# Design of a ZeroMQ-Based Communication Framework for New-Generation Telescope Automatic Control Systems (Postprint)

**Authors:** 邓辉, Zhong Wenjie, Fu Yingxue, Wang Feng, Wei Shoulin

**Date:** 2018-05-15T00:00:00+00:00

## Abstract

With the advancement of astronomical technology, the components of astronomical telescope systems are becoming increasingly complex, and the automatic control system has become a core component for routine telescope observations. Typically, executing a complete observation plan requires cooperation and coordinated operation among different devices; therefore, an efficient underlying communication framework is critical to the success of telescope automatic control systems. ZeroMQ is a high-performance network communication library that provides multiple fundamental communication models for building complex distributed applications, making it highly suitable for distributed astronomical telescope observation control scenarios that require multiple communication patterns and low latency. This paper reviews network technologies widely used in telescope control systems, such as CORBA, DCOM, and native Socket, presents the overall design of a next-generation telescope automatic control system communication framework based on ZeroMQ, and discusses solutions for key technologies including socket design, message model design, and serialization.

## Full Text

## The Design of Communication Framework for a New Generation of Telescope Autonomous Control System Based on ZeroMQ

**Deng Hui[1][2], Zhong Wenjie[1], Fu Yingxue[2], Wang Feng[1][2], Wei Shoulin[2]**

[1]Center for Astrophysics, Guangzhou University, Guangzhou 510006, China
[2]Computer Technology Application Key Lab of Yunnan Province, Kunming University of Science and Technology, Kunming 650500, China

## Abstract

With the advancement of astronomical technology, telescope system components are becoming increasingly complex, and the telescope automatic control system has become a core component for routine observations. Executing a complete observation plan typically requires coordination and collaboration among different devices, making an efficient underlying communication framework critical to the success of the telescope automatic control system. ZeroMQ is a high-performance network communication library that provides various fundamental communication models for building complex distributed programs, making it ideally suited for distributed telescope observation and control systems that require multiple communication modes and low latency. This paper reviews network technologies widely used in telescope control systems, including CORBA, DCOM, and native Socket, presents the overall design of a communication framework for a new generation of telescope automatic control system based on ZeroMQ, and discusses key technical solutions for socket design, message modeling, and serialization.

**Keywords:** Observation Control System; Message Communication; Autonomous Observation; Modular Design; System Framework

## 1. Communication Technologies

The physically distributed nature of various devices in telescope systems necessitates distributed control capabilities in observation control systems. Both open-source solutions like RTS2 and control systems for large telescopes domestically and internationally have adopted distributed control principles, primarily utilizing communication protocols including raw sockets, HTTP, CORBA, and DCOM. While these technologies played important roles at different stages, they exhibit various limitations as communication technology evolves.

RTS2 is a widely used open-source remote telescope control system that supports the integration and control of multiple devices, though it is less commonly employed in large telescope control systems. RTS2 uses raw TCP socket programming, which requires each terminal to maintain all its communication relationships. In telescope systems with multiple devices, services, and communication terminals—along with support for multi-client connections—each participating client must maintain a substantial number of connections, complicating program development and implementation. Furthermore, telescope control systems frequently require synchronous operations across multiple devices, such as sequential observations, combined observations, and synchronized exposures with multiple CCDs. In RTS2, a client establishes independent network connections to multiple devices simultaneously, requiring commands to be sent sequentially through loops. Although the plain text protocol is compact and network load is typically low due to limited terminals, this approach does not constitute true broadcast communication and inadequately addresses collaborative control challenges.

CORBA, an early standard for distributed computing and remote object invocation, has been widely adopted in astronomical telescope control systems. The ALMA Control System (ACS) represents a notable example, supporting component and container model development. CORBA' s advantage lies in its ability to integrate different development technologies: Java for high-level control system development, C++ for time-critical low-level development, and Python for astronomer data queries and analysis. The underlying communication for China' s LAMOST control system is also based on CORBA.

ASCOM is a telescope observation control system for the Windows platform that employs Microsoft' s COM component programming model, offering good extensibility. Due to its short development cycle and easy deployment, ASCOM is suitable for remote control system development in site selection telescopes. The 2.4-meter telescope at Lijiang has implemented CCD filter and dome control based on ASCOM. However, since many telescope device drivers and data processing programs operate in Linux environments while ASCOM requires Windows, this creates significant challenges for device integration and data processing systems.

The rapid development of message-oriented middleware has provided robust technical support for information exchange and control in distributed heterogeneous environments. ActiveMQ and RabbitMQ, with their loosely coupled characteristics, are well-suited for distributed device control. ActiveMQ requires a Java environment but supports multi-language clients. The Observation Control System (OCS) of the Telescopio Nazionale Galileo (TNG) is based on ActiveMQ, supporting concurrent and asynchronous communication modes for device control and leveraging ActiveMQ's built-in master-slave mode to enhance message hub availability. However, this architecture introduces third-party systems, making the stability of the message middleware a prerequisite for stable system operation.

With the rapid development of computer hardware, software, and automation technologies, these protocols and technologies demonstrate limitations at different levels and cannot fully meet the requirements of observation control systems. In recent years, cloud computing and distributed computing technologies have made significant progress, with new communication technologies continuously emerging. ZeroMQ, with its open-source, cross-platform, and high-performance characteristics, has been widely applied in distributed data processing frameworks such as the stream computing framework Storm and the distributed computing framework DALiuGE. ZeroMQ is also suitable for measurement and control systems, such as intelligent building monitoring. Unlike message middleware like ActiveMQ and RabbitMQ that require dedicated servers during deployment, ZeroMQ only needs to be referenced as a library in applications to enable communication. More importantly, ZeroMQ supports multiple communication patterns including Request-Reply, Publish-Subscribe, Push-Pull, and Router-Dealer, which can be flexibly combined to build more complex distributed network communication frameworks that meet the diverse

communication model requirements of telescope control systems. Additionally, ZeroMQ provides language bindings for C, C++, Java, .NET, Python, and supports multiple mainstream operating systems, providing technical support for collaborative control of heterogeneous devices in telescope systems.

## 2. System Architecture

Drawing from the ATST architecture design, the telescope remote control system is divided into five components: Observation Control System (OCS), Telescope Control System (TCS), Instrument Control System (ICS), Data Handling System (DHS), and User Interface (UI). The system structure is shown in [Figure 1: see original paper].

The Observation Control System serves as the core of the entire system, encompassing observation target maintenance, observation plan management, and observation script parsing and execution scheduling. The Instrument Control System and Telescope Control System receive execution commands from the Observation Control System and are responsible for direct control of the telescope and other devices (such as CCDs and domes). After observation data is generated, the system sends data-ready messages to the Data Handling System, which initiates real-time data processing programs. The results of real-time data processing support observation scheduling decisions to select optimal observation targets. All interactions between subsystems employ ZeroMQ. Notably, a WebServer component is added to the Observation Control System to provide HTTP-based REST services and WebSocket for real-time communication with the UI. Using the universal, platform-independent HTTP protocol enables the construction of cross-platform interfaces that support various frontend programming technologies.

## 3. Key Technologies

In early ZeroMQ versions, the API was based on the AMQP exchange and queue model. The author rewrote the API in 2009 to adopt the BSD Socket API, reducing the learning curve and facilitating integration with existing technologies. ZeroMQ objects are exposed as "sockets," making socket design the primary consideration in ZeroMQ programming. Since sockets can only transmit fixed-length binary data blocks, a message model must be defined for mutual understanding among distributed components. Designed as lightweight message-oriented middleware focused on message transmission, ZeroMQ lacks message server storage and forwarding capabilities, and therefore does not support message persistence or crash recovery. Consequently, appropriate high-availability mechanisms must be designed.

### 3.1 Socket Design

ZeroMQ programs communicate through "sockets" that resemble TCP sockets but can handle communication with multiple peers, similar to unbound UDP

sockets. Since telescope control systems fundamentally control devices, both device control systems and telescope control systems can be abstracted to a device layer. This introduces the Steward component, which must support point-to-point, point-to-multipoint, and multipoint-to-multipoint data event-driven control and processing requirements between devices and components. Additionally, real-time status monitoring, exception recovery, unified timing control, and availability detection must be considered. The Observation Control System receives user observation plans and ultimately sends operation commands to devices, functioning as a client to both the Telescope Control System and Instrument Control System. The client must operate devices and receive broadcast updates, but does not connect directly to devices—instead, it connects through the Steward. Therefore, the Steward must create sockets for receiving request commands and broadcasting information for clients. Similarly, devices must receive commands from clients and broadcast information, requiring the Steward to create corresponding sockets for device communication.

[Figure 2: see original paper] illustrates the ZeroMQ socket design within the telescope control system and instrument control system. After a client connects its DEALER socket to the Steward' s client-dedicated ROUTER socket, it receives connection information from the Steward' s PUB interface and subscribes to broadcast messages using a SUB socket. Devices connect to the Steward' s device-dedicated ROUTER socket to report parameters, send periodic heartbeats, and obtain the Steward' s PUB address and port for receiving broadcast messages. For single-device operations, the client connects to the Steward, retrieves device address information, and sends operation commands. For concurrent control of multiple devices, broadcast messages can be sent through the Steward' s PUB interface.

### 3.2 Message Model

ZeroMQ supports multi-part messages that store multiple message frames within a single message. This feature is utilized to format messages for control purposes. ZeroMQ sockets are transient by default, and the connected socket (such as ROUTER) generates a UUID to identify the message source and associate it accordingly. The ROUTER socket prepends the message source address to all received messages. To enable clients to operate specific devices, a fixed identifier is required. Each device has a fixed name specified at startup through parameters, which also serves as the identifier for the device' s ROUTER socket. The message model for client-sent commands to devices is defined in [Figure 3: see original paper].

In [Figure 3: see original paper], Client_Id is an automatically generated unique identifier; Device_Id is a combination of [deviceType] + deviceName, such as [CCD]Camera1; FromClient and ToDevice indicate commands originating from the client and sent to the device; Devices lists the target devices with multiple names separated by commas; Command_Group categorizes commands for subsequent message processing, as detailed in .

**TABLE:1** Command Message Classification

| Command Group | Description |
|---|---|
| READY | Device startup, preparation complete |
| REQUEST | Client request message |
| REPLY | Steward' s response message |
| HEARTBEAT | Heartbeat message |
| DISCONNECT | Disconnection message |
| STEWARD_PORT | Steward' s PUB port information |

After startup, devices send registration information to the Steward (Command_Group: READY). Upon receiving client operation commands, devices execute operations and send result messages to clients (Command_Group: REPLY). Additionally, devices periodically send heartbeat messages (Command_Group: HEARTBEAT) and broadcast information (Command_Group: PUB). The message model is shown in [Figure 4: see original paper].

### 3.3 Compression and Serialization

Data compression during network communication reduces bandwidth requirements, which is particularly crucial for observatories typically located at high altitudes where network bandwidth is precious. The LZ4 compression algorithm is employed to compress data before transmission and decompress it at the receiving end. LZ4 is a lossless compression algorithm characterized by high-speed decompression and multi-language programming interface support. ZeroMQ, designed as lightweight message-oriented middleware focused on message transmission, lacks message server storage and forwarding capabilities and therefore does not support message persistence or crash recovery. When developing network applications, program data often needs to be stored in memory and transmitted to another computer on the network. The process of converting program data into a storable and transmittable format is called "serialization," with the reverse process called "deserialization." ZeroMQ only addresses network connectivity and does not provide serialization methods. While objects can be forcibly cast to char* or void* types for transmission, this requires writing different code for each class object, resulting in substantial workload, poor generality, and potential CPU endianness issues. Therefore, an appropriate serialization library with minimal programming intrusion is needed. MsgPack is selected—a binary object serialization library with cross-language characteristics that is easy to use. Operation commands and update messages are packed using MsgPack, written into ZeroMQ' s message structure (zmq::message_t), transmitted through ZeroMQ, and finally unpacked by the receiver using MsgPack for command analysis and data retrieval.

## 4. High Availability

The system architecture demonstrates that the Steward component is the core of the entire system. To avoid single-point failure on the Steward, a master-slave hot standby mechanism is implemented. When devices lose heartbeat connection with the master Steward node, they switch to the slave node. At any given time, one node serves as the master, receiving all client and device connection requests, while the other operates as a standby. The master Steward periodically broadcasts device connection information to slave Stewards via the PUB socket, ensuring that clients connecting to the slave Steward can immediately obtain device information if the master fails.

Devices are the core of telescope system control, making device availability detection critical in telescope control systems. Availability detection involves periodic checks to determine terminal availability and automatically remove non-functional devices—a mechanism known as heartbeat in distributed systems. The heartbeat mechanism is only implemented between devices and the Steward; clients, which only connect during operations, do not require maintained connections and thus do not use heartbeat mechanisms. In ZeroMQ programming, clients, Stewards, and devices must all read from multiple sockets, using non-blocking zmq_poll() in the main loop and a separate timer to trigger heartbeats. Using the main loop to control heartbeat transmission is avoided because it would either send excessive heartbeats (blocking the network) or too few (causing nodes to disconnect and generating false failures). Heartbeat information is passed asynchronously between components, allowing any component to determine that its peer has failed and terminate communication. The heartbeat frequency is configurable via files or startup parameters and maintains consistent frequency across all nodes.

## 5. Conclusion

The telescope observation control system is a vital component of astronomical telescope systems. Actual telescope operations face demands for observation terminal equipment updates and increasingly complex observation processes. Developing an efficient and extensible telescope automatic control system that enables rapid integration of new equipment and system upgrades with minimal cost can effectively reduce observation difficulty and improve observation efficiency. The normal operation of telescope systems depends on coordination among multiple devices and components, making network communication a critical underlying support for observation control systems. This paper analyzes and compares traditional communication mechanisms in astronomical control software such as CORBA, DCOM, and raw sockets, investigates the implementation of point-to-point and point-to-multipoint data event-driven communication using ZeroMQ, introduces ZeroMQ socket design, message model design, and compression/serialization technologies, and studies high-availability assurance for normal system operation under network constraints and device failures. Future work will focus on implementing observation plan and command conversion,

terminal device control communication, and HTTP protocol-based interfaces between the observation control system and user interface for actual telescope driving and observation control.

## References

[1] Kubánek P, Jelínek M, French J, et al. The RTS2 protocol[C]// Proceedings of SPIE. 2008.
[2] Farris A, Marson R, Kern J. The ALMA telescope control system[C]//10th ICALEPCS International Conference on Accelerator & Large Expt. Physics Control Systems. 2005.
[3] Wang Jian, Jin Ge, Huang Kun, et al. Design and implementation of LAMOST OCS message bus[J]. Nuclear Electronics & Detection Technology, 2006, 26(3): 268-271.
[4] He Shousheng, Fan Yufeng, Wang Chuanjun. An automatic CCD observation system based on the ASCOM standard[J]. Astronomical Research & Technology—Publications of National Astronomical Observatories of China, 2013, 10(4): 386-391.
[5] He Shousheng, Xin Yuxin, Lun Baoli, et al. Astronomical dome control system based on ASCOM and Modbus/TCP standard[J]. Astronomical Research & Technology, 2017, 14(3): 356-362.
[6] Guerra J, San Juan J, Lodi M, et al. OCS: towards a more efficient telescope[C]// Proceedings of SPIE. 2014.
[7] Toshniwal A, Taneja S, Shukla A, et al. Storm@ twitter[C]//Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data. 2014: 147-156.
[8] Wu C, Tobar R, Vinsen K, et al. DALiuGE: a graph execution framework for harnessing the astronomical data deluge[J]. Astronomy and Computing, 2017, 20: 1-15.
[9] Ye Song, Yao Jiangdong. Design for message communication architecture of distributed measurement and control system[J]. Modern Electronics Technique, 2014(2): 105-109.
[10] Goodrich B D, Wampler S B. Software controls for the ATST Solar Telescope[C]// Proceedings of SPIE. 2004.

*Note: Figure translations are in progress. See original paper for figures.*