IoT Protection Through Device to Cloud Synchronization

Christian Gehrmann and Mohammed Ahmed Abdelraheem
SICS Swedish ICT AB
Ideon Science park
Scheelevagen 17
223 70 Lund, Sweden

***Abstract*—This paper addresses the problem of protecting distributed IoT units from network based attacks while still having a high level of availability. In particular we suggest a novel method where the IoT device execution state is modeled with a suitable high level application model and where the execution state of the application of the IoT device is "mirrored" in a cloud executed machine. This machine has very high availability and high attack resistance. The IoT device will only communicate with the mirror machine in the cloud using a dedicated synchronization protocol. All essential IoT state information and state manipulations are communicated through this synchronization protocol while all end application communication directed towards the IoT units is done towards the mirror machine in the cloud. This gives a very robust and secure system with high availability at the price of slower responses. However, for many non-real time IoT application with high security demands this performance penalty can be justified.**

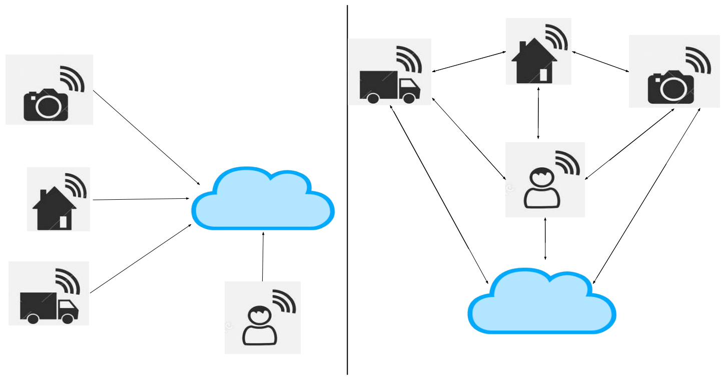
Fig. 1. Centralized IoT (left) vs Decentralized IoT (right)

I. INTRODUCTION

In the near future, a very large number of IoT devices will perform critical security tasks in systems for industry process control, building automation, power control, healthcare etc. The correct operation of each of these units is crucial for the robustness of the system. Failure of a single critical component can give very severe consequences. Many IoT devices are resource constraint with respect to power, CPU capacity, memory etc. Hence, they are hard to protect from network based attacks such as Denial of Service (DoS) attacks, distributed DoS (DDoS) [1]. The main defense strategy against this type of attacks is restricting the communication interface towards the IoT devices which makes it both harder to perform attacks against the IoT device as well as limiting the consequences of successful attacks. However, this obviously has the drawback that the IoT devices might be harder to reach or the power consumption on the IoT devices goes up as the defense strategy requires more CPU cycles. Hence, there is a large need for solutions that find the right balance between availability and security of IoT devices. This paper addresses exactly this area by suggesting a novel principle for IoT protection with fairly high availability.

As shown in Figure 1 IoT architectures can be classified into two categories: centralized IoT and decentralized IoT. Unlike centralized IoT where all the IoT devices are passive as their only task is to provide data, distributed IoT allows connected entities to retrieve, process, combine and provide data and services to other entities [4]. Our work addresses the vulnerabilities of the distributed IoT against DDoS or denial of sleep attacks [7], [5], [6] which could be mounted due to direct connection between the IoT devices and the client. Obviously, a centralized IoT offers better protection to the IoT devices against DDoS since they do not have direct connection with the clients. In order to protect the IoT devices against these attacks, we propose a solution combining the decentralized and centralized IoT architectures by building an "IoT virtualization" cloud infrastructure at a centralized IoT network, where each IoT device is mirrored in the cloud in the form of a virtual machine and the end-user application interact with this virtual machine in exactly the same manner as if it is interacted with the real IoT device directly. The actual IoT device then uses a special purpose synchronization protocol with the virtual machine mirror in the cloud and in this manner indirectly interacts with the end-user applications. The advantage with this approach is that it is much harder for an adversary to attack the IoT device as much better protection can be provided on the "mirror machine" than on the real machine. Hence, a good level of DoS protection is given without the need to modify at all the end IoT application and its interactions patterns.

A. Existing Solutions

The work in [5] presented a survey about denial of sleep attacks and defenses in wireless sensor networks. Many of the threats discussed in [5] can be defeated using encryption and authentication algorithms along with other jamming identification techniques and trigger techniques to preserve energy on affected devices. MAC-based solutions have been previously proposed in [8], [9], [10]. However, the work in [6] showed that if attackers are aware of the MAC protocol used

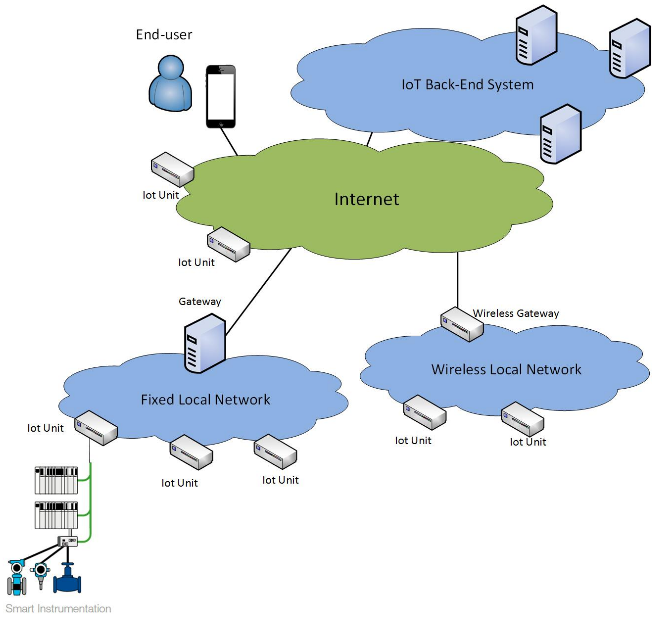

Fig. 2. Typical set-up in current systems.

in the wireless sensor networks then they can mount denial-of-sleep attacks. The authors of [6] also proposed a framework for mitigating these denial-of-sleep attacks using a strong link-layer authentication along with other techniques. More recently, the work in [2] presented another MAC solution for protecting IoT devices against DoS based on a short MAC that can be used by an IoT device to distinguish authorized packets from non-authorized packets.

Other DDoS mitigation techniques typically "hide" the IoT devices between local networks and GateWays (GWs) in the networks where strict filtering rules are applied to protect sensitive IoT devices from attacks as depicted in Figure 2. It is then the responsibility of the GW or firewalls in the network to filter out hostile traffic and/or act as proxies in the network that distinguishes "open domain" (i.e. internet) protocols from "private domain" (i.e. local network) protocols where the IoT devices are situated. The main drawback with filtering based solutions is that it is very hard to code them such that they provide a high security level and at the same time they avoid filtering out valid traffic. With respect to proxy based system, they are often rather costly to deploy as they must be customized typically for each and every IoT application. Furthermore, the proxy security filtering rules must be very strict in order for them to avoid passing through potential hostile traffic targeting the IoT devices behind the proxy.

Our solution enjoys the robust security properties of centralized IoT architectures against DoS attacks provided by the cloud and the direct interaction facility between clients and IoT devices existing in distributed IoT architectures.

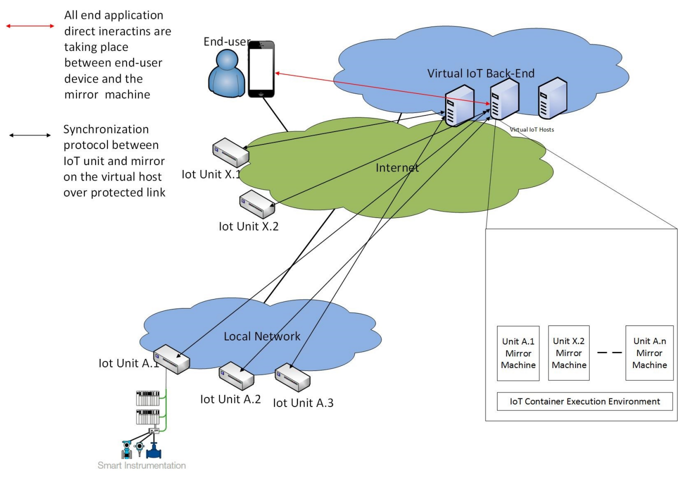
All end application direct ineractins are taking place between end-user device and the mirror machine
Synchronization protocol between IoT unit and mirror on the virtual host over protected link
End-user
Virtual IoT Back-End
Virtual IoT Hosts
Internet
Iot Unit X.1
Iot Unit X.2
Local Network
Iot Unit A.1
Iot Unit A.2
Iot Unit A.3
Unit A.1 Mirror Machine
Unit X.2 Mirror Machine
Unit A.n Mirror Machine
IoT Container Execution Environment
Smart Instrumentation

Fig. 3. Main principle of the solution.

B. Problem Statement

We consider the problem of protecting sensitive and resource constraint distributed IoT devices from hostile traffic like DoS/DDoS and network based attacks on open networks while still being available for legitimate traffic in the network. In particular, we consider the problem of combining a high level of protection with high availability. This means that while we would like to hide resource constraint IoT devices from being directly exposed to Internet traffic, we still want them be accessible by legitimate users.

II. Summary of the New Solution

A. Process

We suggest a novel principle for protection of IoT units against network based attacks. All IoT devices are utilizing a *unique device* virtual machine *mirror* in a virtual infrastructure back-end system. The mirror *is not* a pure virtual machine copy of the IoT execution, but a *high level* machine that regularly *synchronizes* the essential IoT state with the corresponding state in the real IoT device. Furthermore, the mirror machine receives *all* direct requests targeting the real IoT device from the application domain, i.e. all traffic directed towards the real IoT device. This implies that the external world that wants to interact with the IoT devices in the system, *always* needs to communicate with their corresponding *mirror machines* only. No direct communication with the IoT devices themselves is allowed. The IoT devices on the other hand *only* run a dedicated synchronization protocol with their "own" virtual mirror and will not accept any other network based communication at all. The overall principle is depicted in Figure 3. In summary, the proposed system solution has the following main characteristics:

- During IoT device deployment, the IoT device is launched together with a dedicated "IoT mirror machine" in a cloud infrastructure. The IoT devices and these mirror machines are configured with security credentials such that they can interact securely over open networks.

- The IoT device runs a special synchronization protocol that keeps it synchronized with its corresponding mirror machine in the cloud. This synchronization protocol keeps the state information of the most important data objects on the IoT device synchronized with the corresponding data on the mirror machine. Synchronization is always initiated by the IoT device which will only accept synchronization sessions initiated by itself. All other network information is blocked at the IoT device which can also choose to turn off all network interfaces when there is no ongoing synchronization.
- An end-application such as an end-user client or a back-end application system that wants to exchange data or interact with an IoT device, does not have direct network access to the IoT device, but will always interact with the mirror machine in the cloud. Hence, from the end IoT application point of view, the IoT infrastructure consists of the mirror machines only and it is not aware of the physical machines at all or their network addresses.
- When a request reaches the IoT mirror machine which requires direct execution on data sets belonging to the "real" IoT device, the mirror machine will be able to execute those instructions on the mirror machine data (local copy of the data objects from the IoT device) and respond to the request. Any manipulation that might happen on the data due to such request will be synchronized with the IoT device during the next synchronization operation. This implies that there is a "delay" that will depend on the frequency of synchronizations between outside request and their execution on the real IoT device. The exact delay time will depend on how frequent the synchronizations take place.
- An end-application might request an operation that cannot be performed on stored data objects values only. In this case the IoT mirror machine creates one or several "execution requests" similar to the principle described in [3]. These execution commands are then transferred from the mirror machine at the next synchronization operation and the potential responses are transferred back to the mirror machine.

B. Advantages

The present paper gives very high protection of sensitive IoT devices such as resource constraint and battery driven devices. This is accomplished by the fact that a device will never accept any network session which it has not initiated itself and as it will only interact directly with its corresponding mirror machine in the virtual back-end [1]. Instead, it receives all its data and operation requests through synchronization operations with the mirror machine. Hence, all attacks targeting sensitive IoT devices must be launched either against the synchronization interaction protocol or against the mirror machine instead of the real IoT device. As the mirror machine is not operating on a resource constraint platform, it can run with very high protection against DoS and other network based attacks. This makes it a much less attractive attack target than the real IoT device. Furthermore, by creating a high level data and execution model of the IoT device in the mirror machine, it will never be possible for an adversary to make any low level attack attempts toward the real IoT device since such manipulations or execution events cannot take place.

The main drawback with our solution is obviously that real-time communication with the IoT devices is not possible. This might not be acceptable for some IoT applications, on the other hand, there are a large set of applications where the delay in the interactions with the IoT devices does not cause any major problems. Such applications include sensors that regularly reports data for central processing such as temperature, humidity, wind power, IR detection, etc. Other application examples are devices that are centrally managed with respect to configuration, firmware update, etc, but are used for local control, i.e. to control a process over a local wired ore wireless interface. For those applications, the suggested approach gives a very robust and secure system solution.

Another advantage with our proposed solution is that the IoT device does not necessarily need to provide an end application remote interface. It is actually enough that the mirror machine provides such interface. Since the mirror machine runs on a powerful device, it will be able to satisfy integration requirements with a very large number of different application systems. Hence, apart from giving a much more secure system, the solution also gives a bi-effect simpler application system integration.

III. Detailed Design

Next, we give a detailed description of our solution. We discuss three different aspects that constitute the core of our solution:

- The design and principle for the mirror machines in the virtualized infrastructure.
- IoT device and mirror machines deployments.
- IoT device to mirror machine synchronization.

A. Mirror machine design

The basic principle of our solution is based on a design with a "mirror machine" that operates at a virtual back-end infrastructure (a cloud). The design is assuming that the IoT system developer is able to construct a data object model of the most important data sets that the IoT device handles. It also needs to define a distinct set of operations that the IoT devices perform on these data objects. All these operations must then be implemented and supported on the IoT mirror machine as well. Furthermore, the IoT system developer needs to define (if needed) a set of operations performed by the IoT devices which do not simply depend on those defined data objects sets but also on other state or external input devices available at the local IoT network.

These set of operations constitute the "supported command set". This command set is used to transfer command requests

[1]Obviously the IoT unit might need general IP configurations which might include network traffic. If this is the case, this type of traffic will obviously be allowed at the IoT device (at least for a limited amount of time).

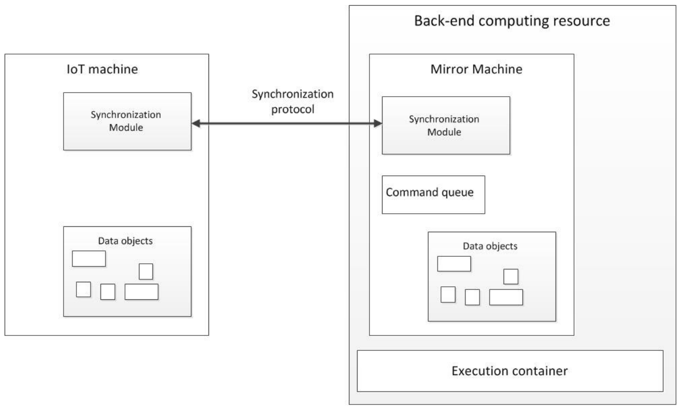

Fig. 4. Mirror machine model.

pending in the mirror machine (which cannot execute them) to the IoT device.

A synchronization execution unit with some regularity both on the IoT device and the mirror machine are the modules in the system responsible for synchronizing the data objects at the IoT devices and the mirror machine respectively.

How often synchronization takes place is system specific and will depend on the real-time requirements of the IoT application. Some of the data objects are typically only changed due to state changes in the IoT devices while some other data objects are changed due to incoming requests to the mirror machine from the IoT end-application system.

Some IoT requests from the end application system toward the mirror machine cannot be executed by the mirror machine itself. These are then placed in a "command queue" at the mirror machine. All pending commands are transferred from the mirror machine to the IoT device when the next synchronization occurs and then executed on the IoT device. The result of the execution is reported back in the form of updated data objects. An overview of the system and the mirror machine model is shown in Figure 4.

B. IoT device and mirror machines deployments

The solution consists of the following main deployment steps:

- Develop the data object model, IoT device model and define shared commands and realize a mirror machine (binary) that follows this model [2].
- Launch the mirror machine on suitable available computing resources. This can be done in the form of a Java virtual machine, a system virtual machine or an application running in a cloud environment. Some different options are outlined in Figure 5.
- Configure the Mirror machine and the IoT device with shared credentials that allow them to set up secure connections during their synchronization process. Also the IoT device must be equipped with the network address of the mirror machine. The shared credentials can be in the form of shared secret keys or shared trusted public keys and corresponding certificates [3].
- Install and start the IoT device.
- Once installed, the IoT device connects to its mirror machine and makes sure it can establish a secure authenticated connection with its mirror machine. If this succeeds, the IoT device is ready for use.

The overall deployment procedure is depicted in Figure 6.

C. IoT device to mirror machine synchronization and machines operations

In the descriptions, we use the following notations:

- We denote an arbitrary IoT device in the local network by u.
- We denote the mirror machine corresponding to u by m_u.
- We denote the command queue at m_u by q_m.
- We denote a data object set used by the mirror machine and the IoT by D.
- We denote the data objects at m_u in D which are changed since the last synchronization attempt by d_m.
- We denote the data objects at u in D which are changed since the last synchronization attempt by d_u.
- We denote the parameter at u determining the time between two consecutive synchronization attempts by t.

The synchronization procedure is completely determined by the configurations at the device u. The IoT device decides when to connect to the mirror machine and perform the next synchronization operations. During the synchronization session, all the data objects (part of the IoT mirror machine model) changes on the mirror machine since the last synchronization session, d_m, are updated on the IoT device to match the changes. Similarly, all the data objects on the IoT device changes since the last synchronization session, d_u, are updated on the mirror machine to make sure the two data object sets on both the IoT device and its mirror machine are identical at the end of the synchronization session. Furthermore, the mirror machine, m_u, maintains a command queue, q_m, which is a command buffer containing all pending commands that must be executed on the real IoT device and which are the results of requests targeting the IoT devices received by the mirror machine since the last synchronization session. The synchronization session takes place in the following order over a secure channel (authenticated, integrity and confidentiality protected such as for instance a TLS or DTLS channel) between u and m_u:

1) u requests all changes in d_m, from m_u and updates its own data set.
2) u requests all commands in the current command queue at m_u.
3) u executes all the pending commands locally (this might result in an update of the data set d_u).

[2]This step can be considerably simplified if a pre-defined model is used.

[3]This can be done using a manual process as depicted in Figure 6 or preferably through automatic processes during the deployment of the IoT device and its mirror machine (Step 4 and Step 2 respectively)

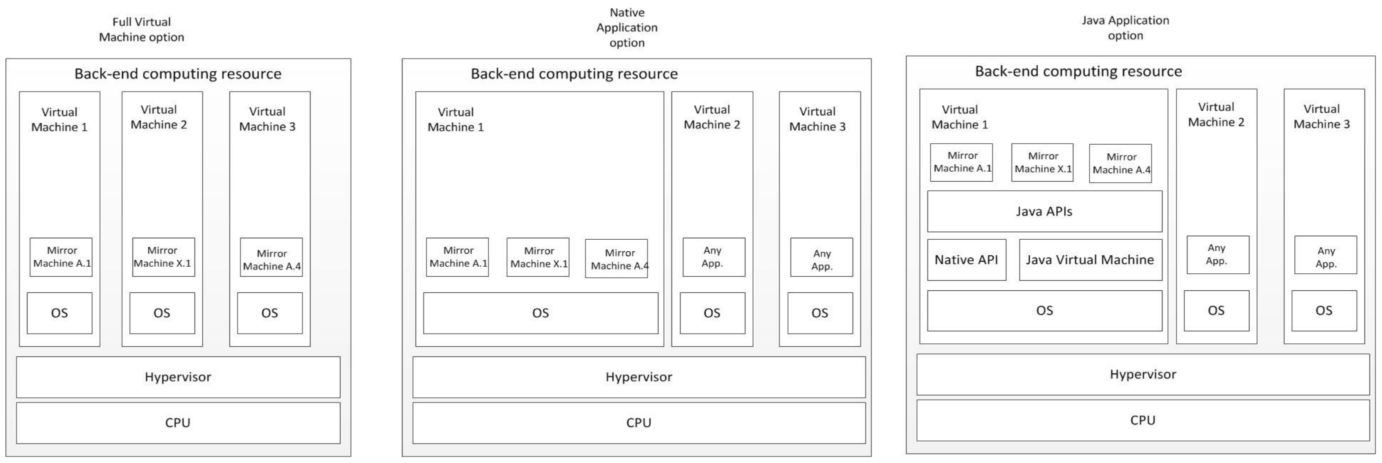

Fig. 5. Some different mirror machine deployment options.

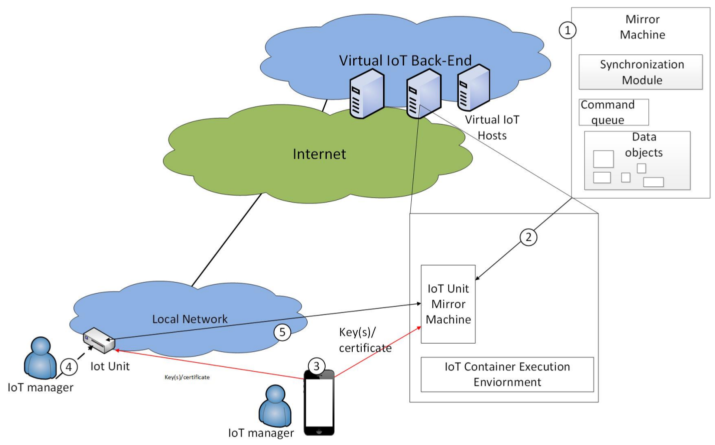

Fig. 6. IoT device and mirror machine deployments.

4) u sends all the updates in d_u to m_u which updates its local data object set such that the two sets now coincides.
5) Depending on the type of commands or the number of changes identified in the data sets at m_u, the device u, might decide to change the synchronization time, t to more or less frequent synchronizations.

Below, we describe the details of the operation principles of the mirror machine and the real IoT device respectively.

Mirror Machine procedure:

1) The mirror machine, m_u, is deployed on the system (see Section III-C) and the sets d_m and q_m are both empty.
2) The mirror machine waits for requests from connecting IoT clients or from synchronization requests from u.
3) If a new IoT client request arrived the following applies:
   a) If the request was authorized (allowed according to the IoT mirror machine security policies), the request is accepted and executed.
   b) All data object changes which are the result of the request in step 3.a are marked and the set d_m is updated.
   c) All requests that results in commands that are not possible to execute on m_u are transformed to commands and put on the command queue, q_m.
   d) Jump to Step 2.
4) If a new synchronization request from u arrives, the following applies:
   a) The request is authenticated using the pre-installed credentials shared with u and if it comes from any other device but u, it is refused (move back to step 2).
   b) m_u sends all updates in the set d_m to u.
   c) m_u sends all the pending commands in q_m and the command queue is cleared.
   d) m_u waits for a data set update commands from u. If such commands are received, the set D is updated to match the corresponding set at u.
   e) m_u waits for an end to the ongoing synchronization session with u.
   f) Execute any of the commands that are pending in the command queue which are due to the ongoing connected client requests.
   g) Move to Step 2.

IoT unit procedure:

1) The IoT device is deployed on the system (see Section III-C) and the set d_u is set to empty.
2) The IoT device starts a timer, $t_c = 0$.
3) If $t_c \geq t$ the following applies:
   a) u uses the pre-configured network address of m_u to make an authenticated new secure synchronization connection attempt to m_u (using a suitable protocols such as TLS or DTLS for instance).
   b) u requests the set d_m and all the updates to the objects in this set from m_u and based on this information it updates its own internal data objects in the set D.
   c) u requests all pending commands in the queue q_m from m_u.

d) u executes all pending received commands from m_u. This results in an update of the data in the set d_u [4]
e) u sends the set d_u and all the changes to this data set to m_u.
f) u closes the synchronization session with m_u.
g) u evaluates the changes in D and the received commands from m_u in the last synchronization session and updates t according to the predefined policy determined by the particular IoT application [5].
h) Move to Step 2.

IV. CONCLUSIONS

The synchronization and mirror machine based protection principle described in this paper gives a very high network attack protection level for distributed IoT units, which do not have strict real-time requirements. The most vulnerable IoT units are battery driven and/or resource constraint units. These are also units typically not having strict real-time requirements. This implies that the solution is useful in a very large set of distributed IoT use-case scenarios where resource constraint IoT units need to be protected from network based attacks. On the other hand, more powerful distributed IoT units with real-time requirements will not benefit from using the suggested approach, but these also have other rather efficient means for protection against network based attacks.

ACKNOWLEDGMENT

The authors would like to thank the reviewers for their valuable comments and suggestions. This work was supported by European Unions Horizon 2020 research and innovation programme under grant agreement No 644814, the PaaSword project within the ICT Programme ICT- 07-2014: Advanced Cloud Infrastructures and Services.

REFERENCES

[1] Cisco white paper, "Strategies to Protect Against Distributed Denial of Service (DDoS) Attacks", April 22, 2008,http://www.cisco.com/c/en/us/support/docs/security-vpn/kerberos/13634-newsflash.html.

[2] Gehrmann, C., Tiloca, M. and Hglund, R., 2015, June. SMACK: Short message authentication check against battery exhaustion in the Internet of Things. In Sensing, Communication, and Networking (SECON), 2015 12th Annual IEEE International Conference on (pp. 274-282). IEEE.

[3] S. H. Maes, Passing client or server instructions via synchronized data objects, US Pat. No. 7,870,412, January, 2011.

[4] Roman, R., Zhou, J. and Lopez, J., 2013. On the features and challenges of security and privacy in distributed internet of things. Computer Networks, 57(10), pp.2266-2279.

[4]This set now includes all local (IoT devices) changes since the last synchronization session as well as potential changes to the data object set due to the executed commands received from m_u.

[5]If a lot of changes are done in the data set this might imply that the IoT device should do more frequent synchronizations to give better responses to connecting clients. Similarly, if any of the commands received from m_u indicates the rapid responses expected by connecting IoT clients, u will change t making more frequent synchronizations. On the other hand, if there are no updates or few and not urgent updates are the result of last synchronization session, u might change t such that the synchronization with m_u happens less frequent.

[5] Raymond, D.R. and Midkiff, S.F., 2008. Denial-of-service in wireless sensor networks: Attacks and defenses. IEEE Pervasive Computing, 7(1), pp.74-81.

[6] Raymond, D.R., Marchany, R.C., Brownfield, M.I. and Midkiff, S.F., 2009. Effects of denial-of-sleep attacks on wireless sensor network MAC protocols. IEEE transactions on vehicular technology, 58(1), pp.367-380.

[7] Martin, T., Hsiao, M., Ha, D. and Krishnaswami, J., 2004, March. Denial-of-service attacks on battery-powered mobile computers. In Pervasive Computing and Communications, 2004. PerCom 2004. Proceedings of the Second IEEE Annual Conference on (pp. 309-318). IEEE.

[8] Ye, W., Heidemann, J. and Estrin, D., 2004. Medium access control with coordinated adaptive sleeping for wireless sensor networks. IEEE/ACM Transactions on networking, 12(3), pp.493-506.

[9] Polastre, J., Hill, J. and Culler, D., 2004, November. Versatile low power media access for wireless sensor networks. In Proceedings of the 2nd international conference on Embedded networked sensor systems (pp. 95-107). ACM.

[10] Van Dam, T. and Langendoen, K., 2003, November. An adaptive energy-efficient MAC protocol for wireless sensor networks. In Proceedings of the 1st international conference on Embedded networked sensor systems (pp. 171-180). ACM