input, is the rate of selfish nodes in the network, and the time to. Concerning the output, first, we will show the delivery ration of messages. Right after, we will present the overhead caused by the control packet generated by our protocol.

B. Results and discussion

Impact of selfish nodes on message delivery

As shown in Fig. 1, our approach gives excellent results; indeed the lower bound of the success rate of message delivery exceeds 86%. Initially, in the absence of selfish nodes in the network, the rate of rescues message reaches 97%, then increasing the rate of selfish nodes in the network, the delivery rate decreases to 86% when the selfish node rate reaches 40%. And we can see that TWOACK degrades significantly when the selfish rate increases. This is due to the fact that they do not discover the selfish note, but the failing link. So whenever a selfish node moves and create a new link with another node, it can behave as it likes.

When we fix the selfish rate to 40% as shown in Fig. 2, we have at the beginning a message delivery rate of 86% because we have not yet detected the selfish nodes, and as and as these nodes are selfish and discovers that avoids, we the message delivery rate increases till 95%, and it corresponds exactly to the 5% of assumed collision. Meanwhile, TWOACK stabilizes between 40 to 45%.

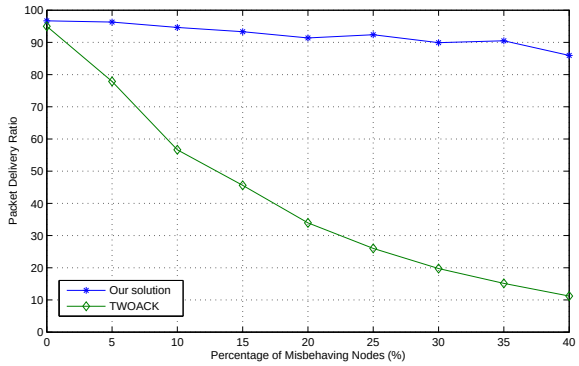

Fig.1: Packet delivery ratio

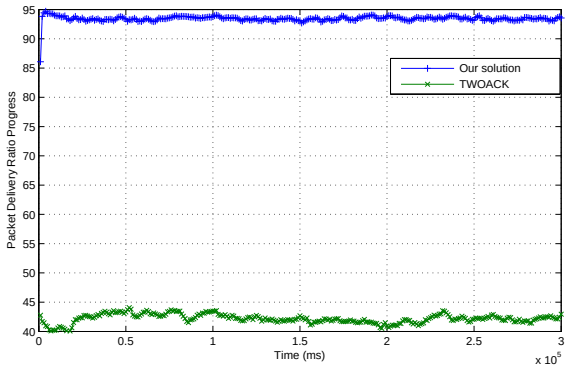

Fig.2: Packet delivery ratio over time

Control packets overhead on the network

The Fig. 3 shows the evolution of the protocol overhead over time. This is important to show to argue if our solution is too heavy to be interesting or not. Unfortunately, we did not see this kind of figure in the studied work. We can see that our solution is a very light one comparing to TWOACK. We do not exceed 4% of the network traffic, and we can do less than 1% in favorable condition. In fact, what makes control packets in our solution does not reach 0% is only caused by failing links, false detections, and collisions. Concerning TWOACK, we see that the control packet decreases, but this is only because the packets do not reach them destination and therefore no control packets are generated.

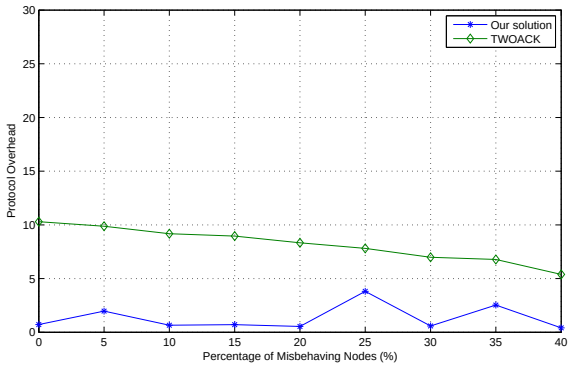

Fig.3: Routing overhead

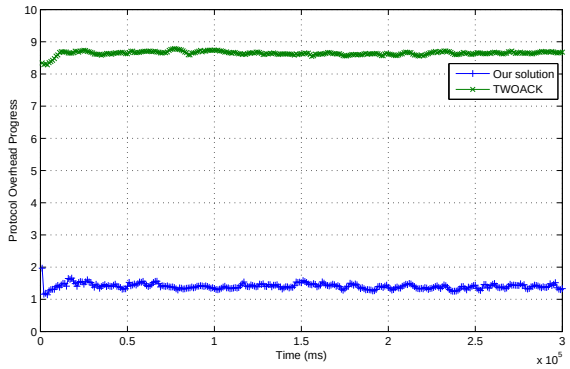

Fig.4: Routing overhead over time

In the Fig. 4, we can see that the two protocols have stabilized over time. Our solution stabilizes in 1.3% and TWOACK in 8.7%. Another time, as we said before, we do not 0% only because of false detections, collisions and link failing.

V. CONCLUSION AND FUTURE WORK

In this paper, we proposed a new approach for detecting selfish nodes. The simulation results that we obtained were largely sufficient, and have proven the effectiveness and robustness of our approach. As noted in the previous section, our approach goes beyond 90% regarding the detection rate of selfish nodes, which avoids these and offer a delivery rate high enough messages (more than 91%). The evaluation of our approach with respect to time clearly demonstrates its advantages. In fact as the time passes, the rate of successful delivery of messages increases, and the message loss decreases. When 40 of network nodes are selfish, the initial packet delivery ratio is 86% of all the sent packets, which increases to 95% over time.

REFERENCES

[1] H. Miranda and L. Rodrigues "Preventing selfishness in open mobile ad hoc networks". 23rd International Conference on Distributed Computing Systems Workshops, May 2003.

[2] S. D. Khatawkar, U. L. Kulkarni, K. K. Pandyaji "Detection of Routing Misbehavior in MANETs", International Conference on Computer and Software Modeling IPCSIT vol.14 IACSIT Press. Singapore 2011.

[3] Kash yap Balakrishnan, Jing Deng, Pramod K. Varshney. "TWOACK: Preventing Selfishness in Mobile Ad Hoc Networks", Wireless Communications and Networking Conference, IEEE, Vol. 4, March 2005.

[4] Kejun Liu, Jing Deng, Pramod K. Varshney, and Kashyap Balakrishnan «An Acknowledgment-based Approach for the Detection of Routing Misbehavior in MANETs". IEEE Transactions on Mobile Computing, Vol. 6, Issue: 5, May 2007.

[5] S. Marti, T. Giuli, K. Lai, and M. Baker, "Mitigating routing misbehavior in mobile ad hoc networks». In Proceedings of the Sixth International Conference on Mobile Computing and Networking. Boston 2000.

[6] Satyanarayana Vuppala, Alokparna Bandyopadhyay, Prasenjit Choudhury, Tanmay De. "A Simulation Analysis of Node Selfishness in MANET using NS-3". Int. J. of Recent Trends in Engineering and Technology, Vol. 4, No. 1, November 2010.

[7] Shailender Gupta, C. K. Nagpal, Charu Singla. “Impact of selfish node concentration in manets”. International Journal of Wireless & Mobile Networks (IJWMN) Vol. 3, No. 2, April 2011.

[8] Shen Zhong, Jiang Chen, Yang Richard Yang, "Sprite: A Simple, Cheat-Proof, Credit- Based System for Mobile Ad-Hoc Networks". Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies. vol. 3, April 2003.

[9] Sundararajan, A.Shanmugam. "Modeling the Behavior of Selfish Forwarding Nodes to Stimulate Cooperation in MANET", International Journal of Network Security & Its Applications (IJNSA), Vol 2, No. 2, April 2010.

[10] Constantinos Dovrolis, Parameswaran Ramanathann, "Proportional differentiated services, part II: loss rate differentiation and packet dropping", Eighth International Workshop on Quality of Service, June 2000.