A Hybrid Layered Multiagent Architecture with Low Cost and Low Response Time Communication Protocol for Network Intrusion Detection Systems

Varsha Sainani and Mei-Ling Shyu
Department of Electrical and Computer Engineering, University of Miami
Coral Gables, FL 33124, USA
v.sainani@umiami.edu, shyu@miami.edu

Abstract

Applying multiagent technology to the management of network security is a challenging task since it requires the management on different time instances and has many interactions. This paper aims at utilizing potential benefits from applying distributed multiagent technology to network security. To facilitate information exchange between different agents in our proposed hybrid layered multiagent architecture, a low cost and low response time agent communication protocol is developed to tackle the issues typically associated with a distributed multiagent system, such as poor system performance, excessive processing power requirement, and long delays. The bandwidth and response time performance of the proposed end-to-end system is investigated through the simulation of the proposed agent communication protocol on our private LAN testbed called Hierarchical Agent Network for Intrusion Detection Systems (HAN-IDS). The simulation results show that our proposed system is efficient and extensible since it consumes negligible bandwidth with low cost and low response time.

Keywords: *Agent Communication, Agent Architecture, KQML, IDS*

I. Introduction

The wide introduction of Web-based applications leads to the interconnection of almost all the computers in the world in a global network that facilitates a number of transactions. With the increase in the users and uses of this global network, there has been an increase in the types of attacks which also bring serious damage to people, corporation, and the whole society. Such situations create the need for efficient and fast intrusion detection systems (IDS), to safeguard the network systems and crucial information.

Recently, multiagent systems have become popular since they promise to provide high-level interactions in intricate applications for modern computing and information processing systems. As agents have to operate and exist in an environment which is both computational and physical, there are situations where an agent can operate by itself but the increasing interconnection and networking of computers is making such situations rare, and in usual state of affairs the agent interacts with other agents [17]. There have been a number of multiagent architectures developed in the literature, such as the *Logic Based Architecture*, *Reactive Architecture*, *Belief Desire Intention Architecture (BDI)*, and *Layered Architecture* [16]. For our purpose, we adopt the layered architecture since it is an approach to design subsystems handling many subproblems of an application.

A number of different layered architectures exist, including *Touring Machines*, *InteR RaP*, *ATLANTIS*, etc. [15]. Layered architectures can be further broadly categorized into the following two types depending on the direction of the flow control: *Horizontal* and *Vertical*. In *Horizontal* architectures, each interaction focuses on assigning central control and each layer performs independently. In other words, each layer comprises of agents. Each layer gets an input and gives an output. If there are 'n' layers and each layer has possible 'm' actions, there are m^n interactions. This type of architecture has conceptual simplicity, but at the same time it lacks coherence and needs a mediator function to decide which layer has the control. The *Touring Machine* is an example of horizontal architecture which consists of reactive, planning, and modeling layers along with a control subsystem. The use of such central control reduces the autonomy.

On the other hand, in the *Vertical* architectures, the inputs and outputs are handled by only one layer at a time and the control flows through each layer until the output is achieved. Under the same definitions of 'm' and 'n' as mentioned previously, it has the complexity of $m^2(n-1)$, which is less than the horizontal approaches. The vertical

architectures include a natural decomposition of the functionality and pragmatic approach to solutions. The *InteR Rap* system is a good example for vertical layered architectures [17]. It constitutes of cooperation, planning, and behavior layers. However, this type of architectures is not fault tolerant as the failure of one layer will pull down the entire system. In addition, how to handle the interaction between layers is also a challenging issue.

Reactive architecture approach was developed by *Rodney Brooks*. It stresses on two characteristics. The first one is that an agent's decision-making is realized through a set of task accomplishing behaviors. Each of these behavior modules is intended to achieve some particular task. The second characteristic is that many behaviors can 'fire' simultaneously. There are many existing studies on reactive architectures such as *Brooks' behavior languages*, *Agre's* and *Chapman's PENGI* system, *Situated Automata*, and *Pattie Maes's Agent Network Architecture*. *Brooks'* reactive architecture brings simplicity, economy, computational tractability, robustness against failure, and elegance. A major selling point of purely reactive systems is that the overall behavior emerges from the interaction of the component behaviors when an agent is placed in its environment. However, the term 'emerges' suggests that the relationship between individual behaviors, environment, and the overall behavior is not easy to realize [3]. This gives rise to the concept of creating hybrid architectures and brings out the best from both layered and reactive architectures. There exist hybrid architectures such as *Procedural Reasoning System (PRS)*, *IRMA*, and *GRATE*, to name a few. Almost all of these architectures constitute of a number of components that are put together in order to solve tasks. Based on our best understanding, the layered architecture approach has yet not been employed for hybrid architectures [1].

In this paper, we propose a hybrid layered multiagent architecture that combines layered and reactive architectures in a hierarchy of three different layers having agents at each layer, namely the *Host*, *Classification*, and *Manager*. This is an attempt to depict the best of both types of layered architectures and its combination with the reactive architecture, which overcomes the disadvantages of the horizontal and vertical layered architectures, making our proposed architecture logically and semantically complete as well as stable. As mentioned earlier, one of the disadvantages of the layered architectures is that the handling of the interactions between layers. However, the cooperation between agents in a multiagent system is a must. To address this issue, we ornament our architecture with a communication protocol that facilitates several desirable functionalities for a multiagent architecture, such as low response time, low cost, and reliability. The *Knowledge Query and Manipulation Language (KQML)* agent communication language *(ACL)* is adopted for our communication protocol [5]. Please note that the main focus of this paper is the proposed communication protocol that enables the information exchanges between/among the agents in the proposed hybrid layered multiagent architecture.

The remaining part of this paper is organized as follows. Section II presents the design of a novel hybrid layered multiagent based IDS architecture. Our proposed efficient and adaptive protocol is discussed in Section III. Section IV describes our experimental setup, and the results of the performance analysis is given in Section V. Finally, Section VI concludes the paper.

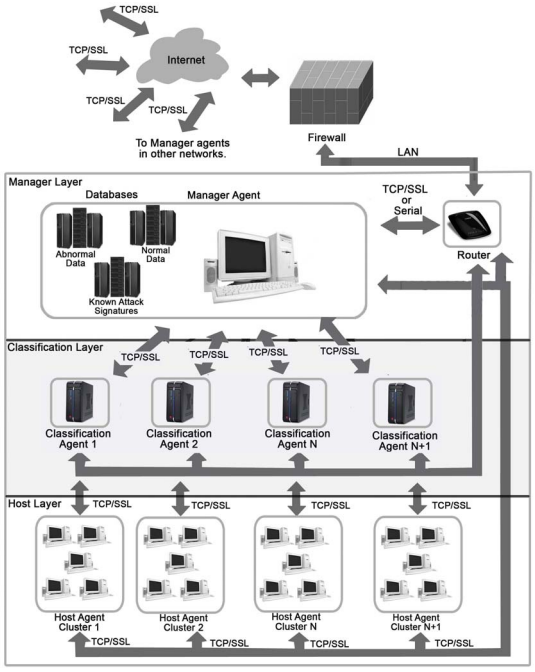

Figure 1. The *HAN-IDS* network testbed with the proposed hybrid layered multiagent architecture

II. Hybrid Layered Multiagent based IDS Architecture

Figure 1 presents our proposed hybrid layered multiagent based IDS architecture which constitutes three layers. Each layer comprises of deliberative agents.

A. Host Layer

This layer consists of the end-user machines. They act as the first layer of our architecture, in other words, they act as

the host agents. The Host Agents are the background daemons which inspect incoming network connections. These machines are a set of workstations which form the network. The agents running on these machines collect information about the network connections in their hosts and classify these connections using the *Principal Component Classifier (PCC)* [12] into either '*normal*' or '*abnormal*' connections. Each Host Agent is connected to the next layer of agents called the Classification Agents, to whom the Host Agents report the connections found to be '*abnormal*' by *PCC*.

The Host Agents are mainly concerned with capturing network traffic, detecting abnormal activities occurring in these incoming connections and properly responding to them. Virtually, every machine in a network can be considered as a Host Agent. Since every connection in a host, whether '*normal*' or '*abnormal*', is analyzed for abnormality, the Host Agent can keep the information of a few normal connection instances and pass it along with the abnormal connection instances to the Manager layer to be saved in a database for the purpose of re-training the classifier at a later time [20].

B. Classification Layer

In the Classification Layer, the Classification Agents attend to the concern of a cluster of Host Agents and their suspicion of a possible attack. The Classification Agents classify the abnormal connection instances found in their hosts into known attack types. This task is performed by the Classification Agent by deploying a misuse detection algorithm called the Collateral Representative Subspace Projection Modeling *(C-RSPM)* [13]. This is important as the attack type will determine the proper response of the IDS to the intrusion.

The agents in this layer are configured to run on dedicated machines capable of delivering the processing power required to handle all classification requests of their Host Agents. In addition, they have to generate 'policies' [14][19] upon the instances which are identified as intrusive. Research effort is currently being invested on the development of a suitable data mining strategy to enable the Classification Agent to form policies. Once the policy is created, it is communicated to the upper layer agent called the Manager Agent. All the Classification Agents present in the network add their policies to the policy repository present in the "Database System" component in the Manager Layer.

C. Manager Layer

Manager layer as the name suggests consists of a Manager Agent. This layer in terms of contemporary agent models is the same as the *planning layer* [17]. This agent manages the entire system, performing several support tasks for the system. It is responsible for the tasks such as assigning specific Host Agents to the specific Classification Agents, assisting the Classification Agents in managing their host clusters and tasks related to them, as well as managing the routers and firewalls in the network.

The main task performed by the Manager Agent is to take the policies from the Classification Agents throughout the network and derive 'rules' [19] based on those. Once the rules are deduced, the Manager Agent conveys them down the hierarchy to the Classification Agents and then the Classification Agents forward them to the Host Agents in their clusters. On receiving the rules from the Classification Agent, the Host Agent who initiated the request implements the rule. This functionality is important as the Classification Agents can prevent or lessen the effects of a possible attack by managing resources in their nodes that they expect to be affected by the incoming attack, such as bandwidth, communication ports, and connection authorization.

III. The Communication Protocol

As mentioned earlier, the main focus of this paper lies in the design and development of the proposed communication protocol. Software agents suggest a paradigm for software development that emphasizes autonomy, adaptability, and cooperation, both at the design time and runtime. This approach seems appealing in a world of distributed, heterogeneous systems [2][7]. An agent communication language (ACL) that allows the agents to interact while hiding the details of their internal workings will result in an agent community with the capability of tackling the problems that no individual agent could. The Knowledge Query and Manipulation Language (KQML) is one of the most commonly used agent communication languages due to its versatility and generality of purposes [5]. KQML supports multiagent communication through an extensible set of reserved primitives called *performatives* that represent communicative acts. In our proposed architecture, KQML was adopted as the default ACL [17].

A simple and manageable communication scheme that utilizes a discrete number of KQML performatives is designed to accommodate the goals of all the agents. We used *TCP/IP Secure Socket Layer (SSL)* for the implementation of secure communication channel among the agents to provide total privacy and authentication capabilities. The adoption of cryptographic communication services is an important step toward making the distributed multiagent IDS architecture immune against attacks that can exploit the relatively simple agent communication scheme [11]. Once the main channel and important control functions are set for execution, there comes a point in the system where the logs are to be transferred to the Manager Agent so that they

can be saved in the database for future reference and training of the *CRSPM* classifier [13]. This requires the Host Agents to transfer the logs to the Manager Agent as it holds 'abnormal' and 'normal' connection instances. If this data is transferred via the main channel, it could add delays to the ongoing threat detection which is definitely not desired. Therefore, we open another channel to transfer these logs and close it as soon as the transfer is done [8]. This concurrency helps avoid any interference or delays with the main channel functions, though the messages for the establishment of these channels are also written in KQML and are conversed through the main channel [6].

Here we describe how different communication messages can be deployed at different levels by the use of the KQML performatives.

RECOMMEND-ONE: As soon as a Host Agent comes online, it sends this KQML message to the Manager Agent as it is aware of the Manager Agent's IP address, requesting for the Classification Agent's IP address to which it should connect. The Manager Agent keeps the count of the number of Host Agents registered with each Classification Agent (called host agent count) to ensure that the load is symmetrically distributed for all agents. Following is an example of the RECOMMEND-ONE message, where the Classification Agent's IP is requested in the 'Content' by asking 'CA_IP' which represents the Classification Agent's IP.
(RECOMMEND-ONE
Sender IP: 10.0.0.3
Sender Port: 1000
Ontology: Agent Communication
Language: KQML
Content: CA_IP)

TELL: The Manager Agent uses this performative for replying to RECOMMEND-ONE from the Host Agent, informing it to which Classification Agent it should connect, in the content of the TELL message. For example, assuming that the Manager Agent is recommending the Classification Agent with IP '10.0.0.5' and port '2000'.
(TELL
Sender IP: 10.0.0.6
Sender Port: 3000
Ontology: Agent Communication
Language: KQML
Content: 10.0.0.5:2000)

REGISTER: The Classification Agents use this performative to register with the Manager Agent as soon as they come online. This message also carries the host agent count (i.e., the number of Host Agents registered with that Classification Agent) which may or may not be equal to zero initially. The Host Agents also use this performative to register with the Classification Agent upon receiving the IP address from the Manager Agent. In the following example, the 'Content' field of the message indicates that the Host Agent wants to register with Classification Agent by stating 'HA_reg'.
(REGISTER
Sender IP: 10.0.0.3
Sender Port: 1000
Ontology: Agent Communication
Language: KQML
Content: HA_reg)

UNREGISTER: As the name suggests, this performative does exactly the opposite of the 'REGISTER' performative. Whenever a Host Agent wants to quit from the network, it sends this UNREGISTER message to the Classification Agent and in turn the Classification Agent updates its host agent count by removing this Host Agent from its cluster. The same procedure follows when the Classification Agent desires to quit. The 'HA_unreg' value in the 'Content' field in the listed example indicates that the Host Agent wants to disconnect.
(UNREGISTER
Sender IP: 10.0.0.14
Sender Port: 1014
Ontology: Agent Communication
Language: KQML
Content: HA_unreg)

FORWARD: Every time a Host Agent registers or unregisters with the Classification Agent, the Classification Agent needs to update the host agent count and informs the Manager Agent about this change to ensure the consistency of the information. This performative is employed by the Classification Agent to inform the Manager Agent about the new host agent count. For example, the 'Content' field of the following message indicates that the new host agent count is '101' for this Classification Agent.
(FORWARD
Sender IP: 10.0.0.5
Sender Port: 2000
Ontology: Agent Communication
Language: KQML
Content: HA_count: 101)

EVALUATE: This performative is employed by the Host Agents for requesting the Classification Agent to evaluate an 'abnormal' instance into the corresponding attack type. It conveys the information of that particular instance which was classified as 'abnormal' to the Classification Agent in this message. The message shown below is a message from the Host Agent to the Classification Agent requesting for the evaluation of the abnormal instance 10.0.0.51 connecting at

port 1345. In real case scenarios, it could be expected that the 'Content' field of the EVALUATE message will carry a number of attributes of the instance.
(EVALUATE
Sender IP: 10.0.0.3
Sender Port: 1000
Ontology: Agent Communication
Language: KQML
Content: Abnormal: 10.0.0.51: 1345)

REPORT: On receiving the EVALUATE message from the Host Agent, the Classification Agent classifies the instance into a known attack type. It also looks at other environmental situations such as different hosts complaining about the same instance which the Classification Agent diagnosed into different attack types, and then it tries to capture the nature of the attacker and derives a 'policy'. The Classification Agent employs this performative to report this policy to the Manager Agent. The 'Content' field of the following message shows how a policy is represented by the Classification Agent, which it concluded 10.0.0.51 is to be blocked.
(REPORT
Sender IP: 10.0.0.5
Sender Port: 2000
Ontology: Agent Communication
Language: KQML
Content: Policy: Block: 10.0.0.51)

BROADCAST: This performative is utilized by the Manager Agent to broadcast rules to all the Classification Agents which it derived based upon the policies received from different Classification Agents. The Classification Agents also employ the same performative to broadcast the rule from the Manager agent to the Host Agents. The example below shows the rule 'Disconnect' for IP 10.0.0.51 as derived by the Manager Agent.
(BROADCAST
Sender IP: 10.0.0.6
Sender Port: 3000
Ontology: Agent Communication
Language: KQML
Content: Rule: Disconnect: 10.0.0.51)

REQUEST: This performative is used by the Host Agent to request the Manager Agent to open another channel which can be used to transfer the logs of both 'normal' and 'abnormal' instances from the Host Agent to the Manager Agent. The Manager Agent can then log these into the database for the re-training of the classifier later on. The 'Content' field of the REQUEST message requests for opening the log channel by sending 'log-ch_open'.
(REQUEST
Sender IP: 10.0.0.3
Sender Port: 1000
Ontology: Agent Communication
Language: KQML
Content: log-ch_open)

REPLY: The Manager Agent uses this performative to reply to the Host Agent's request for another channel, indicating whether it accepts the request or not. The Manager Agent indicates 'log-ch_accept' and 'log-ch_reject' respectively in the 'Content' field of this message.
(REPLY
Sender IP: 10.0.0.6
Sender Port: 3000
Ontology: Agent Communication
Language: KQML
Content: log-ch_accept)

READY: Upon receiving the REPLY message from the Manager Agent, the Host Agent checks if the Manager Agent accepts the request. If it accepts, then the Host Agent sends a READY message notifying that it is ready for transfer. This is indicated by 'log-ch_ready' as the 'Content' field value of the message as shown below.
(READY
Sender IP: 10.0.0.3
Sender Port: 1000
Ontology: Agent Communication
Language: KQML
Content: log-ch_ready)

RESPONSE: In reply to the READY message of the Host Agent, the Manager Agent sends the RESPONSE message to indicate that it is ready to start the transfer. As soon as the Host Agent receives a positive response in this message from the Manager Agent, it opens another channel and sends all the logs. As can be seen from the following message, the 'Content' field has the value 'log-ch_start'.
(RESPONSE
Sender IP: 10.0.0.6
Sender Port: 3000
Ontology: Agent Communication
Language: KQML
Content: log-ch_start)

IV. Experimental Setup

In order to assess the overall performance of our proposed hybrid layered distributed multiagent based intrusion detection system together with the proposed communication protocol in a realistic scenario, a prototype of the proposed architecture was implemented using *Java RMI* [4] package in the particular **lipe RMI** [9]. We have conducted evaluations in terms of scalability-related criteria such as

network bandwidth and system response time for the analysis of our proposed protocol. Figure 1 shows our *HAN-IDS* network testbed, where the different types of agents were placed in mutually exclusive machines within the testbed, in a manner such that any communication among the agents could only be realized through the generation of network traffic rather than local traffic within the same machine. Here we ensure that every conversation between the agents would generate measurable network traffic and yield realistic scalability results. These machines were connected to each other via a router. The Host and Classification agents were simulated on Linux OS and the Manager Agent was executed on Windows OS. The Host Agents were all assigned to different interfaces with different IPs on Linux OS by the use of the 'ifconfig' command and specifying virtual interfaces with virtual IPs. This allows us to simulate up to 100 host agents on the same machine.

ifconfig eth0:1 10.0.0.1
ifconfig eth0:2 10.0.0.2
ifconfig eth0:3 10.0.0.3
etc.

The same procedure was employed for the Classification Agents. Five Classification Agents were simulated on another machine running Linux OS. Next, having decided on the location of the agents within the testbed, a realistic experimental scenario was devised to capture the effects that the communication between an increasing number of Classification and Host Agents would have on the traffic requirements, and consequently on the scalability performance of the proposed architecture. In summary, our experiments consists of instantiating an increasing number of the classification and host agents to simulate an increasing IDS network, in a manner that would reflect the performance of the proposed architecture as it was expanded from a small scale to a large scale. Experiments were conducted with one Manager Agent and the maximum of five Classification Agents, with up to 100 Host Agents for each Classification Agent.

There were a total of 5*10 = 50 experiments, for 1 to 5 classification agents each with 10, 20, ..., 100 host agents. Therefore, the maximum number of agents is 506, simulating a realistic network of 500 Hosts, 5 Classification Agents, and one Manager Agent connecting to the IDS network. For all these experiments, two features were observed, namely the average bandwidth (in Mbps) and the response time (i.e., the time when an abnormal instance is detected by a Host Agent to the time when the last BROADCAST message is received by the Host Agent). Both of these features were analyzed by using the packet capture tool called *Wireshark Network Protocol Analyzer* [18]. As most of the network traffic flows are between the Host and Classification Agents, Wireshark was executed in all of the Host and Classification Agents. Also, not much changes were identified in the figures on the Manager Machine as the communication between the Manager and Host or the Manager and Classification Agents is almost constant. The results which we captured from the network simulation was visualized using MATLAB(2007) [10].

V. Analysis and Results

A. Experimental Methodology

There are several reasons to believe that our proposed architecture and the communication protocol improve intrusion detection performance. It is known that two of the most important elements of network performance are bandwidth and latency. This analysis uses bandwidth and latency as the metrics for evaluating the performance of the system. Bandwidth is the transmission capacity of the network, usually measured in bits per second. Frequency is the number of bytes transferred between agents and the maximum rate at which information can be exchanged in a network. Network latency is the amount of time it takes for a packet to travel from the source to the destination. We use latency to describe the amount of time it takes from the moment an attack takes place till it gets resolved. That is, we consider the round trip time of our communication protocol including the transfer and processing times of the messages. Latency is of interest in systems with real-time constraints.

Since we adopt the layered architecture, hierarchical heuristics reduce the time complexity in logarithmic manner. In a layered structure, it allows means-ends heuristic which reduces the complexity dramatically. As explained in [17], if it is assumed that a hierarchy divides the problem of size 'n' into problems each of size 'k', yielding n/k sub-problems, each of which requires $f(k)$ time to solve. These solutions are fed to the next level up in the hierarchy such that 'k' is given to each of agents in this level. Each of these n/k^2 agents has to synthesize 'k' results, again requiring $f(k)$ time. This aggregation process continues up the hierarchy, such that at the next to the topmost level, n/k^{l-1} agents combine 'k' results from below in the hierarchy with 'l' levels. The topmost agent then combines these n/k^{l-1} results together, requiring $f(n/k^{l-1})$ time. The total expenditure is as follows.

$$f(n/k^{l-1}) + (n/k^{l-1} \times f(k)) + (n/k^{l-2} \times f(k)) + \ldots + (n/k \times f(k))$$

Since 'k' is a constant, and we can choose $l = log_k n$, the equation can be reduced to $O([(k^l - 1)/(k - 1)]f(k))$ which can be simplified to $O(n)$. More importantly, if each level of the hierarchy has the agents that solve their subproblems in parallel, then the time needed below the top of

the hierarchy is simply $f(k)$ for each level, so $(l-1)f(k)$. This is added to the top agent's calculation $f(n/k^{l-1})$.

Again since k is constant and $l = log_k n$, this reduces to $O(log_k n)$. This means that through decomposition and parallel problem solving, the exponential problem can be reduced to logarithmic time complexity [17]. This is one of the main advantages of our proposed architecture. In a multiagent system, latency is particularly important in real-time domains where small changes in latency may mean the difference between success and failure.

B. Results

In ideal case scenarios, when the network related delays and other network constraints are ignored, its performance such as awareness time and scalability should be linear and dependent only on the number of Classification Agents in the network and the number of Host Agents in each of their clusters. The time taken for the Manager agent to send rules to the Classification Agents is usually less than what it will take for the Classification Agents to warn the Host Agents in their cluster since there are a large number of Host Agents per Classification Agent.

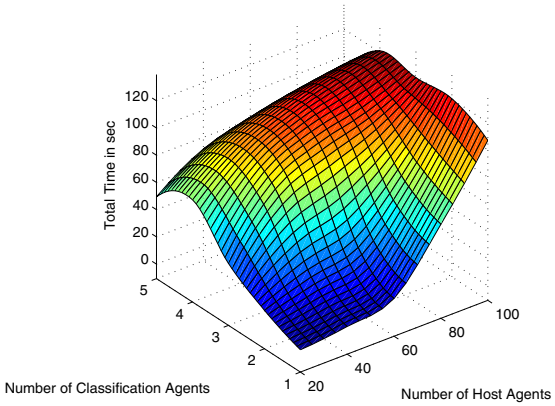

Figure 2. Plot of the *Response Time* for the proposed communication protocol

Figure 2 shows the time plot that can be used to resemble the plane with the number of agents. Linearity of the plot indicates that the system scales linearly in terms of the attack response time, which is a desirable feature. From this figure, it can be seen that the higher response time is about 95 seconds corresponding to the use of 100 host agents with 5 classification agents. These are favorable and substantial results as it demonstrates that our proposed architecture together with the proposed communication protocol produce promising results.

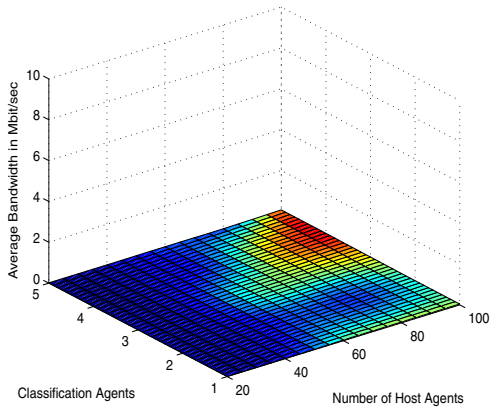

Figure 3. Plot of *Communication Bandwidth* used by the communication protocol vs bandwidth consumption in a regular network with 10 Mbits/sec

Next, we consider bandwidth performance of our proposed protocol. Let's first look at it from the scenario of a regular average sized LAN where the least required bandwidth is at least 10 Mbits/sec. Figure 3 shows the experimental results for bandwidth consumption of our protocol, where the values for bandwidth consumed for almost all the experiments were less than 1 Mbit/sec, which is almost negligible. As can be seen from this figure, the required bandwidth touches the ground as it consumes negligible bandwidth.

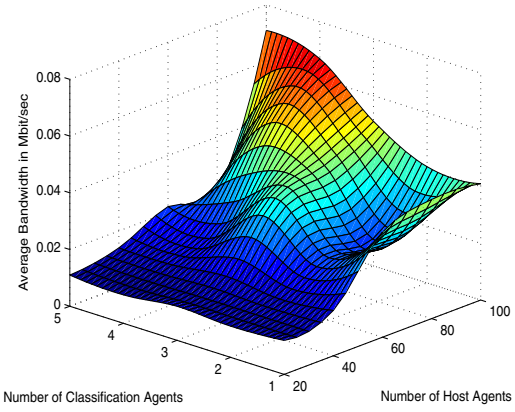

Figure 4. Plot of a zoomed view of Figure 3

Finally, we look at the zoomed view of the bandwidth performance in Figure 4. Ideally, the bandwidth usage of the communication protocol should linearly increase with the number of agents connecting to the network and this is

well depicted by the performance of our protocol. Here, the maximum bandwidth consumed is 0.69 Mbits/sec which is absolutely small. Even in the order of magnitude of 506 agents, the system is still efficient in terms of its bandwidth performance. This demonstrates that our proposed communication protocol enables information exchange with low overhead and system scalability. In addition, the steady curve in the bandwidth consumption shown in Figure 4 is probably the result of the retransmission of lost packets, delays, noise, etc. in the network, which can be considered as the variability in network conditions. Since our proposed communication protocol consumes very low bandwidth, such network conditions do not significantly derogate its performance. As we know, nowadays the major concern of a networked system is the limitation on the bandwidth, and thus maintaining a low consumption of bandwidth is a desirable feature. Furthermore, for most of the existing IDS architectures, when the number of attacks increases, the system becomes slow and finally freezes. In our proposed architecture, the performance of the system will not deteriorate too much with the increase in the number of attacks, which is justified by its low bandwidth consumption behavior.

VI. Conclusion

In this paper, a novel hybrid layered distributed multiagent based IDS architecture is presented. It incorporates desirable features of multiagent design methodology with an agent communication protocol which is capable of exchanging information among different layers with minimum overhead. Several experiments were conducted to show the effective communication between the layers and the scalability feature in our proposed architecture. Theoretical and experimental analysis demonstrate the improved communication efficiency and network intrusion problem handling in terms of low cost and low response time for service, even with a large number of agents in the system.

References

[1] A. Assal and V. Groza. Agent based resource management for smart robotic sensors. In *CCECE-2004*, volume 4, pages 2239–2242, Niagara Falls, 2004.

[2] J. Cao, L. Zhang, J. Yang, and S. Das. A reliable mobile agent communication protocol. In *Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS'04)*, pages 468–475, Tokyo, Japan, 2004.

[3] H. Chalupsky, Y. Gil, C. Knoblock, K. Lerman, J. Oh, D. Pynadath, T. Russ, and M. Tambe. Electric Elves: Agent technology for supporting human organizations,. *AI Magazine*, 23(2):11–24, 2002.

[4] J. A. D. Framework. *Available at http://jade.tilab.com/*, 2008.

[5] T. Froese. Steps toward the evolution of communication in a multi-agent system. In *Symposium for Cybernetics Annual Reserch Projects, SCARP'03*, University of Reading, UK, 2003.

[6] A. Hayzelden, J. Bigham, S. Poslad, P. Buckle, and E. Mamdani. Communication systems driven by software agent technology. *Journal of Network and Systems Management*, 8(3):321–347, 2000.

[7] S. Khasteh, S. Shouraki, R. Halavati, and E. Khameneh. Evolution of a communication protocol between a group of intelligent agents. In *World Automation Congress (WAC)*, pages 1–6, Budapest, Hungary, 2006.

[8] M. Koes, I. Nourbaksh, and K. Sycara. Communication efficiency in multi-agent systems. In *Proceedings of ICRA 2004*, pages 2129–2134, Barcelona, Spain, 2004.

[9] lipeRMI. *Available at http://lipermi.sourceforge.net/*, 2006.

[10] Mathworks.2007.MATLAB. *Available at http://www.mathworks.com/matlabcentral/*, 2007.

[11] J. Park, H. Park, and W. Kwon. A real time communication protocol with contention-resolving algorithm for programmable controllers. In *20th International Conference on Industrial Electronics, Control and Instrumentation (IECON'94)*, pages 1159–1164, Brazil, 1994.

[12] M.-L. Shyu, S.-C. Chen, K. Sarinnapakorn, and L. Chang. Principal component-based anomaly detection scheme. *Foundations and Novel Approaches in Data Mining*, 9:311–329, 2006.

[13] M.-L. Shyu, T. Quirino, Z. Xie, S.-C. Chen, and L. Chang. Network intrusion detection through adaptive sub-eigenspace modelling in multiagent systems. *ACM Transactions on Autonomous and adaptive Systems*, 2(3):1–37, 2007.

[14] M.-L. Shyu, K. Sarinnapakorn, I. Kuruppu-Appuhamilage, S.-C. Chen, L. Chang, and T. Goldring. Handling nominal features in anomaly intrusion detection problems. In *the 15th International Workshop on Research Issues on Data Engineering (RIDE-SDMA'2005), in conjunction with ICDE 2005*, pages 55–62, National Center of Sciences, Tokyo, Japan, April 2005.

[15] K. Sycara, M. Paolucci, M. V. Velsen, and J. Giampapa. The RETSINA MAS infrastructure. *Autonomous Agents and Mulit-Agent Systems*, 7(1-2):29–48, 2003.

[16] K. Vaidehi and B. Ramamurthy. Distributed hybrid agent based intrusion detection and real time response system. In *In Proceedings of the First International Conference on Broadband Networks*, pages 739–741, 2004.

[17] G. Weiss. *Multiagent Systems*. The MIT Press, 2nd edition, 2001.

[18] Wireshark. *Available at http://www.wireshark.org/download.html/*, 2008.

[19] I. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementation*. Morgan Kaufmann Publishers, San Mateo, CA, 2nd edition, 2000.

[20] Z. Xie, T. Quirino, and M.-L. Shyu. A distributed agent-based approach to intrusion detection using the lightweight pcc anomaly detection classifier. In *Proceedings of the IEEE International Conference on Sensor Networks, Ubiqquitous, and Trustworthy Computing (SUTC'06)*, pages 446–453, Taichung, Taiwan, 2006.