NASA/TM–20220005503

Air Traffic Management TestBed: Messaging Performance

Chok Fung Lai
Ames Research Center, Moffett Field, California

Phu V. Huynh
Universities Space Research Association, Moffett Field, California

April 2022

NASA/TM–20220005503

Air Traffic Management TestBed: Messaging Performance

Chok Fung Lai
Ames Research Center, Moffett Field, California

Phu V. Huynh
Universities Space Research Association, Moffett Field, California

National Aeronautics and
Space Administration

Ames Research Center
Moffett Field, CA 94035-1000

April 2022

Acknowledgments

The authors would like to thank the other Air Traffic Management TestBed team members, Alan Lee, Huu Huynh, and Jimmy Nguyen for their contributions to the core capabilities. The authors would also like to thank Dr. Srba Jovic, Mohamad Refai, Dr. Seungman Lee, and Sandra Lozito for reviewing this technical memorandum.

Abstract

The Air Traffic Management (ATM) TestBed is an air traffic management modeling and simulation platform and framework developed by the National Aeronautics and Space Administration (NASA) to help design, configure, integrate, run, and monitor air traffic simulations. The communication middleware, implemented in the TestBed framework layer, is a core feature for data message exchange. The feature provides an abstraction layer called Messaging Support to allow switching one middleware to another without a need to rebuild the simulation components. Messaging performance such as latencies, run durations, and throughputs are important factors. Low latencies can produce accurate results in high-fidelity and visualization models. Short run durations are preferred because better run efficiency can be achieved. High throughputs allow more runs to be executed concurrently. This technical memorandum studies and compares the messaging performance by running a full-day, fast-time simulation using three communication middleware as well as tweaking the default communication middleware settings used by the TestBed. Results indicate that the messaging performance could be improved by disabling either compression or persistence settings, while the run duration and throughput could be further improved by disabling both settings with a tradeoff of the message latencies increased by a factor of ten.

This page intentionally left blank.

Table of Content

List of Figures

List of Tables

1. Nomenclature

This section lists all the symbols used in this technical memorandum.

B	= Message Broker
D	= Decoder in the subscriber
E	= Encoder in the publisher
f	= Encoder function
f^{-1}	= Decoder function
i	= i-th measurement (message length, latency, frequency, or percentile)
M	= Message to be published
M_T	= Message being transmitted
n	= Number of measurements
P	= Publisher module in the publisher component
S	= Subscriber module in the subscriber component
T_B	= Duration required to transmit an encoded message M_T via the message broker
T_D	= Duration required to decode a message M_T to M
T_E	= Duration required to encode a message M to M_T
T_M	= Duration required to transmit a message from the publisher to the subscriber
T_P	= Publication time when a message is published
T_S	= Subscription time when a message is received
v_i	= Value of the i-th measurement
$\bar{v}$	= Mean of the measurement values
σ	= Standard deviation
X	= Publisher component
Y	= Subscriber component

In addition, Table 1.1 lists the units of measurement used in the messaging performance data collection and analysis.

Table 1.1. Units of Measurement

Symbol	**Unit**	**Unit Of**	**Description**
`%`	Percentage	--	The dimensionless unit of a fraction of 100. One percentage equals one-hundredth, i.e., $1\% = 1/100$.
`B`	Byte	Data	One byte has eight bits, or binary digits.
`GB`	Gigabyte	Data	One gigabyte equals $1{,}000^3$ bytes, or 1,000,000,000 bytes.
`MB`	Megabyte	Data	One megabyte equals $1{,}000^2$ bytes, or 1,000,000 bytes.
`Mbit`	Megabit	Data	One megabit equals $1{,}000^2$ bits, or 1,000,000 bits.
`ms`	Millisecond	Time	The unit of time. One millisecond equals one-thousandth of a second, i.e., $1ms = 1/1{,}000\ s$.
`msg`	Message	Count	The number of messages.
`s`	Second	Time	The unit of time. One second equals 1,000 milliseconds.

2. Introduction

The Air Traffic Management (ATM) TestBed, or TestBed in short, is a platform for running air traffic simulations by providing a standard way for exchanging data messages among simulation components [1]. The platform is developed by the National Aeronautics and Space Administration (NASA) to accelerate the concept and technology development by running realistic simulations of current and proposed future air traffic concepts. The envisioned way to achieve effective simulations is to provide access to components, namely realistic air transportation data, air traffic control operational systems, and simulation tools. Simulation layouts can be created via a web-accessible drag-and-drop graphical user interface tool. Simulation components represented by blocks in the layouts can be started up at local, remote, or both facilities. Message flows between publisher components and subscriber components are represented by directed links. Individual simulations can be monitored via a frontend web portal. In addition, data messages published by individual components during a simulation can be recorded and then played back in subsequent simulations. Figure 2.1 shows the simulation layout used in the Boeing 2018 *ecoDemonstrator* [2] flight test.

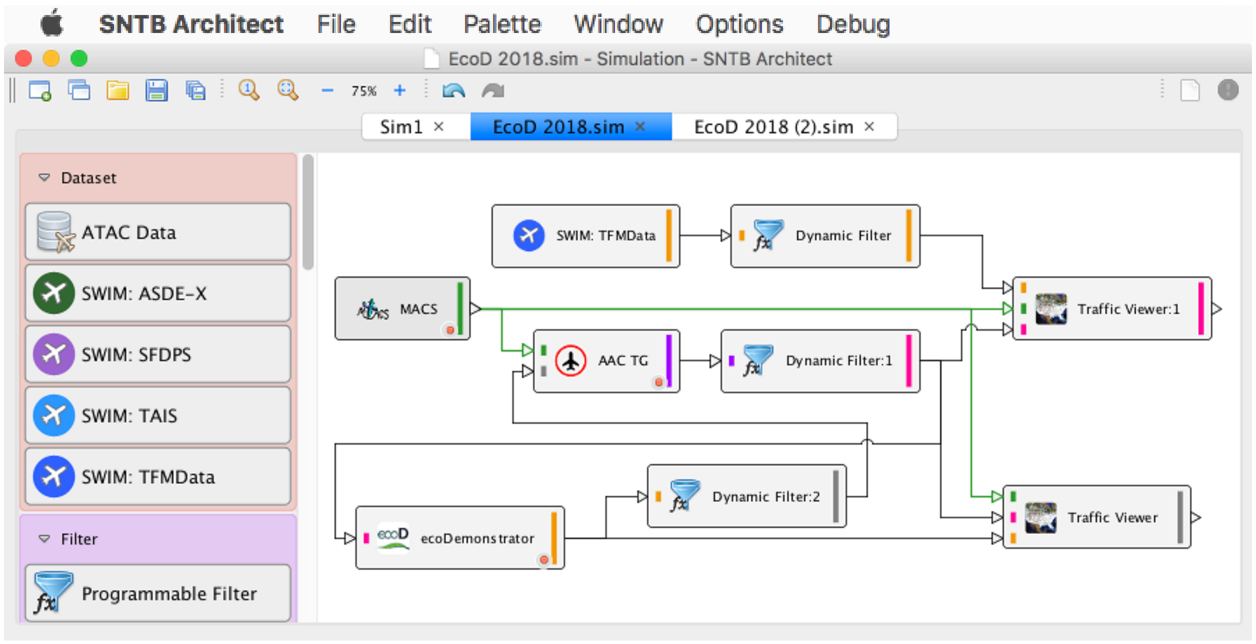


Figure 2.1. Boeing 2018 ecoDemonstrator Flight Test Simulation Layout

The communication middleware, implemented in the TestBed framework layer, is a core feature for data message exchange. The feature provides an abstraction layer called *Messaging Support* to allow switching one middleware to another one without a need to rebuild the simulation components called plugin adapters. Network load and bandwidth affect the communication latency and throughput. For example, a high-loaded network will reduce the bandwidth for exchanging messages, and a poor network connection will have packet drop and message loss. To understand the messaging performance between a publisher component and a subscriber component, there is a need to measure performance metrics without the networking factors. Simulation performance usually includes message count, message latency, and throughput [3, 4, 5]. The measurements can serve as a basis for further messaging performance evaluation using network infrastructure including laboratory networks, virtual private networks, and cloud computing as well as communication features including data encoding and message encryption.

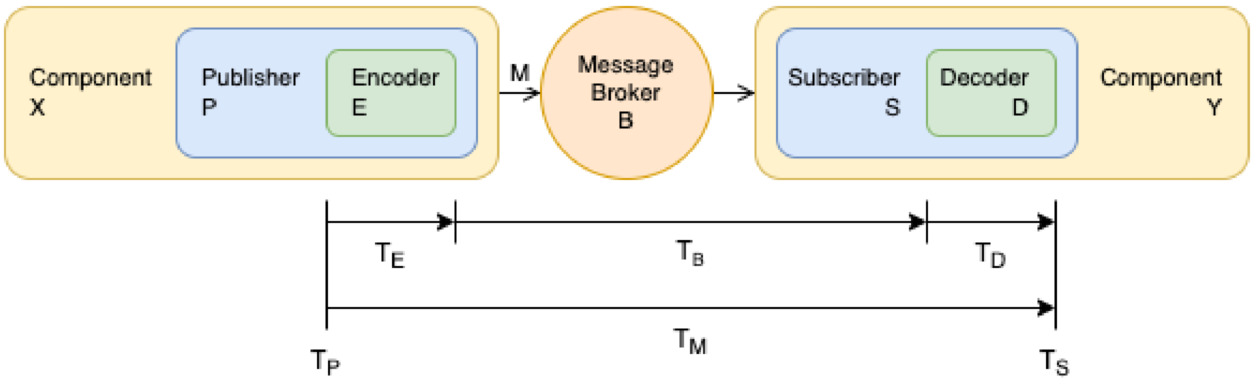


Figure 2.2. Message Flow Diagram

Figure 2.2 depicts a flow diagram of a data exchange model [6] message, *M*, published from a publisher Component *X* to a subscriber Component *Y*:

1. *X* passes *M* to Publisher *P*.
2. *P* converts *M* into a transmissible form, M_T, via Encoder *E*.
3. *P* then publishes M_T to Message Broker *B*.
4. Subscriber *S* receives M_T from *B*.
5. *S* converts M_T back into the original form, *M*, via Decoder *D*.
6. *S* then passes *M* to *Y*.

From a user's perspective, X→B→Y represents the message flow from one component to another component, and the internal parts, including P, E, S, and D, are considered hidden; from a component's perspective, P→B→S indicates the message flow from a publisher to a subscriber as components do not directly connect to the message broker but via the TestBed's *Messaging Support* application programming interfaces (APIs).

Since both Components X and Y communicate with a message having the same data representation, the decoder is an inverse function of the encoder. Assume the encoder applies a series of functions, $M_T = (f_1 \circ f_2 \circ \ldots \circ f_n)(M)$, to convert a message into a transmissible form. To correctly convert a received message back into the original form, the decoder must apply a series of inverse functions, $M = (f_n^{-1} \circ \ldots \circ f_2^{-1} \circ f_1^{-1})(M_T)$. For example, if the encoder performs three functions in sequential order: serialization, compression, and encryption, then the decoder must perform these three functions in sequential order: decryption, decompression, and deserialization.

In a simulation, factors affecting messaging performance include message processing time, network latency, and network bandwidth. The message processing time includes durations required to encode a message (T_E) published by a publisher and decode a message (T_D) received by a subscriber. The network latency includes durations required to transmit a message as one or multiple data packets from the publisher to the subscriber via the message broker (T_B). The network throughput determines the number of messages that can be transmitted in a given period of time. The message latency (T_M) is a sum of the three duration-related factors, i.e., $T_M = T_E + T_B + T_D$. Network bandwidth affects the network latency, hence the message latency.

This technical memorandum studies and compares the performance of the TestBed messaging system by measuring message latencies and run durations, as well as calculating throughput between a publisher component and a subscriber component. The experiment setup is presented in Section 3. Results are discussed in Section 4. Finally, Section 5 concludes the messaging performance findings.

3. Experiment Setup

This section describes the experiment setup used for this study. Historical traffic data on Wednesday, March 23rd, 2016, a day of good weather and high traffic volume, were obtained from the Federal Aviation Administration's (FAA's) System Wide Information Management (SWIM) via the NASA's Sherlock ATM Data Warehouse [7]. Track messages were published to the subscriber every second in fast-time simulations.

The hardware used in the experiment runs was a MacBook Pro (2019 model) with a 2.6 GHz 6-Core Intel Core i7 processor. The machine had 16 GB (2×8 GB) 2,400 MHz DDR4[*] memory manufactured by SK Hynix. Two graphics processing units are available: (a) Radeon Pro 560X 4 GB manufactured by Advanced Micro Devices and (b) Intel UHD Graphics 630 1536 MB. In addition, the network interface is an Apple Wi-Fi with 304.2 Mbit/s link speed.

The machine had the macOS Catalina version 10.15.7 installed. The software used in the experiment runs included TestBed version 2.2, OpenJDK version 11.0.2 (2019-01-15) with Runtime Environment 18.9 (build 11.0.2+9). The Java virtual machine is configured as OpenJDK 64-Bit Server VM 18.9 (build 11.0.2+9, mixed mode).

To measure messaging latency, metadata including publication time, encoding time, decoding time, and subscription time were added to each message. Latency, T_M, of a message M is defined by the duration between the publication time, T_P, when a message is published and the subscription time, T_S, when a message is received, i.e., $T_M = T_S - T_P$.

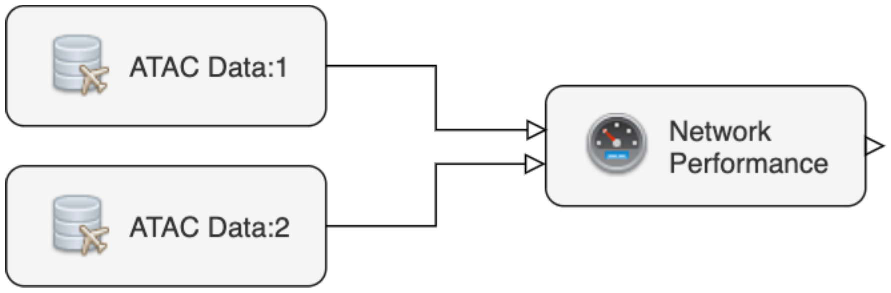


Figure 3.1. Fast-time Simulation Layout

Figure 3.1 depicts a notional diagram of the fast-time simulation layout. The "ATAC Data" blocks were configured to play back, at one-second granularity, historical traffic data processed by Airborne Tactical Advantage Company (ATAC) and stored in the NASA's Sherlock ATM Data Warehouse. Each ATAC data file stores historical track messages of a single day. To play back historical track messages in the United States on Wednesday, March 23rd, 2016, two ATAC Data blocks were used. The first block was configured to play back track messages between 2016/03/23 07:00Z (03/23 00:00 PDT[†]) and 2016/03/24 07:00Z (03/24 00:00 PDT). The second block was configured to play back track messages between 2016/03/24 00:00Z (03/23 17:00 PDT) and 2016/03/24 07:00Z (03/24 00:00 PDT). Table 3.1 lists the properties of the blocks configured in the fast-time simulations.

Three message-oriented middleware used in this study are briefly described in Section 3.1. Section 3.2 lists seven experiments in this study. The data exchange model used in the TestBed is mentioned in Section 3.3. Finally, Section 3.4 documents the approach to collect measurement data.

[*] Double Data Rate 4
[†] Pacific Daylight Time is seven hours behind the Zulu Time.

Table 3.1. Properties of Blocks

Block	Group	Name	Value
ATAC Data:1	User	Scheduled Period	1 s
		Scheduled Delay	0 s
		Traffic Start Time	2016-03-23T07:00:00Z
		Traffic End Time	2016-03-24T07:00:00Z
		Data Category	USA
		Publish Flight Plan	(not selected)
		Publish Track	✓
	Component	Fast Time	✓
	Component Manager	Host Key	localhost
ATAC Data:2	User	Scheduled Period	1 s
		Scheduled Delay	61200 s
		Traffic Start Time	2016-03-24T00:00:00Z
		Traffic End Time	2016-03-24T07:00:00Z
		Data Category	USA
		Publish Flight Plan	(not selected)
		Publish Track	✓
	Component	Fast Time	✓
	Component Manager	Host Key	localhost
Network Performance	Component	Fast Time	✓
	Component Manager	Host Key	localhost

3.1. Message-Oriented Middleware

Three message-oriented middleware software were used to measure the performance: Apache ActiveMQ [8], TestBed Base Architecture for Simulation Integrated Communication (Basic), and Neural Autonomic Transport System (NATS) [9].

Apache ActiveMQ is a widely used open-source messaging server written in Java programming language [10]. Version 5.15.8 is currently used in the TestBed and in this study. It is also bundled in the TestBed Software Development Kit.

TestBed's Basic Server is a native socket-based publish-subscribe system, also written in Java programming language, to provide a lower-latency, higher-throughput performance than the ActiveMQ by implementing a minimal set of features and capabilities, specific for testing the network connectivity among assets. The implementation uses server and client socket connections based on data input and output streams.

NATS is an open-source messaging server written in the Go programming language [11]. NATS is designed for modern distributed systems and performance due to its high sender and receiver throughput. Version 2.1.8 is used in this study.

3.2. Experiments

Table 3.2 and Table 3.3 list the experiment runs in this study. The default runs used the TestBed's existing settings; the ActiveMQ runs used the ActiveMQ middleware with the default and tweaked settings. The Run column indicates the name of the experiment runs. The Middleware column indicates the middleware application to be used in the experiment runs. The baseline experiment (Run 1A) did not use middleware to transmit the Track messages. Messages were transmitted using Java calls and callbacks. Broker B was not used, and both the components X and Y were running within the same Java Virtual Machine. Thus, the message flowed from X to Y was X→P→E→S→D→Y (without B). For the other runs, middleware is used. The ActiveMQ (Run 2A) is the middleware bundled in the TestBed Software Development Kit. The Basic (Run 3A) leverages the Basic middleware implemented by the TestBed team. The NATS (Run 4A) uses the NATS middleware written in a non-Java programming language.

Table 3.2. Experiment Runs (Default Runs)

Run	Middleware	Remarks
1A	--	Baseline, no middleware
2A	ActiveMQ	Default TestBed settings
3A	Basic	Default TestBed settings
4A	NATS	Default TestBed settings

The ActiveMQ connection settings in TestBed enable both message compression and persistence settings. Both the Basic and the NATS do not have such settings. Message compression reduces the message size by using a compression algorithm. Message persistence allows transmitting messages to be recovered in case of the middleware restarts by writing each message to a permanent storage device such as a hard drive. The settings are useful under certain circumstances. However, they may degrade the messaging performance in other circumstances. Enabling the compression setting is desired when there is a need to transmit large messages with redundant information, e.g., element names in an Extensible Markup Language (XML) text appear in both start and end tags. On the other hand, disabling the compression setting is desired when the messages being transmitted are short or contain highly compressed data such as image data. Similarly, enabling the persistence setting is desired when there is a need to recover messages upon hardware or software failure; disabling the persistence setting is desired when middleware redundancy is used, or simulations are running in a local environment without using distributed systems.

Table 3.3. Experiment Runs (ActiveMQ Runs)

Run	Middleware	Remarks
2A	ActiveMQ	Default TestBed settings
2B	ActiveMQ	No message compression: • `jms.useCompression=false`
2C	ActiveMQ	No message persistence: • `delivery_mode=non_persistent`
2D	ActiveMQ	No compression and persistence: • `jms.useCompression=false` • `delivery_mode=non_persistent`

In this study, combinations of the two settings controlling the ActiveMQ were also explored. Thus, a total of seven runs were executed in this study. The first four runs (1A, 2A, 3A, and 4A) used in-memory and three middleware, while the last three runs (2B, 2C, and 2D) used the ActiveMQ middleware with non-default settings.

3.3. Message Type

The TestBed Data Exchange Model (SNDEM) is a standardized format of the information to be exchanged among the components [6]. In this study, Track messages are measured because of their dominance in the ATAC data files. A Track message represents a single, three-dimensional location of a vehicle on the ground or in the airspace. Each track message has a vehicle identifier such as callsign; location such as latitude, longitude, and altitude; time when the track was recorded in the system; and vehicle states such as groundspeed, flight course, and vertical speed. Note that information could be missing in a Track message. Missing numeric values are recorded as *NaN*[‡].

Each SNDEM message also has a section storing metadata and information about the message, including the name of its publisher, a format version, a run name, a publication timestamp recorded when the message was published, and a subscription timestamp when the message was received.

Currently, SNDEM messages are serialized using JavaScript Object Notation (JSON) [13] strings. Future versions will support additional data serialization including a variable-length quantity encoding.

3.4. Data Collection

During an experiment run, the publication and subscription timestamps defined in the message's metadata section are used to calculate the messaging latency. To reduce the overhead of recording the message lengths and latencies, the measurement values are recorded in a compact, pre-memory allocated (about 43.5 MB) data structure residing in the Java Virtual Machine, i.e., computer memory. Upon the completion of an experiment run, the data structure is written to a file for post-processing and performance analysis.

Running TestBed simulations requires three service managers:

1. Simulation Manager manages all simulation runs.
2. Execution Manager manages one simulation run.
3. Component Manager manages individual plugin adapters.

The TestBed service managers and individual plugin adapters log progress information to log files. The progress information includes timestamps of seven simulation states—deployment, startup, initialization, execution, shutdown, archival, and retirement. These timestamps are used for measuring the run durations. Finally, the message lengths and run durations are used to calculate the throughput metrics.

4. Performance Analysis

The fast-time simulation covered a 24-hour period of historical traffic data. The "ATAC Data:1" and "ATAC Data:2" components played back 41,255,168 and 2,053,751 track messages, respectively. Thus, a total of 43,308,919 track messages were handled. Each track message contains two parts: (a) a key which is an identifier (callsign) of a vehicle; and (b) a payload that is in the SNDEM format. The length of a track message is the sum of the lengths of the key and the payload. Table 4.1 lists a sample track message with a length of 207 bytes (7 bytes key and 200 bytes payload) published during a simulation.

Table 4.1. Sample Track Message

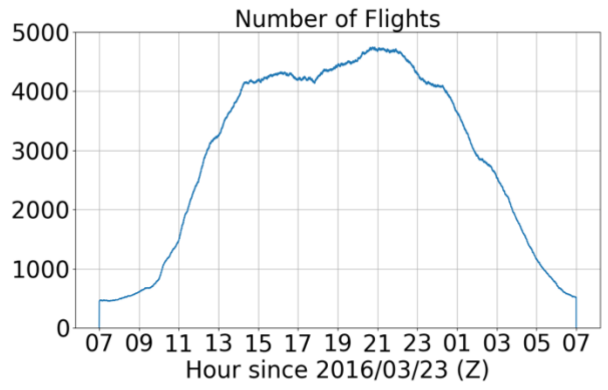
	Content	**Length (bytes)**
Key	UAL1967	7
Payload	{"vid":"UAL1967","latDeg":37.62231,"lonDeg":-122.38169,"altFt":6.0,"time":1458716400000,"gsKt":20.0,"crsDeg":286.0,"vsFpm":NaN,"meta":{"src":"ATAC Data.1","ver":"2.3","run":"1A","tpub":1603483450633}}	200

[‡] Not a Number

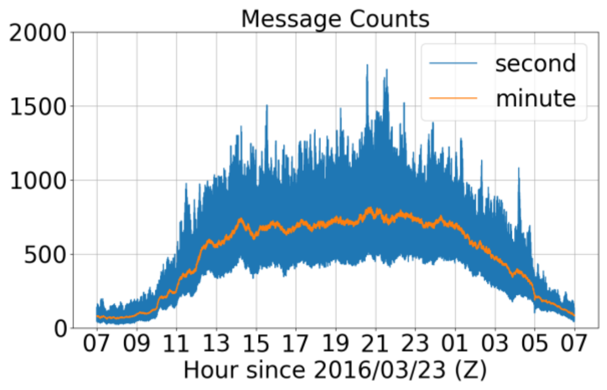
The message frequencies of the setup are described in Section 4.1. Section 4.2 discusses the message latencies among different runs. The run durations are listed in Section 4.3. Finally, the calculated throughputs are presented in Section 4.4.

4.1. Message Frequency

Message frequencies measure the lengths and frequencies of the messages published by the ATAC Data components. The messages are grouped by the simulated time at a one-second granularity level. In addition, ATAC Data recorded track data at a one-minute granularity level. Figure 4.1(a) shows the number of flights in the United States on the chosen simulated day. The maximum number of flights (4,744) was observed at both 20:44:43Z and 20:44:44Z on March 23rd, 2016. Figure 4.1(b) shows the number of published messages per second (blue) and the one-minute rolling average of the published messages (orange). The maximum number of published messages per second (1,779) was observed at 20:35:47Z on March 23rd, 2016.



(a)



(b)

Figure 4.1. Flight and Message Counts

Table 4.2 lists the statistics of the messages with respect to the message lengths and message frequencies. Given a series of n collected measurement values, $V = \{v_1, v_2, \dots, v_n\}$, the *Min* and *Max* columns represent the minimum and the maximum values recorded, v_{min} and v_{max}, respectively:

$$v_{min} = min\{V\}$$
$$v_{max} = max\{V\}$$
$$where\ V \in \{v_1, v_2, \dots, v_n\}$$

The $P_{25\%}$, $P_{50\%}$, and $P_{75\%}$ columns represent the 25th, 50th, and 75th percentiles, respectively. Note that the $P_{50\%}$ values are also the medians. The formula to calculate the *i-th* percentile is

$$P_{i\%} = v_j,$$
$$where\ i = [1, 100]$$
$$and\ j = \min\left\{\max\left\{1, \left\lceil \frac{i}{100} \cdot n \right\rceil\right\}, n\right\}$$

The *Mean* column represents the average measurement values:

$$\bar{v} = \frac{\sum_{i=1}^{n} v_i}{n}$$

The *SD* column represents the corrected sample standard deviations:

$$\sigma = \sqrt{\frac{1}{n-1} \cdot \sum_{i=1}^{n} (v_i - \bar{v})^2}\,, where\ \bar{v}\ is\ the\ mean.$$

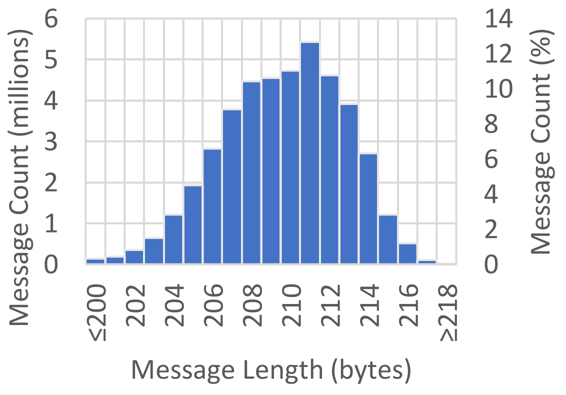
The *Unit* column represents the units of measurement. The *Count* column represents the number of measurements collected, i.e., n., and the count values are always greater than one in this study.

In each experiment run, 43.3 million messages were published, and these messages generated 9.09 billion bytes (or 9.09 GB) for data transfer. The mean and standard deviation of the message lengths were 209.63 bytes and 3.15 bytes, respectively. When the messages were grouped by the simulated time at one-second granularity level, a total of 86,400 (=24 hours × 60 minutes × 60 seconds) message frequencies were collected. The mean and standard deviation of the message frequencies were 501.26 msg/s and 281.79 msg/s, respectively.

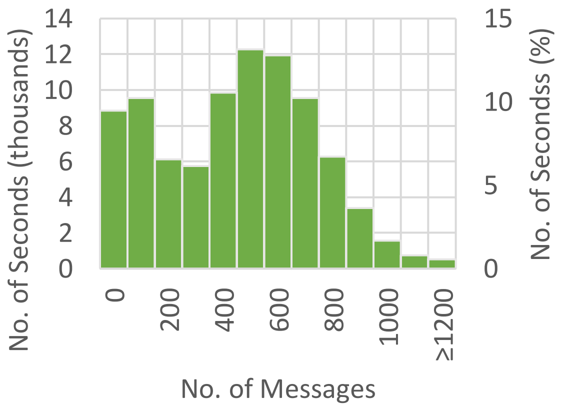
Table 4.2. Message Statistics

Message	Min	$P_{25\%}$	$P_{50\%}$	$P_{75\%}$	Max	Mean	SD	Unit	Count
Lengths	193	207	210	212	219	209.63	3.15	bytes	43,308,919
Frequencies	24	247	525	704	1779	501.26	281.79	msg/s	86,400

Figure 4.2(a) shows the distribution of the lengths of the published messages throughout the simulated day. The message lengths are not fixed because the data exchange model uses the JSON format and numeric values are represented in variable-length characters rather than fixed-length bytes. The chart shows that the message lengths follow a normal distribution. More than half (54.9%) of the messages contained 208-212 bytes of data. In addition, the distribution of the message frequencies is depicted in Figure 4.2(b). During the simulation, half (50.5%) of the simulated seconds published 400-800 messages.



(a) Histogram of Message Lengths



(b) Histogram of Message Frequencies

Figure 4.2. Histograms of Track Messages Grouped by: (a) Lengths and (b) Frequencies

4.2. Message Latency

Latency of a message is the duration between the time when the message was published by the publisher and the time when the message was received by the subscriber. Each SNDEM message has a metadata section recording the message's publication time, T_P, and subscription time, T_S. Thus, latency, T_M, is defined as $T_M = T_S - T_P$. Note that the system clocks on the publisher and subscriber must be synchronized. In this study, since the publishers (ATAC Data) and subscriber (Network Performance) are running on the same machine, the components are using the same system clock.

Ideally, message latency values are zero or a low fixed value so that simulated components can receive messages in a regular time interval. Fixed latency values are important for algorithms and visualizations that require interpolation or extrapolation based on consecutive data points [13, 14].

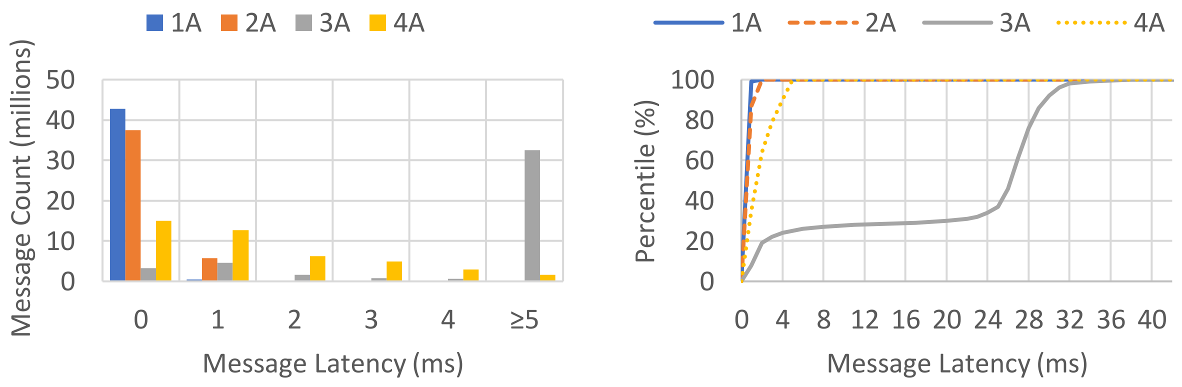
4.2.1. Message Latencies among Default Runs

Table 4.3 lists the statistics of the message latency values among the first four runs. The baseline (Run 1A) has the shortest latency. This is expected as middleware is not used in this run. Thus, Track message exchanges are performed in memory instead of via middleware. When middleware is used, the ActiveMQ (Run 2A) has minimal message latency. Both the Basic (Run 3A) and NATS (Run 4A) have message loss and the final message counts (43,304,541 and 43,308,917) do not match the message count (43,308,919) in the baseline. The message loss might be due to the shutdown of simulation components while the message queue in the middleware is not empty. The table also shows that the Basic (Run 3A) has the largest averaged latency (19.7 ms) and standard deviation (11.5 ms).

Table 4.3. Latency Measurements (Default Runs)

Run	Min	$P_{25\%}$	$P_{50\%}$	$P_{75\%}$	Max	Mean	SD	Unit	Count
1A	0	0	0	0	70	0.0113	0.119	ms	43,308,919
2A	0	0	0	0	38	0.134	0.342	ms	43,308,919
3A	0	5	26	27	57	19.7	11.5	ms	43,304,541
4A	0	0	1	2	139	1.49	3.144	ms	43,308,917

Among the default runs, the histograms of the message latency are shown in Figure 4.3(a) and the percentiles of message latency are shown in Figure 4.3(b). The percentile chart indicates the percentages of messages with at most 1 ms latency among the runs are: 99% (Run 1A), 87% (Run 2A), 8% (Run 3A), and 35% (Run 4A). When middleware is used, in terms of the message latency, the ActiveMQ is the best option among the three tested middleware because of its low and stable latency.



(a) Histogram *(b) Percentile*

Figure 4.3. Message Latency (Default Runs)

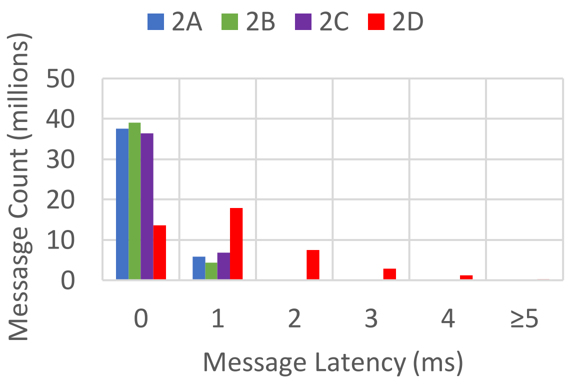
4.2.2. Message Latencies among ActiveMQ Runs

Table 4.4 lists the statistics of the message latency values among the ActiveMQ runs. Disabling the message compression setting (Run 2B) reduces the average latency from 0.134 ms to 0.0996 ms (25.7% reduction). Disabling the message persistence setting (Run 2C) increases the average latency from 0.134 ms to 0.159 ms (19.9% increment). In addition, when both the message compression and persistence settings are disabled (Run 2D), the average latency increases from 0.134 ms to 1.09 ms (713% increment). The behavior is observed because the reduction of the run durations increases the load and utilization of the computer's central processing units (CPUs). That means the frequency of the messages increases due to reduced message processing time, so the subscriber activities result in higher CPU utilization reducing simulation run time. As a result, more messages are queued in the middleware delaying delivery of a higher percentage of messages.

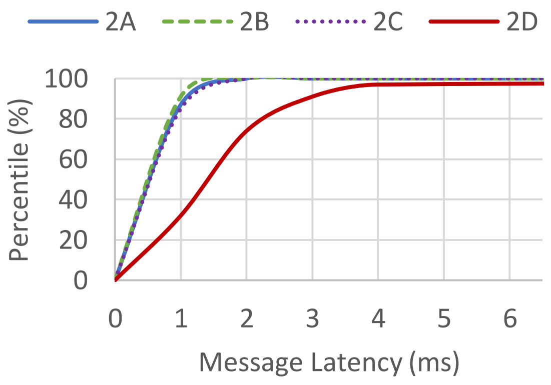
Table 4.4. Latency Measurements (ActiveMQ Runs)

Run	Min	$P_{25\%}$	$P_{50\%}$	$P_{75\%}$	Max	Mean	SD	Unit	Count
2A	0	0	0	0	38	0.134	0.342	ms	43,308,919
2B	0	0	0	0	35	0.0996	0.301	ms	43,308,919
2C	0	0	0	0	43	0.159	0.373	ms	43,308,919
2D	0	0	1	2	50	1.09	1.03	ms	43,308,919

Among the ActiveMQ runs, the histograms of the message latency are shown in Figure 4.4(a) and the percentiles of the message latency are shown in Figure 4.4(b). The percentile chart indicates the percentages of messages with at most 1 ms latency among the runs are: 87% (Run 2A), 91% (Run 2B), 85% (Run 2C), and 32% (Run 2D). When the ActiveMQ middleware is used, in terms of the message latency, disabling the message compression setting is the best option because of its low and stable latency.



(a) Histogram



(b) Percentile

Figure 4.4. Message Latency (ActiveMQ Runs)

4.3. Run Duration

During a simulation run, each simulation component goes through a series of seven states. A state is advanced only if all the components complete the state. For example, the Execution state starts when the components complete the Initialization state. The seven states are listed below.

1. Deployment: Downloads and extracts the component archive file to a component-specific directory.
2. Startup: Starts up the component by calling the operating system-specific run script.
3. Initialization: Prepares the component for execution. For example, an ATAC Data component reads and parses the data files.
4. Execution: Executes the component. For example, an ATAC Data component publishes track messages associated with the current simulation time, and a Network Performance component collects statistics from the incoming track messages.
5. Shutdown: Stops the component and releases resources.
6. Archival: Archives all the files that are generated or modified by the component.
7. Retirement: Does nothing in the current experiments.

The duration for each experiment run is presented in this section. A shorter run time is preferred for efficiency. Durations associated with deployment, startup, and initialization states depend on the hard drive and CPU performance. In addition, the durations associated with the shutdown, archival, and retirement states are short and within ten seconds. These six states are not related to the messaging performance. On the other hand, the duration of the execution state heavily depends on the messaging performance. Thus, execution durations of the experiment runs will be analyzed in the following subsections.

4.3.1. Run Durations among Default Runs

Table 4.5 lists the run durations among the default runs. Note that the durations are measured in seconds. The baseline (Run 1A) has the shortest execution time. This is expected as track message exchanges are performed in memory instead of via middleware. Note that the baseline run has the longest archival time (9.4 s) due to the inclusion of writing message length statistics. The NATS (Run 4A) has the best middleware performance since its execution time is the shortest among the non-baseline runs.

Table 4.5. Durations of Default Runs, in seconds

Run	**1A**	**2A**	**3A**	**4A**
Deployment	2.874	3.281	3.454	3.374
Startup	1.534	1.791	1.645	1.620
Initialization	117.521	98.085	105.340	108.598
Execution	663.702	4786.092	854.402	673.550
Shutdown	0.655	0.301	0.278	0.281
Archival	9.403	1.776	1.359	1.036
Retirement	0.011	0.012	0.010	0.009
Total	795.700 (13’16”)	4,891.338 (81’31”)	966.488 (16’06”)	788.468 (13’08”)

The data messaging exchanges happened in the execution phase. Figure 4.5 shows the execution durations among runs. The first and last Track messages were published at the 0% and the 100% mark, respectively. When middleware is used, in terms of the run durations, the NATS is the best option among the three tested middleware because of its shortest run duration.

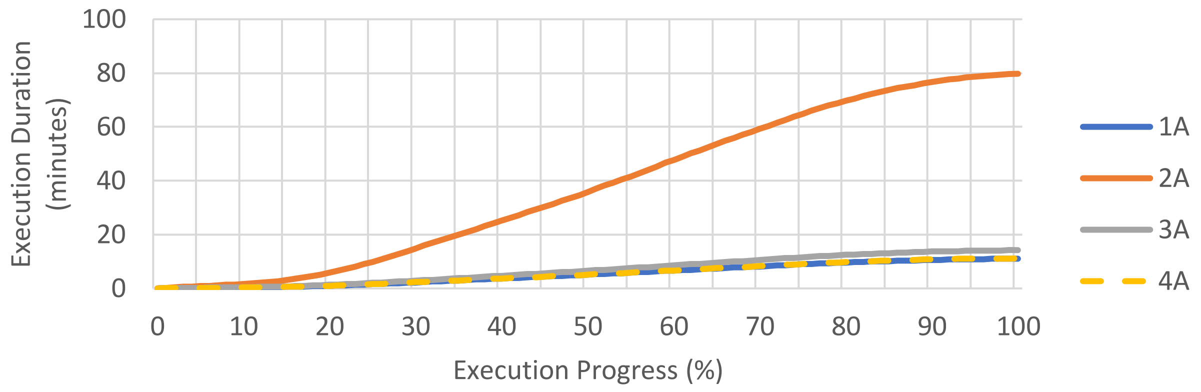


Figure 4.5. Execution Durations (Default Runs)

4.3.2. Run Durations among ActiveMQ Runs

Based on the results presented in Section 4.3.1, the simulation using the ActiveMQ (Run 2A) was the slowest. Fine-tuning the settings, including message compression and persistence, would improve the messaging performance. Table 4.6 shows the run durations of the ActiveMQ middleware with the default settings (Run 2A), the compression setting disabled (Run 2B), the persistence setting disabled (Run 2C), and both the settings disabled (Run 2D).

Table 4.6. Durations of ActiveMQ Runs, in seconds

Run	**2A**	**2B**	**2C**	**2D**
Deployment	3.281	3.506	3.339	3.492
Startup	1.791	2.245	2.290	2.281
Initialization	98.085	108.170	117.378	113.919
Execution	4786.092	3582.173	2651.574	1468.028
Shutdown	0.301	0.275	0.272	0.330
Archival	1.776	1.546	1.366	1.847
Retirement	0.012	0.011	0.012	0.013
Total	4,891.338 (81’31”)	3,697.926 (61’38”)	2,776.231 (46’16”)	1,589.910 (26’30”)

First, disabling the compression setting reduces the execution time by 25.2%, from 4,786 seconds (Run 2A) to 3,582 seconds (Run 2B). Compression reduces the message size with CPU cost for compression and decompression operations. If messages being transmitted are short or have compressed data, compression would reduce the overall messaging performance.

Second, disabling the persistence setting reduces the execution time by 44.6%, from 4,786 seconds (Run 2A) to 2,652 seconds (Run 2C). Persistence allows data to be recovered in case of hardware and software failure. However, persistence requires both CPU cycles and disk access.

Third, disabling both the message compression and the message persistence settings the execution time reduces by 69.3%, from 4,786 seconds (Run 2A) to 1,468 seconds (Run 2D). This allows the ActiveMQ server to run without spending extra cycles on the CPU and disk access. Even though this run requires the shortest run time, the latency presented in Section 4.2.2 increases.

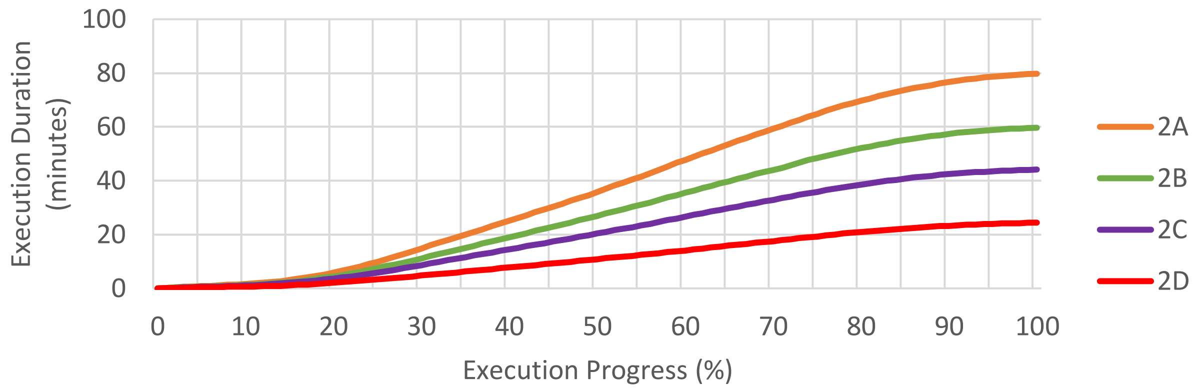


Figure 4.6. Execution Durations (ActiveMQ Runs)

Figure 4.6 shows the execution durations among the ActiveMQ runs. When the ActiveMQ middleware is used, in terms of the run durations, disabling both the message compression and persistence settings is the best option because of its shortest run duration.

4.4. Throughput Rate

In the actual networked simulation environment, network bandwidth is a critical factor to the data throughput. The baseline (Run 1A) has the shortest duration to execute (663.702 seconds) because message transmission does not involve local network connectivity. As described in Section 4.1, track messages with 9.09 billion bytes were transmitted between the publishers and the subscriber without any network connectivity. Thus, the throughput of the baseline run was 9,078,643,231 bytes × 8 bits/byte / 663.702 s = 109.43 Mbit/s. As mentioned in Section 3.2, the maximum Apple Wi-Fi speed is 304.2 Mbit/s, the local network bandwidth is not saturated on the machine.

Since the network bandwidth is not saturated and all the other runs are slower than the baseline, the local network bandwidth is not affecting the performance of the runs. Thus, the throughput of the middleware can be defined as

$$\text{throughput} = \frac{total\ bits\ transmitted}{execution\ duration\ in\ seconds}.$$

Table 4.7. Throughput of Runs, in Mbit/s

Run	**1A**	**2A**	**2B**	**2C**	**2D**	**3A**	**4A**
Middleware	--	ActiveMQ				Basic	NATS
Throughput	109.430	15.175	20.275	27.391	49.474	85.006	107.831

Table 4.7 lists the calculated throughput values among the seven runs. Using the ActiveMQ (Runs 2A, 2B, 2C, and 2D) does not produce high throughput values due to the longer execution durations. Using the NATS (Run 4A) produces a high throughput value that is near the baseline's value. This indicates that the NATS middleware has the highest performance among the three selected middleware in this study.

5. Concluding Remarks

The Air Traffic Management (ATM) TestBed is an air traffic management modeling and simulation platform and framework developed by NASA to help design, configure, integrate, run, and monitor air traffic simulations. The communication middleware, implemented in the TestBed framework layer, is a core feature for data message exchange. This technical memorandum studies and compares the messaging performance by running a full-day, fast-time simulation using three communication middleware as well as tweaking the default communication middleware settings used by the TestBed. Results indicate that the messaging performance could be improved by disabling either compression or persistence settings, while the run duration and throughput could be further improved by disabling both settings with a tradeoff of the message latencies increased by a factor of ten.

The TestBed provides a core feature for switching a communication middleware without rebuilding the simulation components. A baseline without using middleware is performed to gather the minimum messaging latency due to message encoding and decoding. Three middleware are selected: the ActiveMQ, Basic Server, and the NATS. Two selected features, message compression and persistence, in the ActiveMQ, are also explored. Messaging performance for all three middleware including latency, run durations, and throughput rates are analyzed. Low latency values are preferred when running simulations with high-fidelity and visualization models because more accurate data can be presented. On the other hand, short run durations are preferred when evaluating concepts with low-fidelity models. High throughput rates allow simulations to be run concurrently.

Different parameters should be considered when selecting the middleware. If the latency is a concern, then the ActiveMQ would be a better choice. If the run duration is a concern, then the NATS would be a better choice.

When using ActiveMQ middleware, latency and run duration are important. If the latency is a concern, then disabling the message compression setting is a better choice. If the run duration is a concern, then disabling both the message compression and persistence settings would be a better choice. If both the latency and run duration are important factors, then disabling the message persistence would be a better choice.

This study focuses on the local network environment. Future works may focus on network infrastructures such as laboratory networks, virtual private networks, and cloud computing. In addition, communication features such as data encoding and message encryption may also be explored.

6. References

1. Robinson, J.E., Lee, A., and Lai, C.F., “Development of a High-Fidelity Simulation Environment for Shadow-Mode Assessments of Air Traffic Concepts,” Royal Aeronautical Society: Modeling and Simulation in Air Traffic Management Conference, London, UK, 14-15 November 2017.
2. “Boeing: ecoDemonstrator” (Online). https://www.boeing.com/principles/environment/ecodemonstrator. Boeing. Retrieved 2022/03/31.
3. Murphy, J., Otto, N., and Jovic, S., “UAS-NAS Integrated Human in the Loop: Test Environment Report,” NASA UAS-ITE.5.1-008.001, February 17, 2015.
4. Rios, J.L., Smith, I. Mulfinger, D., Cook, B., Venkatesan, P., Baskaran, V., Lakshminarasimha, M., Ishihara, A., Liddell, D., Li, Q., Verma, P., Smith, D., Jurcak, S., and Modi, H., “UAS Service Supplier Network Performance: Results and Analysis from Flight Testing Multiple USSProviders in NASA’s TCL4 Demonstration,” NASA-TM-2020- 220462, NASA Technical Memorandum, January 24, 2020.
5. Jovic, S. and Harper, W., “Characterization Study of TestBed Infrastructure Performance in a Distributed Simulation Environment: Baseline Analysis,” NASA/TM-20205011793, NASA Technical Memorandum, December 2020.

6. Lai, C.F., “Air Traffic Management TestBed Data Exchange Model,” NASA/TM-20205001252, NASA Technical Memorandum, May 1, 2020.
7. Eshow, M.M., Lui, M., and Ranjan, S., “Architecture and Capabilities of a Data Warehouse for ATM Research,” 2014 IEEE/AIAA 33rd Digital Avionics Systems Conference (DASC), Colorado Springs, CO, 2014, pp. 1E3-1-1E3-14.
8. “ActiveMQ” (Online), http://activemq.apache.org/. The Apache Software Foundation. Retrieved 2020/10/02.
9. “NATS - Open Source Messaging System | Secure, Native Cloud Application Development” (Online), https://nats.io/. Synadia Communications, Inc. Retrieved 2020/10/02.
10. “JDK 11” (Online), https://openjdk.java.net/projects/jdk/11/. Oracle Corporation and/or its affiliates. Retrieved 2021/08/26.
11. “The Go Programming Language” (Online), https://golang.org/. Google. Retrieved 2020/10/22.
12. Bray, T., “RFC 8259 - The JavaScript Object Notation (JSON) Data Interchange Format” (Online), https://tools.ietf.org/html/rfc8259. Internet Engineering Task Force. Retrieved 2021/09/23.
13. King, D.W., “Handling Qualities Effects of Display Latency,” Annual Research Briefs-1992, Center for Turbulence Research, July 1, 1993, pp. 329-340.
14. Archdeacon, J.L., Iwai, N.H., Kato, K.H., and Sweet, B., “Reconfigurable Image Generator,” US-Patent-9,583,018, February 28, 2017.