Open Sound Control: Constraints and Limitations

Angelo Fraietta
Smart Controller Pty Ltd
PO Box 859
Hamilton 2303, Australia
61-2-90431806
info@smartcontroller.com.au

ABSTRACT

Open Sound Control (OSC) is being used successfully as a messaging protocol among many computers, gestural controllers and multimedia systems. Although OSC has addressed some of the shortcomings of MIDI, OSC cannot deliver on its promises as a real-time communication protocol for constrained embedded systems. This paper will examine some of the advantages but also dispel some of the myths concerning OSC. The paper will also describe how some of the best features of OSC can be used to develop a lightweight protocol that is microcontroller friendly.

Keywords

MIDI, Open Sound Control, Data Transmission Protocols, Gestural Controllers.

1. INTRODUCTION

Open Sound Control (OSC) has been implemented as a communications protocol in more than a few hardware and software projects. The general impression appears to be that "MIDI is a simple and cheap way to communicate between a controller and computer, but it is limited in terms of bandwidth and precision and on the way out, OpenSound Control [*sic*] being a better alternative"[1]. In some cases, developers felt that they had to implement OSC in new instruments to maintain any sort of credibility in the NIME community [4]. It appears that the general consensus in computer music communities is that OSC is computer music's new 'royal robe' to replace the outdated, slow, 'tattered and torn' MIDI and its "well-documented flaws" [18]. This perception could be implied due to the lack of papers critical of OSC.

OSC has provided some very useful and powerful features that were not previously available in MIDI, including an intuitive addressing scheme, the ability to schedule future events, and variable data types. Although more and more composers are developing and composing for low power, small footprint, and wireless instruments and human interfaces [3, 13, 14]; a move toward OSC in these application is not always possible, nor desirable. Although OSC has addressed some of the limitations of MIDI, OSC does not provide "everything needed for real-time control of sound" [17] and is unsuitable as an end-to-end protocol for most constrained embedded systems.

This paper will first describe some of the powerful features provided by OSC before dispelling some of the myths regarding OSC. Finally, some strategies will be proposed that could be used to develop a protocol to meet the needs of constrained systems.

NIME08, June 5-7, 2008, Genova, Italy

2. OSC FEATURES

2.1 OSC Addressing Scheme

The OSC address scheme provides three main features: the ability to give the mapped address an intuitive name, the ability to increase the maximum number of namespaces, and the ability to define a range of addresses within a single message.

2.1.1 Intuitive Names

OSC is similar to MIDI in that it defines mapped points and values to be assigned to those points. For example, if a gestural controller had the left finger position mapped to 'MIDI controller 12 on Channel 1', a value of '127' would be accomplished by sending the bytes '0xB0[1] 0x0C 0x7F'. The point being mapped is defined by the first two bytes, while the value of the point is defined by the last byte. In OSC, setting a point with a value could be done with the following message: '/minicv/forefinger 127'; the address being '/minicv/forefinger'. The ability to provide an intuitive name to a parameter is a function of composition rather than a function of performance. It is much easier for a composer to map a musical event to a meaningful name, such as '/lefthand' than it is to map to some esoteric set of numbers such as '0xB0 0x0C'.

2.1.2 Increased Namespace

The addressing feature of OSC enables the users to increase the possible number of mapped points. In MIDI, for example, after continuous controllers 0 to 127 on channels 1 to 16 have all been assigned, the namespace for continuous controllers has been exhausted. In OSC, however, if two performers required the namespace 'lefthand', the address space could be expanded "through a hierarchical namespace similar to URL notation" [18]. For example, the two different performers could use '/performer1/lefthand' and '/performer2/lefthand'. Each OSC client will receive these messages, and due to the packet paradigm of OSC [18], the client that does not require the message will discard it.

[1] 0x in front of the number signifies it is a hexadecimal value.

2.1.3 Address Ranges

The namespace feature of OSC is extremely powerful in that it enables a significantly large number of namespaces and the ability to define a range of points in a single message. For example, the OSC namespace '/minicv/left* 127' would set the value of '127' from '/minicv/leftThumb' right through to '/minicv/leftPinkie.'

2.2 OSC Data Types

One of the brilliant features of OSC is the ability to define different data types that can be transmitted in a message. Although it is possible to send any data type of any resolution using MIDI system exclusive messages, OSC has provided a standard for software and hardware developers from different vendors.

2.3 Time Tags

OSC contains a feature where future events can be sent in advance, allowing "receiving synthesizers to eliminate jitter introduced during packet transport" [18] providing "sub-nanosecond accuracy over a range of over 100 years" [19].

3. MYTHS

When one considers that OSC has been used in some very impressive installations and performances, such as "multichannel audio and video streams and robot control sequences at Disneyland" [OSC Newsgroup], it is not too difficult to understand why one may be reluctant to write a critical paper when OSC is gaining a 'legendary' reputation. If one is to consider using OSC on a constrained system, one should separate fact from the fable, using maths to dispel the myths. The two myths this paper will dispel are that OSC is fast and that OSC is efficient.

3.1 OSC is Fast

There is a belief in the NIME community that OSC is a fast communications protocol; for example, "The choice for OSC … was for its high speed, powerful protocol, and driver/OS-independency" [5]. Statements such as these are normally based on a comparison between the data transmission rate between OSC and MIDI in their typical applications [OSC Newsgroup].[2] It is, however, misleading to compare the speed of OSC to MIDI based on the data transmission rate because OSC does not have a data transmission rate.

The Open Systems Interconnect (OSI) model [7] defines a communication model where applications communicate through a layered stack, where the transmitted message passes from the highest layer of the stack to its lowest layer on the transmitting end, and from the lowest layer to the highest layer on the receiver. OSC does not define anything below the presentation layer, but rather assumes the transport layer will have a bandwidth of greater than 10 megabits per second [19]. MIDI, however, can be defined using the OSI model [7], from its Application Layer defining the message type right down to the Physical Layer that defines the connector type and current loop [4]. The speed comparison between OSC and MIDI is always made at the 31.25 kilobits per second Data Layer in MIDI[3], and so Wright and Freed state that MIDI is "roughly 300 times slower" [18] than OSC. Speed, by its definition, is a function of time; in the same way the weight is not just a function of mass, but also a function of gravity. Comparing the speed of MIDI with that of OSC is akin to comparing the weight of a 2Kg ball on earth with a 600Kg ball in outer space where the gravity is zero. A more accurate speed comparison between OSC and MIDI would be made by comparing the two protocols at identical layers on the OSI stack, comparing the time taken for the target data to be encoded and then decoded on identical layers on the stack using identical processors. If one was to measure the number of machine instructions required to parse a typical MIDI message with that of a typical OSC message, MIDI would win hands down.

3.2 OSC is Efficient

"Open SoundControl [*sic*] is … efficient… and readily implementable on constrained, embedded systems." [18]. Efficiency is generally the ability to accomplish a particular task with the minimum amount of wastage or expenditure. In the context of a gestural controller, it would be the ability to provide the same or similar functionality with the minimum amount processor speed, memory, power, and bandwidth while providing the same or similar functionality. Efficiency is a relative term—what is deemed efficient today may be deemed inefficient tomorrow when newer technologies or algorithms are developed. In order to evaluate whether OSC is efficient, one does not necessarily need to compare it in its entirety to a pre-existing system, but rather, to demonstrate how the resources are being wasted.

In a real-time system, such as a music performance, the ability to meet timing constraints is of primary importance [15]. The system "must respond to external events in a timely fashion, which means that for all practical purposes, a late computation is just as bad as an outright wrong computation" [8]. Many newer mobile musical interfaces are communicating wirelessly; for example, the "Pocket Gamelan", which uses mobile telephones that communicate amongst themselves [14]. Although the speed of processors in wireless devices is increasing, this "increase in the processor speed is accompanied by increased power consumption" [13]. An increase in power means a decrease in the period that a battery powered controller can be used in a performance. Furthermore, power usage contributes to the carbon footprint of the instrument. Efficiency, therefore, is also an environmental issue.

The developers of OSC state "our design is not preoccupied with squeezing musical information into the minimum number of bytes. We encode numeric data in 32-bit or 64-bit quantities, provide symbolic addressing, time-tag messages, and in general are much more liberal about using bandwidth for important features"[18]. A major aspect of this "liberal use of bandwidth"

[2] Personal communications on Developer's list for the OpenSound Control [*sic*] (OSC) Protocol osc_dev@create.ucsb.edu will be referred to as OSC Newsgroup.

[3] The MIDI Manufacturers Association (MMA) has approved a standard for MIDI over IEEE-1394 (FireWire) [8] and is already being used by instrument manufacturers including Yamaha, Presonus, Roland, and M-Audio [Personal communications]. Furthermore, other implementations of MIDI over other protocols exist, so the speed limitation of MIDI is no longer technically correct.

is the address, which defines the mapped point being referenced. As stated previously in this paper, the advantages given by the addressing scheme are the intuitive names, the increased namespace, and addressing a range of points and will be addressed later in the paper.

3.2.1 Communications Bandwidth

An example was given previously for a mapped point—one using MIDI, '0xB0 0x0C 0x7F'; the other using OSC '/minicv/forefinger 127'. The first example uses only three bytes while the second uses over twenty bytes. A significant problem with the increased message sizes for wireless systems is that "the more data that is transmitted the greater the chance that part of the message will need to be retransmitted due to noise - increasing latency and jitter" [OSC Newsgroup[4]]. Some developers of embedded and wireless instruments that have been using OSC have resorted to developing pseudo device drivers, whereby OSC is converted to a lightweight protocol before being transmitted [3], reporting a five hundred percent increase in throughput and efficiency [OSC Newsgroup[5]]. Although this is an efficient alternative to transmitting the whole OSC packet over the serial port, it effectively means that OSC is not the complete end-to-end, server to client protocol.

3.2.2 Processing Bandwidth

Although the transmission rate is taken into consideration—hence Wright and Freed's assumption "that Open SoundControl [*sic*] will be transmitted on a system with a bandwidth in the 10+ megabit/sec range" [18]—many seem to forget that after transmission and reception, the packet also needs to be parsed by the target synthesiser. Furthermore, it is not just the target synthesiser that needs to parse the data, but all synthesisers that are not the intended recipients are required to stop what they are doing and parse a significant number of bytes before rejecting the message. This in turn affects the minimum processing requirements of each and every component in entire system. Although many microcontrollers are being developed with higher processing speeds, the "increase in the processor speed is ... accompanied by increased power consumption" [13].

3.2.3 Processing Efficiency

Although the string based OSC namespace is more efficient for a human to evaluate, a numerical value is much more efficient for the computer because computers are arithmetic devices. Apart from the number of bytes that need to be parsed, the OSC implementation requires that the namespace be parsed through some sort of string library, requiring additional computation and the memory space to contain the library. In a performance where a mapped point is changed one hundred times a second, the human would not be expected to read that value for every message sent; the computer, however, is. Hence, the message is optimized for the entity that requires it least during performance. This problem with the OSC addressing model is that the coupling between the human cognition of the namespace and transmission mechanism to the target computer is too tight [6]—the naming, which is effectively the human interface, should be abstracted away from the implementation using a mapping strategy. Two such strategies that uses this type of mapping are the Internet Name Server [10] for addressing domain names, and the Address Resolution Protocol (ARP) [9] used on local Ethernet networks. Without going into the exact details, a brief explanation of how each mechanism operates is presented, showing how similar paradigms to the OSC address space are efficiently implemented.

3.2.3.1 Internet Mapping

The intuitive naming strategy used in OSC is similar to domain names on the internet. When addressing a computer on the internet, one does not normally type in the Internet Protocol (IP)[11] address; rather, they type in the domain name. This makes it very easy for a human to remember how to locate and communicate with a particular computer on the internet. The calling computer, however, does not send a request to every computer on the internet. Instead, the domain name is mapped to an IP address through an Internet Name Server[10]. For example, if one was to 'ping' a domain from the command line, the computer will obtain the IP address from the name server and then send ping messages to the IP address. For example:

```
$> ping smartcontroller.com.au
Pinging smartcontroller.com.au [210.79.16.38]
with 32 bytes of data:
Reply from 210.79.16.38: bytes=32 time=16ms
TTL=56
```

This activity is done behind the scenes and is abstracted away from the user. Although obtaining the IP before sending a message is effectively a two step procedure, these two steps make it much more efficient than sending the domain name to every web server.

3.2.3.2 Local Network Ethernet Mapping

On local networks, the abstraction is done through the Media Access Control (MAC) address through ARP [9]. If, for example, a computer whose IP address was '192.168.0.2' on a local network wanted to send a message to the computer addressed '192.168.0.4', it does not send a message to all the computers on the local network expecting all but '192.168.0.4' to reject it. If this was the case, every time a network card received a message, it would be required to interrupt the computer, impacting on the performance of the rejecting computer. Rather, the ARP layer maps the IP addresses of the computers on the network to MAC addresses. This MAC address is used to address the network card. The other network cards on the local network ignore the message and do not interrupt the computer. This mapping can be viewed on a computer by typing 'arp –a' from the command prompt.

```
$> arp -a
Interface: 192.168.0.2 --- 0x2
  Internet Address Physical Address   Type
  192.168.0.1      00-04-ed-0d-f2-da  dynamic
  192.168.0.4      00-13-ce-f4-63-b6  dynamic
```

Although these steps are complicated, this is the sort of thing computers are good at and it makes communication on complex networks very efficient. A similar approach to these could be used as an underlying layer to OSC. Implementation of such a mechanism for OSC is well beyond the scope of this paper; this does show that such processes are being used by other technologies for improved efficiency and should probably be used in OSC.

3.2.4 Address Pattern Matching

The method of mapping multiple points to a single message, for example, the OSC namespace '/minicv/left* 127' is based

[4] Christopher Graham posting on 23 January 2008.

[5] ibid.

on the UNIX name matching and expansion [18, 19]. Once again we see a tight coupling between the human interface and the computer implementation. Although the developers of OSC claim that "with modern transport technologies and careful programming, this addressing scheme incurs no significant performance penalty ... in message processing"[18], using two numbers to define a range would require significantly less processing than decoding a character string with wildcards. For example, in the address range '/minicv/left*', every character would need to be parsed and tested to see if it was one of the defined wildcard characters. Next, one would have to factor in the string comparison that would be required for every mapped address on the client computer.

Protocols such as MODBUS [http://www.modbus-ida.org/] and DNP [http://www.dnp.org/] are used by telemetry units to control pump stations in real time [2]. These protocols can use a message type that sets a range of mapped points using a single message. When a range is defined using two numbers, it is a simple matter to test if a mapped point is within the range. For example, if a message from a protocol that defined two mapped point ranges 'UPPER_RANGE' and 'LOWER_RANGE', the algorithm to test would be as follows.

```
IF MAPPED_POINT <= UPPER_RANGE
AND MAPPED_POINT >= LOWER_RANGE THEN
    ProcessValue
ENDIF
```

As with the intuitive names, this requires an additional layer of mapping and abstraction, which in turn means work for the developer. Software engineering has a similar paradigm where some languages are scripted and some are compiled. Scripted languages require the server to compile human readable code each time it is executed, while compiled languages use a tool to convert human readable code to something that is more efficient for the computer. The first type is more efficient for the programmer because he or she does not need to compile the code after each modification; however, there is a definite performance hit. Compiled languages require an extra step: compiling the human readable code to machine code; however, there is an enhancement in performance. In terms of communications protocols, OSC is like a scripted language: extremely powerful, but requiring significantly more computing power than what is available to most embedded technologies today.

3.2.5 Message Padding

Another possible inefficiency is the padding of all message parameters to four byte boundaries. For example, a parameter that is only one byte in length is padded to four bytes. The reasoning behind this is that the OSC data structure is optimised for thirty-two bit architectures [OSC Newsgroup]. There have not yet been any conclusive tests to determine whether the gains obtained from this optimisation exceed the additional overhead created by inserting and later filtering these additional padded bytes [OSC Newsgroup]; however, these results should be forthcoming in the near future. It does, however, mean that there would be a decrease in efficiency for eight, sixteen, and sixty-four bit architectures.

4. FAULT TOLERANCE

OSC is a packet driven protocol that does not accommodate failure in the underlying OSI layers. UDP [12] is a protocol that is used by many implementations of OSC [19]. UDP not guarantee that a packet will be received if transmitted; moreover, it does not guarantee that the target will receive packets in the order they were sent. OSC is based on the same paradigm as UDP in that it is packet driven. "This leads to a protocol that is as stateless as possible: rather than assuming that the receiver holds some state from previous communications" [18]. The problem with the paradigm is that it is no longer event driven, and assumes all the relevant data is transmitted at once. If a gestural controller sends an OSC message that was supposed to change a robot motor direction, immediately followed by a message to start the motor, the OSC receiver may receive those in the opposite order, which may be worse than not receiving the information at all. For example, if a server was to send the following messages using UDP:

```
/lefthand/motor/direction 1
/lefthand/motor/start
```

The client could receive them as follows:

```
/lefthand/motor/start
/lefthand/motor/direction 1
```

This now means that the composer will need address the possibility of messages arriving in the wrong order without any notice from the protocol. Although, one could use TCP "in situations where guaranteed delivery is more important than low latency" [19], lower latency has been one of the OSC evangelists' greatest catch cries.

5. STRATEGIES FOR IMPROVEMENT

The first strategy for improvement is the intelligent mapping of namespaces to numbers. OSC must move away from the stateless protocol paradigm and begin to embrace techniques such as caching [16], which has been used for many years now to improve the performance of networks, hard drives, and memory access on CPUs. MIDI's use of running status is an example of how caching can improve performance by nearly thirty-three percent. Caching will be the key to efficient mapping of address patterns to simple numbers without significant impacting upon performance.

OSC must move toward an event delegation model, where clients register whether to receive OSC messages within a particular namespace. Needlessly receiving and parsing large irrelevant messages from OSC servers is a waste of valuable processing power.

The developers of OSC must change their attitude towards MIDI. OSC has been anti-MIDI for a while, with OSC developers often ridiculing MIDI developers [personal correspondence]. Some OSC developers have made token gestures towards MIDI by providing a namespace, which is "an OSC representation for all of the important MIDI messages" [19]. This completely defeats the innovative address pattern provided by OSC. Instead, an underlying network layer should convert an intuitively mapped name, such as '/performer1/lefthand' to a MIDI message and then transport it via MIDI or vice versa. The MIDI controller number should be completely abstracted away from the application layer in order to reduce the coupling between the two. The OSC server should not need to know at the application layer that the motor that controls the robot's left finger is MIDI controller 13. Likewise, the motor that is being controlled by controller 13 should not need to know that the OSC server is really addressing '/performer1/lefthand'. Although these sort of strategies have

been employed in dynamic routing schemes in some OSC projects [19], this should be a function of the network layer, not the application layer. When one considers that the longest domain names on the internet can be addressed with only four bytes, it is not unreasonable to expect that even the most complex OSC namespaces could be translated into simple MIDI messages if required.

There needs to be a greater number of message types—currently there are only two. OSC needs to move towards an object oriented paradigm in the communications protocol [4].

Currently, all the network, data link, and transport layers of transmission have been delegated to the application layer. This is above the presentation layer, which is where OSC exists—this is completely upside down when comparing to the OSI model. OSC needs to develop an underlying OSI stack where the protocol between the client and server is abstracted away from the user. The underlying mapping should direct the message from the source to the destination.

6. CONCLUSION

Although OSC has provided a standard "protocol for communication among computers, sound synthesizers, and other multimedia devices" [19], and was supposed to overcome "MIDI's well-documented flaws … [, its] liberal [use] … of bandwidth" [18] may be its Achilles heel, preventing it from ever being the standard end-to-end protocol for communication for low power and wireless microcontroller interfaces. If OSC is to have any hope in servicing this significant and important area of the NIME community, an OSI stack needs to be developed that has efficiency and performance at the forefront, while at the same time, implementing proven design patterns [6]. This, however, would be a significant research project within itself.

7. ACKNOWLEDGMENTS

I would like to thank Adrian Freed from Center for New Music and Audio at Univ. California, Berkeley for answering the many questions I asked about OSC. I would also like to thank all the members of the Developer's list for the OpenSound Control [*sic*] (OSC) Protocol osc_dev@create.ucsb.edu for their input.

8. REFERENCES

[1] Doornbusch, P., Instruments from now into the future: the disembodied voice. Sounds Australian, 2003(62): p. 18.

[2] Entus, M., Running lift stations via telemetry. Water Engineering & Management, 1989. 136(11): p. 41-43.

[3] Fraietta, A. Mini CV Controller - Conference Poster. in Generate and Test: the Australasian Computer Music Conference. 2005. Queensland University of Technology, Brisbane: Australasian Computer Music Association.

[4] Fraietta, A., The Smart Controller: an integrated electronic instrument for real-time performance using programmable logic control, in School of Contemporary Arts. 2006, University of Western Sydney.

[5] Kartadinata, S. the gluion: advantages of an FPGA-based sensor interface. in International Conference on New Interfaces for Musical Expression (NIME). 2006. IRCAM - Centre Pompidou, Paris, France.

[6] Larman, C., Applying UML and patterns: an introduction to object-oriented analysis and design and the unified process. 2nd ed. 2002, Upper Saddle River, NJ: Prentice Hall PTR. xxi, 627.

[7] Lemieux, J., The OSEK/VDX Standard: Operating System and Communication. Embedded Systems Programming, 2000. 13(3): p. 90-108.

[8] Pawlicki, J. Formalization of embedded system development: history and present. in Quality Congress. ASQ's ... Annual Quality Congress Proceedings. 2003: PROQUEST Online.

[9] Plummer, D.C. RFC 826 - Ethernet Address Resolution Protocol: Or converting network protocol addresses to 48.bit Ethernet address for transmission on Ethernet hardware. < http://www.faqs.org/rfcs/rfc826.html > accessed 28 January 2008

[10] Postel, J. IEN-89 - Internet Name Server. < ftp://ftp.rfc-editor.org/in-notes/ien/ien89.txt > accessed 28 January 2008

[11] Postel, J. RFC 760 - DoD standard Internet Protocol. < http://www.faqs.org/rfcs/rfc760.html > accessed 28 January 2008

[12] Postel, J. RFC 768 - User Datagram Protocol. < http://www.faqs.org/rfcs/rfc768.html > accessed 21 January 2008

[13] Schiemer, G. and M. Havryliv. Wearable firmware: the Singing Jacket. in Ghost in the Machine: the Australasian Computer Music Conference. 2004. University of Victoria, Wellington.

[14] Schiemer, G. and M. Havryliv. Pocket Gamelan: a Pure Data interface for java phones. in International Conference on New Musical Interfaces for Music Expression (NIME-2005). 2005. University of British Columbia, Vancouver.

[15] Son, S.H., Advances in real-time systems. 1995, Englewood Cliffs, N.J.: Prentice Hall. xix, 537.

[16] Vitter, J.S., External memory algorithms and data structures: dealing with massive data. ACM Comput. Surv., 2001. 33(2): p. 209-271.

[17] Wright, M. Introduction to OSC. < http://opensoundcontrol.org/introduction-osc > accessed 21 January 2008

[18] Wright, M. and A. Freed. Open SoundControl: A New Protocol for Communicating with Sound Synthesizers. in International Computer Music Conference. 1997. Thessaloniki, Hellas: International Computer Music Association.

[19] Wright, M. and A. Freed. OpenSound Control: State of the Art 2003. in International Conference on New Interfaces for Musical Expression (NIME-03). 2003. Montreal, Quebec, Canada.