ARL-TR-9524 ● AUG 2022

DEVCOM ARMY RESEARCH LABORATORY

AuroraXR – Motivations and Objectives for Achieving an Interoperable Framework for Cross-Reality Applications

by Mark S Dennison and Theron T Trout

NOTICES

Disclaimers

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturer's or trade names does not constitute an official endorsement or approval of the use thereof.

Destroy this report when it is no longer needed. Do not return it to the originator.

ARL-TR-9524 ● AUG 2022

AuroraXR – Motivations and Objectives for Achieving an Interoperable Framework for Cross-Reality Applications

Mark S Dennison
DEVCOM Army Research Laboratory

Theron T Trout
Stormfish Scientific Corporation

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED (From - To)
August 2022	Technical Report	1 October 2021–27 July 2022

4. TITLE AND SUBTITLE
AuroraXR – Motivations and Objectives for Achieving an Interoperable Framework for Cross-Reality Applications

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Mark S Dennison and Theron T Trout

5d. PROJECT NUMBER
XR-COP

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
DEVCOM Army Research Laboratory
ATTN: FCDD-RLC-IB
Adelphi, MD 20783-1138

8. PERFORMING ORGANIZATION REPORT NUMBER
ARL-TR-9524

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release: distribution unlimited.

13. SUPPLEMENTARY NOTES
ORCID IDs: Mark S Dennison, 0000-0001-9780-0511, Theron T Trout, 0000-0002-9464-0222

14. ABSTRACT
AuroraXR is a framework that provides the interoperability and data synchronization capabilities required to enable Army and other US Department of Defense use cases for networked augmented-, mixed-, and virtual-reality systems. AuroraXR provides mechanisms for establishing bi-directional information sharing from sensors and external systems in the real world with users and systems in virtual environments. Unlike the networking solutions that are typically used in the gaming industry, which presume and are dependent on clients having access to high-bandwidth connectivity between each other and/or servers, AuroraXR was designed for use in tactical networking architectures where bandwidth is at a premium and connectivity is not assured. In this report, we detail the purpose of AuroraXR, its subsystems and security capabilities, and future objectives for deployment of the software.

15. SUBJECT TERMS
cross-reality, virtual reality, common operating picture, human–information interaction, network, server, command and control, Military Information Sciences

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Mark S Dennison
Unclassified	Unclassified	Unclassified	UU	21	19b. TELEPHONE NUMBER (Include area code) (562) 290-9437

Standard Form 298 (Rev. 8/98)
Prescribed by ANSI Std. Z39.18

Contents

List of Figures

1. Introduction: What is AuroraXR?

In simple terms, AuroraXR is a multiuser gaming system designed with the goal of providing capabilities required to support the Army and other DOD use cases and specifically those that enable shared augmented-, mixed-, and virtual-reality (AR, MR, and VR) experiences. Beyond providing synchronization of virtual objects between multiple end-user devices, AuroraXR provides mechanisms for establishing bi-directional information sharing from sensors and external systems in the real world with users and systems in virtual environments. Unlike networking solutions typically used in the gaming industry, which presume and are dependent on clients having access to high-bandwidth connectivity between each other and/or servers, AuroraXR was designed for use in tactical networking architectures where bandwidth is at a premium and connectivity is not assured.

2. Where did AuroraXR Originate?

The work that led to AuroraXR originated in 2017 in what was then called the US Army Combat Capabilities Development Command Army Research Laboratory's Battlefield Information Processing Branch (BIPB), now the Battlefield Information Systems Branch (BISB). A broad survey of unclassified US Army and DOD AR, MR, and VR (collectively referred to as cross-reality [XR], see Fig. 1) research efforts revealed that many exciting capabilities were being explored by military scientists and engineers (S&Es), particularly toward immersive training and data visualization in head-mounted displays (HMDs). It became apparent that many of these S&Es assumed that there existed multiuser gaming engines that could natively support connecting prototype research capabilities to DOD networks, while others left networking and interoperability as a problem for another group to solve. Further analysis concluded that none of the available networking systems on the commercial market were designed with sufficient security and robustness characteristics required to pursue a DOD Authority to Operate (ATO). Moreover, none of the available multiuser engines could adapt to operate on constrained tactical networks at the edge.

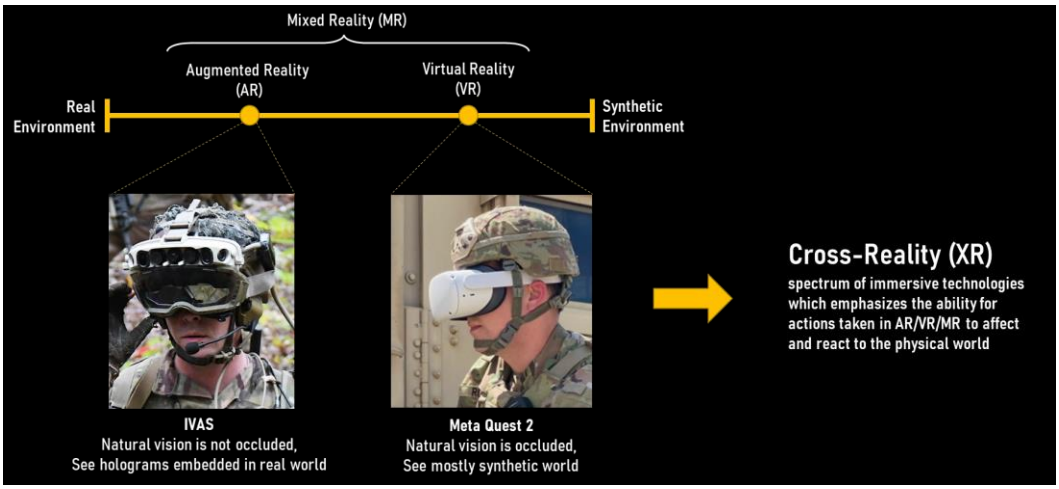

Fig. 1 Depiction of where different immersive technologies fall on the spectrum between the real environment and a fully synthetic environment

Having identified this capability gap, DEVCOM Army Research Laboratory BIPB engaged with Stormfish Scientific Corporation to perform research and development into solving the technical challenges associated with building a multiuser server architecture that could satisfy the DOD's unique requirements. This work ultimately culminated in the Accelerated User Reasoning: Operations, Research, and Analysis (AURORA) Network, aka AuroraNet, which provides the network protocols and transport layer and AURORA for Cross Reality (AuroraXR), which provides a collection of services and APIs for managing and manipulating XR objects and associated data. Additionally, the AuroraXR App Proxy was developed to simplify the integration of traditional and legacy software systems to interoperate easily with AuroraXR systems and users.

3. Multiplayer Gaming for the Military

Most multiplayer gaming engines are optimized for commodity, commercial broadband links. While security features to negate the ever-present problem of cheaters exist, few—if any—of these support Federal Information Processing Standard (FIPS) approved encryption algorithms, role-based access control, or other enterprise-grade security mechanisms.

Furthermore, these commercial networking protocols assume continuous, high-quality connections. If a player's "ping" (i.e., network-packet round-trip times) goes too high or too many packets are lost, they simply kick the player from the server or application. This is certainly not a viable approach for a Soldier using an AR device who only has high-latency, low-bandwidth, intermittent radio connectivity on which to operate. Military applications require a networking

protocol that will automatically recover when network conditions improve, use the available bandwidth intelligently when it is available, and gracefully handle connection losses and recoveries. This is particularly important for XR applications, which are often anchored on a shared world model that is stable between users.

AuroraNet is built on top of the ZeroMQ message framework.[1] ZeroMQ is a high-performance messaging library designed for building distributed applications. The "zero" in the name means it is a message queue with zero brokers. CERN's Large Hadron Collider experiment ranked ZeroMQ as the top-performing messaging library when it was developing data collection framework for its facility.[2] ZeroMQ is also open-source software licensed under the GNU Lesser General Public License V3. This makes it particularly beneficial for use in DEVCOM ARL foundational research projects that seek to transition capabilities to the DEVCOM centers, academic partners, and the private sector.

3.1 Beyond Object Synchronization: Empowering XR Software Applications

The AuroraXR Application System provides a publish (PUB)/subscribe (SUB) architecture for moving data in, between, and out of XR environments. This system has enabled S&Es to easily connect traditional software systems, data sources, and sensor feeds into XR environments. For example, DEVCOM ARL's "Internet of Battlefield Things" (IoBT) Collaborative Research Alliance explored the AuroraNet framework as a data-sharing fabric for envisioned multisite IoBT infrastructures. This collaboration resulted in work[3] to enable users in AR/MR/VR systems to receive data from, and send commands to, IoBT devices through AuroraXR. The AuroraXR Application System was inspired by these experimental results and has become a core component of AuroraXR.

Figure 2 depicts an example scenario where information is being transferred between multiple applications at two different physical locations. Site A has a Java application that publishes information on Topic A and a Python application that subscribes to information on Topic B. An Aurora Application Proxy sits between these applications and a central Aurora Server, enabling them to push and pull data in its original format. The Aurora Application Proxy encodes data as opaque bytes and all traffic between sites and the AuroraXR Server are encrypted using ECC-25519 (see Section 3.6). If desired, the applications can also be configured with additional end-to-end encryption using AES256-GCM. Using the AuroraXR Server, these applications share data via a unique Application Instance ID, which contains Topics A and B. At the second location, Site B, an XR application uses

the same application instance information to push data over Topic B and pull data from Topic A.

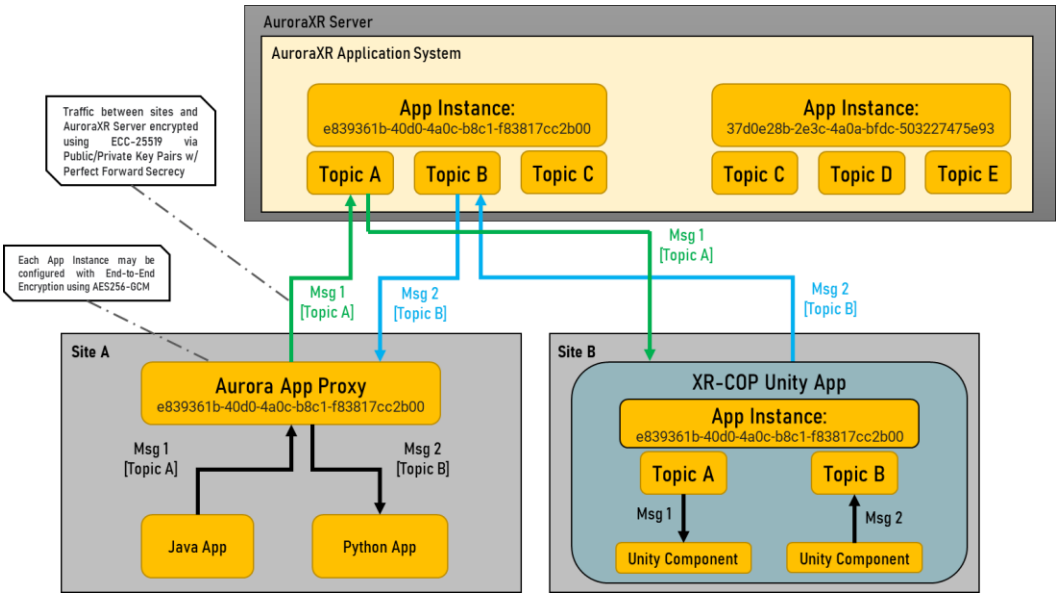

Fig. 2 Example of data moving through the AuroraXR application system

A real-world example of this system being used is pushing camera alert notifications from a Python application into the heads-up display of an AR device. Because the data can simply be pushed in its native format, there is no additional programming required on the camera side of the system, and it is up to the developer of the AR device to parse this information in whatever engine is being used, such as Unity or Unreal.

Thanks to the intelligent implementation of PUB/SUB afforded by ZeroMQ, if there are no subscribers to a particular topic, or even an entire AuroraXR Application Instance, no data will be transmitted by the sender's PUB socket unless there exists a client that has subscribed to that topic. This characteristic is desirable in the constrained network environment. If no one is consuming the data, there is no reason to put it on the network.

3.2 XR Object Metadata System

Initial experiments with the AuroraXR Application System discovered that injecting information into an XR environment in a manner that is disconnected from individual XR objects was insufficient for many use cases. The AuroraXR Metadata System was developed to fill this capability gap.

The metadata system makes it possible to directly associate AuroraXR objects in the virtual environment with data and information specific to those entities. The

metadata system uses a key-value pair framework to imbue XR objects with data, as shown in Fig. 3. The keys are encoded as UTF-8 strings and the values are encoded as opaque binary. The AuroraXR API has tools to automatically serialize and de-serialize Google Protobuf messages assigned as metadata values.

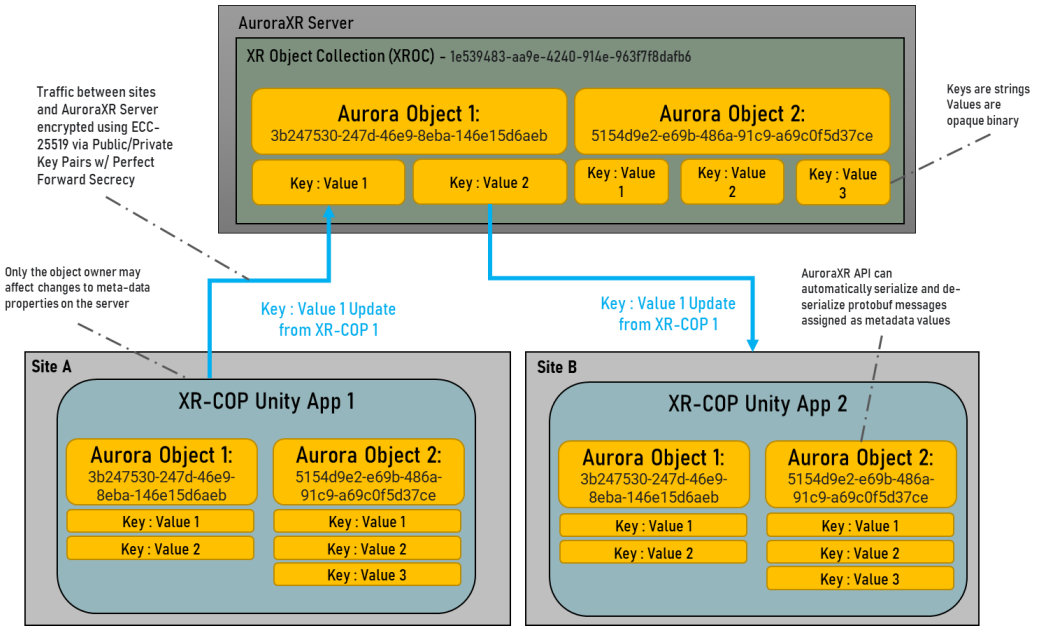

Fig. 3 Example of data moving through the AuroraXR metadata system

As an example, consider an XR object that represents an unmanned aerial system (UAS). The object metadata on the XR object can be used to share information about the UAS including battery level, sensor capabilities, URLs for connecting to video feeds from cameras and sensors on the platform, and other data of interest.

3.3 Virtual and Geospatial Coordinate Systems

AuroraXR provides mechanisms for communication of the positions, orientation, and scale of objects using virtual-world coordinates, geospatial coordinates, or both (Fig. 4). This messaging framework supports latitude and longitude, Universal Transverse Mercator (UTM), Military Grid Reference System (MGRS), and Polar coordinate systems for geospatial information. The messaging schema includes specifiers for the most common geodetic datums, with WGS 84 assumed as the default.

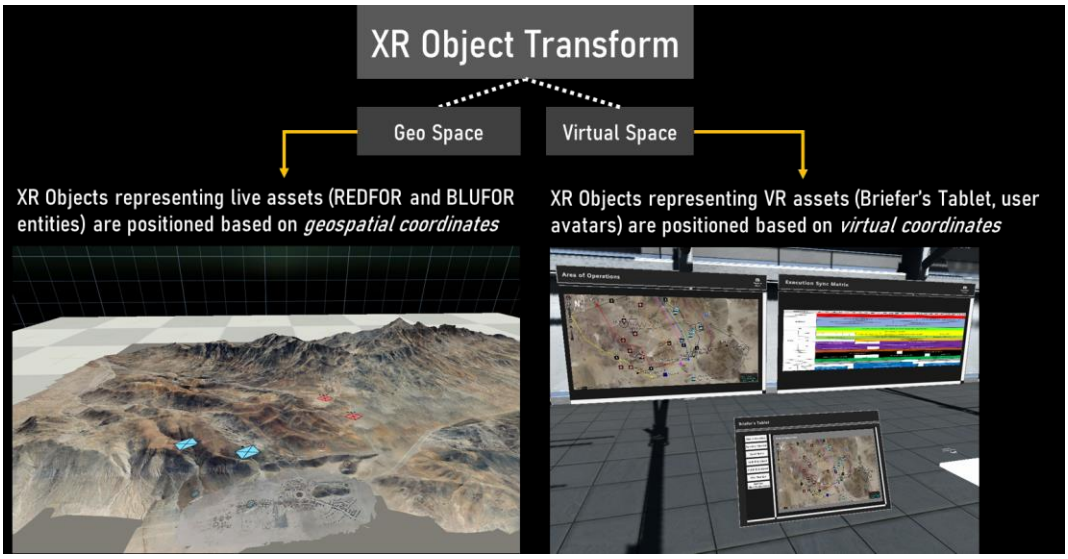

Fig. 4 XR information objects synchronized via geospatial coordinates (left) or virtual coordinates (right)

The architecture provides a geocoordinate adapter framework that allows developers to use the geospatial coordinates on AuroraXR objects to position objects relative to whatever geospatial terrain or mapping system they are using. Currently, a geocoordinate adapter has been implemented that uses the University of Southern California Institute for Creative Technologies' Rapid Integration and Development Environment to enable geospatial positioning of XR objects atop One World Terrain (OWT) models loaded within the Unity engine. OWT models are typically created through photogrammetry and provide high-resolution meshes where each XYZ position on the model is directly attributable to a corresponding latitude, longitude, and altitude.

While geospatial coordinates enable the placement of XR objects relative to a terrain model or map, utilization of virtual-world coordinates allows for synchronization of users and objects in a shared virtual environment. Synchronized virtual objects might include 3-D tablets and displays, user-interface elements, and other interface components that XR users can manipulate inside the shared environment to collaborate—even when distributed over wide geographical distances.

The information about XR object position is synchronized with other AuroraXR applications as fast as possible, or as determined by the developer through configuration of parameters specific to that object. For example, a developer can specify that updates to an avatar representing a user's head should be sent every 0.03 s (Fig. 5). AuroraXR also allows updates to be triggered based on a transform threshold. For example, an object might only trigger an update if it has moved more

than .01 Unity units or if its scale has changed more than 1%. These settings allow precise control over how every object in an application uses precious network resources and is fully configurable during development, or parameters can be changed programmatically during runtime to handle dynamic compute or network conditions.

Fig. 5 Unity interface enabling control over parameters for tuning XR object updates on the network

3.4 XR Object Collections

XR Object Collections (XROCs) provide a means for grouping data from different external sources. For example, this could mean having different XROCs for Cursor-on-Target data originating from a Tactical Assault Kit (TAK) Server, from synthetic data being created by a mission simulator, and from ground robots pushing information through the Robot Operating System. These collections fit well with Unity's hierarchical object grouping convention and allows seamless integration of new objects into existing hierarchies.

3.5 Resiliency and Self-Healing Server Characteristics

The AuroraXR Server consists of a cluster of elements that collaborate to provide a robust, resilient, and high-performance service. Each of these components is designed and implemented to contribute a concise and focused set of functions which, in connection with the other elements, results in a complex capability set in aggregate.

Each element is designed to minimize local state to enable the hand-off of work to other instances of the same type. For example, as the number of users increases, the number of XROC update workers can be increased to maintain a high-quality user experience. Using swarming technology, these components can be distributed across multiple servers, potentially at distributed facilities to ensure survivability and continuity of service in the event that one of the facilities goes offline.

In testing with DEVCOM ARL's BISB, the AuroraXR Server architecture exhibited self-healing characteristics. In instances where server components crashed, new instances would be launched to restore lost functionality and provide continuity of operations.

While there currently exists one key component which, if lost, would result in an unrecoverable condition, work is ongoing to implement a mechanism to provide for a hot standby of this component to eliminate this critical point of failure.

3.6 Encryption and Security

A core focus of the AuroraXR framework is to enable the creation of secure environments where the confidentiality of sensitive information is well protected. To this end, AuroraXR implements cryptographic message authentication and provides two levels of encryption: external encryption and internal encryption. The core elements of these services are highlighted in Fig. 6 and detailed in the following sections.

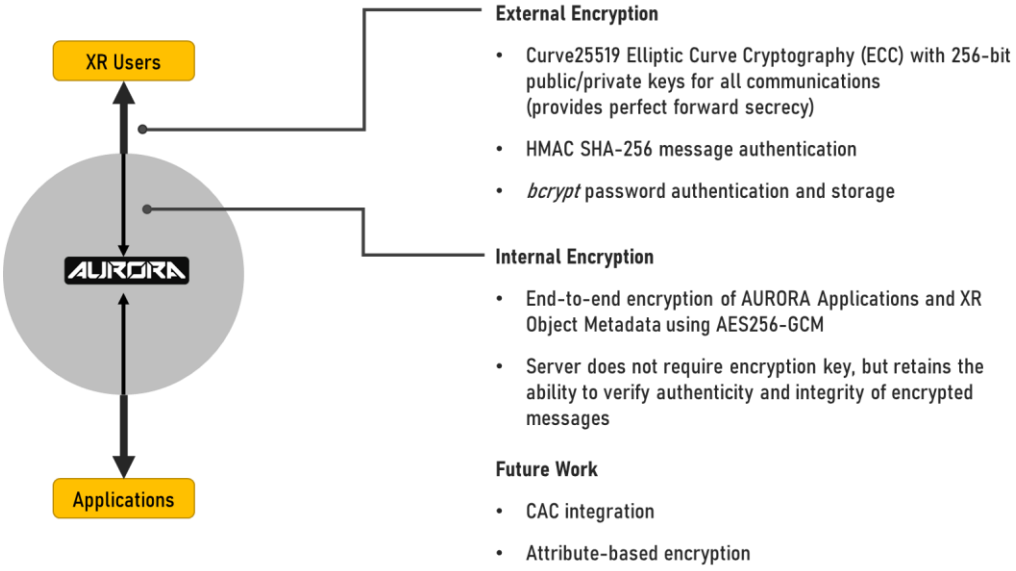

Fig. 6 The defense in-depth model of system security features in AuroraXR

3.6.1 Cryptographic Message Authentication

Every client message entering an AuroraXR Server undergoes SHA256 hash-based message authentication (HMAC) to ensure the authenticity of each message before it is processed. Any message received having an invalid HMAC is silently discarded. The secret data used to authenticate the messages sent by each client is rotated periodically, thus minimizing the chance an attacker can spoof valid messages.

3.6.2 External Encryption

AuroraNet leverages CurveMQ, ZeroMQ's implementation of Elliptic Curve Cryptography (ECC), to secure all information transported between clients.

CurveMQ uses the Curve25519 elliptic curve with 256-bit public–private keys. Only clients who possess a copy of the server's public key can initiate a connection request. The server will only accept connections from clients whose public keys are registered in a collection of authorized users and all other connections are rejected.

The CurveMQ implementation incorporates perfect forward secrecy, which provides robust protection that precludes attackers from decrypting recorded network traffic if they succeed in compromising either a client's or server's private key. Successful message decryption would require the attacker to obtain the private keys of both sides of the message exchange.

ECC is very successful in providing strong encryption for low-power devices like cell phones. The lower computation requirements lend themselves very well to use in XR systems as the reduced CPU load minimizes the impact to display framerates and improves battery usage for untethered devices, like the HoloLens 2.

While ECC is used extensively in commercial applications, adoption by the DOD has been slow. No ECC ciphers have been approved under FIPS 140-2. In late 2019, the National Institute of Standards and Technology (NIST) submitted a Request for Comments to the Federal Register for drafts of FIPS 186-5 and NIST SP 800-186, which seek to approve Curve25519 for use in protecting US Government data and communications. Buoyed by this effort, CurveMQ has remained the external encryption mechanism for all communication between AuroraXR clients and servers.

3.6.3 Internal Encryption

In addition to the external encryption described in Section 3.6.2, AuroraXR offers the ability to further protect application system and metadata system information. The AuroraXR messaging schema provides the transport of information using a wide range of ciphers with the internal encryption mechanism.

AuroraXR currently supports hardware-accelerated, AES256-GCM encryption. The C# library previously used to implement AES256-GCM dropped support for the .NET Framework used in development with the Unity engine. Consequently, C# and Unity-based systems are unable to support internal encryption at this time. Fortunately, Unity3D has recently updated the .NET version supported by its long-term support platform to work with newer versions of the .NET Framework. While this does not restore availability of the previous encryption library, it does mean the official Microsoft .NET libraries that support AES256-GCM are available for the first time in the Unity engine. Transition to these Microsoft libraries is expected in a future release of AuroraXR. It also lays a clear path for FIPS 140-2 compliance of internal encryption for .NET and Unity clients.

Using internal encryption means that AuroraXR can support end-to-end encryption where only the sender and receiver can access the data. While the server is unable to view the message contents in this configuration, it is still able to ensure the authenticity of these messages via the HMAC checks described in Section 3.6.1. Internal encryption is currently supported in the application system and implementation is nearing completion in the metadata system.

3.7 AuroraXR Tactical Data Manager

The AuroraXR Tactical Data Manager (ATDM) was created to enable seamless interoperability with systems and sensors that exist in the real (or simulated) world and whose data need to be processed for visualization in XR systems and synchronized across multiple XR clients. As shown in Fig. 7, the ATDM receives AuroraXR Application Data from the AuroraXR Server or Cursor-on-Target data directly from a TAK Server. The ATDM keeps an internal dictionary of each information object associated with an external system or sensor. If a message comes in that matches an existing object's unique ID, the ATDM will update the information for that object, and if a unique ID does not exist in the dictionary, the ATDM will tell the Aurora Server to spawn a new XR object to represent that entity in 3-D space. These XR objects can be assigned to particular XROCs for easier management of sources across multiple networks. The coupling between the ATDM and the AuroraXR Server provides a highly efficient framework for synchronizing information from multiple objects and maintaining a shared virtual world model across multiple users.

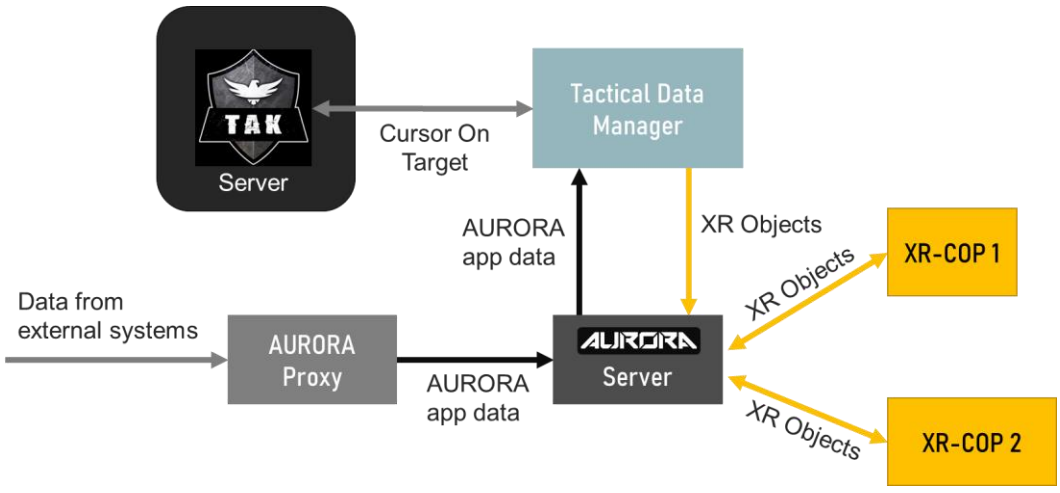

Fig. 7 Flow of data from external systems, including the TAK Server, into the AuroraXR ecosystem

4. Conclusion and Next Steps

4.1 In Pursuit of Authority to Operate (ATO)

From its inception, AuroraXR has been designed and developed with the ultimate goal of achieving ATO on DOD systems. While this is a complex and arduous process, the careful consideration of each design decision has been made with this objective in mind.

In the future, Stormfish Scientific Corporation intends to pursue a FEDRAMP accreditation as a commercial offering that will allow US Government customers and their contractors to accelerate obtaining an ATO on systems built atop AuroraXR by inheriting from this base FEDRAMP accreditation. They can then focus on implementing the remaining controls as appropriate for their specific systems and use cases. While the US Government and their contractors using AuroraXR for government purposes will enjoy the freedom to pursue their own ATOs without using this service, Stormfish Scientific Corporation believes the licensing a FEDRAMP-certified version of the AuroraXR will provide a strong return on investment and maximize efficiencies for those users who chose to inherit from a foundational FEDRAMP ATO that is under continuous maintenance, whether hosted on-premises or in various GovCloud services.

4.2 A Common XR Data Fabric Made Possible Through Government Purpose Rights (GPR)

AuroraXR was developed in collaboration with the DEVCOM ARL's BISB with the US Government receiving limited GPR. This frees the Army and other government entities from the encumbrance of ongoing licensing costs for this technology. As a result, AuroraXR provides a royalty-free mechanism for implementation of a common, cross-reality data fabric enabling the potential for ubiquitous interoperability between XR systems throughout the US military and other supporting agencies.

5. References

1. ZeroMQ getting started guide. The ZeroMQ authors; 2022. [accessed 2022 Aug 3] https://zeromq.org/get-started/.

2. Barroso VC, Fuchs U, Wegrzynek A. Benchmarking message queue libraries and network technologies to transport large data volume in the ALICE O system. 2016 IEEE-NPSS Real Time Conference (RT). 2016 June 6–10; Padua, Italy. IEEE; c2016. p. 1–5. https://ieeexplore.ieee.org/document/7543162.

3. Trout TT, Risinger E, Raber T, Dennison M Jr. An intelligent information mediation framework to enable decentralized decision-making in immersive environments. Proc SPIE 11759, Virtual, Augmented, and Mixed Reality (XR) Technology for Multi-Domain Operations II; 2021 May 6. SPIE; c2021. p. 151–158. Vol. 11759.

List of Symbols, Abbreviations, and Acronyms

3-D	three-dimensional
API	Application Programming Interface
AR	augmented reality
ARL	Army Research Laboratory
ATO	Authority to Operate
ATDM	AuroraXR Tactical Data Manager
AURORA	Accelerated User Reasoning: Operations, Research, and Analysis
BIPB	Battlefield Information Processing Branch
BISB	Battlefield Information Systems Branch
CPU	central processing unit
DEVCOM	US Army Combat Capabilities Development Command
DOD	US Department of Defense
ECC	Elliptic Curve Cryptography
FIPS	Federal Information Processing Standard
GPR	Government Purpose Rights
HMAC	hash-based message authentication
HMD	head-mounted display
ID	identification
IoBT	Internet of Battlefield Things
MGRS	Military Grid Reference System
MR	mixed reality
NIST	National Institute of Standards and Technology
OWT	One World Terrain
PUB	publish
S&Es	scientists and engineers

SUB	subscribe
TAK	Tactical Assault Kit
UAS	unmanned aerial system
URL	Uniform Resource Locator
UTM	Universal Transverse Mercator
VR	virtual reality
XR	cross-reality
XROC	XR Object Collection

1 (PDF)	DEFENSE TECHNICAL INFORMATION CTR DTIC OCA
1 (PDF)	DEVCOM ARL FCDD RLD DCI TECH LIB
2 (PDF)	DEVCOM ARL FCDD RLC IB M DENNISON T TROUT