Homenet Working Group
Internet-Draft
Intended status: Standards Track
Expires: July 9, 2015
M. Stenberg
S. Barth
January 5, 2015

Distributed Node Consensus Protocol
draft-ietf-homenet-dncp-00

Abstract

This document describes the Distributed Node Consensus Protocol (DNCP), a generic state synchronization protocol which uses Trickle and Merkle trees. DNCP is transport agnostic and leaves some of the details to be specified in profiles, which define actual implementable DNCP based protocols.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

DNCP is designed to provide a way for nodes to publish data consisting of an ordered set of TLV (Type-Length-Value) tuples, and to receive the data published by all other reachable DNCP nodes.

DNCP has relatively few requirements for the underlying transport; it requires some way of transmitting either unicast datagram or stream data to a DNCP peer, and if used in multicast mode, a way of sending multicast datagrams. If security is desired and one of the built-in security methods is to be used, support for some TLS-derived transport scheme, such as TLS [RFC5246] on top of TCP, or DTLS [RFC6347] on top of UDP, is also required.

2. Requirements Language

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

3. Terminology

DNCP profile is a definition of a set of rules and values listed in Section 10 specifying the behavior of a DNCP based protocol, such as the used transport method. For readability, any DNCP profile specific parameters with a profile-specific fixed value are prefixed with DNCP_.

DNCP node is a single node which runs a protocol based on a DNCP profile.

DNCP network is a set of DNCP nodes running the same DNCP profile that can reach each other, either via learned shared connections in the underlying network, or using each other's addresses learned via other means. As DNCP exchanges are bidirectional, DNCP nodes connected via only unidirectional links are not considered connected.

Node identifier is an opaque fixed-length identifier of DNCP_NODE_IDENTIFIER_LENGTH bytes which uniquely identifies a DNCP node within a DNCP network.

Link indicates a link-layer media over which directly connected nodes can communicate.

Interface indicates a port of a node that is connected to a particular link.

Connection denotes a locally configured use of DNCP on a DNCP node, that is attached either to an interface, to a specific remote unicast address to be contacted, or to a range of remote unicast addresses that are allowed to contact.

Connection identifier is a 32-bit opaque value, which identifies a particular connection of that particular DNCP node. The value 0 is reserved for DNCP and sub-protocol purposes in the TLVs, and MUST NOT be used to identify an actual connection. This definition is in sync with [RFC3493], as the non-zero small positive integers should comfortably fit within 32 bits.

(DNCP) peer refers to another DNCP node with which a DNCP node communicates directly on a particular connection.

Node data is a set of TLVs published by a node in the DNCP network.

Node state is a set of metadata attributes for node data. It includes a sequence number for versioning, a hash value for comparing and a timestamp indicating the time passed since its last publication. The hash function and the number of bits used are defined in the DNCP profile.

Network state (hash) is a hash value which represents the current state of the network. The hash function and the number of bits used are defined in the DNCP profile. Whenever any node is added, removed or changes its published node data this hash value changes as well. It is calculated over the hash values of each reachable nodes' node data in ascending order of the respective node identifier.

Effective (trust) verdict for a certificate is defined as the verdict with the highest priority within the set of verdicts announced for the certificate in the DNCP network.

4. Data Model

A DNCP node has:

- A timestamp indicating the most recent neighbor graph traversal described in Section 5.4.

A DNCP node has for every DNCP node in the DNCP network:

- A node identifier, which uniquely identifies the node.

- The node data, an ordered set of TLV tuples published by that particular node. This set of TLVs has a well-defined order based on ascending binary content (including TLV type and length). This facilitates linear time state delta processing.

- The latest update sequence number, a 32 bit number that is incremented any time the TLV set is published. For comparison purposes, a looping comparison should be used to avoid problems in case of overflow. An example would be: a < b <=> (a - b) % 2^32 & 2^31 != 0.

- The relative time (in milliseconds) since the current TLV data set with the current update sequence number was published. It is also a 32 bit number on the wire. If this number is close to overflow (greater than 2^32-2^16), a node MUST re-publish its TLVs even if there is no change to avoid overflow of the value. In other words, absent any other changes, the TLV set MUST be re-published roughly every 49 days.

- A timestamp identifying the time it was last reachable based on neighbor graph traversal described in Section 5.4.

Additionally, a DNCP node has a set of connections for which DNCP is configured to be used. For each such connection, a node has:

- A connection identifier.

- An interface, a unicast address of a DNCP peer it should connect with, or a range of addresses from which DNCP peers are allowed to connect.

- A Trickle [RFC6206] instance with parameters I, T, and c.

For each DNCP peer detected on a connection, a DNCP node has:

- The node identifier of the DNCP peer.

o The connection identifier of the DNCP peer.

o The most recent address used by the DNCP peer (in an authenticated message, if security is enabled).

5. Operation

The DNCP protocol consists of Trickle [RFC6206] driven unicast or multicast status messages which indicate the current status of shared TLV data, and additional unicast message exchanges which ensure DNCP peer reachability and synchronize the data when necessary.

If DNCP is to be used on a multicast-capable interface, as opposed to only point-to-point using unicast, a datagram-based transport which supports multicast SHOULD be defined in the DNCP profile to be used for the messages to be sent to the whole link. As this is used only to identify potential new DNCP nodes, and to notify that an unicast exchange should be triggered, the multicast transport does not have to be particularly secure.

5.1. Trickle-Driven Status Update Messages

Each node MUST send either a Long Network State Update message (Section 7.2) or a Short Network State Update message (Section 7.1) every time the connection-specific Trickle algorithm [RFC6206] instance indicates that an update should be sent. The destination address of the message should be multicast in case of an interface which is multicast-capable, or the unicast address of the remote party in case of a point-to-point connection. By default, Long Network State Update messages SHOULD be used, but if it is defined as undesirable for some case by the DNCP profile, Short Network State Update message MUST be sent instead. This may be useful to avoid fragmenting packets to multicast destinations, or for security reasons.

A Trickle state MUST be maintained separately for each connection. The Trickle state for all connections is considered inconsistent and reset if and only if the locally calculated network state hash changes. This occurs either due to a change in the local node's own node data, or due to receipt of more recent data from another node.

The Trickle algorithm has 3 parameters; Imin, Imax and k. Imin and Imax represent the minimum and maximum values for I, which is the time interval during which at least k Trickle updates must be seen on a connection to prevent local state transmission. The actual suggested Trickle algorithm parameters are DNCP profile specific, as described in Section 10.

5.2. Processing of Received Messages

This section describes how received messages are processed. The DNCP profile may specify criteria based on which received messages are ignored. Any 'reply' mentioned in the steps below denotes sending of the specified message via unicast to the originator of the message being processed. If the reply was caused by a multicast message and sent to a link with shared bandwidth it SHOULD be delayed by a random timespan in [0, Imin/2].

Upon receipt of:

Short Network State Update (Section 7.1): If the network state hash within the message differs from the locally calculated network state hash, the receiver MUST reply with a Network State Request message (Section 7.3).

Long Network State Update (Section 7.2):

* If the network state hash within the message matches the locally calculated network state hash, stop processing.

* Otherwise the receiver MUST identify nodes for which local information is outdated (local update sequence number is lower than that within the message), potentially incorrect (local update sequence number matches but the hash of the node data TLV differs) or missing.

* If any such nodes are identified, the receiver MUST reply with one or more Node Data Request message(s) (Section 7.4) containing Request Node Data TLV(s) (Section 8.1.2) for the corresponding nodes.

Network State Request (Section 7.3): the receiver MUST reply with a Long Network State Update (Section 7.2).

Node Data Request (Section 7.4): the receiver MUST reply with the requested data in a Node Data Reply message (Section 7.5). Optionally - if specified by the DNCP profile - multiple replies MAY be sent in order to e.g. keep size of each datagram within the PMTU to the destination. However these replies must be valid stand-alone Node Data Reply messages, with the full state for the particular nodes.

Node Data Reply (Section 7.5): If the message contains Node State TLVs that are more recent than the local state (the received TLV has a higher update sequence number, the node data TLV hash differs from the local one, or local data is missing altogether),

and if the message also contains corresponding Node Data TLVs, the receiver MUST update its locally stored state.

If a message containing Node State TLVs (Section 8.2.3) is received with the node identifier matching the local node identifier and a higher update sequence number than its current local value, or the same update sequence number and a different hash, the node SHOULD re-publish its own node data with an update sequence number 1000 higher than the received one. This may occur normally once due to the local node restarting, and not storing the most recently used update sequence number. If this occurs more than once, the DNCP profile should provide guidance on how to handle these situations as it indicates the existence of a second active node on the network with the same node identifier.

5.3. Adding and Removing Peers

When receiving a message on a connection from an unknown peer:

If it is a unicast message, the remote node MUST be added as a peer on the connection and a Neighbor TLV (Section 8.2.5) MUST be created for it.

If it is a multicast message, the remote node SHOULD be sent a (possibly rate-limited) unicast Network State Request Message (Section 7.3).

If keep-alives are NOT sent by the peer (either DNCP profile does not specify the use of keep-alives, or the particular peer chooses not to send keep-alive messages), some other means MUST be employed to ensure a DNCP peer is present, and when the peer is no longer present, the Neighbor TLV and the local DNCP peer state MUST be removed.

5.4. Purging Unreachable Nodes

When a Neighbor TLV or a whole node is added or removed, the neighbor graph SHOULD be traversed for each node following the bidirectional neighbor relationships. These are identified by looking for Neighbor TLVs on both nodes, that have the other node's identifier in the neighbor node identifier, and local and neighbor connection identifiers swapped. Each node reached should be marked currently reachable.

DNCP nodes MUST be either purged immediately when not marked reachable in a particular graph traversal, or eventually after they have not been marked reachable within DNCP_GRACE_INTERVAL. During the grace period, the nodes that were not marked reachable in the

most recent graph traversal MUST NOT be used for calculation of the network state hash, be provided to any applications that need to use the whole TLV graph, or be provided to remote nodes.

6. Keep-Alive Extension

The Trickle-driven messages provide a mechanism for handling of new peer detection (if applicable) on a connection, as well as state change notifications. Another mechanism may be needed to get rid of old, no longer valid DNCP peers if the transport or lower layers do not provide one.

If keep-alives are not specified in the DNCP profile, the rest of this section MUST be ignored.

A DNCP profile MAY specify either per-connection or per-peer keep-alive support. This document specifies only per-connection keep-alive, thus if per-peer support is required either a lower layer mechanism or a definition within the profile is required.

6.1. Data Model Additions

The following additions to the Data Model (Section 4) are needed to support keep-alive:

Each node MUST have a timestamp which indicates the last time a Network State TLV (Section 8.2.2) was sent for each connection, i.e. on an interface or to the point-to-point peer(s).

Each node MUST have for each peer:

- Last consistent state timestamp: a timestamp which indicates the last time a consistent Network State TLV (Section 8.2.2) was received from the peer. When adding a new peer, it should be initialized to the current time.

6.2. Periodic Keep-Alive Messages

For every connection that a keep-alive is specified for in the DNCP profile, the connection-specific keep-alive interval MUST be maintained. By default, it is DNCP_KEEPALIVE_INTERVAL. If there is a local value that is preferred for that for any reason (configuration, energy conservation, media type, ..), it should be substituted instead. If non-default keep-alive interval is used on any connection, a DNCP node MUST publish appropriate Keep-Alive Interval TLV(s) (Section 8.2.6).

If no traffic containing a Network State TLV (Section 8.2.2) has been sent to a particular connection within the connection-specific keep-alive interval, a Long Network State Update message (Section 7.2) or a Short Network State Update message (Section 7.1) MUST be sent on that connection. The type of message should be chosen based on the considerations in Section 5.1. When such a message is sent, a new Trickle transmission time 't' in [I/2, I] MUST be randomly chosen.

6.3. Received Message Processing Additions

If the received message contains a Network State TLV (Section 8.2.2) which is consistent with the locally calculated network state hash, the Last consistent state timestamp for the peer MUST be updated.

6.4. Neighbor Removal

For every peer on every connection, the connection-specific keep-alive interval must be calculated by looking for Keep-Alive Interval TLVs (Section 8.2.6) published by the node, and if none exist, using the default value of DNCP_KEEPALIVE_INTERVAL. If the peer's last consistent state timestamp has not been updated for at least DNCP_KEEPALIVE_MULTIPLIER times the peer's connection-specific keep-alive interval, the Neighbor TLV for that peer and the local DNCP peer state MUST be removed.

7. Protocol Messages

For point-to-point exchanges, DNCP can run across datagram-based or reliable ordered stream-based transports. If a stream-based transport is used, a 32-bit length-value in network byte order is sent before each message to indicate the number of bytes the following message consists of.

DNCP messages are encoded as a concatenated sequence of Type-Length-Value objects (Section 8). In order to facilitate fast comparing of local state with that in a received message update, all TLVs in every encoding scope (either within the message itself, or within a container TLV) MUST be placed in ascending order based on the binary comparison of both TLV header and value. By design, the TLVs which MUST be present have the lowest available type values, ensuring they will naturally occur at the start of the Protocol Message, resembling a fixed format header.

DNCP profiles MAY add additional TLVs to the message specified here, or even define additional messages as needed.

7.1. Short Network State Update Message

The Short Network State Update Message is used to announce the sender's view of the network state using multicast.

The following TLVs MUST be present:

- One Node Connection TLV (Section 8.2.1) identifying the originating node and connection.

- One Network State TLV (Section 8.2.2) containing the network state hash as calculated by the sender.

The Short Network Status update message MUST NOT contain any Node State TLV(s) (Section 8.2.3).

7.2. Long Network State Update Message

The Long Network State Update Message is used to announce the sender's view of the network state and all node states using multicast or unicast.

The following TLVs MUST be present:

- One Node Connection TLV (Section 8.2.1) identifying the originating node and connection.

- One Network State TLV (Section 8.2.2) containing the network state hash as calculated by the sender.

- One or more Node State TLVs (Section 8.2.3) containing the node state of DNCP nodes as currently known to the sender.

The Long Network State Update message MUST include the corresponding Node State TLV (Section 8.2.3) for each Node Data TLV used to calculate the network state hash.

7.3. Network State Request Message

The Network State Request message is used to request the recipient's view of the network state and all node states currently known to it.

The following TLVs MUST be present:

- One Node Connection TLV (Section 8.2.1) identifying the originating node and connection.

- o One Request Network State TLV (Section 8.1.1) indicating the type of request.

7.4. Node Data Request Message

The Node Data Request message is used to request the node state and data of one or more DNCP nodes in the network.

The following TLVs MUST be present:

- o One Node Connection TLV (Section 8.2.1) identifying the originating node and connection.

- o One or more Request Node Data TLVs (Section 8.1.2) indicating the nodes for which state and data is requested.

7.5. Node Data Reply Message

The Node Data Request message is used to provide the node data of one or more DNCP nodes in the network.

The following TLVs MUST be present:

- o One Node Connection TLV (Section 8.2.1) identifying the originating node and connection.

- o One or more Node State TLV (Section 8.2.3) and Node Data TLV (Section 8.2.4) pairs with matching node identifiers for each node previously requested in a Node Data Request message (Section 7.4).

8. Type-Length-Value Objects

Each TLV is encoded as a 2 byte type field, followed by a 2 byte length field (of the value, excluding header; 0 means no value) followed by the value itself (if any). Both type and length fields in the header as well as all integer fields inside the value - unless explicitly stated otherwise - are represented in network byte order. Zero padding bytes MUST be added up to the next 4 byte boundary if the length is not divisible by 4. These padding bytes MUST NOT be included in the length field.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|            Type               |           Length              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                             Value                             |
|                     (variable # of bytes)                     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

For example, type=123 (0x7b) TLV with value 'x' (120 = 0x78) is encoded as: 007B 0001 7800 0000.

Notation:

.. = octet string concatenation operation.

H(x) = non-cryptographic hash function specified by DNCP profile.

8.1. Request TLVs

8.1.1. Request Network State TLV

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Type: REQ-NETWORK-STATE (2)  |           Length: 0           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

This TLV is used to identify a Network State Request message (Section 7.3).

8.1.2. Request Node Data TLV

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    Type: REQ-NODE-DATA (3)    |          Length: >0           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Node Identifier                        |
|                  (length fixed in DNCP profile)               |
...
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

This TLV is used within a Node Data Request message (Section 7.4) to request node state and node data for the node with matching node identifier, if any, to be included in a subsequent Node Data Reply message (Section 7.5).

8.2. Data TLVs

8.2.1. Node Connection TLV

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Type: NODE-CONNECTION (1)   |          Length: > 4          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Node Identifier                        |
|                  (length fixed in DNCP profile)               |
...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                     Connection Identifier                     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

This TLV identifies both the local node's node identifier, as well as the particular connection identifier. It MUST be sent in all messages.

8.2.2. Network State TLV

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    Type: NETWORK-STATE (10)   |          Length: > 0          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     H(H(node data TLV 1) .. [...] .. H(node data TLV N))      |
|                  (length fixed in DNCP profile)               |
...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

This TLV contains the current locally calculated network state hash. The network state hash is derived by calculating the hash value for each currently reachable node's Node Data TLV, concatenating said hash values based on the ascending order of their corresponding node identifiers, and hashing the resulting concatenated hash values.

8.2.3. Node State TLV

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      Type: NODE-STATE (11)    |          Length: > 8          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Node Identifier                        |
|                  (length fixed in DNCP profile)               |
...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Update Sequence Number                     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                Milliseconds since Origination                 |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        H(node data TLV)                       |
|                  (length fixed in DNCP profile)               |
...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

This TLV represents the local node's knowledge about the published state of a node in the DNCP network identified by the node identifier field in the TLV.

The whole network should have roughly same idea about the time since origination of any particular published state. Therefore every node, including the originating one, MUST increment the time whenever it needs to send a Node State TLV for an already published Node Data TLV. This age value is not included within the Node Data TLV, however, as that is immutable and used to detect changes in the network state.

8.2.4. Node Data TLV

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      Type: NODE-DATA (12)     |         Length: > 4           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        node identifier                        |
|                  (length fixed in DNCP profile)               |
...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Update Sequence Number                     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|            Nested TLVs containing node information            |
```

8.2.5. Neighbor TLV (within Node Data TLV)

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|       Type: NEIGHBOR (13)     |          Length: > 8          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    neighbor node identifier                   |
|                  (length fixed in DNCP profile)               |
...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                 Neighbor Connection Identifier                |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                   Local Connection Identifier                 |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

This TLV indicates that the node in question vouches that the specified neighbor is reachable by it on the specified local connection. The presence of this TLV at least guarantees that the node publishing it has received traffic from the neighbor recently. For guaranteed up-to-date bidirectional reachability, the existence of both nodes' matching Neighbor TLVs should be checked.

8.2.6. Keep-Alive Interval TLV (within Node Data TLV)

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Type: KEEP-ALIVE-INTERVAL (14)|          Length: 8            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                      Connection Identifier                    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                            Interval                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

This TLV indicates a non-default interval being used to send keep-alive messages specified in Section 6.

Connection identifier is used to identify the particular connection for which the interval applies. If 0, it applies for ALL connections for which no specific TLV exists.

Interval specifies the interval in milliseconds at which the node sends keep-alives. A value of zero means no keep-alives are sent at all; in that case, some lower layer mechanism that ensures presence of nodes MUST be available and used.

8.3. Custom TLV (within/without Node Data TLV)

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Type: CUSTOM-DATA (15)    |          Length: > 0          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                             H(URI)                            |
|                   (length fixed in DNCP profile)              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           Opaque Data                         |
```

This TLV can be used to contain anything; the URI used should be under control of the author of that specification. For example:

V = H('http://example.com/author/json-for-dncp') .. '{"cool": "json extension!"}'

or

V = H('mailto:author@example.com') .. '{"cool": "json extension!"}'

9. Security and Trust Management

If specified in the DNCP profile, either DTLS [RFC6347] or TLS [RFC5246] may be used to authenticate and encrypt either some (if specified optional in the profile), or all unicast traffic. The following methods for establishing trust are defined, but it is up to the DNCP profile to specify which ones may, should or must be supported.

9.1. Pre-Shared Key Based Trust Method

A PSK-based trust model is a simple security management mechanism that allows an administrator to deploy devices to an existing network by configuring them with a pre-defined key, similar to the configuration of an administrator password or WPA-key. Although limited in nature it is useful to provide a user-friendly security mechanism for smaller networks.

9.2. PKI Based Trust Method

A PKI-based trust-model enables more advanced management capabilities at the cost of increased complexity and bootstrapping effort. It however allows trust to be managed in a centralized manner and is therefore useful for larger networks with a need for an authoritative trust management.

9.3. Certificate Based Trust Consensus Method

The certificate-based consensus model is designed to be a compromise between trust management effort and flexibility. It is based on X.509-certificates and allows each DNCP node to provide a verdict on any other certificate and a consensus is found to determine whether a node using this certificate or any certificate signed by it is to be trusted.

The current effective trust verdict for any certificate is defined as the one with the highest priority from all verdicts announced for said certificate at the time.

9.3.1. Trust Verdicts

Trust Verdicts are statements of DNCP nodes about the trustworthiness of X.509-certificates. There are 5 possible verdicts in order of ascending priority:

0 Neutral : no verdict exists but the DNCP network should determine one.

1 Cached Trust : the last known effective verdict was Configured or Cached Trust.

2 Cached Distrust : the last known effective verdict was Configured or Cached Distrust.

3 Configured Trust : trustworthy based upon an external ceremony or configuration.

4 Configured Distrust : not trustworthy based upon an external ceremony or configuration.

Verdicts are differentiated in 3 groups:

- Configured verdicts are used to announce explicit verdicts a node has based on any external trust bootstrap or predefined relation a node has formed with a given certificate.

- Cached verdicts are used to retain the last known trust state in case all nodes with configured verdicts about a given certificate have been disconnected or turned off.

- The Neutral verdict is used to announce a new node intending to join the network so a final verdict for it can be found.

The current effective trust verdict for any certificate is defined as the one with the highest priority within the set of verdicts + announced for the certificate in the DNCP network. A node MUST be trusted for participating in the DNCP network if and only if the current effective verdict for its own certificate or any one in its certificate hierarchy is (Cached or Configured) Trust and none of the certificates in its hierarchy have an effective verdict of (Cached or Configured) Distrust. In case a node has a configured verdict, which is different from the current effective verdict for a certificate, the current effective verdict takes precedence in deciding trustworthiness. Despite that, the node still retains and announces its configured verdict.

9.3.2. Trust Cache

Each node SHOULD maintain a trust cache containing the current effective trust verdicts for all certificates currently announced in the DNCP network. This cache is used as a backup of the last known state in case there is no node announcing a configured verdict for a known certificate. It SHOULD be saved to a non-volatile memory at reasonable time intervals to survive a reboot or power outage.

Every time a node (re)joins the network or detects the change of an effective trust verdict for any certificate, it will synchronize its cache, i.e. store new effective verdicts overwriting any previously cached verdicts. Configured verdicts are stored in the cache as their respective cached counterparts. Neutral verdicts are never stored and do not override existing cached verdicts.

9.3.3. Announcement of Verdicts

A node SHOULD always announce any configured trust verdicts it has established by itself, and it MUST do so if announcing the configured trust verdict leads to a change in the current effective verdict for the respective certificate. In absence of configured verdicts, it MUST announce cached trust verdicts it has stored in its trust cache, if one of the following conditions applies:

o The stored verdict is Cached Trust and the current effective verdict for the certificate is Neutral or does not exist.

o The stored verdict is Cached Distrust and the current effective verdict for the certificate is Cached Trust.

A node rechecks these conditions whenever it detects changes of announced trust verdicts anywhere in the network.

Upon encountering a node with a hierarchy of certificates for which there is no effective verdict, a node adds a Neutral Trust-Verdict-TLV to its node data for all certificates found in the hierarchy, and publishes it until an effective verdict different from Neutral can be found for any of the certificates, or a reasonable amount of time (10 minutes is suggested) with no reaction and no further authentication attempts has passed. Such verdicts SHOULD also be limited in rate and number to prevent denial-of-service attacks.

Trust verdicts are announced using Trust-Verdict TLVs:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Type: Trust-Verdict (16)    |        Length: 37-100         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    Verdict    |                 (reserved)                    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
|                                                               |
|                                                               |
|                      SHA-256 Fingerprint                      |
|                                                               |
|                                                               |
|                                                               |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          Common Name                          |
```

Verdict represents the numerical index of the verdict.

(reserved) is reserved for future additions and MUST be set to 0 when creating TLVs and ignored when parsing them.

SHA-256 Fingerprint contains the SHA-256 [RFC6234] hash value of the certificate in DER-format.

Common Name contains the variable-length (1-64 bytes) common name of the certificate. Final byte MUST have value of 0.

9.3.4. Bootstrap Ceremonies

The following non-exhaustive list of methods describes possible ways to establish trust relationships between DNCP nodes and node certificates. Trust establishment is a two-way process in which the existing network must trust the newly added node and the newly added node must trust at least one of its neighboring nodes. It is therefore necessary that both the newly added node and an already

trusted node perform such a ceremony to successfully introduce a node into the DNCP network. In all cases an administrator MUST be provided with external means to identify the node belonging to a certificate based on its fingerprint and a meaningful common name.

9.3.4.1. Trust by Identification

A node implementing certificate-based trust MUST provide an interface to retrieve the current set of effective trust verdicts, fingerprints and names of all certificates currently known and set configured trust verdicts to be announced. Alternatively it MAY provide a companion DNCP node or application with these capabilities with which it has a pre-established trust relationship.

9.3.4.2. Preconfigured Trust

A node MAY be preconfigured to trust a certain set of node or CA certificates. However such trust relationships MUST NOT result in unwanted or unrelated trust for nodes not intended to be run inside the same network (e.g. all other devices by the same manufacturer).

9.3.4.3. Trust on Button Press

A node MAY provide a physical or virtual interface to put one or more of its internal network interfaces temporarily into a mode in which it trusts the certificate of the first DNCP node it can successfully establish a connection with.

9.3.4.4. Trust on First Use

A node which is not associated with any other DNCP node MAY trust the certificate of the first DNCP node it can successfully establish a connection with. This method MUST NOT be used when the node has already associated with any other DNCP node.

10. DNCP Profile-Specific Definitions

Each DNCP profile MUST define following:

- o How the messages are secured:

  * Not at all,

  * optionally or always with the TLS scheme defined here using one or more of the methods, or

  * with something else.

Given that links with DNCP nodes can be sufficiently secured or isolated it is possible to run DNCP in a secure manner without using any form of authentication or encryption.

- Unicast and optionally multicast transport protocol(s) to be used. If TLS scheme within this document is to be used security, TLS or DTLS support for at least the unicast transport protocol is mandatory.

- Transport protocols' parameters such as port numbers to be used, or multicast address to be used. Unicast, multicast, and secure unicast may each require different parameters, if applicable.

- When receiving messages, what sort of messages are dropped, as specified in Section 5.2.

- What is the criteria for sending Trickle-based Long Network State Update message (Section 7.2) on an interface or to a DNCP peer.

- How to deal with node identifier collision as described in Section 5.2. Main options are either for one or both nodes to assign new node identifiers to themselves, or to notify someone about a fatal error condition in the DNCP network.

- Imin, Imax and k ranges to be suggested for implementations to be used in the Trickle algorithm. The Trickle algorithm does not require these to be same across all implementations for it to work, but similar orders of magnitude helps implementations of a DNCP profile to behave more consistently and to facilitate estimation of lower and upper bounds for behavior of the network.

- Hash function H(x) to be used, and how many bits of the input are actually used. The chosen hash function is used to handle both hashing of node specific data, and network state hash, which is a hash of node specific data hashes. SHA-256 defined in [RFC6234] is the recommended default choice.

- DNCP_NODE_IDENTIFIER_LENGTH: The fixed length of a node identifier (in bytes).

- DNCP_GRACE_INTERVAL: How long node data for unreachable nodes is kept.

- Whether to send keep-alives, and if so, on an interface, using multicast, or directly using unicast to peers. Keep-alive has also associated parameters:

* DNCP_KEEPALIVE_INTERVAL: How often keep-alive messages are to be sent by default (if enabled).

* DNCP_KEEPALIVE_MULTIPLIER: How many times the DNCP_KEEPALIVE_INTERVAL (or peer-supplied keep-alive interval value) a node may not be heard from to be considered still valid.

11. Security Considerations

DNCP profiles may use multicast to indicate DNCP state changes and for keep-alive purposes. However, no actual data TLVs will be sent across that channel. Therefore an attacker may only learn hash values of the state within DNCP and may be able to trigger unicast synchronization attempts between nodes on a local link this way. A DNCP node should therefore rate-limit its reactions to multicast packets.

When using DNCP to bootstrap a network, PKI based solutions may have issues when validating certificates due to potentially unavailable accurate time, or due to inability to use the network to either check Certifcate Revocation Lists or perform on-line validation.

The Certificate-based trust consensus mechanism defined in this document allows for a consenting revocation, however in case of a compromised device the trust cache may be poisoned before the actual revocation happens allowing the distrusted device to rejoin the network using a different identity. Stopping such an attack might require physical intervention and flushing of the trust caches.

12. IANA Considerations

IANA should set up a registry for DNCP TLV types, with the following initial contents:

0: Reserved (should not happen on wire)

1: Node connection

2: Request network state

3: Request node data

4-9: Reserved for DNCP profile use

10: Network state

11: Node state

12: Node data

13: Neighbor

14: Keep-alive interval

15: Custom

16: Trust-Verdict

17-31: Reserved for future DNCP versions.

192-255: Reserved for per-implementation experimentation. The nodes using TLV types in this range SHOULD use e.g. Custom TLV to identify each other and therefore eliminate potential conflict caused by potential different use of same TLV numbers.

For the rest of the values (32-191, 256-65535), policy of 'standards action' should be used.

13. References

13.1. Normative references

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC6206] Levis, P., Clausen, T., Hui, J., Gnawali, O., and J. Ko, "The Trickle Algorithm", RFC 6206, March 2011.

[RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, January 2012.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.

13.2. Informative references

[RFC3493] Gilligan, R., Thomson, S., Bound, J., McCann, J., and W. Stevens, "Basic Socket Interface Extensions for IPv6", RFC 3493, February 2003.

[RFC6234] Eastlake, D. and T. Hansen, "US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)", RFC 6234, May 2011.

Appendix A. Some Outstanding Issues

Should per-peer keep-alives be specified here? They are essentially constant unicast keep-alives, as opposed to unicast OR multicast per-connection ones are.

Appendix B. Some Obvious Questions and Answers

Q: Should there be nested container syntax that is actually self-describing? (i.e. type flag that indicates container, no body except sub-TLVs?)

A: Not for now, but perhaps valid design.. TBD.

Q: Add third case for multicast - 'medium' network state, which is 'long' one, but partial?

A: Drops typical convergence on large networks 5->3 packets, at expense of some specification/implementation complexity. However, as anything else than short network state leaks information via multicast, it does not seem worth it as secure protocols probably want to prevent multicast sending of anything else than short network state in any case.

Q: 32-bit connection id?

A: Here, it would save 32 bits per neighbor if it was 16 bits (and less is not realistic). However, TLVs defined elsewhere would not seem to even gain that much on average. 32 bits is also used for ifindex in various operating systems, making for simpler implementation.

Q: Why not doing (performance thing X, Y or Z)?

A: This is designed mostly to be minimal (only timers Trickle ones; everything triggered by Trickle-driven messages or local state changes). However, feel free to suggest better (even more minimal) design which works.

Appendix C. Changelog

draft-stenberg-homenet-dncp-00: Split from pre-version of draft-ietf-homenet-hncp-03 generic parts. Changes that affect implementations:

- TLVs were renumbered.

- TLV length does not include header (=-4). This facilitates e.g. use of DHCPv6 option parsing libraries (same encoding), and

reduces complexity (no need to handle error values of length less than 4).

- o Trickle is reset only when locally calculated network state hash is changes, not as remote different network state hash is seen. This prevents e.g. attacks by multicast with one multicast packet to force Trickle reset on every interface of every node on a link.

- o Instead of 'ping', use 'keep-alive' (optional) for dead peer detection. Different message used!

Appendix D. Draft Source

As usual, this draft is available at https://github.com/fingon/ietf-drafts/ in source format (with nice Makefile too). Feel free to send comments and/or pull requests if and when you have changes to it!

Appendix E. Acknowledgements

Thanks to Ole Troan, Pierre Pfister, Mark Baugher, Mark Townsley, Juliusz Chroboczek and Jiazi Yi for their contributions to the draft.

Authors' Addresses

Markus Stenberg
Helsinki 00930
Finland

Email: markus.stenberg@iki.fi

Steven Barth
Halle 06114
Germany

Email: cyrus@openwrt.org