# Multiplayer Game for DDoS Attacks Resilience in Ad hoc Networks

Antonis Michalas, Nikos Komninos and Neeli R. Prasad

***Abstract*—This paper proposes a multiplayer game to prevent Distributed Denial of Service attack (DDoS) in ad hoc networks. The multiplayer game is based on game theory and cryptographic puzzles. We divide requests from nodes into separate groups which decreases the ability of malicious nodes to cooperate with one another in order to effectively make a DDoS attack. Finally, through our experiments we have shown that the total overhead of the multiplayer game as well as the the total time that each node needs to be served is affordable for devices that have limited resources and for environments like ad hoc networks where nodes must exchange information really fast.**

***Index Terms*—DDoS, DoS, Ad-hoc Networks, Security.**

## I. Introduction

One of the major threats in network nodes is the so called Distributed Denial of Service (DDoS) attack. In the past years we saw lot of popular sites such as Yahoo, eBay, Amazon, CNN and many more to be under such attacks. DDoS attacks present a significant security threat to corporations, and the threat appears to be growing [11]. In the sixth of August (2009) the Social Networks world was under attack, in other words, we were in the middle of a planned attempt to take down two of the world's most popular social media sites: Facebook and Twitter. Even though no user data was at risk, the sites were down for several hours. DDoS, is a relatively simple, yet very powerful technique to attack Internet resources. DDoS attacks add the many-to-one dimension to the Denial of Service (DoS) problem making the prevention and mitigation of such attacks more difficult and the impact proportionally severe. DDoS exploits the intrinsic weakness of the Internet system architecture, its open resource access model, which paradoxically, also happens to be its greatest advantage [8].

In this paper we propose a multiplayer game to prevent DDoS attacks in ad hoc networks, where nodes do not necessarily belong to a single authority and the network topology is altered dynamically. Our approach combines cryptographic puzzles and game theory to develop a multiplayer game, where every node in the network that sends a request needs to solve a puzzle, and becomes a member of a specific group. We assume that all nodes in a group play the multiplayer game and are able to communicate directly. Each multiplayer game is conducted through a series of 2-player sub-games. Once all possible 2-player combinations have been played in a group, the highest scoring node is the one whose initial request is authorized, while the rest of the nodes in the group have to continue playing the game until all of their requests have been authorized. Hence, each group will end up having one last player, in which case a different type of sub-game will be played, between the remaining node and the node that is authorizing the requests. This method ensures that the bulk of the computation overhead is taken up by the requesting node, and not the authorizing node, which is only tasked with checking winning results. With the exception of dealing with the last remaining node in each group, the authorizing node is not competing directly in the game process, and not expending computational resources. In this method, a possible malicious node will be responsible for making the most expensive computations of the protocol in order to get access to the resources of another node in the network.

Following this introduction the paper is organized as follows. In Section 2 we examine related works that have been made in order to solve DoS and DDoS problems with the use of client puzzles and game theory. In Section 3 we describe our technique, combining game theory and puzzles. In Section 4 we analyze our approach and we present our experimental results and we conclude in Section 5.

## II. Related Work

Aura, Nikander, and Leiwo applied client puzzles to authentication protocols in general [13]. Dwork and Naor presented client puzzles as a general solution to controlling resource usage, and specifically for regulating junk email [14]. Waters, Juels and Felten proposed a Puzzle Outsourcing for IP-Level DoS resistance in [9].

C. Meadows in [12] showed how some principles that have already been used to make cryptographic protocols more resistant to DoS. By trading of the cost to defender against the cost of the attacker can be formalized based on a modification of the Gong-Syverson fail-stop model of cryptographic protocols, and indicates the ways in which existing cryptographic protocol analysis tools could be modified to operate within this formal framework.

Afrand Agah and Sajal K. Das in [2] formulated the prevention of DoS attacks in wireless sensor networks as a repeated game between an intrusion detector and nodes of a sensor network, where some of these nodes act maliciously. They proposed a protocol based on game theory which achieves the design objectives of truthfulness by recognizing the presence of nodes that agree to forward packets but fail to do so.

A game based analysis of the client puzzle approach in order to defend against DoS attacks have been introduced in [10], in which the optimal strategy is derived for the attacked server in order to respond to such attacks effectively.

A DoS mitigation technique is proposed in [3] that uses digital signatures to verify legitimate packets, and drop packets

that do not pass the verification. A network game is formulated in which nodes along a network path, are encouraged to act collectively to filter out bad packets in order to optimize their own benefits.

Narasimhan *et al.* in [1] introduced the notion of hidden puzzle difficulty, where the attacker cannot determine the difficulty of the puzzle without expending a minimal amount of computational resources. Game theory was used to develop defense mechanisms.

## III. Multiplayer Game

The communication initiators are involved in a multiplayer game where they are responsible for executing all the expensive computations on their own, while the node that is offering the service is only responsible for executing simple computations, mainly to facilitate the final stage of every multiplayer match. Furthermore, we do not make use of any intrusion detection system, there are no penalties for suspicious acting nodes and there is no database in which possible malicious users can be stored. Our solution is based on the fact that eventually every node that is requesting a service will gain access to it, but only after taking part in at least one game, and winning one (or more) games. The process requires every node to give some of his resources in order to gain access to the service he requested.

To be more precise we consider games with a finite number of players who interact not in a single contest but in pairwise games. The pairwise games are in fact part of one larger multiplayer conflict, with all the added complications that such a relationship implies. We divide our method into two main steps: *Group Formation* and *Multiplayer Game*.

### A. Groups Formation

The first thing that has to be done is to categorize requests into groups. The idea behind group formation is that all nodes in a group have to take part in a multiplayer game. After each game only the winner will have the opportunity to have his request processed, while the rest of the grouped nodes will have to continue playing the game. The multiplayer game for each group consists of multiple 2-player sub-games.

As we can see in figure 1, $node_1$ is receiving requests from multiple neighboring nodes in a DDoS scenario. First, $node_1$ is responsible for creating a set of groups from the nodes trying to communicate with it. In every group the nodes play a multiplayer game which is divided into 2-player sub-games. Each node has to play at least one 2-player game with all the other nodes of the group to complete a game. The node that wins a main game in a group will have his request processed by $node_1$. A new main game then restarts in that group, and the process repeats itself until all possible 2-player games are played. In figure 1, we see that $node_1$ has divided nodes into two groups ($Group_1$, $Group_2$). The first group has three nodes - players ($node_2$, $node_3$ and $node_4$) that will need to play each other at least once. So in this case we would have a minimum of three rounds of the 2-player game in order to have a main

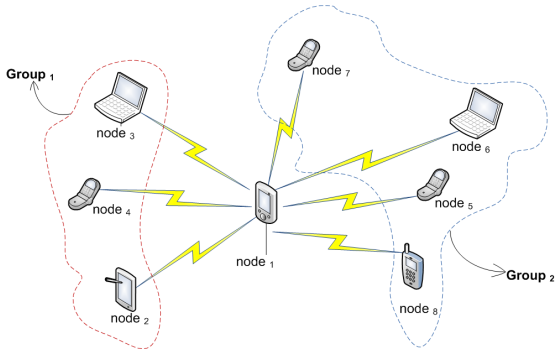


Fig. 1: Divide Nodes into Groups

game winner ($node_2$ vs $node_3$, $node_2$ vs $node_4$ and $node_3$ vs $node_4$).

At this point we make two assumptions. Let's assume there is a time interval $t$ in which each group can accept new players. When $node_1$ receives the first request from $node_2$ in time $t_1$, and a second request from $node_3$ in time $t_2$ then both nodes will be members of the same group if $t_1 < t_2 < t$. Furthermore, each group will have a maximum allowable number of 20 nodes, otherwise the time required to process the last player's request will be large and inefficient for ad hoc nodes. Thus, a group is considered closed and will not accept any new nodes if the maximum number of players has been reached, or if the time interval $t <= 1sec$ has been exceeded.

The advantages of dividing nodes into groups and why this technique is adding a security level to a network are discussed in Section IV.

### B. Multiplayer Game

The game is formed as follows: Each player starts with a number of credits which is the same for all the nodes of the group. Additionally, a cost has been assigned to each player. The costs are not identical but follow the sequence of the request submittals. For example, the node that first made a request will have a smaller cost than the node after it, and so on. During each sub-game, every node has three possible results: win, loose or draw. When the first main game finishes, and all nodes have played with each other, the winner is the one that has the maximum credits overall. In case there are two or more nodes with the same number of credits the node with the smaller cost is served (first in first out).

Supposing that $node_1$ has already created the groups, the next step is to generate five random numbers. A prime number $q$, the credits $x$ which is a random positive integer, generator $g$ of $\mathbb{Z}_q$, the dividend $d$ where $d \ll x$ and the priority $p_i$ that each node have in order to be served and is a positive integer from which each node will calculate its costs. Then, $node_1$ is using the above numbers to generate $y$ from equation (1):

$$y = f(x),\ where\ f(x) = g^x mod\ q \tag{1}$$

$Node_1$ responds to the requests of nodes from $Group_1$ by sending them the parameters $y, g, q, d$ and a value $p_i$ which is

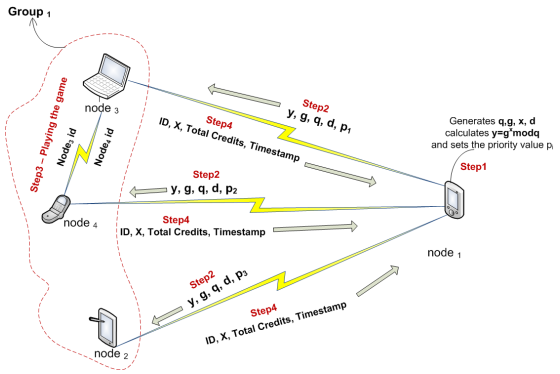



Fig. 2: Sending Parameters to Players / Nodes in order to start playing the game

different for each node and corresponds to the priority with which each node is served (the first requestor will have the highest priority among others and so on). Next, each node is responsible for finding the credits $x$. After that, each node has to calculate the cost of play $c$, where $c > x$, which is different for each node in the group. As we can see in Figure 2, $node_1$ accepts requests from three clients of $Group_1$, each of whom has a value $p_i$ to be served.

The cost for each node is given by equation (2):

$$c_i = x + p_i, \tag{2}$$

where $p_i < p_{i+1} < p_{i+2} < \cdots$.

The reason that $p_i < p_{i+1} < p_{i+2} < \cdots$ must hold, is because we want nodes to have the opportunity to be served in the same sequence that they made their requests. This will give an advantage to the players that requested a service first.

At this point, players have all the necessary information in order to start playing. The payoff matrix is shown in Table I where $S$ represents a strategy, $x$ the credits, $c$ the cost that each player receives for a subgame.

TABLE I: Payoff Matrix

| | $S1_{Player1}$ | $S2_{Player1}$ |
|---|---|---|
| $S1_{Player2}$ | $\frac{x-c_{p1}}{d}, \frac{x-c_{p2}}{d}$ | $0, x$ |
| $S2_{Player2}$ | $x, 0$ | $\frac{x}{d}, \frac{x}{d}$ |

where

$$\frac{x-c_{p_i}}{d} < 0 < \frac{x}{d} < x$$

The players can either have an offensive strategy ($S_1$), or a defensive strategy ($S_2$). When both players play defensively, they will equally share the winning credits, but when one of them meets an offensive strategy, the other leaves all the winning credits to the offensive player. In particular, in any conflict situation, the defensive player will never win credits, but it will never loose credits, even when confronting an offensive strategy and therefore the interactions are neutral with respect to the defensive strategy. Furthermore, in the case were both players play an offensive strategy, they both loose some credits but the looser of the game is the one with the higher cost value.

An important issue in our multiplayer game is the number of players, that each group should contain. If this number is large, then the procedure could be time consuming, which makes it inefficient for ad hoc networks. Therefore, it is important to know the **minimum** number of games that have to be played in a group, in order for each node to be served.

**Lemma 1.** *For every $n$ players in a group, the minimum number of games that have to be played in order for every node to be served is*[1]*: $G(n) = \frac{n^3 - n}{6}$.*

In the case where a group contains three players, from Lemma 1 we see that at least $G(3) = 4$ sub-games has to be played. For four players the sub-games will be at least $G(4) = 10$ and for five players $G(5) = 20$ sub-games.

Thus it is clear that the maximum number of players in each group should be quiet small, otherwise the time that would be needed for the last player to be served would be unaffordable for the nodes of an ad hoc network. For example if a group contains 50 players it will need at least 20,825 sub-games in order for every node's request to be processed.

Considering that each request creates different groups, we achieve two things:

1) Possible malicious nodes are divided into different groups, and thus they are unable to cooperate with each other in order to harm the network.
2) Every node in the network has an amount of resources, which can be translated into the quantity of requests that can serve. By dividing nodes into groups and processing requests from only one node at a time (from each group) we extend nodes lifetime. This means that the whole network can keep working properly much longer, especially if we take into consideration that new nodes are introduced into the network and can also process new requests.

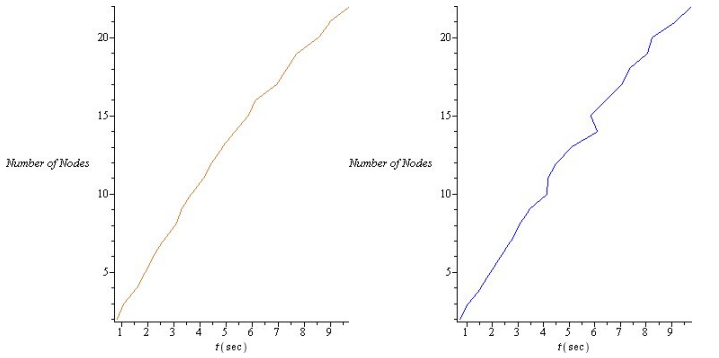
## IV. Experimental Results and Analysis

In order to measure the effectiveness of our solution, we have used the Smart Dust simulator written in Java. Our testbed consists of a laptop computer with Intel Core Duo CPU P7450 at 2.13GHz, 6.00 GB of RAM running Windows 7 64-bit. Our experiments where implemented in a medium scale ad hoc network.

Each node that makes a request, is responsible for finding the solution of the DLP. To do so, we used Baby-Step-Giant-Step algorithm (BSGS) to brute brute equation (1). BSGS is a generic algorithm that falls into the class of square root algorithms and its complexity is $O(\sqrt{n})$.

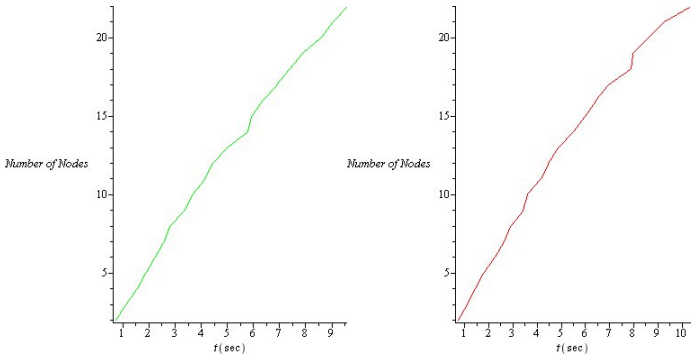
Figures 3 and 4 depicts the time that needed for the last node of a group to be served. From these experiments we found that

[1]The proof is omitted due to space constrains.




(a) (b)

Fig. 3: Time for the last node of the network to be served.




(a) (b)

Fig. 4: Time for the last node of the network to be served.

the optimal number of nodes for each group should be from 2 to 22. Regarding to the strategies, we tested our game for the following four cases:

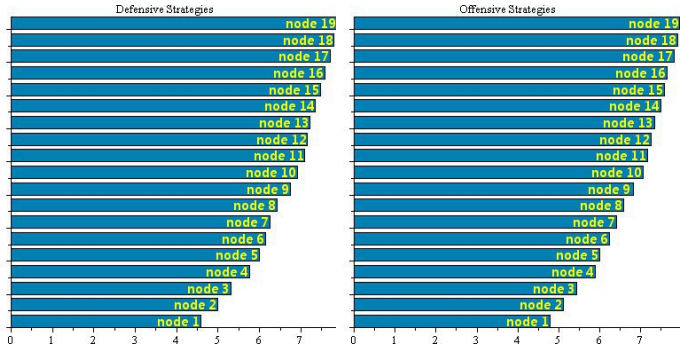
- All nodes choosing their strategy randomly (figure 3a)
- In every sub game one node would play $S_1$ and the other $S_2$ (figure 3b)
- All nodes playing defensively (figure 4a)
- All nodes playing offensively (figure 4b)

In order for all of our measures to be in the same level, we did not generate randomly the parameters for equation (1), instead we used the DLP with an average level of difficulty. Table II shows some representative results from our experiments. As we can see, in the case where a group contains 15 nodes, the average time for the last node to be served is $5.9sec$ which is an affordable time for devices that are part of an ad hoc environment. Especially if we take into consideration the fact that before that time, 14 other nodes have been already served.

Another very important issue is the order in which every node is served. The ideal scenario is that each node would be served based on the priority $p_i$ value. In figure 5 we can see that when all nodes play the same strategy ($S_1$ or $S_2$), they follow a first come first served approach. Here the first node had its request served 3 seconds earlier than the last

TABLE II: Results from figures 3, 4

| Nodes | Random | Equally Split | Defensive | Offensive |
|---|---|---|---|---|
| 5 | 1.93 | 1.92 | 1.88 | 1.79 |
| 10 | 3.69 | 4.14 | 3.66 | 3.59 |
| 15 | 5.86 | 5.84 | 5.93 | 5.98 |
| 20 | 8.55 | 8.25 | 8.59 | 8.61 |




(a) (b)

Fig. 5: Time needed for each node to be served

node. In figure 6 the strategy that each node plays, is chosen randomly. As we can see, nodes are not served according to their priority since the first node is actually served $17^th$. For that reason each node has the ability to play in its best interest in order not to loose the priority.

As mentioned earlier, every node in a group must solve the discrete logarithm problem in order to find the credits $x$ of the game. In the case where one or more nodes act maliciously and guess $x$, $node_1$ has the ability to verify whether the node with a specific $id$ has correctly calculated $x$. So $node_1$ will refuse to serve the winner of that group and will wait for the next winner of this group.

The overhead of the network, as the number of nodes of a

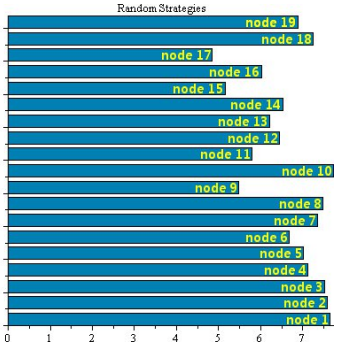



Fig. 6: Time needed for each node to be served when they randomly choose their strategy

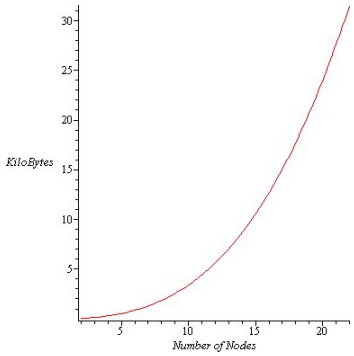



Fig. 7: Network Overhead

group increasing, is shown in figure 7 and the formula form which we can calculate the overhead as the number of nodes in a group is increasing, is shown from Lemma 2.

**Lemma 2.** *For $n$ nodes in a group, the total network overhead in bytes, is given by the following formula*[2]*: $V_{Total}(n) = \frac{4}{3} \cdot (2 \cdot n^3 + 6 \cdot n^2 + n)$*

Security requirements in ad hoc networks must take into account the countermeasures against node misbehavior and DDoS attacks. In our approach, the concept of payoffs and optimum strategies is applied so that potential malicious nodes will be responsible of executing expensive computations. Our game theoretic approach is based on the concept that we want to force a possible malicious node in an ad hoc network to perform heavy computations before it can use the resources of another node. It is vital to keep as many of the receiving node's resources free to ensure the long lasting operation of the ad hoc network.

In DDoS attacks a single node can send multiple requests to $node_1$. For that reason we assume that $node_1$ will only accept up to two active requests from a single node, and for the second request this particular node will become an element of another group as well. With this method, an attacker will be limited in the sense that he will be unable to make many simultaneous attacks on a single node.

## V. Conclusions

Unlike networks using dedicated nodes to support basic functions like packet forwarding, routing, and network management, in ad hoc networks these functions are carried out by all available nodes. Nodes communicate with each other using wireless radios and operate by following a peer-to-peer network model [5]. Since ad hoc networks can be deployed rapidly, sensitive applications raise important security issues. Security requirements in ad hoc networks are different from those of fixed networks. While the security requirements are the common ones, namely availability, confidentiality, integrity, authentication and non-repudiation, they are considered differently for ad hoc networks due to system constraints in mobile devices (i.e. low power microprocessor, small memory and bandwidth, short battery life) and frequent network topology changes [5]. Ad hoc networks have no standard infrastructure and their topology is changing dynamically and all the network operation is based on cooperation of nodes within the neighborhood. Ad hoc network misbehavior may be imposed by malicious nodes, each of which intentionally aims at harming the network operation.

In this paper we proposed a new technique that combines game theory and client puzzles making ad-hoc networks resilient to DDoS attacks. Our approach use client puzzles in combination with game theory to force possible malicious nodes to execute many computations before they get access to the resources of another node. Additionally, dividing malicious nodes into separate groups decreases their ability to cooperate with one another, further weakening an attack. Finally, through our experiments we have shown that the total overhead of the proposed multiplayer game as well as the resources that each node will have to spend are efficient for ad hoc networks.

[2]The proof is omitted due to space constrains.


[1] H. Narasimhan, and V. Varadarajan, and C. P. Rangan, *Game Theoretic Resistance to Denial of Service Attacks Using Hidden Difficulty Puzzles*. In IPSEC, pages 359-376, 2010.

[2] A. Agah and S. K. Das, *Preventing DoS Attacks in Wireless Sensor Networks: A Repeated Game Theory Approach*. In I. J. Network Security, pages 145-153, 2007.

[3] Wu, Xiaoxin, and Yau, and David K. Y., *Mitigating denial-of-service attacks in MANET by distributed packet filtering: a game-theoretic approach*. In ASIACCS '07: Proceedings of the 2nd ACM symposium on Information, computer and communications security, pages 365-367, New York, USA, 2007. ACM.

[4] N. Komninos, *A Transport Layer Security Protocol for Hybrid Networks*. Book Chapter in Wireless Communications Research Trends, pages 267-285, January 2007, Nova Science Publishers Inc.

[5] N. Komninos, and D. Vergados, and C. Douligeris, *Layered Security Design for Mobile Ad-Hoc Networks*. In Journal in Computers & Security, Elsevier, Volume 25, Issue 2, pages 121-130, Elsevier.

[6] J. H. Cheon, and N. Hopper, and Y. Kim, and I. Osipkov, *Timed-Release and Key-Insulated Public Key Encryption*. In Financial Cryptography and Data Security, FC 2006, volume 4107 of LNCS, pages 191-205, 2006, Springer.

[7] J. Cathalo, B. Libert and J.-J. Quisquater, *Efficient and Non-interactive Timed-Release Encryption*. In ICICS 2005, LNCS 3783, pages 291-303, Springer-Verlag, 2005.

[8] C. Douligeris and A. Mitrokotsa, *DDoS attacks and defense mechanisms: classification and state-of-the-art*. In Computer Networks 44(5), pages 643-666, Texas, USA, 2004.

[9] B. R. Waters, A. Juels and E. W. Felten, *New client puzzle outsourcing techniques for dos resistance*. In CCS '04: Proceedings of the 11th ACM conference on Computer and communications security, pages 246256, New York, NY, USA, ACM, May 2004.

[10] B. Bencsath, I. Vajda and L. Buttyan, *A Game Based Analysis of the Client Puzzle Approach to Defend Against DoS Attacks*. In Proceedings of the 2003 International Conference on Software, Telecommunications and Computer Networks, pages 763-767, 2003.

[11] J. Vijayan, *Denial - of - Service Attacks Still a Threat*. In Computer World, April 8, 2002.

[12] C. Meadows, *A cost-based framework for analysis of denial of service in networks*. In J. Comput. Secur., pages 143-164, Amsterdam, The Netherlands, 2001, IOS Press.

[13] T. Aura, P. Nikander and J. Leiwo, *Dos-resistant authentication with client puzzles*. In Revised Papers from the 8th International Workshop on Security Protocols, pages 170-177, 2001, Springer-Verlag.

[14] C. Dwork and M. Naor, *Pricing via processing or combatting junk mail*. In Ernest F. Brickell, editor, Proc. CRYPTO 92, pages 139-147, May 1992, Springer-Verlag.