# XOM Prototype Overview

**Dennis M. Moen**
**C3I Center**
**George Mason University**
**dmoen@gmu.edu**

## Protocol Description

This document provides a brief description of the current prototype of the Extensible Modeling and Simulation Framework [1] Overlay Multicast (XOM). The XOM is an overlay multicast protocol designed to support many-to-many multicast for real-time distributed visual simulations. The objective is to provide multicast service over a unicast network environment using UDP. From the multicast sender and receiver's point of view, each XOM Relay (XOMR)looks like an IP layer multicast router.

XOMR performs as a multicast "relay agent" for any application located on the same subnet as an instance of the XOMR as indicated in Figure 1. For each subnet that participates in the multicast group communication, there must be a host running a XOMR on the subnet. The XOMR listens on a set of multicast addresses to the local LAN for each multicast message generated within its local subnet and forwards this traffic to the downstream XOMRs according to its multicast tree, by using unicast. Figure 2 presents the concept and indicates the concept of group aggregation efficiency gains by using the overlay multicast. The partner XOMR will then multicast the packet to the destination local LAN, and keep on forwarding the packet to other XOMRs if necessary.

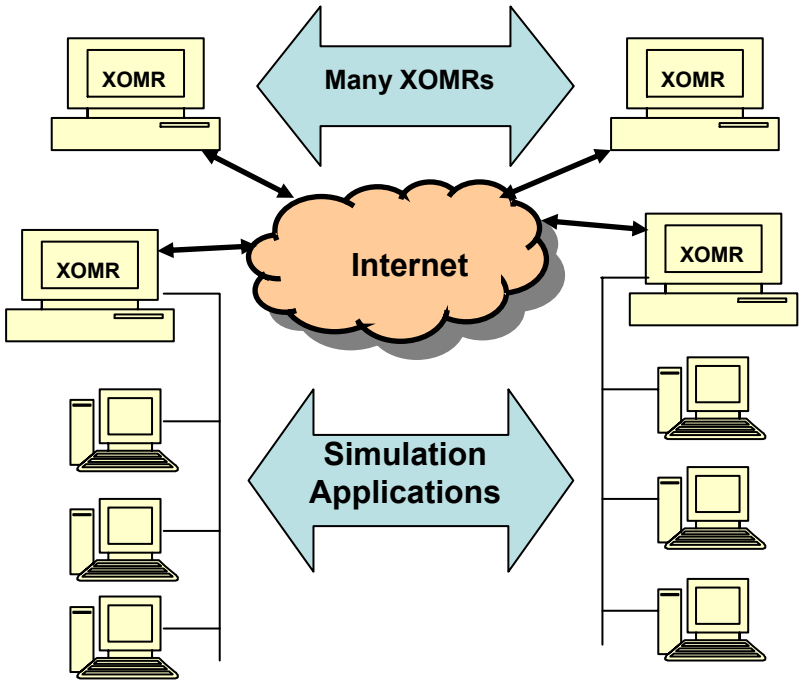



**Figure 1. XOM Service on a Subnet**

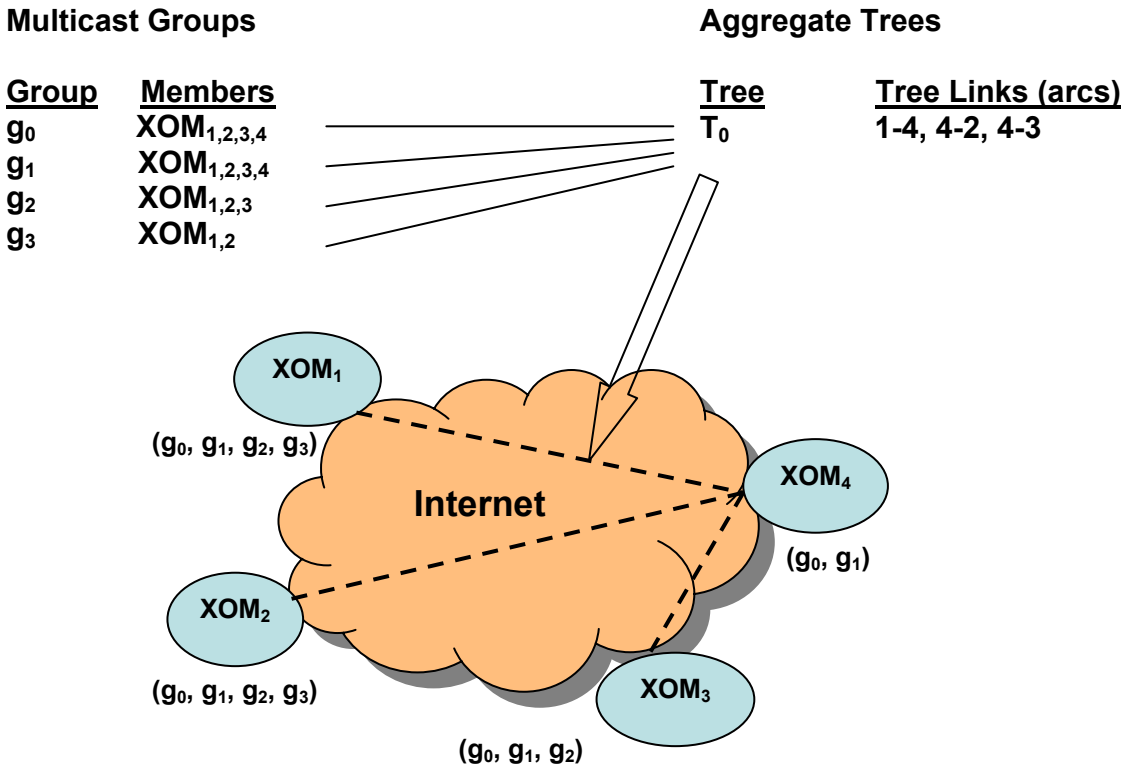



**Figure 2. Group Aggregation**

Key elements of the current XOMR prototype:

- Group Management
  The group addresses and UDP port used by XOMR are specified as command line arguments. IGMP is not implemented in this version of code
- Partner Discovery
  The list of partner XOMRs IP addresses is specified as command line arguments.
- Network Measurement
  XOMR uses aggregate delay as the routing metric. A virtual mesh composed of all the XOMRs is established. This mesh is a concept of the transport layer. The current version establishes a full mesh, and every XOMR maintains this mesh topology in a matrix that also includes the measured network delay between XOMRs. Each XOMR periodically sends an echo request message to all partner XOMRs it knows. Each XOMR will echo messages received from other XOMRs which use the response to measure round trip times RTT) as the measure for delay. Periodically, each XOMR will send its measured RTT values to all its XOMR partners. Thus, every XOMR keeps a delay matrix of RTT value between every two XOMRs.

- Multicast Forwarding Tree calculation
  Periodically, each XOMR calculates a source specific multicast tree according to its delay matrix. The tree is rooted at the source XOMR. The prototype uses the Shortest Path First (SPF) tree algorithm.
- Routing Update
  Periodically, each XOMR will forward its multicast tree to all other partner XOMRs. Each XOMR will keep *n* multicast trees, where *n* equals to the total number of XOMRs.
- Message replicating and Forwarding
  When an XOMR receives a multicast data message from its local LAN, it will encapsulate it, replicate it, and forward it to the downstream neighbors according to its own multicast tree. When an XOMR receives a UDP data message from another XOMR, it will check to see from which source XOMR the UDP message was originally generated (but not the upstream neighbor). It then checks the locally stored multicast trees for that source XOMR to replicate the packet and forward it to other downstream XOMRs, if any. Meanwhile, the XOMR will de-capsulate the UDP message and multicast it to its local LAN.

## Software Design

The XOM was first implemented in JAVA. Then, to improve the performance, the message replicating and forwarding modules were implemented in C++. The control module remains implemented in Java. The JAVA and C++ modules were integrated by using Java JNI callback.

Figure 3. presents the design structure of the JAVA version XOM implementation. The JAVA version is composed of the following four major classes:

- *public class XOM*; This is the main class and creates all the threads and initializes the data structures. This class also provides general methods and auxiliary functions, such as network layer address format conversion, encoding and decoding, searching, etc.
- *public class XOMMulticastRouter extends Object implements Runnable;*This is the forwarding engine. This class replicates and forwards the packets. It listens to both the local LAN and other XOMs. It stores the forwarding table in its member variable *XOMMulticastRouter.forwardingTables:* The forwarding table is periodically updated by class *XOMRoutingTable*.
- *public class XOMRoutingTable implements Runnable;*This class performs the real routing work. It processes the delay information and multicast tree information received from other XOMs. It then calculates its own multicast tree according to this information, and updates the forwarding table in class *XOMMulticastRouter*:
- *public class XOMStatistics implements Runnable;* This class measures the RTT values to all the other XOMs. It also does the statistics work, such as total number of packets sent and received.

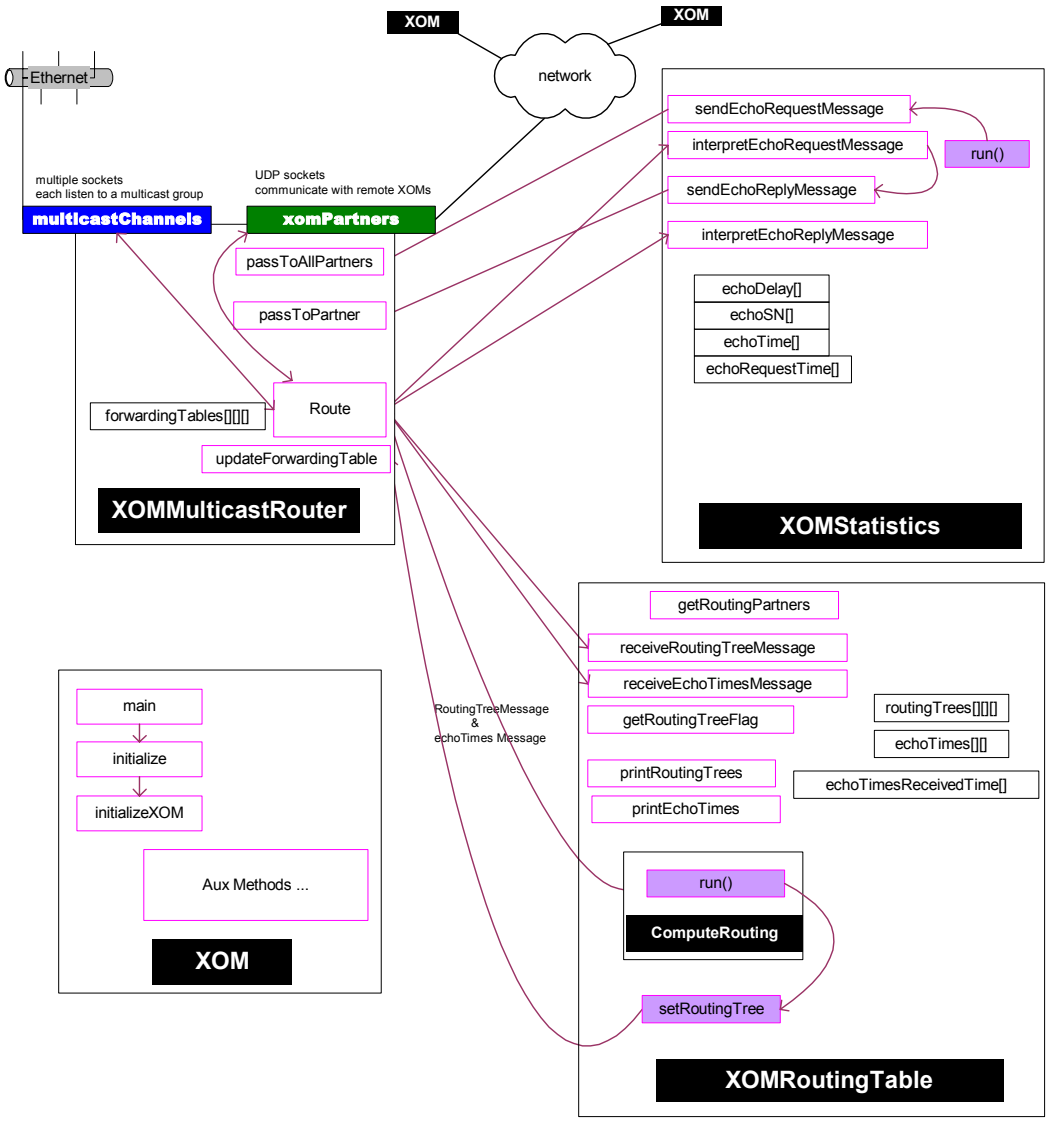



Figure 3. JAVA XOM Implementation

The C++ version code (Figure 4.) implements the *XOMMulticastRouter* class in C++. The other three classes are still in JAVA. The C++ module communicates with the JAVA module by using JNI (Java Native Interface). All the data packets are processed within the C++ module, which includes sending/receiving, encapsulation/de-capsulation, and forwarding. All the control messages are passed to the JAVA module.
The C++ module keeps a forwarding table, which is updated by the JAVA module. The following diagram shows the C++ version design structure:

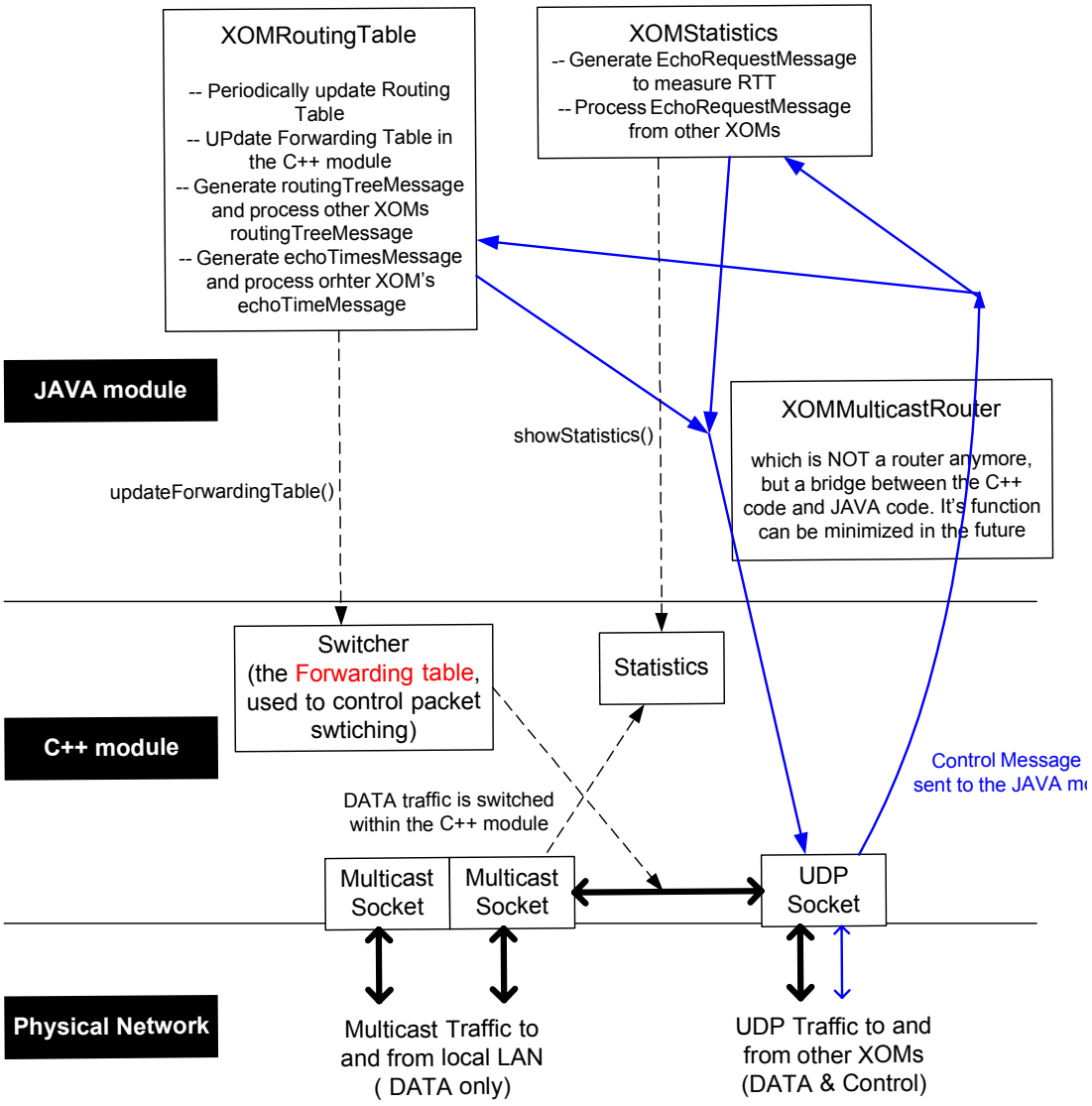



Figure 4. C++ Version of XOM

## Client Interface Specification

The client refers to the multicast sender and receiver. XOM provides a transparent service to the client. The client can simply treat XOM as an IP layer multicast router. Currently, the XOM does not speak IGMP, however, IGMP is planned for future versions.

The UDP port used by XOM for communications other is specified by XOM.XOM_PORT. Currently, it is default set to UDP port 4785.

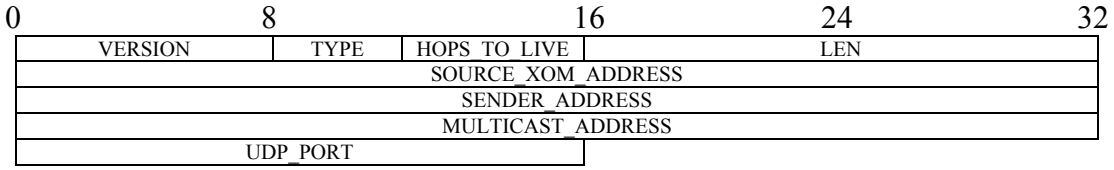
- Command Line Arguments

*java XOM <registryAddress> <numberOfMulticastGroups> <numberOfPortsPerGroup> <lowestMCAddress> <lowestPort> <routingUpdateInterval> <thisSubnetMaskBits> <debugIndexThisXOM> <useTCP> [partnerXOMAddress1, partnerXOMAddress2, ...]*

| | |
|---|---|
| registryAddress | - InetAddress of registry, 0 if non (currently must be 0) |
| numberOfMulticastGroups | - count of groups/ports we will support |
| lowestMCAddress | - first group address to multicast from the subnet, dotted decimal notation (other addresses follow in sequence) |
| lowestPort | - first UDP port to multicast (each address will get one port in sequence) |
| routingUpdateInterval | - time in ms between routing updates (default 10 s) |
| thisSubnetMaskBits | - number of bits used for routing in subnet address (default 24) |
| debugIndexThisXOM | - 0 for operation |
| useTCP | - 0 for UDP tunnels, 1 for TCP tunnels (currently must be set to 0) |
| partnerXOMAddresss | - zero to MAX_PARTNERS addresses, in dotted decimal format, to be used as partners without checking the registry |

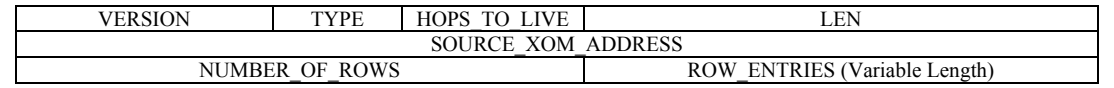
## Message Format

- DATA:




VERSION: 8 bits, 0x 2
TYPE: 4 bits, 0x 1
HOPS_TO_LIVE: 4 bits, the number of hops can be relayed by XOMs
LEN: 16bits, the packet length including the header
SOURCE_XOM_ADDRESSS: 32bits, the IPv4 address of the XOM that generate this UDP packet
SENDER_ADDRESS: 32bits, the IPv4 address of the end host that generate the multicast packet
MULTICAST_ADDRESS: 32bits, the multicast address used by the end host
UDP_PORT: 16 bits, the UDP port used by the end host

- ROUTING_TREE:

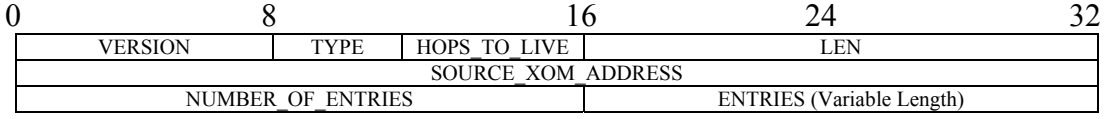



For Each ROW_ENTRY:

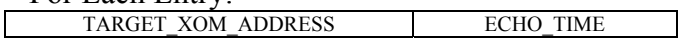

For Each COLUMN_ENTRY:


VERSION: 8 bits, 0x 2
TYPE: 4 bits, 0x 2
HOPS_TO_LIVE: 4 bits, the number of hops can be relayed by XOMs
LEN: 16bits, the packet length including the header
SOURCE_XOM_ADDRESSS: 32bits, the IPv4 address of the XOM that generate this routing information
NUMBER_OF_ROWS: 16bits, the number of rows of the routing_tree matrix
ROW_ENTRIES: variable lengths, may be composed of multiple ROW_ENTRYs
PARTNER_XOM_ADDRESS: 32 bits, the IPv4 address of the XOM corresponding to the row
NUMBER_OF_COLLUMNS: 16 bits, number of downstream partner XOMs
COLUMN_ENTRIES: variable length, may be composed of multiple TARGET_XOM_ADDRESS
TARGET_XOM_ADDRESS: 32 bits, the IPv4 address of the downstream XOM

- ECHO_TIMES:

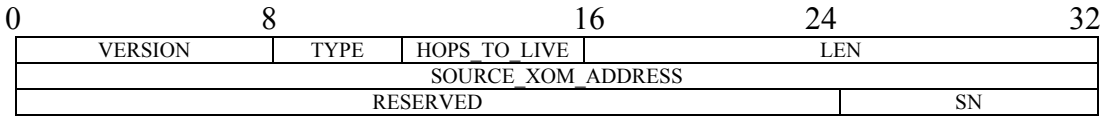



For Each Entry:



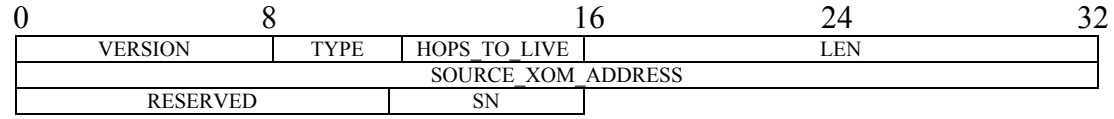

VERSION: 8 bits, 0x 2
TYPE: 4 bits, 0x 3
HOPS_TO_LIVE: 4 bits, the number of hops can be relayed by XOMs
LEN: 16bits, the packet length including the header
SOURCE_XOM_ADDRESSS: 32bits, the IPv4 address of the XOM that generate this echo time list
NUMBER_OF_ENTRIES: 16 bits, the number of entries in the echo time list
ENTRIES: variable length, may be composed of multiple entries
TARGET_XOM_ADDRESS: the IPv4 address of the target XOM
ECHO_TIME: the RTT value between the SOURCE_XOM_ADDRESS and TARGET_XOM_ADDRESS

- ECHO_REQUEST:




VERSION: 8 bits, 0x 2
TYPE: 4 bits, 0x 4
HOPS_TO_LIVE: 4 bits, the number of hops can be relayed by XOMs
LEN: 16bits, the packet length including the header, always equals to 0x12
SOURCE_XOM_ADDRESSS: 32bits, the IPv4 address of the XOM that generate this echo request
RESERVED: 24 bits
SN: 8 bits

- ECHO_REPLY:




VERSION: 8 bits, 0x 2
TYPE: 4 bits, 0x 5
HOPS_TO_LIVE: 4 bits, the number of hops can be relayed by XOMs
LEN: 16bits, the packet length including the header, always equals to 0x12
SOURCE_XOM_ADDRESSS: 32bits, the IPv4 address of the XOM that generate this echo reply
RESERVED: 24 bits
SN: 8 bits

## References:

[1] Brutzman, D., M. Zyda, M., J.M. Pullen, and K.L. Morse, “Extensible Modeling and Simulation Framework (XMSF): Challenges for Web-Based Modeling and Simulation,” US Naval Postgraduate School, 2002