Internet Engineering Task Force (IETF)
Request for Comments: 7787
Category: Standards Track
ISSN: 2070-1721

M. Stenberg
S. Barth
Independent

April 2016

Distributed Node Consensus Protocol

Abstract

This document describes the Distributed Node Consensus Protocol (DNCP), a generic state synchronization protocol that uses the Trickle algorithm and hash trees. DNCP is an abstract protocol and must be combined with a specific profile to make a complete implementable protocol.

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 5741.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at http://www.rfc-editor.org/info/rfc7787.

Copyright Notice

Table of Contents

1. Introduction

DNCP is designed to provide a way for each participating node to publish a small set of TLV (Type-Length-Value) tuples (at most 64 KB) and to provide a shared and common view about the data published by every currently bidirectionally reachable DNCP node in a network.

For state synchronization, a hash tree is used. It is formed by first calculating a hash for the data set published by each node, called node data, 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, once the maximum Trickle interval per link or unicast connection has been reached) 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 peers are present, transitively, the whole network state also stays up to date.

1.1. Applicability

DNCP is useful for cases like autonomous bootstrapping, discovery, and negotiation of embedded network devices like routers. Furthermore, it can be used as a basis to run distributed algorithms like [RFC7596] or use cases as described in Appendix C. DNCP is abstract, which allows it to be tuned to a variety of applications by defining profiles. These profiles include choices of:

- unicast transport: a datagram or stream-oriented protocol (e.g., TCP, UDP, or the Stream Control Transmission Protocol (SCTP)) for generic protocol operation.

- optional transport security: whether and when to use security based on Transport Layer Security (TLS) or Datagram Transport Layer Security (DTLS), if supported over the chosen transport.

- optional multicast transport: a multicast-capable protocol like UDP allowing autonomous peer discovery or more efficient use of multiple access links.

- communication scopes: using either hop by hop only relying on link-local addressing (e.g., for LANs), addresses with broader scopes (e.g., over WANs or the Internet) relying on an existing routing infrastructure, or a combination of both (e.g., to exchange state between multiple LANs over a WAN or the Internet).

- payloads: additional specific payloads (e.g., IANA standardized, enterprise-specific, or private use).

- extensions: possible protocol extensions, either as predefined in this document or specific for a particular use case.

However, there are certain cases where the protocol as defined in this document is a less suitable choice. This list provides an overview while the following paragraphs provide more detailed guidance on the individual matters.

- large amounts of data: nodes are limited to 64 KB of published data.

- very dense unicast-only networks: nodes include information about all immediate neighbors as part of their published data.

- predominantly minimal data changes: node data is always transported as is, leading to a relatively large transmission overhead for changes affecting only a small part of it.

- frequently changing data: DNCP with its use of Trickle is optimized for the steady state and less efficient otherwise.

- large amounts of very constrained nodes: DNCP requires each node to store the entirety of the data published by all nodes.

The topology of the devices is not limited and automatically discovered. When relying on link-local communication exclusively, all links having DNCP nodes need to be at least transitively connected by routers running the protocol on multiple endpoints in order to form a connected network. However, there is no requirement for every device in a physical network to run the protocol. Especially if globally scoped addresses are used, DNCP peers do not need to be on the same or even neighboring physical links. Autonomous discovery features are usually used in local network scenarios; however, with security enabled, DNCP can also be used over unsecured public networks. Network size is restricted merely by the capabilities of the devices, i.e., each DNCP node needs to be able to store the entirety of the data published by all nodes. The data associated with each individual node identifier is limited to about 64 KB in this document; however, protocol extensions could be defined to mitigate this or other protocol limitations if the need arises.

DNCP is most suitable for data that changes only infrequently to gain the maximum benefit from using Trickle. As the network of nodes grows, or the frequency of data changes per node increases, 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.

The suitability of DNCP for a particular application can be roughly evaluated by considering the expected average network-wide state change interval A_NC_I; it is computed by dividing the mean interval at which a node originates a new TLV set by the number of participating nodes. If keep-alives are used, A_NC_I is the minimum of the computed A_NC_I and the keep-alive interval. If A_NC_I is less than the (application-specific) Trickle minimum interval, DNCP is most likely unsuitable for the application as Trickle will not be utilized most of the time.

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. Nevertheless, if doing so does not raise A_NC_I above the (sensibly chosen) Trickle interval parameters for a particular application, using DNCP is probably not suitable for the application.

Another consideration is the size of the published TLV set by a node compared to the size of deltas in the TLV set. If the TLV set published by a node is very large, and has frequent small changes, DNCP as currently specified in this specification may be unsuitable as it lacks a delta synchronization scheme to keep implementation simple.

DNCP can be used in networks where only unicast transport is available. While DNCP uses the least amount of bandwidth when multicast is utilized, even in pure unicast mode, the use of Trickle (ideally with k < 2) results in a protocol with an exponential backoff timer and fewer transmissions than a simpler protocol not using Trickle.

2. Terminology

DNCP profile
the values for the set of parameters given in Section 9. They are prefixed with DNCP_ in this document. The profile also specifies the set of optional DNCP extensions to be used. For a simple example DNCP profile, see Appendix C.

DNCP-based protocol
a protocol that provides a DNCP profile, according to Section 9, and zero or more TLV assignments from the per-DNCP profile TLV registry as well as their processing rules.

DNCP node
a single node that 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 a DNCP-based protocol(s) with a matching DNCP profile(s). 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 that uniquely identifies a DNCP node within a DNCP network.

Interface
a node's attachment to a particular link.

Address
an identifier used as the source or destination of a DNCP message flow, e.g., a tuple (IPv6 address, UDP port) for an IPv6 UDP transport.

Endpoint a locally configured termination point for (potential or established) DNCP message flows. An endpoint is the source and destination for separate unicast message flows to individual nodes and optionally for multicast messages to all thereby reachable nodes (e.g., for node discovery). Endpoints are usually in one of the transport modes specified in Section 4.2.

Endpoint identifier a 32-bit opaque and locally unique 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 at least one 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.

Origination time the (estimated) time when the node data set with the current sequence number was published.

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 (i.e., since the origination time). The hash function and the length of the hash value are defined in the DNCP profile.

Network state hash a hash value that 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.

Bidirectionally reachable a peer is locally unidirectionally reachable if a consistent multicast or any unicast DNCP message has been received by the local node (see Section 4.5). If said peer in return also considers the local node unidirectionally reachable, then bidirectionally reachability is established. As this process is based on publishing peer relationships and evaluating the resulting topology graph as described in Section 4.6, this information is available to the whole DNCP network.

Trickle instance a distinct Trickle [RFC6206] algorithm state kept by a node (Section 5) and related to an endpoint or a particular (peer, endpoint) tuple with Trickle variables I, t, and c. See Section 4.3.

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 it 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 is based on exchanging TLVs (Section 7) and defines a set of mandatory and optional ones for its operation. They are categorized into TLVs for requesting information (Section 7.1), transmitting data (Section 7.2), and being published as data (Section 7.3). DNCP-based protocols usually specify additional ones to extend the capabilities.

DNCP discovers the topology of the nodes in the DNCP network and maintains the liveliness of published node data by ensuring that the publishing node is bidirectionally reachable. 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 the particular DNCP profile. The DNCP profile may specify, for example, a well-known anycast address or provision the remote address to contact via some other protocol such as DHCPv6 [RFC3315].

A hash tree of height 1, rooted in itself, is maintained by each node to represent the state of all currently reachable nodes (see Section 4.1), and the Trickle algorithm is used to trigger synchronization (see Section 4.3). The need to check peer nodes for state changes is thereby determined by comparing the current root of their respective hash trees, i.e., their individually calculated network state hashes.

Before joining a DNCP network, a node starts with a hash tree that has only one leaf if the node publishes some TLVs, and no leaves otherwise. It then announces the network state hash calculated from the hash tree 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 hash 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 hash 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. Hash Tree

Each DNCP node maintains an arbitrary width hash tree of height 1. The root of the tree represents the overall network state hash and is used to determine whether the view of the network of two or more nodes is consistent and shared. Each leaf represents one bidirectionally reachable DNCP node. Every time a node is added or removed from the topology graph (Section 4.6), it is likewise added or removed as a leaf. At any time, the leaves of the tree are ordered in ascending order of the node identifiers of the nodes they represent.

4.1.1. Calculating Network State and Node Data Hashes

The network state hash and the node data hashes are calculated using the hash function defined in the DNCP profile (Section 9) and truncated to the number of bits specified therein.

Individual node data hashes are calculated by applying the function and truncation on the respective node's node data as published in the Node State TLV. Such node data sets are always ordered as defined in Section 7.2.3.

The network state hash is calculated by applying the function and truncation on the concatenated network state. This state is formed by first concatenating each node's sequence number (in network byte order) with its node data hash to form a per-node datum for each node. These per-node data are then concatenated in ascending order of the respective node's node identifier, i.e., in the order that the nodes appear in the hash tree.

4.1.2. Updating Network State and Node Data Hashes

The network state hash and the node data hashes are updated on-demand and whenever any locally stored per-node state changes. This includes local unidirectional reachability encoded in the published Peer TLVs (Section 7.3.1) and -- when combined with remote data -- results in awareness of bidirectional reachability changes.

4.2. Data Transport

DNCP has few requirements for the underlying transport; it requires some way of transmitting either a 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 that 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. They provide for integrity protection and confidentiality of the node data, as well as authentication and authorization using the schemes defined in "Security and Trust Management" (Section 8). A specific definition of the transport(s) in use and its parameters MUST be provided by the DNCP profile.

TLVs (Section 7) 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. DNCP does not fragment or reassemble TLVs; thus, it MUST be ensured that the underlying transport performs these operations should they be necessary. If this document indicates sending one or more TLVs, then the sending node does not need to keep track of the packets sent after handing them over to the respective transport, i.e., reliable DNCP operation is ensured merely by the explicitly defined timers and state machines such as Trickle (Section 4.3). TLVs in general are handled individually and statelessly (and thus do not need to be sent in any particular order) 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 a Node 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 per connection but SHOULD NOT be sent more than 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 speed 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 used, Trickle is not used at all. Whenever the locally calculated network state hash changes, a single Network State TLV (Section 7.2.2) is sent to every unicast peer. Additionally, recently changed Node State TLVs (Section 7.2.3) MAY be included.

   - If only unreliable unicast transport is used, Trickle state is kept per peer, and it is used to send Network State TLVs intermittently, 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 periodically, 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 multicast-enabled link optimization (Section 6.2).

4.3. Trickle-Driven Status Updates

The Trickle algorithm [RFC6206] is used to ensure protocol reliability over unreliable multicast or unicast transports. For reliable unicast transports, its actual algorithm is unnecessary and omitted (Section 4.2). DNCP maintains multiple Trickle states as defined in Section 5. Each such state can be based on different parameters (see below) and is responsible for ensuring that a specific peer or all peers on the respective endpoint are regularly provided with the node's current locally calculated network state hash for state comparison, i.e., to detect potential divergence in the perceived network state.

Trickle defines 3 parameters: Imin, Imax, and k. Imin and Imax represent the minimum value for I and the maximum number of doublings of Imin, where I 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 defined in Section 5 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 the receipt of more recent data from another node as explained in Section 4.1. 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 that 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:

- o the endpoint is in Multicast+Unicast transport mode, in which case the TLV MUST be sent over multicast.
- o 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 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 the sending of the specified TLV(s) to the originator of the TLV being processed. All such replies MUST be sent using unicast. If the TLV being replied to was received via multicast and it was sent to a multiple access link, the reply MUST be delayed by a random time span in [0, Imin/2], to avoid potential simultaneous replies that may cause problems on some links, unless specified differently in the DNCP profile. The sending of replies MAY also be rate limited or omitted for a short period of time by an implementation. However, if the TLV is not forbidden by the DNCP profile, an implementation MUST reply to retransmissions of the TLV with a non-zero probability to avoid starvation, which would break the state synchronization.

A DNCP node MUST process TLVs received from any valid (e.g., correctly scoped) 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 peer 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 to avoid redundant transmission of node data, 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 republish its own node data with a sequence number significantly greater than the received one (e.g., 1000) to reclaim the node identifier. This difference is needed in order to ensure that it is higher than any potentially lingering copies of the node state in the network. 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 republishing 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 (the local sequence number is less than that within the TLV).

        - The local information is potentially incorrect (the 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 hash of the 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 the 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 where (a % b) represents the remainder of a modulo b and (a & b) represents bitwise conjunction of a and b is RECOMMENDED unless the DNCP profile defines another.

- Any other TLV: TLVs not recognized by the receiver MUST be silently ignored unless they are sent within another TLV (for example, TLVs within the Node Data field of a Node State TLV). TLVs within the Node Data field of the Node State TLV not recognized by the receiver MUST be retained for distribution to other nodes and for calculation of the node data hash as described in Section 7.2.3 but are ignored for other purposes.

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

4.5. Discovering, Adding, and Removing Peers

Peer relations are established between neighbors using one or more mutually connected endpoints. Such neighbors exchange information about network state and published data directly, and through transitivity, this information then propagates throughout the network.

New peers are discovered using the regular unicast or multicast transport defined in the DNCP profile (Section 9). This process is not distinguished from peer addition, i.e., an unknown peer is simply discovered by receiving regular DNCP protocol TLVs from it, and dedicated discovery messages or TLVs do not exist. For unicast-only transports, the individual node's transport addresses are preconfigured or obtained using an external service discovery protocol. In the presence of a multicast transport, messages from unknown peers are handled in the same way as multicast messages from peers that are already known; thus, new peers are simply discovered when sending their regular DNCP protocol TLVs using multicast.

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

- If received over unicast, the remote node MUST be added as a peer on the endpoint, and a Peer TLV (Section 7.3.1) 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 used to ensure its presence. If the peer does not send keep-alives, and no means to verify presence of the peer are available, the peer MUST be considered no longer present, and it SHOULD NOT be added back as a peer until it starts sending keep-alives again. When the peer is no longer present, the Peer TLV and the local DNCP peer state MUST be removed. DNCP does not define an explicit message or TLV for indicating the termination of DNCP operation by the terminating node; however, a derived protocol could specify an extension, if the need arises.

If the local endpoint is in the Multicast-Listen+Unicast transport mode, a Peer TLV (Section 7.3.1) MUST NOT be published for the peers not having the highest node identifier.

4.6. Data Liveliness Validation

Maintenance of the hash tree (Section 4.1) and thereby network state hash updates depend on up-to-date information on bidirectional node reachability derived from the contents of a topology graph. This graph changes whenever nodes are added to or removed from the network or when bidirectional connectivity between existing nodes is established or lost. Therefore, the graph MUST be updated either immediately or with a small delay shorter than the DNCP profile-defined Trickle Imin whenever:

- A Peer 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 artificial upper limit for the origination time is used to gracefully avoid overflows of the origination time and allow for the node to republish its data as noted in Section 7.2.3.

The topology graph update starts with the local node marked as reachable and all other nodes marked as unreachable. 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 meets all of the following criteria:

- The origination time (in milliseconds) of R's node data is greater than current time - 2^32 + 2^15.

- R publishes a Peer TLV with:

  - Peer Node Identifier = N's node identifier

  - Peer Endpoint Identifier = NE's endpoint identifier

  - Endpoint Identifier = RE's endpoint identifier

- N publishes a Peer TLV with:

  - Peer Node Identifier = R's node identifier

  - Peer 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 forgotten 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. This reduces the number of queries needed to reconverge during both initial network convergence and when a part of the network loses and regains bidirectional connectivity within that time period.

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 the following 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 in a particular order (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 locally unique value identifying the endpoint within a node. It SHOULD NOT be reused immediately after an endpoint is disabled.

- 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.

- Set of addresses: the DNCP nodes from which connections are accepted.

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

While DNCP provides mechanisms for discovery and adding new peers on an endpoint (Section 4.5), 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 as noted in Section 4.6.

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 (sent using multicast to all DNCP nodes connected to a multicast-enabled link) or per-peer (sent using unicast to each peer individually) 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 an appropriate Keep-Alive Interval TLV(s) (Section 7.3.2) 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 that 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 that indicates the last time a consistent Network State TLV (Section 7.2.2) was received from the peer over multicast or when anything was received over unicast. Failing to update it for a certain amount of time as specified in Section 6.1.5 results in the removal of the peer. When adding a new peer, it is initialized to the current time.

- Last sent: If per-peer keep-alives are enabled, a timestamp that indicates the last time a Network State TLV (Section 7.2.2) was sent 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 interval started, as specified in step 2 of Section 4.2 of [RFC6206]. The actual sending time SHOULD be further delayed by a random time span 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 interval started, as specified in step 2 of Section 4.2 of [RFC6206].

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) that is consistent with the locally calculated network state hash, the Last contact timestamp for the peer MUST be updated in order to maintain it as a peer.

6.1.5. Peer Removal

For every peer on every endpoint, the endpoint-specific keep-alive interval must be calculated by looking for Keep-Alive Interval TLVs (Section 7.3.2) published by the node, and if none exist, use the default value of DNCP_KEEPALIVE_INTERVAL. If the peer's Last contact timestamp has not been updated for at least a locally chosen potentially endpoint-specific keep-alive multiplier (defaults to DNCP_KEEPALIVE_MULTIPLIER) times the peer's endpoint-specific keep-alive interval, the Peer TLV for that peer and the local DNCP peer state MUST be removed.

6.2. Support for Dense Multicast-Enabled Links

This optimization is needed to avoid a state space explosion. Given a large set of DNCP nodes publishing data on an endpoint that uses multicast on a link, every node will add a Peer TLV (Section 7.3.1) 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 used, there will be O(N^2) keep-alives running on the link if the liveliness of peers is not ensured using some other way (e.g., TCP connection lifetime, Layer 2 notification, or per-endpoint keep-alive).

An upper bound for the number of peers 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 it MAY also be chosen at runtime. The main consideration when selecting a bound (if any) for a particular type of link should be whether it supports multicast traffic and whether a too large number of peers 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 used, 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. See Section 4.2 for implications on the specific endpoint behavior. 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 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 form peer relationships with all peers as specified in Section 4.5.

7. Type-Length-Value Objects

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|            Type               |           Length              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|              Value (if any) (+padding (if any))               |
..
|                    (variable # of bytes)                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    (optional nested TLVs)                     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Each TLV is encoded as:

- a 2-byte Type field

- a 2-byte Length field, which contains the length of the Value field in bytes; 0 means no value

- the value itself (if any)

- padding bytes with a value of zero up to the next 4-byte boundary if the Length is not divisible by 4

While padding bytes MUST NOT be included in the number stored in the Length field of the TLV, if the TLV is enclosed within another TLV, then the padding is included in the enclosing TLV's Length value.

Each TLV that does not define optional fields or variable-length content MAY be sent with additional sub-TLVs appended after the TLV to allow for extensibility. When handling such TLV types, each node MUST accept received TLVs that are longer than the fixed fields specified for the particular type and ignore the sub-TLVs with either unknown types or types not supported within that particular TLV. If any sub-TLVs are present, the Length field of the TLV describes the number of bytes from the first byte of the TLV's own Value (if any) to the last (padding) byte of the last sub-TLV.

For example, type=123 (0x7b) TLV with value 'x' (120 = 0x78) is encoded as: 007B 0001 7800 0000. If it were to have a sub-TLV of type=124 (0x7c) with value 'y', it would be encoded as 007B 000C 7800 0000 007C 0001 7900 0000.

In this section, the following special notation is used:

.. = octet string concatenation operation.

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

In addition to the TLV types defined in this document, TLV Types 11-31 and 512-767 are unassigned and may be sequentially registered, starting at 11, by Standards Action [RFC5226] by extensions to DNCP that may be applicable in multiple DNCP profiles.

7.1. Request TLVs

7.1.1. Request Network State TLV

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|Type: Request network state (1)|          Length: >= 0         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

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

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

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

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

This TLV identifies both the local node's node identifier, as well as the particular endpoint's endpoint identifier. Section 4.2 specifies when it is sent.

7.2.2. Network State TLV

```
      0                   1                   2                   3
      0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     |    Type: Network state (4)    |          Length: > 0          |
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     |     H(sequence number of node 1 .. H(node data of node 1) ..  |
     |    .. sequence number of node N .. H(node data of node N))    |
     |                  (length fixed in DNCP profile)               |
     ...
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

This TLV contains the current network state hash calculated by its sender (Section 4.1 describes the algorithm).

7.2.3. Node State TLV

```
      0                   1                   2                   3
      0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     |      Type: Node state (5)     |          Length: > 8          |
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     |                       Node Identifier                         |
     |                  (length fixed in DNCP profile)               |
     ...
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     |                        Sequence Number                        |
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     |                 Milliseconds Since Origination                |
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     |                         H(Node Data)                          |
     |                  (length fixed in DNCP profile)               |
     ...
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     |      (optionally) Node Data (a set of nested TLVs)            |
     ...
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

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 node publishing the node data, 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 republish its TLVs even if there is no change. In other words, absent any other changes, the TLV set MUST be republished roughly every 48 days.

The actual node data of the node may be included within the TLV as well as in the optional Node Data field. 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 are therefore only encoded in the Node Data field of Node State TLVs. If encountered outside Node State TLV, they MUST be silently ignored.

7.3.1. Peer TLV

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

This TLV indicates that the node in question vouches that the specified peer 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 peer recently. For guaranteed up-to-date bidirectional reachability, the existence of both nodes' matching Peer TLVs needs to be checked.

7.3.2. Keep-Alive Interval TLV

```
     0                   1                   2                   3
     0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    | Type: Keep-alive interval (9) |         Length: >= 8          |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |                      Endpoint Identifier                      |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |                           Interval                            |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

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

Endpoint identifier is used to identify the particular (local) endpoint for which the interval applies on the sending node. 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 the 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. Trust Method Based on Pre-Shared Key

A trust model based on Pre-Shared Key (PSK) is a simple security management mechanism that allows an administrator to deploy devices to an existing network by configuring them with a predefined key, similar to the configuration of an administrator password or Wi-Fi Protected Access (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. However, it 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

For some scenarios -- such as bootstrapping a mostly unmanaged network -- the methods described above may not provide a desirable trade-off between security and user experience. This section includes guidance for implementing an opportunistic security [RFC7435] method that DNCP profiles can build upon and adapt for their specific requirements.

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 relations 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 verdicts it has established by itself, and it MUST do so if announcing the configured 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:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Type: Trust-Verdict (10)    |        Length: > 36           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    Verdict    |                  (reserved)                   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
|                                                               |
|                                                               |
|                     SHA-256 Fingerprint                       |
|                                                               |
|                                                               |
|                                                               |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          Common Name                          |
```

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.

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 peer 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 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 that 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 a multicast transport protocol(s) to be used. If a multicast-based node and status discovery is desired, a datagram-based transport supporting multicast has to be available.

- How the chosen transport(s) is 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 addresses 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. While the security of the node data published within the Node State TLVs is already ensured by the base specification (if secure unicast transport is used, Node State TLVs are sent only via unicast as multicast ones are ignored on receipt), if a profile adds TLVs that are sent outside the node data, a profile should indicate whether or not those TLVs should be ignored if they are received via multicast or non-secured unicast. A DNCP profile may 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 a rate that is that is too high; 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 help 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 data and producing network state hash, which is a hash of node data hashes. SHA-256 defined in [RFC6234] is the recommended default choice, but a non-cryptographic hash function could be used as well. If there is a hash collision in the network state hash, the network will effectively be partitioned to partitions that believe they are up to date but are actually no longer converged. The network will converge either when some node data anywhere in the network changes or when conflicting Node State TLVs get transmitted across the partition (either caused by "Trickle-Driven Status Updates" (Section 4.3) or as part of the "Processing of Received TLVs" (Section 4.4)). If a node publishes node data with a hash that collides with any previously published node data, the update may not be (fully) propagated, and the old version of node data may be used instead.

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

- Whether to send keep-alives, and if so, whether it is per-endpoint (requires multicast transport) or per-peer. Keep-alive also has 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) 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 dense multicast-enabled link optimization (Section 6.2) or not.

For some guidance on choosing transport and security options, please see Appendix B.

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 MUST 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 the inability to use the network to either check Certificate Revocation Lists or perform online 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 has set up a registry for the (decimal 16-bit) "DNCP TLV Types" under "Distributed Node Consensus Protocol (DNCP)". The registration procedure is Standards Action [RFC5226]. The initial contents are:

0: Reserved

1: Request network state

2: Request node state

3: Node endpoint

4: Network state

5: Node state

6: Reserved for future use (was: Custom)

7: Reserved for future use (was: Fragment count)

8: Peer

9: Keep-alive interval

10: Trust-Verdict

11-31: Unassigned

32-511: Reserved for per-DNCP profile use

512-767: Unassigned

768-1023: Reserved for Private Use [RFC5226]

1024-65535: Reserved for future use

12. References

12.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, DOI 10.17487/RFC5226, May 2008, <http://www.rfc-editor.org/info/rfc5226>.

[RFC6206] Levis, P., Clausen, T., Hui, J., Gnawali, O., and J. Ko, "The Trickle Algorithm", RFC 6206, DOI 10.17487/RFC6206, March 2011, <http://www.rfc-editor.org/info/rfc6206>.

[RFC6234] Eastlake 3rd, D. and T. Hansen, "US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)", RFC 6234, DOI 10.17487/RFC6234, May 2011, <http://www.rfc-editor.org/info/rfc6234>.

12.2. Informative References

[RFC3315] Droms, R., Ed., Bound, J., Volz, B., Lemon, T., Perkins, C., and M. Carney, "Dynamic Host Configuration Protocol for IPv6 (DHCPv6)", RFC 3315, DOI 10.17487/RFC3315, July 2003, <http://www.rfc-editor.org/info/rfc3315>.

[RFC3493] Gilligan, R., Thomson, S., Bound, J., McCann, J., and W. Stevens, "Basic Socket Interface Extensions for IPv6", RFC 3493, DOI 10.17487/RFC3493, February 2003, <http://www.rfc-editor.org/info/rfc3493>.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <http://www.rfc-editor.org/info/rfc5246>.

[RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <http://www.rfc-editor.org/info/rfc6347>.

[RFC7435] Dukhovni, V., "Opportunistic Security: Some Protection Most of the Time", RFC 7435, DOI 10.17487/RFC7435, December 2014, <http://www.rfc-editor.org/info/rfc7435>.

[RFC7596] Cui, Y., Sun, Q., Boucadair, M., Tsou, T., Lee, Y., and I. Farrer, "Lightweight 4over6: An Extension to the Dual-Stack Lite Architecture", RFC 7596, DOI 10.17487/RFC7596, July 2015, <http://www.rfc-editor.org/info/rfc7596>.

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 a limited node can acquire and maintain a 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 the 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 the 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. DNCP Profile Additional Guidance

This appendix explains implications of design choices made when specifying the DNCP profile to use particular transport or security options.

B.1. Unicast Transport -- UDP or TCP?

The node data published by a DNCP node is limited to 64 KB due to the 16-bit size of the length field of the TLV it is published within. Some transport choices may decrease this limit; if using, e.g., UDP datagrams for unicast transport, the upper bound of the node data size is whatever the nodes and the underlying network can pass to each other as DNCP does not define its own fragmentation scheme. A profile that chooses UDP has to be limited to small node data (e.g., somewhat smaller than IPv6 default MTU if using IPv6) or specify a minimum that all nodes have to support. Even then, if using non-link-local communications, there is some concern about what middleboxes do to fragmented packets. Therefore, the use of stream transport such as TCP is probably a good idea if either non-link-local communication is desired or fragmentation is expected to cause problems.

TCP also provides some other facilities, such as a relatively long built-in keep-alive, which in conjunction with connection closes occurring from eventual failed retransmissions may be sufficient to avoid the use of in-protocol keep-alive defined in Section 6.1. Additionally, it is reliable, so there is no need for Trickle on such unicast connections.

The major downside of using TCP instead of UDP with DNCP-based profiles lies in the loss of control over the time at which TLVs are received; while unreliable UDP datagrams also have some delay, TLVs within reliable stream transport may be delayed significantly due to retransmissions. This is not a problem if no relative time-dependent information is stored within the TLVs in the DNCP-based protocol; for such a protocol, TCP is a reasonable choice for unicast transport if it is available.

B.2. (Optional) Multicast Transport

Multicast is needed for dynamic peer discovery and to trigger unicast exchanges; for that, unreliable datagram transport (=typically UDP) is the only transport option defined within this specification, although DNCP-based protocols may themselves define some other transport or peer discovery mechanism (e.g., based on Multicast DNS (mDNS) or DNS).

If multicast is used, a well-known address should be specified and for, e.g., IPv6, respectively, the desired address scopes. In most cases, link-local and possibly site-local are useful scopes.

B.3. (Optional) Transport Security

In terms of provided security, DTLS and TLS are equivalent; they also consume a similar amount of state on the devices. While TLS is on top of a stream protocol, using DTLS also requires relatively long session caching within the DTLS layer to avoid expensive reauthentication/authorization steps if and when any state within the DNCP network changes or per-peer keep-alive (if enabled) is sent.

TLS implementations (at the time of writing the specification) seem more mature and available (as open source) than DTLS ones. This may be due to a long history of use with HTTPS.

Some libraries seem not to support multiplexing between insecure and secure communication on the same port, so specifying distinct ports for secured and unsecured communication may be beneficial.

Appendix C. Example Profile

This is the DNCP profile of SHSP, an experimental (and for the purposes of this document fictional) home automation protocol. The protocol itself is used to make a key-value store published by each of the nodes available to all other nodes for distributed monitoring and control of a home infrastructure. It defines only one additional TLV type: a key=value TLV that contains a single key=value assignment for publication.

- Unicast transport: IPv6 TCP on port EXAMPLE-P1 since only absolute timestamps are used within the key=value data and since it focuses primarily on Linux-based nodes that support both protocols as well. Connections from and to non-link-local addresses are ignored to avoid exposing this protocol outside the secure links.

- Multicast transport: IPv6 UDP on port EXAMPLE-P2 to link-local scoped multicast address ff02:EXAMPLE. At least one node per link in the home is assumed to facilitate node discovery without depending on any other infrastructure.

- Security: None. It is to be used only on trusted links (WPA2-x wireless, physically secure wired links).

- Additional TLVs to be ignored: None. No DNCP security is specified, and no new TLVs are defined outside of node data.

- Node identifier length (DNCP_NODE_IDENTIFIER_LENGTH): 32 bits that are randomly generated.

- Node identifier collision handling: Pick new random node identifier.

- Trickle parameters: Imin = 200 ms, Imax = 7, k = 1. It means at least one multicast per link in 25 seconds in stable state (0.2 * 2^7).

- Hash function H(x) + length: SHA-256, only 128 bits used. It's relatively fast, and 128 bits should be plenty to prevent random conflicts (64 bits would most likely be sufficient, too).

- No in-protocol keep-alives (Section 6.1); TCP keep-alive is to be used. In practice, TCP keep-alive is seldom encountered anyway, as changes in network state cause packets to be sent on the unicast connections, and those that fail sufficiently many retransmissions are dropped much before the keep-alive actually would fire.

- No support for dense multicast-enabled link optimization (Section 6.2); SHSP is a simple protocol for a few nodes (network wide, not even to mention on a single link) and therefore would not provide any benefit.

Acknowledgements

Thanks to Ole Troan, Pierre Pfister, Mark Baugher, Mark Townsley, Juliusz Chroboczek, Jiazi Yi, Mikael Abrahamsson, Brian Carpenter, Thomas Clausen, DENG Hui, and Margaret Cullen for their contributions to the document.

Thanks to Kaiwen Jin and Xavier Bonnetain for their related research work.

Authors' Addresses

Markus Stenberg
Independent
Helsinki 00930
Finland

Email: markus.stenberg@iki.fi

Steven Barth
Independent
Halle 06114
Germany

Email: cyrus@openwrt.org