FACTA UNIVERSITATIS
Series: **Electronics and Energetics** Vol. 29, N^{o} 3, September 2016, pp. 475 - 487
DOI: 10.2298/FUEE1603475S

BRIDGING THE SNMP GAP: SIMPLE NETWORK MONITORING THE INTERNET OF THINGS

Mihajlo Savić

University of Banja Luka, Faculty of Electrical Engineering,
Republic of Srpska, Bosnia and Herzegovina

Abstract. *Things that form Internet of Things can vary in every imaginable aspect. From simplest devices with barely any processing and memory resources, with communication handled by networking devices like switches and routers to powerful servers that provide needed back-end resources in cloud environments, all are needed for real world implementations of Internet of Things. Monitoring of the network and server parts of the infrastructure is a well known area with numerous approaches that enable efficient monitoring. Most prevalent technology used is SNMP that forms the part of the IP stack and is as such universally supported. On the other hand, "things" domain is evolving very fast with a number of competing technologies used for communication and monitoring. When discussing small, constrained devices, the two most promising protocols are CoAP and MQTT. Combined, they cover wide area of communication needs for resource constrained devices, from simple messaging system to one that enables connecting to RESTful world. In this paper we present a possible solution to bridge the gap in monitoring by enabling SNMP access to monitoring data obtained from constrained devices that cannot feasibly support SNMP or are not intended to be used in such a manner.*

Key words: *IoT, monitoring, SNMP, CoAP, MQTT*

1. INTRODUCTION

Internet of Things (IoT) may mean many different things to many different people, but few would disagree that in order to achieve the full potential of smart environment based on IoT one needs to be able to fully monitor all of the things that do make IoT possible. Although there is a wealth of monitoring products as well as comparable number of standards and platforms that go hand in hand with them, there is one standard that has been around for a long time, is implemented in almost all networking devices and is even a part of the set of the protocols that enable modern networking to exist.

Received June 30, 2015; received in revised form November 12, 2015
Corresponding author: Mihajlo Savić
University of Banja Luka, Faculty of Electrical Engineering, Patre 5, 78000 Banja Luka, Republic of Srpska, Bosnia and Herzegovina
(e-mail: badaboom@etfbl.net)

As it often is, with age it gained robustness and reliability, but lost some of the appeal to newer generations and younger monitoring systems, though one would be hard pressed to find a monitoring product that does not support it. It is also important to note that monitoring is never easy and in production tried and true solutions have proven themselves worthy throughout the history. To monitor the IoT we need to monitor any and every device that makes it or provides the services to it, from smallest and simplest single function sensors to ritualized back-end services needed to transform raw data into usable information.

Currently, Simple Network Management Protocol (SNMP) is the protocol that enables uniform monitoring of all parts of the IoT infrastructure, save for the simplest of devices. As even those devices need to be monitored, presented in this paper one of the possible SNMP based solutions for end-to-end monitoring is.

Solution described in this paper covers one possible use of SNMP in monitoring IoT infrastructures, enabling monitoring of just IoT devices as well as larger heterogeneous infrastructures that can also contain complex IaaS entities that provide services to IoT devices.

2. SIMPLE NETWORK MANAGEMENT PROTOCOL

Simple Network Management Protocol (SNMP) is a part of Internet Protocol Suite (IP) set of protocols as defined by Internet Engineering Task Force (IETF) [1], organization in charge of defining standards and protocols that provide base for existence and exchange of data over the Internet. SNMP defines a set of standards for network management that include application protocol, database schema, as well as the definition of data sets. Relatively small numbers of what we generally consider to be standards are in fact full standards and this only gives weight to SNMP and its use in management and monitoring areas. Common use of SNMP is in default configuration that consists of at least one computer or other device that has administrative role (master) and a group of managed networked devices that are controlled by the master device. Every managed device (slave) is running a software component called an agent that is in charge of communication with master node. Agents provide for access to various system variables of managed device (e.g. system identification, available resources, resource consumption, etc.) but also provide a mechanism to control the device by setting the values of specified variables to desired values (e.g. bringing network interfaces up or down, changing their addresses, etc). Data transfer is typically done over User Datagram Protocol (UDP) and default port numbers 161 on the agent side and 162 on master side. Communication can be initiated by the master through use of GET operations for accessing the data and SET operations used to modify the data, as well as by the managed device through the use of TRAP or INFORM operations used to send data to management node.

2.1. Versions of SNMP protocol

SNMP standard has been so far defined by three versions as will be described in following text. SNMP version 1 (SNMPv1) was defined by RFC documents number 1155, 1157 and 1215. Although it has a “historic” status today, it is still widely used as it is supported by almost all network equipment manufacturers for nearly all networking devices. Security model leaves a lot to be desired as it is based on so called “community” strings that can be seen as a shared secret or access passwords. Biggest issue lies in the fact that all communication, including community strings, is performed in unencrypted

form. SNMP version 2 (SNMPv2) was defined by RFC documents 1441-1452 and introduced a host of improvements in the area of security, by utilizing more complex security model, and performance, by introduction of GETBULK operation.

SNMP version 3 (SNMPv3) as defined by RFC documents 3411-3418 is also known as STD0062 and represents the official version of the standard recognized by IETF. Older versions of the standard are considered to be historic or obsolete. The main improvement in this version is advanced security model based on version v2.It is important to note that there is no compatibility between different versions of SNMP protocol as the message format and the protocol itself was changed. Possible scenarios for coexistence between different versions of SNMP protocol are described in RFC 2576.

2.2. Data organization

Every network device accessible by SNMP protocol is defined by one or more Management Information Bases (MIB) – a virtual database representing a hierarchically organized set of information available for a given device. MIB consists of managed objects (MIB objects) that are uniquely identifiable in MIB hierarchy by value named object identifier (OID). MIB tree has an unnamed root node that is branched out to branches controlled by organizations in charge of standards that are further divided on lower levels of hierarchy. MIB object consist of at least one instance that can be seen as a variable or variables. There are two types of MIB objects: scalars (that define a single instance of the object) and tables (that define multiple linked instances that make up the MIB table).

One of the aims of the SNMP standard is to solve the problem of differing data representations on various platforms, a task that was solved by the use of subset of ISO OSI Abstract Syntax Notation One (ASN.1) – Structure of Management Information (SMI). SNMPv1 SMI specific data types can be either *simple* (integer, octet-string, OID) or *application-wide* (network address, counter, gauge, time tick, opaque, integer, unsigned integer).

2.3. Extending the SNMP functionality

As was previously described, SNMP allows for a flexible approach and management of networked devices, but is unfortunately limited to functionality implemented in the agent component. If one desires to access additional data or enable new functionality, there are several approaches, among which the most used are: modification of the agent, use of external programs and use of AgentX protocol.

The most efficient, but also the most difficult to implement and least flexible approach is modifying the agent to implement required functions through access to and modification of the source code of the agent in question. If it is impossible or infeasible to modify the agent, or if there is a need for several agents on the same device, solution can be obtained by the use of SNMP proxy software. Use of proxy increases the complexity of the system as the introduction of additional layer in the architecture also requires full support for all relevant requirements on this layer as well (e.g. proxy layer becomes a key component in security aspect).

Alternative solution is the use of external programs for access to required data. The simplest solution is execution of the external program every time the need for a specific data arises. This approach can have severely degraded performances as the program could be executed during any SNMP operation. Better solution is parallel execution of

both agent and external program, providing the means for communication between them. As this problem was present since the early days of SNMP, parallel to development of various *ad-hoc* solutions, a process for standardized solution of the problem was created. Result of this process is AgentX protocol [2] that is based on master-slave principle within one or more devices. This protocol is continuation of SNMP-SMUX and SNMP-DPI protocols that were relegated to historic and experimental statuses. In 1995 IETF formed SNMP Agent Extensibility Working Group [3] which defined an extension framework [2] and corresponding MIB document [4]. These documents define the protocol, master agent, sub-agents, coding of all required data types, as well as the handling of all communication between parties.

3. INTERNET OF THINGS AND MONITORING

When talking about IoT and monitoring, there are two major protocols that cannot be overlooked: CoAP (Constrained Application Protocol) and MQTT (Message Queuing Telemetry Transport). As per RFC 7252 that defines it, CoAP "is a specialized web transfer protocol for use with constrained nodes and constrained (e.g., low-power, lossy) networks" aimed at M2M (machine to machine) applications and is intended to be usable on devices with very limited processor, memory and networking resources [5]. It is UDP based and employs an adapted subset of HTTP optimized for M2M use cases, offering features not present in HTTP but highly valuable in M2M environment such as discovery, multicast support, and asynchronous message exchanges[5]. It was specifically designed to utilize insignificant processing resources in normal operation. From request perspective, CoAP messages are very similar to HTTP request methods, but are limited to GET, POST, PUT and DELETE messages that implement corresponding HTTP method functions. CoRE (Constrained RESTful Environments) link format as described by RFC 6690 [6] defines a well-known entry point ("/.well-known/core") that enables client to list the links hosted by the server and as such can be used for discovery, resource collection and resource directory and similar needs. There is an ongoing work on implementing CoAP on alternative transports such as TCP, P2P, WebSockets, ZigBee and other network protocols that would enable wider use of CoAP in IoT scenarios.

MQTT as defined by OASIS [7] is a light weight, open and simple client server oriented publish/subscribe messaging transport protocol. Like CoAP, it is aimed at use in M2M applications and resource constrained devices. It runs over TCP/IP or other network protocols that need to provide ordered, lossless and bi-directional connections (for example ZigBee protocol [8]). There is a special version of MQTT aimed at sensor networks under the name MQTT-SN that enables use of MQTT in very unreliable networking conditions by severely resource constrained devices via MQTT-SN Forwarders and MQTT-SN Gateways [9].

MQTT utilizes publish-subscribe pattern in which clients, here referred to as publishers, connect to servers (messaging brokers) and are able to send the messages to select topics with no need to specify exact recipient of the message, in this context called subscriber. Messages are filtered by their attributes, chief of which is called topic and is represented by UTF-8 string. Topics can have hierarchical organization in which different levels are separated by forward slash. An example of such topic is "building1/room007/rack02/server27/temperature". As messaging is asynchronous, topics can exist even with

no currently connected publishers or subscribers which enables for use in unreliable environments as individual nodes can connect and disconnect as the need arises. This allows for considerable flexibility as subscriber can precisely choose to listen only to messages in topics related to, for example, certain room or building, or to listen to all messages related to temperature data in all rooms or buildings.

But, IoT does not consist of constrained devices only. Fundamental to proper functioning of any IoT infrastructure is also the proper functioning of interconnecting network as well as, most often, proper functioning of back-end services, running on any kind of server device. Further complicating the things is the fact that both networking and service components of modern architectures can be virtualized. This represents a problem specially for monitoring of the performance as the NMS traditionally has access to monitoring data inside virtualized environment and performance data of actual physical device running the virtualization software is available on to infrastructure provider.

When discussing the networking component, outside of possible specialized hardware, for example MQTT-SN forwarders and similar, almost all networking devices support SNMP for monitoring. Devices that do not support it are usually unmanageable devices that provide no means for remote monitoring and are as such not suitable for use in described circumstances.

Virtualized servers running back-end services are under control of infrastructure user and can be easily configured to support SNMP monitoring if it is not already the case. As mentioned earlier, the real problem lies in the fact that the virtual machine that contains the service has no access to non-virtualized performance data of physical host. Following example illustrates the issue. Let's assume that the server running our hourly data collection service is spending proportionally large percentage of time waiting for database server to complete processing of new records. In non-virtualized situation we could monitor the processes in the system and see that, for example, we are waiting for storage system to complete the writing to disk as another process, archiving of previous data in this example, is consuming the resource at same time. This would give us enough information to solve the issue by rescheduling the offending process or decreasing the priority in order to ensure that data collection is completed properly. But, in virtualized environment, if another virtual machine is consuming resources, we have no idea that is happening, as all the performance data suggests nothing is consuming resources but they are unavailable to our service. This is but one example that illustrates how any of the limited resources on the physical host (processor, memory, networking, storage, etc) can be temporarily unavailable without having any means to determine whether the issue lies with our code or just wider environment. Fact that virtual machines can be migrated, without shutting down, from one host to another with different resources available further complicates the monitoring aspect of back-end.

3.1. Use of SNMP for monitoring the internet of things

We can divide devices we want to monitor into three categories depending on their support for SNMP. First category consists of devices that do support SNMP and provide needed monitoring data. Second category includes devices that do support SNMP but do not provide needed data directly, while the third category would be made of devices that do not support SNMP. For our needs, second and third category are essentially the same, as there is no simple way for our monitoring system to directly access the required data, whatever it may be.

First group mostly consists of devices providing network connectivity as they were usually designed to be remotely managed and monitored by SNMP. There is very little to do for us here, barring the cases where supported version of SNMP does not provide sufficient security (versions 1 and 2) or there are other reachability issues (VPN, NAT, etc). Most of these issues can be solved by using SNMP proxy services or other similar technique. Physical infrastructure in cloud environment can also be in this category, providing that we are self hosting operation or have specific arrangements with hosting provider.

There are four principal ways to gather data from devices that do not provide them in suitable form for monitoring:

1. Devices that support messaging or event notifications allow us to subscribe to relevant topics and queues or implement listeners and receive the data as it is generated by the device. This is the best approach as all the data is current and the required resources are minimal, but is limited by the support by the monitored device.
2. Polling (predefined intervals) is a simple, robust and enables us to estimate needed resources in advance. Down sides are possible monitoring of devices that are not required, risk of stale data or higher resource consumption if polling more frequently.
3. Proxying data collection as requests are made. This provides for minimal resource usage as we are collecting only the data that is needed when it is needed, but introduces unknown response delay in the system as we have to wait for all required devices to respond, makes estimates about resource usage difficult as we are dealing with, for us, random requests (example would be frequent monitoring of a slow responding device by large number of clients) and makes aggregate data calculations almost impossible.
4. Proxying with caching extends previous approach by introducing a proxy level cache that can reduce system load at the price of not returning current data to all requests and significantly increasing the complexity of the system.

Described approaches can be combined in a number of ways to create hybrid solution that would tailor to one's specific needs, again at the price of increasing already significant level of complexity.

As Lindholm-Ventola and Silverajan have shown in [10], monitoring of constrained devices using CoAP can be done by using CoAP to SNMP proxy, with or without database component, in principle corresponding to third and fourth approach described above. In their work they conclude that further work must be done on research regarding implementation of notifications in IoT monitoring systems.

Of the four described ways to monitor the devices in IoT environment, only the first approach provides for meaningful handling and generating of notifications. Remaining three approaches will either introduce a possibly significant delay in case of polling, or might completely miss the event if there were no requests to monitor the device. If a device supports messaging or can generate SNMP notifications we can process and respond to event with minimal delay.

3.2. MQTT-SNMP Bridge

In order to enable SNMP monitoring of MQTT and MQTT-SN devices, we need to implement a system that would listen to messages generated by monitored devices, if needed send requests to monitored devices and transform collected data into form suitable

for serving to SNMP clients. Although it is possible to serve standalone SNMP clients, most often setup like this are a part of larger monitoring infrastructure where SNMP clients are in fact NMSs (Network Management Systems).

Architecture of such IoT-SNMP Bridge system is presented in Figure 1. The system consists of: monitored devices either supporting MQTT or in case of severely constrained devices MQT-SN protocol, MQTT-SN Gateways and Forwarders, MQTT Broker, IOTSNMP Collector and Server and various number of SNMP clients.

MQTT-SN Forwarders and Gateways exist in configurations where there is a need to monitor MQTT-SN devices. MQTT-SN Gateways can, and usually are a part of MQTT Broker. The Broker itself should be chosen to be a polyglot type broker, enabling simple use of different messaging protocols by other endpoints in the system. Choice of a suitable broker would also enable simplifying the infrastructure of a complete system that will be described later in the text. When it comes to collecting the data and serving SNMP clients, it is possible to create monolithic system where both functions would be centralized, but by separating the collector and server we can easily scale the system or introduce additional load balancing and fault tolerance by employing multiple instances of needed service. Broker infrastructure can also be made scalable and/or fault tolerant by employing suitable broker like Apache ActiveMQ [11] that can function in both classic clustered environment as well as in a so called *network of brokers* that enables distributed queues and topics across a number of brokers.

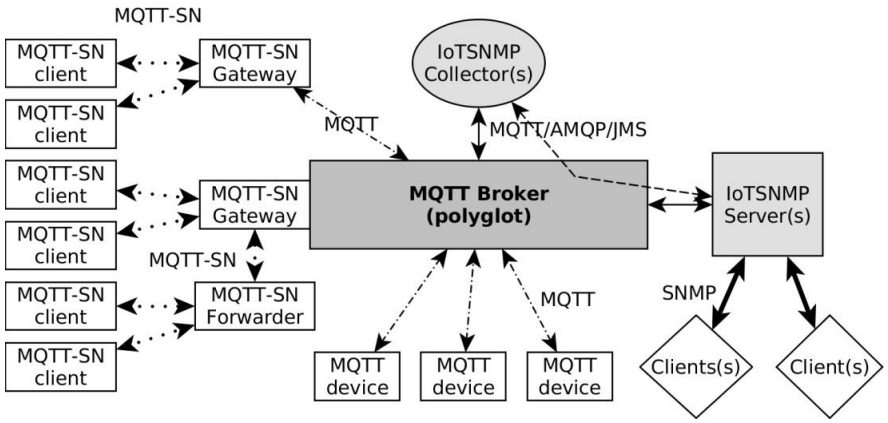

Fig. 1 Overview of IoT-SNMP Bridge

3.3. SNMP monitoring of IaaS

Development of monitoring component for IaaS in this paper is a continuation of work performed in the areas of grid computing and monitoring of distributed services started in SEE-GRID-SCI project [12] that resulted in BBmGRIDSNMP system [13] and is heavily influenced by implemented solutions. Architecture of CloudSNMP system is given in Figure 2. Data is collected from various IaaS endpoints via listening to messages generated by endpoints and sent through queue server (broker), by listening to SNMP notifications and performing SNMP monitoring of physical devices that are a part of the infrastructure as well as accessing needed information through IaaS API specific for a

given IaaS implementation or through generalized and standardized interfaces like ones produced by DMTF CMWG (Distributed Management Task Force Cloud Management Working Group) [14], ETSI (European Telecommunications Standards Institute) [15], OASIS CAMP TC (Organization for the Advancement of Structured Information Standards Cloud Application Management for Platforms Technical Committee) [16] and OGF OCCI (Open Grid Forum Open Cloud Computing Interface) [17]. Depicted queue server also supports at least one of the JMS (Java Message Service) [18] or AMQP (Advanced Message Queuing Protocol) [19] protocols.

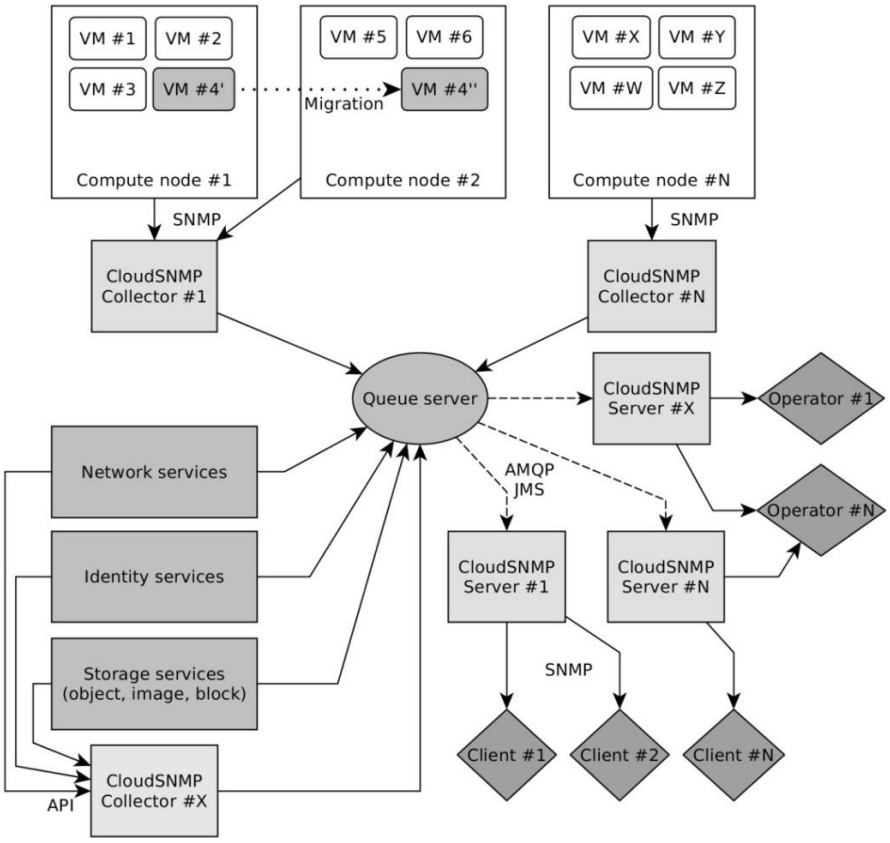

Fig. 2 Architecture of CloudSNMP (IaaS-SNMP Bridge)

Overall architecture mirror the one used in collecting and processing the data from constrained devices enabling unification of many of the components in this complex infrastructure. For example, it is possible to use the same brokers connected in load balancing and fault tolerant architecture to handle messages from both constrained devices as well as IaaS services endpoints. This also enables for sharing the code on the IoT SNMP and CloudSNMP collector and server components and further modularization of the code.

There are two principal users of served data: operator and client. Operator access should allow for full access to real monitored data and should provide for any information of interest to the operator. This can be achieved by designing and implementing a custom MIB that contains tables where rows represent monitored resources and enable the operator to easily access summary data for any required parameter. As the monitoring is already done, at least in part, by using SNMP there are existing SNMP servers with already configured access rules, thus the simplest solution is to extend their functionality by using AgentX protocol. Client access has various restrictions imposed and enables the client to access only the data relevant for a specific virtual machine, or set of individual virtual machines. This requirement mandates either the use of many instances of SNMP server, one for every monitored virtual machine, or some other mechanism that would allow for efficient access to the monitoring data.

In order to provide possibility of the client of IaaS infrastructure to access the data of the physical server hosting the monitored virtual machine, we employ SNMP contexts. In simple terms, SNMP contexts provide for creating multiple instances of data structure, be a full tree or some subset, serving the right instance to client. In our use, this enables one SNMP server to perform the function of several servers, one for each context, without unnecessary duplication of resources. As CloudSNMP server has the data from all physical virtualization servers in the infrastructure, by connecting a certain context value to a unique virtual machine, client can be served data from the correct virtualization server even after migration to another server has taken place.

Example in Figure 3 presents data propagation for a SNMP sub-tree providing data for processor, memory and basic storage statistics from physical device to CloudSNMP server to be served for infrastructure operator as an extension of existing SNMP data by utilizing AgentX protocol as well as for the client by using custom SNMP server that masks and transforms the data prior to replying to client request.

Depending on the requirements of the system, it is possible to serve different versions of data to clients, both to ensure that we are serving only the data that needs to be served and to avoid sudden changes in configuration of monitored device after migration. For example, it is possible to provide following levels of data masking:

1. No masking – served data is identical to data gathered from physical server. This enables for best performance monitoring by the client but also provides deep insight into actual configuration of infrastructure and can cause troubles for monitoring software as it is possible for a server to suddenly gain or lose CPU cores, RAM or networking interfaces.
2. Normalize to virtual machine resource – data will be normalized to maximum resources that can be occupied by monitored virtual machine. For example, if the virtual machine can utilize up to 8 CPU cores and server has 16 CPU cores, served data will be scaled to 8 CPU cores, even after migration to different server with 64 CPU cores. This provides for both limiting the amount of information we are publishing to client and for consistent measurements as the maximum values remain the same. The issue arises from the fact that it is now possible to serve data that is in collision with data recorded within the virtual machine.
3. Normalize to fixed value – any resource is to be normalized to a predefined fixed value and be seen as proportion of resource currently utilized. This hides almost all information from end users while still providing for limited performance monitoring and troubleshooting.

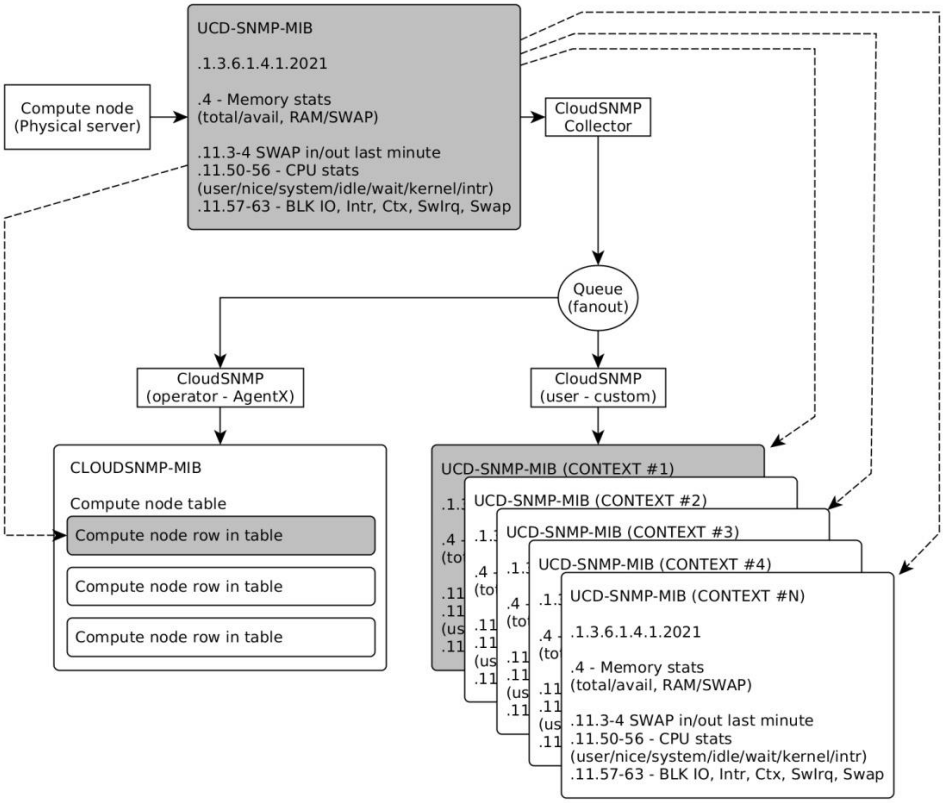

Fig. 3 Data propagation in CloudSNMP to operator and client

3.4. Overview of security aspects

While IoT promises a wealth of future possibilities in future, there are also some worrisome aspects that cannot go unmentioned, security as being the chief one. Due to pervasive nature of IoT and access to sensitive information, any compromise can have potentially grave consequences. When discussing the security of the described system, we can divide it into several possible attack surfaces: SNMP based components, messaging components and IoT components.

When discussing the security of SNMP it is important to distinguish between different versions. Versions 1 and 2c are prone to packet sniffing and other general attacks applicable to unencrypted communication. Only non-obsolete version of the protocol is version 3 that employs standard cryptographic features. Due to the limits imposed by stateless nature of the protocol, the protocol can be attacked by brute force and dictionary attacks. Modular architecture of SNMP enables use of TLS [21] and DTLS [22] within transport subsystem [23]. Proper configuration and utilization is of paramount importance in order to provide for secure operating environment.

Messaging components allow for use of complex authentication and authorization mechanisms as well as use of encryption. While this component and its security analysis lie outside of the scope of this paper, it is worth noting that there have been a number of

security vulnerabilities in various widely used SSL/TLS libraries in the past few years, affecting systems ranging from simple embedded solutions to mobile devices and dedicated servers [24][25][26][27].

Discussing security models of IoT is complicated by the nature of IoT and the fact that it covers everything from simple sensors to connected cars and vast industrial infrastructures. Examples of security issues range from vulnerabilities in widely used ZigBee protocol [28] to vulnerabilities present in connected cars [29]. Concise overview is given by Sadeghi, Wachsmann and Waidner in [30].

One of the benefits of described monitoring system is a possibility to provide effective monitoring to users of the infrastructure while limiting possible attack surfaces to exposed monitoring servers. It is also worth noting that this approach also enables the system to function as a proxy that exposes secure SNMP version 3 to outside world although the monitored devices might be able to support only insecure versions of the protocol. In a stark contrast to resource constrained devices, these servers can possess ample hardware and software resources and are much better equipped to handle possible attacks, possible through detection in cooperation with IDS (Intrusion Detection System) or mitigation when coupled with IPS (Intrusion Prevention System).

3.5. Integration with existing systems

Although there is a possibility to use specialized systems to gather, analyze and present monitoring data related to IoT, most organizations already use some form of NMS (Network Management System) that can be used for both management and monitoring of the infrastructure. There exists a vast variety of monitoring system, running on different platforms, utilizing different architectures, operational procedures and data collection methods. Some of the representatives of popular NMSs are Nagios [31], Zenoss [32], Zabbix [33] and OpenNMS [34]. One example of using Zenoss in IoT monitoring was given in [35]. Mazhelis et al have analyzed the possibilities of use of the CoAP protocol for monitoring of IoT infrastructure as well as adapting existing Accounting and Monitoring of Authentication and Authorization Infrastructure Services (AMAAIS) project [36] for such use [37]. Although NMS products can differ significantly from each other, practically all of them support at least data gathering via SNMP. This enables previously described system to extend the reach of general purpose network monitoring systems to IoT part of the infrastructure. Depending on the exact purpose and system configuration, it is possible to serve either raw collected data or data derived after previously defined transformations. This can be used to also mitigate or solve some of the privacy aspects of possibly sensitive data as the said data can be thoroughly filtered and modified to provide anonymization and/or aggregation. One example of complex monitoring system in the heterogeneous and distributed computing infrastructure such as SEE-GRID [12] was described in [13].

Developing software for systems as diverse as IoT infrastructures are can be a daunting task. Shear diversity of available devices and implementations provides for a very dynamic environment, often difficult to set up for testing purposes. While developing the system Contiki [38] based Cooja network simulator [39] can be used in place of physical devices. For testing MQTT and CoAP as well as stress testing the system simple load generator was developed in Java utilizing Californium CoAP framework [40] and Fusesource MQTT libraries [41]. Proof of concept SNMP server was first created in Java using JAX toolkit [42] utilizing AgentX protocol, but was rewritten in Python programming language [43] utilizing PySNMP library [44].

4. CONCLUSION

In this paper we presented one solution for end-to-end monitoring of IoT devices, including severely constrained devices such as sensors, IaaS installations, as well as the networking infrastructure that connects them together. On the constrained devices end of spectrum, use of CoAP and MQTT was covered, while networking infrastructure natively supports SNMP and four approaches to IaaS and virtualization equipment data gathering were presented. Integration into existing network management and monitoring systems enables simpler transition to full utilization of IoT infrastructures in practice. Often neglected aspect of harmonization of operational procedures in different domains can be significantly simplified by enabling uniform view and/or control interface for the whole infrastructure. By limiting exposed attack surfaces to simpler to manage and secure monitoring servers, security of the complete system can be increased, also alleviating some of the privacy aspects of the data gathering through the use of data transformation and anonymization prior to serving.

Described solution provides for non-blocking asynchronous data collection, scalable and fault tolerant data processing and serving, but most importantly, it provides an uniform standards based interface needed for reliable monitoring.

REFERENCES

[1] "RFC 2571 - An Architecture for Describing SNMP Management Frameworks." [Online]. Available: https://tools.ietf.org/html/rfc2571.
[2] "RFC 2741 - Agent Extensibility (AgentX) Protocol Version 1." [Online]. Available: https://tools.ietf.org/html/rfc2741.
[3] "Agent Extensibility Working Group (agentx)." [Online]. Available: http://www.ietf.org/html.charters/agentx-charter.html.
[4] "RFC 2742 - Definitions of Managed Objects for Extensible SNMP Agents." [Online]. Available: https://tools.ietf.org/html/rfc2742.
[5] "RFC 7252 - The Constrained Application Protocol (CoAP)." [Online]. Available: https://tools.ietf.org/html/rfc7252.
[6] "RFC 6690 - Constrained RESTful Environments (CoRE) Link Format." [Online]. Available: https://tools.ietf.org/html/rfc6690.
[7] "MQTT Version 3.1.1." [Online]. Available: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html.
[8] ZigBee Alliance. Zigbee specification. Technical Report Document 053474r06, Version 1.0, 2005.
[9] A. Stanford-Clark and H. Linh Truong, MQTT For Sensor Networks (MQTT-SN) Protocol Specification,. IBM, http://mqtt.org/new/wp-content/uploads/2009/06/MQTT-SN_spec_v1.2.pdf.
[10] Lindholm-Ventola, Hanna; Silverajan Bilhanan , "CoAP-SNMP Interworking IoT Scenarios," Tampere University of Technology, Department of Pervasive Computing. Report 3, Tampere, 2013.
[11] "Apache ActiveMQ." [Online]. Available: http://activemq.apache.org/.
[12] A. Balaž, O. Prnjat, D. Vudragović, V. Slavnić, I. Liabotis, E. Atanassov, B. Jakimovski, M. Savić, "Development of grid e-infrastructure in south-eastern Europe," *J of Grid Comp*, Vol. 9, No. 2, pp. 135-154, 2011.
[13] M. Savic, S. Gajin, M. Bozic, "SNMP based Grid infrastructure monitoring system," In Proceedings of the 34th International Convention MIPRO, 2011, pp. 231-235.
[14] D. Davis, G. Pilz, "Cloud Infrastructure Management Interface (CIMI) Model and RESTful HTTP-based Protocol," Technical report, Distributed Management Task Force (DMTF), 2012.
[15] "ETSI - ICT Standards, GSM, TETRA, NFV, GPRS, 3GPP, ITS, UMTS, UTRAN, M2M." [Online]. Available: http://www.etsi.org/standards.
[16] "OASIS Cloud Application Management for Platforms (CAMP) Technical Committee | Charter." [Online]. Available: https://www.oasis-open.org/committees/camp/charter.php.
[17] "Open Cloud Computing Interface" [Online]. Available: http://occi-wg.org/.

[18] M. Hapner, R. Burridge, R. Sharma, J. Fialli, and K. Stout, "Java message service," Sun Microsystems Inc., Santa Clara, CA, 2002.
[19] "Advanced Message Queuing Protocol Website" [Online]. Available at http://www.amqp.org/.
[20] "RFC 2576 - Coexistence between Version 1, Version 2, and Version 3 of the Internet-standard Network Management Framework." [Online]. Available: https://tools.ietf.org/html/rfc2576.
[21] "The Transport Layer Security (TLS) Protocol Version 1.2" [Online]. Available: https://tools.ietf.org/html/rfc5246.
[22] "Datagram Transport Layer Security" [Online]. Available: https://tools.ietf.org/html/rfc4347.
[23] "Transport Layer Security (TLS) Transport Model for the Simple Network Management Protocol (SNMP)" [Online]. Available: https://tools.ietf.org/html/rfc5953.
[24] "CVE-2014-1266" [Online]. Available: https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2014-1266.
[25] "CVE-2015-0282" [Online]. Available: https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2015-0282.
[26] "CVE-2014-0160" [Online]. Available: https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2014-0160.
[27] "Microsoft Security Advisory 3046015." [Online]. Available: https://technet.microsoft.com/en-us/library/security/3046015.
[28] "Zigbee Exploited – The Good, the Bad and the Ugly" [Online]. Available: http://cognosec.com/zigbee_exploited_8F_Ca9.pdf
[29] S. Kamkar, "Drive it Like You Hacked it"[Online]. Available: http://samy.pl/defcon2015/2015-defcon.pdf
[30] A.-R. Sadeghi, C. Wachsmann, and M. Waidner, "Security and privacy challenges in industrial internet of things", In Proceedings of the 52nd Annual Design Automation Conference, 2015, p. 54.
[31] "Nagios Core. Nagios Open Source Project.," Nagios. [Online]. Available: https://www.nagios.org/.
[32] "Zenoss," Zenoss. [Online]. Available: http://www.zenoss.com/.
[33] "Zabbix: The Enterprise-Class Open Source Network Monitoring Solution." [Online]. Available: http://www.zabbix.com/.
[34] "The OpenNMS Project." [Online]. Available: http://www.opennms.org/.
[35] U. Gupta, "Monitoring in IOT enabled devices," arXiv preprint arXiv:1507.03780, 2015.
[36] O. Mazhelis, M. Waldburger, G. S. Machado, B. Stiller, and P. Tyrväinen, "Extending Monitoring and Accounting Infrastructure Towards Constrained Devices in Internet-of-Things Applications", Technical paper, University of Zurich, 2013. Available: https://www.merlin.uzh.ch/contributionDocument/download/5076
[37] B. Stiller, "Accounting and monitoring of AAI services." SWITCH Journal, 2010(2):12–13,October 2010.
[38] A. Dunkels, B. Grönvall, and T. Voigt, "Contiki-a lightweight and flexible operating system for tiny networked sensors," In Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks, 2004, pp. 455-462.
[39] F. Osterlind, A. Dunkels, J. Eriksson, N. Finne, and T. Voigt, "Cross-level sensor network simulation with cooja", In Proceedings of the 31st IEEE Conference on Local Computer Networks, 2006, pp. 641-648.
[40] "Californium (Cf) CoAP framework - Java CoAP Implementation." [Online]. Available: http://people.inf.ethz.ch/mkovatsc/californium.php.
[41] "Fusesource MQTT libraries." [Online]. Available: https://github.com/fusesource/mqtt-client.
[42] "Jasmin: JAX - Java AgentX Client Toolkit." [Online]. Available: https://www.ibr.cs.tu-bs.de/projects/jasmin/jax.html.
[43] G. VanRossum and F. L. Drake, The Python Language Reference. Python Software Foundation, 2010.
[44] "SNMP library for Python." [Online]. Available: http://pysnmp.sourceforge.net/.