Distributed Node Consensus Protocol: Analysis, Evaluation and Performance

Kaiwen Jin*†, Pierre Pfister†, Jiazi Yi*
*Laboratoire d'Informatique (LIX) – Ecole Polytechnique
Palaiseau, 91128, France
{kaiwen.jin jiazi.yi}@polytechnique.edu
†Cisco Systems, Paris, France
ppfister@cisco.com

***Abstract*—This paper analyzes and evaluates the Distributed Node Consensus Protocol (DNCP), a state synchronization mechanism developed by the IETF Homenet working group. DNCP enables network function automation for home networks, which are growing in size and complexity. The basic mechanisms of DNCP are studied in this paper, including the state abstraction, synchronization process and keep-alive mechanism. The overhead is analyzed in single-link topology type. To evaluate the performance of DNCP in more complex scenarios, a reference implementation of DNCP is integrated into *ns3* simulator. The convergence time and transmission overhead in various topology types are measured. Based on the obtained results, the correctness of DNCP is verified, and the behavior of DNCP can be concluded.**

I. Introduction

With the advances in micro-controller and wireless technology, the concept of "being online" is no longer exclusively reserved for computers, but expected also for phones, vehicles, televisions, refrigerators, utility meters, etc. "The Internet of Things" (IoT) assumes objects in our environment to be part of the Internet, communicating with users and with each other. Even in a home network, there can be body sensors, electric meters, water meters, temperature sensors, fire alarms, computers, etc., which are all connected to the Internet. As a consequence, how to manage more complex home networks is becoming an important issue.

A. Background

The Home Networking Working Group (Homenet) was chartered in 2011 by the IETF[1] focusing on residential networks' next generation protocols. Unlike current home networks, which commonly comprise a single residential gateway performing Network Address Translator with a single IPv4 address, future home networks will:

- provide end-to-end connectivity based on IPv6;
- comprise multiple routers (e.g., WiFi extension, devices hosting virtual machines, IoT gateway) and links (e.g., wired, wireless, low power);
- be connected to the internet through multiple uplinks (e.g., cable, mobile or VPN).

One of the most important challenge is to provide basic network management (prefix and address assignment, routing, name resolution, service discovery, etc.) in a way which does not imply any configuration input from the user.

To resolve the issue, the Distributed Node Consensus Protocol (DNCP) [1] is currently being developed by the IETF Homenet working group as a generic state synchronization protocol. It makes use of hash trees and the Trickle algorithm [2] and provides neighbor discovery as well as dead neighbor detection, topology discovery, and allows every participating node to publish information that is propagated to all other nodes. Originally designed for home networks, DNCP can also be used for other networks that require synchronization between nodes. Some DNCP parameters are therefore specified by specific profiles. For example, the DNCP profile to be used in home networks is specified as part of the Home Networking Control Protocol (HNCP) [3].

Features provided by DNCP are very close to what any link state routing protocol such as OSPF (Open Shortest Path Forwarding) [4] would provide. But the Homenet working group decided to develop a distinct protocol in order to not interfere with the routing protocol operations and specifications. In addition, as it uses Trickle, DNCP is more suitable to networks in which state changes infrequently.

B. Statement of Purpose

HNCP, based on DNCP, is a fundamental element of future home networks protocol stack. The protocol was specified in parallel to a reference implementation (The hnet project: http://www.homewrt.org/) and was therefore tested on small scale testbeds, but there has not been any analysis of the protocol behavior and performances at different scales and on different topologies yet.

In the literature, there has been some work analyzing the performances of the Trickle Algorithm. [5] models the broadcasting process of a network using Trickle with a Markov chain and measures the Trickle's message count. In [6], qualitative results are provided to study the algorithm in wireless sensor environments. [7] presents an analytic model of a static Trickle-based network under steady state conditions, as a function of the redundancy constant and the average node degree. From the aspect of convergence, [8] analyzes end-to-end delay distributions in terms of the Trickle parameters and network density.

[1]Internet Engineering Task Force, http://www.ietf.org

Nevertheless, there are some important differences between the way Trickle was designed to function in wireless sensor networks [9] and the way it is used by DNCP. This paper, to the best of the authors' knowledge, provides the first analysis and evaluation of DNCP in various large scale networks with different topology types.

C. Paper Outline

The remainder of this paper is organized as follows: Section II analyzes the basic behavior of DNCP, including its synchronization process and keep-alive mechanism. The number of packets and time needed for network convergence are analyzed when all the nodes are connected to a single link. To further study DNCP performance in more complex scenarios with different network topologies, network simulations are conducted in section III. Section IV concludes this paper .

II. Analysis of the Distributed Node Consensus Protocol

A. DNCP Overview

In a DNCP network, each node publishes a set of *node data* TLV (Type-Length-Value) tuples. DNCP defines three types of such TLVs: Peer, Keep-Alive interval and Trust-Verdict. Other TLVs are defined by DNCP profiles, e.g., assigned prefix, node address or node name TLVs, defined by HNCP.

A two level hash tree is used for synchronization, as shown in figure 1. Each *node data* (in ascending TLV order) is first hashed into a *Node State Hash* using a profile-specific hash function. Each of these Node State Hash is then concatenated with its respective node's sequence number to form a datum, and the *Network State Hash* is calculated over all the nodes' data in ascending order according to their node identifier, which represents the view that a node has of the network state. The protocol can thus verify if two nodes shares the same information by simply comparing the *Network State Hash*. The *Network State Hash* transmission is controlled by Trickle algorithm.

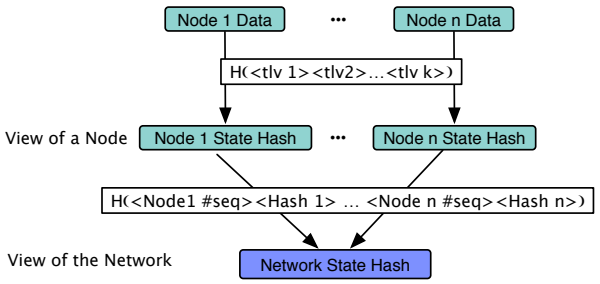

Figure 1. DNCP Hash Tree

The Trickle algorithm [2] is a distributed exponential back-off-based inconsistency detection algorithm. It limits its transmissions when the network is consistent but is reactive when an update needs to be done. The algorithm has 4 parameters:

- Redundancy constant k.
- The minimal interval size I_{min} and the maximal interval size I_{max}.
- The listen-only parameter η, defining the length of a listen-only period.

In addition to these parameters, Trickle requires *consistent* and *inconsistent* messages to be defined. In DNCP, a *consistent* message is a message which Network State Hash is the same as the locally computed Network State Hash. But DNCP makes a non-typical use of Trickle as it does not reset Trickle timer on receiving an *inconsistent* message. Instead, the Trickle timer is only reset when the locally computed Network State Hash changes. This approach offers the advantage of being less verbose and does not suffer from unidirectional links.

B. Synchronization Process

The synchronization process of DNCP is different from the protocols in which Trickle is commonly used (e.g., the IPv6 Routing Protocol for Low-Power and Lossy Networks [10]). Instead, Trickle is only used to detect differences in neighbor's *Network State Hash* (i.e., the root in figure 1), and unicast communications follow in order to synchronize states when necessary, as shown in figure 2. When Node B receives a *Network State TLV* from Node A, it first checks if the included network state hash is consistent with the locally calculated network state hash. If the hash values are different, it indicates that the Node A and Node B do not share a common view of the network and the following process is followed:

1) After waiting a random jitter between $[0, I_{min}/2]$ (to reduce the collision probability), Node B replies a *Request Network State TLV*.
2) Upon receipt of the *Request Network State TLV*, Node A sends a message including a *Node State TLV* for each node state used to calculate the network state hash. They contain node's meta-data such as the *Node State Hash*. The actual *Node Data* is not included.
3) Upon receipt of the *Node State TLVs*, Node B is able to detect which nodes have more recent information, and sends back *Request Node State TLVs* to the corresponding node.
4) Node A replies by sending *Network State TLV* containing the *Node Data TLV*s of the requested nodes.
5) Node B can thus update the local node data base with the *Node Data*s received, and reset the Trickle timer. The next Trickle message will be sent out after waiting $[\eta I_{min}, I_{min})$.

Compared to usual applications of the Trickle algorithm, DNCP has additional message exchanges and delays (due to jitter, round trip time of messages, etc.). In [5], it is proved that in a stable single link network with n nodes and redundancy constant set to k, the expected message count needed for a single update is:

For $\eta > 0$:

$$\mathbf{E}\left[N^{(k,n)}\right] \sim \frac{k}{\eta} - \frac{k}{\eta^2}\sqrt{\frac{\pi(1-\eta)}{2n}} + \mathcal{O}(n^{-1}) \qquad (1)$$

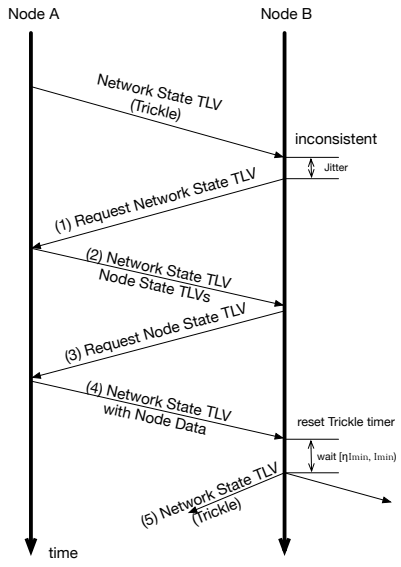

Figure 2. DNCP synchronization message exchange

which indicates the message count is bounded by k/η as $n \to \infty$, with convergence rate of $\sqrt{n}$.

In an equivalent DNCP network, when a single node updates its *Node Data*, other nodes will detect the change by the mean of an inconsistent Trickle-based *Network State* multicast transmission, and will then synchronize by the mean of 4 unicast transmissions. The additional DNCP overhead is thus $4 \times (n-1)$ packets transmissions for the whole network.

DNCP also performs an additional optimization when transmissions do not need to be encrypted. When all *Node State TLVs* can be put within a single multicast packet without exceeding the minimal IPv6 MTU, *Node State TLVs* without the node data are directly put in the Trickle-based multicast packets. In such situation, synchronization only requires 2 unicast transmissions instead of 4, inducing a reduced overhead of $2 \times (n-1)$ packets transmissions.

C. Keep-Alive Mechanism

The Trickle algorithm reacts fast on new information update, however, it may also prevent a node from sending messages for a long time. As shown in figure 3, the network is already converged at t_0, i.e., Node A and Node B share the same information. With a redundancy constant $k = 1$, Node A transmits at t_0, thus suppresses Node B's scheduled transmission at t_1. After the waiting interval of each node expires, Node A and Node B schedule the next transmission at t_2 and t_3 respectively. It turns out that t_2 is before t_3, the transmission of Node B is thus suppressed again.

When there is no other mean of detecting neighbors departure, DNCP makes use of a "keep-alive" mechanism. If it is applied per-link, in which multicast is supported in the network, a *Network State* message is sent on a link whenever no message was sent on that link for a defined keep-alive interval. This results in a lower-bound traffic of n transmitted

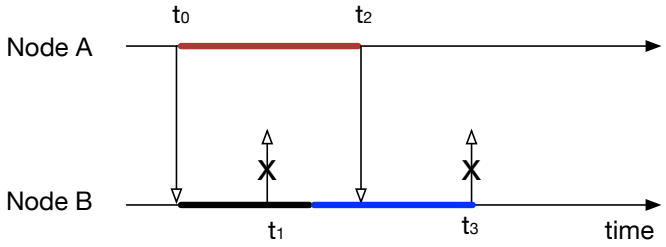

Figure 3. An example of Trickle message suppression

packet per keep-alive interval. If keep-alive is applied per-peer, in which case only unicast is used, the *Network State* messages are sent point-to-point, which produces at least $\binom{n}{2}$ per keep-alive messages interval.

Therefore, in either consistent state or inconsistent state, it is expected that DNCP will generate more traffic than most simple Trickle based protocols.

III. Protocol Evaluation

In the previous section, the behavior of DNCP is studied and the performance is analyzed in single link networks. To further investigate the characteristics of DNCP in scenarios with more complex topology types, it is evaluated with the help of ns3 network simulator.

This section first introduces the environment and protocol parameters used for the simulations, and then discusses the obtained results.

A. Simulation Environment

ns-3[2] is used for simulating the DNCP protocol. It is a discrete-event network simulator for Internet systems and the next generation of the famous ns-2 simulator (although the design is very different and they are not compatible). For the purpose of these simulations, *libdncp*[3] is modified so that it can run over the ns-3 simulator, and the same DNCP profile as the one specified by HNCP is used.

ns-3's CSMA (Carrier Sense Multiple Access) model is used to simulate Ethernet-like links. IPv6 and UDP are used as layer 3 and layer 4 protocol respectively. The simulation architecture is shown in figure 4. Each setting is run for 40 times.

The parameters are set as in table I. The default DNCP profile as specified in [3] is applied. As ns3 does not include a packet processing delay, an additional 1 millisecond delay is added between IP packet reception and packet processing by the DNCP layer. The link parameters are set as to provide a maximum throughput which DNCP is not expected to exceed.

Simulated networks have n (with n varies from 5 to 100) nodes. Two metrics are used to evaluate DNCP performance:

- *Protocol overhead:* The number of packets transmitted before convergence.
- *Convergence time:* The time needed for the network to converge.

[2]https://www.nsnam.org/
[3]https://github.com/sbyx/hnetd/

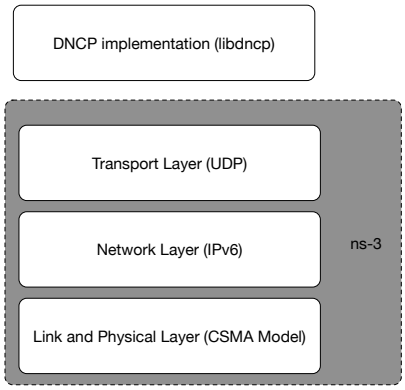

Figure 4. Simulation architecture with ns-3

Parameter	Value
CSMA data rate	1Gbps
Layer 2 MTU	1500 bytes
Propagation delay	1 microsecond
Packet processing delay	1 millisecond
Trickle redundancy constant k	1
Trickle I_{min}	200ms
Trickle I_{max}	40 seconds
Trickle listen-only parameter η	$1/2$
DNCP keep alive interval	24 seconds

Table I
SIMULATION SETTINGS

The following representative topology types are considered, as shown in figure 5.

- *Single link:* all the nodes are on the same link. Each node has $n-1$ neighbors.
- *Star:* consists of one single central node. All the other nodes have a single link to a central node.
- *String:* all nodes are chained one after the other. Each node has two neighbors (except the two extremities).
- *Tree:* a binary tree. Tree topology is typical in simple network deployment.
- *Double tree:* a variant of binary tree, in which each node is paired with a redundancy node. It is common in scenarios in which backup routes are desired, such as professional networks.

Two types of node updates are simulated:

- *Single Node Update*: The simulation starts will all nodes in a converged state, i.e., they already share the same information. A chosen node (the red ones in figure 5) will begin sending update information, and DNCP will be in charge of propagating the update to the whole network.
- *Network Initialization*: All the nodes simultaneously start synchronizing with each other until the network is converged. It is a worst case scenario in terms of convergence and may for instance happen after a power supply failure.

B. Simulation Results

1) Single Node Update: Figure 6 illustrates the packet overhead of single link networks. Each point represents the

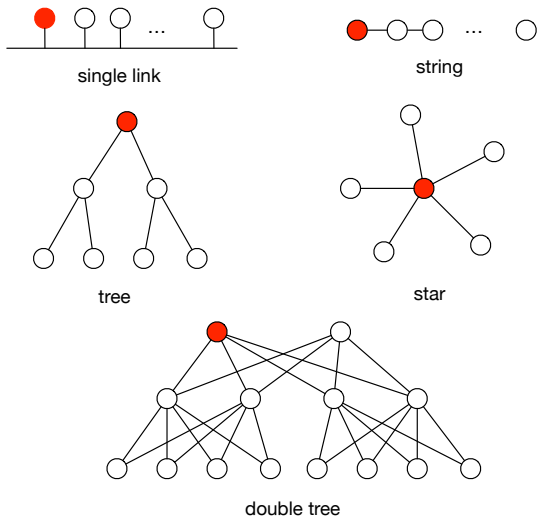

Figure 5. Simulation topology

value of one simulation. The analytic curve is also plotted following the analysis in section II-B. The simulation results are divided into two segments around 50 nodes. This is the threshold from which DNCP can not carry all *Node State TLV*s within a single multicast IPv6 MTU. Before the threshold, optimization is used. Also note that when there are more than 70 nodes, the overhead is higher than the expected value. This is due to that fact that more *Node State TLV*s have to be carried, causing packets to exceed the link MTU, and cause IP fragmentation.

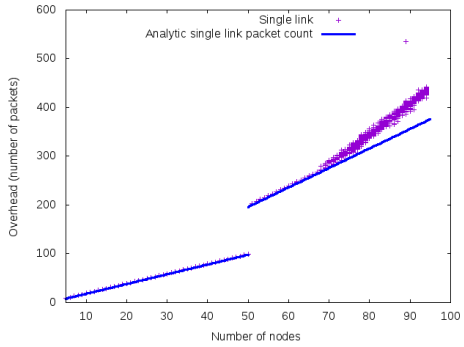

Figure 6. Protocol overhead of single link networks, single node update

Figure 7 and figure 8 depict the protocol overhead of star, string, tree, and double tree networks respectively. The segmentation due to the protocol optimization can also be observed.

Figure 9 shows the convergence time of single link and star networks. Both topology types have almost constant behavior as the number of node grows, except that the single link is segmented at the point of fragmentation (around 70 nodes). For the star networks, as it's the central node which triggers the update, the convergence time is much more stable. For the string topology (figure 10), the behavior is very different: the convergence time grows linearly with the number of nodes

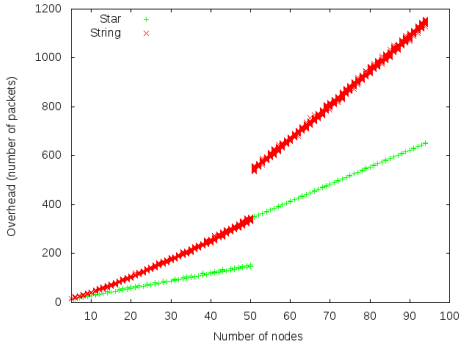

Figure 7. Protocol overhead of star and string networks, single node update

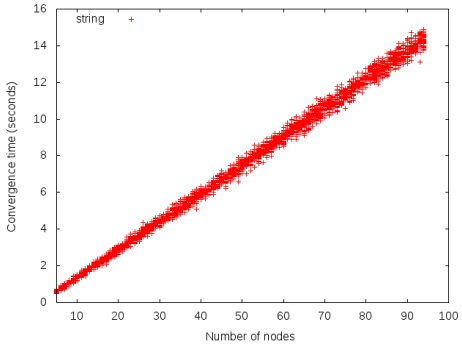

Figure 10. Convergence time of string networks, single node update

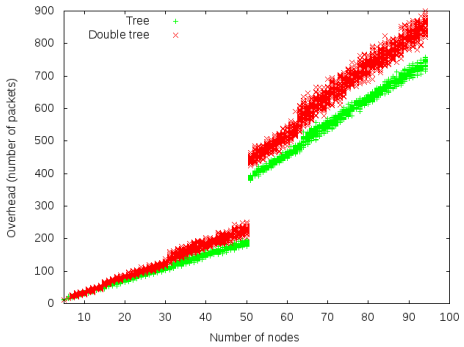

Figure 8. Protocol overhead of tree and double-tree networks, single node update

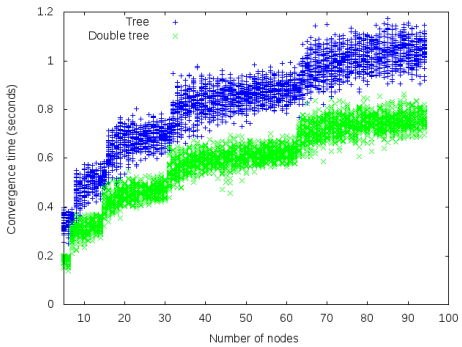

Figure 11. Convergence time of tree and double-tree networks, single node update

(i.e., number of hops of the networks), because the update has to be propagated hop-by-hop to the other end of the network. Each hop corresponds to ∼150 milliseconds, which is mainly due to the jitter and listen-only period, as indicated in figure 2. The impact of hops of the network can also be observed in figure 11 for tree topology types, in which each "step" indicates a depth growth of the tree.

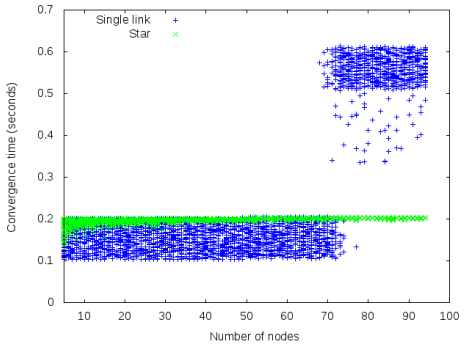

Figure 9. Convergence time of single link and star networks, single node update

2) Network Initialization: Figure 12 and 13 illustrate the protocol overhead of different topology types in network initialization scenarios. The single link networks have the most significant overhead, as each node have to synchronize with every other neighbor. Whereas for star networks, as all synchronizations have to go through the central node, the overhead can be greatly reduced. The double tree topology type, although with less hop count than the single tree topology type, has higher overhead. It can be concluded that the average number of neighbors is the most important factor of the packet overhead during the network initialization.

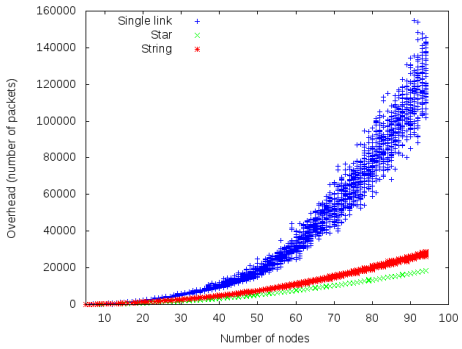

Figure 12. Protocol overhead of single link, star and string networks initialization

Figure 14 and 15 depict the convergence time for network initialization. In most scenarios (except the string topology type), the networks are able to converge in several seconds. The star topology type has the best performance, in which case the time required is close to constant.

The simulation results convey that for a given topology type,

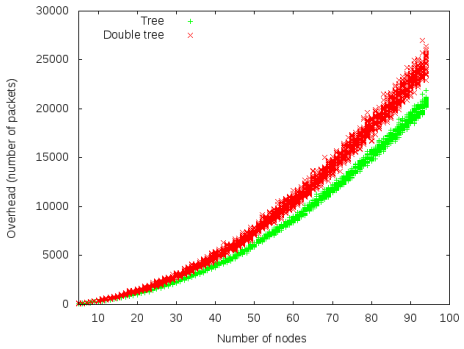

Figure 13. Protocol overhead of tree and double tree networks initialization

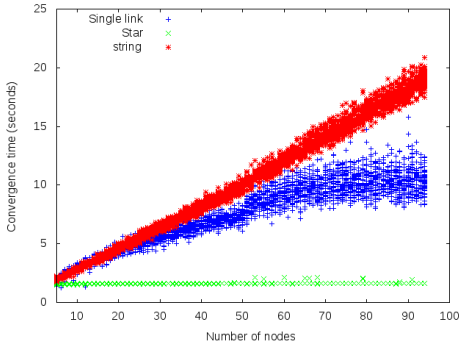

Figure 14. Convergence time of single link, star and string networks initialization

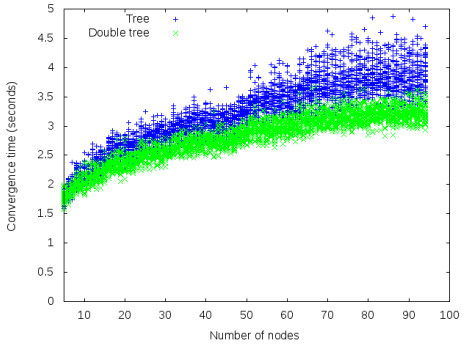

Figure 15. Convergence time of single tree and double tree networks initialization

the message overhead $N \propto N_b^2$ and $N \propto d$, where N_b is average number of neighbors and d is the network diameter. For the convergence time, it can be concluded that $T \propto N_b$ and $T \propto d$.

IV. Conclusion

DNCP, a fundamental block for home networks that synchronizes information between the nodes, is studied in this paper. The basic functions of DNCP, including hash tree, synchronization process and keep-alive mechanism are analyzed. The correctness of the protocol is verified.

To evaluate the protocol with different topology types and larger networks, the reference implementation *libdncp* is integrated in ns3 for simulation study. The protocol overhead and time needed for network convergence are measured using the same DNCP profile as HNCP.

The results show that the DNCP is able to converge quickly (less than 10 seconds) in most scenarios, for both single node update and network initialization. On the other hand, the protocol overhead depends on the topology type of the network. In dense networks (such as single link topology), non-trivial amount of packets are required to initialize the network. Overall, the simulation results show that DNCP behaves correctly in typical home networks.

For a specific topology, the number of nodes has significant impact to the packet overhead. If the *node state* TLVs can be carried in one Trickle multicast packet, half of the packets can be saved by using the multicast optimization. From the aspect of time needed for convergence, it is linear with the radius of the network.

Future work includes evaluating performances of DNCP in networks with fragile links, as well as optimizing the overhead in dense scenarios.

Acknowledgement

This research was partially supported by Ecole Polytechnique-Cisco Chaire. We thank Thomas Clausen and Mark Townsley for valuable comments and advices during the course of the research.

References

[1] M. Stenberg and S. Barth, "Distributed Node Consensus Protocol," IETF Internet Draft, work in progress, IETF, August 2015.

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

[3] M. Stenberg, S. Barth, and P. Pfister, "Home Networking Control Protocol," IETF Internet Draft, work in progress, IETF, August 2015.

[4] R. Coltun, D. Ferguson, J. Moy, and A. Lindem, "OSPF for IPv6," IETF RFC 5340, IETF, July 2008.

[5] T. Meyfroyt, S. Borst, O. Boxma, and D. Denteneer, "On the scalability and message count of trickle-based broadcasting schemes," *Queueing Systems*, vol. 81, no. 2-3, pp. 203–230, 2015. [Online]. Available: http://dx.doi.org/10.1007/s11134-015-9438-x

[6] P. Levis, E. Brewer, D. Culler, D. Gay, S. Madden, N. Patel, J. Polastre, S. Shenker, R. Szewczyk, and A. Woo, "The emergence of a networking primitive in wireless sensor networks," *Communication of the ACM*, vol. 51, no. 7, pp. 99–106, 2008.

[7] H. Kermajani, C. Gomez, and M. Arshad, "Modeling the message count of the trickle algorithm in a steady-state, static wireless sensor network," *Communications Letters, IEEE*, vol. 16, no. 12, pp. 1960–1963, December 2012.

[8] T. Meyfroyt, S. Borst, O. Boxma, and D. Dentenner, "A data propagation model for wireless gossiping," *Performance Evaluation*, vol. 85-86, pp. 19–32, 2015.

[9] P. Levis, N. Patel, D. Culler, and S. Shenker, "Trickle: A self-regulating algorithm for code propagation and maintenance in wireless sensor networks," in *Proceedings of the 1st Conference on Symposium on Networked Systems Design and Implementation - Volume 1*, ser. NSDI'04. Berkeley, CA, USA: USENIX Association, 2004, pp. 2–2. [Online]. Available: http://dl.acm.org/citation.cfm?id=1251175.1251177

[10] T. Winter, P. Thubert, A. Brandt, J. Hui, R. Kelsey, P. Levis, K. Pister, R. Struik, and J. Vasseur, "RPL: IPv6 Routing Protocol for Low power and Lossy Networks," RFC 6550, IETF, March 2012.