X Display Manager Control Protocol

X.Org Standard

Keith Packard, X Consortium, Laboratory for Computer Science, Massachusetts Institute of Technology

X Display Manager Control Protocol: X.Org Standard

by Keith Packard

Version 1.1

Copyright © 1989, 2004 The Open Group

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE OPEN GROUP BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Except as contained in this notice, the name of The Open Group shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Software without prior written authorization from The Open Group.

X Window System is a trademark of The Open Group.

Table of Contents

Chapter 1. Purpose and Goals

The purpose of the X Display Manager Control Protocol (XDMCP) is to provide a uniform mechanism for an autonomous display to request login service from a remote host. By autonomous, we mean the display consists of hardware and processes that are independent of any particular host where login service is desired. (For example, the server cannot simply be started by a `fork/exec` sequence on the host.) An X terminal (screen, keyboard, mouse, processor, network interface) is a prime example of an autonomous display.

From the point of view of the end user, it is very important to make autonomous displays as easy to use as traditional hardwired character terminals. Specifically, you can typically just power on a hardwired terminal and be greeted with a login prompt. The same should be possible with autonomous displays. However, in a network environment with multiple hosts, the end user may want to choose which host(s) to connect to. In an environment with many displays and many hosts, a site administrator may want to associate particular collections of hosts with particular displays. We would like to support the following options:

- The display has a single, fixed host to which it should connect. It should be possible to power on the display and receive a login prompt, without user intervention.
- Any one of several hosts on a network or subnetwork may be acceptable for accepting login from the display. (For example, the user's file systems can be mounted onto any such host, providing comparable environments.) It should be possible for the display to broadcast to find such hosts and to have the display either automatically choose a host or present the possible hosts to the user for selection.
- The display has a fixed set of hosts that it can connect to. It should be possible for the display to have that set stored in RAM, but it should also be possible for a site administrator to be able to maintain host sets for a large number of displays using a centralized facility, without having to interact (physically or electronically) with each individual display. Particular hosts should be allowed to refuse login service, based on whatever local criteria are desired.

The control protocol should be designed in such a way that it can be used over a reasonable variety of communication transport layers. In fact, it is quite desirable if every major network protocol family that supports the standard X protocol is also capable of supporting XDMCP, because the end result of XDMCP negotiation will be standard X protocol connections to the display. However, because the number of displays per host may be large, a connection-based protocol appears less desirable than a connection-less protocol. For this reason the protocol is designed to use datagram services with the display responsible for sequencing and retransmission.

To keep the burden on displays at a minimum (because display cost is not a factor that can be ignored), it is desirable that displays not be required to maintain permanent state (across power cycles) for the purposes of the control protocol, and it is desirable to keep required state at a minimum while the display is powered on.

Security is an important consideration and must be an integral part of the design. The important security goals in the context of XDMCP are:

- It should be possible for the display to verify that it is communicating with a legitimate host login service. Because the user will present credentials (for example, password) to this service, it is important to avoid spoof attacks.

- It should be possible for the display and the login service to negotiate the authorization mechanism to be used for the standard X protocol.
- It should be possible to provide the same level of security in verifying the login service as is provided by the negotiated authorization mechanism.
- Because there are no firm standards yet in the area of security, XDMCP must be flexible enough to accomodate a variety of security mechanisms.

Chapter 2. Overview of the Protocol

XDMCP is designed to provide authenticated access to display management services for remote displays. A new network server, called a \fIDisplay Manager\fP, will use XDMCP to communicate with displays to negotiate the startup of X sessions. The protocol allows the display to authenticate the manager. It also allows most of the configuration information to be centralized with the manager and to ease the burden of system administration in a large network of displays. The essential goal is to provide plug-and-play services similar to those provided in the familiar mainframe/terminal world.

Displays may be turned off by the user at any time. Any existing session running on a display that has been turned off must be identifiable. This is made possible by requiring a three-way handshake to start a session. If the handshake succeeds, any existing session is terminated immediately and a new session started. There is the problem (at least with TCP) that connections may not be closed when the display is turned off. In most environments, the manager should reduce this problem by periodically XSync'ing on its own connection, perhaps every five to ten minutes, and terminating the session if its own connection ever closes.

Displays should not be required to retain permanent state for purposes of the control protocol. One solution to packets received out of sequence would be to use monotonically increasing message identifiers in each message to allow both sides to ignore messages that arrive out-of-sequence. For this to work, displays would at a minimum have to increment a stable crash count each time they are powered on and use that number as part of a larger sequence number. But if displays cannot retain permanent state this cannot work. Instead, the manager assumes the responsibility for permanent state by generating unique numbers that identify a particular session and the protocol simply ignores packets that correspond to an invalid session.

The Manager must not be responsible for packet reception. To prevent the Manager from becoming stuck because of a hostile display, no portion of the protocol requires the Manager to retransmit a packet. Part of this means that any valid packet that the Manager does receive must be acknowledged in some way to prevent the display from continuously resending packets. The display can keep the protocol running as it will always know when the Manager has received (at least one copy of) a packet. On the Manager side, this means that any packet may be received more than once (if the response was lost) and duplicates must be ignored.

Chapter 3. Data Types

XDMCP packets contain several types of data. Integer values are always stored most significant byte first in the packet ("Big Endian" order). As XDMCP will not be used to transport large quantities of data, this restriction will not substantially hamper the efficiency of any implementation. Also, no padding of any sort will occur within the packets.

Type Name	Length (Bytes)	Description
CARD8	1	A single byte unsigned integer
CARD16	2	Two byte unsigned integer
CARD32	4	Four byte unsigned integer
ARRAY8	n+2	This is actually a CARD16 followed by a collection of CARD8. The value of the CARD16 field (n) specifies the number of CARD8 values to follow
ARRAY16	2*m+1	This is a CARD8 (m) which specifies the number of CARD16 values to follow
ARRAY32	4*l+1	This is a CARD8 (l) which specifies the number of CARD32 values to follow
ARRAYofARRAY8	?	This is a CARD8 which specifies the number of ARRAY8 values to follow.

Chapter 4. Packet Format

All XDMCP packets have the following information:

Length (Bytes)	**Field Type**	**Description**
2	CARD16	version number
2	CARD16	opcode packet header
2	CARD16	n = length of remaining data in bytes
n	???	packet-specific data

The fields are as follows:

Version number	This specifies the version of XDMCP that generated this packet in case changes in this protocol are required. Displays and managers may choose to support older versions for compatibility. This field will initially be one (1).
Opcode	This specifies what step of the protocol this packet represents and should contain one of the following values (encoding provided in section below): **BroadcastQuery**, **Query**, **IndirectQuery**, **ForwardQuery**, **Willing**, **Unwilling**, **Request**, **Accept**, **Decline**, **Manage**, **Refuse**, **Failed**, **KeepAlive** or **Alive**.
Length of data in bytes	This specifies the length of the information following the first 6 bytes. Each packet-type has a different format and will need to be separately length-checked against this value. Because every data item has either an explicit or implicit length, this can be easily accomplished. Packets that have too little or too much data should be ignored.

Packets should be checked to make sure that they satisfy the following conditions:

1. They must contain valid opcodes.
2. The length of the remaining data should correspond to the sum of the lengths of the individual remaining data items.
3. The opcode should be expected (a finite state diagram is given in a later section).
4. If the packet is of type **Manage** or **Refuse**, the Session ID should match the value sent in the preceding **Accept** packet.

Chapter 5. Protocol

Each of the opcodes is described below. Because a given packet type is only ever sent one way, each packet description below indicates the direction. Most of the packets have additional information included beyond the description above. The additional information is appended to the packet header in the order described without padding, and the length field is computed accordingly.

Query

BroadcastQuery

IndirectQuery

Display -> Manager

Additional Fields:

Authentication Names: ARRAYofARRAY8

Specifies a list of authentication names that the display supports. The manager will choose one of these and return it in the **Willing** packet.

Semantics

A `Query` packet is sent from the display to a specific host to ask if that host is willing to provide management services to this display. The host should respond with `Willing` if it is willing to service the display or `Unwilling` if it is not.

A `BroadcastQuery` packet is similar to the `Query` packet except that it is intended to be received by all hosts on the network (or subnetwork). However, unlike `Query` requests, hosts that are not willing to service the display should simply ignore `BroadcastQuery` requests.

An `IndirectQuery` packet is sent to a well known manager that forwards the request to a larger collection of secondary managers using `ForwardQuery` packets. In this way, the collection of managers that respond can be grouped on other than network boundaries; the use of a central manager reduces system administrative overhead. The primary manager may also send a `Willing` packet in response to this packet.

Each packet type has slightly different semantics:

The `Query` packet is destined only for a single host. If the display is instructed to `Query` multiple managers, it will send multiple `Query` packets. The `Query` packet also demands a response from the manager, either `Willing` or `Unwilling`.

The `BroadcastQuery` packet is sent to many hosts. Each manager that receives this packet will not respond with an `Unwilling` packet.

The `IndirectQuery` packet is sent to only one manager with the request that the request be forwarded to a larger list of managers using `ForwardQuery` packets. This list is expected to be maintained at one central site to reduce administrative overhead. The function of this packet type is similar to `BroadcastQuery except that` `BroadcastQuery` is not forwarded.

Valid Responses:

`Willing`, `Unwilling`

Problems/Solutions:

Problem:

Not all managers receive the query packet.

Indication:

None if `BroadcastQuery` or `IndirectQuery` was sent, else failure to receive `Willing`.

Solution:

Repeatedly send the packet while waiting for user to choose a manager.

Timeout/Retransmission policy:

An exponential backoff algorithm should be used here to reduce network load for long-standing idle displays. Start at 2 seconds, back off by factors of 2 to 32 seconds, and discontinue retransmit after 126 seconds. The display should reset the timeout when user-input is detected. In this way, the display will wakeup when touched by the user.

`ForwardQuery`

Primary Manager -> Secondary Manager

Additional Fields:

Client Address: ARRAY8

Specifies the network address of the client display.

Client Port: ARRAY8

Specifies an identification of the client task on the client display.

Authentication Names: ARRAYofARRAY8

Is a duplicate of Authentication Names array that was received in the `IndirectQuery` packet.

Semantics:

When primary manager receives a `IndirectQuery` packet, it is responsible for sending `ForwardQuery` packets to an appropriate list of managers that can provide service to the display using the same network type as the one the original `IndirectQuery` packet was received from. The Client Address and Client Port fields must contain an address that the secondary manager can use to reach the display also using this same network. Each secondary manager sends a `Willing` packet to the display if it is willing to provide service.

`ForwardQuery` packets are similar to `BroadcastQuery` packets in that managers that are not willing to service particular displays should not send a `Unwilling` packet.

Valid Responses:

`Willing`

Problems/Solutions:

Identical to `BroadcastQuery`

Timeout/Retransmission policy:

Like all packets sent from a manager, this packet should never be retransmitted.

`Willing`

Manager -> Display

Additional Fields:

Authentication Name: ARRAY8

Specifies the authentication method, selected from the list offered in the `Query` , `BroadcastQuery` , or `IndirectQuery` packet that the manger expects the display to use in the subsequent `Request` packet. This choice should remain as constant as feasible so that displays that send multiple `Query` packets can use the Authentication Name from any `Willing` packet that arrives.

The display is free to ignore managers that request an insufficient level of authentication.

Hostname: ARRAY8

Is a human readable string describing the host from which the packet was sent. The protocol specifies no interpretation of the data in this field.

Status: ARRAY8

Is a human readable string describing the status of the host. This could include load average/number of users connected or other information. The protocol specifies no interpretation of the data in this field.

Semantics:

A `Willing` packet is sent by managers that may service connections from this display. It is sent in response to either a `Query` , `BroadcastQuery` , or `ForwardQuery` but does not imply a commitment to provide service (for example, it may later decide that it has accepted enough connections already).

Problems/Solutions:

Problem:

`Willing` not received by the display.

Indication:

None if `BroadcastQuery` or `IndirectQuery` was sent, else failure to receive `Willing` .

Solution:

The display should continue to send the query until a response is received.

Timeout/Retransmission policy:

Like all packets sent from the manager to the display, this packet should never be retransmitted.

`Unwilling`

Manager -> Display

Additional Fields:

The Hostname and Status fields as in the `Willing` packet. The Status field should indicate to the user a reason for the refusal of service.

Semantics:

An `Unwilling` packet is sent by managers in response to direct `Query` requests (as opposed to `BroadcastQuery` or `IndirectQuery` requests) if the manager will not accept requests for management. This is typically sent by managers that wish to only service particular displays or that handle a limited number of displays at once.

Problems/Solutions:

Problem:

`Unwilling` not received by the display.

Indication:

Display fails to receive `Unwilling` .

Solution:

The display should continue to send `Query` messages until a response is received.

Timeout/Retransmission policy:

Like all packets sent from the manager to the display, this packet should never be retransmitted.

`Request`

Display -> Manager

Additional Fields:

Display Number: CARD16

Specifies the index of this particular server for the host on which the display is resident. This value will be zero for most autonomous displays.

Connection Types: ARRAY16

Specifies an array indicating the stream services accepted by the display. If the high-order byte in a particular entry is zero, the low-order byte corresponds to an X-protocol host family type.

Connection Addresses: ARRAYofARRAY8

For each connection type in the previous array, the corresponding entry in this array indicates the network address of the display device.

Authentication Name: ARRAY8

Authentication Data: ARRAY8

Specifies the authentication protocol that the display expects the manager to validate itself with. The Authentication Data is expected to contain data that the manager will interpret, modify and use to authenticate itself.

Authorization Names: ARRAYofARRAY8

Specifies which types of authorization the display supports. The manager may decide to reject displays with which it cannot perform authorization.

Manufacturer Display ID: ARRAY8

Can be used by the manager to determine how to decrypt the Authentication Data field in this packet. See the section below on Manufacturer Display ID Format.

Semantics:

A `Request` packet is sent by a display to a specific host to request a session ID in preparation for a establishing a connection. If the manager is willing to service a connection to this display, it should return an `Accept` packet with a valid session ID and should be ready for a subsequent `Manage` request. Otherwise, it should return a `Decline` packet.

Valid Responses:

`Accept` , `Decline`

Problems/Solutions:

Problem:

Request not received by manager.

Indication:

Display timeout waiting for response.

Solution:

Display resends `Request` message.

Problem:

Message received out of order by manager.

Indication:

None.

Solution:

Each time a `Request` is sent, the manager sends the Session ID associated with the next session in the `Accept .` If that next session is not yet started, the manager will simply resend with the same Session ID. If the session is in progress, the manager will reply with a new Session ID; in which case, the `Accept` will be discarded by the display.

Timeout/Retransmission policy:

Timeout after 2 seconds, exponential backoff to 32 seconds. After no more than 126 seconds, give up and report an error to the user.

`Accept`

Manager -> Display

Additional Fields:

Session ID: CARD32

Identifies the session that can be started by the manager.

Authentication Name: ARRAY8

Authentication Data: ARRAY8

Is the data sent back to the display to authenticate the manager. If the Authentication Data is not the value expected by the display, it should terminate the protocol at this point and display an error to the user.

Authorization Name: ARRAY8

Authorization Data: ARRAY8

Is the data sent to the display to indicate the type of authorization the manager will be using in the first call to `XOpenDisplay` after the `Manage` packet is received.

Semantics:

An `Accept` packet is sent by a manager in response to a `Request` packet if the manager is willing to establish a connection for the display. The Session ID is used to identify this connection from any preceding ones and will be used by the display in its subsequent `Manage` packet. The Session ID is a 32-bit number that is incremented each time an `Accept` packet is sent as it must be unique over a reasonably long period of time.

If the authentication information is invalid, a `Decline` packet will be returned with an appropriate `Status` message.

Problems/Solutions:

Problem:

`Accept` or `Decline` not received by display.

Indication:

Display timeout waiting for response to `Request` .

Solution:

Display resends `Request` message.

Problem:

Message received out of order by display.

Indication:

Display receives `Accept` after `Manage` has been sent.

Solution:

Display discards `Accept` messages after it has sent a `Manage` message.

Timeout/Retransmission policy:

Like all packets sent from the manager to the display, this packet should never be retransmitted.

`Decline`

Manager -> Display

Additional Fields:

Status: ARRAY8

Is a human readable string indicating the reason for refusal of service.

Authentication Name: ARRAY8

Authentication Data: ARRAY8

Is the data sent back to the display to authenticate the manager. If the Authentication Data is not the value expected by the display, it should terminate the protocol at this point and display an error to the user.

Semantics:

A `Decline` packet is sent by a manager in response to a `Request` packet if the manager is unwilling to establish a connection for the display. This is allowed even if the manager had responded `Willing` to a previous query.

Problems/Solutions:

Same as for `Accept` .

Timeout/Retransmission policy:

Like all packets sent from a manager to a display, this packet should never be retransmitted.

`Manage`

Display -> Manager

Additional Fields:

Session ID: CARD32

Should contain the nonzero session ID returned in the `Accept` packet.

Display Number: CARD16

Must match the value sent in the previous `Request` packet.

Display Class: ARRAY8

Specifies the class of the display. See the Display Class Format section, which discusses the format of this field.

Semantics:

A `Manage` packet is sent by a display to ask the manager to begin a session on the display. If the Session ID is correct the manager should open a connection; otherwise, it should respond with a `Refuse` or `Failed` packet, unless the Session ID matches a currently running session or a session that has not yet successfully opened the display but has not given up the attempt. In this latter case, the `Manage` packet should be ignored. This will work as stream connections give positive success indication to both halves of the stream, and positive failure indication to the connection initiator (which will eventually generate a `Failed` packet).

Valid Responses:

X connection with correct auth info, `Refuse` , `Failed` .

Problems/Solutions:

Problem:

`Manage` not received by manager.

Indication:

Display timeout waiting for response.

Solution:

Display resends `Manage` message.

Problem:

`Manage` received out of order by manager.

Indication:

Session already in progress with matching Session ID.

Solution:

`Manage` packet ignored.

Indication:

Session ID does not match next Session ID.

Solution:

`Refuse` message is sent.

Problem:

Display cannot be opened on selected stream.

Indication:

Display connection setup fails.

Solution:

`Failed` message is sent including a human readable reason.

Problem:

Display open does not succeed before a second manage packet is received because of a timeout occuring in the display.

Indication:

`Manage` packet received with Session ID matching the session attempting to connect to the display.

Solution:

`Manage` packet is ignored. As the stream connection will either succeed, which will result in an active session, or the stream will eventually give up hope of connecting and send a `Failed` packet; no response to this `Manage` packet is necessary.

Timeout/Retransmission policy:

Timeout after 2 seconds, exponential backoff to 32 seconds. After no more than 126 seconds, give up and report an error to the user.

`Refuse`

Manager -> Display

Additional Fields:

Session ID: CARD32

Should be set to the Session ID received in the `Manage` packet.

Semantics:

A `Refuse` packet is sent by a manager when the Session ID received in the `Manage` packet does not match the current Session ID. The display should assume that it received an old `Accept` packet and should resend its `Request` packet.

Problems/Solutions:

Problem:

Error message is lost.

Indication:

Display times out waiting for new connection, `Refuse` or `Failed` .

Solution:

Display resends `Manage` message.

Timeout/Retransmission policy:

Like all packets sent from a manager to a display, this packet should never be retransmitted.

`Failed`

Manager -> Display

Additional Fields:

Session ID: CARD32

Should be set to the Session ID received in the `Manage` packet.

Status: ARRAY8

Is a human readable string indicating the reason for failure.

Semantics:

A `Failed` packet is sent by a manager when it has problems establishing the initial X connection in response to the `Manage` packet.

Problems/Solutions

Same as for `Refuse` .

`KeepAlive`

Display -> Manager

Additional Fields:

Display Number: CARD16

Set to the display index for the display host.

Session ID: CARD32

Should be set to the Session ID received in the `Manage` packet during the negotiation for the current session.

Sematics:

A `KeepAlive` packet can be sent at any time during the session by a display to discover if the manager is running. The manager should respond with `Alive` whenever it receives this type of packet.

This allows the display to discover when the manager host is no longer running. A display is not required to send `KeepAlive` packets and, upon lack of receipt of `Alive` packets, is not required to perform any specific action.

The expected use of this packet is to terminate an active session when the manager host or network link fails. The display should keep track of the time since any packet has been received from the manager host and use `KeepAlive` packets when a substantial time has elapsed since the most recent packet.

Valid Responses:

`Alive`

Problems/Solutions:

Problem:

Manager does not receive the packet or display does not receive the response.

Indication:

No `Alive` packet is returned.

Solution:

Retransmit the packet with an exponential backoff; start at 2 seconds and assume the host is not up after no less than 30 seconds.

`Alive`

Manager -> Display

Additional Fields:

Session Running: CARD8

Indicates that the session identified by Session ID is currently active. The value is zero if no session is active or one if a session is active.

Session ID: CARD32

Specifies the ID of the currently running session; if any. When no session is active this field should be zero.

Semantics:

An `Alive` packet is sent in response to a `KeepAlive` request. If a session is currently active on the display, the manager includes the Session ID in the packet. The display can use this information to determine the status of the manager.

Chapter 6. Session Termination

When the session is over, the initial connection with the display (the one that acknowledges the `Manage` packet) will be closed by the manager. If only a single session was active on the display, all other connections should be closed by the display and the display should be reset. If multiple sessions are active simultaneously and the display can identify which connections belong to the terminated sesssion, those connections should be closed. Otherwise, all connections should be closed and the display reset only when all sessions have been terminated (that is, all initial connections closed).

The session may also be terminated at any time by the display if the managing host no longer responds to `KeepAlive` packets. The exact time-outs for sending `KeepAlive` packets is not specified in this protocol as the trade off should not be fixed between loading an otherwise idle system with spurious `KeepAlive` packets and not noticing that the manager host is down for a long time.

Chapter 7. State Diagrams

The following state diagrams are designed to cover all actions of both the display and the manager. Any packet that is received out-of-sequence will be ignored.

Display:

start:	User-requested connect to one host -> *query* User-requested connect to some host -> *broadcast* User-requested connect to site host-list -> *indirect*
query:	Send `Query` packet -> *collect-query*
collect-query:	Receive `Willing` -> *start-connection* Receive `Unwilling` -> *stop-connection* Timeout -> *query*
broadcast:	Send `BroadcastQuery` packet -> *collect-broadcast-query*
collect-broadcast-query:	Receive `Willing` -> *update-broadcast-willing* User-requested connect to one host -> *start-connection* Timeout -> *broadcast*
update-broadcast-willing:	Add new host to the host list presented to the user -> *collect-broadcast-query*
indirect:	Send `IndirectQuery` packet -> *collect-indirect-query*
collect-indirect-query:	Receive `Willing` -> *update-indirect-willing* User-requested connect to one host -> *start-connection* Timeout -> *indirect*
update-indirect-willing:	Add new host to the host list presented to the user -> *collect-indirect-query*
start-connection:	Send `Request` packet -> *await-request-response*
await-request-response:	Receive `Accept` -> *manage* Receive `Decline` -> *stop-connection*

Timeout -> *start-connection*

manage: Save Session ID

Send `Manage` packet with Session ID

-> *await-manage-response*

await-manage-response: Receive `XOpenDisplay` : -> *run-session*

Receive `Refuse` with matching Session ID -> *start-connection*

Receive `Failed` with matching Session ID -> *stop-connection*

Timeout -> *manage*

stop-connection: Display cause of termination to user

-> *start*

run-session: Decide to send `KeepAlive` packet -> *keep-alive*

wait close of first display connection

-> *reset-display*

keep-alive: Send `KeepAlive` packet with current Session ID

-> *await-alive*

await-alive: Receive `Alive` with matching Session ID -> *run-session*

Receive `Alive` with nonmatching Session ID or FALSE Session Running -> *reset-display*

Final timeout without receiving `Alive` packet -> *reset-display*

Timeout -> *keep-alive*

reset-display: (if possible) -> close all display connections associated with this session

Last session -> close all display connections

-> *start*

Manager:

idle: Receive `Query` -> *query-respond*

Receive `BroadcastQuery` -> *broadcast-respond*

Receive `IndirectQuery` -> *indirect-respond*

Receive `ForwardQuery` -> *forward-respond* Receive

`Request` -> *request-respond*

Receive `Manage` -> *manage*

An active session terminates -> *finish-session*

Receive `KeepAlive` -> *send-alive*

-> *idle*

query-respond: If willing to manage -> *send-willing*

-> *send-unwilling*

broadcast-respond: If willing to manage -> *send-willing*

-> *idle*

indirect-respond: Send `ForwardQuery` packets to all managers on redirect list

If willing to manage -> *send-willing*

-> *idle*

forward-respond: Decode destination address, if willing to manage -> *send-willing*

-> *idle*

send-willing: Send `Willing` packet

-> *idle*

send-unwilling: Send `Unwilling` packet -> *idle*

request-respond: If manager is willing to allow a session on display -> *accept-session*

-> *decline-session*

accept-session: Generate Session ID and save Session ID, display address, and display number somewhere

Send `Accept` packet

-> *idle*

decline-session: Send `Decline` packet

-> *idle*

manage: If Session ID matches saved Session ID -> *run-session*

If Session ID matches Session ID of session in process of starting up, or currently active session -> *idle*

-> *refuse*

refuse: Send `Refuse` packet

-> *idle*

run-session: Terminate any session in progress

`XOpenDisplay`

Open display succeeds -> *start-session*

-> *failed*

failed: Send `Failed` packet

-> *idle*

start-session: Start a new session

-> *idle*

finish-session: `XCloseDisplay`

-> *idle*

send-alive: Send `Alive` packet containing current status

-> *idle*

Chapter 8. Protocol Encoding

When XDMCP is implemented on top of the Internet User Datagram Protocol (UDP), port number 177 is to be used. When using UDP over IPv4, Broadcast Query packets are sent via UDP broadcast. When using UDP over IPv6, Broadcast Query packets are sent via multicast, either to an address in the IANA registered XDMCP multicast address range of FF0*X*:0:0:0:0:0:0:12B (where the *X* is replaced by a valid scope id) or to a locally assigned multicast address. The version number in all packets will be 1. Packet opcodes are 16-bit integers.

Packet Name	Encoding
BroadcastQuery	1
Query	2
IndirectQuery	3
ForwardQuery	4
Willing	5
Unwilling	6
Request	7
Accept	8
Decline	9
Manage	10
Refuse	11
Failed	12
KeepAlive	13 [a]
Alive	14 [b]

[a] A previous version of this document incorrectly reversed the opcodes of **Alive** and **KeepAlive**.

[b] A previous version of this document incorrectly reversed the opcodes of **Alive** and **KeepAlive**.

Per packet information follows:

`Query`, `BroadcastQuery`, `IndirectQuery`

```
     2     CARD16     version number (always 1)
     2     CARD16     opcode (always Query, BroadcastQuery or IndirectQuery)
     2     CARD16     length
     1     CARD8      number of Authentication Names sent (m)
     2     CARD16     length of first Authentication Name (m1)
     m1   CARD8     first Authentication Name
     ...          Other Authentication Names
```

Note that these three packets are identical except for the opcode field.

```
ForwardQuery
     2     CARD16     version number (always 1)
     2     CARD16     opcode (always ForwardQuery)
     2     CARD16     length
     2     CARD16     length of Client Address (m)
```

```
     m     CARD8     Client Address
     2     CARD16     length of Client Port (n)
     n     CARD8     Client Port
     1     CARD8     number of Authentication Names sent (o)
     2     CARD16     length of first Authentication Name (o1)
     o1     CARD8     first Authentication Name
     ...          Other Authentication Names


Willing
     2     CARD16     version number (always 1)
     2     CARD16     opcode (always Willing)
     2     CARD16     length (6 + m + n + o)
     2     CARD16     Length of Authentication Name (m)
     m     CARD8      Authentication Name
     2     CARD16     Hostname length (n)
     n     CARD8      Hostname
     2     CARD16     Status length (o)
     o     CARD8      Status


Unwilling
     2     CARD16     version number (always 1)
     2     CARD16     opcode (always Unwilling)
     2     CARD16     length (4 + m + n)
     2     CARD16     Hostname length (m)
     m     CARD8      Hostname
     2     CARD16     Status length (n)
     n     CARD8      Status


Request
     2     CARD16     version number (always 1)
     2     CARD16     opcode (always Request)
     2     CARD16     length
     2     CARD16     Display Number
     1     CARD8      Count of Connection Types (m)
     2xm   CARD16     Connection Types
     1     CARD8      Count of Connection Addresses (n)
     2     CARD16     Length of first Connection Address (n1)
     n1     CARD8      First Connection Address
     ...              Other connection addresses
     2     CARD16     Length of Authentication Name (o)
     o     CARD8      Authentication Name
     2     CARD16     Length of Authentication Data (p)
     p     CARD8      Authentication Data
     1     CARD8      Count of Authorization Names (q)
     2     CARD16     Length of first Authorization Name (q1)
     q1     CARD8      First Authorization Name
     ...              Other authorization names
     2     CARD16     Length of Manufacturer Display ID (r)
     r     CARD8      Manufacturer Display ID

Accept
```

```
     2     CARD16     version number (always 1)
     2     CARD16     opcode (always Accept)
     2     CARD16     length (12 + n + m + o + p)
     4     CARD32     Session ID
     2     CARD16     Length of Authentication Name (n)
     n     CARD8      Authentication Name
     2     CARD16     Length of Authentication Data (m)
     m     CARD8      Authentication Data
     2     CARD16     Length of Authorization Name (o)
     o     CARD8      Authorization Name
     2     CARD16     Length of Authorization Data (p)
     p     CARD8      Authorization Data


Decline
     2     CARD16     version number (always 1)
     2     CARD16     opcode (always Decline)
     2     CARD16     length (6 + m + n + o)
     2     CARD16     Length of Status (m)
     m     CARD8      Status
     2     CARD16     Length of Authentication Name (n)
     n     CARD8      Authentication Name
     2     CARD16     Length of Authentication Data (o)
     o     CARD8      Authentication Data


Manage
     2     CARD16     version number (always 1)
     2     CARD16     opcode (always Manage)
     2     CARD16     length (8 + m)
     4     CARD32     Session ID
     2     CARD16     Display Number
     2     CARD16     Length of Display Class (m)
     m     CARD8      Display Class


Refuse
     2     CARD16     version number (always 1)
     2     CARD16     opcode (always Refuse)
     2     CARD16     length (4)
     4     CARD32     Session ID


Failed
     2     CARD16     version number (always 1)
     2     CARD16     opcode (always Failed)
     2     CARD16     length (6 + m)
     4     CARD32     Session ID
     2     CARD16     Length of Status (m)
     m     CARD8      Status


KeepAlive
     2     CARD16     version number (always 1)
     2     CARD16     opcode (always KeepAlive)
```

```
     2     CARD16     length (6)
     2     CARD16     Display Number
     4     CARD32     Session ID


Alive
     2     CARD16     version number (always 1)
     2     CARD16     opcode (always Alive)
     2     CARD16     length (5)
     1     CARD8      Session Running (0: not running 1: running)
     4     CARD32     Session ID (0: not running)
```

Chapter 9. Display Class Format

The Display Class field of the `Manage` packet is used by the display manager to collect common sorts of displays into manageable groups. This field is a string encoded of ISO-LATIN-1 characters in the following format:

`ManufacturerID``-`*`ModelNumber`*

Both elements of this string must exclude characters of the set { `-`, `.`, `:`, `*`, `?`, <*space*> }. The ManufacturerID is a string that should be registered with the X Consortium. The ModelNumber is designed to identify characteristics of the display within the manufacturer's product line. This string should be documented in the users manual for the particular device and should probably not be specifiable by the display user to avoid unexpected configuration errors.

Chapter 10. Manufacturer Display ID Format

To authenticate the manager, the display and manager will share a private key. The manager, then, must be able to discover which key to use for a particular device. The Manufacturer Display ID field of the `Request` packet is intended for this purpose. Typically, the manager host will contain a map between this number and the key. This field is intended to be unique per display, possibly the ethernet address of the display in the form:

```
-Ethernet-8:0:2b:a:f:d2
```

It can also be a string of the form:

```
ManufacturerID-ModelNumber-SerialNumber
```

The ManufacturerID, ModelNumber and SerialNumber are encoded using ISO-LATIN-1 characters, excluding { `-`, `.`, *, ?, *<space>* }

When the display is shipped to a customer, it should include both the Manufacturer Display ID and the private key in the documentation set. This information should not be modifiable by the display user.

Chapter 11. Authentication

In an environment where authentication is not needed, XDMCP can disable authentication by having the display send empty Authentication Name and Authentication Data fields in the `Request` packet. In this case, the manager will not attempt to authenticate itself. Other authentication protocols may be developed, depending on local needs.

In an unsecure environment, the display must be able to verify that the source of the various packets is a trusted manager. These packets will contain authentication information. As an example of such a system, the following discussion describes the "XDM-AUTHENTICATION-1" authentication system. This system uses a 56-bit shared private key, and 64 bits of authentication data. An associated example X authorization protocol "XDM-AUTHORIZATION-1" will also be discussed. The 56-bit key is represented as a 64-bit number in network order (big endian). This means that the first octet in the representation will be zero. When incrementing a 64-bit value, the 8 octets of data will be interpreted in network order (big endian). That is, the last octet will be incremented, subsequent carries propogate towards the first octet.

Assumptions:

1. The display and manager share a private key. This key could be programmed into the display by the manufacturer and shipped with the unit. It must not be available from the display itself, but should allow the value to be modified in some way. The system administrator would be responsible for managing a database of terminal keys.
2. The display can generate random authentication numbers.

Some definitions first:

- $\{D\}^{\kappa}$ = encryption of plain text D by key κ
- $\{\Delta\}^{*\kappa}$ = decryption of crypto text Δ with key κ
- τ = private key shared by display and manager
- ρ = 64 bit random number generated by display
- α = authentication data in XDMCP packets
- σ = per-session private key, generated by manager
- β = authorization data

Encryption will use the Data Encryption Standard (DES, FIPS 46-3); blocks shorter than 64 bits will be zero-filled on the right to 64 bits. Blocks longer than 64 bits will use block chaining:

$\{D\}^{\kappa} = \{D_1\}^{\kappa} \; \{D_2 \text{ xor } \{D_1\}^{\kappa}\}^{\kappa}$

The display generates the first authentication data in the `Request` packet:

$\alpha_{\text{Request}} = \{\rho\}^{\tau}$

For the `Accept` packet, the manager decrypts the initial message and returns α_{Accept}:

$\rho = \{\alpha_{\text{Request}}\}*^{\tau}$

$\alpha_{\text{Accept}} = \{\ \rho + 1\}^{\tau}$

The `Accept` packet also contains the authorization intended for use by the X server. A description of authorization type "XDM-AUTHORIZATION-1" follows.

The `Accept` packet contains the authorization name "XDM-AUTHORIZATION-1". The authorization data is the string:

$\beta_{\text{Accept}} = \{\sigma\}^{\tau}$

To create authorization information for connection setup with the X server using the XDM-AUTHORIZATION-1 authorization protocol, the client computes the following:

N = X client identifier

T = Current time in seconds on client host (32 bits)

$\beta = \{\rho N T\}^{\sigma}$

For TCP connections N is 48 bits long and contains the 32-bit IPv4 address of the client host followed by the 16-bit port number of the client socket. Formats for other connections must be registered. The resulting value, β, is 192 bits of authorization data that is sent in the connection setup to the server. The server receives the packet, decrypts the contents. To accept the connection, the following must hold:

- ρ must match the value generated for the most recent XDMCP negotiation.
- T must be within 1200 seconds of the internally stored time. If no time been received before, the current time is set to T.
- No packet containing the same pair (N, T) can have been received in the last 1200 seconds (20 minutes).