Live Video Transmission over Data Distribution Service with Existing Low-Power Platforms

A. Bagley
Northumbria University
The Department of Computer and Information Sciences,
Northumbria University, Newcastle upon Tyne NE1 8ST, U.K.
Anthony.bagley@northumbria.ac.uk

G. Fehringer
Northumbria University
The Department of Computer and Information Sciences,
Northumbria University, Newcastle upon Tyne NE1 8ST, U.K.
Gerhard.fehringer@northumbria.ac.uk

N.Jin
Northumbria University
The Department of Computer and Information Sciences,
Northumbria University, Newcastle upon Tyne NE1 8ST, U.K.
nanlin.jin@northumbria.ac.uk

S Cammish
Prismtech
Prismtech Houe, 5th Ave,
Gateshead
steve.cammish@prismtech.com

ABSTRACT

This paper investigates video transmission over a middleware layer based on the Object Management Group's Data-Distribution Service (DDS) standard, with a focus on low power platforms. Low power platforms are being widely utilised to implement IoT devices. One important type of IoT application is live video sharing which requires higher bandwidth than current typical applications. However, only limited research has been carried out on quality of services of data distribution utilising low end platforms.

This paper discusses the development of prototypes that consist of both a Raspberry Pi 2 and an Android smartphone with client applications using Prismtech's Vortex line of DDS middleware. Experiments have yielded interesting performance results: DDS middleware implementations that run on low power hardware with native code can provide sufficient performance. They are efficient enough to consistently handle high bandwidth live video with the network performance proving to be the bottleneck rather than the processing power of the devices. However, virtual machine implementations on an Android device did not achieve similar performance levels.

These research findings will provide recommendations on adopting low power devices for sharing live video distribution in IoT over DDS middleware.

Categories and Subject Descriptors

[**Software and its engineering]**: Message oriented middleware

General Terms

Measurement, Performance, Design, Experimentation

Keywords

Data Distribution Service, Middleware, DDS performance, Live Video, Raspberry Pi, Android.

1. INTRODUCTION

The Internet of Things (IoT) is an area of computing application with rapid commercial interest and growth in current times. The concept of connecting many data producing and consuming devices in different environments and combining real-time data analysis has yielded the ideas of smart cities and smart factories, which are now becoming a reality. Video analysis is being coupled with the camera devices as part of the IoT, to better monitor vast and busy environments and automatically draw conclusions without human interaction. This could include real-time detection of visual flaws in manufactured products such as small hardware components. In smart cities live video analysis could be used to detect vehicles breaking road laws, such as occupying bus lanes or making illegal maneuvers. However, such systems require a large amount of devices, a robust networking setup, scope for upgradability and 100% uptime with consistent performance. DDS can be an effective solution for live video transfer between many different devices over a variety of different connections but can current low-power platforms meet the demands of reliable, high data throughput for live video delivery?

PrismTech's Vortex Lite provides "low latency, real-time data sharing for resource constrained Internet of Things (IoT) devices and environments with limited memory and processing capabilities" [1]. The performance of this lightweight implementation will be tested with live video via a DDS test application on a Raspberry Pi 2 and compared to the performance of an Android smartphone with a comparable DDS implementation using PrismTech's Vortex Café. Both devices are chosen to explore how current low end/ mobile hardware handles the distribution of high bandwidth video via DDS middleware. An investigation into the performance of high-bandwidth, real-time, video transmission via middleware with low power hardware is needed to evaluate how a worst case scenario performs and identify any associated bottlenecks/ limiting factors.

2. PLATFORMS

2.1 Raspberry Pi 2 (Low Power Hardware)

The low power hardware platform chosen is the Raspberry Pi 2 model B with the Pi Camera module for video capture (See Table 1). This platform has been chosen for its software development flexibility and hardware backed video capture support for the efficiency that would be expected of a dedicated IoT camera device. This platform will be running the Raspbian Linux OS to enable swift application development and to run a custom build of Vortex Lite DDS middleware.

Table 1. Raspberry Pi Hardware Specification

CPU	Rapberry Pi 2
RAM	ARM Cortex A7 quad core processor overclocked at 1000MHz
GPU	1GB
Network Support	VideoCore IV GPU @250MHz, allocated 256 MB of memory
CPU	100Mbps Ethernet

The Pi Camera is specifically developed to work with the Raspberry Pi's GPU, which provides hardware accelerated H.264 and JPEG encoding. The main point of interaction with the Pi Camera will be via the V4L2 driver, with application development using the C++ programming language

2.2 Android Smartphone (Mobile Device)

For The Android smartphone used for this investigation will be the Sony Xperia Z3 Compact smartphone, running Android 6.0.1 (See Table 2).

Table 2. Android Smartphone Specification

	Sony Xperia Z3 Compact
CPU	Snapdragon 801 2.5 GHz Quad-core (Krait 400)
RAM	2GB
GPU	Adreno 330 GPU @578MHz
Network Support	Qualcomm® VIVE™ 1-stream 802.11n/ac with MU-MIMO 433Mbps

2.3 Linux Desktop

A generic Desktop computer running Linux Mint 17.3 Rosa is used for subscribing DDS applications and is sufficiently powerful to ensure it does not impose a bottleneck in the system (See Table 3). In a production system this could be replaced with a server on the edge or even the Cloud with sufficiently high bandwidth Internet connection.

Table 3. Generic Debian based Linux Desktop

	Dell Optiplex 390
CPU	Intel® Core™ i3-210 CPU @3.3GHz 2 cores, 4 threads
RAM	3.8GB
Network Support	Gigabit Ethernet

2.4 Network Environment

DDS defines a data centric publish and subscribe data transfer model and is an OMG standard. DDS middleware uses a Global Data Space (GDS) which provides distributed subscriber access to any published data without a centralised copy of the actual data; hence it is a distributed system. The publisher/subscriber distributed application is composed of processes, each running in a separate address space usually on different connected devices [2]. Data is transferred over the fastest medium between publisher and subscriber, be it across device memory, LAN or WAN including the Internet. The Data-Centric Publish-Subscriber (DCPS) layer of the DDS standard focuses on efficient receipt of information by the correct recipients and is the layer exposed by PrismTech's Vortex DDS API; the Data-Local Reconstruction Layer (DLRL) is not used for this investigation.

3. DDS MIDDLEWARE OVERVIEW

DDS defines a data centric publish and subscribe data transfer model and is an OMG standard. DDS middleware uses a Global Data Space (GDS) which provides distributed subscriber access to any published data without a centralised copy of the actual data; hence it is a distributed system. The publisher/subscriber distributed application is composed of processes, each running in a separate address space usually on different connected devices [2]. Data is transferred over the fastest medium between publisher and subscriber, be it across device memory, LAN or WAN including the Internet. The Data-Centric Publish-Subscriber (DCPS) layer of the DDS standard focuses on efficient receipt of information by the correct recipients and is the layer exposed by PrismTech's Vortex DDS API; the Data-Local Reconstruction Layer (DLRL) is not used for this investigation.

Each device that intends to publish and/or subscribe to data in the DDS global data space is required to have a DDS participant which acts as a factory to create all DDS entities that operate in the GDS domain (such as publishers/writers, subscribers/readers). The most important part of the DDS publish/subscribe system are the data definition models used to express the data 'topics' to be shared (i.e. their names, structures and Quality-of-Service policies related to the non-functional properties of the data-sharing). The Quality of Service (QoS) policies can be configured to support aspects such as how data is transmitted, its lifespan in the GDS and many more useful properties. One of DDS middleware's strengths is how it removes the developer burden of data transmission and allows focus to be on the information being shared and what the application should be doing with it. The GDS thus will be populated with samples of these topics where 'matching' readers/writers agree both on the type of a topic (as modeled in the OMG IDL data-definition standard) as well as the specified QoS policies w.r.t. non-functional properties such as

urgency, importance, persistency, reliability etc. Furthermore, the GDS can be partitioned by using the PARTITION QoS-policy which allows further grouping of data.

The Vortex DDS products can be configured to use either TCP or UDP as an underlying communications protocol (See Fig. 1). UDP is typically used on LAN environments where 1-to-n data-distribution is efficiently supported by multicast, TCP is typically used in WAN environments that typically don't support multicast. Note that DDS specifies reliability as a high-level QoS that is applied independently of the underlying transport (implying that DDS implements a reliable-multicast over UDP when required). There are over 20 useful QoS policies outlined in the OMG DDS specification, which allow the tailoring of data delivery limits, data lifespan and data accessibility. Another very useful policy is DURABILITY, which provides a time-decoupling between publishers and subscribers thus allowing late-joining applications to obtain historical data independent of the lifecycle of a publishing application (i.e. non-volatile data will be 'remembered' by DDS thus offering a data-lifecycle that is independent of the lifecycle of the actual publishing/subscribing applications.

Middleware provides a solid bridge between different applications and operating systems, abstracting communication and I/O details for the developer. Prismtech provide a high performance implementation of the latest OMG DDS specification along with a plethora of APIs for different platforms and programming languages.

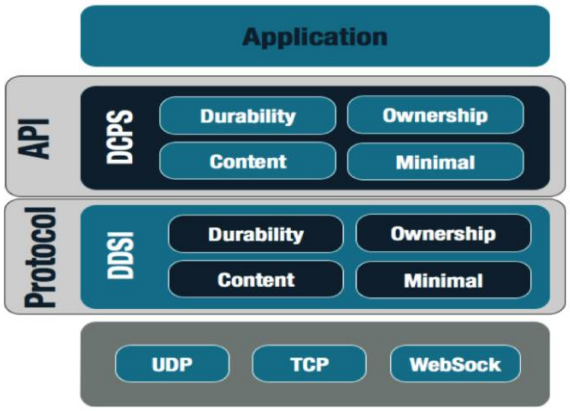

Figure 1: DDS middleware architecture stack.

4. RELATED WORK

Although research has been carried out to explore effective video transmission techniques using a DDS, the platforms and network environments are typically generous in resources or the work focuses on an efficient solution. Such as Detti, Loreti and Blefari-Melazzi [3] who devised a mechanism for streaming H.264 scalable video using multiple topics and rate control based on the connection between publisher and subscriber. Their result was a system that could alter video quality in order to maintain video delivery when network bandwidth becomes restricted. An ad-hoc WLAN providing a connection rate of 5.5Mbps was used with four PCs, showing that this study focused on efficient data usage, leveraging the scalable H.264.

Al-Madina, Al-Roubaiy and Al-Shehari were among the first to properly examine the behaviour of real time video streaming over WLAN using DDS middleware [4]. They constrained the network to 802.11g 54Mbps and used Linux desktop computers for testing network throughput with varying numbers of subscribers. Their conclusion proved that DDS middleware can provide a robust and flexible solution for transmitting video in real time, also noting that the QoS policies provided offer far more control than is traditionally available by typical streaming protocols. Bandwidth usage increased linearly with subscriber growth although jitter was increased when transmitting video compared to smaller data writes.

So it is proven and expected that data throughput across a network may grow linearly with the use of DDS middleware with multiple subscribers, providing UDP multicast is available. Networks can be saturated with data, be it routers, switches or other network nodes but with modern networking configurations the limiting factors are more likely to be the network cards of the client devices. In order to use DDS middleware to deliver high quality live video, the implementation must be able to efficiently write data into a network making the most of the available bandwidth. This may lead to bottlenecks in one of three areas. The device's network throughput which may be limited by hardware/OS, the DDS middleware implementation used and the efficiency of the application using the DDS middleware.

Garcia-Valls, Basanta-Val and Estévez-Ayres tested RTI's implementation of DDS middleware to see the average delay of message writes but only tested messages up to a maximum size of 1000 bytes which took on average 0.103ms [5]. Proposed experiments for this study will be using high bandwidth video in which sample sizes will be significantly larger to simulate worst case scenarios of high bandwidth video. Testing sample sizes up of 500KB or more may yield interesting results as to how the DDS middleware copes on low power platforms.

5. INVESTIGATION

To explore the performance of live video via DDS on both a Raspberry Pi and an Android smartphone, three test applications have been developed which are all interoperable. The first was for the Raspberry PI, implemented using C++ with PrismTech's Vortex Lite DDS product with ISOCPP API. The V4L2 Linux driver was used to access the video camera data and threading implemented to separate frame capture from DDS data writes. Each frame was published into the DDS global space as soon as it was available. The test application supports both H.264 and MJPEG video for flexibility but MJPEG was chosen as a worst case scenario for large data writes per DDS sample. This could even be compared to a 4K video worst case scenario for streamed data rates, as compressed 4K video streams currently approach up to 26Mbps. This approach also mimics possible real-world implementations that may require a full frame of image data per sample in order to perform image analysis. The V4L2 driver provides fixed image data sizes, of which 128x720 pixels was the larger available, with support for 640x480 pixels also added for data comparison.

The Android app developed targeted the latest version on the Android SDK (version 6.0.1, API 23) and made use of Android's Camera2 API to ensure optimal use of the camera hardware and used PrismTech's Vortex Café Java DDS API. The test app developed also produced MJPEG video with variable frame data rates depending on Android JPEG compression algorithm and the

scenes image complexity. The video size was 1280x720 pixels and a preview of the camera was shown on the devices screen which showed no negative performance issues with the DDS task execution time. Each frame was published to the DDS global space as it was available, making use of background threads where possible. Tuning the JPEG compression algorithm's quality threshold allowed different data sizes for the video frames, enabling investigation into the effect of DDS sample size and the time for DDS data writes to complete.

Finally, a subscribing application running on a generic Linux desktop was implemented using Vortex Lite with ISOCPP, similarly to the publishing service developed for the Raspberry Pi. This service received the video frames from compatible publishers and also displayed the MJPEG video in real-time using OpenCV.

Performance metrics have been tracked using program embedded timing for DDS write times and logging of data sent. Wireshark has been used to measure packet loss and how data is received on the subscribing application end.

Testing the performance of both the publishing applications focuses on the data throughput achieved, comparing DDS sample write time to the amount of data being written and then relating this back to the overall video quality. Two aspects are of interest for measuring video quality, quantitatively the final video frame rate (which is equal to the sample rate) and qualitatively the image fidelity i.e. is the image overly compressed or acceptable as a good representation of the real scene viewed by the camera. From the sample data size and the sample rate, the total data throughput can be calculated and compared to the theoretical maximum of the LAN connection in place to gauge how efficiently network utilisation is occurring. Combing all the data should allow for high level performance assessment of the DDS implementations and possibly highlight areas of improvement or advice for achieving optimal live video via DDS implementations for real-world use. This may include software, hardware and implementation recommendations.

6. RESULTS

In Experimentation of live video via a DDS using a Raspberry Pi 2 found that the Vortex Lite platform offers excellent data publishing write performance with data frame sizes of 921KB and 307KB. Both scenarios achieved 85-88Mb/s continuous network writes on the 100Mb/s Ethernet connection (See Fig. 2). Along with this CPU utilisation was stable at 24% and RAM usage at 6MB. 921KB samples maxed out at 11 samples per second and 307KB samples maxed out at 30 samples per second which was the maximum number of video frames available for dispatch and was limited by the camera device configuration used. The DDS write times were generally consistent with variations in the region of 4-5ms once outliers were excluded.

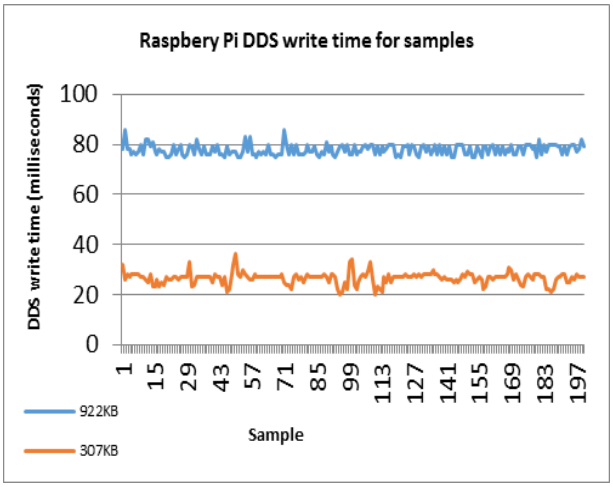

Figure 2: DDS write times for a sequence of video samples on the Raspberry Pi

Due to the high data sample size being transmitted using DDS middleware with UDP being the underlying protocol used; packetisation was very high but the network environment used did not produce any packet loss during tests of 200 sample transmissions.

Multiple tests were carried out with varying video quality and therefore varying video sample data sizes with the Android DDS video app (See Table 4).

Table 2. Table captions should be placed above the table

Average	TestA	TestB	TestC	TestD	TestE
Kilobytes per Sample	17.59	39.67	65.38	92.85	581.98
DDS Write Times (ms)	12.065	26.69	42.93	64.50	361.34
Sample Rate	16	12	11	8	2

The Android app proved to be less consistent in network performance despite efforts to produce a clean, interference free WLAN connection using the 802.11n compliant wireless router and the high performance network chip of the Android smartphone. Network throughput reached a maximum of just 6Mb/s and erratic DDS write times were prevalent throughout all tests. In test C DDS write times varied regularly by upwards of 20ms and that's excluding some more extreme spikes. Test D saw write sizes of around 578KB with DDS write times fluctuating by over 100ms which is edging towards the upper limit of acceptability for live video transmission. Yet this sample size is smaller than what the Raspberry Pi handled, and has a theoretically faster network connection be it a wireless one instead of wired.

7. CONCLUSIONS

At present the Raspberry Pi 2 shows that a cheap, low-power mini SoC style computer is capable of utilising DDS middleware to efficiently transmit high bandwidth video in real time. The

specialized Vortex Lite DDS middleware implementation does an excellent job of operating on a low-power system, with both low memory usage and high network utilization. The middleware coped very well with effectively no packet loss. Overall, current low-power SoC computers are sufficient for real-time, high-bandwidth video transmission

In contrast, the Android platform performed poorly considering its capable mobile hardware and ample mobile networking chip. Bottlenecks in either hardware, the Android OS (including JVM) or the DDS middleware implementation used lead to inconsistent DDS data write times and poor video playback on the subscribing end despite the video samples transmitted being much smaller than what the Raspberry Pi was producing and publishing in real-time. Packet loss was still minimal with only 1% which partially shows that the network environment was stable and unlikely to be of any issue. Compiled native code on low-power hardware is definitely more likely to yield higher DDS middleware performance, based on the testing carried out. The Android platform could not meet the same level of consistent performance when transmitting high-bandwidth video, meaning it is not the best platform for a real-time video. Further experimentation is required to identify the bottleneck encountered.

For the implementation of real-time video via DDS for mass monitoring or video analysis; based on the investigation findings it is recommended that for general or specialized low-power hardware, executing native code should be used. Client devices connected to a network via a wired Ethernet should have a 1Gbps port to alleviate any client side bottlenecks.

8. FUTURE WORK

Although this paper has investigated the current performance of industry standard DDS middleware with the use case of high bandwidth live video, more test cases may be needed. Running as many identical clients as possible on a wider range of both low power, embedded systems and also across a broader spectrum of Java based virtual machine mobile devices is needed. The reason for this is to increase validity and also help isolate the true performance issue that was found with the Android client which executed on a virtual machine. This may highlight hardware shortfalls or OS issues as different Android devices have different hardware and different customized versions of the Android OS, which in itself is continuously evolving and improving.

The conclusion of this paper does show that any mass implementation of live video over DDS, with potentially high bandwidth video should use native applications on hardware that can support video processing. In addition, the network connection requires attention to ensure consistent performance with DDS middleware and especially for future proofing as video standards increase in both resolution and data rate, such as the growth in 4K video.

9. ACKNOWLEDGMENTS

We would like to acknowledge and thank PrismTech (An ADLink Company) for their continued support and access to their DDS middleware product throughout the duration of this project.

10. REFERENCES

[1] Vortex Lite | PrismTech: 2016. http://www.prismtech.com/vortex/vortex-lite. Accessed: 2016- 07- 22.

[2] Pardo-Castellote, G. 2003 OMG Data-Distribution Service: architectural overview. *Distributed Computing Systems Workshops. Proceedings. 23rd International Conference on*, 2003, pp. 200-206. DOI=10.1109/ICDCSW.2003.1203555.

[3] Detti, A. Loreti, P. Blefari-Melazzi, N. and Fedi, F. 2010 Streaming H.264 scalable video over data distribution service in a wireless environment. *World of Wireless Mobile and Multimedia Networks (WoWMoM), 2010 IEEE International Symposium on a, Montreal.* (QC, Canada, 2010). pp. 1-3. DOI=10.1109/WOWMOM.2010.5534937.

[4] Al-madani, B. Al-Roubaiey, A. and Al-shehari, T. Wireless video streaming over Data Distribution Service middleware. *2012 IEEE International Conference on Computer Science and Automation Engineering, (*Beijing, 2012). pp. 263-266. DOI=10.1109/ICSESS.2012.6269456

[5] García-Valls, M. Basanta-Val, P. and Estévez-Ayres, I. Adaptive real-time video transmission over DDS. *2010 8th IEEE International Conference on Industrial Informatics*. (Osaka, 2010). pp. 130-135. DOI=10.1109/INDIN.2010.5549450