*SMPTE Meeting Presentation*

# Scalability and Performance of the AMWA IS-04 and IS-05 NMOS Specifications for Networked Media

**Robert Porter**

Sony Europe Limited

**Gareth Sylvester-Bradley**

Sony Europe Limited

**Written for presentation at the**

**SMPTE 2018 Annual Technical Conference & Exhibition**

**Abstract.** *The use of IP networks for professional AV media is becoming prevalent within the broadcast industry with standards such as SMPTE ST 2110 now available for streaming uncompressed video, audio and ancillary data. To allow full interoperability between different manufacturers' equipment, common methods for discovery, registration and connection management of media devices are essential. The Advanced Media Workflow Association (AMWA) has been developing the Networked Media Open Specifications (NMOS) for this purpose, leading to the publication of two interface specifications: IS-04 for discovery/registration of NMOS Nodes and IS-05 for management of connections between Nodes. These use RESTful APIs to communicate between applications and devices and are gaining widespread adoption.*

*However, a key requirement within the industry is that these APIs scale successfully in the very large installations typical of real world deployments. To help address this, Sony has been leading an AMWA NMOS Scalability study to test these protocols for installations comprising thousands of media devices.*

*The study includes: confirming that operations such as the registration of thousands of Nodes can occur within an acceptable timeframe; testing recovery of the Registry after a failure; testing behaviour with multiple clustered Registries; testing different methods of Registry discovery; testing connection management at scale; and confirming behaviour for architectures with redundant network interfaces. The test suite has been shared with other AMWA members and tests have been repeated and confirmed with different Registry and Node implementations.*

*This paper describes our test methodology, provides several results, discusses their implications, identifies best practices for deployment and suggests whether further API enhancements may be beneficial.*

**Keywords.** Networked Media, IP, ST 2110, AMWA, NMOS, IS-04, IS-05, API, Device Discovery, Device Registration, Connection Management, Scalability, Registry Discovery, DNS-SD, mDNS, Unicast DNS, Registry Clustering, Redundancy, Mininet, Virtualized Network.

## Introduction

The transition from SDI to IP-based architectures for routing professional AV media within broadcast facilities is now well underway. The SMPTE ST 2110 specification has gained widespread acceptance within the broadcast industry as the de facto standard for streaming video, audio and metadata essence between media devices [1][2]. A key benefit of this is that equipment from different manufacturers can successfully interoperate within a deployment using a common standard rather than requiring conversion between a multiplicity of proprietary protocols.

While SMPTE ST 2110 defines the protocols for the transport of the media essence across the network, it does not describe either how devices are discovered when they are first attached to a network or how connections between sending and receiving devices should be managed. To address these requirements, the Advanced Media Workflow Association (AMWA) have developed a set of specifications known as the Networked Media Open Specifications (NMOS) [3]. This paper will focus on two of these which are already in active use by many manufacturers: the interface specifications IS-04 and IS-05.

**AMWA IS-04** defines the methods for the **discovery and registration** of devices on a professional media network.

**AMWA IS-05** defines the methods for **connection management** between devices on the network.

These interface specifications take the form of RESTful APIs, defined using the RESTful API Modelling Language (RAML) and a set of JSON schemas, and are freely available on the AMWA GitHub internet pages [4]. IS-04 comprises three APIs: the **Node API**, the **Registration API** and the **Query API**; IS-05 comprises just one API: the **Connection API**.

The sequence of operations on connecting a media endpoint to the network can be briefly summarised as follows. The media endpoint must first discover an IS-04 Registration and Discovery System (RDS) using Domain Name System – Service Discovery (DNS-SD). It then registers itself using the IS-04 Registration API. An NMOS Client (e.g. a Broadcast Controller) is able to query the RDS for a list of registered resources using the IS-04 Query API. Connections between sending and receiving devices may then be made by the NMOS Client using the IS-05 Connection API to communicate with the devices.

It is important that all these operations are scalable to work successfully with the many thousands of devices that are typical of very large real-world deployments. For this reason, the AMWA NMOS Scalability study was set up to test the performance of the APIs at scale and feed back to the industry, as well as making recommendations for best practice.

Several tests have been performed during the study, including:

- Confirming that operations such as the initial registration of thousands of Nodes can occur within an acceptable timeframe.
- Testing recovery of the Registry after a failure.
- Testing performance with multiple clustered Registries.
- Testing different methods of Registry discovery (mDNS, unicast DNS).

Several other areas of investigation are also being actively studied, including the possibility of a bulk API for registration, connection management at scale, and confirming behaviour for architectures with redundant network interfaces.

This paper is organised as follows. A brief description of the AMWA IS-04 and IS-05 APIs is given in the next section. The testbed used to measure the performance of the APIs, comprising a virtualized network environment based on Mininet, is then outlined. Each area of study, along with results where appropriate and some discussion, is then described in the ensuing sections. The paper concludes with some further discussion and recommendations for future work.

## The AMWA Networked Media Open Specifications (NMOS)

### *Content Model*

The NMOS specifications use a content model that is based on the Joint Task Force on Networked Media (JT-NM) Reference Architecture [5]. In this model, a logical host connected to the network is known as an NMOS **Node**. A Node may host one or more NMOS **Devices**, which in turn may be associated with any number of NMOS **Senders**, **Receivers**, **Sources** and **Flows**. For example, in this model a camera may be represented by a single Node with a single Device containing two Senders (video and audio) and two Receivers (two video return channels), as illustrated in Figure 1. Each Sender may have a Source and a Flow associated with it. For a full definition of each term, refer to the glossary of terms on the AMWA NMOS GitHub pages [6].

Nodes, Devices, Senders, Receivers, Source and Flows are all known as NMOS **resources**.

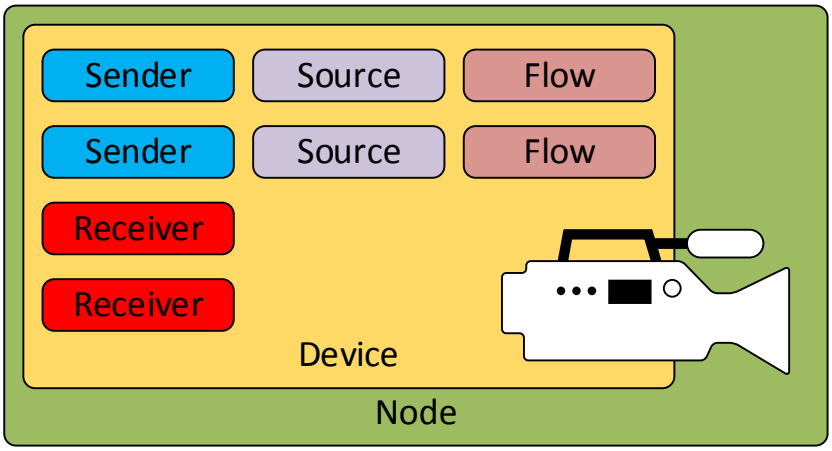



Figure 1. Example of an NMOS Node.

### *Discovery and Registration – AMWA IS-04*

AMWA IS-04 [7] specifies the methods for the discovery and registration of NMOS Nodes. Nodes may operate in two different modes: **peer-to-peer mode** and **registered mode**.

## Peer-to-Peer Mode

In a system without an RDS, Nodes operate in peer-to-peer mode. This allows two Nodes to communicate directly over a network connection. Each Node must advertise itself on the network via multicast DNS-SD to allow it to be discoverable by other Nodes. Additionally, each Node must expose its NMOS resources using the IS-04 Node API. This allows the Node itself and all its sub-resources to be accessible to other Nodes through HTTP GET requests. An example of the use of this request to get information about the Node, and the Node's response, is shown in Figure 2 below.

**Request:**
GET /x-nmos/node/v1.2/self HTTP/1.1

**Response:**
HTTP/1.1 200 OK
Content-Type: application/json

```
{
   "version": "1441700172:318426300",
   "hostname": "host1",
   "caps": {},
   "href": "http://172.29.80.65:12345/",
   "api": {
      "versions": ["v1.1", "v1.2"],
      "endpoints": [
         {
            "host": "172.29.80.65",
            "port": 12345,
            "protocol": "http"
```

Figure 2. Using the Node API to GET a Node's Capabilities.

Peer-to-peer mode is useful for allowing direct communication between two media devices. However, to allow efficient orchestration of multiple professional media devices on the network, a more centralised approach is preferable.

## Registered Mode

The use of an RDS allows the information about all the NMOS Nodes on the network to be cached centrally. The RDS may comprise a single Registry instance or several Registries depending on the network topology and the system's redundancy requirements. When multiple Registry instances are active, the information about the Nodes should be replicated between them, forming a clustered or federated RDS.

Each Registry must expose two APIs: the IS-04 Registration API and the IS-04 Query API.

Nodes populate the RDS using the IS-04 Registration API. On connecting a Node to the network, the following sequence of operations occur:

1. (Optionally) Node starts in peer-to-peer mode and advertises its Node API using mDNS.

2. Node discovers all available Registration APIs which are currently being advertised on the network using DNS-SD (unicast or multicast).
3. Node determines Registration API with the highest priority (given by its “pri” value in the advertisement).
4. Node registers its Node resource with the selected Registration API.
5. Node, now in registered mode:

   (i) registers each of its sub-resources (Devices, Senders, Receivers, Sources, Flows) in turn with the selected Registration API;

   (ii) posts intermittent heartbeats to the selected Registration API.

An example of NMOS Node Registration using the Registration API is shown in Figure 3. As can be seen, the information sent to the Registry is equivalent to the information returned from the Node API when an HTTP GET is performed on the Node.

**Request:**

```
POST /x-nmos/registration/v1.2/resource HTTP/1.1
Content-Type: application/json

{
   "type": "node",
   "data": {
      "version": "1441973902:879053935",
      "hostname": "host1",
      "label": "host1",
      "description": "host1",
      "tags": {},
      "href": "http://172.29.80.65:12345/",
      "api": {
         "versions": ["v1.1", "v1.2"],
         "endpoints": [
            {
               "host": "172.29.80.65",
               "port": 12345,
               "protocol": "http"
```

**Response:**

```
HTTP/1.1 201 Created
/x-nmos/registration/v1.2/resource/nodes/3b8be755-08ff-452b-b217-
c9151eb21193/
```

Figure 3. POSTing an NMOS Resource.

An example of a regular Node heartbeat is shown in Figure 4.

**Request:**
POST /x-nmos/registration/v1.2/health/nodes/3b8be755-08ff-452b-b217-c9151eb21193

**Response:**
HTTP/1.1 200 OK

```
{
   "health": "1441974485"
}
```

Figure 4. Heartbeating an NMOS Node.

An NMOS Client, such as a Broadcast Controller, may wish to get a list of all the registered Nodes and sub-resources, such as Senders and Receivers, to present on its user interface. It does this using a Registry's IS-04 Query API. This exposes the capabilities and sub-resources of each Node in the same way as the Node API, and also supports collection endpoints to return information about multiple resources simultaneously.

In addition, the Query API also supports push notification of changes to the registered resources via WebSockets.

Once an NMOS Client's user interface has been populated with all the Nodes on the network in this way, the user may wish to make a connection between two Nodes.

### *Connection Management – AMWA IS-05*

Connections between Senders and Receivers may be made using the IS-05 Connection API [8]. This is exposed by NMOS Nodes, allowing the connection settings for all Sender and Receiver sub-resources to be modified by the NMOS Client, either individually or as a bulk operation.

By examining the information it has received from the Registry about each Node, the NMOS Client is able to determine which Senders are compatible with which Receivers. The user may then choose to connect two of them together.

To connect a Sender to a Receiver, the following sequence of operations is performed:

1. NMOS Client stages and activates any required changes to transport parameters on the Sender, through an HTTP PATCH operation on the Sender's Connection API.
2. NMOS Client requests the transport file, typically a Session Description Protocol (SDP) file, from the Sender, containing all the information necessary to subscribe to the Sender's multicast transport stream, through an HTTP GET operation on the Sender's Connection API.

3. NMOS Client performs an HTTP PATCH of the transport file on the Receiver's Connection API. Receiver uses information in the transport file to prepare itself to receive data from the Sender.
4. Receiver joins the Sender's multicast group, using for example IGMP Join.
5. Network switch routes packets from Sender to Receiver using standard networking protocols.

### *NMOS API Summary*

In summary, the diagram in Figure 5 shows how NMOS Nodes, an NMOS Registry and an NMOS Client interact using the AMWA IS-04 and IS-05 APIs.

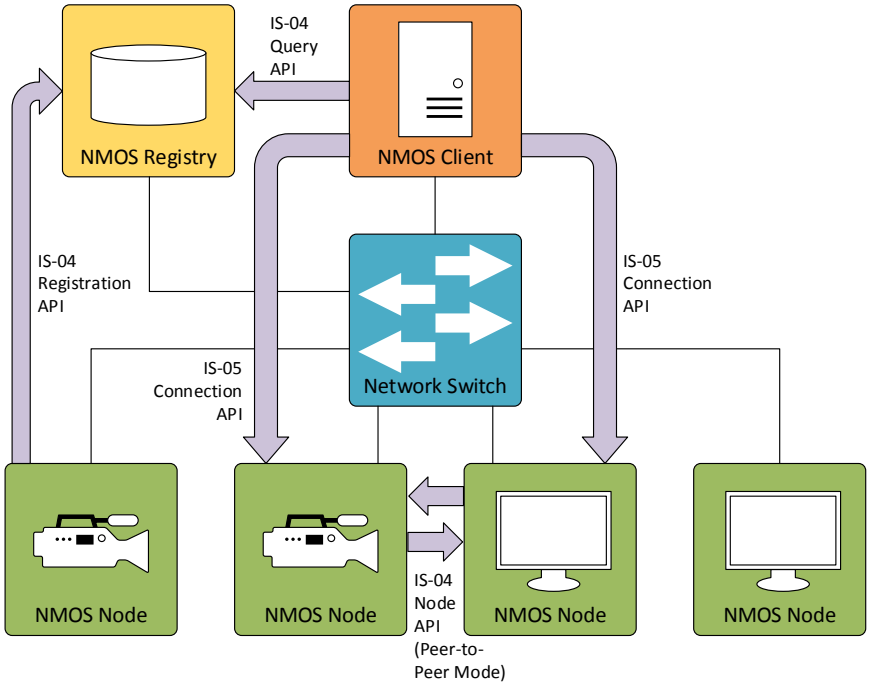



Figure 5. A Professional Media Network showing AMWA NMOS APIs.

## Test Environment

The purpose of the AMWA NMOS Scalability study is to test device management scalability, not data transport scalability; it is the performance of the NMOS APIs that is under test rather than the performance of sending and receiving the media essence data itself. It is unnecessary, and indeed impractical, to use many thousands of real media devices on a large physical network to perform such tests. Instead it was decided that the most practical approach would be to use a virtualized network with software implementations of the NMOS Registry and Nodes that expose all the required APIs.

The open-source virtual network, Mininet [9], was identified as a suitable means to perform the desired network simulation.

### *Mininet Virtual Network*

Mininet allows realistic virtual networks to be created running real kernel, switch and application code on a single machine [9]. The networks are scalable and can take on any topology required by the user.

Figure 6 shows how Mininet has been used within the AMWA NMOS Scalability study. A physical host machine, running Windows, Linux or MacOS, is used to host the Linux-based Mininet Virtual Machine. This can be set up to simulate a network comprising a number of switches, links and, optionally, a network controller. Switches may have a number of endpoints attached to them, known as Mininet hosts. Each host is akin to a lightweight container and is able to run Linux processes. The Mininet network is set up using the Mininet command line interface (CLI). As part of the AMWA NMOS Scalability study, this has been extended by Sony to allow some NMOS-specific commands.

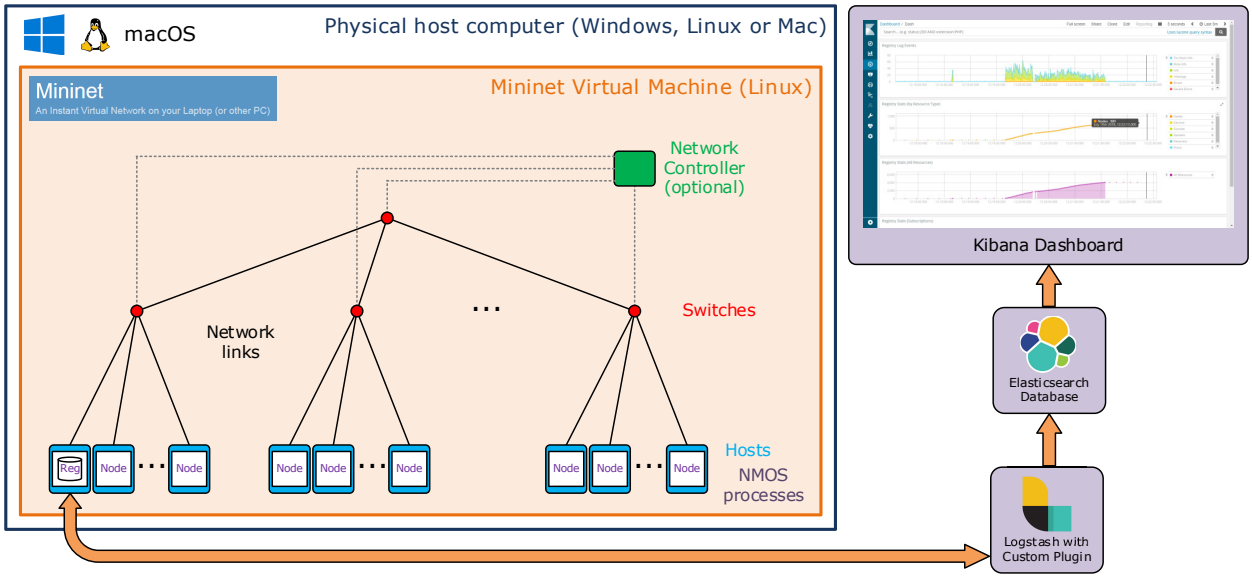



Figure 6. Mininet Test Environment.

### *Mininet Extensions for NMOS*

The NMOS extensions for Mininet include commands to allow operations such as:

- Running an NMOS Registry on a specified Mininet host.
- Running a number of NMOS Nodes on a set of specified Mininet hosts.
- Making timing measurements.

An example of setting up a virtual network and then running some NMOS processes on the network is shown below. Figure 7 shows how the virtual network is initially set up from the Linux

command line, with the network topology and number of switches specified on the command line. In this instance a simple hierarchical (tree) topology is constructed with two levels below the central switch and a branching factor of 40, resulting in 1600 virtual hosts. Mininet provides some common configurable topologies by default, but is fully extensible via a Python API to allow custom topologies e.g. with redundant links and switches.

Once the initial virtual network switches, hosts and links are constructed, the command line interface (CLI) for the extended version of Mininet is started, shown in Figure 8. Here, it can be seen that two NMOS-specific commands have been used to start up an NMOS Registry along with a number of NMOS Nodes.

```
sudo nmos-mn -topo=tree,2,40
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 … h1600
*** Adding switches:
s1 … s41
*** Adding links:
(h1, s1) (h2, s1) …
*** Starting controller
c0
*** Starting 41 switches
s1 … s41
*** Starting CLI:
```

Figure 7. Setting up the Virtual Network using Mininet.

```
mininet> start_registry h1
*** Starting NMOS registry
…

mininet> start_nodes h2 h1600
*** Starting NMOS nodes
h2 h3 h4 … h1600
…

mininet> _
```

Figure 8. Using the Mininet NMOS Extensions to Set Up the NMOS Registry and Nodes.

The test environment can be used to run any Linux-compatible NMOS Registry and Node implementations. Our tests have been run primarily using the Sony open-source implementations known as *nmos-cpp* and available at [10], but other implementations have also been used during the study with similar results.

It was found that various optimizations of the Linux Virtual Machine were necessary to allow the NMOS processes to work at scale, due to some default Linux parameters being unsuitable.

The Mininet environment with extensions and optimizations has been shared with the AMWA community, allowing other members to perform their own scalability tests.

The results produced during the AMWA NMOS Scalability study were generated using an HP Z820 with two eight-core Xeon E5-2670 2.6GHz processors and 64GB RAM. Of this, 12 cores and 40GB RAM was assigned to the Linux Virtual Machine running Mininet.

### *Dashboards*

The Sony open-source *nmos-cpp* Registry and Node implementations produce access logs in the Common Log Format (CLF) [11] to enable automated processing by a wide range of web analysis tools. Error logs are also generated in an easily consumable text file format, and also accessible via a JSON-based REST API closely modelled on the AMWA IS-04 Query API.

In the test system, the open-source Elastic Stack [12] is used to process these logs. Log events are automatically collected using Filebeat or the *nmos-cpp* Logging API, and stored via Logstash in an Elasticsearch database. Dashboards constructed with Kibana allow the test scenarios to be visualized and have been used to prepare the figures in the following sections.

## Resource Registration at Scale

### *Initial Registration with a Single Registry*

A critical metric for networked systems is the time taken after power-on for the system to become fully available. In the case of a large-scale NMOS deployment, the system can be said to be ready once all Nodes have successfully registered all their resources with the RDS.

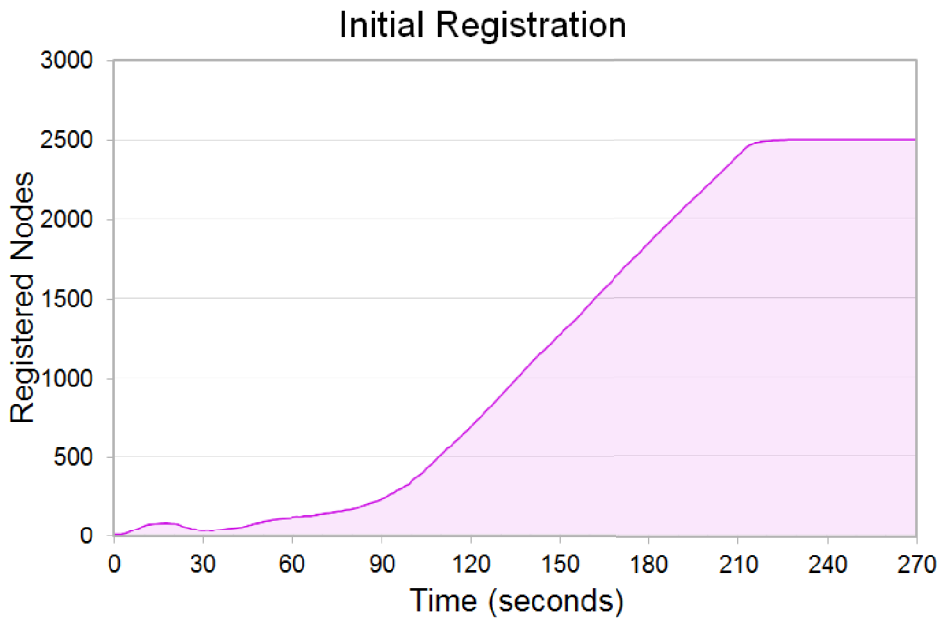



Figure 9. Initial registration of 2,500 Nodes.

Figure 9 shows an initial example of the registration of approximately 15,000 resources of 2,500 Nodes with a single Registry in a Mininet virtual network. The total time taken is 3:42 [minutes:seconds].

Repeating this test, deploying different numbers of Nodes in the network, the relationship between the size of the deployment and the time until all Nodes are registered can be seen to be broadly linear in the Mininet test environment. This is shown in Figure 10.

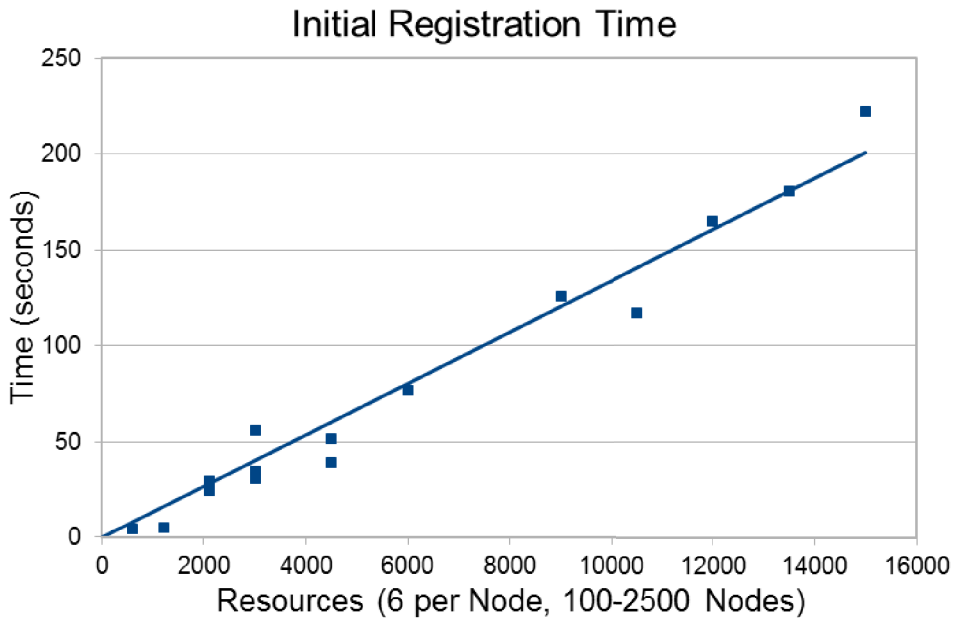



Figure 10. Initial registration time for different numbers of Nodes.

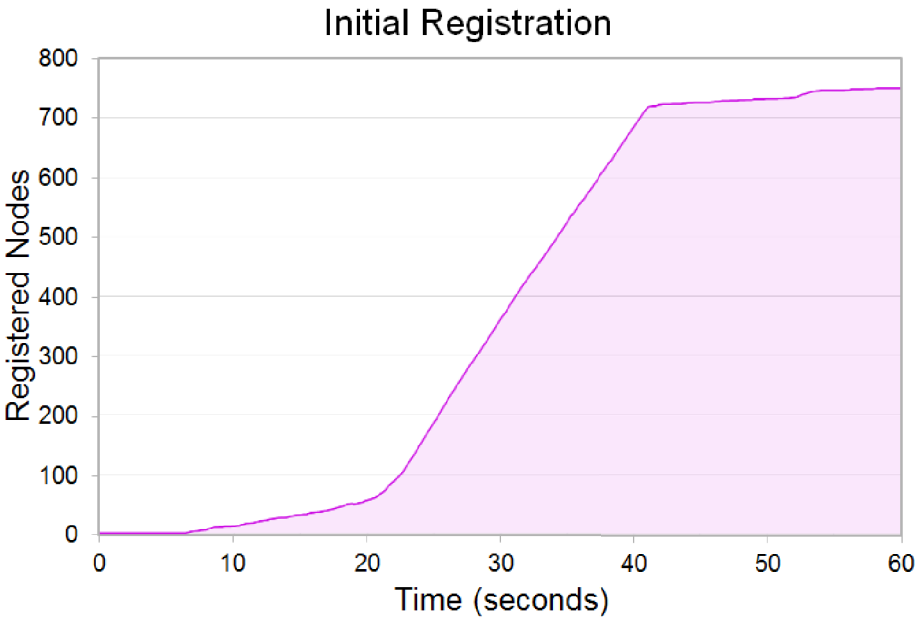
However, examining the detail of Figure 9, it can be seen that the total time does not fully reflect the registration activity on the network; the rate of registrations varies over time. In particular registrations are found to be infrequent initially, and after most registrations have completed, there is a period in which a tail of Nodes complete their registrations. This pattern is repeated in other test runs, for example, Figure 11(a) for 750 Nodes.

Investigation of these results led to the conclusion that several effects were at work.

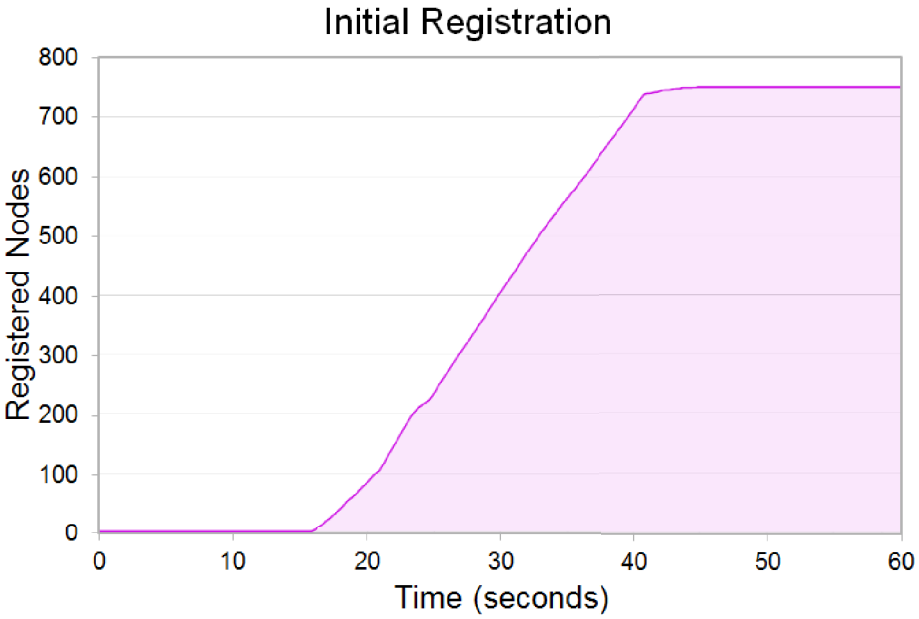
Firstly, as previously mentioned, NMOS Nodes use multicast DNS-SD to discover a Registry. In the Mininet test environment, the NMOS Nodes rely on an mDNS daemon to issue the DNS-SD questions and answers. The daemon implementation (Bonjour) appeared to be overwhelmed by the number of simultaneous requests, sometimes failing to return any results for a period of time. The tests have been repeated with an alternative implementation (Avahi) and found to show similar issues. We concluded that examination of unicast DNS-SD as the discovery mechanism for large-scale deployments should be considered. However, it must also be noticed that the issues encountered with these implementations in the Mininet virtual network may not be completely reproduced in a real network. Mininet uses process-based virtualization, and does not by default fully isolate the processes running for each host, which appears to have had

some impact here. Optimizing the configuration of the DNS-SD daemons improved the results somewhat.




(a) Before: Slow start (13s) and long tail (18s). (Total: 52s)




(b) After: Fast start and short tail (4s). (Total: 29s)

Figure 11. Initial registration improvement for 750 Nodes.

Work to study the utility of multicast and unicast DNS-SD in real networks is ongoing. The efficacy of unicast DNS-SD in the Mininet test environment is covered below.

Secondly, the NMOS Nodes being used in these tests adopted peer-to-peer operation when they failed to discover a Registry (as required by AMWA IS-04), and in this mode were polling infrequently for a Registry. Adjusting this (e.g. using long-running DNS-SD questions, or more frequent polling) improved the “slow start” seen in Figure 9 and Figure 11 (a).

Thirdly, the long tail was found to be related to the handling of network failures, some classes of which are not always apparent to application software. The NMOS Nodes being used in this test initially used a 30 second timeout for HTTP requests before retrying. On adjusting this configuration parameter to a value similar to the NMOS-specified default heartbeat interval of 5 seconds, the long tail is avoided.

The result of these two practices can be seen in Figure 11(b).

The improvements were demonstrable across all test deployment sizes. We conclude that configurability of DNS-SD and HTTP timeouts and retry intervals are an example of best practice for NMOS implementations, with selection of appropriate values based on intended deployments.

### *Recovery After Network Link Failure*

The AMWA IS-04 specification does discuss a pair of parameters that could be tuned based on deployment requirements: the registration expiry interval and the related heartbeat interval, whose default values are 12 seconds and 5 seconds respectively. This allows for 1-2 missed heartbeats before a Node’s registered resources are removed from the Registry cache.

In varying these parameters, there is a tradeoff between the frequency of communication between Nodes and the Registry (and additional load on these and the network) and the effect on the Registration & Discovery System as a whole, e.g.

- the time that stale resource data is kept in the Registry cache after a Node is removed from the network (so-called uncontrolled unregistration)
- the time before a Node recognizes a Registry it has been using is unresponsive and retries with the next discovered Registry instance
- the resilience of the RDS to temporary, short-duration, network failures

The Mininet virtual network provides the means to evaluate these effects. The Mininet CLI provides a simple “link” command to make individual network links go up or down, and access to the Mininet Python API provides the opportunity to script more complex network events.

Figure 12 shows the effect of a failure of a network link close to the single Registry instance in a test deployment of 750 Nodes, using the specified default heartbeat and expiry intervals. Failures of two different durations are shown. For the brief (5 second) failure, the resources of a small number of Nodes are automatically expired from the Registry cache but most Nodes resume heartbeats and avoid this fate. In the case of the longer failure (30s) it can be seen that all Nodes have to re-register.

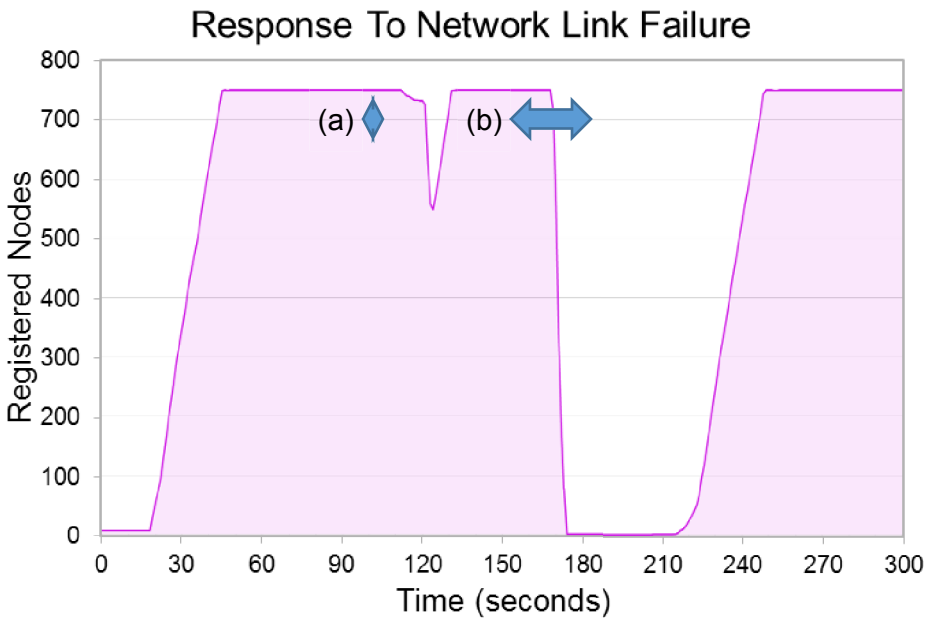



Figure 12. Response to network link failure: (a) 5s, (b) 30s.

As part of the AMWA NMOS Scalability study, it has been found that the default values provide a good balance across a range of deployment sizes.

In the Mininet test environment, it has been seen that decreasing the two values significantly impacts the initial registration metric. This is largely due to the increased number of re-registrations required after Nodes have been expired from the Registration cache during the initial period of high network and Registry load. For example, with a test deployment of 750 Nodes, configuring the Nodes with a 1-second heartbeat interval and the Registry with a 3-second expiry interval increases the time taken for all Nodes to be fully registered from 29 seconds to 42 seconds. Conversely, increasing the heartbeat interval (10s) and expiry interval (25s), does not impact the time taken to complete all Nodes' registrations either way – but has the other tradeoffs previously mentioned.

### *Multiple Registries*

The NMOS Specifications identify the deployment of multiple Registry instances as a means to achieve scalable, resilient, registered operation. As previously described, information about the Nodes should be replicated between the Registry instances, forming a clustered or federated RDS. As *interface* specifications, the mechanism for replication is not specified, in order to allow implementers and system integrators to differentiate their offerings and choose the appropriate technology for their deployments.

NMOS Nodes select from multiple Registries using the priority mechanism, and automatically retry with the next available Registry after a failure. Provided that this fail-over happens within the expiry interval, the resource registrations are maintained in the cache, and NMOS Clients need not be aware of the failure.

Replication is being investigated by several members of the AMWA NMOS Networked Media Incubator. For example, the BBC Reference Implementation [11] can be configured to allow clustering of Registry instances utilizing a distributed key-value store, etcd [14], to implement the cache replication.

The authors of this paper are investigating replication architectures that support eventually-consistent (also known as optimistic or lazy) replication across federated Registry instances, with the possibility of master-master, master-slave and other topologies.

As part of the AMWA NMOS Scalability study, the efficacy of multiple Registry instances and replication has been demonstrated in the Mininet test environment. In contrast to the setup previously described, for these tests, a second Registry instance is started, and federation is configured between both Registries, using the authors' own eventually-consistent replication mechanism.

Figure 13 shows the time taken for all resources of 750 Nodes to be included in the cache at one of the Registries is about 15 seconds. For comparison, Figure 11(b) showed that with a single Registry, the time taken was 29 seconds. The initial "bump" in the number of registered resources visible in Figure 13 indicates the first synchronization from the other Registry instance which started receiving registration requests approximately 2 seconds earlier.

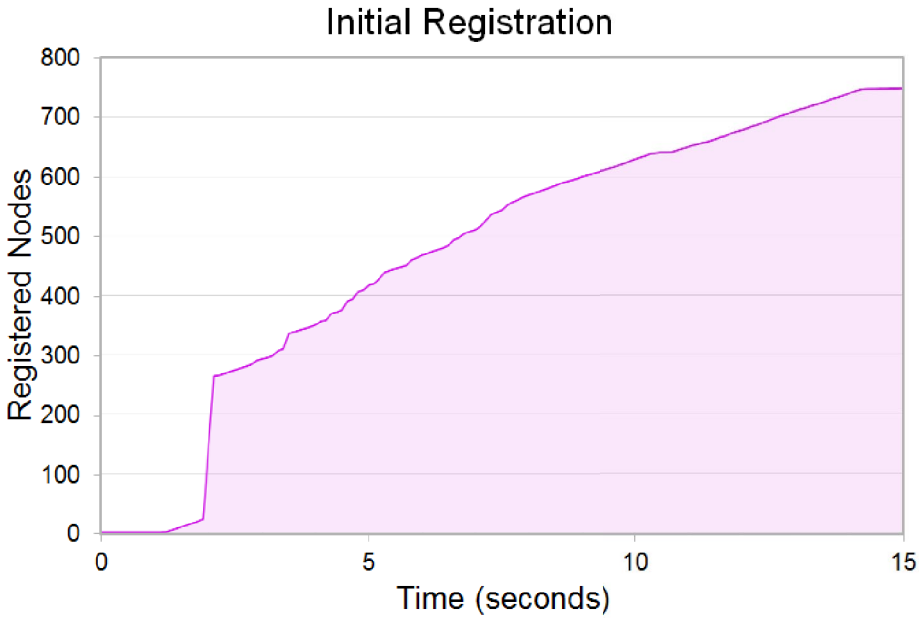



Figure 13. Two-way replication in a federated RDS. 750 Nodes, 15 seconds.

It has also been possible to demonstrate that counter to the result in Figure 12, in the presence of federated registries, some network link failures can be tolerated without any Nodes being expired from the Registry caches. Testing with network topologies that include redundant network links is described below.

As mentioned, there are many possible replication architectures, that cannot all be examined as part of the AMWA NMOS Scalability study. However, the results thus far demonstrate replication is an important facet of scalability, and the tools available to the AMWA community should enable further investigations. As another result of this study, the specification language regarding the Node behaviour after failures with the Registry has been clarified.

### *Registry Discovery – mDNS vs Unicast DNS*

As discussed, multicast DNS-SD seems appropriate for small networks, where there will be at most a few hundred advertisements. Unicast DNS-SD has been suggested as an alternative for large-scale deployments, and is required in layer 3 networks.

During the AMWA NMOS Scalability study, Service Discovery has been found to influence the time taken for initial registration and recovery after a failure.

In the Mininet test environment, it has recently been demonstrated that not using mDNS to advertise and discover the available Registries, results in significant performance improvement in large test deployments (e.g. more than 500 Nodes). We conclude that supporting unicast DNS for Service Discovery should be best practice for NMOS Node implementers.

## Current Studies

### *Bulk API for Registration*

The need for a bulk registration mechanism [15] has been publicly identified by members of the AMWA NMOS Networked Media Incubator, as a possible enhancement to the Registration API. The need for such an enhancement is currently being studied with the Mininet test environment.

Figure 14 demonstrates that the total time taken for Nodes to be fully registered does depend on the number of sub-resources to be registered. This is certainly in part due to the number of separate single-resource registration requests that need to be made.

However, it is to be noted that in an otherwise identical test setup, the time taken to register 15,000 resources from 750 Nodes (1:48) is significantly lower than the time taken to register that many resources from approximately 2,500 Nodes (3:42), as was shown in Figure 10. This seems to be in part due to successful use of persistent HTTP connections, also known as “keep-alive”. Use of this technique should be best practice for NMOS implementers.

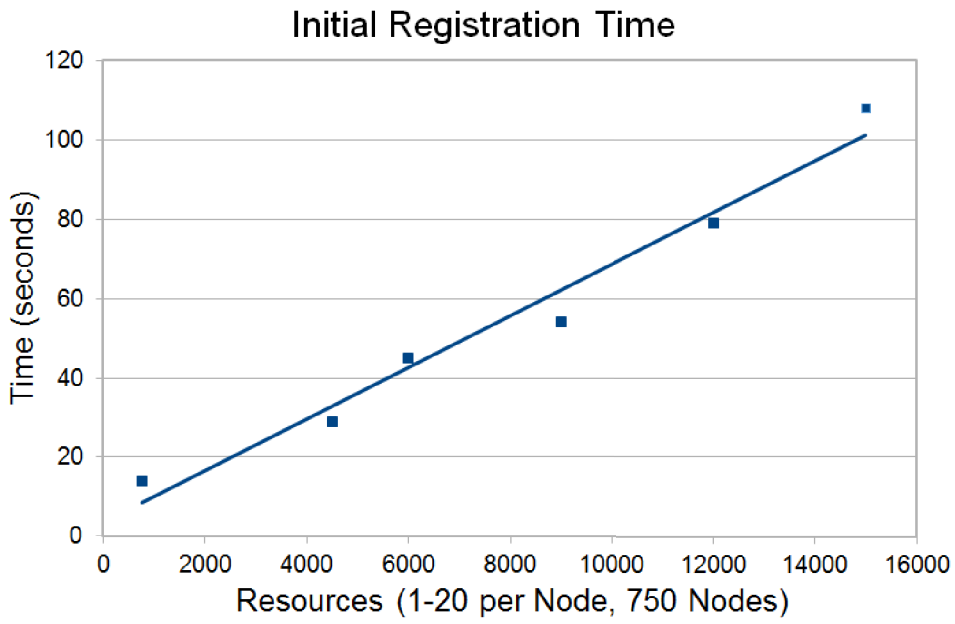



Figure 14. Time taken for 750 Nodes to register. 1-20 Resources per Node.

### *Connection Management at Scale*

As part of the AMWA NMOS Scalability study, the scalability of the AMWA IS-05 Connection API is also being considered. Important metrics include the time taken to make many hundreds of connection changes initiated more-or-less simultaneously. These results are being prepared for presentation.

### *Multiple Network Interfaces*

Another factor for resilience in large-scale deployments, is the use of multiple network interfaces and redundant network devices and links. The NMOS Specifications allow the use of ST 2022-7 [16] for redundant streams of video/audio/ancillary data to be described by the NMOS resources for Senders and Receivers. The NMOS Specifications also describe the behaviour required of NMOS Nodes and Registries to use multiple network interfaces to provide redundancy of control.

The Mininet test environment can be used to create a virtual network with a topology such as the one in Figure 15. This enables testing of resilience to failures of links, interface cards and switches, including for example the Nodes' fail-over between different Registry instances or interfaces of a single Registry.

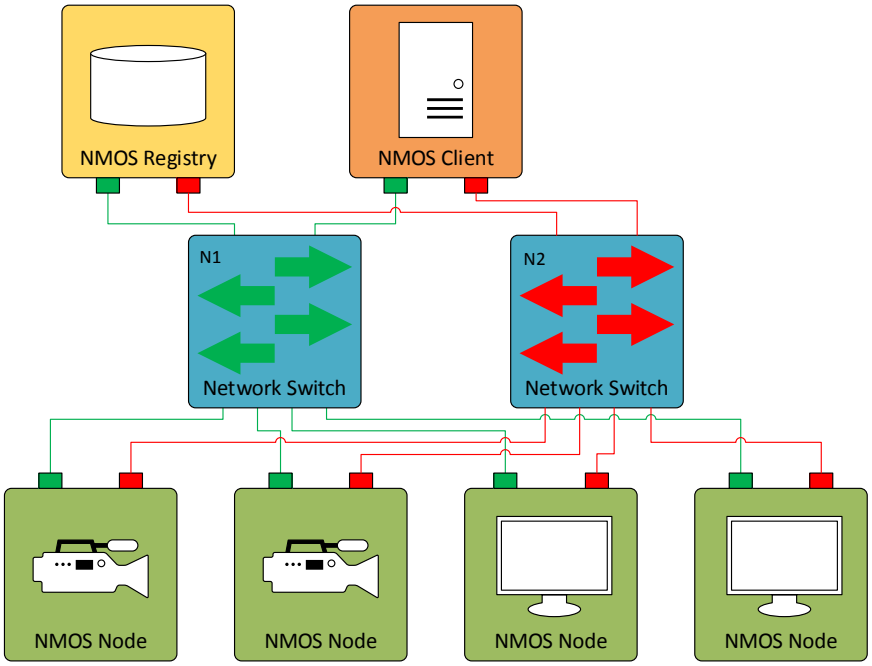


Figure 15. Operation with Multiple Network Interfaces for Redundancy.

## Conclusion

The AMWA NMOS Scalability study is focused on testing the NMOS APIs for the discovery, registration and connection management of professional media devices on an IP network at scale, a key requirement within the industry.

This paper has provided results for several areas under study, including: confirming that the registration of thousands of Nodes can occur within an acceptable timeframe; testing the recovery of the Registry after a failure; testing behaviour with multiple clustered Registries; and testing different methods of Registry discovery. The results have been generated using a virtualized network and indicate that these operations can be achieved in acceptable timeframes. To help achieve this, several recommendations for best practices for implementation and deployment have been made.

Ongoing studies into the possibility of a bulk API for registration, connection management at scale and behaviour with multiple network interfaces have also been discussed.

To confirm that results achieved on a virtualized network reflect practical deployments, performing some additional testing on a real network, using a more limited number of physical end devices, would be recommended as an area of future work.

## Acknowledgements

The authors wish to thank the other members of the Advanced Media Workflow Association for their support and feedback during the course of the AMWA NMOS Scalability study. Additionally, they would like to thank their colleagues at Sony Europe and Sony Japan for supporting and contributing to this work.

## References

[1] Society of Motion Picture & Television Engineers (SMPTE), ST 2110-x:2017, "Professional Media Over Managed IP Networks."

[2] Society of Motion Picture & Television Engineers (SMPTE), "SMPTE ST 2110 FAQ," https://www.smpte.org/st-2110. [Accessed 29 September 2018].

[3] Advanced Media Workflow Association (AMWA), "What are the Networked Media Open Specifications?," https://nmos.tv/. [Accessed 29 September 2018].

[4] Advanced Media Workflow Association (AMWA), "Networked Media Open Specifications," https://github.com/AMWA-TV/nmos. [Accessed 29 September 2018].

[5] Joint Task Force on Networked Media (JT-NM), "Reference Architecture - Joint Task Force on Networked Media (JT-NM)," http://jt-nm.org/RA-1.0/. [Accessed 31 August 2018].

[6] Advanced Media Workflow Association (AMWA), "Glossary - AMWA-TV/nmos Wiki," https://github.com/AMWA-TV/nmos/wiki/Glossary. [Accessed 29 September 2018].

[7] Advanced Media Workflow Association (AMWA), "AMWA IS-04 NMOS Discovery and Registration Specification," https://github.com/AMWA-TV/nmos-discovery-registration. [Accessed 29 September 2018].

[8] Advanced Media Workflow Association (AMWA), "AMWA IS-05 NMOS Device Connection Management Specification," https://github.com/AMWA-TV/nmos-device-connection-management. [Accessed 29 September 2018].

[9] Mininet Team, "Mininet: An Instant Virtual Network on your Laptop (or other PC) – Mininet," http://mininet.org/. [Accessed 29 September 2018].

[10] Sony Corporation, "An NMOS (Networked Media Open Specifications) Registry and Node in C++ (IS-04, IS-05)," https://github.com/sony/nmos-cpp. [Accessed 29 September 2018].

[11] Apache HTTP Server Project, "Access Log: Common Log Format," https://httpd.apache.org/docs/2.4/logs.html#common. [Accessed 29 September 2018].

[12] Elastic, "The Elastic Stack," https://www.elastic.co/products. [Accessed 29 September 2018].

[13] British Broadcasting Corporation, "BBC reference implementation demonstrating both AMWA IS-04 and IS-05," https://github.com/bbc/nmos-joint-ri. [Accessed 31 August 2018].

[14] CoreOS, "Distributed reliable key-value store for the most critical data of a distributed system," https://github.com/etcd-io/etcd. [Accessed 29 September 2018].

[15] AMWA, "Bulk API for Registrations," https://github.com/AMWA-TV/nmos-discovery-registration/issues/17. [Accessed 29 September 2018].

[16] Society of Motion Picture & Television Engineers (SMPTE), ST 2022-7:2013, "Seamless Protection Switching of SMPTE ST 2022 IP Datagrams."