NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

Command and Control Data Dissemination Using IP Multicast

by

Raymond C. Barrera

December 1999

Thesis Advisor: Bert Lundy
Second Reader: John Iaia

Approved for public release; distribution is unlimited.

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	December 1999	Master's Thesis

4. TITLE AND SUBTITLE
Command and Control Data Dissemination Using IP Multicast

5. FUNDING NUMBERS

6. AUTHOR(S)
Barrera, Raymond C.

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

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

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

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

12b. DISTRIBUTION CODE

13. ABSTRACT *(maximum 200 words)*

Tools have been developed which allow tactical data to be exchanged over Internet Protocol networks, but the quality of service necessary to operate these tools is not available for most Naval vessels at this time. The objective of this thesis is to show that using Multicast IP, distributing data in layers using an efficient protocol, and sending data with no inherent mechanism to ensure that packets arrive at their destinations will allow data to be exchanged over IP networks at much lower bandwidths than is required today while still maintaining a common tactical picture. Software was developed which interfaces to GCCS-M and exchanges data over a multicast network. This software was tested in a laboratory which simulated a Naval environment. The results of testing demonstrate the potential of using the characteristics of the track data being exchanged in a true multicast architecture to develop a efficient tactical data distribution system for users operating in the Naval environment.

14. SUBJECT TERMS	15. NUMBER OF PAGES
multicast, command, control, communications, common operational picture	88
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

Approved for public release; distribution is unlimited

Command and Control Data Dissemination Using IP Multicast

Raymond C. Barrera - SPAWAR Systems Center, San Diego
B.S., California State Polytechnic University, Pomona, 1989

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN SOFTWARE ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
December 1999

Author: Raymond C. Barrera
Raymond C. Barrera

Approved by:
Bert Lundy, Thesis Advisor

John Iaia, Second Reader

DAN BOGER, CHAIRMAN
DEPARTMENT OF COMPUTER SCIENCE

ABSTRACT

Tools have been developed which allow tactical data to be exchanged over Internet Protocol networks, but the quality of service necessary to operate these tools is not available for most Naval vessels at this time. The objective of this thesis is to show that using Multicast IP, distributing data in layers using an efficient protocol, and sending data with no inherent mechanism to ensure that packets arrive at their destinations will allow data to be exchanged over IP networks at much lower bandwidths than is required today while still maintaining a common tactical picture. Software was developed which interfaces to GCCS-M and exchanges data over a multicast network. This software was tested in a laboratory which simulated a Naval environment. The results of testing demonstrate the potential of using the characteristics of the track data being exchanged in a true multicast architecture to develop a efficient tactical data distribution system for users operating in the Naval environment.

TABLE OF CONTENTS

ACKNOWLEDGMENT

The author would like to acknowledge the support of the Over The Horizon - Targeting Program at SPAWARSYSCEN San Diego for their support in facilities at the Reconfigurable Land Based Test Site (RLBTS) and guidance in the development of this thesis.

I. INTRODUCTION

A. OVER THE HORIZON - TARGETING

The advent of the cruise missile required the US Navy to develop mechanisms to exchange Over The Horizon Targeting (OTH-T) data in order to employ the missile against targets beyond the range of the weapons platforms' sensors. This led to the establishment of Battle Group Data Base Management (BGDBM) which used OTH-T GOLD (OTG) text messages transmitted over the Officer in Tactical Command Information Exchange Subsystem (OTCIXS) to maintain a Common Operational Picture (COP) between participants within a battle group. OTCIXS is an Ultra High Frequency Satellite Communication (UHF SATCOM) network with a channel access protocol designed for a low number of users. Messages transmitted on OTCIXS are acknowledged by a single net controller rather than by the intended recipient; consequently the message delivery path is inherently unreliable. The effective bandwidth of OTCIXS is less than 600 BPS. This mechanism has been used for over a decade by shore sites and surface combatants at both the force and unit levels, as well as submarines and some aircraft.

B. NAVAL COMMUNICATIONS

The Advanced Digital Networking System (ADNS) and its predecessors provided IP connectivity over SATCOM to afloat

units at the SECRET and UNCLAS levels. This allowed the use of Common Operational Picture (COP) software within the Global Command and Control System - Maritime (GCCS-M) to use TCP/IP connections to exchange data between force level ships and shore. This software evolved into the COP Synchronization Tools (CST) segment on GCCS-M which exchanged tactical data over IP networks. In order to address the serious bandwidth limitations of Naval ships a Multicast IP protocol was developed as part of the CST. This is a reliable protocol at the application layer. The database consistency is managed through a hierarchy of participants. Transactions are propagated throughout the entire network. This software still requires minimum available bandwidth on the order of 16-32 KBPS for each user. This quality of service is difficult to provide on unit level ships.

C. OBJECTIVES

Given the ability to operate on OTCIXS with multiple users it appears intuitive that a single user should not require an order of magnitude greater bandwidth than the entire theater requires on OTCIXS. The current CST software adds the ability to manage a great deal more data than OTCIXS and provides a reliable protocol to exchange data. This should ensure that databases replicated using CST on high bandwidth links should be identical. The objective of

this thesis is to show that using Multicast IP, distributing the data in layers using an efficient protocol, and sending data with no inherent mechanism to ensure that packets arrive at their destinations will allow data to be exchanged over IP networks at much lower bandwidths than is required today while still maintaining a common tactical picture.

D. SCOPE, LIMITATIONS, ASSUMPTIONS

Although software is being developed for this thesis which allows the exchange of tactical data through IP Multicast, it is not intended to be a deployable system. Since the focus of the thesis is to measure the bandwidth savings and data base consistency only software which supports these functions is being developed. Additional software would be needed to support fielding of similar functionality. This would include functions such as an enhanced operator interface, session control and announcement capabilities, and encryption techniques to verify the senders' identity and assure need to know.

The testing is being done on a network of Cisco routers using PIM-Sparse IP multicast routing. It does not take into account the effect of ATM switches or ADNS Proteon or Bay Networks routers using DVMRP.

THIS PAGE INTENTIONALLY LEFT BLANK

II. MULTICAST

A. MULTICAST VS UNICAST

Multicast IP networking was proposed in Internet Engineering Task Forces (IETF) Request For Comment (RFC) 966 written by Steve Deering and Dave Cheriton in 1985. Since then, several RFC's have been developed to describe the Internet Group Management Protocol (IGMP) and the necessary routing protocols to support Multicast IP. The concept of multicasting gained strength after the development of the Multicast Backbone (Mbone) in 1992. The Mbone provides a development environment to explore new designs and applications before these new concepts are adopted and deployed by commercial vendors.

Most applications that operate over Wide Area Networks (WANs) send data to each user in unicast packets to the users individual address. This results in many similar packets sent over the same network link, and increased bandwidth requirements near the data source as in Figure 1.

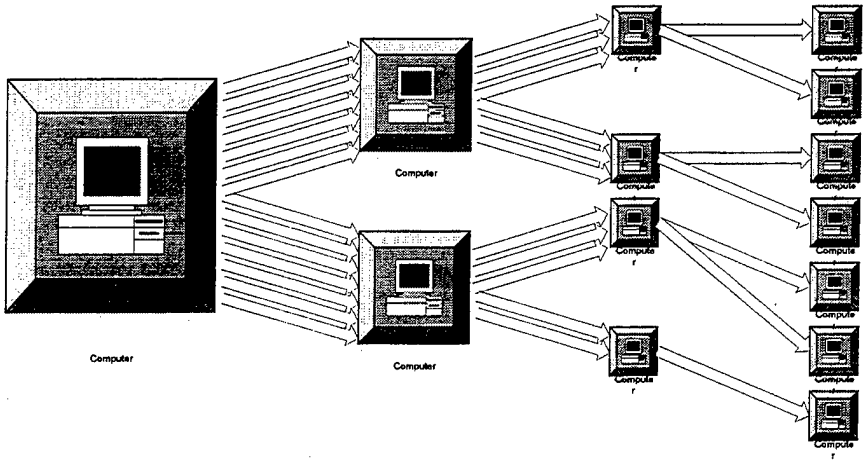

Figure 1 : Unicast Data Flow

Some local area network applications reduce bandwidth by broadcasting data on the LAN to all hosts on the network at one time (Figure 2). Multicast IP provides a mechanism to reduce the bandwidth required to send the same data to multiple hosts while restricting the dissemination of that data to interested participants.

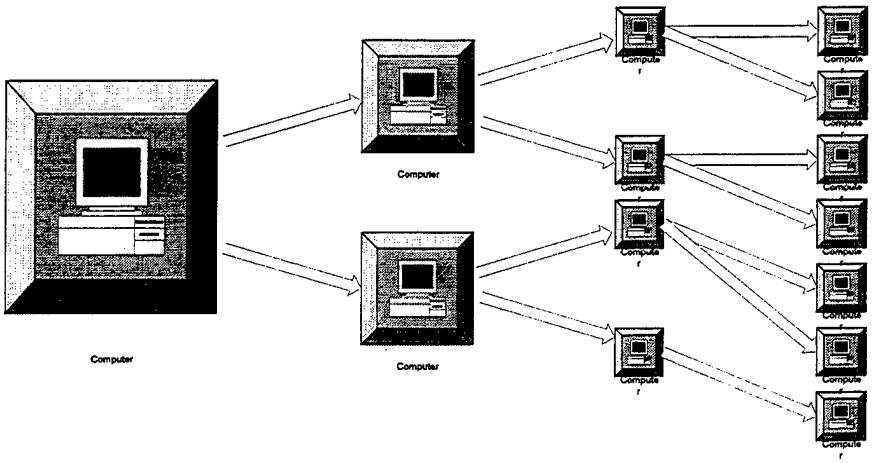

Figure 2 : Multicast Data Flow

Typical applications that transmit data over the Internet today use either User Datagram Protocol (UDP) or Transmission Control Protocol (TCP) as the transport layer protocol. UDP is a connectionless protocol which allows individual packets to be transmitted over the Internet Protocol (IP) in an unreliable manner between hosts. Because it is connectionless, it can be scaled easily to be addressed to broadcasts or multicasts. TCP is a reliable, connection-oriented protocol that ensures that packets can be reassembled without error in order to ensure a stream of data between a pair of hosts. It provides error checking, a sliding window of acknowledgments, flow control, and retransmission of data to ensure reliable transmission.

Applications built on the transmission of UDP packets tend to be easier to adapt to multicast than those utilizing TCP are. There is currently no standard for ensuring reliable transmission of data to a multicast groups. There are several proposed which will be discussed later in this paper. Often it is difficult to scale the design of an application that relies on reliable transmission of data to multicast.

B. IP MULTICAST BASICS

Multicast IP routes packets by sending them to hosts that are members of a host group. A multicast group is identified by a Class D IP address in the range 224.0.0.0-

239.255.255.255. These Class D addresses are not related to the IP addresses of the hosts that make up the group. Hosts indicate their interest in a group by using the Internet Group Management Protocol (IGMP). Hosts generate Membership Reports that are sent to a nearby router requesting to join a host group. The router will periodically query for at least one host in the local network that still wants to remain in the group. If no hosts respond, the router may be able to be pruned so that traffic for that host group no longer passes through it. Beginning with IGMP version 2, a host can explicitly leave a group.

Multicast IP, like the underlying IP protocol, is inherently unreliable. Devices that pass Multicast IP packets make a best effort to deliver them. Reliable protocols that can be built on top of Multicast IP are discussed in a later section.

Only a few well known global Class D addresses have been specified. The Class D address space is flat. It is important for applications to manage Class D addresses efficiently. They must use techniques for scoping the distance multicast packets they generate can travel on a network to avoid entering another network where the same Class D address is already being used. Currently that is done by setting the TTL of a packet. The TTL effectively limits the number of hops an IP packet can travel before it is rejected by a router. The TTL is used to avoid loops in

unicast routing that could cause a packet to travel forever. When a unicast packet is dropped by a router, an error message is sent back to the sender. No such message is generated for Multicast packets that exceed their TTL. A future method of controlling the dispersion of multicast traffic will be to limit the packet's scope administratively. It is impossible to determine the number of hops necessary to reach all members of an enterprise without going outside of the controlled network. Administrative scoping will allow the restriction of packets to a specified domain.

C. ROUTING

Multicast routing protocols establish a distribution tree to route data to all members of a group in an efficient manner. Two basic techniques are used to advertise the groups to potential group members, dense mode and sparse mode. Dense mode protocols flood the entire network with advertisements when a source begins transmission and prunes off connections which are not needed. These protocols are best suited to networks containing a concentration of members of a group with network resources not affected by the addition of often unwanted data. Sparse mode protocols are initiated from the receivers and thus only utilize links which are required to access interested parties. These

protocols are useful over WANs which contain widely dispersed group members and limited bandwidth. (Kosuir)

1. DVMRP

Distance Vector Multicast Routing Protocol (DVMRP) was originally used to establish the Mbone. Routers using DVMRP check the reverse path to a source when a packet is received in order to determine if it came from the shortest path between the source and the router. If it did, the packet is sent to all other links. If it did not, presumably it is a duplicate packet and it is discarded. DVMRP maintains its own unicast routing tables to determine the reverse paths. When a router has no attached subnets which want to receive data sent to a group it issues a prune message to the next router up the tree. This prune lasts for a limited time at which point the subnet may receive flooded packets again. When new members are added to a group graft messages can be used to add a new section to the tree.(Miller)

2. PIM Sparse

Protocol Independent Multicast (PIM)-Sparse is designated a "shared tree" protocol. This "shared tree" refers to a common distribution tree for all members of a group, regardless of their position relative to the source. Within each subnet, a Designated Router (DR) sends requests for any router in its subnet. Routers join a group by sending explicit join messages to the group's Rendezvous Point (RP). All of the data sent to this group passes

through the RP. Since the group may be widely dispersed, this may result in a sub-optimal distribution tree. (Miller)

D. RELIABLE VS UNRELIABLE

As a transport layer protocol, IP Multicast by itself is unreliable. That is, there is no inherent mechanism to ensure that packets arrive at their destinations. Macker (Macker 1996) describes a taxonomy of reliable requirements:

1) Best effort - similar to UDP in which no guarantees are made
2) Absolute - Similar to TCP in which all packets are delivered.
3) Bounded Latency - Each packet has a useful lifetime. After this time it may be discarded.
4) Most Recent - Only the most recent data is useful. In many command and control applications the most recent tactical data essentially supercedes all other reports.

1. Reliable protocols

A survey of reliable protocols and their application to military networking was performed by Petit (Petit 1996). Since that time there has been significant work to address some of the shortcomings of each protocol, but the fact remains that there is not a single solution to reliable multicasting for all applications.

In order for a reliable multicast protocol to be effective, it must be scalable. Simply applying the same concepts used to develop TCP would result in a unmanageable Ack implosions from a large group. Work continues in this area to determine efficient methods of sending repairs, dealing with late joiners, and handling asymmetric networks.

The Distributed Interactive Simulation effort established mechanisms to transmit data to members of groups participating in a distributed system. It has been succeeded by the High Level Architecture for Simulation (HLA). Its work is being continued by the Large Scale Multicast Applications (LSMA) group of the IETF. As the purpose of these protocols is to distribute situational awareness amongst the group members they are tackling many of the same problems affecting command and control data distribution.

2. Unreliable Protocols

Unreliable multicast protocols are simply streams of UDP packets destined for a class D IP address. There is no more than best effort to deliver the packets to all hosts in the group as there would be for unicast UDP packets. Furthermore, due to the complexity of multicast routing trees it is more likely to receive duplicate or out of order multicast packets than unicast packets.

III. NAVAL COMMUNICATIONS

A. NAVY LEGACY COMMAND AND CONTROL CAPABILITIES

1. Battle Group Data Base Management

In order to engage a target using an OTH-T weapon it is necessary to know the location and movements of all contacts in a large area. This requires the combination of data from multiple sources.

OTG messages are used to distribute and manage data from the originator's database. There is no single source to get an overall picture for the Navy. Early OTH-T experiments showed that it was important for all members of a battle to have the same picture. This allowed battle group commanders to make consistent decisions.

This consistent tactical picture is accomplished by software which complies with the Battle Group Data Base Management (BGDBM) specification. BGDBM defines the roles of Coordinator and Participant. The Coordinator provides the overall track management of the battle group database. The battle group database is a construction of the tactical picture which can be viewed by any of the participants. Participants provide inputs and receive direction from the coordinator. In the original instantiation of BGDBM, all data had to be sent from the Coordinator in order to be processed by the Participant. This simplified track

management but reduced the timeliness of the data. This was repaired by allowing the Coordinator to designate track reports from a given source for a contact to be accepted by the Participants.

Tactical data is exchanged using OTG messages transmitted over OTCIXS. Since OTCIXS is inherently unreliable, periodic SITREPs are transmitted by the coordinator to the participants. The SITREP contains the track number of each track in the database and the last time it was updated. Each Participant compares this SITREP to its own database. When there are discrepancies the Participant can manually request retransmission of missed data. In short, rather than ensuring reliable delivery of every transaction in the database, periodic SITREPs are used to bring the databases throughout the battle group back into synchronization. When latecomers join the group they are transmitted a data base dump which contains all of the contacts being reported to the group along with their last reported position.

This method of synchronization only supports updating the current position at the time of the SITREP. Though it is important that the current position be the same throughout the battlegroup at any given time, it is also important to keep track history points synchronized. OTH-T weapons use the track history to project to movement of a contact into the future. This is necessary to allow for the

long travel times of OTH-T weapons. Different track histories can lead to different projections. In turn, these different projections can lead to different decisions on how to employ or aim the OTH-T weapons.

2. Messaging

The Operational Specification for Over The Horizon GOLD (OS-OTG) describes the formatted text messages used to exchange non real-time tactical data. GOLD messages have been used for over a decade in Naval tactical systems. Originally, man-readable text messages were used in order to be displayed on a teletype terminal. Messages are formatted so they can be parsed automatically by tactical data processors (TDP). Messages are used to pass contact information and track management directives between TDPs. Contact information consists of positional information, attributes, and parametric information.

Contact attributes provide minimal identification characteristics. The information supplied varies depending on the type of contact and the sensor used to detect it. These may include ship class, name, hull number, IFF code and other discrete identifiers. The purpose of transmitting these attributes is to uniquely identify a contact. It is unnecessary to transmit redundant data once identification has been made. In many cases additional data may contradict previous reports and cause more ambiguity rather than clarification.

Track management messages are used to pass database transactions. No explicit creation messages are used. Track deletion messages can be sent as well as merge messages. Merge messages have the effect of joining data from two track records in the track database. Tracks are identified by a local track number assigned by the message originator and transmitted in each GOLD message. The originator can only update or manage data it has previously sent.

Parametric data can be sent in GOLD messages to allow sensor data to be processed at remote sites. Parametric information may include Electronic Intelligence (ELINT) frequency and spectral measurements and acoustic signature data.

3. OTCIXS

The Officer in Tactical Command Information Exchange Subsystem (OTCIXS) provides the current connectivity used to exchange OTH-T data between Naval platforms today. OTCIXS is a UHF SATCOM network which uses a slotted Aloha with reservations random access protocol. It is managed by a Net Controller which allows access to the net and acknowledges every message it receives without error. Messages that are not acknowledged are retransmitted by the originator. All receivers on the network receive all messages then apply a filter based on destination addresses to reduce the amount of data processed. The destination addressee must receive a

message without error at the same time the Net Controller does or it will not receive the message at all. Because messages are only acknowledged by the Net Controller and not the individual destination addressees the network is unreliable.

The design of broadcasting all messages to all sites and filtering on the receiving end allows for a type of multicast network to be established. All receivers operating as a battlegroup accept traffic destined for a multicast address for the battlegroup. Receivers in another battlegroup would have their own designated address and would ignore other battlegroup addresses. These multicast addresses are used to exchange Battle Group Data Base Management data between members of the battlegroup.

OTCIXS has additional characteristics that make it suitable for the transmission of tactical data. It has the capability to allow sites with higher precedence traffic to be transmit before those that do not. It can allow receivers to operate in Emissions Control states that do not allow acknowledgments to be transmitted. It also provides its operators enough insight into the transmission process to troubleshoot problems when they occur.

4. Track Data from other sources

Real-time organic contact data is shared between platforms over one or more Tactical Data Links (TADILs). Most prevalent in the Navy today are Link-16 and Link-11.

The TADILs often operate on Line of Sight radio equipment and have protocols ranging from simplistic basic contact data transfer mechanisms to very sophisticated protocols allowing voice and imagery to be transferred with contact data. These TADILs can provide updates on every track reported on the link in each net cycle time which can vary from a few seconds to a minute or more. Hundreds of tracks can be reported on these TADILs. There are mechanisms being explored to extend the TADILs Beyond Line of Sight using satellite and landline connections.

B. CURRENT IP CONNECTIVITY TO SHIPS

1. Types of RF connectivity

The Automated Digital Networking System (ADNS) allows IP connectivity through the available RF media through a consistent interface. The quality of service afforded by these links varies widely. Commercial SHF connections typically provide the highest bandwidth and lowest error rate available to ADNS. High access costs and limited availability of terminal equipment constrain its use on platforms which can support commercial SHF. DSCS SHF can have limited bandwidth available for tactical data exchange due to competition with other military requirements. In the case of either SHF the available bandwidth is related to size of antenna located on the vessel, but typically between 64Kbs and 512 KBPS are allocated for tactical data exchange

and related applications' IP connectivity. In order to avoid blockage of the antennae by parts of the vessel while maneuvering the antennae must be located high on the ships' masts. The size and weight of SHF antennae currently limit their use to large ships. A future capability provided by INMARSAT B may allow smaller ships to have IP connectivity at 32Kbs rates.

In addition to the relatively high data rates SHF provides Navy platforms, other military channels provide minimal IP capabilities when SHF is not available. These include UHF SATCOM, UHF DAMA, and EHF. Future capabilities based on GBS broadcast technology are currently being tested. This would allow an asymmetric connection with high bandwidth from shore to ship while a slower backchannel from ship to shore completed the connection.

2. Sources of errors on Naval IP networks

Any protocol designed to support tactical data exchange between Naval vessels needs to withstand errors which are peculiar to shipboard environment. One typical problem is antenna blockage. This can lead to connectivity disruptions lasting from minutes to hours depending on maneuvers. Another is the effect of low look angles toward geo-synchronous satellites from high latitudes ships often operate in. The low look angles lead to additional errors caused by thermal radiation from the earth, increased atmospheric interference, and increased distance traveled.

3. EMCON

One shipboard condition not typically found in the commercial sector is restrictive emission control (EMCON). In order to reduce the chance of detection a ship may avoid the use of transmitters which do not have a low probability of intercept. This condition is commonly found on submarines. This severely reduces or eliminates altogether the available bandwidth from the platform to other sites.

C. GCCS-M API'S

The Global Command and Control System - Maritime (GCCS-M) provides a software development environment used to create applications which utilize tactical data. This environment includes a set of Application Programmer Interfaces (APIs) which provide an interface to the Tactical Data Base Manager, an application which maintains the tactical database aboard a platform. These API's provide notification when an event such as an update, new position, or deletion occurs. They also allow applications to access tactical data, provide updates, and manage tactical data within the platform. The same set of tactical data is used by all GCCS-M applications on the same platform.

IV. PROTOCOL DESIGN GOALS

A. DATA BASE CONSISTENCY

The purpose of this project is to provide a capability which replaces the unreliable mechanisms available to the users with limited IP bandwidth. It is expected that those users which require high reliability will utilize the COP Synchronization Tools which provide a reliable mechanism for exchanging tactical data over IP networks. Therefore, it is not necessary to maintain perfect synchronization between participating sites. For purposes of design, we shall assume a design goal of maintaining 95% of the tracks in the database identical between the source of the data and any of the receivers. This goal is selected based on experience with current capabilities and the use of these capabilities by operators.

B. BANDWIDTH CONSTRAINTS

The goal of this protocol design is to support the user with limited bandwidth. The heaviest load of input tracks planned is from the organic tactical data links. These links can currently operate at 9.6 KBPS. Therefore, the desired bandwidth required should not exceed 9.6Kbs. This bandwidth will most likely be shared by other applications. This may lead to periods of time when the available bandwidth is not sufficient for the track update load. When

sufficient bandwidth is not available, the protocol designed should allow for graceful degradation rather than collapsing. For instance, if a reliable protocol were designed and sufficient bandwidth were not available, packets would monotonically become more timelate. As tactical data can quickly become replaced by more useful data, maintaining these old reports in queues in order for them to be reliably transferred would be counterproductive. Attempts to add delays in between packets to allow for queues in the transmission path only serve to reduce the timeliness of the data while delaying the inevitable breakdown of a reliable protocol on a link which cannot provide sufficient quality of service to support the reliable transfer of every packet in order with the same amount of overhead.

C. PROTOCOL CONSTRAINTS

The protocol designed should support all current capabilities and those that can be foreseen in the near future. In order to meet this requirement, all data fields which can be stored in the TDBM should be supported by this protocol. The protocol should be expandable in the future while maintaining backward compatibility. All data is expected to be transmitted in the message rather than referring to lookup tables which can become obsolete.

The protocol should operate under conditions common to the Naval operating environment, namely significant error rates, high latency, and restrictive EMCON settings. In order to operate over these restrictive conditions while any number of input data sources are updating the local data base, the protocol used to transmit data from a site should not be directly slaved to the data base update events. That is, the rate messages are transmitted from the node should be independent from the events driving those messages.

It is assumed that the source IP address of a packet uniquely identifies its source and no other unique identifier is required. This obviates the need for a user to enter a unique host id. No attempt is made to authenticate the source of data at this time. The protocol should support any number of sources and receivers in the group. Although it is intended for a battlegroup with only 5-10 nodes, it is conceivable that the same unreliable protocol could be scaled to support a large number of users interested in a particular data source.

D. PROTOCOL DESIGN

The protocol designed is based on the same mechanism used to transfer tactical data using Battle Group Data Base Management today. All members of the multicast group which have data inputs transmit that data to the entire group. Each member has control over the data it has provided to the

group. Messages are transmitted at a given broadcast interval. Each message is transmitted without acknowledgment. At a predetermined interval, Situation Reports (SITREPs) are transmitted which contain a data base dump of all contacts reported by that particular site. This allows receivers to reconcile their database with the rest of the group. Since occasionally messages will be transmitted with errors over RF paths, presumably as time goes on the number of messages received in error will be monotonically increasing. Since the most important goal is to maintain the most recent report on a given contact, when an update is received on a track without error that replaces the previous report with errors, it has in effect repaired the error. If the number of updates is sufficiently large in comparison to the number of messages received in error the data bases will converge over time rather than diverge. Since the recent track history is important to calculate a tracker solution, occasional track history updates are transmitted to the entire group. The SITREPs and history all serve as redundant reports to replace any errors on contacts without having to wait for an update. The advantage of sending the history reports redundantly as a collection rather than reliably sending each report is the savings on overhead. The track reports can be sent more efficiently in a group than as individuals.

In order to transmit data more efficiently events are queued until a minimum number of events are received, a maximum timeout is reached, or the encoded message exceeds a size threshold. This reduces the percentage of overhead transmitted by combining many events into a single message. The protocol should support layering of data by combining data from multiple groups into a common picture.

Contacts in a data base are uniquely identified by the source IP address and its associated track record number key. This allows maximum flexibility without requiring the management of another unique identifier besides the host IP address. Each event which causes a notification from the GCCS-M TDBM API is keyed to the track record number. This number is in turn used to identify a slot in a message containing events for that track. These events can include updates, new reports, deletions, etc. Each of these is queued until a message is transmitted. The receivers then process the events in a similar manner to those received locally.

E. COMPRESSION METHODS

In order to transmit data redundantly while operating over bandwidth-limited connections it is necessary to minimize the size of messages used to transfer data. The size of the record used to store the track which contains all of the data which may be transmitted is approximately

1200 bytes. As a first step, only fields which typically hold data were transmitted. This reduced the amount of data transmitted by nearly half. A further attempt was made to establish default values for all of the fields in the data structure. When the value stored was the default, a flag was set in the message indicating the field was not being sent and the receiver assumed the default value. In addition, fixed length strings were converted to variable length and only the amount of the string containing data and a null terminator was transmitted. These reduced a typical update message on a contact from 1200 bytes to fewer than 300 bytes. The driving design factor has been to support the injection of organic track data into the COP. Organic track reports were reduced to fewer than 200 bytes. This is consistent with the fact that Link-11 tracks contain a limited amount of data.

Given the reduction in message sizes by simply sending fields that had data it was unnecessary to employ more radical compression methods. Rather than simply randomizing the data, if compression was necessary some assumptions about the data can be made. For instance, a time is associated with every report. This time is typically reported as the number of seconds since Jan 1, 1970. This requires a long unsigned integer to store the number of bits necessary to report such a high number. However, knowing that most of the reports in a message are going to be

recent, the number of seconds before the generation of the message could be reported. Thus, the message generation time could be reported, then a smaller field used to report the offset between the time of event and the time of the message for each report contained in the message. A similar method could be used for position reporting, where the offset from a reference point is reported rather than absolute latitude and longitude. The values of these savings are multiplied if they can be applied to multiple reports in a single message.

Another mechanism that can be employed is the use of hash tables to encode commonly used fields. For instance, a sensor table could be transmitted within the message. For each report in the same message, an index to the sensor table could be transmitted rather than the string used to describe the sensor.

Given that there is some predictability to the data Huffman encoding could be used to further compress some fields. Going back to the sensor example, the most common sensor reported may be GPS. The next most common may be NTDS. A Huffman table can be created in which 1 bit is required to send GPS, two is required for NTDS, and the number increasing until the least likely sensor may take more bits to report using the tables than to explicitly report the sensor. The savings come in compressing the most

commonly used values. A mechanism for sending any unforeseen values in strings should be kept available.

Though not strictly a compression method, a system of layering data allows only the data which the receiver whishes to accept to be transmitted. This is implemented by establishing filters on the data sent to each group. Each group represents an orthogonal set of track data. By combining these layers the entire track picture is created. This scheme has been used to transmit interlaced motion imagery data, with the basic picture going to a group, additional detail going to another group, and increasing higher resolution to other groups. The receiver can then select which groups to subscribe to in order to receive the resolution desired. Presumably, multicast routing mechanisms are in place to prune off any undesired groups. In the same manner a set of high interest tracks may make up the essential group with higher resolution data being sent to other groups. The group members can then individually subscribe to the groups that interest them. The end user can then control the amount of data being sent to them rather than relying on a "smart-push" to select which data they want to receive. Note that simply moving the receiver from one group to another may not have the same effect. Given that there is typically a lag between the last host no longer requesting data for a group and the time the group is actually pruned from local routers, a user attempting to

drop from a high fidelity group to a lower one may experience even worse performance while subscribing to both groups until the higher one is pruned.

F. SPREADSHEET SIMULATION

A spreadsheet simulation of the effects of message size on messages received in error was used to gauge the approximate message size needed to maintain a common data base for a given percentage of the time.

The test set planned to be used to inject data into the experimental network updates Link-11 tracks on the average twice a minute, so this value was used for the spreadsheet simulation. The spreadsheet formulae determined that the databases were consistent ("in sync") if the receiving node had latest update of track. Therefore a lost packet was repaired if a subsequent message was successfully transmitted. A sample run of the spreadsheet scenario is shown in Table 1 for messages of 250 bytes long with update times of twice a minute, operating on a communications link with random errors of $1x10^{-5}$ errors/bit:

Table 1 : Message Transmission Scenario

Time	Msg Transmission	Cumulative Errors	total time	Cumulative Time in Sync	Cumulative Time in Error	Percentage of Time in Sync
1	Message #: 1	0	1	1	0	100%
1.1	|	0	1.1	1.1	0	100%
1.2	|	0	1.2	1.2	0	100%
1.3	|	0	1.3	1.3	0	100%
1.4	|	0	1.4	1.4	0	100%

1.5 |	0	1.5	1.5	0	100%
1.6 |	0	1.6	1.6	0	100%
1.7 |	0	1.7	1.7	0	100%
1.8 |	0	1.8	1.8	0	100%
1.9 Message #: 2 Error	1	1.9	1.8	0.1	95%
2 |	1	2	1.8	0.2	90%
2.1 |	1	2.1	1.8	0.3	86%
2.2 |	1	2.2	1.8	0.4	82%
2.3 |	1	2.3	1.8	0.5	78%
2.4 |	1	2.4	1.8	0.6	75%
2.5 Message #: 3	1	2.5	1.9	0.6	76%
2.6 |	1	2.6	2	0.6	77%
2.7 |	1	2.7	2.1	0.6	78%
2.8 |	1	2.8	2.2	0.6	79%
2.9 |	1	2.9	2.3	0.6	79%
3 |	1	3	2.4	0.6	80%
3.1 |	1	3.1	2.5	0.6	81%
3.2 |	1	3.2	2.6	0.6	81%
3.3 |	1	3.3	2.7	0.6	82%
3.4 Message #: 4	1	3.4	2.8	0.6	82%
3.5 Message #: 5	1	3.5	2.9	0.6	83%
3.6 |	1	3.6	3	0.6	83%
3.7 |	1	3.7	3.1	0.6	84%
3.8 |	1	3.8	3.2	0.6	84%
3.9 |	1	3.9	3.3	0.6	85%
4 Message #: 6	1	4	3.4	0.6	85%
4.1 |	1	4.1	3.5	0.6	85%
4.2 |	1	4.2	3.6	0.6	86%
4.3 |	1	4.3	3.7	0.6	86%
4.4 |	1	4.4	3.8	0.6	86%
4.5 |	1	4.5	3.9	0.6	87%
4.6 |	1	4.6	4	0.6	87%
4.7 |	1	4.7	4.1	0.6	87%
4.8 |	1	4.8	4.2	0.6	88%
4.9 Message #: 7	1	4.9	4.3	0.6	88%
5 Message #: 8	1	5	4.4	0.6	88%
5.1 |	1	5.1	4.5	0.6	88%
5.2 |	1	5.2	4.6	0.6	88%
5.3 Message #: 9	1	5.3	4.7	0.6	89%
5.4 |	1	5.4	4.8	0.6	89%
5.5 |	1	5.5	4.9	0.6	89%
5.6 |	1	5.6	5	0.6	89%
5.7 |	1	5.7	5.1	0.6	89%
5.8 Message #: 10	1	5.8	5.2	0.6	90%

5.9 |	1	5.9	5.3	0.6	90%
6 |	1	6	5.4	0.6	90%
6.1 |	1	6.1	5.5	0.6	90%
6.2 |	1	6.2	5.6	0.6	90%
6.3 |	1	6.3	5.7	0.6	90%
6.4 |	1	6.4	5.8	0.6	91%
6.5 |	1	6.5	5.9	0.6	91%
6.6 |	1	6.6	6	0.6	91%
6.7 Message #: 11	1	6.7	6.1	0.6	91%
6.8 |	1	6.8	6.2	0.6	91%
6.9 |	1	6.9	6.3	0.6	91%
7 |	1	7	6.4	0.6	91%
7.1 |	1	7.1	6.5	0.6	92%
7.2 |	1	7.2	6.6	0.6	92%
7.3 |	1	7.3	6.7	0.6	92%
7.4 |	1	7.4	6.8	0.6	92%
7.5 |	1	7.5	6.9	0.6	92%
7.6 Message #: 12	1	7.6	7	0.6	92%
7.7 |	1	7.7	7.1	0.6	92%
7.8 |	1	7.8	7.2	0.6	92%
7.9 |	1	7.9	7.3	0.6	92%
8 |	1	8	7.4	0.6	93%
8.1 |	1	8.1	7.5	0.6	93%
8.2 |	1	8.2	7.6	0.6	93%
8.3 |	1	8.3	7.7	0.6	93%
8.4 |	1	8.4	7.8	0.6	93%
8.5 Message #: 13	1	8.5	7.9	0.6	93%
8.6 |	1	8.6	8	0.6	93%
8.7 |	1	8.7	8.1	0.6	93%
8.8 |	1	8.8	8.2	0.6	93%
8.9 Message #: 14	1	8.9	8.3	0.6	93%
9 |	1	9	8.4	0.6	93%
9.1 |	1	9.1	8.5	0.6	93%
9.2 Message #: 15	1	9.2	8.6	0.6	93%
9.3 |	1	9.3	8.7	0.6	94%
9.4 |	1	9.4	8.8	0.6	94%
9.5 |	1	9.5	8.9	0.6	94%
9.6 |	1	9.6	9	0.6	94%
9.7 |	1	9.7	9.1	0.6	94%
9.8 |	1	9.8	9.2	0.6	94%
9.9 |	1	9.9	9.3	0.6	94%
10 Message #: 16	1	10	9.4	0.6	94%

399.5 Message #: 738	22	399.5	388.1	11.4	97%
399.6 Message #: 739	22	399.6	388.2	11.4	97%
399.7 |	22	399.7	388.3	11.4	97%
399.8 |	22	399.8	388.4	11.4	97%
399.9 Message #: 740	22	399.9	388.5	11.4	97%
400 |	22	400	388.6	11.4	97%
400.1 Message #: 741	22	400.1	388.7	11.4	97%
400.2 Message #: 742	22	400.2	388.8	11.4	97%
400.3 |	22	400.3	388.9	11.4	97%
400.4 |	22	400.4	389	11.4	97%
400.5 |	22	400.5	389.1	11.4	97%
400.6 |	22	400.6	389.2	11.4	97%
400.7 |	22	400.7	389.3	11.4	97%
400.8 |	22	400.8	389.4	11.4	97%

Summary

Error rate	1.00E-05
Packet size	250
Total Number of Messages	742
Total Number of Messages in error	22
Percentage Messages in Error	3%

This example demonstrates how the picture converges rather than diverges over time. After an early error on the second message subsequent messages were received without error and brought the current state of the receiver's tactical data base to be consistent with the sender's. The error caused a delay in synchronization, but the data bases were inherently stable. The simulation was run several times for various message sizes and the results recorded in Figure 3 below:

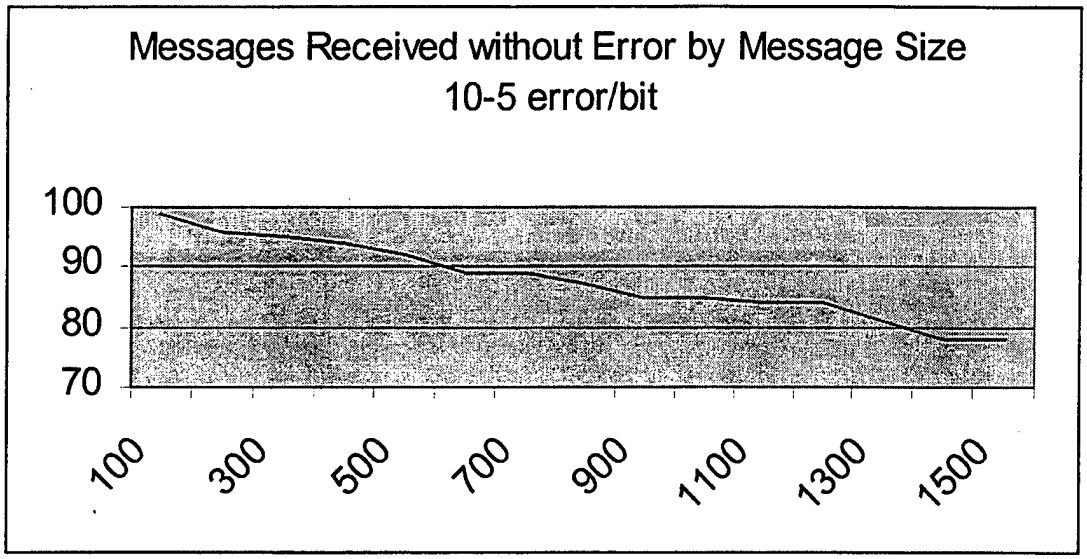

Figure 3 : Percentage Messages Received Without Error

These figures reflect errors on the line which may eventually be repaired through the data link level protocol used on the simulated satellite link. Therefore this represents the worst case scenario with no error correction. This chart indicates that a message size of approximately 200 bytes should be used to maintain a successful transmission rate of over 95%. This size is the design goal of the transmission protocol for a single track update so that it would be possible to generate messages this small for testing.

THIS PAGE INTENTIONALLY LEFT BLANK

V. TEST METHODOLOGY

A. TEST ARCHITECTURE

A test network was established in a laboratory to allow the software developed to demonstrate multicast data dissemination in a realistic but controlled environment. The GCCS-M nodes shown in Figure 4 were connected using CISCO 2514 routers via ADTECH SX-12 Satellite simulators. These allowed various bandwidths, delays, and random error rates to be configured to characterize a transmission media. The nodes represented a Commander, Joint Task Force (CJTF), Aircraft Carrier (CV), Cruiser (CG) and Destroyer (DDG). The connection between the CV and CJTF was expected to be relatively good compared to the unit level nodes. Available

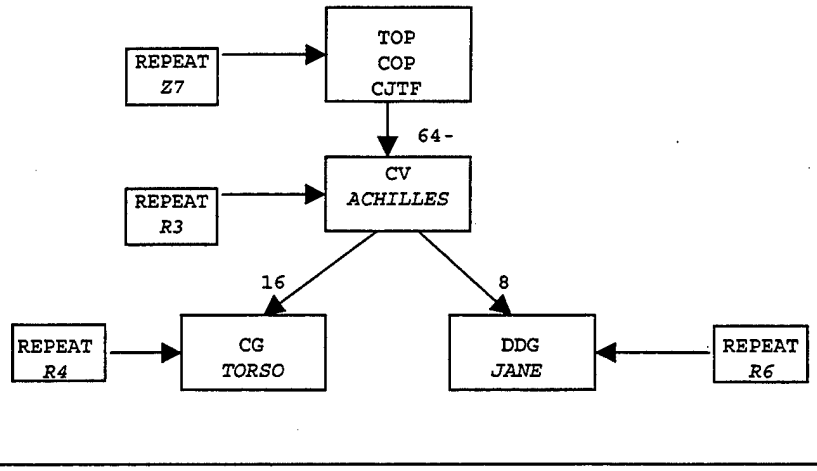

Figure 4: Test Architecture

bandwidths of between 8-64kps were used on these links along with a consistent delay of 900 ms and random error rate of $1x10^{-5}$ errors/bit.

PC's running the REpeatable Performance Evaluation Analysis Tool (REPEAT) were used to inject track data scenarios and to record track data broadcasts configured with inherent GCCS-M functionality. Data recorded by REPEAT was processed using the REPEAT message and contact compare analysis programs. LAN traffic at the CV node was recorded using a Network General Sniffer to independently verify transmitted message and contents.

VI. RESULTS

A. DATA COLLECTED

Data collected during the exchange of an unclassified test set was used to calculate the performance of this protocol. This test set contained 111 Link-11 tracks updated approximately 2 times a minute. The data collected show that the size of a message to update a track's attributes averaged 194 bytes. This message is 82% smaller than uncompressed CST messages and 67% smaller than compressed messages. The size of a message containing an individual track report averaged 88 bytes. The sizes of the components of a message used to transmit data are given in Table 2.

Table 2 : Track Data Element Size

Component	Average (bytes)	Max (bytes)	Min (bytes)	Std Dev (bytes)
Data	54	63	50	2.5
Header	61	61	61	0
Report	79	82	78	1.9
Total	194	206	189	4.3

Table 3 describes the size of messages transmitted from the test node. During the measurement period 200 messages were transmitted from this node. The message sizes ranged

from 24 bytes representing a message with no data to 1387 bytes, the maximum which can be sent with the configuration settings used.

Table 3 : Messages transmitted from Test Node

Total Messages Sent	200
Average Total Message Size	1200 bytes
Max Total Message Size	1387 bytes
Min Total Message Size	24 bytes
Std Dev	296 bytes

Table 4 describes messages received by the test node from other group members. The average message size from other sites was smaller than those transmitted since the test node had much more data to send and thus had fewer small, empty messages to reduce the average.

Table 4 : Messages Received by Test Node

Total Messages Received	325 (10 missing)
Average Total Message Size	1105 bytes
Max Total Message Size	1387 bytes
Min Total Message Size	24 bytes
Std Dev	400 bytes

A Link-11 file containing 111 tracks was injected into the CV node shown in Figure 4. REPEAT XR devices recorded a GENBCST from each of the members of the group to capture the changing database as it was updated from the multicast group. In a five minute period 688 updates were injected

into the CV node for 111 tracks. The timelate of the reports was measured after allowing for the time necessary for the data recording. This provided the average timelate values for each of the nodes given in Table 5.

Table 5 : Event Timelate

Node	Average Timelate (seconds)
CG	45
JTF	47
DDG	56

These measurements indicate that the latency of organic tracks remained below one minute in each of the configurations tested. Given the characterization of the test set it was calculated that 7600 BPS was required to maintain the common operational picture. At 8000 BPS the picture remained stable with all 111 tracks being held at all sites. When the available bandwidth for one site was reduced to 4800 BPS, the routers' serial interfaces toggled on and off. This resulted in even lower throughput. The insufficient bandwidth allowed only 95 tracks to be passed on the multicast group after two minutes. Reducing the bandwidth to only 2400 BPS allowed only 30 tracks to be exchanged on the multicast broadcast. These results demonstrated the desired characteristic of the multicast dissemination protocol to gracefully degrade in harsh conditions rather than coming to a complete halt.

THIS PAGE INTENTIONALLY LEFT BLANK

VII. CONCLUSIONS

Simulation results indicated that by reducing the size of messages used to exchange track data, the data could be sent often enough to maintain the desired 95% data base consistency. Software to demonstrate this capability was successfully developed and tested in the laboratory environment. The average message used to exchange track data was reduced to 194 bytes. This message is 82% smaller than uncompressed CST messages and 67% smaller than current compressed messages. The average report size was reduced to 88 bytes allowing track history to be sent efficiently. These smaller messages, coupled with the reduced overhead of an unreliable multicast transmission mechanism, allowed the challenging Link data to be transmitted over an 8 KBPS link with associated errors and delays. It is expected that although other types of data may contain more attributes and therefore require larger messages, their update rates will be considerably less than that of the Link-11 data used in these tests. These results with relatively simple optimization techniques demonstrate the potential bandwidth savings possible by encoding data in a manner which takes advantage of prior knowledge about the data being transmitted.

Although no explicit mechanism was used to ensure individual messages were reliably exchanged, the effect of

updating older tactical data with newer reports was to repair messages lost to error on transmission links. The small message sizes allowed for occasional broadcasts of the state of the entire track data base. These redundant reports also helped maintain the common operational picture.

An architecture conducive to exchanging data with multicast groups was utilized for the software used in this test. There are no restrictions on the number of members of a group or the number of groups a host may be a member of. These concepts are important if the ability to exchange tactical data through multicast groups is to expand to a global scale.

The results of testing of the software developed to test the capability of transferring the common operational picture using multicast groups demonstrate the potential of using the characteristics of the track data being exchanged in a true multicast architecture to develop a efficient tactical data distribution system for users operating in the Naval environment.

VIII. RECOMMENDATIONS

Although the results of this testing showed that taking the characteristics of the data being transferred into account when developing a communications protocol is a very powerful tool, only relatively simple mechanisms were implemented. In order to implement an effective protocol, the entire data base structure should be considered. Other encoding schemes described in section IV.E should be examined. The rules governing the uses of these compression methods should be established in a manner allowing for future systems to be backward compatible while promoting growth in functionality.

The reduction in message size alone would have a significant effect on the ability to exchange the common operational picture, but implementing an unreliable multicast mechanism would allow that picture to be sent to many Naval participants which are unable to receive it now due to insufficient bandwidth or asymmetric network configuration. To support the disadvantaged users, the capability to transmit and receive data on independent groups which are joined at the receiver's request should be established.

All of the concepts presented here have been well known for years. In order to support Fleet users, these concepts

should now be applied to specific Naval communications environments.

APPENDIX. SOURCE CODE

The source code files which contain the significant subroutines used to implement a test program to demonstrate distribution of a common operational picture using multicast are included in this appendix.

A. UMCOP_MAIN.C

```
/*****************************************************************
 *                                                               *
 *      PROGRAM:          UMCOP                                  *
 *      FILENAME:         umcop_main.c                           *
 *      CREATION DATE:    01/12/99                               *
 *      AUTHOR:           Raymond Barrera                        *
 *    CLASSIFICATION:     UNCLASSIFIED                           *
 *    DESCRIPTION:        Contains main routine for multicast COP*
 *                        test program.                          *
 *****************************************************************/


extern Event_Queue event_queue[],dump_event_queue[];
int number_of_events = 0,number_of_dump_events = 0;
int number_received_events=0;
Event_Queue received_events[MAX_EVENTS];
extern int receive_message();
extern void parse_message();
extern void process_events();
extern int calculate_message_size();
extern unsigned char ttl_value;

void main(int argc, char *argv[])
{
    int     irecord,          /* ILOG record              */
            orecord,          /* OLOG record              */
            error,            /* Error code               */
            delay_time = 0,   /* Delay for reconnections */
            interval = 30,    /* Delay interval           */
            msg_delay = 0,    /* Delay for win updates    */
            event_cnt = 0,    /* Event counter            */
            result;           /* TDBM event poll result   */
    fd_set ifds,              /* Active input fd set      */
            ofds;             /* Active output fd set     */
    struct timeval timeout;   /* Select timeout           */
    char    file[80];         /* Email filename           */
    XEvent xevent;            /* Next X event             */
int broadcast_interval = 60*15;
int data_dump_interval = 60*60;
int max_events_in_message = 20;
time_t current_time, *tptr, last_broadcast_time=0,
last_data_dump_time=0; /* for testing, send dumps at startup. later
change to time program started */
```

```
unsigned long group_address;
int port_number = 9123;
int socket;
Event_Queue last_event;
int i;
int data_dump_in_progress=0,last_trkrec_dumped =
0,number_dump_msgs_cycle=4;
int history_dump_in_progress,history_number;


        FILE *fd;           /* File descriptor    */
        char  filename[32], /* File with PID info */
              pid[16];      /* Process ID         */
        unsigned char *message_body;
int message_size,received_size;
struct sockaddr_in from_ip;
unsigned char *received_message;
Message_Header header;
int maximum_message_size;


/* following for testing */
     group_address = inet_addr("234.5.6.7");
     broadcast_interval = 30;
     maximum_message_size = 1500-780;
    for(i=1;i<argc;i++) {



fprintf(stdout,"Group Address: %s:%d broadcast interval %d  data dump
interval %d msgs cycle %d size %d TTL
%d\n",address_to_a(group_address),port_number,broadcast_interval,data_du
mp_interval,number_dump_msgs_cycle,maximum_message_size,ttl_value);
    /*
     * Open connection with TDBM
     */
    open_tdbm();
    /*
     * Bring up the search window.
     */
    VtInitSearchFilter(&filter);
    VtEditSearchFilter(&filter, DEF_FLTR);
    /*
     * Clear the active file descriptor set, set up the set,
     * and set up the timer for the select call.
     */



     socket = initialize_socket(group_address,port_number);

     initialize_event_queue(event_queue,&number_of_events);
     initialize_event_queue(dump_event_queue,&number_of_events);

    while (1)
    {
        /*
         * Handle the track events. Up to 5 events are handle at a time.
         * If an error is detected, reconnect to TDBM.
```

```
         */
       result = VtGetNextTdbmEvent(tdbm_fd, &tdbm_event);
         fprintf(stdout,"Result = %d\n",result);
         if(result == 0)
         {
           HandleTdbmEvents(tdbm_event);
         }
         if (result == ERROR)
         {
             fprintf(stderr,"Error returned from TDBM");
             close_tdbm();
             open_tdbm();
         }
         current_time = time(tptr);
         if((number_of_events > max_events_in_message)||((current_time -
last_broadcast_time) > broadcast_interval)||(maximum_message_size <
calculate_message_size(event_queue,number_of_events))) {
#ifdef TESTING
fprintf(stdout,"Printing all events from main\n");
print_all_events(event_queue,number_of_events);
#endif

send_broadcast_message(group_address,port_number,socket,event_queue,numb
er_of_events);
            last_broadcast_time = current_time;
            initialize_event_queue(event_queue,&number_of_events);

          }
         if((current_time - last_data_dump_time) > data_dump_interval) {
            data_dump_in_progress = True;
            last_trkrec_dumped = 0;
            last_data_dump_time = current_time;

initialize_event_queue(dump_event_queue,&number_of_dump_events);
#ifdef TESTING
fprintf(stdout,"Data dump intiated\n");
#endif
         }
         if(data_dump_in_progress) {
         for(i = 0; i< number_dump_msgs_cycle;i++) {
            while(data_dump_in_progress && (maximum_message_size >
calculate_message_size(dump_event_queue,number_of_dump_events))) {

add_track_to_dump(last_trkrec_dumped,dump_event_queue,&number_of_dump_ev
ents);
/*
               number_of_dump_events++;
*/
               last_trkrec_dumped++;
#ifdef TESTING
/*last_trkrec_dumped = MAXRECS; */
#endif
               if(last_trkrec_dumped >= MAXRECS) {
                        data_dump_in_progress = False;
                        history_dump_in_progress = True;
                        last_trkrec_dumped = 0;
               }

            }
#ifdef TESTING
```

```
fprintf(stdout,"data dump message being sent with %d
events\n",number_of_dump_events);
#endif

send_broadcast_message(group_address,port_number,socket,dump_event_queue
,number_of_dump_events);

initialize_event_queue(dump_event_queue,&number_of_dump_events);

          }
          }
          else if(history_dump_in_progress) {
         for(i = 0; i< number_dump_msgs_cycle;i++) {
            while(history_dump_in_progress && (maximum_message_size >
calculate_message_size(dump_event_queue,number_of_dump_events))) {
             while((history_number == 0)&&(last_trkrec_dumped < MAXRECS))
{
#ifdef TESTING
                  fprintf(stdout,"looking for history point in
%d\n",last_trkrec_dumped);
#endif

get_next_history_point_number(last_trkrec_dumped,&history_number);
                  if(history_number == 0) last_trkrec_dumped++;
               }
#ifdef TESTING
                  fprintf(stdout,"using history point %d in
%d\n",history_number,last_trkrec_dumped);
#endif
               if(last_trkrec_dumped >= MAXRECS) {
                        history_dump_in_progress = False;
                        last_trkrec_dumped = 0;
               }
               else  {
#ifdef TESTING
                  fprintf(stdout,"adding history point %d in %d to
dump\n",history_number,last_trkrec_dumped);
#endif

add_history_point_to_dump(last_trkrec_dumped,history_number,dump_event_q
ueue,&number_of_dump_events);
                  history_number--;
                  if(history_number <= 0) {
                         last_trkrec_dumped++;
                         history_number = 0;
/*  number_of_dump_events++ */
                  }
               }

            }

send_broadcast_message(group_address,port_number,socket,dump_event_queue
,number_of_dump_events);

initialize_event_queue(dump_event_queue,&number_of_dump_events);
         }

         }
```

```
            received_size =
receive_message(socket,&received_message,&from_ip,&header);
#ifdef TESTING
fprintf(stderr,"Received %d bytes\n",received_size);
#endif
            while(received_size > 0) {
               fprintf(stdout,"Received %d bytes\n",received_size);
               fprintf(stdout,"main: From host:%s port:%d\n",
inet_ntoa(from_ip.sin_addr), ntohs(from_ip.sin_port));


parse_message(received_message,received_size,from_ip,header,received_eve
nts,&number_received_events);
               free(received_message);

process_events(received_events,&number_received_events,from_ip,tdbm_fd);
               received_size =
receive_message(socket,&received_message,&from_ip,&header);
            }
    }
    close_tdbm();

    exit(0);
}
```

THIS PAGE INTENTIONALLY LEFT BLANK

B. EVENT_QUEUE.H

```
/*****************************************************************
 *                                                               *
 *      FILENAME:       event_queue.h                            *
 *      AUTHOR:         Raymond Barrera                          *
 *      CREATION DATE:  01/12/99                                 *
 *      CLASSIFICATION: UNCLASSIFIED                             *
 *      DESCRIPTION:    Contains structures used for TDBM event queue*
 *                                                               *
 *****************************************************************/

#define MAX_HISTORY 26
#define MAX_EVENTS 1000


typedef struct {
     int affected_trkrec;
     int deleted;
     VtTrackDeleteMsg delete_event;
     int merged;
     VtTrackMergeMsg merge_event;
     int reports_added;
     VtTrackAddReportMsg add_report_events[MAX_HISTORY];
     int reports_deleted;
     VtTrackDeleteReportMsg delete_report_events[MAX_HISTORY];
     int track_updated;
     VtTrackUpdateMsg update_track_event;
} Event_Queue;


typedef struct {
     int local_trkrec;
     int group_owned_track;
     unsigned long owner_ip;
     int owner_trkrec;
} Track_List;

typedef struct {
   unsigned short int message_type;
   unsigned short int msn;
} Message_Header;
```

THIS PAGE INTENTIONALLY LEFT BLANK

C. MESSAGING.C

```
/*****************************************************************
 *                                                               *
 *      FILENAME:       messaging.c                              *
 *      AUTHOR:         Raymond Barrera                          *
 *      CREATION DATE:  01/12/99                                 *
 *      CLASSIFICATION: UNCLASSIFIED                             *
 *      DESCRIPTION:    Contains routines for encoding and decoding *
 *                      messages.                                *
 *                                                               *
 *****************************************************************/


#define SET_MASK(X,Y) X |= 1<<Y
#define MASK_SET(X,Y) X & (1<<Y)



int encode_header(header,hdr_ptr)
VtTrackHeader header;
unsigned char *hdr_ptr;
{
int tmp_size;
unsigned int mask = 0;
int field = 0;
int size = 0;
unsigned char *ptr;


     ptr = hdr_ptr;
     tmp_size = sizeof(mask);
     memcpy(ptr,&mask,tmp_size);
     size += tmp_size;
     ptr += tmp_size;
     tmp_size = sizeof(header.type);
     memcpy(ptr,&(header.type),tmp_size);
     size += tmp_size;
     ptr += tmp_size;
     tmp_size = sizeof(header.trkrec);
     memcpy(ptr,&(header.trkrec),tmp_size);
     size += tmp_size;
     ptr += tmp_size;
     tmp_size = sizeof(header.source);
     memcpy(ptr,&(header.source),tmp_size);
     size += tmp_size;
     ptr += tmp_size;
     tmp_size = sizeof(header.assoc);
     memcpy(ptr,&(header.assoc),tmp_size);
     size += tmp_size;
     ptr += tmp_size;
     tmp_size = sizeof(header.child);
     memcpy(ptr,&(header.child),tmp_size);
     size += tmp_size;
     ptr += tmp_size;
     tmp_size = sizeof(header.machine);
     memcpy(ptr,&(header.machine),tmp_size);
```

```
        size += tmp_size;
        ptr += tmp_size;
        tmp_size = encode_string(header.serial,ptr);
        size += tmp_size;
        ptr += tmp_size;
        tmp_size = 8;
        memcpy(ptr,&(header.ltn),tmp_size);
        size += tmp_size;
        ptr += tmp_size;
        tmp_size = UID_TRIGRAPH_SIZE;
        memcpy(ptr,&(header.last_send_uid),tmp_size);
        size += tmp_size;
        ptr += tmp_size;
        tmp_size = UID_TRIGRAPH_SIZE;
        memcpy(ptr,&(header.rr_uid),tmp_size);
        size += tmp_size;
        ptr += tmp_size;
        tmp_size = 4;
        memcpy(ptr,&(header.spare),tmp_size);
        size += tmp_size;



        tmp_size = sizeof(mask);
        memcpy(hdr_ptr,&mask,tmp_size);
        return(size);
}

int decode_header(header,hdr_ptr)
VtTrackHeader *header;
unsigned char *hdr_ptr;
{
int tmp_size;
unsigned int mask = 0;
int field = 0;
int size = 0;
unsigned char *ptr;


        ptr = hdr_ptr;
        tmp_size = sizeof(mask);
        memcpy(&mask,ptr,tmp_size);
        size += tmp_size;
        ptr += tmp_size;
        tmp_size = sizeof(header->type);
        memcpy(&(header->type),ptr,tmp_size);
        size += tmp_size;
        ptr += tmp_size;
        tmp_size = sizeof(header->trkrec);
        memcpy(&(header->trkrec),ptr,tmp_size);
        size += tmp_size;
        ptr += tmp_size;
        tmp_size = sizeof(header->source);
        memcpy(&(header->source),ptr,tmp_size);
        size += tmp_size;
        ptr += tmp_size;
        tmp_size = sizeof(header->assoc);
        memcpy(&(header->assoc),ptr,tmp_size);
        size += tmp_size;
        ptr += tmp_size;
```

```
        tmp_size = sizeof(header->child);
        memcpy(&(header->child),ptr,tmp_size);
        size += tmp_size;
        ptr += tmp_size;
        tmp_size = sizeof(header->machine);
        memcpy(&(header->machine),ptr,tmp_size);
        size += tmp_size;
        ptr += tmp_size;
        tmp_size = decode_string(header->serial,ptr);
        size += tmp_size;
        ptr += tmp_size;
        tmp_size = 8;
        memcpy(&(header->ltn),ptr,tmp_size);
        size += tmp_size;
        ptr += tmp_size;
        tmp_size = UID_TRIGRAPH_SIZE;
        memcpy(&(header->last_send_uid),ptr,tmp_size);
        size += tmp_size;
        ptr += tmp_size;
        tmp_size = UID_TRIGRAPH_SIZE;
        memcpy(&(header->rr_uid),ptr,tmp_size);
        size += tmp_size;
        ptr += tmp_size;
        tmp_size = 4;
        memcpy(&(header->spare),ptr,tmp_size);
        size += tmp_size;
        ptr += tmp_size;

        tmp_size = sizeof(mask);
        memcpy(hdr_ptr,&mask,tmp_size);
        return(size);

}


int decode_string(string,ptr)
char *string;
unsigned char *ptr;
{
int tmp_size;

        tmp_size = strlen((char *)ptr) + 1;
        memcpy(string,ptr,tmp_size);
        return(tmp_size);
}

int encode_string(string,ptr)
char *string;
unsigned char *ptr;
{
int tmp_size;

        tmp_size = strlen(string) + 1;
        memcpy(ptr,string,tmp_size);
        return(tmp_size);
}



int decode_platform_data(data,data_ptr)
```

```
VtPlatformData *data;
unsigned char *data_ptr;
{
int tmp_size,size=0;
unsigned int mask = 0,field = 0;
unsigned char *ptr;

     ptr = data_ptr;
     tmp_size = sizeof(mask);
     memcpy(&mask,ptr,tmp_size);
     size += tmp_size;
     ptr += tmp_size;
     tmp_size = sizeof(VtTrknum);
     memcpy(&data->ftn,ptr,tmp_size);
     size += tmp_size;
     ptr += tmp_size;
/*
     for(i=0;i<VT_MAX_RTN;i++) {
     tmp_size = sizeof(VtTrknum);
     memcpy(&(data->rtn[i]),ptr,tmp_size);
     size += tmp_size;
     ptr += tmp_size;
     }
*/
     if(MASK_SET(mask,field)) {
     tmp_size = decode_string(data->shipclass,ptr);
     size += tmp_size;
     ptr += tmp_size;
     }
     else data->shipclass[0] = '\0';
        field++;
     if(MASK_SET(mask,field)) {
     tmp_size = decode_string(data->name,ptr);
     size += tmp_size;
     ptr += tmp_size;
     }
     else data->name[0] = '\0';
        field++;
     if(MASK_SET(mask,field)) {
     tmp_size = decode_string(data->trademark,ptr);
     size += tmp_size;
     ptr += tmp_size;
     }
     else data->trademark[0] = '\0';
        field++;
     if(MASK_SET(mask,field)) {
     tmp_size = decode_string(data->type,ptr);
     size += tmp_size;
     ptr += tmp_size;
     }
     else data->type[0] = '\0';
        field++;
     if(MASK_SET(mask,field)) {
     tmp_size = decode_string(data->hull,ptr);
     size += tmp_size;
     ptr += tmp_size;
     }
     else data->hull[0] = '\0';
        field++;
     if(MASK_SET(mask,field)) {
```

```
tmp_size = decode_string(data->flag,ptr);
size += tmp_size;
ptr += tmp_size;
}
else data->flag[0] = '\0';
   field++;
if(MASK_SET(mask,field)) {
tmp_size = decode_string(data->sconum,ptr);
size += tmp_size;
ptr += tmp_size;
}
else data->sconum[0] = '\0';
   field++;
if(MASK_SET(mask,field)) {
tmp_size = decode_string(data->pif,ptr);
size += tmp_size;
ptr += tmp_size;
}
else data->pif[0] = '\0';
   field++;
if(MASK_SET(mask,field)) {
tmp_size = decode_string(data->ntds,ptr);
size += tmp_size;
ptr += tmp_size;
}
else data->ntds[0] = '\0';
   field++;
if(MASK_SET(mask,field)) {
tmp_size = decode_string(data->di,ptr);
size += tmp_size;
ptr += tmp_size;
}
else data->di[0] = '\0';
   field++;
if(MASK_SET(mask,field)) {
tmp_size = decode_string(data->callsign,ptr);
size += tmp_size;
ptr += tmp_size;
}
else data->callsign[0] = '\0';
   field++;
if(MASK_SET(mask,field)) {
tmp_size = decode_string(data->uic,ptr);
size += tmp_size;
ptr += tmp_size;
}
else data->uic[0] = '\0';
   field++;
if(MASK_SET(mask,field)) {
tmp_size = sizeof(data->quantity);
memcpy(&data->quantity,ptr,tmp_size);
size += tmp_size;
ptr += tmp_size;
}
   field++;
if(MASK_SET(mask,field)) {
tmp_size = decode_string(data->home_base,ptr);
size += tmp_size;
ptr += tmp_size;
}
```

```
else data->home_base[0] = '\0';
   field++;
if(MASK_SET(mask,field)) {
tmp_size = sizeof(data->db_type);
memcpy(&data->db_type,ptr,tmp_size);
size += tmp_size;
ptr += tmp_size;
}
   field++;
if(MASK_SET(mask,field)) {
tmp_size = decode_string(data->db_num,ptr);
size += tmp_size;
ptr += tmp_size;
}
else data->db_num[0] = '\0';
   field++;
if(MASK_SET(mask,field)) {
tmp_size = decode_string(data->alert,ptr);
size += tmp_size;
ptr += tmp_size;
}
else data->alert[0] = '\0';
   field++;
if(MASK_SET(mask,field)) {
tmp_size = decode_string(data->fcode,ptr);
size += tmp_size;
ptr += tmp_size;
}
else data->fcode[0] = '\0';
   field++;
if(MASK_SET(mask,field)) {
tmp_size = decode_string(data->category,ptr);
size += tmp_size;
ptr += tmp_size;
}
else data->category[0] = '\0';
   field++;
if(MASK_SET(mask,field)) {
tmp_size = decode_string(data->threat,ptr);
size += tmp_size;
ptr += tmp_size;
}
else data->threat[0] = '\0';
   field++;
if(MASK_SET(mask,field)) {
tmp_size = decode_string(data->shortname,ptr);
size += tmp_size;
ptr += tmp_size;
}
else data->shortname[0] = '\0';
   field++;
if(MASK_SET(mask,field)) {
tmp_size = decode_string(data->xref,ptr);
size += tmp_size;
ptr += tmp_size;
}
else data->xref[0] = '\0';
   field++;
if(MASK_SET(mask,field)) {
tmp_size = decode_string(data->orig_xref,ptr);
```

```
      size += tmp_size;
      ptr += tmp_size;
      }
      else data->orig_xref[0] = '\0';
         field++;
      if(MASK_SET(mask,field)) {
      tmp_size = decode_string(data->chxref,ptr);
      size += tmp_size;
      ptr += tmp_size;
      }
      else data->chxref[0] = '\0';
         field++;
      if(MASK_SET(mask,field)) {
      tmp_size = sizeof(data->latfixed);
      memcpy(&data->latfixed,ptr,tmp_size);
      size += tmp_size;
      ptr += tmp_size;
      }
         field++;
      if(MASK_SET(mask,field)) {
      tmp_size = sizeof(data->lngfixed);
      memcpy(&data->lngfixed,ptr,tmp_size);
      size += tmp_size;
      ptr += tmp_size;
      }
         field++;
      if(MASK_SET(mask,field)) {
      tmp_size = sizeof(data->jtn);
      memcpy(&data->jtn,ptr,tmp_size);
      size += tmp_size;
      ptr += tmp_size;
      }
         field++;
      if(MASK_SET(mask,field)) {
      tmp_size = sizeof(data->spare);
      memcpy(&data->spare,ptr,tmp_size);
      size += tmp_size;
      ptr += tmp_size;
      }
         field++;
      return(size);
}

int encode_platform_data(data,data_ptr)
VtPlatformData data;
unsigned char *data_ptr;
{
int tmp_size,size=0;
unsigned int mask = 0,field = 0;
unsigned char *ptr;

      ptr = data_ptr;
      tmp_size = sizeof(mask);
      memcpy(ptr,&mask,tmp_size);
      size += tmp_size;
      ptr += tmp_size;
      tmp_size = sizeof(data.ftn);
      memcpy(ptr,&data.ftn,tmp_size);
      size += tmp_size;
      ptr += tmp_size;
```

```
/*
     for(i=0;i<VT_MAX_RTN;i++) {
     tmp_size = sizeof(VtTrknum);
     memcpy(ptr,&(data.rtn[i]),tmp_size);
     size += tmp_size;
     ptr += tmp_size;
     }
*/
     if(strlen(data.shipclass) > 0) {
     tmp_size = encode_string(data.shipclass,ptr);
     size += tmp_size;
     ptr += tmp_size;
        SET_MASK(mask,field);
     }
        field++;
     if(strlen(data.shipclass) > 0) {
     tmp_size = encode_string(data.name,ptr);
     size += tmp_size;
     ptr += tmp_size;
        SET_MASK(mask,field);
     }
        field++;
     if(strlen(data.trademark) > 0) {
     tmp_size = encode_string(data.trademark,ptr);
     size += tmp_size;
     ptr += tmp_size;
        SET_MASK(mask,field);
     }
        field++;
     if(strlen(data.type) > 0) {
     tmp_size = encode_string(data.type,ptr);
     size += tmp_size;
     ptr += tmp_size;
        SET_MASK(mask,field);
     }
        field++;
     if(strlen(data.hull) > 0) {
     tmp_size = encode_string(data.hull,ptr);
     size += tmp_size;
     ptr += tmp_size;
        SET_MASK(mask,field);
     }
        field++;
     if(strlen(data.flag) > 0) {
     tmp_size = encode_string(data.flag,ptr);
     size += tmp_size;
     ptr += tmp_size;
        SET_MASK(mask,field);
     }
        field++;
     if(strlen(data.sconum) > 0) {
     tmp_size = encode_string(data.sconum,ptr);
     size += tmp_size;
     ptr += tmp_size;
        SET_MASK(mask,field);
     }
        field++;
     if(strlen(data.pif) > 0) {
     tmp_size = encode_string(data.pif,ptr);
     size += tmp_size;
```

```
ptr += tmp_size;
   SET_MASK(mask,field);
}
   field++;
if(strlen(data.ntds) > 0) {
tmp_size = encode_string(data.ntds,ptr);
size += tmp_size;
ptr += tmp_size;
   SET_MASK(mask,field);
}
   field++;
if(strlen(data.di) > 0) {
tmp_size = encode_string(data.di,ptr);
size += tmp_size;
ptr += tmp_size;
   SET_MASK(mask,field);
}
   field++;
if(strlen(data.callsign) > 0) {
tmp_size = encode_string(data.callsign,ptr);
size += tmp_size;
ptr += tmp_size;
   SET_MASK(mask,field);
}
   field++;
if(strlen(data.uic) > 0) {
tmp_size = encode_string(data.uic,ptr);
size += tmp_size;
ptr += tmp_size;
   SET_MASK(mask,field);
}
   field++;
if(data.quantity > 0) {
tmp_size = sizeof(data.quantity);
memcpy(ptr,&data.quantity,tmp_size);
size += tmp_size;
ptr += tmp_size;
   SET_MASK(mask,field);
}
   field++;
if(strlen(data.home_base) > 0) {
tmp_size = encode_string(data.home_base,ptr);
size += tmp_size;
ptr += tmp_size;
   SET_MASK(mask,field);
}
   field++;
if(data.db_type > 0) {
tmp_size = sizeof(data.db_type);
memcpy(ptr,&data.db_type,tmp_size);
size += tmp_size;
ptr += tmp_size;
   SET_MASK(mask,field);
}
   field++;
if(strlen(data.db_num) > 0) {
tmp_size = encode_string(data.db_num,ptr);
size += tmp_size;
ptr += tmp_size;
   SET_MASK(mask,field);
```

```
}
   field++;
if(strlen(data.alert) > 0) {
tmp_size = encode_string(data.alert,ptr);
size += tmp_size;
ptr += tmp_size;
   SET_MASK(mask,field);
}
   field++;
if(strlen(data.fcode) > 0) {
tmp_size = encode_string(data.fcode,ptr);
size += tmp_size;
ptr += tmp_size;
   SET_MASK(mask,field);
}
   field++;
if(strlen(data.category) > 0) {
tmp_size = encode_string(data.category,ptr);
size += tmp_size;
ptr += tmp_size;
   SET_MASK(mask,field);
}
   field++;
if(strlen(data.threat) > 0) {
tmp_size = encode_string(data.threat,ptr);
size += tmp_size;
ptr += tmp_size;
   SET_MASK(mask,field);
}
   field++;
if(strlen(data.shortname) > 0) {
tmp_size = encode_string(data.shortname,ptr);
size += tmp_size;
ptr += tmp_size;
   SET_MASK(mask,field);
}
   field++;
if(strlen(data.xref) > 0) {
tmp_size = encode_string(data.xref,ptr);
size += tmp_size;
ptr += tmp_size;
   SET_MASK(mask,field);
}
   field++;
if(strlen(data.orig_xref) > 0) {
tmp_size = encode_string(data.orig_xref,ptr);
size += tmp_size;
ptr += tmp_size;
   SET_MASK(mask,field);
}
   field++;
if(strlen(data.chxref) > 0) {
tmp_size = encode_string(data.chxref,ptr);
size += tmp_size;
ptr += tmp_size;
   SET_MASK(mask,field);
}
   field++;
if(data.latfixed > 0) {
tmp_size = sizeof(data.latfixed);
```

```
        memcpy(ptr,&data.latfixed,tmp_size);
        size += tmp_size;
        ptr += tmp_size;
           SET_MASK(mask,field);
        }
           field++;
        if(data.lngfixed > 0) {
        tmp_size = sizeof(data.lngfixed);
        memcpy(ptr,&data.lngfixed,tmp_size);
        size += tmp_size;
        ptr += tmp_size;
           SET_MASK(mask,field);
        }
           field++;
        if(data.jtn > 0) {
        tmp_size = sizeof(data.jtn);
        memcpy(ptr,&data.jtn,tmp_size);
        size += tmp_size;
        ptr += tmp_size;
           SET_MASK(mask,field);
        }
           field++;
        if(data.spare[0] > 0) {
        tmp_size = sizeof(data.spare);
        memcpy(ptr,&data.spare,tmp_size);
        size += tmp_size;
        ptr += tmp_size;
           SET_MASK(mask,field);
        }
           field++;
        tmp_size = sizeof(mask);
        memcpy(data_ptr,&mask,tmp_size);
        return(size);
}


int
generate_broadcast_message(event_queue,message_body,number_of_events)
Event_Queue *event_queue;
unsigned char **message_body;
int number_of_events;
{
unsigned char temp_message[20000];
unsigned char *temp_message_ptr;
int event_number = 0;
int message_size = 0;
int segment_size = 0;
unsigned int number_of_type_of_event;
unsigned char *ptr_to_number_of_type_of_event;
int i;

temp_message_ptr = temp_message;

/*
fprintf(stdout,"Printing from encoder\n");
print_all_events(event_queue,number_of_events);
*/
/* merge */
number_of_type_of_event = 0;
```

```
ptr_to_number_of_type_of_event = temp_message_ptr;
segment_size = sizeof(unsigned int);
message_size += segment_size;
temp_message_ptr += segment_size;
for(event_number = 0;event_number < number_of_events;event_number++)
{
   if(event_queue[event_number].merged > 0) {
#ifdef TESTING
fprintf(stdout,"Encoding merges in event %d\n",event_number);
#endif
      segment_size =
add_merge_message(temp_message_ptr,event_queue[event_number].merge_event
);
      if(segment_size >0) {
fprintf(stdout,"Merge message size: %d\n",segment_size);
         message_size += segment_size;
         temp_message_ptr += segment_size;
         number_of_type_of_event++;
      }
      else {
         fprintf(stderr,"Error occurred adding Merge message\n");
      }
   }
}

memcpy(ptr_to_number_of_type_of_event,&number_of_type_of_event,sizeof(un
signed int));
/* deleted */
number_of_type_of_event = 0;
ptr_to_number_of_type_of_event = temp_message_ptr;
segment_size = sizeof(unsigned int);
message_size += segment_size;
temp_message_ptr += segment_size;
for(event_number = 0;event_number < number_of_events;event_number++)
{

   if(event_queue[event_number].deleted > 0) {
#ifdef TESTING
fprintf(stdout,"Encoding deletes in event %d\n",event_number);
#endif
      segment_size =
add_delete_message(temp_message_ptr,event_queue[event_number].delete_eve
nt);
      if(segment_size >0) {
#ifdef TESTING
fprintf(stdout,"Delete message size: %d\n",segment_size);
#endif
         message_size += segment_size;
         temp_message_ptr += segment_size;
         number_of_type_of_event++;
      }
      else {
         fprintf(stderr,"Error occurred adding delete message\n");
      }
/* don't send any other updates on deleted track */
      initialize_event(event_queue,event_number);
   }
}
```

```
memcpy(ptr_to_number_of_type_of_event,&number_of_type_of_event,sizeof(un
signed int));


/* deleted reports */
number_of_type_of_event = 0;
ptr_to_number_of_type_of_event = temp_message_ptr;
segment_size = sizeof(unsigned int);
message_size += segment_size;
temp_message_ptr += segment_size;
for(event_number = 0;event_number < number_of_events;event_number++)
{
   if(event_queue[event_number].reports_deleted > 0) {
#ifdef TESTING
fprintf(stdout,"Encoding deleted reports in event %d\n",event_number);
#endif
      segment_size =
add_delete_report_message(temp_message_ptr,event_queue[event_number].rep
orts_deleted,event_queue[event_number].delete_report_events);
      if(segment_size >0) {
fprintf(stdout,"Delete Report message size: %d\n",segment_size);
         message_size += segment_size;
         temp_message_ptr += segment_size;
         number_of_type_of_event++;
      }
      else {
         fprintf(stderr,"Error occurred adding delete reports
message\n");
      }
   }
}

memcpy(ptr_to_number_of_type_of_event,&number_of_type_of_event,sizeof(un
signed int));

/* added reports */
number_of_type_of_event = 0;
ptr_to_number_of_type_of_event = temp_message_ptr;
segment_size = sizeof(unsigned int);
message_size += segment_size;
temp_message_ptr += segment_size;
for(event_number = 0;event_number < number_of_events;event_number++)
{
   if(event_queue[event_number].reports_added > 0) {
#ifdef TESTING
fprintf(stdout,"Encoding added reports in event %d\n",event_number);
#endif
      segment_size =
add_add_report_message(temp_message_ptr,event_queue[event_number].report
s_added,event_queue[event_number].add_report_events);
      if(segment_size >0) {
fprintf(stdout,"Add report message size: %d\n",segment_size);
         message_size += segment_size;
         temp_message_ptr += segment_size;
         number_of_type_of_event++;
      }
      else {
         fprintf(stderr,"Error occurred adding add reports message\n");
      }
```

```
    }
}

memcpy(ptr_to_number_of_type_of_event,&number_of_type_of_event,sizeof(un
signed int));
/* changed tracks */
number_of_type_of_event = 0;
ptr_to_number_of_type_of_event = temp_message_ptr;
segment_size = sizeof(unsigned int);
message_size += segment_size;
temp_message_ptr += segment_size;
for(event_number = 0;event_number < number_of_events;event_number++)
{
   if(event_queue[event_number].track_updated > 0) {
#ifdef TESTING
fprintf(stdout,"Encoding change to trkrec
%d\n",event_queue[event_number].affected_trkrec);
#endif
      segment_size =
add_update_message(temp_message_ptr,event_queue[event_number].update_tra
ck_event);
#ifdef TESTING
      fprintf(stdout,"Encoded change to trkrec %d. Segment size =
%d\n",event_number,segment_size);
#endif
      if(segment_size >0) {
fprintf(stdout,"Update message size: %d\n",segment_size);
         message_size += segment_size;
         temp_message_ptr += segment_size;
         number_of_type_of_event++;
      }
      else {
         fprintf(stderr,"Error occurred adding update message\n");
      }
   }
}

memcpy(ptr_to_number_of_type_of_event,&number_of_type_of_event,sizeof(un
signed int));

*message_body = malloc(message_size);
memcpy(*message_body,temp_message,message_size);

#ifdef TESTING
fprintf(stdout,"Encoded message\n");
#endif

return(message_size);

}

int
parse_update_message(temp_message_ptr,received_events,number_received_ev
ents)
unsigned char *temp_message_ptr;
Event_Queue *received_events;
int *number_received_events;
{
int size,tmp_size;
unsigned char *tmp_ptr;
```

```
VtTrackUpdateMsg new_event;

     size = 0;
     tmp_ptr = temp_message_ptr;
     tmp_size = decode_header(&(new_event.track.hdr),tmp_ptr);
#ifdef TESTING
fprintf(stdout,"parse: header size = %d\n",tmp_size);
fprintf(stdout,"new event trkrec = %d\n",new_event.track.hdr.trkrec);
#endif
     size += tmp_size;
     tmp_ptr += tmp_size;
     tmp_size = decode_report(&new_event.track.ptrk.rpt,tmp_ptr);
#ifdef TESTING
fprintf(stdout,"decoded report size = %d\n",tmp_size);
#endif
     size += tmp_size;
     tmp_ptr += tmp_size;
     switch(new_event.track.hdr.type) {
        case VtPlatformTrackType:
           tmp_size =
decode_platform_data(&new_event.track.ptrk.data,tmp_ptr);
           size += tmp_size;
           tmp_ptr += tmp_size;
           break;
        case VtEmitterTrackType:
           tmp_size = sizeof(new_event.track.etrk.rad);
           memcpy(&new_event.track.etrk.rad,tmp_ptr,tmp_size);
           size += tmp_size;
           tmp_ptr += tmp_size;
           tmp_size = sizeof(new_event.track.etrk.data);
           memcpy(&new_event.track.etrk.data,tmp_ptr,tmp_size);
           size += tmp_size;
           tmp_ptr += tmp_size;
           break;
        case VtAcousticTrackType:
           tmp_size = sizeof(new_event.track.atrk.signa);
           memcpy(&new_event.track.atrk.signa,tmp_ptr,tmp_size);
           size += tmp_size;
           tmp_ptr += tmp_size;
           tmp_size = sizeof(new_event.track.atrk.data);
           memcpy(&new_event.track.atrk.data,tmp_ptr,tmp_size);
           size += tmp_size;
           tmp_ptr += tmp_size;
           break;
        case VtLinkTrackType:
           tmp_size =
decode_link_data(&new_event.track.ptrk.data,tmp_ptr);
           size += tmp_size;
           tmp_ptr += tmp_size;
           break;
        case VtUnitTrackType:
           tmp_size = sizeof(new_event.track.utrk.data);
           memcpy(&new_event.track.utrk.data,tmp_ptr,tmp_size);
           size += tmp_size;
           tmp_ptr += tmp_size;
           break;
        default: fprintf(stderr,"Unknown type updated\n");
      }

#ifdef TESTING
```

```
fprintf(stdout,"tmp_size = %d\n",tmp_size);
#endif

#ifdef TESTING
fprintf(stdout,"Adding update event %d trkrec
%d\n",*number_received_events,new_event.track.hdr.trkrec);
#endif

add_update_track_event(received_events,new_event,number_received_events)
;
     return(size);
}

void
parse_broadcast_message(message_body,message_size,received_events,number
_received_events)
unsigned char *message_body;
int message_size;
Event_Queue *received_events;
int *number_received_events;
{
unsigned char *temp_message_ptr;
int event_number = 0;
int segment_size = 0;
int remaining_size;
unsigned int number_of_type_of_event;
unsigned char *ptr_to_number_of_type_of_event;
int i;

temp_message_ptr = message_body;
remaining_size = message_size;
#ifdef TESTING
fprintf(stdout,"Decoding merges\n");
#endif

memcpy(&number_of_type_of_event,temp_message_ptr,sizeof(unsigned int));
segment_size = sizeof(unsigned int);
remaining_size -= segment_size;
temp_message_ptr += segment_size;

for(i=0;i<number_of_type_of_event;i++) {
      segment_size =
parse_merge_message(temp_message_ptr,received_events,number_received_eve
nts);
      if(segment_size >0) {
         remaining_size -= segment_size;
         temp_message_ptr += segment_size;
      }
      else {
         fprintf(stderr,"Error occurred decoding merge message\n");
      }
}

#ifdef TESTING
fprintf(stdout,"Number of merges = %d\n",number_of_type_of_event);
#endif

#ifdef TESTING
fprintf(stdout,"Decoding deletes\n");
#endif
```

```
memcpy(&number_of_type_of_event,temp_message_ptr,sizeof(unsigned int));
segment_size = sizeof(unsigned int);
remaining_size -= segment_size;
temp_message_ptr += segment_size;

#ifdef TESTING
fprintf(stdout,"Decoding %d  deletes\n",number_of_type_of_event);
#endif
for(i=0;i<number_of_type_of_event;i++) {
      segment_size =
parse_delete_message(temp_message_ptr,received_events,number_received_ev
ents);
      if(segment_size >0) {
         remaining_size -= segment_size;
         temp_message_ptr += segment_size;
      }
      else {
         fprintf(stderr,"Error occurred decoding delete message\n");
      }
}
#ifdef TESTING
fprintf(stdout,"Number of deletes = %d\n",number_of_type_of_event);
#endif


#ifdef TESTING
fprintf(stdout,"Decoding deleted_reports\n");
#endif

memcpy(&number_of_type_of_event,temp_message_ptr,sizeof(unsigned int));
segment_size = sizeof(unsigned int);
remaining_size -= segment_size;
temp_message_ptr += segment_size;

for(i=0;i<number_of_type_of_event;i++) {
      segment_size =
parse_delete_report_message(temp_message_ptr,received_events,number_rece
ived_events);
      if(segment_size >0) {
         remaining_size -= segment_size;
         temp_message_ptr += segment_size;
      }
      else {
         fprintf(stderr,"Error occurred decoding deleted_reports
message\n");
      }
}
#ifdef TESTING
fprintf(stdout,"Number of deleted_reports =
%d\n",number_of_type_of_event);
#endif


#ifdef TESTING
fprintf(stdout,"Decoding added reports\n");
#endif

memcpy(&number_of_type_of_event,temp_message_ptr,sizeof(unsigned int));
segment_size = sizeof(unsigned int);
```

```
remaining_size -= segment_size;
temp_message_ptr += segment_size;

for(i=0;i<number_of_type_of_event;i++) {
      segment_size =
parse_add_report_message(temp_message_ptr,received_events,number_receive
d_events);
      if(segment_size >0) {
         remaining_size -= segment_size;
         temp_message_ptr += segment_size;
      }
      else {
         fprintf(stderr,"Error occurred decoding add_report message\n");
      }
}
#ifdef TESTING
fprintf(stdout,"Number of added reports =
%d\n",number_of_type_of_event);
#endif

#ifdef TESTING
fprintf(stdout,"Decoding changes\n");
#endif

memcpy(&number_of_type_of_event,temp_message_ptr,sizeof(unsigned int));
segment_size = sizeof(unsigned int);
remaining_size -= segment_size;
temp_message_ptr += segment_size;

for(i=0;i<number_of_type_of_event;i++) {
      segment_size =
parse_update_message(temp_message_ptr,received_events,number_received_ev
ents);
      if(segment_size >0) {
         remaining_size -= segment_size;
         temp_message_ptr += segment_size;
      }
      else {
         fprintf(stderr,"Error occurred decoding update message\n");
      }
   }
}

#define NUM_STORED_MSNS 20
void
parse_message(received_message,received_size,from_ip,header,received_eve
nts,number_received_events)
unsigned char *received_message;
int received_size;
struct sockaddr_in from_ip;
Message_Header header;
Event_Queue *received_events;
int *number_received_events;
{
int i,difference,found=0;
static unsigned short int last_msn[NUM_STORED_MSNS];
static char sources[NUM_STORED_MSNS][80];

#ifdef TESTING
   fprintf(stdout,"Received message of type %d\n",header.message_type);
```

```
    fprintf(stdout,"parse_message: From host:%s port:%d\n",
inet_ntoa(from_ip.sin_addr), ntohs(from_ip.sin_port));
#endif
    for(i=0;i<NUM_STORED_MSNS;i++) {
      if(!strcmp(sources[i],inet_ntoa(from_ip.sin_addr))) {
         difference = header.msn - (last_msn[i] + 1);
         last_msn[i] = header.msn;
         found = 1;
         break;
      }
    }
    if(!found) {
       for(i=0;i<NUM_STORED_MSNS;i++) {
          if(sources[i][0] == '\0') {
             sprintf(sources[i],"%s",inet_ntoa(from_ip.sin_addr));
             last_msn[i] = header.msn;
             if(header.msn != 1) difference = header.msn;
fprintf(stdout,"First message recieved from
%s\n",inet_ntoa(from_ip.sin_addr));
fprintf(stdout, "adding source %s in slot
%d\n",inet_ntoa(from_ip.sin_addr),i);
             break;
          }
       }
     }

   fprintf(stdout,"Received msn %d from
%s\n",header.msn,inet_ntoa(from_ip.sin_addr));
   if(difference > 0) fprintf(stdout,"MISSING %d Messages from
%s\n",difference,inet_ntoa(from_ip.sin_addr));

   switch(header.message_type) {
      case 1:

parse_broadcast_message(received_message,received_size,received_events,n
umber_received_events);
            break;
      default: fprintf(stderr,"Unknown message type %d
received\n",header.message_type);

   }


}
```

THIS PAGE INTENTIONALLY LEFT BLANK

REFERENCES

Kosuir, David R., *IP Multicasting: The Complete Guide to Interactive Corporate Networks*, John Wiley & Sons, 1998.

Macker, Joseph P., Klinker, J. Eric,and Corson, M. Scott, "Reliable Multicast Delivery for Military Networking", URL http://tonnant.itd.nrl.navy.mil/papers/RM/RM.html, 1996.

Miller, Kenneth C., *Multicast Networking and Communications*, Addison Wesley Longman, 1999.

Petit, David G., "Solutions for Reliable Multicasting", Master's Thesis, Naval Postgraduate School, 1996.

THIS PAGE INTENTIONALLY LEFT BLANK

BIBLIOGRAPHY

1. Acharya, Arup, Bakre, Ajay and Badrinath, B.R. "IP Multicast Extensions for Mobile Internetworking," IEEE INFOCOM '96, Volume 1 1996.
2. Aguilar, Lorenzo, "Datagram Routing for Internet Multicasting," ACM Computer Communications Review, Volume 14, Number 2 1984.
3. Bernstein, Arthur J., "A Loosely Coupled Distributed Sytem for Reliably Storing Data," IEEE Transactions on Software Engineering, Volume SE-11, Number 5 1985.
4. Deering, S.E, and Cheriton, D.E., "Host Groups: A Multicast Extension to the Internet Protocol," RFC-966, 1985.
5. Deering, Stephen E., "Multicast Routing in InterNetworks and Extended LANs," ACM Computer Communications Review, Volume 18, Number 4 1988.
6. Deering, Steve, "Host Extensions for IP Multicasting," Request for Comments (RFC) 1112, Internet Engineering Task Force (IETF), August 1989.
7. Floyd, S., Jacobson, V., McCanne, S., Liu, C.-G., and Zhang, L., "A Reliable Multicast Framework for Light-weight Sessions and Application Level Framing," ACM SIGCOMM, Aug 1995.
8. Grossglauser, Matthias, "Optimal Deterministic Timeouts for Reliable Scalable Multicast," IEEE INFOCOM '96, Volume 3 1996.
9. Halsall, Fred, *Data Communications, Computer Networks, and Open Systems*, Addison-Wesley Publishing, 1992.
10. Kim, John C., *Naval Shipboard Communications Systems*, Prentice Hall, 1995.
11. Koifman, Alex, Zabele, Stephen "RAMP: A Reliable Adaptive Multicast Protocol," IEEE INFOCOM '96, Volume 3 1996.
12. Kosuir, David R., *IP Multicasting: The Complete Guide to Interactive Corporate Networks*, John Wiley & Sons, 1998.
13. Lin, John C., Paul, Sanjoy "RMTP:A Reliable Multicast Transport Protocol," IEEE INFOCOM '96, Volume 3 1996.

14. Macker, Joseph P., Klinker, J. Eric,and Corson, M. Scott, "Reliable Multicast Delivery for Military Networking", URL http://tonnant.itd.nrl.navy.mil/papers/RM/RM.html, 1996.

15. Miller, Kenneth C., *Multicast Networking and Communications*, Addison Wesley Longman, 1999.

16. Moy, John, "Multicast Routing Extensions for OSPF," Communications of the ACM, vol. 37 no. 8 Aug 1994.

17. Ofek, Yoram, Yener, Bulent "Reliable Concurrent Multicast from Bursty Sources," IEEE INFOCOM '96, Volume 3 1996.

18. O'Neil, Patrick, *Database-Principles, Programming, Performance*, Morgan Kaufmann Publishers, 1994.

19. Perkins, Charles E., *Mobile IP: Design Principles and Practices*, Addison-Wesley Publishing, 1998.

20. Petit, David G., "Solutions for Reliable Multicasting", Master's Thesis, Naval Postgraduate School, 1996.

21. Pingali, S., Towsley, D., and Kurose, J., "A Comparison of Sender-Initiated and Receiver-Initiated Reliable Multicast Protocols," ACM SIGMETRICS, May 1994.

22. Schulzrinne, H., Casner, S., Frederick, R., and Jacobson, V., "RTP: A Transport Protocol for Real-Time Applications," RFC 1889, Nov. 1995.

23. Tanenbaum, Andrew S., *Computer Networks*, Prentice Hall, 1996

24. Zakhor, Avideh, "Image and Video Compression", IEEE INFOCOM, 1996.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center.....2
   8725 John J. Kingman Rd., STE 0944
   Ft. Belvoir, VA 22060-6218

2. Dudley Knox Library..... 2
   Naval Postgraduate School
   411 Dyer Rd.
   Monterey, CA 93943-5101

3. Dr. Dan Boger..... 1
   Chairman, Code CS
   Computer Science Department
   Naval Postgraduate School
   Monterey, CA 93943-5000

4. Dr. Bert Lundy, Professor 2
   Computer Science Department Code CS
   Naval Postgraduate School
   Monterey, CA 93943-5000

5. John Iaia 1
   Head, Code D45
   Space and Naval Warfare Systems Center
   53540 Hull Street
   San Diego, CA 92152-5001

6. Bob Stephenson..... 2
   Space and Naval Warfare Systems Center Pacific D424
   675 Lehua Ave.
   Pearl City, HI 96782-3356

7. Cheryl Putnam..... 2
   OTH-T Program Manager Code D4123
   Space and Naval Warfare Systems Center
   53540 Hull Street
   San Diego, CA 92152-5001