Marlin - Proximity Specification

Version 1.0.1

Final

Source	Marlin Developer Community
Date	January 20, 2012

Notice

Intellectual Property

A commercial implementation of this specification requires a license from the Marlin Trust Management Organization.

Contact Information

Feedback on this specification should be addressed to: editor@marlin-community.com

Contact information for the Marlin Trust Management Organization can be found at: http://www.marlin-trust.com/

Contents

1 Introduction

This specification describes the protocol and Octopus binding mechanism used for proximity check.

1.1 Terminology and Conventions

The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in this specification are to be interpreted as described in IETF RFC 2119 [RFC2119].

These capitalized key words are used to unambiguously specify requirements and behavior that affect the interoperability and security of implementations. When these key words are not capitalized they are meant in their natural-language sense.

All elements of this specification are considered Normative unless specifically marked Informative. All Normative Elements are Mandatory to implement, except where such an element is specifically marked OPTIONAL. Finally, where Normative elements are described as OPTIONAL, they MAY be omitted from an implementation, but when implemented, they MUST be implemented as described.

1.2 Namespaces and Identifiers

This specification defines schemas conforming to XML Schemas [Schema] and normative text to describe the syntax and semantics of XML-encoded objects and protocol messages. In cases of disagreement between the schema documents and the schema listings in this specification, the schema documents take precedence. Note that in some cases the normative text of this specification imposes constraints beyond those indicated by the schema documents.

1.2.1 **Namespaces and Notation**

The following table summarizes the normative schemas defined by this specification, and their XML namespace [XMLns] URIs. These URIs MUST be used by implementations of this specification:

Prefix	XML Namespace	Schema File Name	Description

In addition to the schemas defined by this specification, we leverage existing schemas to achieve our design goals. The following table summarizes the external schemas used in this specification:

Prefix	XML Namespace	Description
xsd:	http://www.w3.org/2001/XMLSchema	[Schema]

1.3 Abbreviations

HTTP	Hypertext Transfer Protocol
NEMO	Networked Environment for Media Orchestration
RTT	Round-Trip Time
SOAP	Simple Object Access Protocol

UDP	User Datagram Protocol
WSDL	Web Services Description Language
XML	Extensible Markup Language

1.4 *References*

1.4.1 Normative References

[8pus]	Octopus DRM Technology Platform Specifications, Version 1.0
[MCS]	Marlin – Core System Specification, version1.3 and its latest errata
[SHA1]	FIPS PUB 180-1. *Secure Hash Standard.* U.S. Department of Commerce/National Institute of Standards and Technology. http://www.itl.nist.gov/fipspubs/fip180-1.htm
[RFC2119]	S. Bradner, *Key words for use in RFCs to Indicate Requirement Levels*, IETF RFC 2119, March 1997. http://www.ietf.org/rfc/rfc2119.txt.

2 Mechanism for Proximity Check

This section defines the mechanism which SHALL be used to check proximity.

- When two implementations are connected through IP, the proximity check SHALL be done by using the Proximity Check Protocol over UDP defined in §2.1.
- When two implementations are connected through USB, the proximity check SHALL always be considered valid without measurement.
- In all other cases, the proximity check SHALL be considered failure.

2.1 Proximity Check Protocol over UDP

2.1.1 Overview

This protocol allows an anchor to check the proximity of a target.
This protocol is asymmetric as the anchor generates a secret seed and is the only one that requires a secure timer. Moreover, the target does not need to trust the anchor. It is also cryptographically efficient requiring only two public key operations.

2.1.2 Preconditions

The target is a NEMO client node and has, as such, a set of NEMO keys and credentials. The anchor is not required to be a NEMO node, as no anchor NEMO credentials are used in this protocol.

2.1.3 Generation of the set R of Q pairs from a Seed S

The set R is obtained from randomly generated seed using the following method:
$R_i = H^{2Q-i}(S)$

H(M) is the digest value of the hash function H over the message M.
$H^n(M) = H(H^{n-1}(M))$ for n>=1 and $H^0(M) = M$

The algorithm used for the hash function H() SHALL be [SHA1].

2.1.4 Protocol Steps

2.1.4.1 Target Setup

The target chooses a UDP port number TargetPort that it is ready to receive ChallengeRequest UDP datagrams on. The target chooses 32-bit number, TargetSessionId, that it can use to differentiate between several concurrent protocol sessions if necessary. The target also chooses the target timing parameters SetupDelay and LoopDelay.
The target sends a TargetSetupRequest message to the anchor. The payload of this message contains TargetSessionId, TargetPort, SetupDelay and LoopDelay.

Upon receiving and validating the TargetSetupRequest, the anchor chooses a random seed S, a maximum loop count Q, and generates a set R of Q pairs of random numbers as specified in §2.1.3~~2.1.2~~. The anchor chooses a UDP port number AnchorPort that it is ready to receive ChallengeResponse UDP datagrams on. Q MUST NOT exceed 254. The anchor chooses 32-bit number, AnchorSessionId, that it can use to differentiate between several concurrent protocol sessions if

necessary. The anchor also chooses the anchor timing parameterTerminationTimeout.
The anchor then sends a TargetSetupResponse reply back to the target. The payload of the reply contains AnchorSessionId, Q, S, AnchorPort and TerminationTimeout.

Upon receiving this reply, the target computes R from Q and S, as specified in §2.1.3~~2.1.2~~..
The target is now ready to receive ChallengeRequest UDP datagrams on TargetPort.

2.1.4.2 RTT Measurement Loop

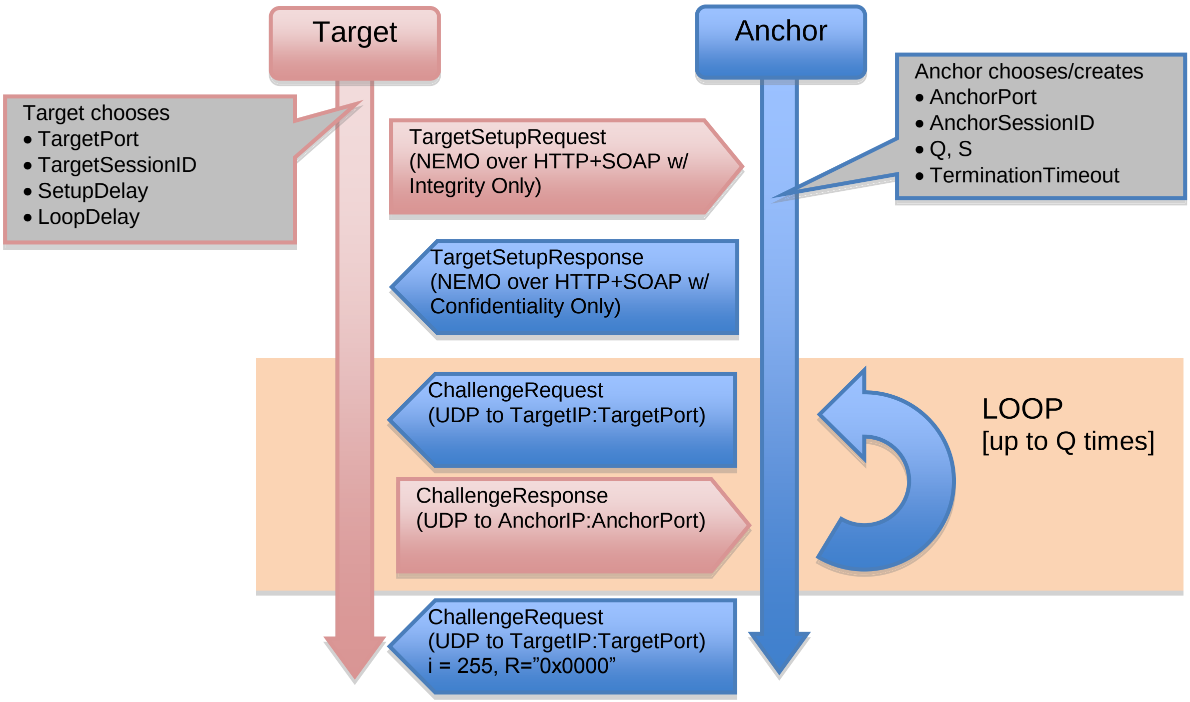
The Anchor MAY perform up to Q RTT measurement loops for each protocol session. Each loop consists of the following steps. The loop counter i is initialized at 0 and is incremented by 1 for each iteration through the loop.

a) The anchor measures T = now
b) The anchor sends to the target a ChallengeRequest UDP datagram on port Target-IP:TargetPort, where Target-IP is the IP address of the TCP endpoint used by the target to send the TargetSetupRequest message. The ChallengeRequest payload contains TargetSessionId (established during the setup step), i, and R_{2*i}
c) The target receives the ChallengeRequest UDP datagram. If the value of R_{2*i} is correct, it responds with ChallengeResponse UDP datagram sent to Anchor-IP:AnchorPort, where Anchor-IP is the IP address of the TCP endpoint to which the TargetSetupRequest was sent. The ChallengeResponse payload contains AnchorSessionId (established during the setup step), i, and R_{2*i+1}.
d) The anchor receives the ChallengeResponse UDP datagram. The anchor measures RTT = now-T. If the value of R_{2*i+1} is correct, the value of RTT is accepted as a valid measurement. The anchor may keep the lowest valid measured RTT along with the date and time of the measurement if it does not terminate the loop before it has exhausted the Q possible iteration.

The anchor MAY terminate the loop at any point before it has exhausted the Q possible iterations (for example if it determines that the lowest measured RTT value is below a certain threshold, or if it receives no response from the target for a long time). When the anchor terminates the loop, it MUST send at least one special “Termination” ChallengeRequest UDP datagram. A “Termination” ChallengeRequest UDP datagram is one where the value of ‘i’ is equal to 255 and the bytes for the ‘R_{2*i}’ field are all set to 0. The anchor MAY send more than one “Termination” ChallengeRequest UDP datagram for redundancy (delivery of UDP datagrams is not guaranteed).

Since the datagrams exchanged during this loop are exchanged over UDP, it is possible that either the target and/or the anchor receive on the same UDP port some datagrams belonging to different sessions. It is therefore important that when a session is started, both the target and the anchor use the TargetSessionId and AnchorSessionId, respectively, to decide which session a received datagram is a part of. All datagrams processed during the RTT measurement loop MUST match the TargetSessionId or AnchorSessionId that were established during the setup phase; all other datagrams MUST NOT be considered part of the session (but they may still be part of a different session happening at the same time).

2.1.5 Sequence Diagram



2.1.6 Timing Parameters

The target timing parameters sent in the TargetSetupRequest message, SetupDelay and LoopDelay, are both expressed in milliseconds.
The anchor MUST wait at least SetupDelay between the transmission of the TargetSetupResponse reply and the transmission of the first ChallengeRequest datagram.
The anchor MUST wait at least LoopDelay between two consecutive ChallengeRequest messages while in the RTT measurement loop.

The anchor timing parameter sent in the TargetSetupResponse message, TerminationTimeout, is expressed in miliseconds. After the setup, the target SHOULD keep listening for UDP datagrams on TargetPort for at least TerminationTimeout after the last received message from the anchor (either the TargetSetupResponse or any ChallengeRequest) unless the protocol can be determined to have terminated (either a “Termination” ChallengeRequest UDP datagram has been received, or the last possible ChallengeRequest, with the loop counter i equal to Q-1, has been received). If no message has been received for more than that amount of time, the protocol is implicitly terminated.

A valid under-threshold RTT measurement MUST be 7 milliseconds or less, unless it is overridden by another specification.

2.1.7 Security Considerations

When engaging in this protocol, care must be taken to follow the following basic requirements.

The anchor MUST choose the seed S with a non-guessable secure random or pseudo-random number generator such that the chances of using the same value S in two separate protocol sessions is infinitesimal.

The RTT measurement loop MUST NOT be repeated with the same value of i during a protocol session.

The protocol MUST be aborted if any unexpected message is received by either party, including:

- If the target receives an incorrect value for R_{2*i} in step c.
- If Q is larger than the maximum allowed value.
- If i is repeated in the loop
- If i exceeds Q

2.1.8 NEMO Security Policies

The TargetSetupRequest request MUST follow the ‘Integrity Only’ policy, as defined in [MCS] §5.2.3.3.
The TargetSetupResponse reply MUST follow the ‘Confidentiality Only’ policy as defined in [MCS] §5.2.4.4.

The identifier for this protocol’s security policy is

urn:marlin:proximityoverudp:1-0:nemo:services:proximity-check:policy:1

2.1.9 Message Encodings

The XML schema for this protocol is defined in the XML Namespace urn:marlin:proximityoverudp:1-0:nemo:services:schemas

A copy of the XML schema and WSDL is in Appendix A.1 and A.2, respectively.

2.1.9.1 Setup

The TargetChallengeRequest and TargetChallengeResponse messages are defined in the XML schema

2.1.9.2 RTT Measurement Loop

2.1.9.2.1 ChallengeRequest

The payload for the ChallengeRequest is the following byte sequence.

Byte	0	1-4	5-24
Description	i	TargetSessionId, encoded as a 32-bit integer in big-endian byte order	R_{2*i}

2.1.9.2.2 ChallengeResponse

The payload for the ChallengeResponse is the following byte sequence.

Byte	0	1-4	5-24
Description	i	AnchorSessionId, encoded as a 32-bit integer in big-endian byte order	R_{2*i+1}

3 Octopus Bindings

3.1 Constraints

An Octopus control can signal that it requires a proximity measurement to be done by carrying a ProximityRequired constraint in an ESB.
The ProximityRequired constraint is in the spatial constrains category, indicated by the local flag SPATIAL_CONSTRAINT as specified in §3.3.2.1 of [8pus].
The constraint entry in the ESB has the following format:

Name	Type	Description
ProximityRequired	Integer	Expected freshness of the proximity measurement in seconds, or 0 if there is no fixed expected freshness. The expected freshness is the amount of time elapsed, at the time of measurement, since the last valid under-threshold proximity measurement of the peer target.

3.2 Control Context

When a running control signals that it requires a proximity measurement by carrying a ProximityRequired constraint described in §3.1, in a NEMO protocol session such as the License Transfer protocol defined in [MCS], the host application SHALL reveal the date of the last valid proximity check between the host and the session's peer NEMO node in the context of that running control. The Host Object path for this value is Sink/Proximity/LastProbe, as specified in §11 of [MCS]. The value is of type Integer, representing the number of minutes elapsed since January 1, 1970 00:00:00 (UTC). The most significant bit MUST be 0. It contains the date of the last valid proximity check of the target that is the NEMO peer of the ongoing NEMO protocol session.

Appendix: XML Schema and WSDL File Names

- **A.1: proximity-check.xsd**
  proximityoverudp.xsd
- **A.2: proximity-check.wsdl**