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

June 3, 2015

Distributed Node Consensus Protocol
draft-ietf-homenet-dncp-05

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

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on December 5, 2015.

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 validates the set of data within it by ensuring that it is reachable via nodes that are currently accounted for; therefore, unlike Time-To-Live (TTL) based solutions, it does not require periodic re-publishing of the data by the nodes. On the other hand, it does require the topology to be visible to every node that wants to be able to identify unreachable nodes and therefore remove old, stale data. Another notable feature is the use of Trickle to send status updates as it makes the DNCP network very thrifty when there are no updates. DNCP is most suitable for data that changes only gradually to gain the maximum benefit from using Trickle, and if more rapid state exchanges are needed, something point-to-point is recommended and just e.g. publishing of addresses of the services within DNCP.

DNCP has relatively few requirements for the underlying transport; it requires some way of transmitting either unicast datagram or stream data to a 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 — a definition of the set of rules and values listed in Section 9 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	a single node which runs a protocol based on a DNCP profile.
DNCP network	a set of DNCP nodes running the same DNCP profile that can reach each other, either via discovered connectivity 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.
DNCP message	an abstract concept - when using a reliable stream transport, the whole stream of TLVs can be considered a single message, with new TLVs becoming one by one available once they have been fully received. On a datagram transport, each individual datagram is considered a separate message.
Node identifier	an opaque fixed-length identifier consisting of DNCP_NODE_IDENTIFIER_LENGTH bytes which uniquely identifies a DNCP node within a DNCP network.
Link	a link-layer media over which directly connected nodes can communicate.
Interface	a port of a node that is connected to a particular link.
Endpoint	a locally configured use of DNCP on a DNCP node. It is attached either to an interface, a specific remote unicast address to be contacted, or a range of remote unicast addresses that are allowed to contact.
Endpoint identifier	a 32-bit opaque value, which identifies a particular endpoint of that particular DNCP node. The value 0 is reserved for DNCP and sub-protocol purposes and MUST NOT be used to identify an actual endpoint. This definition is in sync with the interface index definition in [RFC3493], as the non-zero small positive integers should comfortably fit within 32 bits.
Peer	another DNCP node with which a DNCP node

	communicates directly using a particular local and remote endpoint pair.
Node data	a set of TLVs published by a node in the DNCP network. The whole node data is owned by the node that publishes it, and it MUST be passed along as-is, including TLVs unknown to the forwarder.
Node state	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	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 a node is added, removed or updates its published node data this hash value changes as well. It is calculated over each reachable nodes' update number concatenated with the hash value of its node data. For calculation these tuples are sorted in ascending order of the respective node's node identifier.
Trust verdict	a statement about the trustworthiness of a certificate announced by a node participating in the certificate based trust consensus mechanism.
Effective trust verdict	the trust verdict with the highest priority within the set of trust verdicts announced for the certificate in the DNCP network.
Neighbor graph	the undirected graph of DNCP nodes produced by retaining only bidirectional peer relationships between nodes.

4. Data Model

A DNCP node has:

- A timestamp indicating the most recent neighbor graph traversal described in Section 5.4.
- A data structure containing data about the most recently sent Request Network State TLVs (Section 7.1.1). The simplest option

is keeping a timestamp of the most recent request (see Section 5.2).

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

- Node identifier: the unique identifier of the node.
- Node data: the ordered set of TLV tuples published by that particular node. This set of TLVs MUST be strictly ordered based on ascending binary content (including TLV type and length). This facilitates linear time state delta processing.
- Latest update sequence number: the 32-bit sequence 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.
- Relative time delta: the 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. In other words, absent any other changes, the TLV set MUST be re-published roughly every 49 days.
- Timestamp: the time it was last reachable based on neighbor graph traversal described in Section 5.4.

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

- Endpoint identifier: the 32-bit opaque value uniquely identifying it.
- Trickle [RFC6206] instance: the endpoint's individual trickle instance with parameters I, T, and c.

and one (or more) of the following:

- Interface: the assigned local network interface.
- Unicast address: the DNCP node it should connect with.
- Range of addresses: the DNCP nodes that are allowed to connect.

For each remote (peer, endpoint) pair detected on a local endpoint, a DNCP node has:

- Node identifier: the unique identifier of the peer.
- Endpoint identifier: the unique endpoint identifier used by the peer.
- Peer address: the most recently used address of the peer (authenticated and authorized, if security is enabled).

5. Operation

The DNCP protocol consists of Trickle [RFC6206] driven unicast or multicast status payloads which indicate the current status of shared TLV data and additional unicast exchanges which ensure 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 TLVs to be sent to the whole link. As this is used only to identify potential new DNCP nodes and to notify that a unicast exchange should be triggered, the multicast transport does not have to be particularly secure.

To form bidirectional peer relationships DNCP requires identification of the endpoints used for communication. A DNCP node therefore MUST include an Endpoint TLV (Section 7.2.1) in each message intended to maintain a DNCP peer relationship.

5.1. Trickle-Driven Status Updates

When employing unreliable transport, each node MUST send a Network State TLV (Section 7.2.2) every time the endpoint-specific Trickle algorithm [RFC6206] instance indicates that an update should be sent. Multicast MUST be employed on a multicast-capable interface; otherwise, unicast can be used as well. If possible, most recent, recently changed, or best of all, all known Node State TLVs (Section 7.2.3) SHOULD be also included, unless it is defined as undesirable for some reason by the DNCP profile. Avoiding sending some or all Node State TLVs may make sense to avoid fragmenting packets to multicast destinations, or for security reasons. If the DNCP profile supports dense broadcast link optimization (Section 6.2), and if a node does not have the highest node identifier on a link, the endpoint may be in a unicast mode in which multicast traffic is only listened to. In that mode, multicast updates MUST NOT be sent.

A Trickle state MUST be maintained separately for each endpoint which employs unreliable transport. The Trickle state for all endpoints 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 an endpoint to prevent local state transmission. The actual suggested Trickle algorithm parameters are DNCP profile specific, as described in Section 9.

5.2. Processing of Received TLVs

This section describes how received TLVs are processed. The DNCP profile may specify criteria based on which particular TLVs are ignored. Any 'reply' mentioned in the steps below denotes sending of the specified TLV(s) via unicast to the originator of the TLV being processed. If the TLV being replied to was received via multicast and it was sent to a link with shared bandwidth, the reply SHOULD be delayed by a random timespan in [0, Imin/2]. Sending of replies SHOULD be rate-limited by the implementation, and in case of excess load (or some other reason), a reply MAY be omitted altogether.

A DNCP node MUST reply to a request from any valid address, as specified by a given DNCP profile, whether this address is known to be the address of a neighbour or not. (This provision satisfies the needs of monitoring or other host software that needs to discover the DNCP topology without adding to the state in the network.)

Upon receipt of:

- Request Network State TLV (Section 7.1.1): The receiver MUST reply with a Network State TLV (Section 7.2.2) and a Node State TLV (Section 7.2.3) for each node data used to calculate the network state hash. The Node State TLVs SHOULD NOT contain the optional node data part.

- Request Node State TLV (Section 7.1.2): If the receiver has node data for the corresponding node, it MUST reply with a Node State TLV (Section 7.2.3) for the corresponding node. The optional node data part MUST be included in the TLV.

- Network State TLV (Section 7.2.2): If the network state hash differs from the locally calculated network state hash, and the receiver is unaware of any particular node state differences with the sender, the receiver MUST reply with a Request Network State TLV (Section 7.1.1). These replies MUST be rate limited to only

at most one reply per link per unique network state hash within Imin. The simplest way to ensure this rate limit is a timestamp indicating requests, and sending at most one Request Network State TLV (Section 7.1.1) per Imin. To facilitate faster state synchronization, if a Request Network State TLV is sent in a reply, a local, current Network State TLV SHOULD be also sent.

- Node State TLV (Section 7.2.3):
  - If the node identifier matches the local node identifier and the TLV has 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 another active node with the same node identifier.
  - If the node identifier does not match the local node identifier, and the local information is outdated for the corresponding node (local update sequence number is lower than that within the TLV), potentially incorrect (local update sequence number matches but the node data hash differs), or the data is altogether missing:
    - If the TLV does not contain node data, and the hash of the node data differs, the receiver MUST reply with a Request Node State TLV (Section 7.1.2) for the corresponding node.
    - Otherwise the receiver MUST update its locally stored state for that node (node data if present, update sequence number, relative time) to match the received TLV.
- Any other TLV: TLVs not recognized by the receiver MUST be silently ignored.

If secure unicast transport is configured for an endpoint, any Node State TLVs received via insecure multicast MUST be silently ignored.

5.3. Adding and Removing Peers

When receiving a Node Endpoint TLV (Section 7.2.1) on an endpoint from an unknown peer:

- If it comes via unicast, the remote node MUST be added as a peer on the endpoint and a Neighbor TLV (Section 7.3.2) MUST be created for it.

- If it comes via multicast, the node SHOULD be sent a (possibly rate-limited) unicast Request Network State TLV (Section 7.1.1).

If keep-alives specified in Section 6.1 are NOT sent by the peer (either the DNCP profile does not specify the use of keep-alives or the particular peer chooses not to send keep-alives), some other means MUST be employed to ensure its presence. When the peer is no longer present, the Neighbor TLV and the local DNCP peer state MUST be removed.

If the DNCP profile supports dense broadcast link optimization (Section 6.2), and if a node does not have the highest node identifier on a link, the endpoint may be in a unicast mode in which multicast traffic is only listened to. In that mode, all peers except the one with the highest node identifier MUST NOT have Neighbor TLV (Section 7.3.2) published nor any local state.

5.4. Purging Unreachable Nodes

DNCP validates the set of data within it by ensuring that it is reachable via nodes that are currently accounted for; therefore, unlike Time-To-Live (TTL) based solutions, it does not require periodic re-publishing of the data by the nodes. On the other hand, it does require the topology to be visible to every node that wants to be able to identify unreachable nodes and therefore remove old, stale data.

When a Neighbor TLV or a whole node is added or removed, the neighbor graph SHOULD be traversed, starting from the local node. The edges to be traversed 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 endpoint 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. Optional Extensions

This section specifies extensions to the core protocol that a DNCP profile may want to use.

6.1. Keep-Alives

Trickle-driven status updates (Section 5.1) provide a mechanism for handling of new peer detection (if applicable) on an endpoint, as well as state change notifications. Another mechanism may be needed to get rid of old, no longer valid 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 subsection MUST be ignored.

A DNCP profile MAY specify either per-endpoint or per-peer keep-alive support.

For every endpoint that a keep-alive is specified for in the DNCP profile, the endpoint-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 a non-default keep-alive interval is used on any endpoint, a DNCP node MUST publish appropriate Keep-Alive Interval TLV(s) (Section 7.3.3) within its node data.

6.1.1. 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 7.2.2) was sent for each endpoint, i.e. on an interface or to the point-to-point peer(s).

Each node MUST have for each peer:

- Last contact timestamp: a timestamp which indicates the last time a consistent Network State TLV (Section 7.2.2) was received from the peer via multicast, or anything was received via unicast. When adding a new peer, it should be initialized to the current time.

6.1.2. Per-Endpoint Periodic Keep-Alives

If per-endpoint keep-alives are enabled on an endpoint with a multicast-enabled link, and if no traffic containing a Network State TLV (Section 7.2.2) has been sent to a particular endpoint within the endpoint-specific keep-alive interval, a Network State TLV (Section 7.2.2) MUST be sent on that endpoint, and a new Trickle transmission time 't' in [I/2, I] MUST be randomly chosen. The actual sending time SHOULD be further delayed by a random timespan in [0, Imin/2].

6.1.3. Per-Peer Periodic Keep-Alives

If per-peer keep-alives are enabled on a unicast-only endpoint, and if no traffic containing a Network State TLV (Section 7.2.2) has been sent to a particular peer within the endpoint-specific keep-alive interval, a Network State TLV (Section 7.2.2) MUST be sent to the peer and a new Trickle transmission time 't' in [I/2, I] MUST be randomly chosen.

6.1.4. Received TLV Processing Additions

If a TLV is received via unicast from the peer, the Last contact timestamp for the peer MUST be updated.

On receipt of a Network State TLV (Section 7.2.2) which is consistent with the locally calculated network state hash, the Last contact timestamp for the peer MUST be updated.

6.1.5. Neighbor Removal

For every peer on every endpoint, the endpoint-specific keep-alive interval must be calculated by looking for Keep-Alive Interval TLVs (Section 7.3.3) published by the node, and if none exist, using the default value of DNCP_KEEPALIVE_INTERVAL. If the peer's last contact state timestamp has not been updated for at least DNCP_KEEPALIVE_MULTIPLIER times the peer's endpoint-specific keep-alive interval, the Neighbor TLV for that peer and the local DNCP peer state MUST be removed.

6.2. Support For Dense Broadcast Links

An upper bound for the number of neighbors that are allowed for a (particular type of) link that an endpoint runs on SHOULD be provided by a DNCP profile, user configuration, or some hardcoded default in the implementation. If an implementation does not support this, the rest of this subsection MUST be ignored.

If the specified limit is exceeded, nodes without the highest Node Identifier on the link SHOULD treat the endpoint as a unicast endpoint connected to the node that has the highest Node Identifier detected on the link. The nodes MUST also keep listening to multicast traffic to both detect the presence of that node, and to react to nodes with a higher Node Identifier appearing. If the highest Node Identifier present on the link changes, the remote unicast address of unicast endpoints MUST be changed. If the Node Identifier of the local node is the highest one, the node MUST keep the endpoint in multicast mode, and the node MUST allow others to peer with it over the link via unicast as well.

6.3. Node Data Fragmentation

A DNCP profile may be required to support node data which would not fit the maximum size of a single Node State TLV (Section 7.2.3) (roughly 64KB of payload), or use a datagram-only transport with a limited MTU and no reliable support for fragmentation. To handle such cases, a DNCP profile MAY specify a fixed number of trailing bytes in the Node Identifier to represent a fragment number indicating a part of a node's node data. The profile MAY also specify an upper bound for the size of a single fragment to accommodate limitations of links in the network.

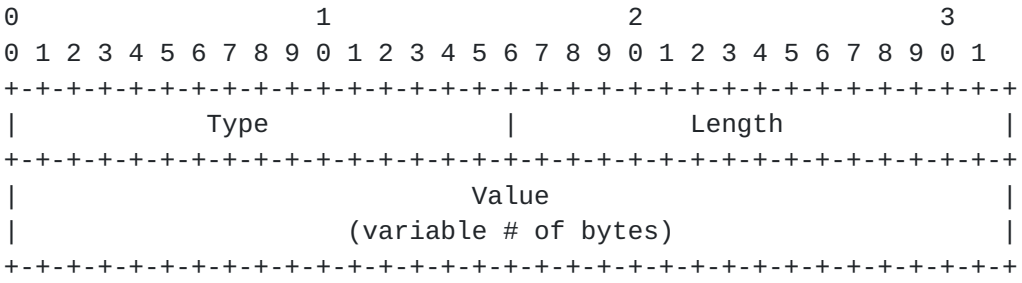
The data within Node State TLVs of fragments with non-zero fragment number must be treated as opaque (as they may not contain even a single full TLV). However, the concatenated node data for a particular node MUST be produced by concatenating all node data for each fragment, in ascending fragment number order. The concatenated node data MUST follow the ordering described in Section 4.

Any Node Identifiers on the wire used to identify the own or any other node MUST have the fragment number 0. For algorithm purposes, the relative time since the most recent fragment change MUST be used, regardless of fragment number. Therefore, even if just part of the node data fragments change, they all are considered refreshed if one of them is.

If using fragmentation, the unreachable node purging defined in Section 5.4 is extended so that if a Fragment Count TLV (Section 7.3.1) is present within the fragment number 0, all fragments up to fragment number specified in the Count field are also considered reachable if the fragment number 0 itself is reachable based on graph traversal.

7. 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. Padding bytes with value zero 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 number stored in the length field.



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

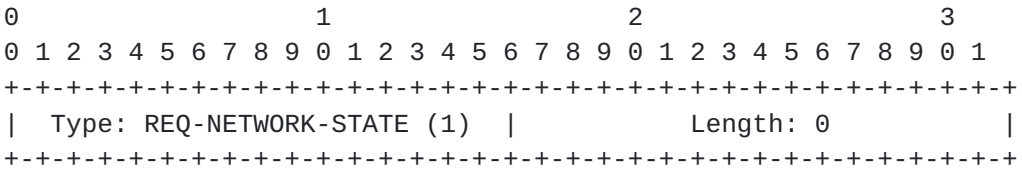
In this section, the following special notation is used:

.. = octet string concatenation operation.

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

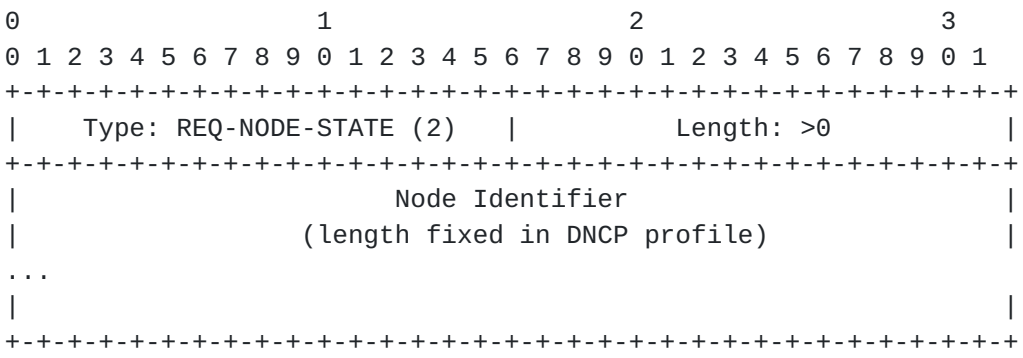
7.1. Request TLVs

7.1.1. Request Network State TLV



This TLV is used to request response with a Network State TLV (Section 7.2.2) and all Node State TLVs (Section 7.2.3).

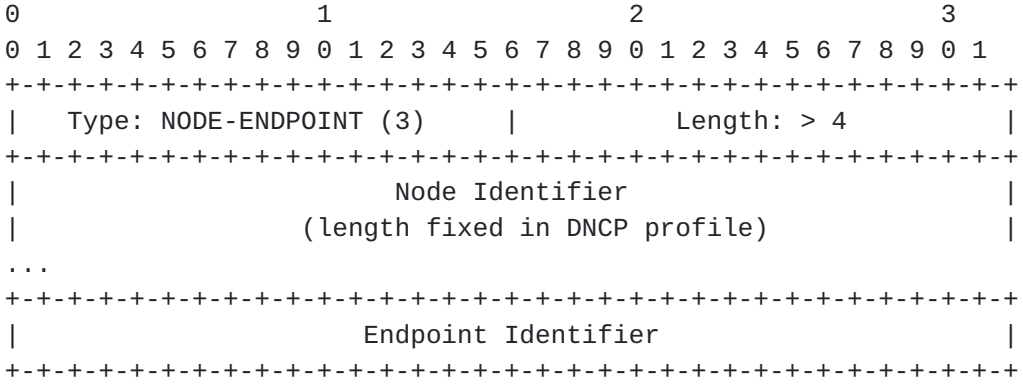
7.1.2. Request Node State TLV



This TLV is used to request a Node State TLV (Section 7.2.3) (including node data) for the node matching the node identifier.

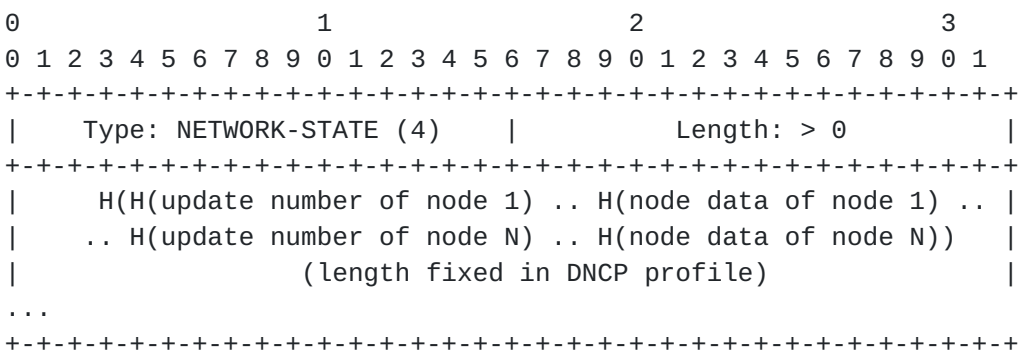
7.2. Data TLVs

7.2.1. Node Endpoint TLV



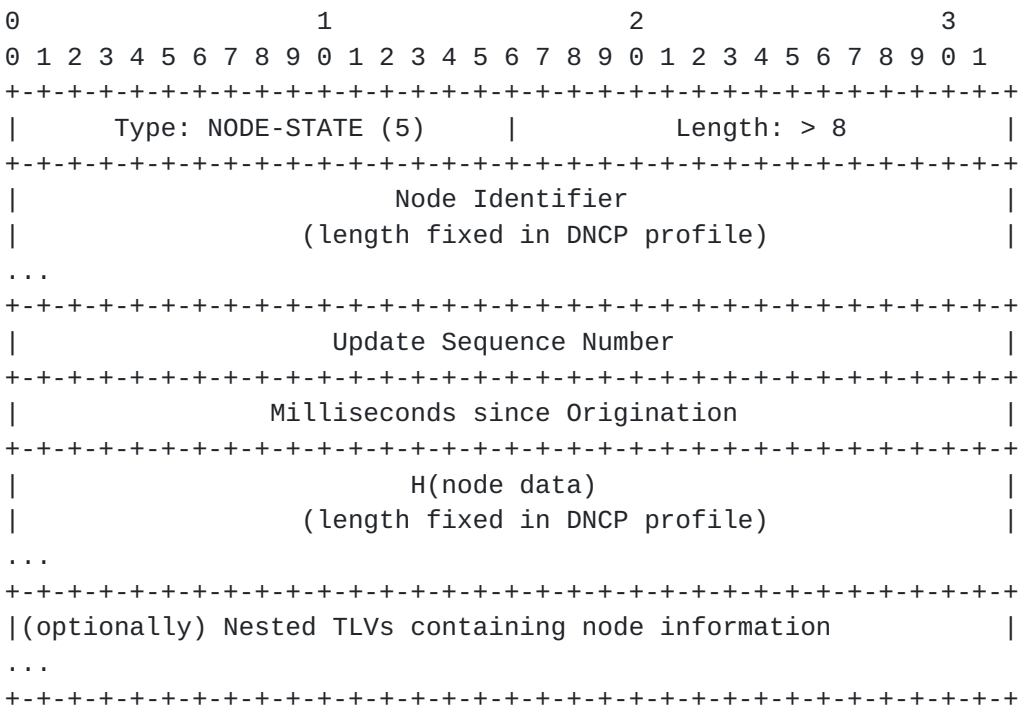
This TLV identifies both the local node's node identifier, as well as the particular endpoint's endpoint identifier. It MUST be sent in every message if bidirectional peer relationship is desired with remote nodes on that endpoint. Bidirectional peer relationship is not necessary for read-only access to the DNCP state, but it is required to be able to publish data.

7.2.2. Network State TLV



This TLV contains the current locally calculated network state hash. It is calculated over each reachable nodes' update number concatenated with the hash value of its node data in ascending order of the respective node identifiers.

7.2.3. Node State TLV

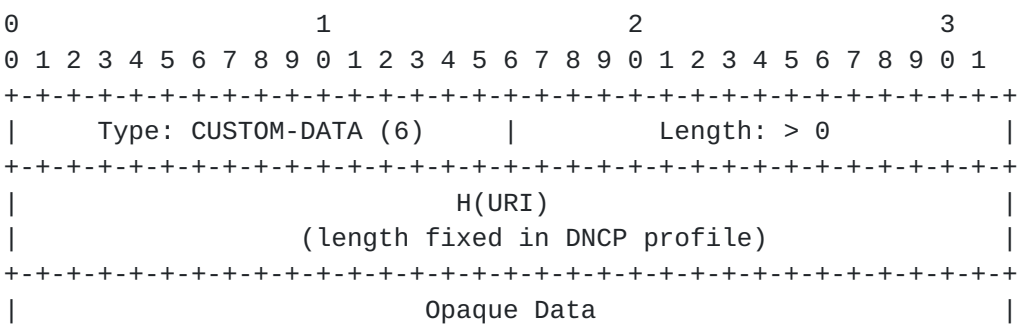


This TLV represents the local node's knowledge about the published state of a node in the DNCP network identified by the node identifier field in the TLV.

The whole network should have roughly the 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 already published node data.

The actual node data of the node may be included within the TLV as well; see Section 5.2 for the cases where it MUST or MUST NOT be included. In a DNCP profile which supports fragmentation, described in Section 6.3, the TLV data may be only partial and not really usable without other fragments.

7.2.4. Custom TLV



This TLV can be used to contain anything; the URI used should be under control of the author of that specification. The TLV may appear within protocol exchanges, or within Node State TLV (Section 7.2.3). For example:

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

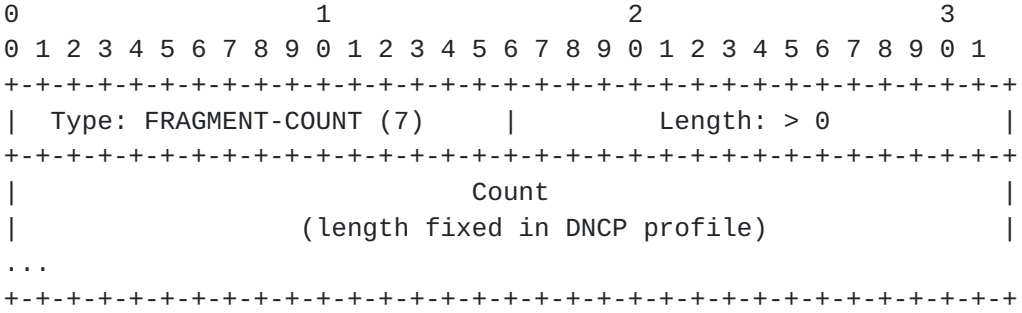
or

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

7.3. Data TLVs within Node State TLV

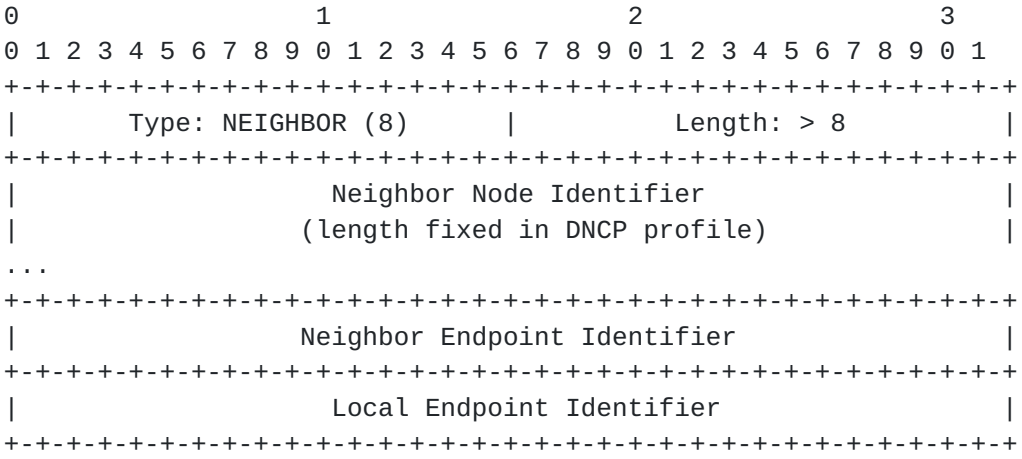
These TLVs are DNCP-specific parts of node-specific node data, and are encoded within the Node State TLVs. If encountered outside Node State TLV, they MUST be silently ignored.

7.3.1. Fragment Count TLV



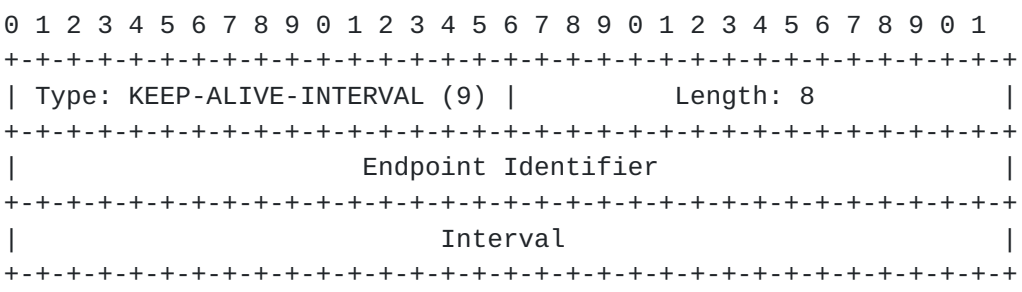
If the DNCP profile supports node data fragmentation as specified in Section 6.3, this TLV indicates that the node data is encoded as a sequence of Node State TLVs. Following Node State TLVs with Node Identifiers up to Count higher than the current one MUST be considered reachable and part of the same logical set of node data that this TLV is within. The fragment portion of the Node Identifier of the Node State TLV this TLV appears in MUST be zero.

7.3.2. Neighbor TLV



This TLV indicates that the node in question vouches that the specified neighbor is reachable by it on the specified local endpoint. 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.

7.3.3. Keep-Alive Interval TLV



This TLV indicates a non-default interval being used to send keep-alives specified in Section 6.1.

Endpoint identifier is used to identify the particular endpoint for which the interval applies. If 0, it applies for ALL endpoints 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. 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.

8.1. 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.

8.2. 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.

8.3. 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 trust 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 trust verdicts announced for said certificate at the time.

8.3.1. Trust Verdicts

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

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

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

2 (Cached Distrust): the last known effective trust 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.

Trust verdicts are differentiated in 3 groups:

- Configured verdicts are used to announce explicit trust 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 trust 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 trust 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 trust verdict of (Cached or Configured) Distrust. In case a node has a configured verdict, which is different from the current effective trust verdict for a certificate, the current effective trust verdict takes precedence in deciding trustworthiness. Despite that, the node still retains and announces its configured verdict.

8.3.2. 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 trust 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.

8.3.3. 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 trust 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:

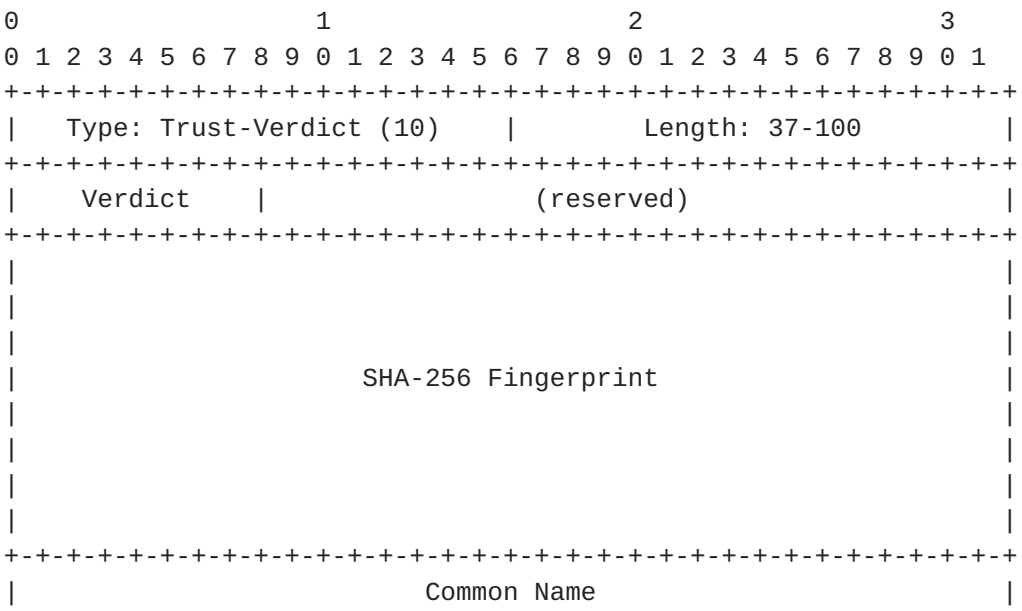
- The stored trust verdict is Cached Trust and the current effective trust verdict for the certificate is Neutral or does not exist.

- The stored trust verdict is Cached Distrust and the current effective trust 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 trust 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 trust 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 trust verdicts SHOULD also be limited in rate and number to prevent denial-of-service attacks.

Trust verdicts are announced using Trust-Verdict TLVs:



Verdict represents the numerical index of the trust 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.

8.3.4. 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.

8.3.4.1. 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.

8.3.4.2. 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).

8.3.4.3. 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.

8.3.4.4. 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.

9. DNCP Profile-Specific Definitions

Each DNCP profile MUST specify the following aspects:

- Unicast and optionally multicast transport protocol(s) to be used.
- How the chosen transport(s) 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. If the 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.
- 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.
- 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 the 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 convergence 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-alives 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.

- Whether to support fragmentation, and if so, the number of bytes reserved for fragment count in the node identifier.

10. 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.

11. 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: Request network state

2: Request node state

3: Node endpoint

4: Network state

5: Node state

6: Custom

7: Fragment count

8: Neighbor

9: Keep-alive interval

10: Trust-Verdict

32-191: Reserved for per-DNCP profile use

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 (11-31, 256-65535), policy of 'standards action' should be used.

12. References

12.1. 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.

12.2. 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 Questions and Answers [RFC Editor: please remove]

Q: 32-bit endpoint 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 have topology information at all?

A: It is an alternative to the more traditional seq#/TTL-based flooding schemes. In steady state, there is no need to e.g. re-publish every now and then.

Appendix B. Changelog [RFC Editor: please remove]

draft-ietf-homenet-dncp-04:

- Added mandatory rate limiting for network state requests, and optional slightly faster convergence mechanism by including current local network state in the remote network state requests.

draft-ietf-homenet-dncp-03:

- Renamed connection -> endpoint.

- !!! Backwards incompatible change: Renumbered TLVs, and got rid of node data TLV; instead, node data TLV's contents are optionally within node state TLV.

draft-ietf-homenet-dncp-02:

- Changed DNCP "messages" into series of TLV streams, allowing optimized round-trip saving synchronization.

- Added fragmentation support for bigger node data and for chunking in absence of reliable L2 and L3 fragmentation.

draft-ietf-homenet-dncp-01:

- Fixed keep-alive semantics to consider unicast requests also updates of most recently consistent, and added proactive unicast request to ensure even inconsistent keep-alive messages eventually triggering consistency timestamp update.

- Facilitated (simple) read-only clients by making Node Connection TLV optional if just using DNCP for read-only purposes.

- Added text describing how to deal with "dense" networks, but left actual numbers and mechanics up to DNCP profiles and (local) configurations.

draft-ietf-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).

- 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.

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

Appendix C. Draft Source [RFC Editor: please remove]

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 D. Acknowledgements

Thanks to Ole Troan, Pierre Pfister, Mark Baugher, Mark Townsley, Juliusz Chroboczek, Jiazi Yi, Mikael Abrahamsson and Brian Carpenter 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