JOURNAL OF APPLIED
COMPUTER SCIENCE
Vol. 23 No. 2 (2015), pp. 39-54

Multi-Agent Systems Registration and Maintenance of Address Mapping without Agent Self-Registration

Artur Opalinski[1]

[1]*Gdansk University of Technology*
Faculty of Electrical and Control Engineering
Narutowicza 11/12, 80-233 Gdansk, Poland
Artur.Opalinski-at-pg-gda-pl

Abstract. *Monitoring of dynamic multi-agent systems, where agents are allowed to appear and disappear, and can migrate between network nodes is a complex tasks. Applying the traditional monitoring methods is not effective, as little can be assumed in advance about such environments. It is necessary to track changes in addressing and availability of agents to create and maintain mapping between agents and their network addresses. The paper presents a novel method of monitoring, which creates and maintains the address mapping by scanning the network address pool, aided by reports from agent monitors. The method deals with the dynamic environment offering more up-to-date information and less scanning than by using plain network address pool scanning.*

Keywords: *service registration, mulit-agent systems, monitoring.*

1. Introduction

When working with multi-agent systems it is often desirable to manage the agent set. One of the management tasks is agents monitoring [1]-[3] which may be useful to e.g. ensure a sufficient agent number or keep track of the agent services available [2],[4],[5].

The solution described in this paper has been invented for and implemented in a commercial data center for management of environmental sensors. The environment monitoring system is planned to utilize up to 100 agents. Each agent is responsible for sensors in its rack. A typical number of sensors, into industrial rack is around 6-10. The sensors are of different types (temperature, humidity, air flow, power consumption, switches, etc.). The agent task is to:

- detect its sensor number and types of sensors connected to it,
- sample and store reading from its sensors short-term,
- respond to monitoring center queries with the readings.

Due to the scale of the system, agents should be automatically detected and registered or de-registered on-the-fly by the monitoring system during typical operations, i.e. when agent is being:

- connected to the running system (installation),
- disconnected from the running system (deinstallation or maintenance tasks)
- restarted or provided with a new IP address while system is running.

For communication, agents can only use HTTP server software. This means that the agents are not capable of initiating connections. Specifically, agents are not capable of self-registration, and therefore their presence and their currently assigned IP address must be detected by a remote party. The number of agents makes it impractical to change software on every device, therefore the necessary tasks must be performed by entities external to the agents.

When agents are distributed over TCP/IP network, their IP addressing is usually entirely dependent on the network infrastructure services, consisting of DHCP [6] servers and routing equipment. Pairing DHCP with dynamic DNS [7] is a solution used for client devices, but it neglects software agent identity - and thus addressing information is separated from the multi-agent system. Therefore it is very difficult to supervise changes in agent population, i.e. the number of agents or other population's features. Assuming that agents may be on or off at any time, and that they may relinquish their current network addresses and acquire new addresses, the problems arise when trying to

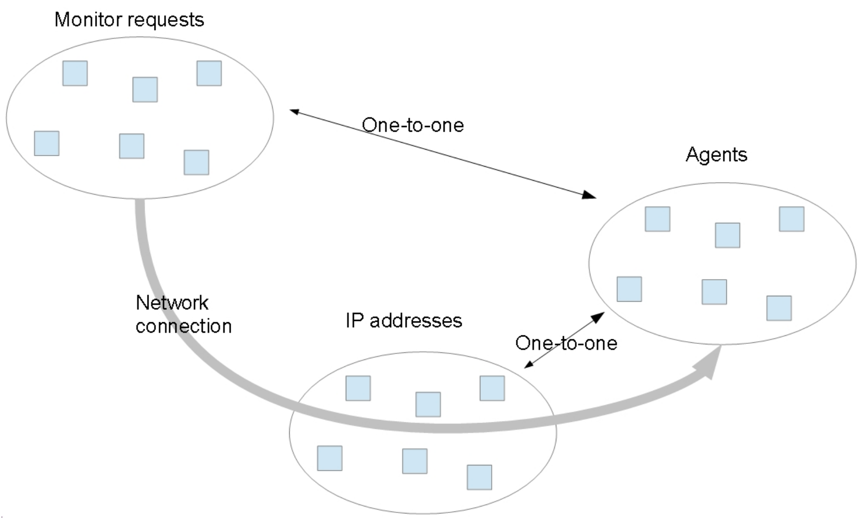

Figure 1. The problem of incomplete mapping

- detect new agents,
- detect agents which are gone,
- identify agents by network address.

The core of the problem is to complete a possibly current mapping (Fig.1) between agents represented by agent identifiers, and agents' network addresses.

The existing methods of central monitoring stems from two different fields of application. The first has its roots in in computer network monitoring [8]-[12]: it is assumed that all monitored entities are defined in advance, and that detecting unavailability of any of these predefined entities entail to raise alarm. Such scenario does not apply to dynamic multi-agent systems, where agents are allowed to appear and disappear, and can migrate between network nodes. In multi-agent systems usually not a particular agent is important, but the (non)existence of a reasonable-sized set of agents possessing the desired properties.

The other approach to monitoring is inherent in multi-agent programming environments or in multi-agent operating environments, when it is assumed that a well-known number of agents is created, and monitored [1],[3]. Communication with those agents is then considered reliable, and their total population is considered known. While necessary for some research, this approach is not general enough to cater for loosely controlled environments, where the total agent population is not known and often fluctuating, due to various factors, including but not limited to unreliable network connections.

To keep the mapping between IP addresses and agent or service identifiers, repositories are created in distributed environments [13],[15]. In the distributed services scenario [14],[15] the service providers self-register at a central repository, thus maintaining a central service directory, containing at least the network address and its associated agent or service identifier. E.g. in Web Services the provider publishes the WSDL to UDDI [16].

When using such service repository, discrepancies between the state of the agents and the state of the repository agents are limited. For every service requested, only the following cases are possible:

- A service is not registered, and it is off. In other words, the service does not exist yet.
- A service is not registered, and it is on. It is the duty of the service to complete the registration.
- A service is registered, and it is off. In other words, a service went off after registration or is unreachable, but the repository entry did not timeout yet, so the repository contains stale data.
- A service is registered, and it is on. This is the most straightforward case, when the repository reflects service's current state.

The case of stale data, mentioned above, remains sometimes unresolved [17]. This is unacceptable in a monitoring system.

The above approach offers very clear outcomes and is easy to implement, assuming that the agents are capable of registering to the central repository. This may not always be the case, though. The current information on the central repository location may not be available to agents, especially in setups involving multiple redundant repositories for fail-over. Also, agents may not be capable of initiating connections to the repository, e.g. due to lack of client-side software.

Still, a central mapping repository remains indispensible to allow to contact agents over network by other entities using agent identifiers. But the maintenance of such repository becomes more complex when agents do not play active role in maintaining it. Their passive behavior forces to use some form of autodetection of agents.

The contribution of this paper is to propose a solution involving dynamic central repository for an environment where agents are not capable of self-registering and must therefore be externally examined to maintain the mapping between agent network address and agent identifier. The solution encompasses protocol for resolving agent identifiers to names and for repository maintenance.

2. Solution

The environmental monitoring system finally implemented in the datacenter consists of:

- agents capable of detecting connected environmental sensors and sampling the measurements from sensors,
- monitoring console with plugin instances (monitors) , which connect to agents periodically, using agentID,
- a mapping provider, described further in this paper.

The features of the enviromental monitoring system shaped the following assumptions on the solution:

- each agent stores locally its unique ID, which is constant,
- agents do not initiate connections,
- agent software can not be modified,
- IP addressess are assigned dynamically to agents by separate infrastructure,
- monitoring console uses agent IDs to identify the agents to humans, and to contact the agents by means of its monitors over TCP/IP and HTTP.

In the solutions presented below it is assumed that the same agent's availability and addressing is perceived by monitors and by mapping provider. Monitors are multiple and possibly distributed. Network infrastructure ensures that the real mapping between network addresses and agent identifiers is one-to-one, which is anyway a requirement for proper network communication. It is further assumed that agent identifiers are unique.

The basic solution components are depicted in Fig.2 and are as follows:

- **Agents** which, when available and contacted, include their own identifier in response.
- **Monitors**, which are multiple, external, distributed entities making contact to the managed agents. Monitors only know the pertaining agent identifiers. Monitors may be any external entity, including, but not limited to other agents. Monitors send queries to resolver. These queries contain agent identifier.
- **Resolver**, which is responsible for answering monitors' queries with the currently known agent network address, matching the agent identifier. Resolver takes this information from the mapping repository. While maintainer is busy updating the repository, resolver continues to answer monitor queries.
- **Repository**, which stores the relation between network addresses and agent identifiers. Repository is read by the resolver on behalf of the monitors. Repository is kept by maintainer.
- **Maintainer**, which initiates the repository by scanning its associated network address pool and by registering in repository the responding agents by their identifiers. Maintainer then periodically re-scans its whole associated network address pool, or a consecutive part of the pool.

The basic solution has some inherent drawbacks, which are hard to overcome, assuming dynamic changes in agent population and changes in addresses of agents at any time:

- **Lack of scalability**: scanning of the address pool may incur heavy load on Mapping Provider and induces additional load on agents. It may also generate significant amount of network traffic, assuming that scanning needs to be performed often to cater for dynamic changes in agent addressing.
- **Arbitrary time of storing outdated information**: the point in time when repository information gets synchronized with current agents state, by scanning the address pool, leaves periods of time when repository information pertaining to some agents is outdated. The length of these periods of time is completely unrelated to the rate of changes in agent addressing and unrelated to the diversified communication needs of the monitors.

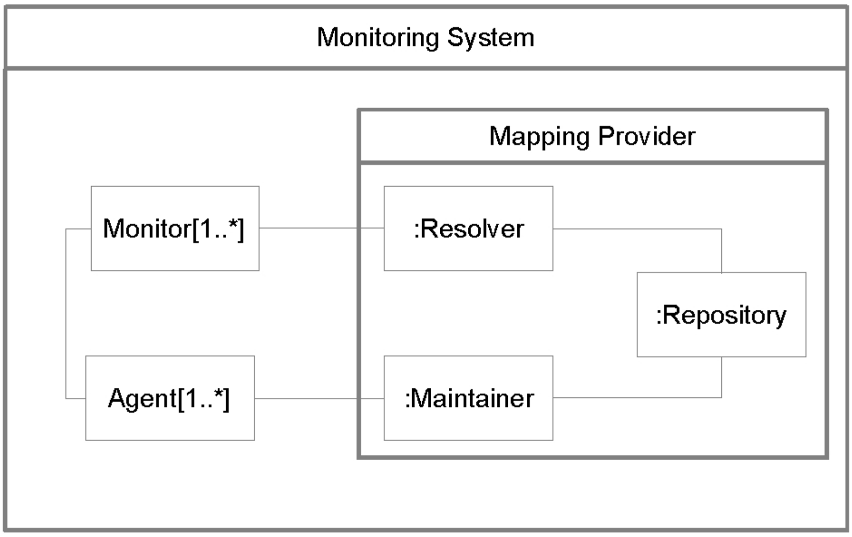

Figure 2. Internal structure diagram of the basic solution

- **Unmeasurable information divergency**: The divergency between repository information and agent state in time periods between address pool scans remains unknown.

Without agent self-registration, the number and complexity of possible divergency cases grows, as presented in Fig.3, but these discrepancies are impossible to discriminate in the basic solution.

When agents do not self-register, from the point of view of keeping the repository intact with the real set of available agents and with current agent addressing, the following cases need to be distinguished based on Fig.3:

- Need to invalidate (mark stale) a single stale entry in the repository (found by agent network address).
- Need to refresh network address in a single entry in the repository, by agent ID.
- Need to create a new entry by agent ID.
- Need to delete an entry by agent ID.

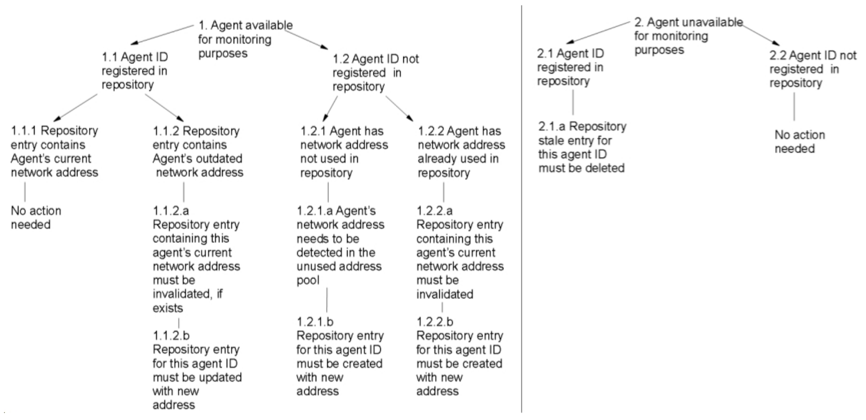
1. Agent available for monitoring purposes
1.1 Agent ID registered in repository
1.2 Agent ID not registered in repository
1.1.1 Repository entry contains Agent's current network address
No action needed
1.1.2 Repository entry contains Agent's outdated network address
1.1.2.a Repository entry containing this agent's current network address must be invalidated, if exists
1.1.2.b Repository entry for this agent ID must be updated with new address
1.2.1 Agent has network address not used in repository
1.2.1.a Agent's network address needs to be detected in the unused address pool
1.2.1.b Repository entry for this agent ID must be created with new address
1.2.2 Agent has network address already used in repository
1.2.2.a Repository entry containing this agent's current network address must be invalidated
1.2.2.b Repository entry for this agent ID must be created with new address
2. Agent unavailable for monitoring purposes
2.1 Agent ID registered in repository
2.1.a Repository stale entry for this agent ID must be deleted
2.2 Agent ID not registered in repository
No action needed

Figure 3. Possible cases for agents not capable of self-registration

Thus for each agent identifier three repository entry states are possible:

- agent entry non-existing,
- agent entry exiting, but marked stale,
- agent entry existing (and not marked as stale).

Entry state change diagram is presented in Fig.4.

It it worth to note that there are three possible sources of information for detecting an agent's availability and network address:

- re-scanning the address pool assigned to the Mapping Provider; re-scanning of the pool may be in part or as a whole,
- requests from Monitors, carrying agent identifiers,
- results of Monitors' connections to agents, which may be fed into the Mapping Provider.

Using all the above sources of information resembles working of cache memory [18]. In cache memories, the consumers of information determine which pieces of data are stored in cache entries. Even if cache is warmed initially, maintaining data in entries is the result of multiple memory accesses by consumers. The data is stored in the cache due to their activity. Similarly, the activity of monitors can

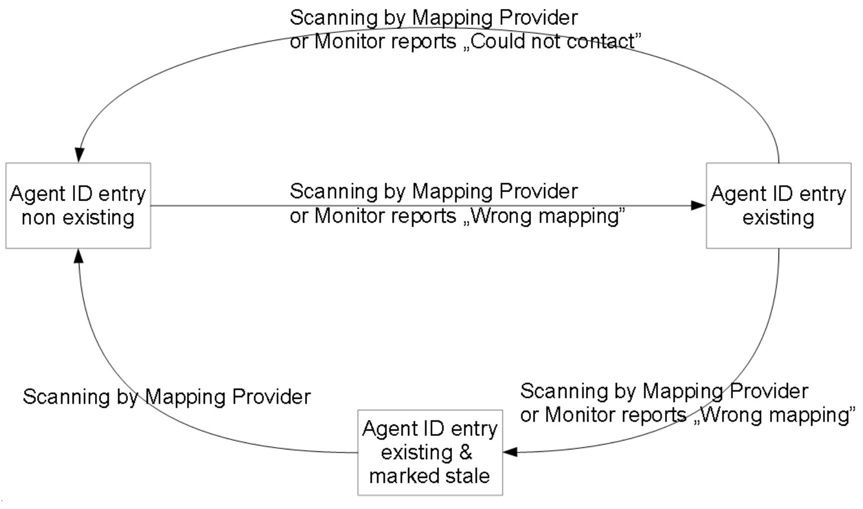
Scanning by Mapping Provider
or Monitor reports „Could not contact"
Agent ID entry non existing
Scanning by Mapping Provider
or Monitor reports „Wrong mapping"
Agent ID entry existing
Scanning by Mapping Provider
Scanning by Mapping Provider
or Monitor reports „Wrong mapping"
Agent ID entry existing & marked stale

Figure 4. State change diagram for agent identifier entry in repository

be exploited to at least confirm that repository mapping between agent's identifier and agent's address remains correct. Monitors' activity can also be used to report incorrect mappings. As it is typical in monitoring environments that most of the agents are responding, so multiple monitors can do most of the mapping checks and thus relive the load from maintainer. Only discovery of new or changed mapping needs to be done by maintainer then.

By encompassing the above mentioned additional information sources and by using timestamped repository entries as in Listing 1, a complete solution (Fig.5) introduces the following, additional functionality and components:

- **Agents**, acting a previously described.
- **Monitors**, which act as previously described. Additionally, when requesting resolving agent identifier to its corresponding network address, monitors get in response also the timestamp of the corresponding repository entry. In case the mapping proves incorrect when monitor tries to contact the agent, the incorrect mapping is reported back to the mapping provider, with indication of the issue, and with the unmodified timestamp value. There are two possible issues to be reported by Monitor: no answer from the resolved address, or an answer with an unexpected agent identifier.

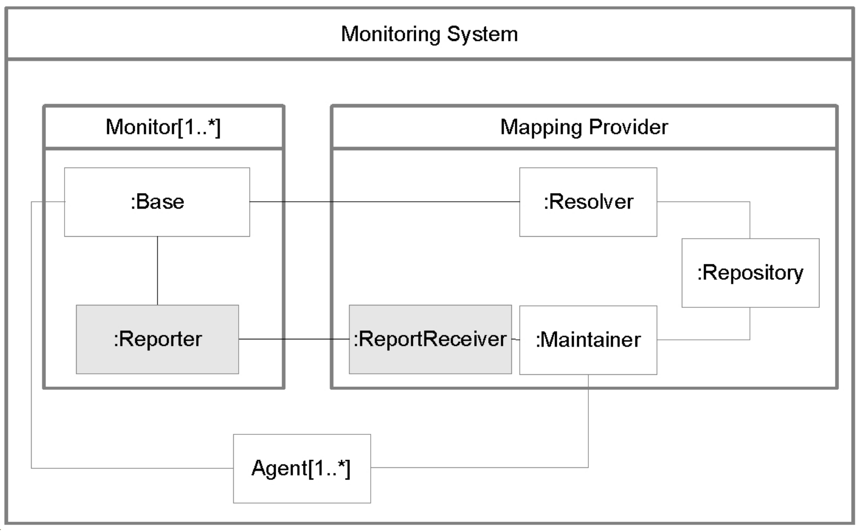

Figure 5. Internal structure diagram of the complete solution

- **Resolver**, which acts as previously, but it additionally sends to the requester the repository entry timestap along with the addressing information. Then the resolver updates this entry's timestamp.

- **Repository**,which acts as previously. Repository entry is presented on Listing 1.

- **Maintainer**, which scans the entire network address pool initially, and only parts of it later. Later scans of the pool can be performed less frequently than in the basic solution. It is achieved by reacting to monitor reports to update single repository entries, or by triggering re-scans on parts of its associated network address pool. Parts of network address pool to scan may be limited by considering only entries with old timestamps.

The monitor reports received when maintainer is busy get queued and are served on a FIFO basis. If an agent's entry got updated in the meantime by other activities, the pertaining report is silently ignored.

Listing 1. Repository entry structure

```
struct entry_s {
        char                agentID[N];
        unsigned int        agentIP;
        bool                stale;
        time_t              timestamp;
};
```

After initial creation, every repository entry has the stale flag reset, i.e. set to false. Assuming an entry with identifier ID1 and address IP1, when the response from the entry network address IP1 bears an agent identifier ID2 that does not match the ID1 stored in the corresponding entry, then

- the stale flag for entry ID1 is set true,
- the entry for ID2 is updated, or created, if not yet existing, with address IP1.

Response from agent can be obtained by the monitor, in the course of routine monitoring, and then reported to mapping provider through ReportReceiver. Response from agent can also be obtained by the maintainer itself, during address pool scanning, causing the same, above described behavior (Fig.6). When a monitor requests resolving an ID, and the repository entry for that ID is stale, the monitor gets a failure message from resolver, and the maintainer starts to scan the address space to refresh the mappings.

When the entry is read by the resolver on behalf of monitor, the timestamp is sent to monitor along the network address associated with this entry. Then timestamp of the entry in repository is updated. The timestamps in repository are also updated after scanning of associated addresses by maintainer. In case monitor reports mapping problem, the report contains the original value of the timestamp got during address resolution. All entries not newer than this timestamp are considered for scanning by maintainer. It may happen that the entries with newer timestamps are also incorrect, because timestamps get updated after entry read access. They will be corrected when reports pertaining to their agent identifiers arrive, again by only scanning the entries which are not newer.

Complete network address pool associated with maintainer needs to be scanned only initially, if at all. When no mapping exists for a given agent identifier, the monitor receives a resolution failure response, but the maintainer starts a scan to create the mapping, if possible. The scan does not need to be a full scan; the network addresses not used yet in repository entries are to be scanned first; the stale

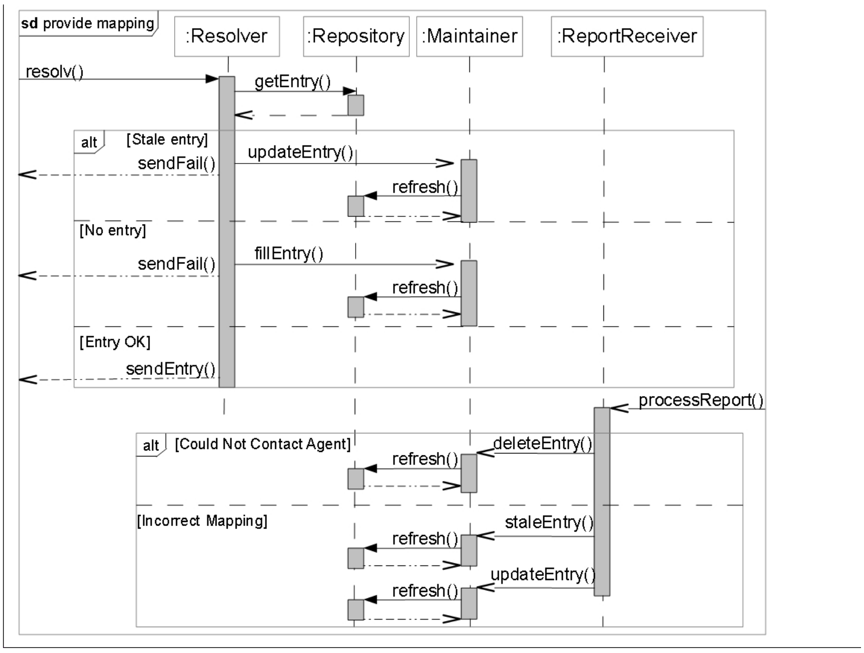

Figure 6. Mapping provider sequence diagram

entries in the repository constitute a next possible address set to scan; then the rest of the repository may be re-scanned, but entries with older timestamps are again more likely to offer a solution. Therefore complete network address pool scans are rarely performed in the complete solution.

3. Verification

To conduct verification in a predictable environment, the following has been set up: Twelve agents were assigned addresses from a pool of 13 IP addresses. These agent were only capable of serving content over HTTP, so they were incapable o self-registraion in any central repository. The HTML content contained agent's identifier as a string.

Maintainer's network address pool encompassed exactly the same 13 addresses as the pool assigned to agents. The mapping provider and monitor code was written in C an C++.

Table 1. Test cases

Case	No. of IP addresses	No. agents with static IP addresses	No. agents with dynamic IP addresses	No. of IP addresses in dynamic pool
1	13	12	0	1
2	13	11	1	7
3	13	6	6	2
4	13	0	6	12
5	13	0	12	13

A set of 12 monitors was monitoring the 12 agents; each monitor was using one of the agent identifiers. Monitoring was performed independently by the monitors, in 1 minute periods; i.e. in average 12 monitors were active in each minute.

The 13 IP addresses were assigned to agents in the following schema: agent were divided into two groups. One agent group was assigned a set of static IP addresses. The rest of agents were assigned dynamic IP addresses over DHCP, with lease time 180 seconds (3 minutes), from the pool of remaining IP addresses. The IP addresses from the DHCP pool were assigned alternately, i.e. agent got a new, different IP each time it requested an IP lease. Details of the test cases are presented in Table 1.

During verification, number of IP addresses scanned by maintainer have been counted. The results were as presented in Table 2. Verification results confirm that using information from monitor reports relieves mapping provider from performing scans: even when a significant number of agents (see case 3) change addresses, these changes will be detected and reported by monitors without need to rescan the address pool - as long as the changes are confined to the initially used set of IP addresses. Even if all agents change their addresses continuously (see case 5), little scanning is enough for the same reason as described above.

Only changes which result in addresses out of the previously assigned pool require heavier scanning (see case 4). But is must be remembered, that when neglecting reports from monitors, to get a 2 minute update time, 6 scans of the whole address space would be needed, making 6 * 13 = 78 addresses scanned in 12 minutes. So the complete solution method in case 4 is still significantly more efficient.

Table 2. Verification results

Case	No. of IP addresses scanned	Test time [min]	IP addresses scanned per minute
1	12	12	1.00
2	15	12	1.25
3	15	12	1.25
4	30	12	2.50
5	15	12	1.25

4. Conclusions

This paper presents a working solution to the problem of monitoring of dynamic multi-agent systems, where agents do not self-register, and are allowed to appear and disappear, and can migrate between network nodes. Thus the total agent population is not well-defined and often fluctuating, due to various factors, including but not limited to unreliable network connections. Such an environment creates a much greater challenge than monitoring entities with static network addressing, or monitoring entities capable of self-registration.

The term monitoring by itself may not be fully appropriate when dealing witch such complex problem. The issue, when researched, seems to more fit the description of patrolling an area, where agents are at play.

An interesting feature of the proposed solution is that agents are unaware they are under supervision. No special activity is required from agents, and only their standard interfaces are used. Thus the solution can be easily introduced in big and distributed agent networks - as no changes on the agent side are required.

The solution avoids frequent scans of the assigned address space by also utilizing information from monitor connection attempts. Thus the solution puts less load on the network and on the mapping provider than simple scanning, while generally keeping more up-to-date with changes in the environment.

The meaning of keeping the mapping repository up-to-date with changes in agent population or agent addressing becomes fuzzy - the agents which are contacted more often by monitors have their repository entries checked more frequently, while agents that no one connects to are only contacted periodically by maintainer, when other changes force network address pool scans. This means that practically the effectiveness is better than when using only network address pool scans.

Multiple sets of resolvers, repositories and maintainers would be useful for high-availability patrolling solution, but the related details have to be worked out yet.

5. Bibliography

References

[1] Waligora, I., Malysiak-Mrozek, B., Mrozek, D., Uniwersalna platforma wieloagentowa UMAP, Studia Informatica, 2011, Vol.32, No.2B(97).

[2] Opalinski, A., Adapting A General Tool To Monitoring Multi-Agent Systems Through Virtual Host Layer Extension, Studia Informatica, Vol.33, No 2A(105), pp.99-110, PL ISSN 0208-7286, Silesian University of Technology Press, Gliwice, 2012.

[3] JADE project Home Page, URL:http://jade.tilab.com/, (DOA: 1.06.2013).

[4] Miller, P. The Smart Swarm: How understanding flocks, schools, and colonies can make us better at communicating, decision making, and getting things done, ISBN 9781583333907, New York:Avery, 2010.

[5] Parunak, H., Brueckner, S.,A., Sauter, J.,A., Matthews, R., Global convergence of local agent behaviors, AAMAS ’05 Proc. Fourth International Joint Conference on Autonomous Agents and Multiagent Systems, 2005.

[6] Droms, R., Dynamic Host Configuration Protocol, IETF RFC 2131, available online, URL: http://tools.ietf.org/html/rfc2131 (DOA: 1.06.2013).

[7] Vixie, P., Thomson, S., Rekhter, Y., Bound, J., Dynamic Updates in the Domain Name System, IETF RFC 2136, available online, URL: http://tools.ietf.org/html/rfc2136 (DOA: 1.06.2013).

[8] IBM Tivoli Monitoring: Administrator’s Guide, available online, URL: http://publib.boulder.ibm.com/infocenter/tivihelp/v42r1/index.jsp?topic=%2Fcom.ibm.omegamon.share.doc%2Fzconfigcommon08.htm (DOA: 1.06.2013).

[9] IBM Tivoli Netcool/OMNIbus Administrator's Guide (former Micromuse Netcool/OMNIbus), available online, URL:http://publib.boulder.ibm.com/infocenter/tivihelp/v8r1/topic/com.ibm.netcool_OMNIbus.doc_7.3.0/web_pdf_adm_master_73.pdf (DOA: 1.06.2013).

[10] BMC Event Manager documentation, https://communities.bmc.com/ communities/BEM (DOA: 1.06.2013).

[11] HP OpenView documentation portal, available online, URL: http://www.openview.hp.com (DOA: 1.06.2013).

[12] Network Monitoring VPS with Nagios, OpenNMS, Zenoss and More, blog entry by Chinonye, July 5, 2011, available online, URL: http://myhosting.com/blog/2011/07/networkmonitoring-vps-nagios-opennms-zenoss-more/ (DOA: 1.06.2013).

[13] Mockapetris, P., Domain Names - Implementation And Specification, IETF RFC 1035, available online, URL: http://tools.ietf.org/html/rfc1035 (DOA: 1.06.2013).

[14] SOAP Version 1.2 Part 1: Messaging Framework (Second Edition), W3C Recommendation, 2007, available online, URL: http://www.w3.org/TR/soap12-part1/ (DOA: 1.06.2013).

[15] LU - JiniTM Lookup Service Specification, Jini Technology Core Platform Specifications, available online, URL: http://river.apache.org/doc/specs/html/lookup-spec.html (DOA: 1.06.2013).

[16] OASIS UDDI Specifications TC - Committee Specifications, available online, available online, URL: https://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm (DOA: 1.06.2013).

[17] Discussion on Requirement 19 in UDDI specification, see e.g. item 7.11 Stale Data in Minutes of UDDI Spec TC Telecon, available online, available online, URL: https://www.oasis-open.org/committees/download.php/9730/Minutes20041019.doc (DOA: 1.06.2013).

[18] Jia Wang, A survey of web caching schemes for the Internet, ACM SIGCOMM Computer Communication Review, Vol. 29, Issue 5, pp.36-46, 1999.