Dynamic M2M Device Attachment and Redirection in Virtual Home Gateway Environments

Apostolos Papageorgiou, Roberto Bifulco, Ernö Kovacs
NEC Laboratories Europe
Heidelberg, Germany
apostolos.papageorgiou@neclab.eu, roberto.bifulco@neclab.eu, ernoe.kovacs@neclab.eu

Hans-Jörg Kolbe
Deutsche Telekom AG
Darmstadt, Germany
Hans-Joerg.Kolbe@telekom.de

***Abstract*—Virtualized access to M2M (Machine-to-Machine) devices can offer to operators various benefits in terms of hardware and management costs. However, it is often impeded by the inability to efficiently handle the traffic from the devices towards servers that are appropriate to serve as virtual hosts for the given devices. In this paper we present a solution which can be applied in a virtual Home Gateway environment and is based on (i) encoding information about the device and its virtualization technology into TCP/IP packet headers, (ii) selecting virtual hosts based on their compatibility to certain virtualization technologies, and (iii) using Software-defined Networking to make this virtual host selection at the network level in a dynamic manner. Compared to the main alternatives, which are based either on Deep Packet Inspection or on homogeneous servers, our solution is the only one that combines a tailored and non-costly server infrastructure with low networking complexity.**

I. Introduction

The number of Internet-connected devices is significantly growing in the last years, with already 9 billion units in 2012 and an expected growth to more than 24 billion units by 2020 [4]. The possibilities enabled by such broad deployments are fueling the creation of a number of new services for automation, energy control, entertainment, ambient-assisted living, and more. In several cases, different Machine-to-Machine (M2M) devices are attached to a gateway device (GW) which is in turn connected to the Internet, as shown in Figure 1. The attachment of devices is performed over either a wired or a wireless channel, using one (or more) of the so-called M2M Area Network Technologies, e.g., USB, Ethernet, ZigBee, WiFi, Bluetooth, UPnP, DECT. This setting is typical in home networking, but also in similar scenarios such as buildings, facilities, or small outdoor areas.

The big number of devices and the increasing variety of M2M area network technologies require the GWs to become quite complex, comprising full-fledged networking capabilities, operating systems, drivers, and sophisticated mechanisms. Furthermore, GWs should support reconfiguration, usually through remote management, in order to enable the deployment of new devices and applications, while meeting user expectations about the devices plug-and-play fashion.

The complexity of such GWs raises a number of concerns, in particular among operators who usually deploy GWs as part of their subscription plans. First, a complex GW has a usually higher cost than simpler devices [1]. Network operators are left with the options of either affording a higher deployment cost, reducing their revenues, or transferring the cost to their subscribers, increasing subscription fees. Second, complex devices are likely to increase maintenance costs, since several different components may eventually fail [1]. For instance, a malfunctioning GW may require the expensive intervention of on-site technicians and/or the deployment of replacements. Finally, the troubleshooting complexity increases, making more difficult the effective detection of problem causes. In turn, one of the already mentioned expensive maintenance actions may be triggered, even if not actually required.

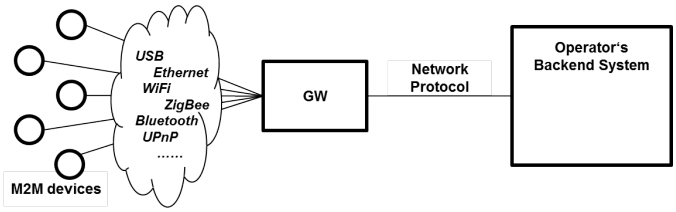


Fig. 1. Traditional GW-based M2M deployment

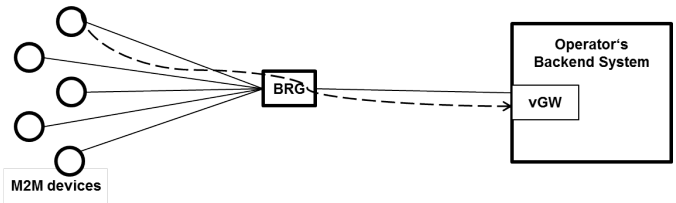


Fig. 2. vGW-based M2M deployment

The telecommunications industry has started trying to face these issues by virtualizing the GW device, i.e., replacing it with a much simpler device, which is called Bridged Residential Gateway (BRG) or M2M anchor (if it is only targeted to M2M devices) [11], [2]. These devices simply forward the traffic to the telecommunication operators core network. All the protocols, drivers, and networking functions that are needed for the communication with the M2M devices are implemented in a Virtual Gateway (vGW), which resides on the core network side, as shown in Figure 2. To seamlessly support this communication scheme, one of the employed technologies is *protocol virtualization*, which enables the attachment of a device's Local Area Network technology card to a remote operating system, as if the device was directly attached to the remote system. With protocol virtualization

the low level card control is handled on the local device, but the remote operating system drivers control the M2M devices attached to the card. The communication with the remote operating system happens using the same network that provides Internet connectivity.

Since the M2M device physical attachment point is now decoupled from the GW's operating system, it is possible to dynamically change the operating system that handles the device, opening the field to features such as broader device compatibility and better system scalability. For example, a device with a driver that is available only for GNU/Linux may be attached to a vGW instance that runs in such an operating system. Likewise, a user may be associated with multiple vGW instances, to handle a higher number of devices without degrading performance, or for wider support of M2M devices (e.g., in case the vGWs run different software).

To achieve such a flexible management of the M2M devices attachment procedures, in this paper we present a solution that addresses the underlying issues of *M2M device technology identification*, *compatible M2M server selection*, and *M2M device traffic redirection*. We believe that solving these issues provides a building block to implement more complex M2M device management strategies. We develop our system assuming that the network operator adopts a Software-defined network [9], which provides us with a tool to handle device redirection directly at the network level. To the best of our knowledge, our proposal is the first solution for *GW virtualization* via *M2M device protocol virtualization*.

The rest of this paper is organized as follows: Section II explores related works, explains the addressed challenges and provides background information, while Section III describes the details of our solutions and presents examples of its operation. Finally, Section IV provides a short qualitative evaluation by comparing features of our solution with current approaches for handling automated remote device mounting. Section V concludes the paper.

II. Background, related work and challenges

In this section, we provide an overview on Software-defined Networking (SDN), as it constitutes part of our proposed solution. Then, since there is no directly comparable solution in the state of the art, we discuss the current application of protocol virtualization. Finally, we explain the challenges of applying this technology to the GW virtualization case.

SDN. Software-defined networking is a network architecture in which the forwarding devices, i.e., the network switches, are programmed by a logically centralized entity, typically called *controller*. Any forwarding decision is taken by the controller, which in turn programs the switches with the required forwarding rules that enforce such decision. A forwarding rule is composed of a flow identifier and a set of forwarding actions. The flow identifier includes a set of network packet header's fields, such as IP source and destination addresses, transport protocol's source and destination ports, etc. The forwarding actions include the indication of a switch's output port and the setting of some packet header's fields values. Whenever a switch receives a network packet, it checks the forwarding rules until it finds a rule's flow identifier that matches the packet header's fields values. When such rule is found, the packet is forwarded according to the rule's forwarding action, otherwise it gets dropped. The network controller is usually implemented in software, which enables an easy extension of the network behavior. For our solution, we assume that the implemented SDN technology is OpenFlow [8]. OpenFlow is a standard architecture and protocol for SDN that has been already deployed in a number of production networks [7].

Protocol virtualization. Relevant applications of protocol virtualization are presented in [5], which introduces a virtual peripheral bus driver technology for sharing access to USB-attached devices in an IP network. In [10], the author discusses further engineering details of "USB port remoting". Actually, the concept of using a network, specifically based on IP, for "Network-Attached Peripherals" [6] is almost two decades old, though it has been mainly used for memory disks [3] and terminals rather than for M2M devices.

Challenges. Protocol virtualization is typically applied for simple remote access of users to their devices upon a statically pre-configured system. Its extensive usage is subject to a number of restrictions, which impede its application in the context of GW virtualization:

1) The selection of the backend server is pre-configured.
2) The selection of the backend server is static.
3) The device's communication flow cannot be easily identified inspecting the network flows

In a GW virtualization scenario, the network operator infrastructure does not know which backend servers can handle the technology of newly appearing M2M devices, furthermore, a device communication flow cannot be detected in a network flow. Considering that a GW virtualization solution should be able to dynamically redirect devices to different vGW instances, there is a clear mismatch between state of the art protocol virtualization and our requirements.

To understand the issue, assume that two different devices are attached to the same M2M anchor. The drivers to mount the first device are provided for GNU/Linux operating systems, while those for the second device are supported only on Microsoft Windows. The M2M anchor needs to contact the right backend server(s) when performing the protocol virtualization to finally mount the devices. In this case, several issues arise. First, the M2M anchor does not know the capabilities of the backend servers. Second, the M2M anchor does not know about the specific device attached and its requirements, since this can only be discovered by establishing a conversation with the device using its drivers, which are now located remotely on the backend server. Third, the network operator lacks any information on the identity of a given device so that the corresponding network flows (i.e., generated by the protocol virtualization procedure) can be directed to a given server using a redirection in the network operators premises.

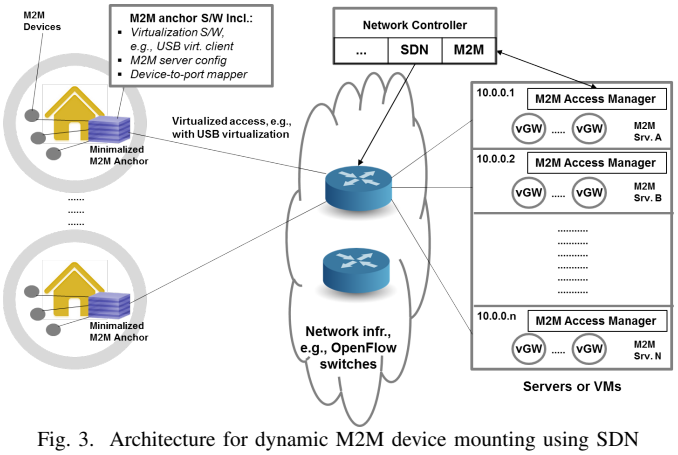


Fig. 3. Architecture for dynamic M2M device mounting using SDN

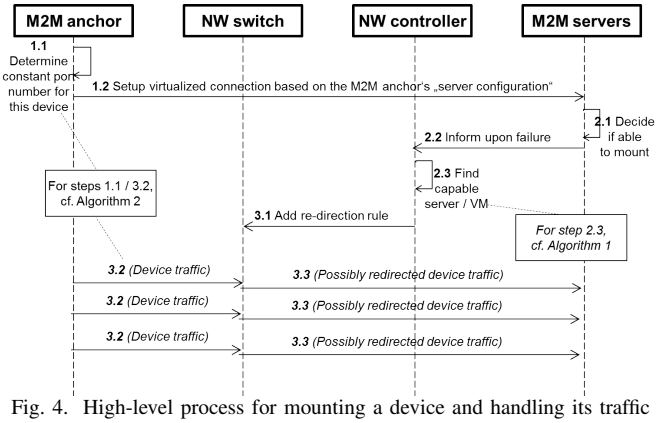


Fig. 4. High-level process for mounting a device and handling its traffic

III. Protocol-independent remote M2M device mounting based on SDN

In this section we present our solution introducing the designed architecture and the flow of operations. Then, we present an example of operations and present two algorithms for server selection and multi-protocol support.

Our solution builds on existing M2M technology virtualization protocols for mounting a device to a remote computer. We enhance the state-of-the-art by using this kind of virtualization together with gateway virtualization, combining it with M2M server selection and traffic redirection techniques, and orchestrating all the aforementioned mechanisms inside an operators network infrastructure in order to achieve efficient M2M device virtualization.

A. Architecture

We defined the following entities (cf. Figure 3):

M2M Anchor: An M2M device physical attachment point. The M2M Anchor is in charge of virtualizing the access to the devices. We introduce a *device attachment logic* that is coupled with the M2M Anchor in order to encode a temporary device identifier, along with additional information, into the headers of the packets belonging to the network flows generated by that given device.

M2M Server: The M2M server is where the M2M devices are mounted; hence it is the destination of the network flows the M2M Anchor generates as part of the process for the virtualization of an M2M device. After mounting a device, the M2M server provides its functions and data to the application layer, using an ad-hoc driver/software stack. We enhance the M2M Server by introducing an *M2M Access Manager* entity, which interacts with the M2M server software stack, in order to provide notifications about the current device status to external entities.

Software-defined network: We assume that the operators network is built using SDN technologies. As such, the network is composed of Forwarding Elements that expose an interface for their configuration to a Network Controller. The Network Controller includes an *SDN flow management entity* which performs the network configurations required to properly steer network flows. Furthermore, inside the Network Controller, we introduce an *M2M control logic* that is able to interact with the M2M Access Manager. Furthermore, the Network Controller exploits the device attachment logic adopted by the M2M Anchors in order to manage the network flows and steer a selected device towards the proper M2M server.

B. Flow of operation with example steps

The entities of Figure 3 coordinate according to the flow of Figure 4 in order to mount a new M2M device and handle its traffic. The main steps unfold as follows:

Device attachment:

- A *device is attached to the M2M anchor*. The M2M anchor starts the M2M device protocol virtualization. This procedure involves the creation of network packets used to communicate with the M2M server where the device will be then mounted. These *packets will be tagged with an unambiguous value in the packets header* (e.g., by using pre-specified port ranges as shown in Figure 5 and explained later). The tagging will be in place until the device is disconnected. That is, only in case of a disconnection and subsequent new connection to the M2M Anchor, the same device may acquire a different tag for the network packets. Although variations are possible, Algorithm 1 will clarify how the M2M anchor handles the traffic of a newly attached device.
- The network packets are sent *from the M2M anchor to the M2M server*. This sequence of packets is hereinafter referred to as the *M2M device network flow*.
- The network forwards the network flows by examining the values in the packet headers; the *forwarding rules are installed by the network controller*, which can use the tag in the network packets to extract high level information about the M2M device, according to the tagging policy implemented by the M2M anchor.

Mounting negotiation and server selection:

- The M2M server (which is selected according to a logic illustrated in Figure 6 specified later in this document) attempts *to mount the device*. At the end of the mounting

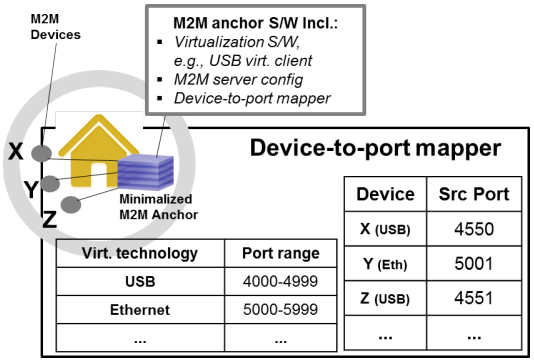


Fig. 5. Device attachment logic of the M2M anchor

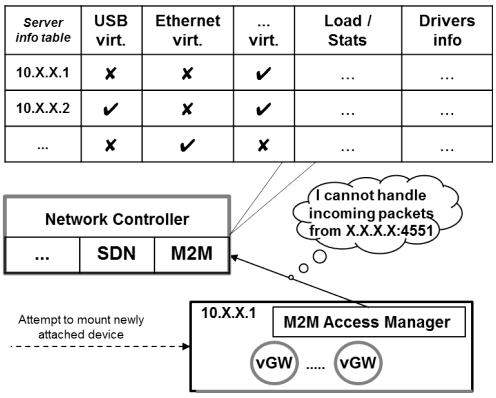


Fig. 6. Troubleshooting and negotiation of device mounting and M2M server selection

process, the software stack of the M2M server informs the M2M access manager if the mounting was successful or not, moreover, the M2M access manager is provided with the current tag used by the packets in the M2M devices network flow.

- The *M2M access manager forwards this information to the M2M control logic* in the Network Controller.
- The M2M control logic can start a process to decide if a *redirection* of this M2M device towards a different M2M server is required. Although variations are possible, Algorithm 2 will clarify how the virtualization-aware server selection process is performed.

SDN-based configuration of M2M traffic:

- In case the redirection is required, the M2M control logic instructs the SDN flow management entity *to redirect the M2M devices network flows towards a different M2M server* (see Figure 7). The M2M device network flows can be identified using the information provided by the M2M access manager to the M2M control logic. This information, paired with the M2M anchor device attachment logic, ensures the unambiguous identification of the flow. The new destination M2M server is provided by the M2M control logic at the end of the redirection decision process.

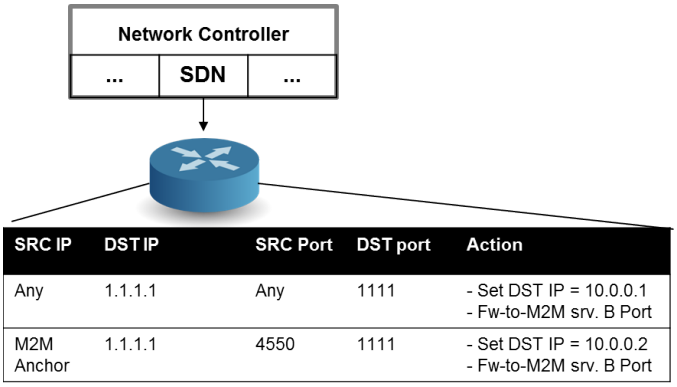


Fig. 7. SDN-based configuration and re-direction of M2M traffic

TABLE I
CONFIGURATION PARAMETERS AND VARIABLES OF THE EXAMPLE

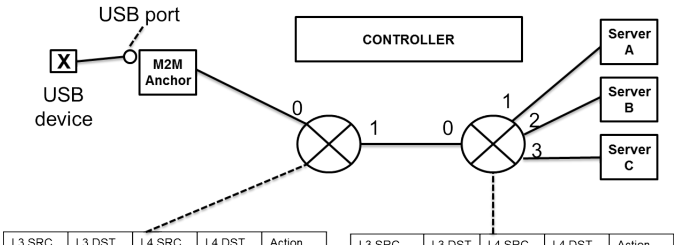
USB technology L4 port range	22000-22500
Default M2M server in M2M anchors configuration	1.1.1.1:1234
M2M anchor IP address	1.1.1.100
USB enabled M2M servers	A, B
M2M servers that can support device X	B

C. Summary of complete device mounting example

In this paragraph we summarize a multi-phase operation example, assuming a scenario in which an M2M device (Device X) is attached to an M2M anchor using a USB port.

A scenario that goes through the different phases of the solution (device attachment, discovery, and re-direction) is described in the following, using the example configuration parameters and variables that are summarized in Table I:

1) Device X is attached to the M2M anchor.
2) M2M anchor assigns L4 port 22001 (included in the USB port range) to Device X.
3) M2M anchor starts protocol virtualization, contacting server 1.1.1.1:1234.
4) The first switch on the path forward the packets to the next switch following the configuration of its forwarding entries (cf. Figure 8).
5) The second switch on the path forwards the packets towards server A, since the L4 source port is in the range 22000-22500 (i.e., the USB technology L4 port range, for which server A is designed as handler; cf. Figure 8 ignoring the red rule which has actually not been added yet).
6) Server A is unable to handle device X.
7) Server As M2M Access Manager informs the Controller about this issue.
8) The Controller installs a new flow table entry in the second switch to redirect Device X network packets towards a different server, which is likely to be able to mount the device (cf. the highlighted entry in Figure 8).
9) Network packets related to Device X are now forwarded to server B that can handle the device.
10) Device X is finally mounted at server B.



L3 SRC	L3 DST	L4 SRC	L4 DST	Action
*	1.1.1.1	*	*	Fwd-to: 1

L3 SRC	L3 DST	L4 SRC	L4 DST	Action
1.1.1.100	***1.1.1.1***	***22001***	***1234***	***Fwd-to: 2***
*	1.1.1.1	22000-22500	1234	Fwd-to: 1
*	1.1.1.1	23000-23500	1234	Fwd-to: 3

Fig. 8. Example scenario of M2M device attachment, discovery, and re-direction

D. Algorithm for server selection

A core feature of our solution is the fact that the M2M server that is chosen to host the virtual device (i.e., to receive the packets of the channel that is used by the virtualization program for this device) is selected based on its virtualization support features, i.e., based on its compatibility to the used virtualization technology, its installed drivers, and further parameters related to device virtualization. The logic of the M2M module of the network controller, i.e., the concrete *server selection algorithm*, can vary. This algorithm can take into account additional information, e.g., its ability to understand and process data of devices with concrete identifiers, its current load, and more, as already implied in Figure 6. A variant for the server selection logic is specified in Algorithm 1.

Algorithm 1. M2M controller logic for performing server selection based on information about virtualization technologies

```
DEFINITIONS

// rejectorIP:     The IP of the server which just rejected
//                 handling a packet from the M2M anchor
// sourcePort:     The source port of the rejected (see above) packet
// portRangeMap:   A table containing one <virtualizationTechnology, portRange>
//                 entry for each virtualization technology known by the system
//                 (e.g., bottom left table of Figure 5).
// serverTable:    A table containing one entry for each server
//                 (e.g., upper table of Figure 6).
// serverList:     List of (server) IPs to be given to SDN controller
//                 for trying to re-direct to one of them
------------------------------------------------------------------------------
// Inputs: rejectorIP, sourcePort, portRangeMap, serverTable
// Output: serverList

serverList = {};
for (virtualizationTechnology : portRangeMap)
   if (sourcePort is in portRangeMap.getValue(virtualizationTechnology))
      requiredTech = virtualizationTechnology;
      break;
   end if
end for
for (server : serverTable)
   if (!server.supports(requiredTech)) continue;
   if (server.IP == rejectorIP) continue;
   if (server.load > loadLimit) continue; // loadLimit is a preset constant
   // if (...) continue; (any further constraints,
   // e.g., with regard to drivers or past behavior)
   serverList.add(server.IP);
end for
return serverList;
```

E. Algorithm for multi-protocol support

Another core feature of our solution is the usage of port ranges as identifiers of the used virtualization technology. The previously described server selection logic enhances the solution thanks to the fact that information about the used virtualization technology are "implicitly included" in the header of the packets sent by the M2M anchor. Although there might be different ways to encode this information, we suggest the use of a different outgoing port (of the M2M anchor) for the traffic of each different M2M device, in addition to the mapping of certain port ranges to certain virtualization technologies. A variant for the M2M anchor logic that performs this is specified in Algorithm 2. For example, as illustrated in Figure 5, the packets that contain data from a device that uses USB virtualization could be sent by the M2M anchor to the backend using an outgoing port between 4000 and 4999. Note that the IP addresses of the M2M servers are anyway normally hidden from the M2M anchor. The latter might know and use a single IP. Thus, the information encoded in this port number can even be used for pre-selection, i.e., for choosing the first server that will attempt to mount a device that has just appeared.

Algorithm 2. M2M anchor logic for handling device messages by encoding virtualization technology information in IP packet headers

```
DEFINITIONS

// deviceMessage:  A piece of information in a technology-specific format
//                 (e.g., Ethernet packets) that the corresponding
//                 virt. client (e.g., Ethernet virtualization client)
//                 on the M2M anchor transforms into an IP packet for
//                 sending to the virtualization server.
// deviceMAC:      The MAC address of the device sending the deviceMessage.
// devicePortMap:  A table containing one <deviceMAC, anchorOutgoingPort>
//                 entry for each attached M2M device.
// techPortMap:    A table containing one <virtualizationTechnology, firstPort>
//                 entry for each virtualization technology (e.g., USB)
//                 supported by the M2M anchor.
// outPacket:      A TCP/IP packet to send to the server
//                 (as part of the virtualized M2M connection)
------------------------------------------------------------------------------
// Inputs: deviceMessage, deviceMAC, devicePortMap, techPortMap
// Output: outPacket

// The first line is not described in detail and is a common task
// of virtualization clients, i.e., encapsulating
// technology-specific pieces of information into IP packets.
outPacket = getPacketFromVirtualizationClient(deviceMessage);
outPacket.destinationIP = M2Mserver.IP; // preset, then maybe
                                        // translated by SDN switch.
outPacket.destinationPort = M2Mserver.port; // preset
outPacket.sourceIP = M2Manchor.IP;
if (devicePortMap.contains(deviceMAC))
   outPacket.sourcePort = devicePortMap.getValue(deviceMAC);
else
   tech = identifyTechnology(deviceMessage);
   count = currentCounter(tech); // global counter holding the number
                                 // of already attached devices using
                                 // this virtualization technology.
   nextPort = techPortMap.getValue(tech) + count;
   outPacket.sourcePort = nextPort;
   currentCounter(tech) = currentCounter(tech) + 1;
end if
return outPacket;
```

IV. Qualitative Evaluation

Due to the broad scope of our solution and the various sub-systems and sub-mechanisms that could be evaluated, we opted for a lightweight qualitative evaluation, in which we analyze the *complexity*, *costs*, and *networking efficiency* of our solution compared to what can be achieved by the main alternative technologies, namely based either on DPI (Deep Packet Inspection) or on server homogeneity. It is assumed that the operator desires a vGW deployment (cf. Figure 2 because of the advantages it offers in terms of costs and control, so that solutions that rely on "thick" gateways are not considered.

The comparison is presented in Table II, in which our solution is referred to as "Capability-based using SDN" while the alternatives are "DPI-based" and "Homogeneous cluster". In the DPI-based solution, no port mapping is used, but network packet contents have to be analyzed to identify the M2M protocol and the origin device, and infer server requirements.

TABLE II
COMPLEXITY, COSTS, AND NETWORKING EFFICIENCY OF OUR SOLUTION COMPARED TO ITS MAIN ALTERNATIVES.

Criteria / Approaches	**Complexity**	**Costs**	**Networking Efficiency**
Capability-based using SDN	+ No network packet contents need to be analyzed. + Protocol and device identification based on fixed port mappings have straightforward implementation. - The Controller needs to implement additional logic for server selection.	+ Can exploit the existing server infrastructure. + Only cheap M2M anchors required at the customer premises - Transition to SDN technology required for the network.	+ No CPU-intensive tasks for networking equipment, i.e., switches. - Traffic re-directions can increase total load.
DPI-based	+ Slightly less logic required on the M2M anchor (no port mappings). - Knowledge of the specifics of each protocol is required on all switches in order to perform the DPI. - Many and complex redirection rules will be needed if they are based on DPI insights.	+ Can exploit the existing server infrastructure. + Minimal M2M gateway (BRG) required at the customer premises. - Transition to SDN (or other programmable network) required. - Complex programming for networking equipment (for DPI) can imply high development costs.	- CPU-intensive tasks for networking equipment, i.e., switches. - Traffic re-directions can increase total load.
Homogeneous cluster	+ No M2M controller must be implemented. + No server selection or redirection logic must be implemented.	+ Only cheap M2M anchors required at the customer premises. + Existing network can be exploited. - Homogenization of server infrastructure might imply big hardware and deployment costs.	+ No CPU-intensive tasks for networking equipment. + Bandwidth efficiency due to hardware switching and no re-directions. - Danger of many unsupported devices. - No option for routing enhancements as in software-programmable networks.

With an homogeneous cluster, there is no need for server selection and packet redirection, because the operator makes sure that all servers run the OS and the drivers that are required to mount devices of all supported virtualization technologies.

Obviously, each approach has advantages and disadvantages. However, our solution opens up new perspectives, especially for operators (or other M2M providers) that either do not want to change their server infrastructure or really need to exploit the flexibility and optimization that can be achieved by having heterogeneous, tailored, and specialized servers (or VMs). All in all, it is the only solution that combines a tailored and non-costly server infrastructure with low networking complexity.

V. CONCLUSION

This paper has presented a solution for attaching virtualized M2M devices to appropriately selected M2M servers by using an M2M anchor with a special device attachment logic, virtualization-aware M2M servers, and a network controller which includes an SDN controller and an M2M controller. All together, they enable the selection and usage of appropriate servers to act as virtual hosts of M2M devices. This is achieved seamlessly, by combining the encoding of information about the virtualized protocol inside L4 packet headers with the usage of respective rules in the intermediate switches, as well as with M2M server selection functions.

The above features make our solution unique and partly advantageous compared to alternative approaches. Compared to the main alternatives, which are based either on DPI or on homogeneous server clusters, our solution is the only one that combines a tailored and non-costly server infrastructure with low networking complexity.

REFERENCES

[1] T. Cruz, P. Simoes, N. Reis, E. Monteiro, F. Bastos, and A. Laranjeira. An Architecture for Virtualized Home Gateways. In *2013 IFIP/IEEE International Symposium on Integrated Network Management (IM 2013)*, pages 520–526, May 2013.

[2] Ericsson. Virtual CPE and Software Defined Networking. http://www.ericsson.com/res/docs/2014/virtual-cpe-and-software-defined-networking.pdf.

[3] G. A. Gibson and R. Van Meter. Network Attached Storage Architecture. *Communications of the ACM*, 43(11):37–45, Nov. 2000.

[4] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami. Internet of things (iot): A vision, architectural elements, and future directions. *Future Gener. Comput. Syst.*, 29(7):1645–1660, Sept. 2013.

[5] T. Hirofuchi, E. Kawai, K. Fujikawa, and H. Sunahara. USB/IP - A Peripheral Bus Extension for Device Sharing over IP Network. In *Proceedings of the FREENIX Track: 2005 USENIX Annual Technical Conference, April 10-15, 2005, Anaheim, CA, USA*, pages 47–60, 2005. Awarded FREENIX Track Best Paper Award.

[6] S. Hotz, R. V. Meter, and G. Finn. Internet Protocols for Network-Attached Peripherals. In *Proceedings of the sixth NASA Goddard Conference on Mass Storage Systems and Technologies in Cooperation with Fifteenth IEEE Symposium on Mass Storage Systems*, 1998.

[7] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Hölzle, S. Stuart, and A. Vahdat. B4: Experience with a Globally-deployed Software Defined Wan. *SIGCOMM Comput. Commun. Rev.*, 43(4):3–14, Aug. 2013.

[8] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. Openflow: enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, 2008.

[9] B. Nunes, M. Mendonca, X.-N. Nguyen, K. Obraczka, T. Turletti, et al. A survey of software-defined networking: Past, present, and future of programmable networks. *Communications Surveys & Tutorials, IEEE*, 16(3):1617–1634, 2014.

[10] R. Roshan. Remote USB Ports, December 2013. Master Thesis at Texas A&M University.

[11] Telefonica. Telefonica and NEC start the first virtual customer premises equipment trial in Brazil. http://pressoffice.telefonica.com/documentos/nprensa/131010/_Telefonica_Eng_final.pdf.