Multi-Protocol Transport Layer QoS: An Emulation Based Performance Analysis for the Internet of Things

James Wilcox
Center for Communications Research
University of Bristol, UK
james.wilcox@bristol.ac.uk

Dritan Kaleshi
Center for Communications Research
University of Bristol, UK
dritan.kaleshi@bristol.ac.uk

Mahesh Sooriyabandara
Telecommunication Research Laboratory
Toshiba Research Europe Limited, UK
Mahesh@toshiba-trel.com

***Abstract*—Application specific interaction models will be required to support efficient communications between distributed applications with disparate network requirements within the Internet of Things. This paper shows that intelligently selected transport protocols are able to provide increased efficiency of network resource usage under specific network conditions. Real-time adaptive selection of transport protocols makes it possible to achieve a distributed embedded system with heterogeneous actors that can react to both application-specified Quality of Service (QoS) requirements and varying network conditions. vNET, a custom, virtualisation based, distributed network emulation test bed will be presented and validated using an MQTT performance analysis before using it to validate the premise of multi-protocol transport layer QoS.**

***Keywords*–Quality of Service, Internet of Things, MQTT, Adaptive Transport Layer, Network Emulation, Middleware, Smart Grid**

I. Introduction

This paper is an extended piece based on [1]. It provides additional content on the virtualisation based network emulation test bed developed for this work.

Traditionally networked applications are designed with a pre-selected transport layer protocol. Optimisations for a specific application are done at the application layer and all messages are transported using the same protocol, either TCP (Transmission Control Protocol) [2], UDP (User Datagram Protocol), or with an overlay transport protocol such as RTP (Real Time Protocol) [3]; Figure 1 represents this paradigm. This is typically fixed at application development; however, there is no fundamental requirement for this to be the general rule. Whilst networked applications need to exchange information, there is no reason why application layer code should be concerned with how that information is transported. There are a multitude of existing, mature transport layer protocols available each designed to tackle specific network problems [4]. Utilising these many protocols, a single application could leverage the advantages of each protocol individually at the appropriate time given an environment with dynamic network conditions and application requirements. Acknowledging these points raises the challenge of defining a generic framework that allows for run-time selection of transport protocols to dynamically match specific application requirements and, specifically, message patterns used by the application. If the low level network interactions enforced by a specific transport protocol and higher level architectural messaging pattern are completely decoupled from the application then dynamically modifying the combination can be used as standard. If certain transport protocols and messaging pattern combinations are able to provide higher performance in terms of bandwidth, latency and reliability in certain network environments than others can do, then by supporting adaptive selection of these combinations it becomes possible to have a distributed real-time embedded (DRE) system with heterogeneous actors that can react to both dynamic application QoS requirements and network conditions. This model is shown in Figure 2. The middleware system required for managing the selection of the large numbers of transport protocols referenced in Figure 2 has been developed and presented in DIRECTOR: A Distributed Communication Transport Manager for the Smart Grid [5].

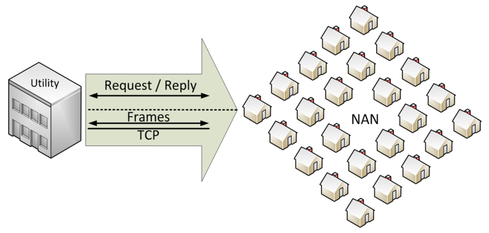


Figure 1. Traditional: Applications are supported by a single interaction model (in this case, TCP Request / Reply). The utility represents systems providing the back end infrastructure.

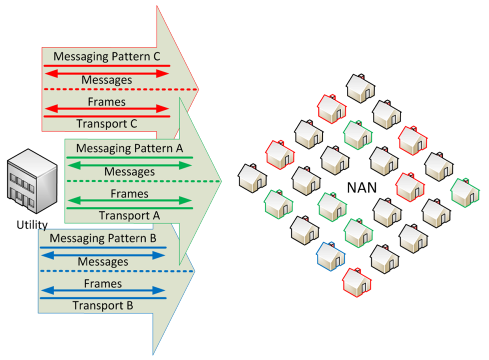


Figure 2. Proposed: Applications using multiple run time optimised interaction models (managed using middleware) instead of only TCP Request / Reply.

An "interaction model" as used throughout this work describes the virtual patterns of packets, datagrams or frames (transport protocol and messaging pattern combination) that constitute a data communication session between at least two networked devices. A specific interaction model will communicate over compatible interfaces. For example there is a clear difference in interaction model between communicating with unicast and communicating with multicast transports. Figure 3 further demonstrates how multi-protocol QoS could be used to exploit the under utilised optimisation opportunities using a more specific example. Both approaches shown achieve the same overall goal - the three receiving nodes obtain the sending nodes data. However, in this scenario, multicast offers clear bandwidth efficiency increases compared to unicast. Therefore, an intelligent system should be able to automatically make this determination to improve performance in a selected metric. This premise extends much further than unicast, multicast and bandwidth efficiency however. There are numerous approaches to getting packet X to destination(s) Y. Each interaction model has its own set of attributes which affect the performance of both the application and the underlying network in different ways. The attributes can be classed and characterised to provide groups of interaction models that provide specific advantages in certain communication scenarios. So while it is possible to support many disparate IoT applications with a fixed interaction model, typically tied in with a single transport protocol, it would be more efficient in terms of network resources to be able to provide tailored interaction models on a per-application basis. These tailored interaction models would be designed to meet specific Quality of Service (QoS) levels utilising network resources more efficiently than the case when only a fixed interaction model approach, often set at system design stage.

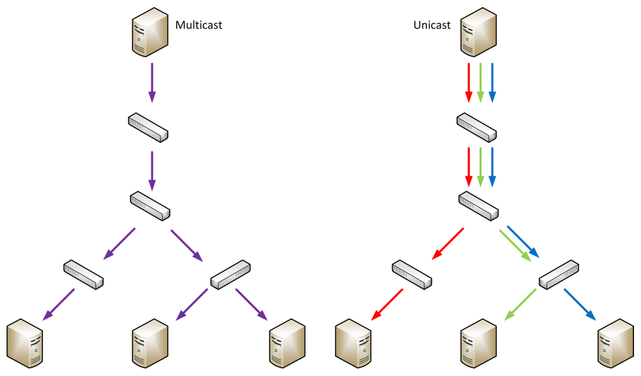


Figure 3. Interaction model differences for disseminating the same data using Multicast vs. Unicast

This paper shows experimental results which demonstrate that specific interaction models will provide performance gains over a pre-defined communication transport architecture and that certain combinations provide useful gains over other potentially viable options. The potential of generating a large scale mapping of transport combinations and application requirements will be explored. The main contributions of this paper are:

1) An analysis of performance of a distributed system for different interaction models by running application scenarios using a fixed, realistic, network topology.
2) Results that show that the performance of the Distributed Real-time Embedded (DRE) applications varies significantly for different interaction models, suggesting that this then can be used to optimise the performance of the individual transactions that make up the application network traffic.
3) Presentation of vNET, a virtualisation based, distributed network emulator.

The paper is organized into the following sections: section II provides the related material and further motivation for this work. Section III presents the experimental emulation test bed setup and the viable communication interaction models. Section IV presents the experimental parameters and the results. Section V presents a potential middleware solution that can capitalise on these findings. Section VI presents the conclusions from the experiments and the direction of future work.

A. *Related Work and Motivation*

Environments targeted by this work have the follow characteristics:

1) They are distributed and built from a large number of heterogeneous embedded devices, running a number of different applications.
2) They are typically loosely-coupled.
3) The majority of the actors are communication network-constrained rather than resource-constrained.
4) Each device is expected to run many different applications with varying network requirements.

One example of such a system is the SG, and in particular, the subset of applications that intend to use consumer / demand side equipment and systems to achieve grid specific goals such as load shedding or load shifting or in more general terms, Demand Side Management (DSM). Section A introduces the SG and its edge applications. Several related works exist which support with the premise of providing DRE software applications, such as those that will operate in a SG, with flexible communication choices in order to either achieve better network resource allocations or meet specific communication requirements. These overviews are presented in sections B and C.

1) The Smart Grid and Demand Side Management (DSM): The SG can be seen as a large scale distributed IoT system, with a large component being embedded sensor-actuator networks to support distribution power network monitoring and control and DSM interactions. DSM focuses around the control of demand side loads in the electricity distribution network in order to manipulate network conditions [6]. DSM breaks down into a number of related but still significantly different enough sub-applications to warrant different communications approaches. DSM can be broken down into two major sub categories, Demand Control (DC) [7] and Demand Response (DR) [8]. DC is defined as DSM programs that have centralized direct control over consumer loads [7]; DR is defined as DSM programs that use indirect methods (typically pricing) to affect changes [9]. Each approach requires a different communications paradigm in order to utilise network resources efficiently and operate optimally. The above presents an ideal system for this work. It presents the rare opportunity to take a completely different approach to facilitating machine-to-machine communications in a DRE environment. The SG will eventually call for millions of networked geographically-distributed embedded devices to be deployed into the demand side of the power distribution grid. These devices are either designed to utilise existing networks such as domestic broadband or cellular networks and coexist with the existing traffic, or to utilise purpose built resource constrained networks such as various forms of wireless mesh or power line communications [10]. Both approaches result in strict network resource constraints for the applications. These constraints further increase the impact run-time transport level adaptive QoS will have in such environments. The main argument against dynamically matching interaction models to application requirements and real time network conditions has been one of complexity. With sensor networks, the SG and the Internet of Things (IoT) in general becoming more prevalent, the environment is changing and these arguments are no longer valid. Our proposition is that the performance gain introduced by dynamically matching interaction models outweighs the required increase in complexity of the architecture and embedded hardware. DSM can be shown to be a good example of how tailored communication paradigms could be beneficial in the SG and similar environments.

2) Transport Mechanisms : The concept of adaptive transport layer services for resource-constrained environments is well explored; however, the approach taken usually considers the transport protocol to be used already pre-selected at development stage, and only considers adaptations above the transport layer. They do not propose to provide application optimisations at the lower transport level. However, applications, regardless of the type of resource-constrained environment, can benefit from tailored communication service at the transport layer. Mutlu et al. [11] presents a middleware solution for performing transport level QoS focused on Bluetooth application profiles and uses CORBA (Common Object Request Broker Architecture) [12] to facilitate the middleware. While the scope is clearly limited, and transport protocol choices are not part of the QoS mechanism, the motivation is similar. Furthermore, it can be shown that different communication protocols have inherently different QoS characteristics and that using targeted protocols with specific applications can improve performance with a number of chosen metrics. Weishan et al. [13] recognise this and provide experimental results related to protocol switching overhead and also implement the system using a middleware solution. They conclude that protocol switching overhead is minimal with their chosen transport protocols and that protocol switching is beneficial to DRE environments.

3) QoS Architectures for DRE systems: The works highlighted here are attempting to improve or maintain DRE application performance in sub-optimal or resource-constrained networks by utilising real-time adaptive QoS management mechanisms. [14], [15], [16] focus on a single interaction model and attempt to provide adaptive QoS within these confines. It demonstrates that additional QoS optimisation opportunities are available if the scope of the system includes controlling lower level attributes such as interaction models in conjunction with the adaptive QoS mechanisms. For example Wenjie et al. [15] propose a QoS adaptive framework for Publish-Subscribe Service called QoS Adaptive Publish-Subscribe (QAPS). They define several QoS policies and focus on fault tolerance and dependability of services. Schantz et al. [17] present a distributed, real-time embedded system capable of adaptive QoS. They describe in detail several methods of implementing end-to-end adaptive QoS mechanisms and explain how the work gives DRE applications more precise control over how their end-to-end resource allocations are managed. These proposed adaptive QoS mechanisms all address the same problem as this paper, but these implementations are limited to the application layer instead of considering a multi-protocol transport layer to access additional optimisation opportunities. Zieba et al. [16] develop the concept of quality-constrained routing in publish / subscribe messaging architectures. They develop a system which integrates application quality requirements into the message routing architecture in order to better support dealing with varying network conditions such as dynamic network topologies and link characteristics. The idea of integrating the dynamic application requirements into the communication paradigm provides a critical distinction from the others and further reinforces the need for verified optimised communication paradigms in order to meet these dynamic requirements.

II. vNET: A Virtualised Network Emulation Test-bed for the Embedded Internet of Things

Developing and evaluating network solutions for IoT environments, especially those near deployment, requires a method of allowing the participating entities and critical hardware components to interact in a controllable, scaled way. A real world trial would be ideal as it would produce highly detailed and accurate results but the cost of this approach is often prohibitive. Another solution is to run simulations which are cheap, especially if there is no real-time requirement, but only provide results as accurate as the models used and the assumptions made. There are many network simulation tools available such as Qualnet and NS2/3 [18], [19] that already have tried and tested networking models for a variety of scenarios. However, for systems that can be described as *complex networks of discrete components* like multi-application IoT environments, simulation does not naturally lend itself. Providing models for each disparate networked entity is time consuming and furthermore if the goal is to evaluate how software or devices that utilise custom interaction models perform in a distributed network, simulating these entities would often require re-implementing the network code so that it is compatible with the simulator. This is inefficient and difficult especially if the system is required to react to unscripted network events.

Network *emulation* provides a viable and valuable alternative in these cases. With emulation, the network and the devices that a system consists of are completely represented and inherently provide the same interfaces and functionality the real world system would. This allows real development code to be directly evaluated in an easily controllable and scalable manner without resorting to a physical deployment. The emulation test-bed can be seen as a condensed version of a real life system with all the varying levels of complexity a real world system would have, from the standard open source software running on each node down to the physical layer of the network. In addition to this, the ability to pass real hardware to specific nodes within the network allows OS driver interactions and hardware choices to be evaluated potentially revealing incompatibilities and areas for optimisation before moving onto scaled real-world trials. Until relatively recently, full network emulation on a useful scale for evaluating DRE environments has been difficult to achieve, mainly due to computing resource constraints. Recent advances in virtualisation technology can be used to address this problem. Furthermore, as IoT devices are typically resource constrained and relatively low performance, large numbers can be fully emulated with modest, and therefore inexpensive, host hardware. This makes an emulation test-bed a valuable and more suited tool in evaluating embedded distributed networks compared to simulation. Furthermore, real network hardware is incorporated into the test-bed as a proof-of-concept for hardware-in-the-loop (HIL) emulation based testing which provides the ability to evaluate real hardware and driver choices for possible incompatibilities.

This section presents:

1) A virtualisation based network emulation test bed that allows unmodified code and real hardware / driver interactions to be evaluated in an extensible, fully controllable distributed, software defined, networked environment.
2) A series of experimental scenarios based on the IBM MQTT performance analysis [1]. These experiments will focus on analysing CPU and bandwidth consumption of the various QoS levels MQTT provides and will demonstrate the functionality and capabilities of the test-bed.

A. Network Emulation Related Work

Basic requirement set for a viable distributed network, embedded device test-bed:

- Complete flexibility in terms of evaluating software in both user and kernel space.
- Full integration with physical networks.
- Ability to evaluate real hardware and driver choices.
- Scalable in terms of node numbers.
- Low emulation overhead.

Several candidates were identified which met some of these requirements. The rest of this sections evaluates these potential choices.

Network analysis through the use of emulation and virtualisation is not a new area [20], [21], [22], [23]. There is a large amount of work that uses network emulation due to the benefits it provides over purely simulation based analysis.

The Common Open Research Emulator (CORE) used by [23] provides an ideal example of the capabilities of existing emulation test-beds. CORE is the closest comparable technology to the emulation test bed described in this work. However, there are 2 areas where CORE lacks features that allow access to previously unexplored analysis opportunities:

1) CORE does not support passing through physical hardware such as NICs, storage controllers and other application specific hardware.
2) The LXC (Linux Containers) used by CORE *must* use the same host system kernel and so each emulated node cannot use a custom kernel or kernel level modifications and this limits flexibility.

Mininet [24] is another network emulator that uses LXC. It is provides the ability to evaluate large networks using consumer grade PC hardware. Mininet explicitly targets Software Defined Network (SDN) environments and allows vendor independent OpenFlow interface compatible controllers to be experimented with. While Mininet, like CORE, are extremely useful network emulation tools, they are still limited by LXC. While LXC allows greater numbers of nodes to be emulated, the lack of complete isolation and ability to implemented kernel level or individual network stack modifications is clearly limiting.

Cooja [25] is a Java based contiki [26] mote simulator. Contiki is an OS for small embedded sensor platforms and Cooja was developed in order to evaluate interactions between small numbers of them without resorting to physical trials. Cooja is able to simulate at multiple layers including, the network layer, the OS layer and the machine instruction set layer if required. While Cooja is ideally suited for testing small (<100) node mote networks, its usefulness outside such network environments is limited.

TABLE I. Related Network Evaluation Tools [27] [28]

	Key Tech	**Node Limits**	**Network Limits**	**Target Scens**	**Comments / Use cases**
CORE	LXC, NS	No hard limit	Not Real Time	Non specific	Limited interactions with physical networks
Cooja	Java, NS	<100 nodes	Mote Interfaces	Contiki wireless mesh	Limited to evaluating motes.
Mininet	LXC	4096 per host	4Gbit/s with 4GHz CPU core	SDN OpenFlow	Large scale topology evaluation, SDN algorithms, Limited interactions with physical networks
vNET	ESXi, VMs	512 per host	9.5Gbit/s vmxnet3	Near deployment testing	Embedded Hardware Evaluation. Efficient interactions with physical networks. Software flexibility.

Expanding on these points:

Using a virtualisation technology that can take advantage of AMD-V's IOMMU or Intel's VT-d virtualisation technologies would allow real hardware devices to be passed to a virtual machine. The virtual machine kernel recognises the hardware as it would in a real system and loads the standard drivers for its operation. This allows the intricacies of real driver / hardware interactions to be evaluated. For example, a previously emulated key backbone connection could be seamlessly brought out into the real world using physical connections and network interfaces so that hardware chip sets and driver optimisations could be tested under varying load levels before resorting to comparatively expensive and time consuming real world scaled trials. This physical hardware can quickly be assigned to any hosted virtual machine and used natively. This is especially useful for systems that are near deployment.

LXC does not provide *virtual machines*, it provides a *virtual environment*. Therefore, the major problem with using LXC and similar container based virtualisation technologies, is the lack of complete isolation between virtual machines. Each LXC container shares the same kernel with one another and also the host itself. This means all software must be compiled for the same CPU architecture and kernel modifications between nodes are not possible. Emulating a network of disparate devices with this technology is therefore more difficult and restrictive.

For these reasons a new and custom fully virtualised network emulator was required.

B. Test-bed Components

The following section describes the various software tools and components that make up the virtualised emulation test bed. The following components are discussed:

1) *Virtualisation hypervisor*. Provides the virtual distributed environment and resource usage monitoring.
2) *WANem*. Provides control over the behaviour of specific individual communication links and network segments
3) *TCPDump and WireShark*. Provides data capture and offline analysis.
4) A set of custom Windows Power Shell VMware vSphere CLI scripts for deploying and manipulating custom virtual machines and their allocated resources quickly and efficiently.

C. Hypervisor Choice

There are a number of commercial and open source virtualisation technologies available. Specifically a native or 'bare metal' hypervisor was desired over a 'hosted' hypervisor in order for the computational virtualisation overheads to be as small as possible.

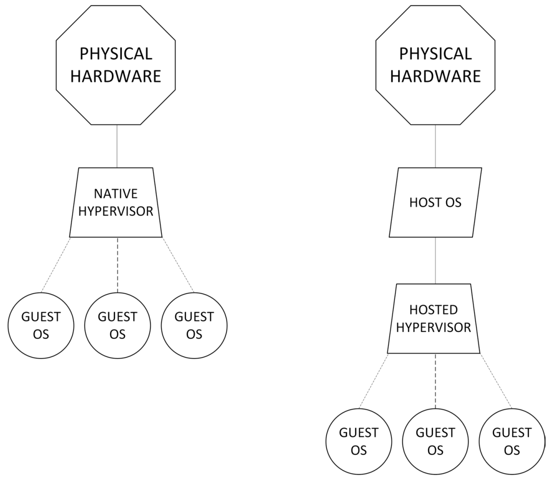


Figure 4. Native (bare metal) vs. hosted hypervisors.

Figure 4 shows the key difference between the choice of native and hosted hypervisors. A hosted hypervisor requires the support of an underlying general purpose operating system (OS), or *host OS*. This host OS requires computational resources such as CPU time, storage I/O and Memory I/O in order to operate and this means these resources are not available to the hosted hypervisor guest OS. A bare metal or native hypervisor does not require this support and therefore more resources are available to the guest OS. The requirement of a bare metal hypervisor limits the technology choices to:

1) VMWare ESXi
2) Xen Hypervisor
3) Microsoft Hyper-V

Each of these technologies has the same basic functionality and could have been used to implement the emulation test-bed. ESXi is largely considered the most mature product within enterprise and provides a robust VM management infrastructure that would make implementing and operating the test-bed easier. For these reasons ESXi was chosen as the emulation test beds native hypervisor.

D. Virtualised Distributed Environment - ESXi

Enterprise-class virtualisation provides a convenient approach to large scale cheap and efficient emulation. The test-bed described in this paper uses the VMware ESXi 5.1 bare metal hypervisor virtualisation technology [29]. ESXi allows the complete virtualisation of x86-64 based computers on a large scale. The proliferation of enterprise class technology such as Intels VT-x and AMDs AMD-V into consumer PC hardware provides dedicated on die hardware for accelerating virtualisation operations. This significantly improves the performance of virtualisation over previous generations of hardware. While typically this virtualisation technology is used to consolidate services onto a single set of server grade hardware it also has the potential to be used in unconventional ways. In this case the technology has been used to host 350 micro guests. Each of these guest nodes can be allocated a fixed amount of computing resources and / or physical hosted hardware in order to emulate a specific real world resource constrained device. Figure 5 shows an overview of the test environment. Each network segment represents an isolated network. Each network segment has an network bridge which connects it to neighbouring segments. Traffic is controlled here to emulate various network types and conditions. The system is managed by an external desktop computer which runs the VMware vSphere client and executes the custom CLI scripts detailed in section II-G. WAN access through Network Address Translation (NAT), and DHCP and DNS services are optionally provided to experimental nodes using a dedicated VM running the open source pfsense [30] 2.1 routing OS. The diagram shown in Figure 5 shows a network topology for emulating a Open Automated Demand Response Real Time Pricing usecase (OpenADR RTP) [31] but the topology is easily reconfigured through the vSphere software.

The physical hardware of the host consists of a 3.12GHz 8 core AMD FX-8120, 32GB DDR3 memory and 960GB of Solid State Drives (SSD). This hardware provides enough computing performance to support 350 fully implemented nodes. This hardware is now several generations old. Commercial offerings from both AMD and Intel have increased both Instructions Per Clock and base clock speeds of their processors. This means that for the same cost more nodes can now be supported.

E. Network Connectivity Emulation WANem

Network connectivity is emulated through the use of virtualised switches, traffic shaping and virtualised software network bridges. Complex network topologies can be emulated by introducing these virtual components at specific points in the virtualised environment. Traffic shaping is provided by WANem [32] a software WAN emulator. It provides the ability to manipulate many common network characteristics including bandwidth limitation, latency, packet loss and random network disconnections. Non ideal networks can be emulated by setting various levels of packet loss and packet corruption and this allows a system's behaviour to be evaluated under sub optimal conditions. Multiple, dedicated TCPDump traffic sinks are

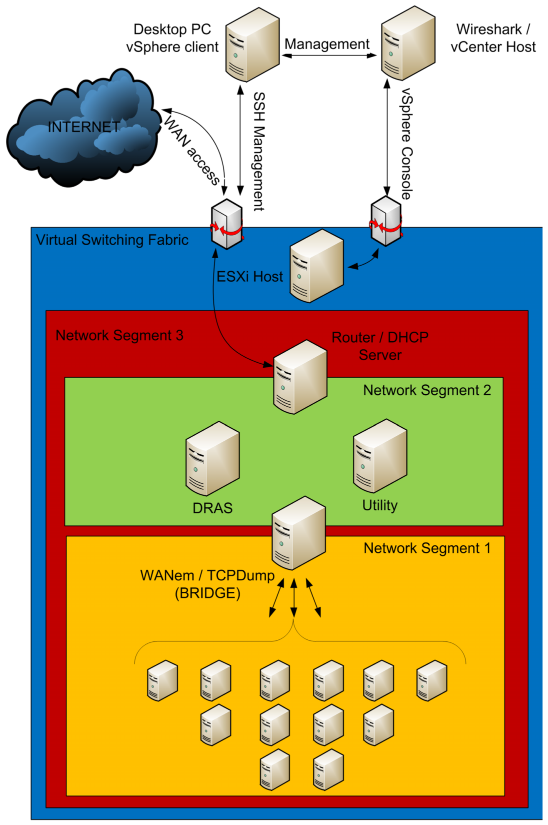


Figure 5. Overview of vNET. S1 represent a NAN. S2 represents a utility's systems. S3 represents the utility's network infrastructure. All segments are software defined and are reconfigurable.

deployed throughout the topology to collect packets at key points.

F. Result capture and analysis TCPDump, Wireshark and ESXi

Utilised computing resources (CPU, Memory, Disk and RAM IO etc.) are recorded using the built in monitoring features of the ESXi host which provides statistics for all virtual machines. This feedback allows development code to be optimised before deployment potentially reducing the end target hardware cost. TCPdump[33] provides a packet level capture of all transmissions and Wireshark [34] provides the facility to analyse this capture in detail for lost and corrupted packets, retransmissions, packet latency and protocol behaviour. This is a key feature for developing distributed application code. It allows code to be certified as functional before deployment.

G. PowerShell scripts

A method of managing this network of virutal machines was needed. This was provided through Windows PowerShell which is a task automation and configuration framework. The VMWare vSphere CLI can be controlled using custom PowerShell. Therefore, to take advantage of the management interface, a number of custom scripts and a CLI was developed to facilitate the following operations:

1) Begin executing the chosen scenario.
2) Set Maximum Transmission Unit (MTU) of specific nodes. This is useful for emulating transactions with increased protocol overhead.
3) Clone VMs from template. Useful for deploying new scenarios rapidly.
4) Destroy VMs.
5) Power up all VMs.
6) Set CPU limit with MHz resolution. Useful for emulating CPU constrained devices.
7) Set CPU reservation. Useful for guaranteeing certain nodes specific processing resources.
8) Power on specific VMs.
9) Restart specific VMs.
10) Change VM network host name.
11) Issue custom command. Useful for providing additional configuration flexibility - takes standard BASH commands and passes them to the chosen nodes.

These scripts allow easy manipulation of the emulation test bed and the virtual machines within it. These files are currently being made publicly available and in the mean time can be requested by email.

III. vNET Validation: Experimental Scenarios

For these experimental scenarios the Message Queue Telemetry Transport (MQTT) [35] protocol was chosen. MQTT is designed to be an efficient, broker-based, publish / subscribe transport protocol. MQTT was chosen to demonstrate vNET as it is well suited to environments with constrained resources, for example where the network is expensive or the embedded devices involved are CPU or RAM constrained. This, coupled with the existing IBM performance [36] analysis will provide a good base for vNET validation.

Few performance analyses have been attempted for MQTT to date [36], [37] and neither has attempted a fully emulated testing approach. Fenton [36] used physical server grade hardware to analyse the performance of the IBM MQTT broker software under load but used artificial traffic generation techniques to emulate incoming traffic which limited the scope of the performance analysis to the broker. Perez took a fully simulated approach using the OMNeT++ [38] network simulator, which restricts the flexibility and limits usefulness of the test-bed. The experiments described here take the fully emulated approach with the addition of real hardware at critical points in the network topology in order to evaluate HIL potential. Unlike the experiments performed by Fenton which focused solely on the broker, the whole network is emulated all the way from broker to clients.

The IBM performance analysis of MQTT [36] specified two scenarios. These have been replicated (but scaled down) in order to show the emulator it can be used to experiment with arbitrary scenarios and network topologies. They are:

1) Multi-publisher, single-subscriber.
2) Multi-publisher, multi-subscriber.

In the first scenario messages are sent to the broker at a rate of 100 per second from randomly selected clients. A single subscriber receives all of the publishes.

In the second scenario each client subscribes to one single topic. Message rates are the same as in scenario 1. It also publishes to a different single topic. Each topic a client subscribes to is published to by only one other client. Therefore,

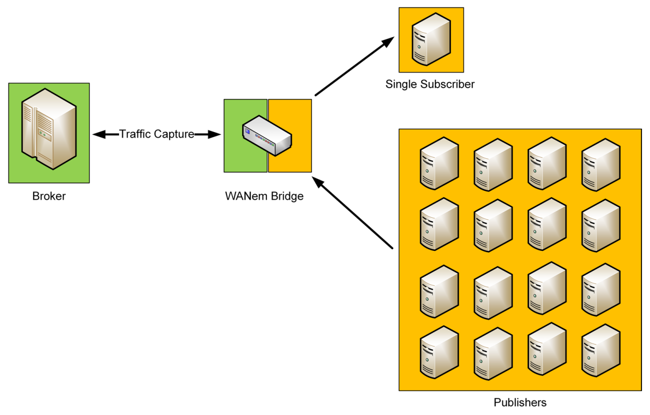


Figure 6. Scenario 1. The broker bridge link uses HIL and brings the traffic into the real world using physical network adapters.

this scenario can be seen as each client having a one to one mapping with another client.

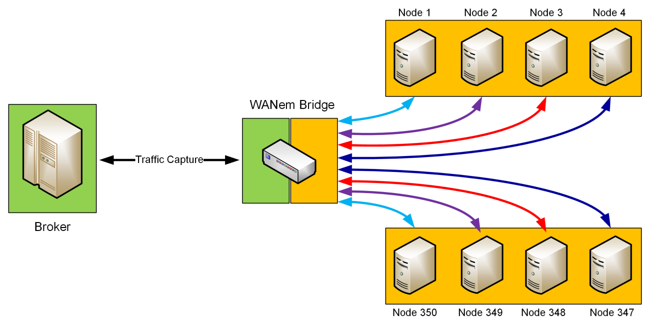


Figure 7. Scenario 2. HIL is used in the same position.

For each scenario all three MQTT QoS levels are experimented with.

1) QoS 0 - The broker/client will deliver the message at most once, with no confirmation.
2) QoS 1 - The broker/client will deliver the message at least once, with confirmation.
3) QoS 2 - The broker/client will deliver the message exactly once, with confirmation.

These six experiments provide results on CPU and network utilisation for both broker and clients.

While the IBM experiments have influenced the scenarios tested in this work, we are are not simply replicating them and there are several key differences. The experiments are scaled to 350 nodes. This is the limit of the hardware being used. Additonal VM hosts would be needed to increase the number. Open source mosquitto [39] is used instead of the IBM Websphere suite. Mosquitto has a smaller resource overhead than the IBM Websphere making it more suited to embedded environments. The clients generating the traffic are individually fully emulated; there is no use of Telemetry Device Daemons to increase the traffic as with the IBM experiments. Since the aim of this experiment is not to simply benchmark the broker, emulating all the clients provides new results which will be used to further analyse MQTT QoS levels and validate the test-bed. The payload for the MQTT messages is a 717 byte XML file that represents a metering update message. The syntax has been borrowed from the Zigbee SEP [40]. Each node client is connected to the broker sequentially. This will allow the CPU time and network bandwidth consumption to be monitored as the number of nodes is increased from 1 to the maximum node number for the experiment. Once the number of nodes reaches this maximum the simulation runs for 15 minutes and automatically shuts down. The network connection between the broker and the software Ethernet bridge utilises a real hardware Intel gigabit ET dual port server adapter looped back on its self. This presents a proof-of-concept for HIL in the test-bed.

Further experimental scenarios based on OpenADR [31] have been performed using the test-bed and the results and analysis of these have been published in *DIRECTOR: A Distributed Communication Transport Manager for the Smart Grid* [5].

A. *vNET Validation: Results*

Figures 8 and 9 show a predictable increase in both CPU usage and bandwidth consumption as the QoS level is increased from 0 to 2.

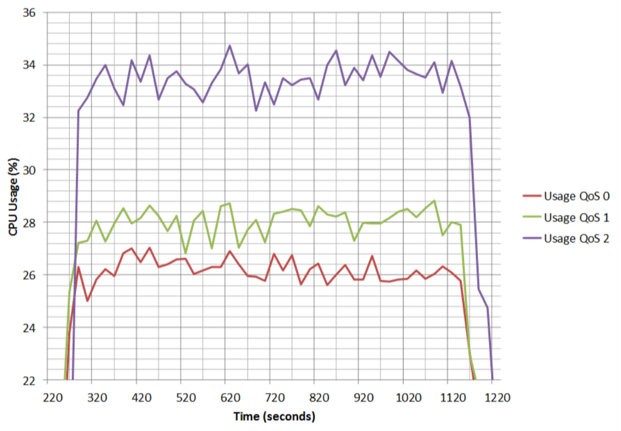


Figure 8. Scenario 1 Broker CPU usage

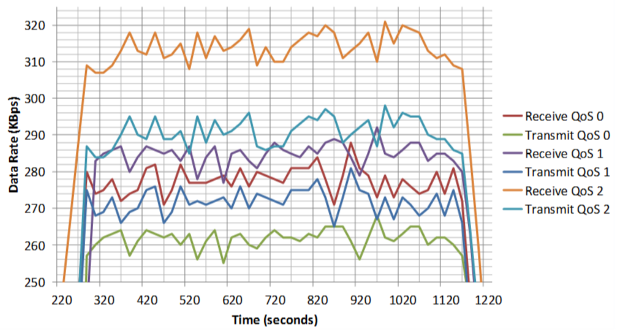


Figure 9. Scenario 1 Broker Network Usage

It can be noted that QoS level 2 is significantly more expensive in terms of processing and networking resources than the previous two levels. This can be seen in Figures 8, 9, 10, and 11. For comparison the average CPU usage for a publishing node in this scenario has been given in Table II.

Figure 10 shows that at most the single subscriber is using approximately 6 times the CPU resources the publishers are

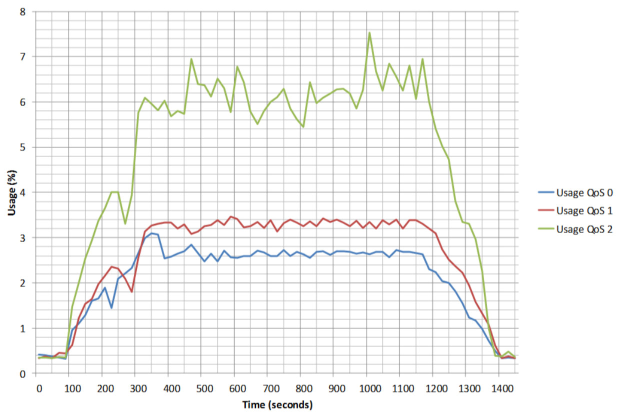


Figure 10. Scenario 1 Single subscriber CPU usage.

even though the network utilization is as much as 360 times as large (The average network utilization for the publishing nodes was between 0.5 and 1 KBps).

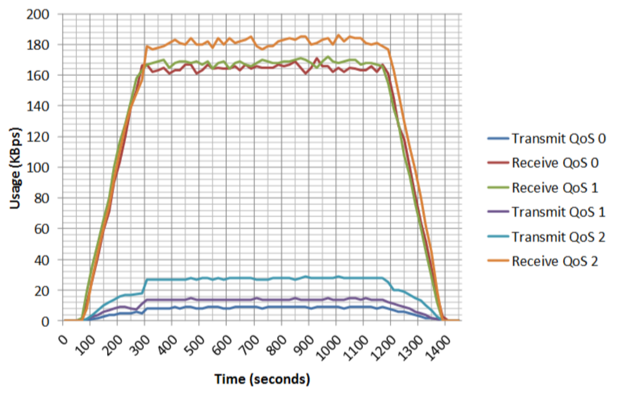


Figure 11. Scenario 1 Single Subscriber Network Usage

Overall the results show that while increasing the QoS level has a significant effect of the resources consumed by the broker, the effect is much less apparent on the individual subscribing nodes. The results from the single subscriber experiments show that as the number of connections increases the cost of higher QoS levels also increases. All results show that the jump from QoS 1 to 2 is much larger than QoS 0 to 1 in terms of processing and networking resources.

TABLE II. CPU usage of the nodes and the host

MQTT QoS Level	CPU per Node (%)	CPU VM Host (%)
0	0.971	69.0
1	0.977	69.7
2	1.022	71.1

B. *Scenario 2 - Multi-publisher, multi-subscriber*

Figure 12 shows that changing the network topology had little effect on the broker's CPU usage compared to Figure 8. Figure 13 however, does show a significant difference.

In this scenario the data received rate matches the data transmitted rate almost exactly where as in scenario 1 Figure 9 shows that the receive rate was always significantly higher than the transmit rate for a given QoS level. This is due to the single subscriber being overwhelmed by the large flow of traffic being directed at it. Where as in the second scenario this load is distributed across a much larger number of virtual CPUs.

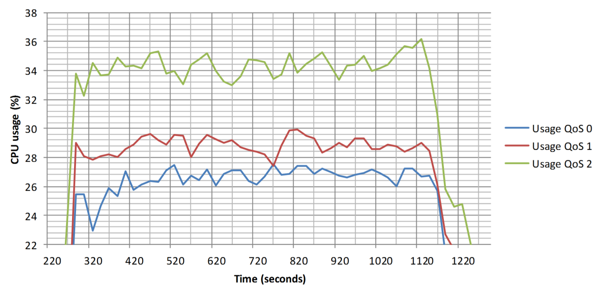


Figure 12. Scenario 2 Broker CPU Usage

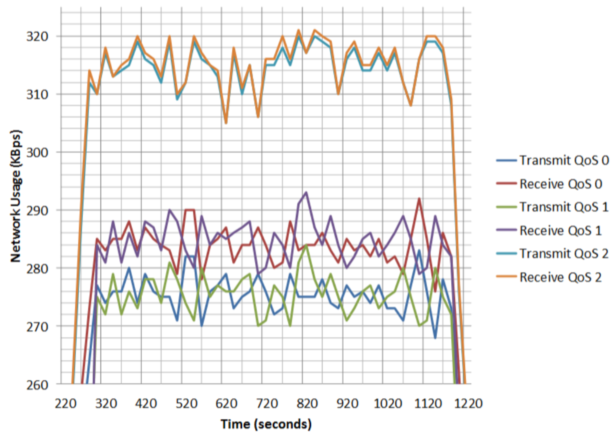


Figure 13. Scenario 2 Broker Network Usage

C. *Packet Level Analysis*

Using TCPdump to collect the traffic passing over the bridge and then Wireshark to analyse it, the following results were obtained.

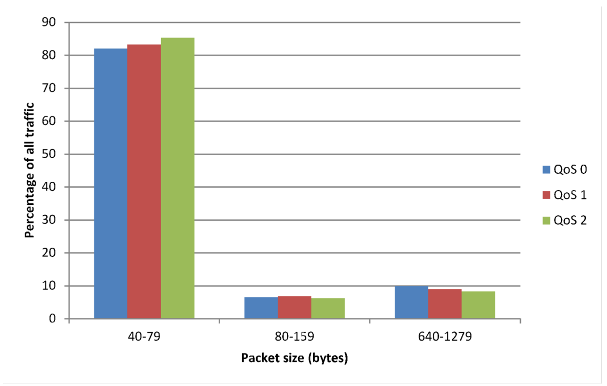


Figure 14. Scenario 1 Distribution of packet sizes

Figure 14 shows that as the QoS level is increased the distribution of packets sizes shifts to having a larger number of smaller packets and fewer larger ones. The packets lying in the 40-79 byte range are either ARP packets (less than 0.05%) or TCP control messages such as individual ACK, SYN, RST, FIN packets. The 80-159 bytes range are MQTT control packets that have very small payloads such as CONNECT, DISCONNECT or SUBSCRIBE etc. The 640-1279 range are exclusively the PUBLISH packets containing the example XML data. This indicates the large majority of packets are actually TCP control packets, i.e., over 80% of all packets regardless of QoS level are TCP overhead.

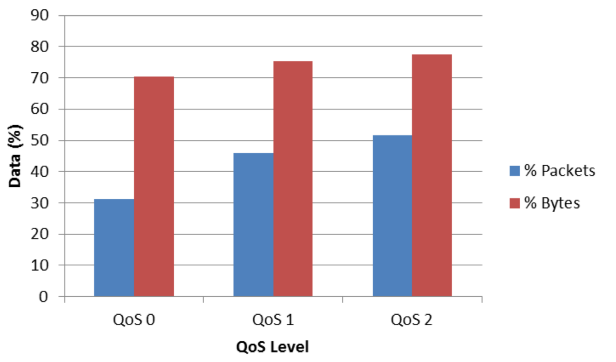


Figure 15. Scenario 1 Proportion traffic that is TCP payload (goodput)

Figure 15 also supports this analysis. At QoS level 1 only 30% of packets actually had a data payload. This includes all of the MQTT packets as well as a large number of PSH-ACK packets with 2-4 byte payloads.

Finally, the use of the Intel Gigabit ET Dual Port Server Adapter presented no noticeable issues. The driver included in the Linux 2.6.32 kernel showed no compatibility issues and its use validated the HIL facility of the test bed.

The results presented are as would be expected. MQTT Level 2 consumes the most bandwidth and CPU time, followed by Level 1 and then Level 0 consumes the least. The results do however provide a valuable step in validating the emulation test-bed. The test-bed was able to produce accurate and detailed results which can actually be used to comment on the appropriateness of using MQTT in resource constrained environments. The main issue is that MQTT operates on top of TCP and that the MQTT replicates large portions of TCP functionality. This means that for DRE environments, MQTT is arguably not well suited. For example, QoS 0 is supposed to be for non-critical messages. There is no confirmation or guarantee that this message is received at the application level. But due to TCP, at the transport level, there inherently is. Normally a low priority, low QoS level message, should provide a low overhead method of communicating. However, due to operating on top of TCP there is in fact a very good chance that the message will be delivered due to TCP's inherent reliable transmission mechanisms. This reduces the usefulness of all the application level QoS levels. Low priority messages do not benefit from as much overhead reduction as they could and high priority messages are partially redundant and overhead is increased with little gain. If MQTT were to not operate over TCP exclusively some transactions would be more efficient in terms of network resources and therefore more appropriate for resource constrained environments. For example, UDP with no reliability features and therefore low overhead, would seem to be a better choice for MQTT QoS 0.

IV. Multi-Protocol QoS: Experimental Scenarios

Using the validated flexible emulation test-bed, vNET, the network topology shown in Figure 16 was configured. The topology is a simple fan out type network where one node is distributing data to a group of 300 nodes representing consumer smart meters.

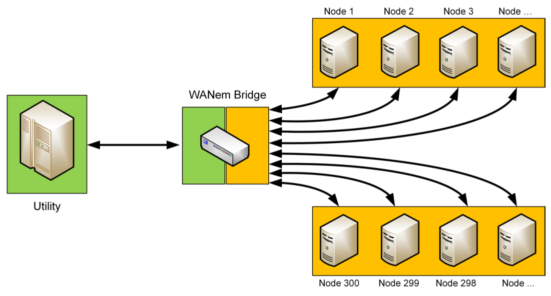


Figure 16. Test bed network topology. 300 nodes are connected to a utility system through a software Ethernet bridge. The Utility publishes the update.

The topology represents the logical grouping that could be used in a Real Time Pricing DSM operation [41]. Traffic shaping is provided by WANem [32], which is a software wide-area network emulator. It provides the ability to manipulate many common network characteristics including available bandwidth, latency and packet loss. The network connection between the utility system and the software Ethernet bridge utilises a real hardware Intel gigabit ET dual port server adapter looped back on its self. This presents a proof-of-concept for HIL in the test-bed.

A. Selected Communication Paradigms

Four viable interaction models were chosen to experiment with; these are shown in Tables III and IV. All are tested with both ideal and resource constrained, lossy network conditions in a set of eight experiments.

TABLE III. - Downlink (utility to consumers) transport choices

Scenario	Transport Protocol	Messaging Pattern
1	TCP	Router / Dealer
2	TCP	Publish / Subscribe
3	PGM	Publish / Subscribe
4	UDP	Request / Response

TABLE IV. Uplink (consumers to utility) transport choices

Scenario	Transport Protocol	Messaging Pattern
1	TCP	Request / Response
2	TCP	Request / Response
3	TCP	Request / Response
4	UDP	Request / Response

Router / Dealer is a tightly-coupled request-response style messaging pattern belonging to the ZeroMQ [42] socket API. It allows messages prefixed with a globally unique identifier (GUID) to be routed to a socket, remote or local, which has that same GUID. Each message sent needs to be prefixed with a valid GUID of a node, which requires additional initialisation steps in order to acquire this information. Publish / Subscribe is a loosely coupled data distribution style messaging pattern. A publisher publishes a message prefixed with a topic / channel identifier. Only subscribers which have confirmed their interest in messages belonging to this topic / channel get the message routed to them. Scenarios 2 and 3 both use publish / subscribe but they use different transport protocols. Scenario 2 uses standard TCP. TCP is a unicast transport which implies that if a pricing update is to be sent to 300 nodes the Utility node will have to generate and send 300 individually addressed packets (assuming no fragmentation). Conversely, Pragmatic General Multicast [43] (PGM) is an experimental IETF (Internet Engineering Task Force) transport protocol designed to provide reliable multicast communications. In this case, the utility generates only a single packet (again assuming no fragmentation). Whereas TCP and PGM are both reliable transport protocols, UDP is unreliable. It does not have any mechanisms for ensuring reliable delivery but this does mean that it exhibits a lower network overhead.

B. Link configurations of selected network scenarios

Table V shows the network condition scenarios used in conjunction with the scenarios shown in Tables III and IV.

TABLE V. Network Conditions

Network Condition	Description
Ideal	No restrictions on bandwidth (10Gbps nominal effectively unlimited) or any additional latency or packet loss
Resource Constrained	Bandwidth limited to 250Kb/s, additional 30ms +/- 5ms latency and 30% packet loss.

The resource constrained experiment emulates specific network conditions and represents a hypothetical resource-constrained lossy network on the link from the consumers to the utility such as an IEEE 802.15.4 based solution. Even though this represents two opposite extreme scenarios the results would still support the conclusions made for other network conditions. Further experimental details are:

1) In all experiments, the application layer maximum transmission unit (MTU) was configured for each transport protocol to ensure the packet size on the wire did not exceed 127 bytes. This was done to emulate the larger transport overhead (due to fragmentation) that would be seen when using these transport protocols with data link layers that can only support small packet sizes.
2) The virtualised Ethernet bridge interface cards were configured for half duplex communication in order to emulate a half-duplex radio link.
3) The payload used was a 1699 byte Extensible Markup Language (XML) string which is compatible with the OpenADR EventState.xsd XML schema [44].
4) A price update was issued every 0.15 seconds in the request / response architectures and every 45 (0.15*300 = 45) seconds for the publish / subscribe architectures. This approach produces comparable test results as the fundamental differences in how the data is distributed between request / response and publish / subscribe would otherwise make this difficult. All scenarios achieve the goal of generating the same total number of responses from the consumers.
5) All experiments issued price updates for up to 90 seconds and generated 600 responses from the consumers. Tests were allowed to run until all inflight responses were obtained.

The frequency of the Real Time Pricing (RTP) update is higher than any real world application. However, as the number of packets being generated, and hence the congestion, vary linearly with the RTP update frequency, using this frequency simply allows results to be collected easier. The higher frequency has no effect on the conclusions that are made in these experiments.

V. Multi-Protocol QoS: Experimental Results

The results in this section use the UDP scenarios as a baseline. The raw results are shown in Table VI. This is done due to the UDP scenarios representing the simplest combination being experimented with. By using this scenario as benchmark it is easier to see how the other combinations perform in the given network topology against a well understood, ubiquitous transport protocol.

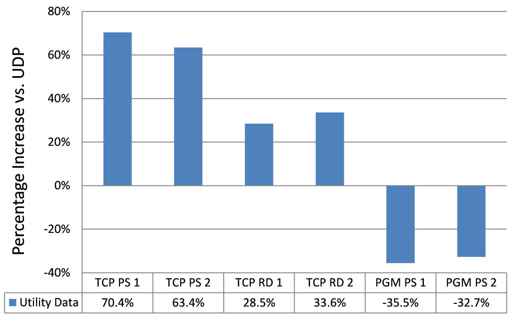
TABLE VI. The raw UDP results that can be used for comparison when results are shown as percentage increases.

	Ideal	Constrained
Utility Data	1.424 MBytes	1.424 MBytes
Consumer Data	1.424 MBytes	1.202 MBytes
Overhead	586.080 KBytes	586.080 KBytes
Message Round Trip Delay	2.183 ms	46.814 ms
Message Loss	0 %	39.3 %

TABLE VII. Message Latency Round Trip Delay (RTD) and Message Loss (PS: Publish / Subscribe, RD: Router / Dealer, RR: Request / Response)

Experiment Number	Message Round Trip Delay (RTD) (ms)	Message Loss (%)
1. TCP PS Ideal	345.6	0.00
2. TCP PS Constrained	21500.0	0.33
3. TCP RD Ideal	5.4	0.00
4. TCP RD Constrained	604.3	0.00
5. PGM PS Ideal	363.7	0.00
6. PGM PS Constrained	17040.0	0.33
7. UDP RR Ideal	2.2	0.00
8. UDP RR Constrained	46.8	39.30

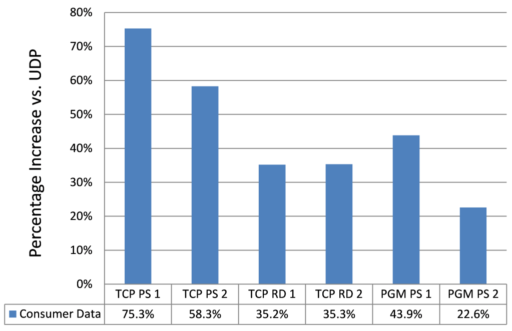
Under the lossy, congested network conditions it took on average 17.04 seconds to complete a round trip for the PGM experiment (Experiment 6) and 21.5 seconds for the TCP (Experiment 2). This is extremely high and is due to the way the consumers are responding; the PGM and TCP publish / subscribe consumers use the same TCP request response architecture to respond with. There is no rate limiting which is causing a large amount of congestion. PGM provides an interface to limit the multicast data rate which would be very useful in this case. It can also be seen that even under perfect network conditions, rate unlimited publish / subscribe architectures are not suitable for applications requiring low latency as individual message delays are over 150 times that of the UDP case (Experiment 1 and 5 vs. 7). The results indicate that the publish / subscribe architectures need a mechanism for rate limiting publishes and responses. The congestion generated when a published pricing update is sent and then responded to



	TCP PS 1	TCP PS 2	TCP RD 1	TCP RD 2	PGM PS 1	PGM PS 2
Utility Data	70.4%	63.4%	28.5%	33.6%	-35.5%	-32.7%

Figure 17. Data sent by the Utility system compared to the UDP scenario.

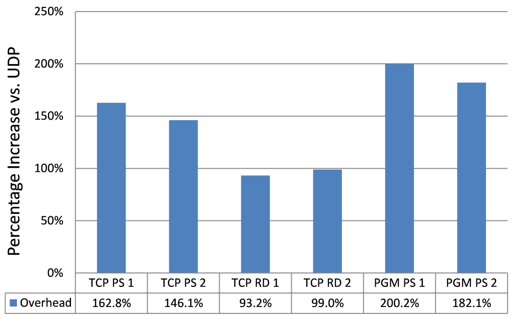
by all of the consumers simultaneously quickly overwhelms the resource constrained network generating message losses, which in turn cause retransmissions which contribute to the large amount of data generated. This can be seen in Figures 17 and 18. The TCP publish / subscribe scenario shows this better than the PGM scenario. In this scenario the resource constrained, lossy network test actually performs better than the ideal case as the artificially imposed packet delay is having the effect of limiting the packet rate which even with the 30% packet loss and the retransmissions this would introduce, causes the scenario to generate less traffic than the ideal case. This also indicates that a component in the virtual network is being stressed to the point of packet loss under the high packet rates being generated by the low MTU. Even with acknowledging this it shows that high messages rates with relatively large payload compared to the MTU will cause worse congestion problems than 30% packet loss does.



	TCP PS 1	TCP PS 2	TCP RD 1	TCP RD 2	PGM PS 1	PGM PS 2
Consumer Data	75.3%	58.3%	35.2%	35.3%	43.9%	22.6%

Figure 18. Data sent by the consumer nodes compared to the UDP scenario.

The latency (Tables VI, VII, Experiment 7 and 8) and overhead (Figure 19) results show the UDP experiments outperform both the TCP or PGM equivalents with the TCP. This is not unexpected given the TCP and PGM are both reliable transports, with retransmissions that introduce increased delays against UDP. The notable observation is the performance gap between them. TCP is a generic transport capable of serving many different application requirements quite adequately, but the overhead involved in being so generic is clearly shown in these experiments. There is a clear opportunity to bridge this large gap with a number of UDP based messaging patterns, both unicast and multicast, and apply various application layer reliability mechanisms to them. This would allow applications access to a range of communications service combinations at a higher granularity, so that applications can get a communication service with only the features they need and avoid the general overhead of a one-transport-fits-all approach. Figure



	TCP PS 1	TCP PS 2	TCP RD 1	TCP RD 2	PGM PS 1	PGM PS 2
Overhead	162.8%	146.1%	93.2%	99.0%	200.2%	182.1%

Figure 19. Protocol overhead compared to the UDP scenario measured as any data that was not the XML payload. (Percentage increase is shown)

19 shows a large PGM percentage overhead. Given the much lower overall bandwidth consumed using multicast, this is to be expected. Overhead is calculated as any bytes put onto the wire that are not part of the XML payload. In order to generate the 600 responses (the experimental scenario criteria) from the consumers the utility only has to generate 2 PGM packets (ignoring fragmentation due to the low MTU). The overhead is almost entirely due to the TCP request response uplink from the consumer to the utility. Normalising these results against the total data exchanged shows the PGM architectures are in fact the most efficient next to the UDP architectures. The results show that even though TCP publish / subscribe would appear to be a potential choice for this type of scenario (data distribution with a large fan-out) given that on face value it appears to provide the necessary interface for providing efficient data distribution, it actually performed the worst. TCP-based publish / subscribe involves a high amount of overhead to effectively allow a unicast architecture to emulate services that require a multicast architecture in order to operate efficiently. It provides no network orientated benefit over TCP Router / Dealer. The only benefit it provides is the ability to distribute messages at a more abstract level due to the use of topics / channels. In fact the lack of control on the distribution rate of the messages means that TCP router / dealer is more flexible and consistently generates less overhead and congestion as can be seen in Figures 17-19.

VI. Conclusions

This paper has presented and validated an argument for exploiting the performance gains achievable by specifically selecting application appropriate transport protocols dynamically at runtime based on specific application requirements. Given the varied network requirements demanded by SG applications and DRE applications in general this approach provides previously inaccessible optimisation opportunities. Furthermore, these gains are achievable without the need to perform costly modifications to any intermediate network infrastructure and would only require modifications to existing networked applications network interfacing code. The cost of this modification could be mitigated by using a middleware system for managing the transport selection. To summarise, the results have shown:

1) For an ideal RTP update distribution use case PGM publish / subscribe and UDP request / response

should be used on the down links and up links respectively for best performance and lowest resource utilisation.

2) For the non-ideal case, the unreliable UDP is only viable if the application can suffer lost responses from the consumer this is a possible scenario. If more reliability is required then another low overhead reliable transport should be used with TCP based options used as a last resort.
3) There is a significant gap between the performance of the Reliable TCP / PGM scenarios and the unreliable UDP scenario in terms of overhead and latency. Additional transports are needed to fill the gap.
4) TCP based Publish / Subscribe provides no network level benefits.
5) Rate unlimited Publish / Subscribe is not viable for applications with a low latency requirement. The packet rate needs to be limited at the point of transmission in order to ensure congestion is not generated.

A large number of additional supported transport protocols, would make it possible for a system to generate custom network interfaces for a much wider range of scenarios in order to improve application performance through manipulation at the transport level. Future work will consider how to automatically manage the large number of potential transport protocol choices which are being suggested using middleware solutions. The virtualisation based network emulation test-bed has been shown to produce useful and viable results. The unique features that the test-bed offers has provided the level of detail normally reserved for scaled real world trials but at a much lower cost. Collecting individual CPU, bandwidth and packet statistics for each node and the overall system has been shown to be quick and convenient. Reconfiguring the test-bed for different topologies can be achieved by consuming much less time and resources compared to a scaled real world trial. Additionally a hardware-in-loop (HIL) proof-of-concept has been presented. The ease at which the technology allows hardware to be assigned and used natively by any virtualized node presents a powerful hardware / driver evaluation tool. The other tools highlighted in this work cannot provide a feature set optimised for the IoT and other distributed, real-time and embedded environments. Therefore, the development of vNET has been justified.

References

[1] J. Wilcox, D. Kaleshi, and M. Sooriyabandara, “Multi-Protocol Transport Layer QoS: A Performance Analysis for The Smart Grid,” in *IARIA ENERGY 2014, The Fourth International Conference on Smart Grids, Green Communications and IT Energy-aware Technologies*, pp. 13–18.

[2] C. Fraleigh, S. Moon, B. Lyles, C. Cotton, M. Khan, D. Moll, R. Rockell, T. Seely, and S. C. Diot, “Packet-level traffic measurements from the sprint ip backbone,” *Network, IEEE*, vol. 17, no. 6, pp. 6–16, 2003.

[3] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, “RTP: A Transport Protocol for Real-Time Applications.” RFC 3550 (INTERNET STANDARD), July 2003. Updated by RFCs 5506, 5761, 6051, 6222, 7022.

[4] Internet Assigned Numbers Authority, “Protocol Numbers.” http://www.iana.org/assignments/protocol-numbers/protocol-numbers.xml. Accessed: 24/03/14.

[5] J. Wilcox, D. Kaleshi, and M. Sooriyabandara, “Director: A distributed communication transport manager for the smart grid,” in *Communications (ICC), 2014 IEEE International Conference on*, pp. 4227–4232, June 2014.

[6] A. Mohsenian-Rad, V. W. S. Wong, J. Jatskevich, R. Schober, and A. Leon-Garcia, “Autonomous demand-side management based on game-theoretic energy consumption scheduling for the future smart grid,” *IEEE Transactions on Smart Grid*, vol. 1, no. 3, pp. 320–331, 2010.

[7] A. Gabaldon, A. Molina, C. Roldan, J. Fuentes, E. Gomez, I. Ramirez-Rosado, P. Lara, J. Dominguez, E. Garcia-Garrido, and E. Tarancon, “Assessment and simulation of demand-side management potential in urban power distribution networks,” in *IEEE Bologna Power Tech Conference Proceedings, 2003*, vol. 4, pp. 5 pp. Vol.4–, June 2003.

[8] M. LeMay, R. Nelli, G. Gross, and C. Gunter, “An Integrated Architecture for Demand Response Communications and Control,” in *Proceedings of the 41st Annual Hawaii International Conference on System Sciences*, pp. 174–174, Jan 2008.

[9] C.-L. Su and D. Kirschen, “Quantifying the Effect of Demand Response on Electricity Markets,” *Power Systems, IEEE Transactions on*, vol. 24, pp. 1199–1207, Aug 2009.

[10] Z. M. Fadlullah, M. M. Fouda, N. Kato, A. Takeuchi, N. Iwasaki, and Y. Nozaki, “Toward intelligent machine-to-machine communications in smart grid,” *Communications Magazine, IEEE*, vol. 49, no. 4, pp. 60–65, 2011.

[11] U. Mutlu, R. Edwards, and P. Coulton, “Qos aware bluetooth middleware,” in *Information and Communication Technologies, 2006. ICTTA ’06. 2nd*, vol. 2, pp. 3239–3244.

[12] M. Henning, “The Rise and Fall of CORBA,” *Queue*, vol. 4, no. 5, pp. 28–34, 2006.

[13] Z. Weishan, K. M. Hansen, J. Fernandes, Schu, x, J. tte, and F. M. Lardies, “Qos-aware self-adaptation of communication protocols in a pervasive service middleware,” in *Green Computing and Communications (GreenCom), 2010 IEEE/ACM Int’l Conference on Cyber, Physical and Social Computing (CPSCom)*, pp. 17–26.

[14] K. Young-Jin, L. Jaehwan, G. Atkinson, K. Hongseok, and M. Thottan, “SeDAX: A Scalable, Resilient, and Secure Platform for Smart Grid Communications,” *IEEE Journal on Selected Areas in Communications*, vol. 30, no. 6, pp. 1119–1136, 2012.

[15] Z. Wenjie, Y. Nanhua, C. Hui, H. Jiajian, L. Chuanjian, and L. Xin Jie, “Providing adaptive QoS for Real-Time Publish-Subscribe Service in SGIOC-HQ,” in *Innovative Smart Grid Technologies - Asia (ISGT Asia), 2012 IEEE*, pp. 1–5.

[16] B. Zieba, M. v. Sinderen, and M. Wegdam, “Quality-constrained routing in publish/subscribe systems,” 2005.

[17] R. E. Schantz, J. P. Loyall, C. Rodrigues, D. C. Schmidt, Y. Krishnamurthy, and I. Pyarali, “Flexible and adaptive QoS control for distributed real-time and embedded middleware,” 2003.

[18] Scalable Network Technologies, “Qualnet.” http://web.scalable-networks.com/content/qualnet, 2012. Accessed: 24/03/14.

[19] G. F. Riley and T. R. Henderson, “The NS-3 Network Simulator Modeling and Tools for Network Simulation,” in *Modeling and Tools for Network Simulation*, ch. 2, pp. 15–34, Springer Berlin Heidelberg, 2010.

[20] S. Doshi, U. Lee, R. Bagrodia, and D. McKeon, “Network design and implementation using emulation-based analysis,” in *Military Communications Conference, 2007. MILCOM 2007. IEEE*, pp. 1–8.

[21] K. Fall, “Network emulation in the vint/ns simulator,” in *IEEE International Symposium on Computers and Communications, 1999. Proceedings.*, pp. 244–250.

[22] S. Maier, A. Grau, H. Weinschrott, and K. Rothermel, “Scalable network emulation: A comparison of virtual routing and virtual machines,” in *12th IEEE Symposium on Computers and Communications, 2007. ISCC 2007.*, pp. 395–402.

[23] T. Song, S. Wen-Zhan, D. Qifen, and T. Lang, “Score: Smart-grid common open research emulator,” in *IEEE Third International Conference on Smart Grid Communications (SmartGridComm), 2012*, pp. 282–287.

[24] B. Lantz, B. Heller, and N. McKeown, “A network in a laptop: Rapid prototyping for software-defined networks,” in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, Hotnets-IX, (New York, NY, USA), pp. 19:1–19:6, ACM, 2010.

[25] F. Osterlind, A. Dunkels, J. Eriksson, N. Finne, and T. Voigt, “Cross-Level Sensor Network Simulation with COOJA,” in *Proceedings 2006*

31st IEEE Conference on Local Computer Networks, pp. 641–648, Nov 2006.

[26] A. Dunkels, B. Gronvall, and T. Voigt, "Contiki. A lightweight and flexible operating system for tiny networked sensors," in *29th Annual IEEE International Conference on Local Computer Networks, 2004*, pp. 455–462, Nov 2004.

[27] "VMware vSphere 5.1 Configuration Maximums." http://www.vmware.com/pdf/vsphere5/r51/vsphere-51-configuration-maximums.pdf. Accessed: 24/03/14.

[28] "Introduction to Mininet ." https://github.com/mininet/mininet/wiki/Introduction-to-Mininet. Accessed: 24/03/14.

[29] VMware, "Esxi overview." http://www.vmware.com/products/vsphere-hypervisor. Accessed: 24/03/14.

[30] C. M. Buechler and J. Pingle, *pfSense: The Definitive Guide*. Reed Media Services, 2009.

[31] G. Ghatikar, "Open automated demand response technologies for dynamic pricing and smart grid," *Lawrence Berkeley National Laboratory*, 2010.

[32] H. K. Kalitay and M. K. Nambiar, "Designing WANem : A Wide Area Network emulator tool," in *Third International Conference on Communication Systems and Networks (COMSNETS), 2011*, pp. 1–4.

[33] "Tcpdump." http://www.tcpdump.org/manpages/tcpdump.1.html. Accessed: 24/03/14.

[34] W. Foundation, "Wireshark." http://www.wireshark.org. Accessed: 24/03/14.

[35] D. Locke, "MQ Telemetry Transport (MQTT) V3.1 Protocol Specification." http://public.dhe.ibm.com/software/dw/webservices/ws-mqtt/MQTT_V3.1_Protocol_Specific.pdf, 2010. Accessed: 24/03/14.

[36] O. Fenton, "WebSphere MQ Telemetry V7.0.1 - Performance Evaluation." ftp://public.dhe.ibm.com/software/integration/support/supportpacs/individual/mp0a.pdf, 2010. Accessed: 24/03/14.

[37] J. Perez, "MQTT Performance Analysis with OMNeT++." Networking Insitut Eurecom; IBM Zurich Research Laboratory Switzerland, September 2005.

[38] "OMNeT++, an extensible simulation library." http://www.omnetpp.org/. Accessed: 24/03/14.

[39] R. Light, "Mosquitto. An Open Source MQTT v3.1 Broker." http://linuxcommand.org/man_pages/brctl8.html. Accessed: 24/03/14.

[40] Z. S. Organization, "ZigBee Smart Energy Profile Specification," tech. rep., 01/12/08 2008.

[41] Z. Wei and A. Feliachi, "Residential load control through real-time pricing signals," in *Proceedings of the 35th Southeastern Symposium on System Theory.*, pp. 269–272.

[42] P. Hintjens, "ZeroMQ: The Guide." http://zguide.zeromq.org/, 2010. Accessed: 24/03/14.

[43] T. Speakman, J. Crowcroft, J. Gemmell, D. Farinacci, S. Lin, D. Leshchiner, M. Luby, T. Montgomery, L. Rizzo, A. Tweedly, N. Bhaskar, R. Edmonstone, R. Sumanasekera, and L. Vicisano, "PGM Reliable Transport Protocol Specification." RFC 3208 (Experimental), Dec. 2001.

[44] OpenADR, "OpenADR EventState.xsd XML schema." http://openadr.lbl.gov/src/EventState.xsd. Accessed: 24/03/14.