A Peer-to-Peer Architecture to collaboratively Propagate and Traceback DDoS Attack information using DST

D. Thamizh Selvam
Department of Computer Science
School of Engineering & Technology
Pondicherry University, India.

P.S. Vinayagam
P. Syam Kumar
Department of Computer Science
School of Engineering & Technology
Pondicherry University, India.

Dr. R. Subramanian
Department of Computer Science
School of Engineering & Technologyz
Pondicherry University, India

ABSTRACT
Distributed Denial of Service attacks has become prevalent in the context of ever growing Internet. Numerous attacks have taken place in the past and numerous solutions have been suggested. Intrusion detection and filtering are necessary mechanisms to combat against these attacks and secure networks. However, the existing detection techniques for DDoS attacks have their entities work in isolation. In this paper, we propose an efficient and distributed collaborative architecture that allows the placement and the cooperation of the defense entities to better address the main security challenges. The use of Distributed Spanning Trees (DST) algorithm controls the damage caused by Distributed Denial of Service attacks by using propagation and traceback mechanism. Simulations show that DST-based tracing behave better than randomly generated graphs and trees as it generates less messages to query all computers while avoiding the tree bottlenecks.

General Terms
Peer-to-Peer Network, Security.

Keywords
DDoS, DST, P2P Overlay, Propagation, Traceback, IDS.

1. INTRODUCTION
Denial of Service (DoS) occurs when legitimate users are prevented from getting access to shared resources or services. If DoS is originated from a large number of distributed attackers, the event is termed a Distributed Denial-of-Service (DDoS) attack. Distributed Denial of Service (DDoS) attack has been identified as one of the most serious problems on the Internet. It has been identified as the most urgent Internet security concern by Arbor Network's 2008 survey [1]. The roots of the DDoS problem lie in the design of the Internet architecture [2] itself: (i) In order to gain the most of the Internet, its network link resources are shared, but there is no enforcement of fair sharing; (ii) The network core processes high volumes of traffic so core components can do very little processing per packet, thus all computations (e.g. those ensuring security) must be performed at the edge; (iii) The Internet's constituent networks are managed by different authorities, and this heterogeneity makes widespread deployment of DDoS defense mechanisms difficult.

The Peer-to-Peer (P2P) model offers a promise to exploit all the resources of a vast numbers of hosts. The distribution of data storage among several nodes allows this model, in comparison with a centralized scheme, to reduce the possibility of storage overload at some points and to have a single point of failure. The use of a P2P model can also be justified by its robustness, high scalability, and fast resource lookup. Many solutions have been proposed which can be classified into structured and non-structured solutions regarding resource localization methods. Protocols developed on structured P2P networks have recently gained popularity for the implementation of large-scale distributed systems. The proposed architecture uses a self-organized structure, called "Distributed Spanning Tree (DST)" [3].

In this paper, we propose scalable defense architecture based on overlay routing against DDoS attacks in consideration of the capacity of each node, providing speedy notification of attack detection on converged network and detouring of the normal packets before the fundamental exclusion of attack agents. Our simulation results show that our architecture provides the speedy notification and decreases the damage of normal traffic even in the form of converged DDoS attacks [4].

This mechanism pursues the following design goals and we can confirm the performance by simulation.

- Speedy notification of attack detection to nodes of other networks in the converged attack case, as well as a highest defense system.
- Detouring the normal traffic before the fundamental process of attack agents, thus decreasing the damage.
- Scalable and dynamic defense structure of overlay in consideration of the capacity of each node.

The rest of this paper is structured as follows. Section II gives a general overview of DDoS attacks and its related works. We describe the architecture and the functionality of the proposed approach in Section III. The Section IV elaborates our propagation and traceback algorithm. We evaluate the performance of DST in Section V. Finally, we conclude and introduce our future work in Section VI.

2. RELATED WORKS
Some recent studies developed by Moore et al. [5] estimated the DoS activity by a backscatter method on packet traces and showed that more than 2000 DDoS attacks are launched every week. The problem is that it becomes very easy for any Internet user to create disruptions using limited resources. Moreover the attack damages are increased by the distributed computing techniques. Many existing systems are successful in one aspect of defense, but none of them offers a comprehensive complete solution. In such context, there is a tremendous need for distributed and cooperative defense architecture in order to avoid the threat of DDoS attacks.

When a DDoS defense system is deployed at the victim network it is difficult, due to the aggregation, to identify attack packets at the ingress [6] of the targeted network although this deployment can facilitate the observation of the victim. This is why it's important to push the detection upstream to the ingress points of the service provider. This implicitly implies the distribution of the detection scheme among several locations, which raises the problem of how to coordinate the different detection systems. Some systems like DIDS [7] or NSTAT [8] have been proposed to work in a distributed environment. In these propositions the audit of data collected is done in several points of the network and the analysis is executed by a central location. With CSM [9] and AAFID [10], the usage of distributed analysis agents is very relative.

Current IDS do not offer a global solution that satisfies users' need in coping with the evolution of the attack types. The deployment of DDoS defense system at the attack source network cannot permit the collection of necessary information about the attack traffic and so, detection at this level will not be efficient. On the other hand, attack flows can be stopped before they enter the Internet core. And this is why response can be more effective at the attack source level. The mechanism for identifying the sources of attacks and for limiting the rate of malicious flows is commonly known as traceback mechanism. Current DDoS solutions are many, ranging from host-based solutions to network and infrastructure solutions. Our architecture is basically proposed for DoS detection and IP traceback solutions. For DDoS detection we have 2 main groups:

- The signature-based detection schemes that search for a known identity or signature for each attack event [11]. This category is not efficient against new types of attacks.
- Anomaly based detection schemes [12] that detect anomalies caused by DDoS attacks. In this case a model must be established according to standard protocol normal system activities.

In general the intrusion detection entities are deployed on hosts or routers and the agent is deployed at a single point or network-based where the agents cooperate either in a centralized [8] or a decentralized [9] manner. A decentralized approach is more scalable but needs more complex communication schemes to effectively share the information between the detection entities.

For IP traceback schemes we have 2 main classes:

- Backtracking techniques [13] that work in a hop by hop manner to construct a summary of routed flow. In this class we have the proactive measures category where the flow is generated independently from the presence of the attacks and the reactive measures where the summary is generated on demand.
- Flow extension techniques bring additional information to flows during their travel. We have the in-band messaging (packet marking), that can be probabilistic [14] or deterministic [15], and the out-of-band messaging that sends the traceback data in separated packets.

Some proposed DoS solutions have a global scope. They start from the victim side where detection is most suitable and propagate attack alerts through intermediate networks in order to deploy filtering rules as near as possible to attack source networks. Mahajan et al. [16] proposes pushback (also implemented in [17]) as a complete method to deal with DDoS.

In this proposition, DDoS are treated as a congestion-control problem. A new functionality is added to each router to detect and preferentially drop packets that probably belong to an attack. Upstream routers are also notified to drop such packets (hence the term pushback) in order to have router's resources used to route legitimate traffic. This presents an interesting approach, however, router vendors did not show any interest in implementing this scheme. A draft was proposed at IETF which expired in 2002. Canonico et al. [18] use the same concept of pushback in defense "propagation". They propose ASSYST, a distributed system, in which network routers cooperate in order to react to DDoS attacks in a flexible and dynamic fashion. DefCOM [19] is a distributed collaborative framework to defend against flooding DDoS attack. As a global architecture, it combines the advantages of source-end, victim-end and core defenses and allows the existing heterogeneous defense systems to cooperate through an overlay. Nodes collaborate by exchanging messages, marking packets for high or low priority handling, and prioritizing marked traffic. However, it was not clearly described how to authenticate and establish economic cooperative relationship across different management domains. Radwane et al. [20] proposed a collaborative peer-to-peer architecture which uses DHT (Distributed Hash Tables). DHT which is a dynamic overlay structure are theoretically scalable and resistant to failures. But, the index implementation of a DHT lays on a global view of the system and determines the data placement. Dahan et al. [3] proposed DST (Distributed Spanning Tree) structure, a self-organizing structure, where every peer is independent and provides its own data or resources from its computer to the network.

3. PROPOSED ARCHITECTURE

The objective is to propose a global architecture that permits an efficient Intrusion Detection System where participants can exchange information following a P2P model, and thus providing services to a traceback application that strengthens the network security against DDoS attacks; or at least permits a fast and effective reaction to this kind of threats.

The solution proposed is designed to elaborate the defense against these large scale attacks by the correlation of the suspicious evidence provided and stored by the architecture entities from different geographical locations. Each participant gains a global view of the intrusion activity through this collaboration. To perform this objective we take into consideration some requirements in terms of performance and deployment. In fact, the processing and the bandwidth used, as well as the storage must be minimized and conceptual security mechanism must be added to permit the access control for each entity in the architecture.

A. Problem Statement

A DDoS attack is usually characterized by a high traffic rate, an IP spoofing and several paths are taken to reach the victim. These elements are particular in a distributed attack. Our system proposes that the detection relies on the most frequently routed destination IP addresses during a short time period (Δ) on different network points. Each IDS deployed on the network and especially in the ingress of a network, will analyze the traffic that

passes by the router which this IDS is added too. In fact, the same equipment can also do both jobs.

B. The Distributed Spanning Tree Structure

The development and widespread usage of peer-to-peer networks is mainly due to their efficient overlay routing and the location function, especially in global storage utilities and applications. We propose to apply an algorithm that defines a deterministic way to efficiently store and share the collected data between the nodes. These nodes are able to form a distributed and decentralized network with a dynamic adaptation without affecting the whole functioning of the network. In order to realize the flexible and balanced collection of information among the nodes, we required an efficient approach that can scale to a large number of peers exchanging many control messages. The choice of a DST algorithm satisfies these requirements. These solutions present a relevant robustness since the global functioning of the network is totally independent of each application node.

A DST can be described at three different levels as shown in Fig.1. The *logical level* is an abstract vision of the DST. At this level, tree nodes—that are groups of computers—are linked together by abstract links. Then comes *the interconnection level* that implements internodes links with TCP/IP links. Finally, there is the *topological level* which describes how the TCP/IP links map on a real network [3].

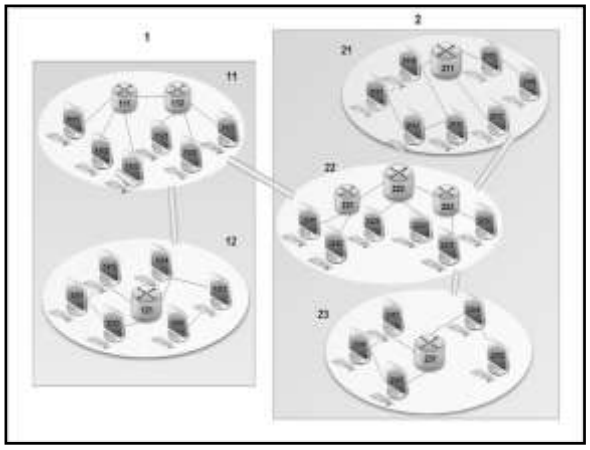

Fig.1 DST Structure

C. Description of the architecture

Our principal contribution is the proposition of the architecture that serves as a model to implement a security system against DDoS attacks. This model targets detection systems with the possibility to place some applications at the very top level that can use the collected data of the detection system. The innovation in this case is the addition of a new level between the application one and the detection system which is the use of the DST. This new level permits to index the information and to distribute it among the participants instead of implementing a central collection entity.

The Fig. 2 describes each level of the architecture. We present it with an abstraction degree that permits us to put both independent and specific functions on each level. In fact, a node can contain one or more levels. We will detail this later by giving examples.

The first level is the closest to the physical network. We called it the *Network Level*. In this level, equipment belongs to the underlay network. To be more specific, an entity in this level can for example be an IP router with the basic routing and addressing functionalities.

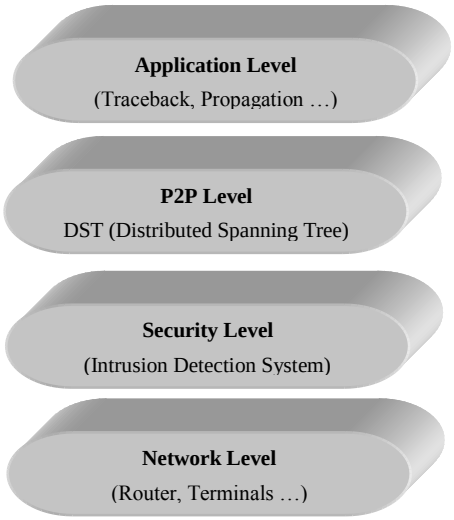

Fig.2 Architecture Levels

The second level is the *Security Level*. In this level we can classify our detection system entities. The implementation of all solutions in terms of detection system can be done here. This level's functions are the ones of detection. When the analyzed traffic in this level is detected as an attack, an alert is triggered and a primitive is sent to the upper level. This level will react to the alert accordingly. We can note that a detection system can be integrated to the equipment and the concerned entity will be represented by the 2 first levels of the architecture.

The third level of the model is the *P2P Level* which includes the DST proposed in our architecture to index and to distribute the information among the nodes. This level receives the information collected concerning the analyzed traffic from the *Security Level*. When an alert is sent by the lower level, which means that an attack is detected, the P2P level treats the received data and indexes the information on the specific DST node (identified by a *nodeId*) depending on the *objected* calculated. We also see the possibility in this case, to integrate this level module to the equipment that already has the *Network* and the *Security Levels.*

The last level is the *Application Level*. This last level is general in the description for our architecture. It can implement all possible management systems that use the indexed information on attacks by the DST level to react. We propose in our case to add a traceback application solution to integrate more complete defense architecture than only a detection system. This traceback module is a part of the future works since the three first levels of the architecture were implemented as we will see in the next sections.

By removing any central analysis entity in the architecture we propose a fully distributed solution. But in choosing this method we must be sure that the collected data are correlated to ensure a better detection of DDoS attacks and also a reaction to them.

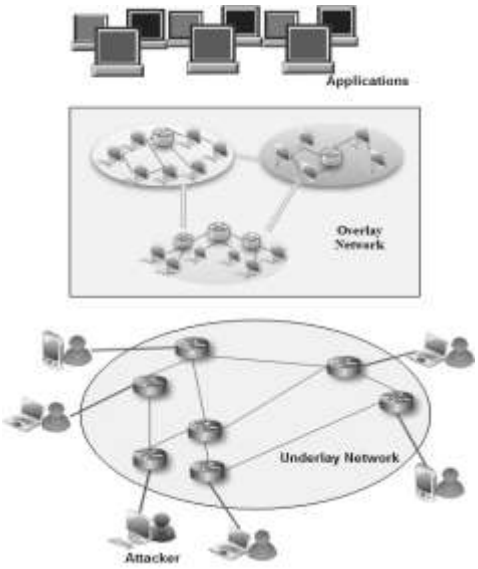

Fig.3 Architecture Entities

Indeed we propose that some applications retrieve the traffic information to analyze it deciding what to do. Fig.3 gives the details architecture with the different entities.

4. PROPAGATION AND TRACEBACK ALGORITHM

A. Propagation Algorithm

The aim of the propagation algorithm is to efficiently send messages of to every computer that is part of a DST about the DDoS attack. This algorithm is the simplest one and is very similar to the classical tree parallel traversal. The root node initiates the traversal by sending a message to all its children. Then, recursively, when a non-leaf node receives a message, it forwards it to its children.

Algorithm 1. Propagation algorithm

procedure Propagate (*msg*)

Propagate_Sub(*h*, *msg*)

end procedure

procedure Propagate_Sub (*s*, *msg*)

if *s* = 0 **then** // The message is sent to a leaf

process *msg* locally . // End of the Propagation

else // The message is sent to a non-leaf node

for all *child* ∈ routing_ table[*s*] do

child → Propagate_Sub(*s* - 1, *msg*)

end for

end if

end procedure

The DST propagation algorithm is presented as Algorithm.1. This algorithm uses two procedures. *Propagate_Sub* is a recursive procedure which propagates the message of an attack and *Propagate* is the procedure which initializes the propagation.

The procedure *Propagate_aux* takes two parameters: ***msg***, the propagated message, and ***s***, the level of the called node in the tree (DST). Because non-leaf nodes are distributed over their descendants, every computer acts as a leaf, as a node of *stage* 1, …. and as the root node. The parameter **s** tells the computer which node receives the message. If ***s*** = ***h***, the computer must act as the root node. If ***s*** = ***h*** - 1, it must act as the child of the root node. If **s** = 0, the computer must act as a leaf and it does not forward the message further.

If **s** ≠ 0, the computer acts as a *stage* **s** forwarding node. So, this node forwards the message to its children. To do it, the computer takes the list of computers that represent the children of its stage **s** node. For each of them, it asks him to forward the message as a stage ***s*** - 1 node, child of the stage **s** node.

The propagation of attack is initialized by the *Propagate* procedure. This procedure must be called by a computer that is part of the DST. By calling *Propagate_sub*(*h*, *msg*), it asks to himself to act as the root node to forward the message ***msg***. Because a computer always uses itself as the representative of the nodes that contain it, every computer receives only one distant message. We can conclude that the number of distant messages of a broadcast on an n-node DST is **n** - 1 because the computer that initializes a broadcast does not receive any distant messages. So, the complexity order of the broadcast is ***O(n)*** messages. Traces of Algorithm.1 show that the algorithm runs ***h*** + 1 recursions. So, the algorithm runs in $h + 1 \leq \log_a(|C|) + 2$ steps. We can conclude that the algorithm time complexity is ***O(log(n))*** time units.

B. Traceback Algorithm

The *Traceback* algorithm aims at querying a number of computers which grows exponentially like a TTL-based graph flooding algorithm. This algorithm uses a propagate- like algorithm to query a subtree of stage 1, then a subtree of stage 2, and so on, until the DST is completely flooded or until the query is positively answered about an DDoS Attack.

Algorithm 2. Traceback algorithm

procedure Traceback (*tfs*)

$\mathcal{AL} \leftarrow$ Traceback_sub (0, *tfs*)

$\mathcal{TA} \leftarrow 1$

while $\mathcal{TA} \leq h \wedge \mathcal{AL} \equiv \emptyset$ **do**

temp ← ø

for all *child* ∈ routing_table[$\mathcal{TA}$] do

if $\mathcal{TA} \neq$ *self* **then**

temp[*child*] ← *child* → Traceback_sub ($\mathcal{TA}$, *tfs*)

end if

end for

for all $temp\mathcal{AL} \in$ *temp* **do** // Joins found resources

$\mathcal{AL} \leftarrow \mathcal{AL} \cup temp\mathcal{AL}$

end for

$\mathcal{TA} \leftarrow \mathcal{TA} + 1$

end while

return $\mathcal{AL}$

end procedure

procedure Traceback_sub ($\mathcal{TA}$, *tfs*)

if $\mathcal{TA} = 0$ **then**

$\mathcal{AL} \leftarrow$ List of Attackers.

else

temp $\leftarrow$ø

for all *child* $\in$ routing_table[$\mathcal{TA}$] **do**

temp[*child*] $\leftarrow$ *child* $\leftarrow$ Traceback_sub ($\mathcal{TA}$- 1, *tfs*)

end for

$\mathcal{AL} \leftarrow$ø // List of found resources

for all $temp\mathcal{AL} \in$ *temp* **do** // Join found resources

$\mathcal{AL} \leftarrow \mathcal{AL} \cup temp\mathcal{AL}$

end for

end if

return $\mathcal{AL}$

end procedure

The *Traceback* algorithm uses two procedures. *Traceback_Sub* is a recursive procedure which propagates requests in subtrees and *Traceback* is the procedure which controls the search. To look for a resource matching a query, you need to call the *Traceback* procedure and pass your query as a parameter.

The *Traceback_Sub* procedure takes two parameters to broadcast the request: $\mathcal{TA}$ the height of the subtree and *tfs,* traffic flow signature description. If $\mathcal{TA}$ equals 0, we query a leaf. In this case, the computer gathers the list of resources that match the query and returns the list to the caller.

If $\mathcal{TA}$ is not equal to 0, the computer propagates the query to its subtree of height $\mathcal{TA}$. - 1 To do it, for each child of its node of level $\mathcal{TA}$, the computer sends a message to its representative. This message asks the computer to gather the list of resources that match the query in its subtree of height $\mathcal{TA}$ - 1. Then, the computer waits for the answers of the queried subtrees and merges the lists of matching resources. Finally, it returns the merged list to its caller.

The *Traceback* procedure takes one parameter (*tfs*) which describes the Traffic Flow Signature. The procedure starts by searching locally a resource that matches the *tfs* and stores the list of found resources in the list $\mathcal{AL}$. Then, starting with a height of 1, the procedure queries subtrees of increasing height until querying the whole tree or until finding a resource.

To query a subtree of height $\mathcal{TA}$, the Search procedure sends a message to a computer of each child of its node of level $\mathcal{TA}$ but one. We remind that a computer always chooses itself to represent nodes that contains the computer.

So, with the test $\mathcal{TA} \neq$ *Self*, we avoid to query the subtree that contains the computer because this subtree is the subtree that has been queried during the previous iteration. Once the queried subtrees return their lists of found resources, the computer merges the lists and prepares a new iteration in the case where no resources is found. At the end, it returns to the client the list of found resources that match the query.

By following traces of the *Traceback* algorithm, it is easy to notice that each computer is only queried once and that only 2(***n*** - 1) messages are used to query the ***n*** computers of the DST. So, the complexity order of the search algorithm is O(***n***) messages. Because propagations are parallel, only 2***s*** time units are needed to query a subtree of height s. Thus, to query an **h** stages DST, we need 2.(1 + 2 + …. + h) = h (h + 1) time units. We can conclude that the search algorithm has a time complexity order of $O((\log(\mathbf{n}))^2)$ time units.

5. PERFORMANCE STUDY

The architecture was implemented and its performance was proved to be good. The aim of these simulations is to compare the behaviors and the performances of tracing algorithms on top of three overlay network topologies: tree, pseudorandom graph, and DST. Our comparison is limited to these two topologies because; they are the most commonly used for self-organized networks whereas static or index-oriented topologies cannot be used in this context.

To simulate the execution of tracing algorithms, we use the algorithm for the tree and the graph topologies: the initiator peer contacts its neighbors and waits for a reply; if no resource is found, then the initiator asks its neighbors to contact their neighbors and waits for their replies, and so on, until the end of the tree or the graph is reached. Hundred different types of resources are available, and every computer has a probability of 10 percent to own a resource of each type. Each search request stops either when it finds a node with the requested resources or when the whole structure is traversed.

About the overlay topologies characteristics, trees are bidirectional and their arity is 5. Graphs are also bidirectional, connected, and the degree of each node is 5. Finally, the DST is made in a way that each node has five children. These degrees were chosen because they show the best performances in our simulations. More precisely, we run some tests at various scales to find out these optimal degrees. Then, we use them for all the simulations by considering that these degrees are always optimal in our experiments. However, these values depend on the links throughput and the probability to find a service. Changing one of these parameters implies that the chosen degrees would no longer be optimal.

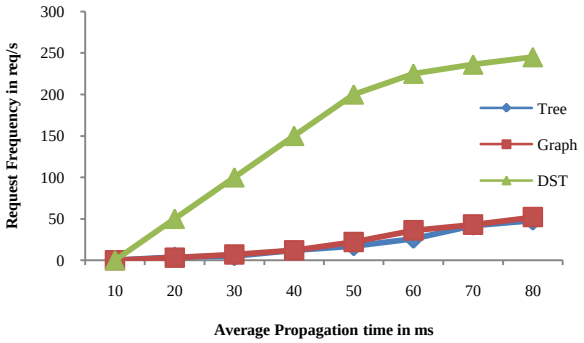

Fig.4 Performances for networks of 10 peers

The simulations results for the 10 peer overlay networks are displayed in Fig.4. The simulations show that the average time needed to process a request depends on the request arrival rate. This is an ordinary observation. When the number of initiated requests increases, the system becomes more and more loaded and messages spend more time in a waiting queue before being sent.

When the number of requests that enter the system becomes higher that the number of requests that leave it, the system becomes saturated. This saturation is easily identifiable for the DST in Fig. 4: the average time needed to process a request increases slowly for frequencies from 1 req.s^{-1} to a frequency of 150 req.s^{-1}; but it increases very quickly for frequencies greater than 200 req.s^{-1}. On the other hand, the graph and the tree become very quickly saturated.

The DST has the best behavior for these simulations. Because a DST is a tree, a search request needs only **2.n** messages to query n peers, which is less than the graph. But, because it distributes the load of father nodes between its children, it does not suffer from the tree bottlenecks.

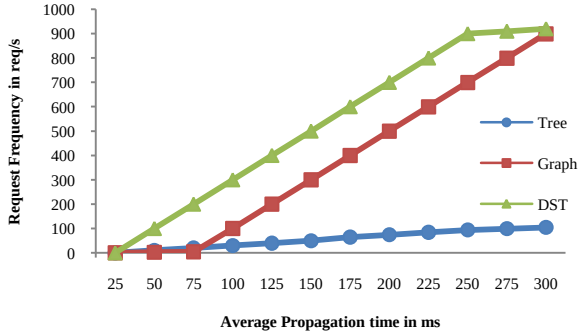

Fig.5 Performances for networks of 100 peers

Fig.5 presents the simulation results for 100 peers. Like before, the three topologies saturate when the query arrival rate becomes too high. From the simulations, it is clear that the tree has the worst performances in term of supported load. A frequency of 100 req.s^{-1} is enough to overload trees, while graphs and DST start to be overloaded for a frequency of 700 req.s^{-1}. It is revealed that the probability for a peer to be contacted at least twice is less than 1/3. If every peer of a graph is contacted only once, then only **2.n** messages are needed to query n peers and the average number of messages is optimal, like for the tree. So, the main result of this experience is that the tree topologies are not efficient for the leaves, compared to graphs or DST, as their first request will only query one node: their parent.

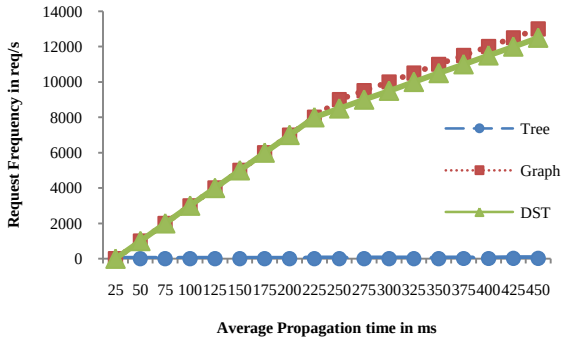

***Fig.6 Performances for networks of 1000* peers**

Fig.6 gives the simulated performances for overlay networks of 1,000 peers. A load of 300 req.s^{-1} is enough to saturate the tree. Graphs and DST start to be overloaded around 8,000 req.s^{-1}. Before being overloaded, the average search time of DST and graphs increases slowly when the load is increasing.

DST's performances are better than the graph's ones for two reasons:

1. A DST sends fewer messages as it uses the spanning tree for its traversal algorithm. The number of sent messages depends on the number of nodes, while it depends on the number of links for the graph.

2. A DST distributes fairly its load between computers as the spanning trees used by the nodes are distributed across the network nodes. Thus, no bottleneck is generated compared to the tree topology.

6. CONCLUSION AND FUTURE WORK

We proposed in this paper a modular Peer-to-Peer architecture to collaboratively manage propagation and traceback of DDoS attacks relying on the performance and scalability of DST structure. This structure provides characteristics similar to trees while avoiding the bottlenecks generated by the tree root. It behaves better than randomly generated graphs as it needs only two times the number of nodes messages to query all nodes, and thus, significantly improves traversal algorithm efficiency. Simulations validate that the DST improves the performances of traceback algorithms, whatever the size of the network, from 10 to several thousand nodes. The DST also provides good results when propagating a DDoS attack signal on the overall network. The main drawback of this structure is its cost of construction but this cost is spread out over the lifetime of the DST as the structure is incrementally generated by nodes that join (or leave) the overlay network.

This model was designed with the aim of proposing the integration of an intrusion detection system that can bring an attacks' information collect service to some applications like traceback. The load-balancing is ensured by the consistent updation of DST and every node keeps information on specific victims. However, the scope of this paper is the description of the architecture and the way that the lower levels offer services to the application level. The management and the traceback application that can retrieve and analyze the data sent by the lower levels to react to the possible attacks must be more specifically specified, and this is a principal issue for future work.

7. REFERENCES

[1] "Arbor networks worldwide infrastructure security report",2008. *http://www.arbornetworks.com*

[2] O. Demir, "A survey of network denial of service attacks and countermeasures," City University of New York, ComputerScience Department, Tech. Rep., 2009.

[3] Sylvain Dahan, Laurent Philippe, and Jean-Marc Nicod, "The Distributed Spanning Tree Structure", *IEEE Trans. Parallel and Distributed Systems*, vol.20, no.12, pp.1738-1751, December 2009.

[4] Mihui Kim, Inshil Doh and Kijoon Chae, "Defense Mechanism using Overlay against DDoS Attacks on Converged Networks", *ICACT2007*, pp. 1539-1543, February 2007.

[5] D. Moore, G. Voelker, and S. Savage, "Inferring Internet Denial of Service Activity". In *Proceedings of the 2001 USENIX Security Symposium, Washington D.C.*, August 2001.

[6] P. Ferguson, D. Senie, "Network ingress filtering: defeating denial of service attacks which employ IP source address spoofing". In *IETF, RFC 2267*, January 1998.

[7] SR. Snapp et al. "DIDS (Distributed Intrusion Detection System)- motivation architecture and an early prototype". In *Proceedings of the 14th national computer security conference, Washington DC*, October 1999.

[8] RA. Kemmerer. "NSTAT: a model-based real-time network intrusion detection system". In *Technical Report TRCS97-18, Reliable Software Group, Department of Computer Science, University of California at Santa Barbara*, 1997.

[9] G. B. White, E. A. Fisch, U. W. Pooch, "Cooperating security managers: A peer-based intrusion detection system". *IEEE Network*, 10(1):20-23, January / February 1996.

[10] E. H. Spafford, and D. Zamboni, "Intrusion detection using autonomous agents", In *Computer Networks, vol. 34, No. 4, pp. 547-570, 2000*.

[11] "Snort: The Open Source Network Intrusion Detection System", www.snort.org.

[12] T.M. Gil, M. Poleto, "MULTOPS: a data-structure for bandwidth attack detection" In *Proceedings of 10th Usenix Security Symposium,Washington, DC, pp. 23–38*, August 2001.

[13] H. Hazeyama, Y. Kadobayashi, D. Miyamoto, and M. Oe. "An autonomous architecture for inter-domain Traceback across the borders of network operation". In *Proceedings of 11th IEEE Symposium on Computers and Communications (ISCC '06), pages 378–385*, June 2006.

[14] J. Liu, Z. Lee, and Y. Chung, "Efficient dynamic probabilistic packet marking for IP traceback". *In Proceedingd of the 11th International Conf. Networks (ICON 2003), Sydney, Australia, , pp.475-480*, September 2003.

[15] A. Belenky, N. Ansarin, "Tracing multiple attackers with deterministic packet marking (DPM)". In *Proceedings of IEEE PacRim vol. 1, pp. 49-52*, August 2003,

[16] R. Mahajan, S. M. Bellovin, S. Floyd, J. Ioannidis, V. Paxson, and S. Shenker, "Controlling high bandwidth aggregates". In *The network. SIGCOMM Comput. Commun. Rev., 32(3) :62_73*, 2002.

[17] J. Ioannidis and S. M. Bellovin, "Implementing pushback : Routerbased defense against ddos attacks". In *NDSS. The Internet Society*, 2002.

[18] L. Peluso, D. Cotroneo, S. P. Romano, G. Ventre, "ASSYST: an Active Security System against DoS attacks". Technical Report. Dept.of Computer Sciences, University of Napoli, Italy, April 2001.

[19] J.Mirkovic, M.Robinson, P.Reiher, and G.Oikonomou, "Distributed Defense Against DDOS Attacks". University of Delaware CIS Department Technical Report CIS-TR-2005-02, 2005.

[20] Radwane Saad, Farid Nait-Abdesselam and Ahmed Serhrouchni, "A Collaborative Peer-to-peer Architecture to Defend Against DDoS Attacks", Prod. 33rd IEEE Conference LCN2008, pp.427-434, 2008.