Homenet Working Group
Internet-Draft
Intended status: Standards Track
Expires: January 4, 2016

M. Stenberg

S. Barth

July 3, 2015

Distributed Node Consensus Protocol
draft-ietf-homenet-dncp-07

Abstract

This document describes the Distributed Node Consensus Protocol (DNCP), a generic state synchronization protocol which uses Trickle and Merkle trees. DNCP leaves some details unspecified or provides alternative options. Therefore, only profiles which specify those missing parts 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 January 4, 2016.

Table of Contents

1. Introduction

DNCP is designed to provide a way for each participating node to publish a set of TLV (Type-Length-Value) tuples, and to provide a shared and common view about the data published by every currently or recently bidirectionally reachable DNCP node in a network.

For state synchronization a Merkle tree is used. It is formed by first calculating a hash for the dataset, called node data, published by each node, and then calculating another hash over those node data hashes. The single resulting hash, called network state hash, is transmitted using the Trickle algorithm [RFC6206] to ensure that all nodes share the same view of the current state of the published data within the network. The use of Trickle with only short network state hashes sent infrequently (in steady state) makes DNCP very thrifty when updates happen rarely.

For maintaining liveliness of the topology and the data within it, a combination of Trickled network state, keep-alives, and "other" means of ensuring reachability are used. The core idea is that if every node ensures its neighbors are present, transitively, the whole network state also stays up-to-date.

DNCP is most suitable for data that changes only infrequently to gain the maximum benefit from using Trickle. As the network of nodes, or the rate of data changes grows over a given time interval, Trickle is eventually used less and less and the benefit of using DNCP diminishes. In these cases Trickle just provides extra complexity within the specification and little added value. If constant rapid state changes are needed, the preferable choice is to use an additional point-to-point channel whose address or locator is published using DNCP.

2. Terminology

DNCP profile	a definition of the set of rules and values defining the behavior of a fully specified, implementable protocol which uses DNCP. The DNCP profile specifies transport method to be used, which optional parts of the DNCP specification are required by that particular protocol, and various parameters and optional behaviors. In this document any parameter that a DNCP profile specifies is prefixed with DNCP_. Contents of a DNCP profile are specified in Section 9.
DNCP-based protocol	a protocol which provides a DNCP profile, and potentially much more, e.g., protocol-specific TLVs and guidance on how they should be used.
DNCP node	a single node which runs a DNCP-based protocol.
Link	a link-layer media over which directly connected nodes can communicate.
DNCP network	a set of DNCP nodes running the same DNCP-based protocol. The set consists of nodes that have discovered each other using the transport method defined in the DNCP profile, via multicast on local links, and/or by using unicast communication.
Node identifier	an opaque fixed-length identifier consisting of DNCP_NODE_IDENTIFIER_LENGTH bytes which uniquely identifies a DNCP node within a DNCP network.
Interface	a node's attachment to a particular link.
Address	As DNCP itself is relatively transport agnostic, an address in this specification denotes just something that identifies an endpoint used by the transport protocol employed by a DNCP-based protocol. In case of an IPv6 UDP transport, an address in this specification refers to a tuple (IPv6 address, UDP port).
Endpoint	a locally configured communication endpoint of a DNCP node, such as a network socket. It is either bound to an Interface for multicast and unicast communication, or configured for explicit unicast communication with a predefined set of remote addresses. Endpoints are usually in one of the transport modes specified in Section 4.2.

Endpoint identifier	a 32-bit opaque value, which identifies a particular endpoint of a particular DNCP node. The value 0 is reserved for DNCP and DNCP-based protocol purposes and not 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 using a particular local and remote endpoint pair.
Node data	a set of TLVs published and owned by a node in the DNCP network. Other nodes pass it along as-is, even if they cannot fully interpret it.
Node state	a set of metadata attributes for node data. It includes a sequence number for versioning, a hash value for comparing equality of stored node data, and a timestamp indicating the time passed since its last publication. The hash function and the length of the hash value 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 length of the hash value are defined in the DNCP profile. Whenever a node is added, removed or updates its published node data this hash value changes as well. For calculation, please see Section 4.1.
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.
Topology graph	the undirected graph of DNCP nodes produced by retaining only bidirectional peer relationships between nodes.

2.1. Requirements Language

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

3. Overview

DNCP operates primarily using unicast exchanges between nodes, and may use multicast for Trickle-based shared state dissemination and topology discovery. If used in pure unicast mode with unreliable transport, Trickle is also used between peers.

DNCP discovers the topology of its nodes and maintains the liveliness of published node data by ensuring that the publishing node was - at least recently - bidirectionally reachable. This is determined, e.g., by a recent and consistent multicast or unicast TLV exchange with its peers. New potential peers can be discovered autonomously on multicast-enabled links, their addresses may be manually configured or they may be found by some other means defined in a later specification.

A Merkle tree is maintained by each node to represent the state of all currently reachable nodes and the Trickle algorithm is used to trigger synchronization. The need to check neighboring nodes for state changes is thereby determined by comparing the current root of their respective trees, i.e., their individually calculated network state hashes.

Before joining a DNCP network, a node starts with a Merkle tree (and therefore a calculated network state hash) only consisting of the node itself. It then announces said hash by means of the Trickle algorithm on all its configured endpoints.

When an update is detected by a node (e.g., by receiving a different network state hash from a peer) the originator of the event is requested to provide a list of the state of all nodes, i.e., all the information it uses to calculate its own Merkle tree. The node uses the list to determine whether its own information is outdated and - if necessary - requests the actual node data that has changed.

Whenever a node's local copy of any node data and its Merkle tree are updated (e.g., due to its own or another node's node state changing or due to a peer being added or removed) its Trickle instances are reset which eventually causes any update to be propagated to all of its peers.

4. Operation

4.1. Merkle Tree

Each DNCP node maintains a Merkle tree of height 1 to manage state updates of individual DNCP nodes, the leaves of the tree, and the network as a whole, the root of the tree.

Each leaf represents one recently bidirectionally reachable DNCP node (see Section 4.6), and is represented by a tuple consisting of the node's sequence number in network byte order concatenated with the hash-value of the node's ordered node data published in the Node State TLV (Section 7.2.3). These leaves are ordered in ascending order of the respective node identifiers. The root of the tree - the network state hash - is represented by the hash-value calculated over all such leaf tuples concatenated in order. It is used to determine whether the view of the network of two or more nodes is consistent and shared.

The leaves and the root network state hash are updated on-demand and whenever any locally stored per-node state changes. This includes local unidirectional reachability encoded in the published Neighbor TLVs (Section 7.3.2) and - when combined with remote data - results in awareness of bidirectional reachability changes.

4.2. Data Transport

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. As multicast is used only to identify potential new DNCP nodes and to send status messages which merely notify that a unicast exchange should be triggered, the multicast transport does not have to be secured. If unicast 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. A specific definition of the transport(s) in use and their parameters MUST be provided by the DNCP profile.

TLVs are sent across the transport as is, and they SHOULD be sent together where, e.g., MTU considerations do not recommend sending them in multiple batches. TLVs in general are handled individually and statelessly, with one exception: To form bidirectional peer relationships DNCP requires identification of the endpoints used for communication. As bidirectional peer relationships are required for validating liveliness of published node data as described in Section 4.6, a DNCP node MUST send an Endpoint TLV (Section 7.2.1).

When it is sent varies, depending on the underlying transport, but conceptually it should be available whenever processing a Network State TLV:

- If using a stream transport, the TLV MUST be sent at least once, and it SHOULD be sent only once.

- If using a datagram transport, it MUST be included in every datagram that also contains a Network State TLV (Section 7.2.2) and MUST be located before any such TLV. It SHOULD also be included in any other datagram, to speeds up initial peer detection.

Given the assorted transport options as well as potential endpoint configuration, a DNCP endpoint may be used in various transport modes:

Unicast:

  * If only reliable unicast transport is employed, Trickle is not used at all. Where Trickle reset has been specified, a single Network State TLV (Section 7.2.2) is sent instead to every unicast peer. Additionally, recently changed Node State TLVs (Section 7.2.3) MAY be included.

  * If only unreliable unicast transport is employed, Trickle state is kept per each peer and it is used to send Network State TLVs every now and then, as specified in Section 4.3.

Multicast+Unicast: If multicast datagram transport is available on an endpoint, Trickle state is only maintained for the endpoint as a whole. It is used to send Network State TLVs every now and then, as specified in Section 4.3. Additionally, per-endpoint keep-alives MAY be defined in the DNCP profile, as specified in Section 6.1.2.

MulticastListen+Unicast: Just like Unicast, except multicast transmissions are listened to in order to detect changes of the highest node identifier. This mode is used only if the DNCP profile supports dense broadcast link optimization (Section 6.2).

4.3. Trickle-Driven Status Updates

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.

The Trickle state for all Trickle instances 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. A node MUST NOT reset its Trickle state merely based on receiving a Network State TLV (Section 7.2.2) with a network state hash which is different from its locally calculated one.

Every time a particular Trickle instance indicates that an update should be sent, the node MUST send a Network State TLV (Section 7.2.2) if and only if:

- the endpoint is in Multicast+Unicast transport mode, in which case the TLV MUST be sent over multicast.

- the endpoint is NOT in Multicast+Unicast transport mode, and the unicast transport is unreliable, in which case the TLV MUST be sent over unicast.

A (sub)set of all Node State TLVs (Section 7.2.3) MAY also be included, unless it is defined as undesirable for some reason by the DNCP profile, or to avoid exposure of the node state TLVs by transmitting them within insecure multicast when using also secure unicast.

4.4. Processing of Received TLVs

This section describes how received TLVs are processed. The DNCP profile may specify when to ignore particular TLVs, e.g., to modify security properties - see Section 9 for what may be safely defined to be ignored in a profile. Any 'reply' mentioned in the steps below denotes sending of the specified TLV(s) over 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], to avoid potential simultaneous replies that may cause problems on some links. Sending of replies MAY also be rate-limited or omitted for a short period of time by an implementation. However, an implementation MUST eventually reply to similar repeated requests, as otherwise state synchronization breaks.

A DNCP node MUST process TLVs received from any valid address, as specified by the DNCP profile and the configuration of a particular endpoint, whether this address is known to be the address of a neighbor 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 MUST NOT contain the optional node data part unless explicitly specified in the DNCP profile.

- 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 MAY also be sent.

- Node State TLV (Section 7.2.3):

  * If the node identifier matches the local node identifier and the TLV has a greater sequence number than its current local value, or the same sequence number and a different hash, the node SHOULD re-publish its own node data with an sequence number significantly (e.g., 1000) greater than the received one, to reclaim the node identifier. This may occur normally once due to the local node restarting and not storing the most recently used sequence number. If this occurs more than once or for nodes not re-publishing their own node data, the DNCP profile MUST 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 one or more of the following conditions are true:

+ The local information is outdated for the corresponding node (local sequence number is less than that within the TLV).

+ The local information is potentially incorrect (local sequence number matches but the node data hash differs).

+ There is no data for that node altogether.

Then:

+ If the TLV contains the Node Data field, it SHOULD also be verified by ensuring that the locally calculated H(Node Data) matches the content of the H(Node Data) field within the TLV. If they differ, the TLV SHOULD be ignored and not processed further.

+ If the TLV does not contain the Node Data field, and the H(Node Data) field within the TLV differs from the local node data hash for that node (or there is none), 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 based on Node Data field if present, sequence number and relative time) to match the received TLV.

For comparison purposes of the sequence number, a looping comparison function MUST be used to avoid problems in case of overflow. The comparison function a < b <=> (a - b) % 2^32 & 2^31 != 0 is RECOMMENDED unless the DNCP profile defines another.

o 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 over insecure multicast MUST be silently ignored.

4.5. Adding and Removing Peers

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

o If received over 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 received over multicast, the node MAY 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 existing local transport-specific means (such as Ethernet carrier-detection or TCP keep-alive) 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 local endpoint is in the Multicast-Listen+Unicast transport mode, a Neighbor TLV (Section 7.3.2) MUST NOT be published for the peers not having the highest node identifier.

4.6. Data Liveliness Validation

The topology graph MUST be traversed either immediately or with a small delay shorter than the DNCP profile-defined Trickle Imin, whenever:

- A Neighbor TLV or a whole node is added or removed, or
- the origination time (in milliseconds) of some node's node data is less than current time - 2^32 + 2^15.

The topology graph traversal starts with the local node marked as reachable. Other nodes are then iteratively marked as reachable using the following algorithm: A candidate not-yet-reachable node N with an endpoint NE is marked as reachable if there is a reachable node R with an endpoint RE that meet all of the following criteria:

- The origination time (in milliseconds) of R's node data is greater than current time in - 2^32 + 2^15.
- R publishes a Neighbor TLV with:
  - Neighbor Node Identifier = N's node identifier
  - Neighbor Endpoint Identifier = NE's endpoint identifier
  - Endpoint Identifier = RE's endpoint identifier
- N publishes a Neighbor TLV with:
  - Neighbor Node Identifier = R's node identifier
  - Neighbor Endpoint Identifier = RE's endpoint identifier

* Endpoint Identifier = NE's endpoint identifier

The algorithm terminates, when no more candidate nodes fulfilling these criteria can be found.

DNCP nodes that have not been reachable in the most recent topology 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. They MAY be removed immediately after the topology graph traversal, however it is RECOMMENDED to keep them at least briefly to improve the speed of DNCP network state convergence and to reduce the number of redundant state transmissions between nodes.

5. Data Model

This section describes the local data structures a minimal implementation might use. This section is provided only as a convenience for the implementor. Some of the optional extensions (Section 6) describe additional data requirements, and some optional parts of the core protocol may also require more.

A DNCP node has:

- 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 (required to fulfill reply rate limiting specified in Section 4.4).

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

- Node identifier: the unique identifier of the node. The length, how it is produced, and how collisions are handled, is up to the DNCP profile.

- Node data: the set of TLV tuples published by that particular node. As they are transmitted ordered (see Node State TLV (Section 7.2.3) for details), maintaining the order within the data structure here may be reasonable.

- Latest sequence number: the 32-bit sequence number that is incremented any time the TLV set is published. The comparison function used to compare them is described in Section 4.4.

- Origination time: the (estimated) time when the current TLV set with the current sequence number was published. It is used to populate the Milliseconds Since Origination field in a Node State TLV (Section 7.2.3). Ideally it also has millisecond accuracy.

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 within the local node.
- Trickle instance: the endpoint's Trickle instance with parameters I, T, and c (only on an endpoint in Multicast+Unicast transport mode).

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).
- Trickle instance: the particular peer's Trickle instance with parameters I, T, and c (only on an endpoint in Unicast mode, when using an unreliable unicast transport) .

6. Optional Extensions

This section specifies extensions to the core protocol that a DNCP profile may specify to be used.

6.1. Keep-Alives

Trickle-driven status updates (Section 4.3) provide a mechanism for handling of new peer detection 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 can 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 5) are needed to support keep-alives:

For each configured endpoint that has per-endpoint keep-alives enabled:

- Last sent: If a timestamp which indicates the last time a Network State TLV (Section 7.2.2) was sent over that interface.

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

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

- Last sent: If per-peer keep-alives are enabled, a timestamp which indicates the last time a Network State TLV (Section 7.2.2) was sent to to that point-to-point peer. When adding a new peer, it is initialized to the current time.

6.1.2. Per-Endpoint Periodic Keep-Alives

If per-endpoint keep-alives are enabled on an endpoint in Multicast+Unicast transport mode, 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 over 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 timestamp has not been updated for at least locally chosen potentially endpoint-specific keep-alive multiplier (defaults to 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

This optimization is needed to avoid a state space explosion. Given a large set of DNCP nodes publishing data on an endpoint that actually uses multicast on a link, every node will add a Neighbor TLV (Section 7.3.2) for each peer. While Trickle limits the amount of traffic on the link in stable state to some extent, the total amount of data that is added to and maintained in the DNCP network given N nodes on a multicast-enabled link is O(N^2). Additionally if per-peer keep-alives are employed, there will be O(N^2) keep-alives running on the link if liveliness of peers is not ensured using some other way (e.g., TCP connection lifetime, layer 2 notification, per-endpoint keep-alive).

An upper bound for the number of neighbors that are allowed for a particular type of link that an endpoint in Multicast+Unicast transport mode is used on SHOULD be provided by a DNCP profile, but MAY also be chosen at runtime. Main consideration when selecting a

bound (if any) for a particular type of link should be whether it supports broadcast traffic, and whether a too large number of neighbors case is likely to happen during the use of that DNCP profile on that particular type of link. If neither is likely, there is little point specifying support for this for that particular link type.

If a DNCP profile does not support this extension at all, the rest of this subsection MUST be ignored. This is because when this extension is employed, the state within the DNCP network only contains a subset of the full topology of the network. Therefore every node must be aware of the potential of it being used in a particular DNCP profile.

If the specified upper bound is exceeded for some endpoint in Multicast+Unicast transport mode and if the node does not have the highest node identifier on the link, it SHOULD treat the endpoint as a unicast endpoint connected to the node that has the highest node identifier detected on the link, therefore transitioning to Multicast-listen+Unicast transport mode. The nodes in Multicast-listen+Unicast transport mode MUST keep listening to multicast traffic to both receive messages from the node(s) still in Multicast+Unicast mode, and as well to react to nodes with a greater node identifier appearing. If the highest node identifier present on the link changes, the remote unicast address of the endpoints in Multicast-Listen+Unicast transport mode MUST be changed. If the node identifier of the local node is the highest one, the node MUST switch back to, or stay in Multicast+Unicast mode, and normally form peer relationships with all peers.

6.3. Node Data Fragmentation

A DNCP-based protocol 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. Note that the maximum size of fragment also constrains the maximum size of a single TLV published by a node.

The data within Node State TLVs of all fragments MUST be valid, as specified in Section 7.2.3. The locally used node data for a particular node MUST be produced by concatenating node data in each fragment, in ascending fragment number order. The locally used

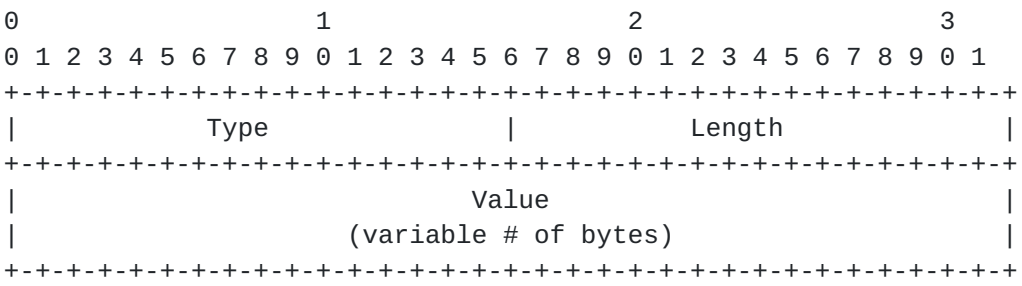
concatenated node data MUST still follow the ordering described in Section 7.2.3.

Any transmitted node identifiers 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 some of the node data fragments change, they all are considered refreshed if one of them is.

If using fragmentation, the data liveliness validation defined in Section 4.6 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, in bytes, 0 meaning 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.

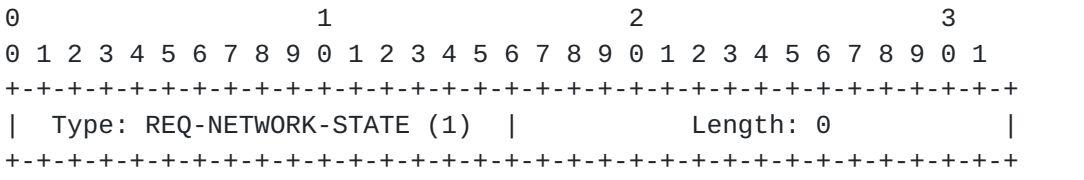
In this section, the following special notation is used:

.. = octet string concatenation operation.

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

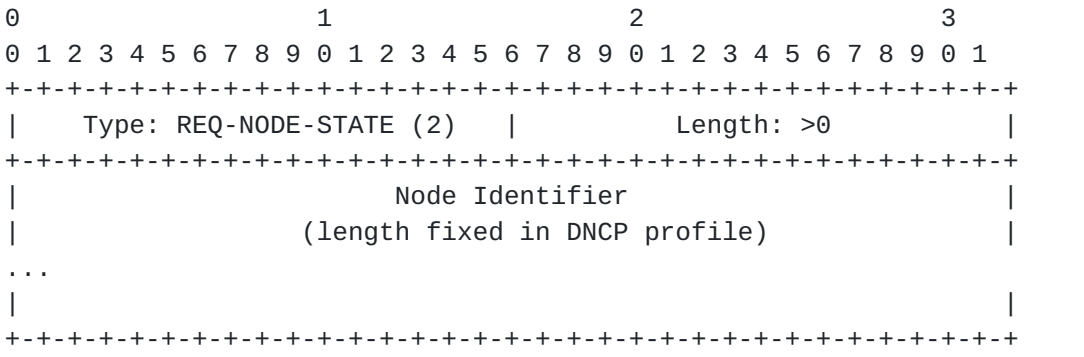
7.1. Request TLVs

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) (without node data).

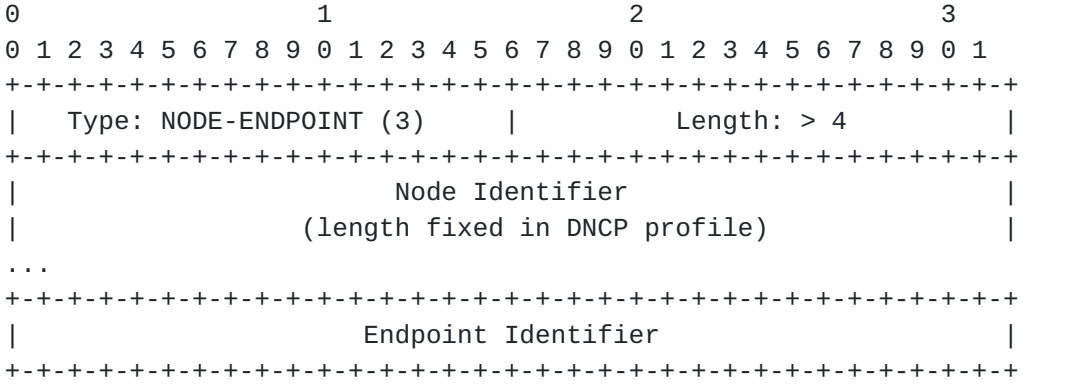
7.1.2. Request Node State TLV



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

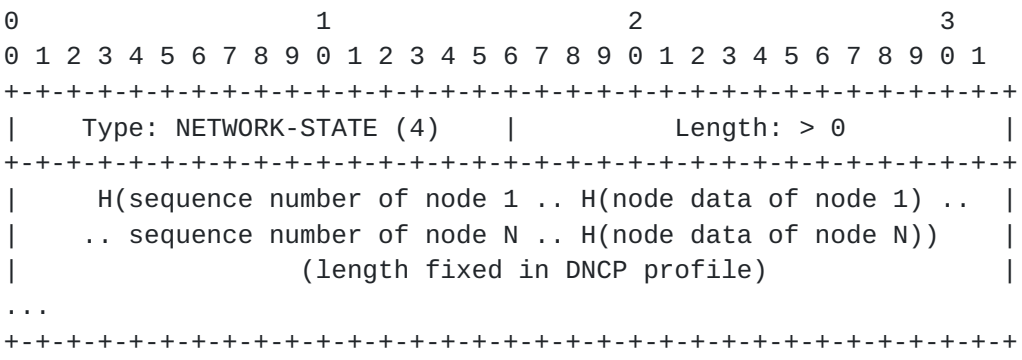
7.2. Data TLVs

7.2.1. Node Endpoint TLV



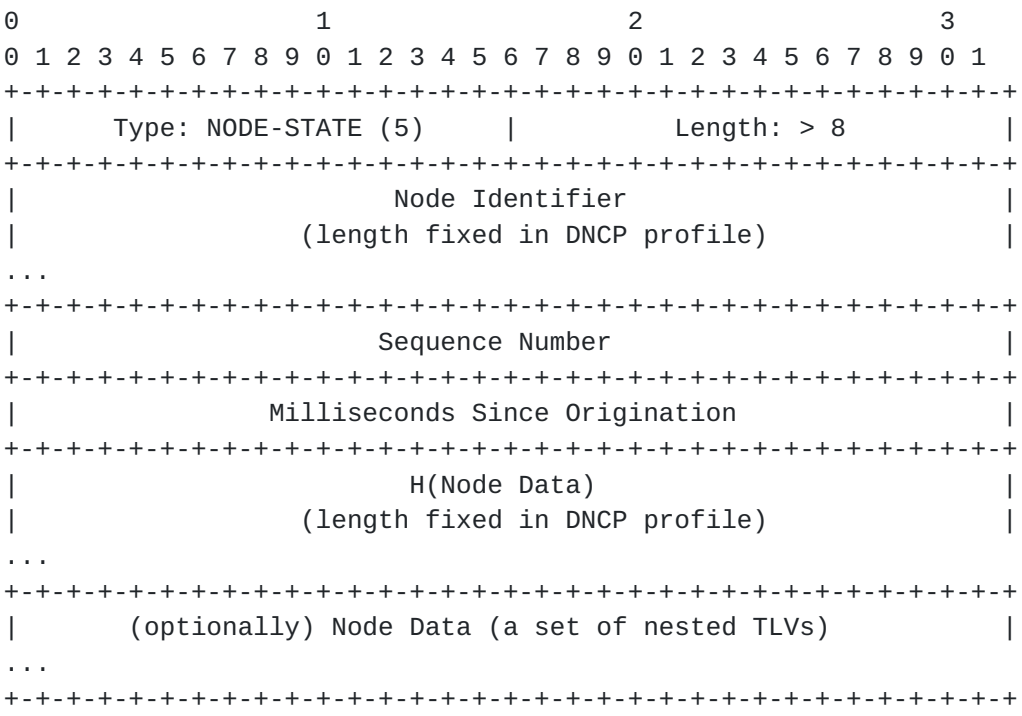
This TLV identifies both the local node's node identifier, as well as the particular endpoint's endpoint identifier.

7.2.2. Network State TLV



This TLV contains the current locally calculated network state hash, see Section 4.1 for how it is calculated.

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.

Every node, including the originating one, MUST update the Milliseconds Since Origination whenever it sends a Node State TLV based on when the node estimates the data was originally published. This is, e.g., to ensure that any relative timestamps contained within the published node data can be correctly offset and

interpreted. Ultimately, what is provided is just an approximation, as transmission delays are not accounted for.

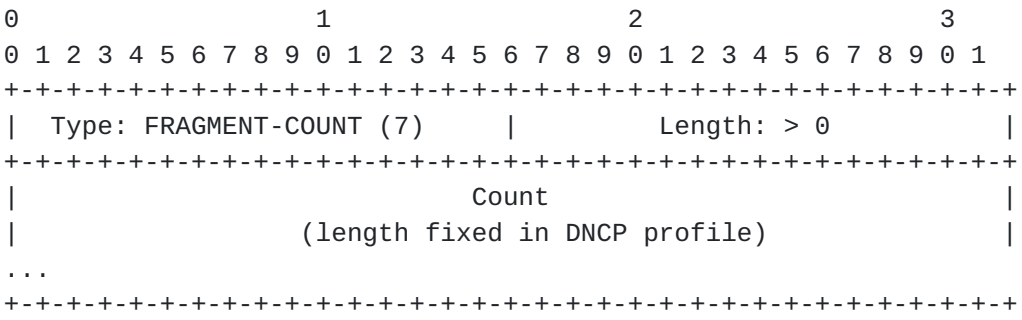
Absent any changes, if the originating node notices that the 32-bit milliseconds since origination value would be close to overflow (greater than 2^32-2^16), the node MUST re-publish its TLVs even if there is no change. In other words, absent any other changes, the TLV set MUST be re-published roughly every 48 days.

The actual node data of the node may be included within the TLV as well in the optional Node Data field. In a DNCP profile which supports fragmentation, described in Section 6.3, the TLV data may be only partial but it MUST contain full individual TLVs. The set of TLVs MUST be strictly ordered based on ascending binary content (including TLV type and length). This enables, e.g., efficient state delta processing and no-copy indexing by TLV type by the recipient. The Node Data content MUST be passed along exactly as it was received. It SHOULD be also verified on receipt that the locally calculated H(Node Data) matches the content of the field within the TLV, and if the hash differs, the TLV SHOULD be ignored.

7.3. Data TLVs within Node State TLV

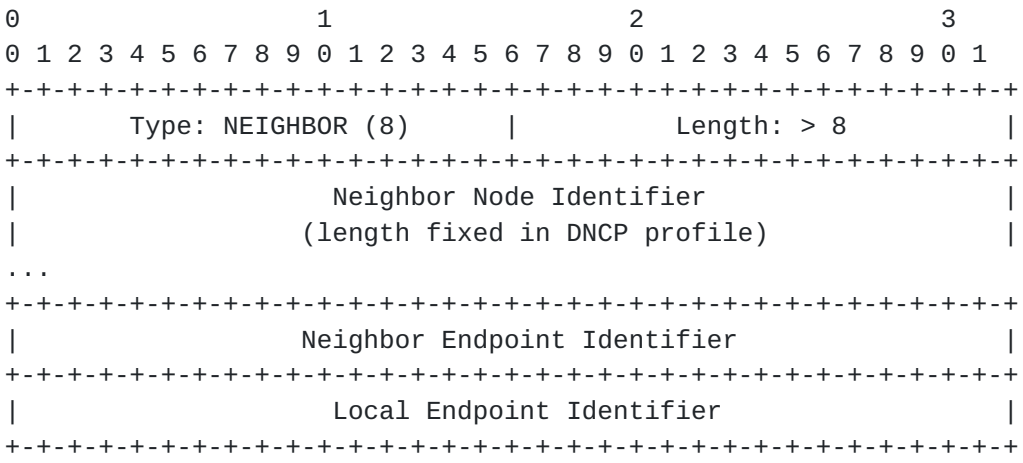
These TLVs are published by the DNCP nodes, and therefore only encoded within the Node State TLVs. If encountered outside Node State TLV, they MUST be silently ignored.

7.3.1. Fragment Count TLV



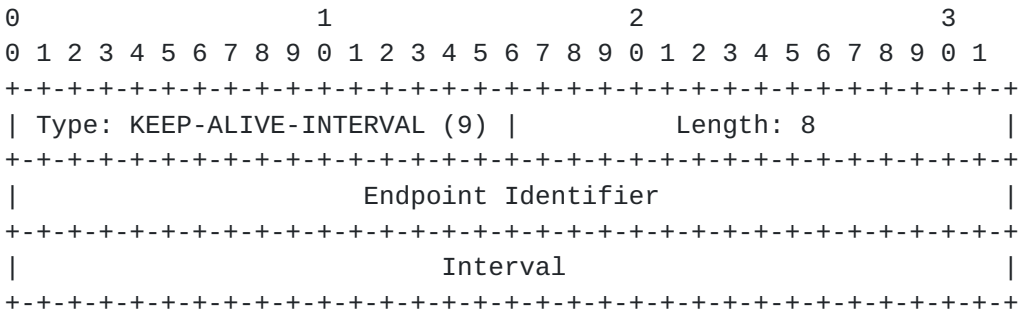
If the DNCP profile supports node data fragmentation as specified in Section 6.3, this TLV indicates that the node data is encoded as a sequence of Node State TLVs. Following Node State TLVs with Node Identifiers up to Count greater than the current one MUST be considered reachable and part of the same 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 needs to 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.

A DNCP node not using this security method MUST ignore all announced trust verdicts and MUST NOT announce any such verdicts by itself, i.e., any other normative language in this subsection does not apply to it.

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:

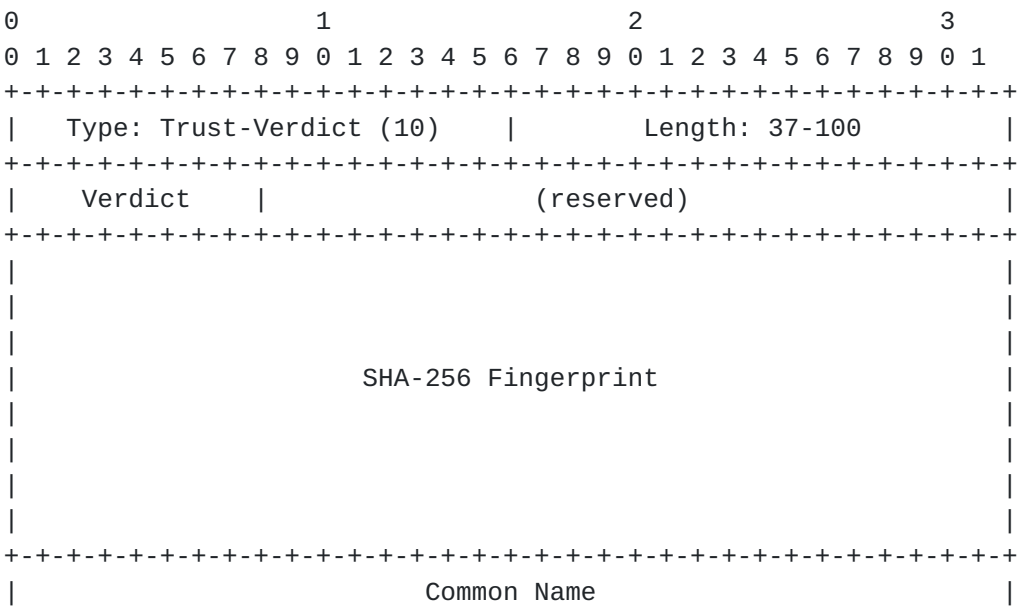
- The stored trust verdict is Cached Trust and the current effective trust verdict for the certificate is Neutral or does not exist.

- 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. If multicast-based node and status discovery is desired, a datagram-based transport supporting multicast has to be available.

- 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 TLVs, what sort of TLVs are ignored in addition - as specified in Section 4.4 - e.g., for security reasons. A DNCP profile may safely define the following DNCP TLVs to be safely ignored:

    * Anything received over multicast, except Node Endpoint TLV (Section 7.2.1) and Network State TLV (Section 7.2.2).

    * Any TLVs received over unreliable unicast or multicast at too high rate; Trickle will ensure eventual convergence given the rate slows down at some point.

- How to deal with node identifier collision as described in Section 4.4. 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 output 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, but a non-cryptographic hash function could be used as well.

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

- Whether to send keep-alives, and if so, whether per-endpoint (requires multicast transport), or per-peer. 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. This is just a default used in absence of any other configuration information, or particular per-endpoint configuration.

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

10. Security Considerations

DNCP-based protocols may use multicast to indicate DNCP state changes and for keep-alive purposes. However, no actual published 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

1: Request network state

2: Request node state

3: Node endpoint

4: Network state

5: Node state

6: Reserved (was: 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. How collision is avoided is out of scope of this document.

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. Alternative Modes of Operation

Beyond what is described in the main text, the protocol allows for other uses. These are provided as examples.

A.1. Read-only Operation

If a node uses just a single endpoint and does not need to publish any TLVs, full DNCP node functionality is not required. Such limited node can acquire and maintain view of the TLV space by implementing the processing logic as specified in Section 4.4. Such node would not need Trickle, peer-maintenance or even keep-alives at all, as the DNCP nodes' use of it would guarantee eventual receipt of network state hashes, and synchronization of node data, even in presence of unreliable transport.

A.2. Forwarding Operation

If a node with a pair of endpoints does not need to publish any TLVs, it can detect (for example) nodes with the highest node identifier on each of the endpoints (if any). Any TLVs received from one of them would be forwarded verbatim as unicast to the other node with highest node identifier.

Any tinkering with the TLVs would remove guarantees of this scheme working; however passive monitoring would obviously be fine. This type of simple forwarding cannot be chained, as it does not send anything proactively.

Appendix B. 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 C. Changelog [RFC Editor: please remove]

draft-ietf-homenet-dncp-06:

- Removed custom TLV.
- Made keep-alive multipliers local implementation choice, profiles just provide guidance on sane default value.
- Removed the DNCP_GRACE_INTERVAL as it is really implementation choice.
- Simplified the suggested structures in data model.
- Reorganized the document and provided an overview section.

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 D. 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 E. Acknowledgements

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