Performance Assessment of Integration in the Cloud of Things via Web Services

Stamatis Karnouskos, Veselin Somlev
SAP Research, Karlsruhe, Germany
Email: {stamatis.karnouskos,veselin.somlev}@sap.com

***Abstract*—The last years an increasing number of "things" acquires networking capabilities and therefore can be connected to the global Internet infrastructure. The machine to machine (M2M) interactions combined with business processes empower several new innovative applications, changing the way we interact with our physical environment. With the emergence of trends such as cloud computing, it is expected that these cyber-physical systems will harness its benefits such as resource-flexibility, scalability, etc. and not only enhance their own functionality but also enable a much wider consumption of their own data and services. Internet technologies and more specifically web services are expected to play paramount role towards creating this "Cloud of Things", as most of the coupling is expected to happen via them. We present here some initial thoughts on the "Cloud of Things" and an assessment of example web services and protocols such as REST, DPWS, CoAP, etc. that could be used to realise the vision.**

I. Towards the Cloud of Things

Machine to Machine (M2M) is a key area in the emerging Internet of Things where billions of devices will need to interact with each-other and exchange information in order to fulfil their purpose. Much of this communication is expected to happen over Internet technologies [1] and tap to the extensive experience acquired with architectures and experiences in the Internet over the last decades. More sophisticated, though still overwhelmingly experimental approaches, go beyond simple communication integration and target more complex interactions where collaboration of devices and systems is taking place [2].

There cross-layer interaction and cooperation is pursued:

- at M2M level where the machines cooperate with each-other (machine focused interactions), as well as
- at machine to business (M2B) where machines cooperate also with network-based services and business systems (business service focus)

As depicted in Figure 1, we can see several devices in the lowest layer, which can communicate with each-other over short-range protocols e.g. over ZigBee, Bluetooth, or even longer distances e.g. over Wi-Fi etc. Some of them may host services e.g. REST services, and even have dynamic discovery capabilities based on the communication protocol e.g. WS-Eventing in DPWS. Some of them may be very resource constrained which means that auxiliary gateways could provide additional support such as mediation of communication, protocol translation, etc. Independent if the devices are able to discover and interact with other devices and systems directly or via the support of the infrastructure, M2M interactions enables them to empower several applications and interact with each-other in order to fulfil their goals.

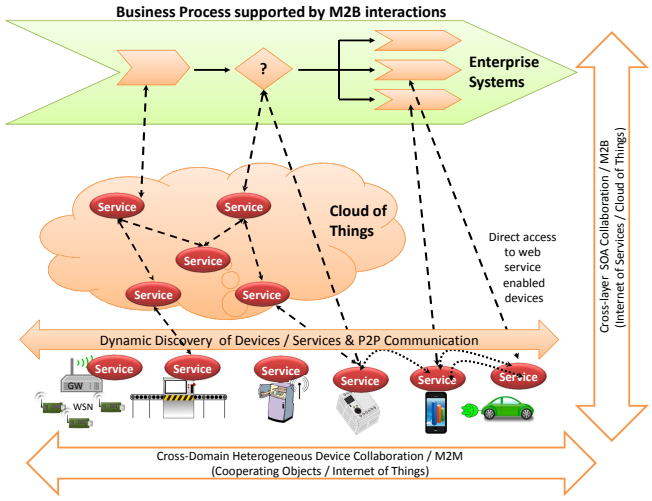

Figure 1. A collaborative infrastructure driven by M2M and M2B

Promising real-world integration is done via service oriented approaches, by interacting directly with the respective physical elements e.g. via traditional web services running on devices (if supported) or via more lightweight approaches such as REST. In the case of legacy systems, gateways and service mediators are in place to enable such integration challenges. However in the era of sophisticated networked embedded devices, open interactions and virtualized resources, a dilemma is emerging about which functionalities can be migrated out of devices and out-of-premise, to the public or private cloud with all the benefits and challenges it poses. This is not an easy decision to take, and in practice we expect an amalgamation of them to occur, depending on the concrete scenario and application domain's requirements.

Many of the services that will interact with the devices are expected to be network services available e.g. in the cloud. The main motivation for device-dependent enterprise services, is to take advantage of the cloud characteristics such as virtualization, scalability, multi-tenacy, performance, lifecycle management etc. Similarly we expect to see that a large number of devices and generally cyber-physical systems will make their functionality available on the cloud or enable virtual proxies of the devices to exist there.

A key motivator is the minimization of communication overhead with multiple endpoints by e.g. transmission of data to a single or limited number of points in the network, and letting the cloud to do the load-balancing and further mediation of communication. For instance, as depicted in Figure 2, a Content Delivery Network (CDN) can be used in order to get access to the generated data from locations that are far away from the M2M infrastructure (geographically, network-wise etc.). To this end, the data acquired by the device can be offered without overconsumption of the device's resources, while in parallel better control and management can be applied. Typical examples include enabling access to the full historical data, preprocessing of information, transparently upgrading the cloud services, or even not providing access to internal systems for security reasons. This clear decoupling of "things" and the usage of their data is expected to further empower information-driven applications that can operate over federated infrastructures.

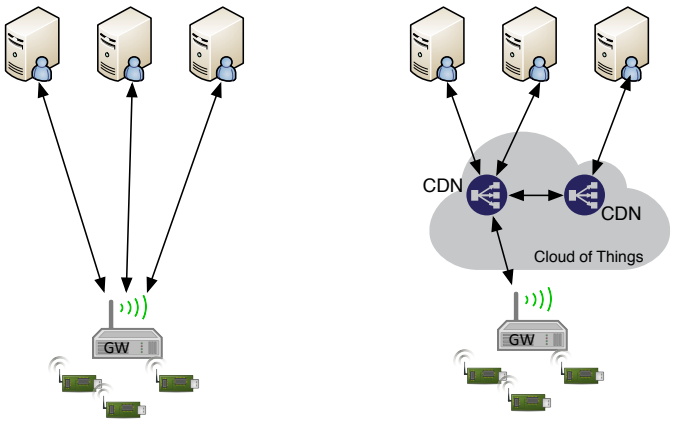

Figure 2. Enabling the things to harness the cloud benefits

For some domains e.g. in industrial automation, timely access to monitoring and control functions is of high importance, depending on the requirements the application poses. For instance the "cloud of things" may be used to empower the next generation of SCADA/DCS systems [3] in conjunction with several services that may be hosted on the devices, in gateways and systems, in the cloud as well as cross-layer compositions and interactions among them. For many of these, reliability and high performance interactions are needed, which poses the problem of finding the equilibrium of computation, communication, resource optimisation, openness and user-friendliness in the interactions between the different systems, devices, etc.

Within this work we assume that each device or system (generally each "thing"), can be empowered with web services either directly (the device is powerful enough to host them locally) or indirectly (the services are provided by a gateway or any other device they are attached to). These services can be accessed directly by applications, systems and other services independent of where they reside e.g. the cloud as depicted in Figure 3. The assessment and the follow-up discussions

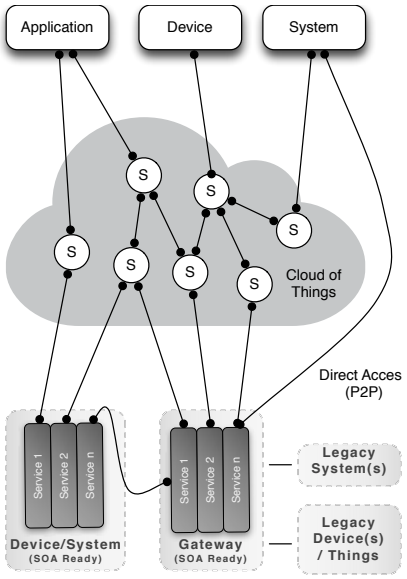

Figure 3. General view of a SOA-ready infrastructure with M2M service proxying/mediation and composition in the cloud

assume this web service based interaction under different conditions.

II. Experiments and Evaluation

Several technologies used in enterprise world will be used also in the future Internet to realize the partial delegation of functionalities in the cloud. Hence we look here at some performance benchmarks of them, to see what one should expect and for which aspects these might be fit. We experiment with the latest integration technologies i.e. traditional web services (with Axis2), Devices Profile for Web Services (DPWS), REpresentational State Transfer (REST), and Constrained Application Protocol (CoAP), provide some initial performance view and discuss on their suitability for future distributed (on-device and in-network) system functions.

The interest is primarily in measuring the performance of a service implemented with various technologies. Although different technology stacks were used, and some differences in the interactions were unavoidable, our interest was to make some real-world experiment in getting some notion of the performance of the service and also do a comparative analysis. To this end, we have measured mainly serialization, transmission Round Trip Time (RTT) and deserialization time. An initial overview of the results is given in Table I, while the details of the tests and discussion of the results per individual case as well as a comparative analysis follow in the next sections.

A. Test Environment and Experiments

For all experiments, Java was used as a common implementation language, hence our tests have used only stacks available for Java (this is a known limitation as other stacks e.g. in C may perform significantly better). All the test scenarios employ a server-client model, with the server and the client

applications running in the fixed corporate LAN. The effects of running the test cases over more unreliable links such as a wireless network is left for further research; however this is expected to have an impact only on the transmission RTT. The latter is highly dependent on the network setup and distance between the client and the server; hence if the client contacts a cloud service several hops away and with the security systems (such as firewall, intrusion detection system etc.) intervening, this can vary significantly.

Table I
AVERAGE MEASURED TIMES FOR ALL TESTS

	Serialization	*Transmission RTT*	*Deserialization*
Axis2	54340 ns	12221588 ns	167866 ns
DPWS	297461 ns	8467042 ns	262764 ns
REST	20019 ns	1064523 ns	60482 ns
CoAP	5296 ns	5996933 ns	7600 ns

The exchanged data between the client and server applications is a simple message with several value fields – strings, integers, floating point, etc. which is suitable mostly for monitoring purposes where e.g. a ICS transmits its data at high frequency. How the different frameworks handle messages of varying complexity is beyond the scope of this work. For all experiments depicted, 10000 individual messages were transmitted among the client and the server.

B. Traditional web services (with Axis2)

Traditional web service integration is done with SOAP based message exchange. Apache Axis2 (http://axis.apache.org/axis2/java/) is a modular stack that supports a wide range of WS-* protocols and provides the capability to add web services interfaces to web applications while it can also act as a standalone server application. We have used Axis2 1.6.1 and the server-side application is deployed as a Web application ARchive (*.war* file) to a Glassfish application server v3.1.2. For serialization of the payload no external library was used as Axis2 itself generates a stub for the POJO that was used for defining the message. The client-side application is running as standard Java standalone application. Upon deployment the server generates all networking code plus a WSDL file; the written POJO implementation of the service is used for the core service logic. Given that WSDL file, the client programmer generates a stub that is used by the client application and hides all networking code and internals dealing with WS. These stub class(es) have been modified in order to record the necessary measurements; no functionality of the Axis2 stack has been changed.

As we can see in Figure 4, there is noticeable drastic difference between the serialization and deserialization speed when using Axis2. This asymmetry is attributed to Axis' usage of Plain Old Java Objects (POJOs) to define the messages – using Java Reflection to read an uncast object is far more costly than creating it in the first place. The serialization time itself is acceptable in comparison with the other frameworks. The low cost for creating a message combined with (or rather balanced by) the higher cost of decoding the message by the receiver, makes Axis2 a good candidate for networks where a resource-constrained device is the sender, while the receiver is a server with plenty resources (e.g. a cloud-based service). The transmission time however, is quite high (seen in Table I), meaning that the sender has to wait a significant amount of time before receiving a response. Additionally, the usage of XML/SOAP imposes a quite noticeable overhead on the data, leading to a high message size (which also impacts transmission).

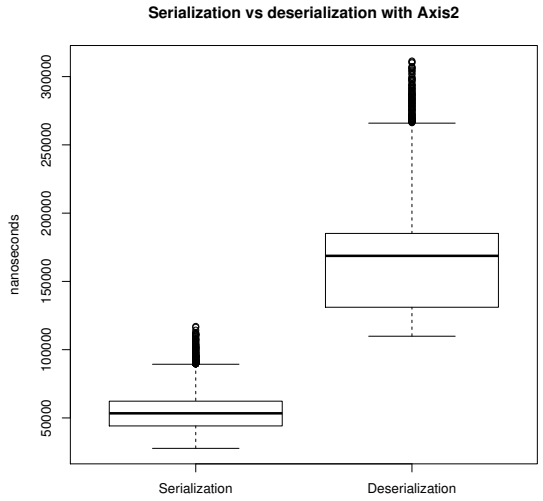

Figure 4. Comparison of Axis2 serialization and deserialization time

C. Devices Profile for Web Services (DPWS)

DPWS defines a minimal set of implementation constraints to enable web service messaging, discovery, description, and eventing on resource-constrained devices. There are several open source implementations, and we have selected to use the WS4D (ws4d.org) and more specifically the Java based stack WS4D-JMEDS v2.0-beta5. The "server"-side is a virtual DWPS-enabled device (emulated) with a web service interface. This virtual device is discoverable on the local network (due to the support of WS-Eventing from the DPWS stack). The client-side program runs WS discovery by broadcasting *Probe* messages on the network. Upon finding a web service matching the search parameters (*ProbeMatch returned*), a connection is established and the test is initiated. Both applications (client and server) run as stand-alone Java programs. No modification of the WS4D library was necessary for the realization of the test.

In this experiment, XML is used to define the message format and SOAP is used to encode the data according to this format. Serialization and deserialization times are comparable. The XML parsing and the string-based nature of data is probably the bottleneck, as compared to potentially more efficient message presentation and encoding methods. The verbosity of XML/SOAP also again impacts the message size and transmission time. Emerging efforts for compressing XML [4] may minimize the impact. Another factor that potentially limits the speed of DPWS is the usage of dynamic discovery

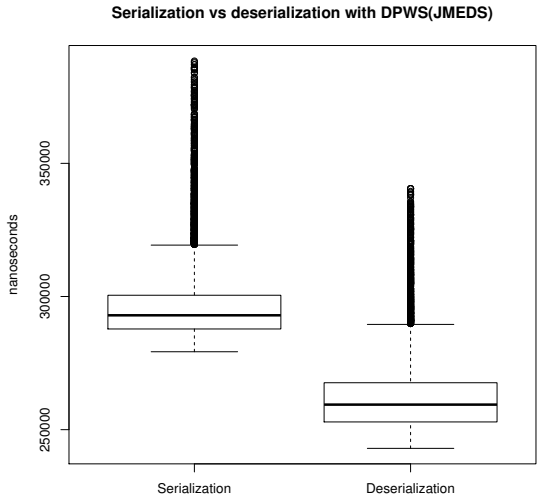

Figure 5. Comparison of DPWS serialization and deserialization time

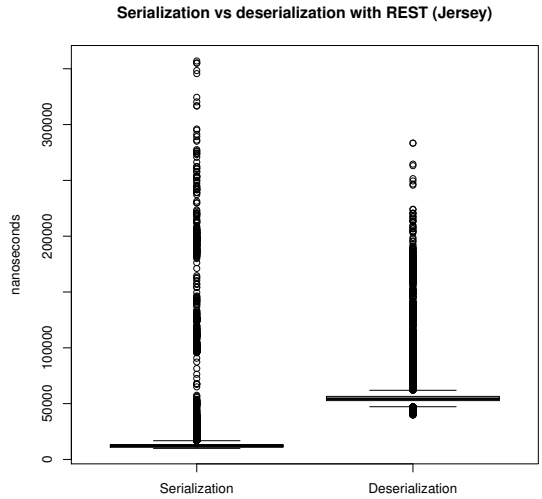

Figure 6. Comparison of REST serialization and deserialization time

(included in measurements). While the other methods tested here use a preset endpoint as message recipient, DPWS has to discover a server that provides the appropriate service. Network topology severely limits device and service discovery (especially if not set-up correctly). There are concepts of cross-network discovery available, however these were considered as outside of the current scope. Generally the performance of DPWS including the dynamic discovery is a good match for devices that want to use its advanced features such as discovery; however there is an impact on the overall performance (as shown in Figure 5). Again the decision for its usage will depend on the concrete scenario and its requirements.

D. REpresentational State Transfer (REST)

REST is an alternative to the traditional web services by offering a stateless client-server architecture where resources are mapped to URLs and are modified using the standard HTTP actions such as *GET*, *POST*, *PUT*, and *DELETE*. In this experiment we have used the REST server and client implementation Jersey v1.10 from Sun Microsystems. An alternative was the implementation from restlet.org, but Jersey was chosen for being the reference implementation, having a lot of community support, and being able to run both as a standalone application and as a web application. The server-side application is deployed as a Web application ARchive (*.war* file) to a Glassfish application server v3.1.2. The client-side application is running as standard Java standalone application.

No modification on the Jersey library was considered necessary for the experiment. The library itself (or any REST implementation in general) does not serialize the message, meaning that the payload is transmitted in the form it was passed over to the framework. In order to achieve high performance from the already lightweight REST approach, we have additionally used Google Protocol Buffers (http://code.google.com/p/protobuf/), which is a way of encoding structured data in an efficient yet extensible format. In this case the payload is a protobuf message encoded byte array, with MIME type the non-standard *"application/x-protobuf"*.

A RESTful architecture is concerned only with the part of the messaging process that is addressing and transmission. How the message is encoded is up to the programmer – any type of data can be used as the payload of HTTP – the transmission protocol of choice for RESTful applications. This freedom allows a serialization method to be chosen that suits the available device and network resources. The Google Protocol Buffers (GPB) was chosen due to their high efficiency, as well as excellent cross-language compatibility. For operation with messages (de-/serialization) GPB provides very high performance Since only simple HTTP is used for payload transmission, the implementation of a client and server is fairly easy and has no special requirements. As it can be seen in Figure 6 and Table I, high performance was achieved while the transmission time was very low in comparison.

E. Constrained Application Protocol (CoAP)

CoAP provides a method/response interaction model between application end-points, supports built-in resource discovery, and includes key web concepts such as URIs and content-types. CoAP easily translates to HTTP for integration with the web, while meeting specialized requirements such as multicast support, very low overhead and simplicity for constrained environments. Being a relative new protocol, not many implementations are available (at least in Java). The WS4D-JCoAP implementation ws4d.org was chosen for its completeness, simplicity and available documentation. This open-source implementation had no official release at the time we conducted the experiments, hence we have used the latest stable code from the repository (http://code.google.com/p/jcoap/).

The server-side application is deployed as a Web application ARchive (*.war* file) to a Glassfish application server v3.1.2. The client-side application is running as standard Java standalone application. Due to the non-blocking (asynchronous)

architecture of the library a slightly different approach has been taken when making measurements. This resulted to some code changes to the core classes in order to be able to track the messages during their round-trip and record the correct timestamps. These lightweight modifications had no impact on the functionality and performance of the jCoAP stack.

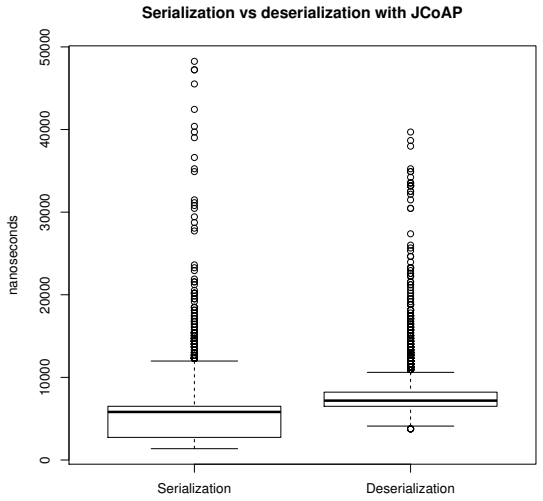

Figure 7. Comparison of jCoAP serialization and deserialization time

CoAP is a protocol inspired by the RESTful architecture, but with focus on resource-constrained devices; hence it defines its own protocol header, designed for efficiency and small size. The architectural similarity to REST also leads to very good (one-way) transmission times, although CoAP (or at least the used implementation) is asynchronous, which means that in some cases it can lead to poorer response time from the point of view of the client. Thus in our results the asynchronous CoAP implementation was slower to receive an answer back (indicated by the transmission time in Table I), in comparison to the synchronous REST implementation experiments already mentioned (we measure the round trip time). Since CoAP can transport any binary data in its payload, we chose to use Google Protocol Buffers for encoding the payload again (as in the REST experiment), the results of which are depicted in Figure 7. The design-goals of CoAP and performance results of the tested jCoAP implementation point out the high suitability especially for resource-constrained devices as expected [5].

F. Discussion and Future Work

Different combinations of operating systems i.e. Windows 7 (client) and Linux (server) were used for running the tests including repetition over IPv4 and IPv6 (with no worth-mentioning deviations). Figure 8 provides an overview on the serialization time on the four stacks we experimented with. DPWS seems to have a higher average time than the others, which may be due to the specific stack implementation. On the deserialization, the two XML/SOAP based stacks (Axis2 and DPWS) have significantly higher times than REST and jCoAP that may be due to message size and stack's implementation.

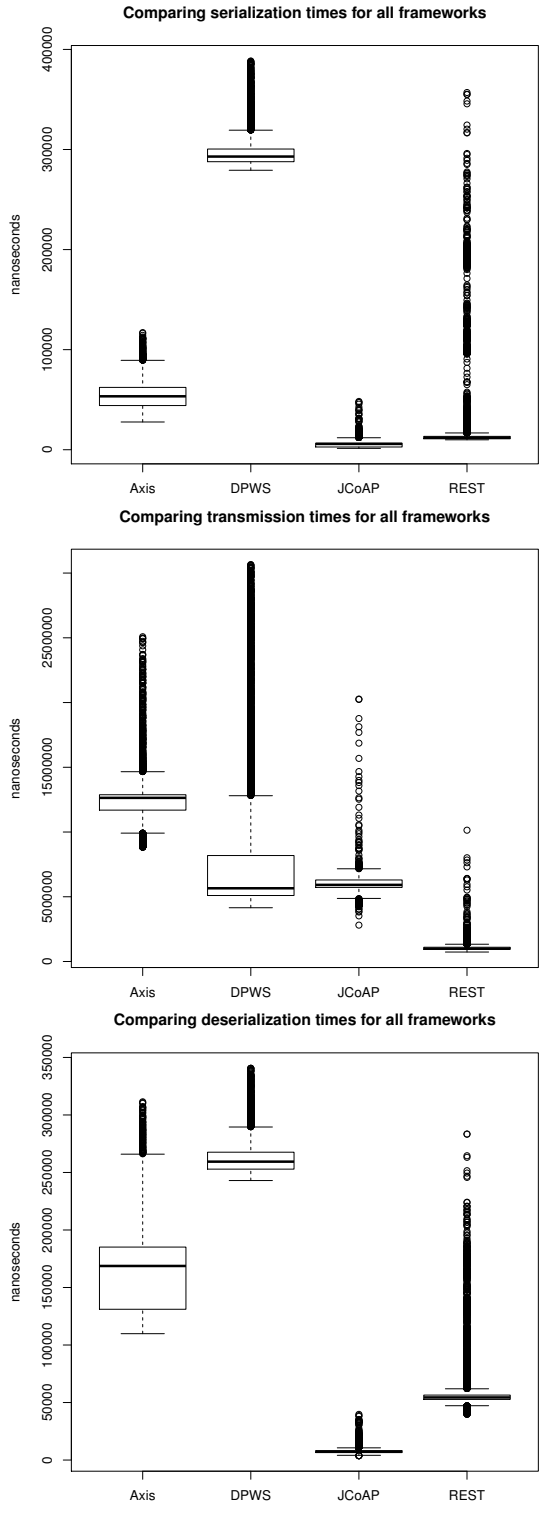

Figure 8. Comparative overview of all experiments

The message size impacts also the transmission time (as depicted in Figure 8), however we see that the device-designed stacks already perform much better in the communication part than the standard web services.

Although DPWS is already available and supported by sev-

eral devices in multiple domains, we can clearly see that in its standard form it has a significant impact on computational and communication resources. Hence devices that may consider this stack, should be usually devices that are on the upper scale with respect to their resource availability. REST and CoAP are designed for much more lean environments and as we see these are a much better fit for resource constrained devices e.g. in comparison to DPWS [6]. Additional combinations of DPWS with compression techniques however could remove this barrier [7]. In addition the REST and CoAP approaches are more lightweight (from CPU and memory utilization) and more user-friendly implementation-wise, and therefore could empower even simple sensors to take part in the Cloud of Things. On the cloud side, since we do not have significant resource problems, any of the stacks can be used, but maybe for scalability reasons the lightweight REST might also be preferred, unless some specific functionality is needed e.g. WS-Discovery from the DPWS in order to dynamically discover embedded devices and their services. Further customizations may enable hybrid approaches such as SOAP over CoAP [8]. Additionally ongoing work e.g. in EXI [9] may also enable better performance when combined with the XML based approaches.

Although the depicted tests are not conclusive and offer only a notion of performance, there are several other issues that need to be investigated and which may be of critical importance, depending on the application domain targeted. Security is an issue, and the impact has not been investigated here as we considered only *HTTP* calls. The impact also of *HTTP pipelining* as well as new future internet *HTTP*-modified networking protocols like *SPDY* [10] and *HTTP Speed+Mobility* [11] that offer reduced latency through compression, multiplexing, and prioritization need to be assessed. Additionally, other issues such as excess buffering of packets may cause high latency and jitter [12], and this may have significant impact on network performance, which might be a show-stopper e.g. for time-critical applications.

III. Conclusion

We have presented an initial assessment for major technologies that can be used to realise the "Cloud of Things". More specifically we have used available open source stack implementations to get a feeling on the performance of traditional web services (based on XML/SOAP), the ones customized for devices i.e. DPWS as well as REST and CoAP, all of which can nowadays be used to make the networked embedded devices SOA-ready i.e. enable them to offer their functionality as a service and be part of a service-based ecosystem. The initial results have shown that although all approaches will enable the embedded devices to take part in the "Cloud of Things", the REST and CoAP have much better performance due to their lightweight concepts and implementations. Hence, depending on the available resources on the devices, as well as the requirements on the specific application domain, a concious choice has to be made. The impact also of future internet *HTTP*-modified networking protocols like *SPDY* and *HTTP Speed+Mobility* as well as other issues such as latency [12] will need to be assessed when used in conjunction with some of the experiments demonstrated to see what the potential benefit might be. We have hardly scratched the surface, as more real-world tests need to be made with real-world devices that host these stacks and interact via the investigated protocols, while several other issues e.g. scalability, security, on-device resource utilization etc. will need to be taken into consideration in under real conditions.

Acknowledgment

The authors would like to thank the partners of European Commission co-funded projects IMC-AESOP (www.imc-aesop.eu) and CONET (www.cooperating-objects.eu) for the fruitful discussions.

References

[1] J.-P. Vasseur and A. Dunkels, *Interconnecting Smart Objects with IP: The Next Internet*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 15 Jun. 2010.

[2] S. Karnouskos, V. Vilaseñor, M. Handte, and P. J. Marrón, "Ubiquitous Integration of Cooperating Objects," *International Journal of Next-Generation Computing*, vol. 2, no. 3, 2011.

[3] S. Karnouskos and A. W. Colombo, "Architecting the next generation of service-based SCADA/DCS system of systems," in *37th Annual Conference of the IEEE Industrial Electronics Society (IECON 2011), Melbourne, Australia*, 7–10 Nov 2011.

[4] S. Sakr, "XML compression techniques: A survey and comparison," *Journal of Computer and System Sciences*, vol. 75, no. 5, pp. 303–322, Aug. 2009.

[5] C. Lerche, K. Hartke, and M. Kovatsch, "Industry adoption of the internet of things: A constrained application protocol survey," in *Proceedings of the 7th International Workshop on Service Oriented Architectures in Converging Networked Environments (SOCNE 2012)*, Kraków, Poland, Sep. 2012.

[6] G. Moritz, E. Zeeb, S. Prter, F. Golatowski, D. Timmermann, and R. Stoll, "Devices Profile for Web Services and the REST," in *8th International Conference on Industrial Informatics (INDIN), Osaka, Japan.*, Jul. 2010.

[7] G. Moritz, D. Timmermann, R. Stoll, and F. Golatowski, "Encoding and compression for the devices profile for web services," in *24th IEEE International Conference on Advanced Information Networking and Applications Workshops, WAINA 2010, Perth, Australia*, Apr. 2010.

[8] G. Moritz, F. Golatowski, and D. Timmermann, "A lightweight SOAP over CoAP transport binding for resource constraint networks," in *Mobile Adhoc and Sensor Systems (MASS), 2011 IEEE 8th International Conference on*, oct. 2011, pp. 861 –866.

[9] A. Castellani, M. Gheda, N. Bui, M. Rossi, and M. Zorzi, "Web Services for the Internet of Things through CoAP and EXI," in *Communications Workshops (ICC), 2011 IEEE International Conference on*, june 2011.

[10] M. Belshe and R. Peon, "SPDY Protocol," IETF Internet-Draft, Internet Engineering Task Force, Feb. 2012. [Online]. Available: http://tools.ietf.org/html/draft-mbelshe-httpbis-spdy-00

[11] R. Trace, A. Foresti, S. Singhal, O. Mazahir, H. F. Nielsen, B. Raymor, R. Rao, and G. Montenegro, "HTTP Speed+Mobility," IETF Internet-Draft, Internet Engineering Task Force, Jun. 2012. [Online]. Available: http://tools.ietf.org/html/draft-montenegro-httpbis-speed-mobility-02

[12] J. Gettys and K. Nichols, "Bufferbloat: dark buffers in the internet," *Commun. ACM*, vol. 55, no. 1, pp. 57–65, Jan. 2012. [Online]. Available: http://doi.acm.org/10.1145/2063176.2063196