ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ
ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

«Παροχή υπηρεσιών με αυτόνομα χαρακτηριστικά σε δυναμικό περιβάλλον»

ΔΙΔΑΚΤΟΡΙΚΗ ΔΙΑΤΡΙΒΗ

Παναγιώτης Α. Γκουβάς

Αθήνα, Απρίλιος 2011

NATIONAL TECHNICAL UNIVERSITY OF ATHENS
SCHOOL OF ELECTRICAL AND COMPUTER ENGINEERING

«Service Provision with autonomic characteristics in mesh environments»

PHD THESIS

Panagiotis A. Gouvas

Athens, April 2011

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ
ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

«Παροχή υπηρεσιών με αυτόνομα χαρακτηριστικά σε δυναμικό περιβάλλον»

ΔΙΔΑΚΤΟΡΙΚΗ ΔΙΑΤΡΙΒΗ

Παναγιώτης Α. Γκουβάς

Συμβουλευτική Επιτροπή : Γρηγόριος Μέντζας
Δημήτριος Ασκούνης
Ιωάννης Ψαρράς

Εγκρίθηκε από την επταμελή εξεταστική επιτροπή την ……

.....	.....	.....
Γρηγόριος Μέντζας Καθηγητής Ε.Μ.Π.	Δημήτριος Ασκούνης Αν. Καθηγητής Ε.Μ.Π.	Ιωάννης Ψαρράς Καθηγητής Ε.Μ.Π.
.....	.....	.....
Μιχαήλ Θεολόγου Καθηγητής Ε.Μ.Π.	Γεώργιος Στασινόπουλος Καθηγητής Ε.Μ.Π.	Παναγιώτης Τσανάκας Καθηγητής Ε.Μ.Π.

.....
Μάριος Δικαιάκος
Αν. Καθηγητής
Πανεπιστήμιο Κύπρου

Αθήνα, Απρίλιος 2011

.....

Παναγιώτης Α. Γκουβάς

Διδάκτωρ Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Η παρούσα διατριβή αφιερώνεται
στην μνήμη του παππού μου Σπύρου,
η σοφία του οποίου είναι οδηγός …

Abstract

This doctoral thesis focuses on the domain of mobile Peer-to-Peer applications and specifically it provides a Distributed Hash Table (DHT) architecture and implementation that is able to bootstrap and operate in Mesh Environments without any form of centralization. Merits of DHTs, such as decentralization, scalability and fault-tolerance, gave them a dominant position in the architecture of Peer-to-Peer systems in fixed Networks. However, severe problems arise when as-is DHT algorithms are ported in Mesh Environments. The contribution of the present doctoral thesis can be summarized in four main axes:

The first axis refers to the identification of the problems that arise when existing DHT implementations are ported to a Mesh Environment. The majority of problems exist because already developed DHTs base their operation to the efficiency and robustness of the underlying fixed Networking environment which is not considered as granted at Mesh Environments.

The second axis focuses on an architectural proposal that overcomes the identified problems. The architectural proposal relies on principles of autonomicity and introduces a layered approach that can be realized in various ways. This layered approach is benchmarked in a simulator and specific modifications are further introduced in order to achieve scaling and efficiency issues.

The third axis refers to a reference Implementation of the proposed layered approach. This reference implementation will be addressed as UbiChord. Since, specific technical choices have been followed in order to realize the layered approach, these choices will be analyzed. Finally the fourth axis focuses on the creation and the benchmarking of indicative Mobile Peer-to-Peer Applications based on UbiChord implementation.

An extensive set of simulations and emulations have been conducted in the frames of this thesis. Simulations refer to the simulations model that has been developed while emulations refer to the actual UbiChord implementation. UbiChord can be used to create mobile Peer-to-Peer services such as mobile social networking, file sharing etc.

Keywords: Mobile Peer-to-Peer, DHT, Gossiping, Overlay Networks, MANET, Mesh, Reactive Routing, Autonomic Computing

Περίληψη

Η παρούσα διδακτορική διατριβή εστιάζει στο πεδίο των υπηρεσιών σε κινητούς ομότιμους κόμβους. Πιο συγκεκριμένα, στα πλαίσια της διατριβής προτείνεται μια αρχιτεκτονική και αναλύεται μια υλοποίηση ενός Κατανεμημένου Πίνακα Κατακερματισμού (Distributed Hash Table-DHT) η οποία είναι ικανή να λειτουργήσει χωρίς καμία κεντρικοποίηση. Τα ιδιαίτερα χαρακτηριστικά των DHT όπως η αποκέντρωση, η επεκτασιμότητα και η ανεκτικότητα, τους δίνει μια δεσπόζουσα θέση στην αρχιτεκτονική των δικτύων ομότιμων κόμβων σε σταθερά δίκτυα. Ωστόσο, σοβαρά προβλήματα προκύπτουν όταν υπάρχουσες υλοποιήσεις 'μεταφέρονται' σε κινητά δίκτυα. Η συνεισφορά της παρούσας διδακτορικής διατριβής μπορεί να συνοψισθεί σε τέσσερις κύριους άξονες:

Ο πρώτος άξονας αφορά την αναγνώριση των προβλημάτων που δημιουργούνται, όταν οι υπάρχουσες υλοποιήσεις DHT μεταφερθούν σε ένα δίκτυο κινητών κόμβων. Η πλειοψηφία των προβλημάτων απορρέουν από το γεγονός ότι οι υπάρχουσες υλοποιήσεις βασίζονται στην αξιοπιστία και την αποδοτικότητα του υποκείμενου σταθερού δικτύου. Αυτά τα χαρακτηριστικά δεν υπάρχουν σε ασύρματα μη κεντρικοποιημένα δίκτυα.

Ο δεύτερος άξονας επικεντρώνεται σε μια αρχιτεκτονική πρόταση για την υπέρβαση των παραπάνω προβλημάτων. Η αρχιτεκτονική πρόταση βασίζεται στις αρχές των αυτόνομων συστημάτων και εισάγει μια πολυεπίπεδη προσέγγιση που μπορεί να υλοποιηθεί με διάφορους τρόπους. Η προσέγγιση αυτή αρχικά αξιολογείται σε έναν προσομοιωτή και κατόπιν ειδικές τροποποιήσεις εισάγονται προκειμένου να επιτευχθούν μεγαλύτερη κλιμάκωση και αποτελεσματικότητα.

Ο τρίτος άξονας αναφέρεται σε μια υλοποίηση της προτεινόμενης αρχιτεκτονικής. Αυτή η υλοποίηση ονομάζεται UbiChord. Δεδομένου ότι συγκεκριμένες τεχνικές επιλογές ακολουθήθηκαν προκειμένου να υλοποιηθεί η πολυεπίπεδη προσέγγιση, αυτές οι επιλογές αναλύονται. Τέλος, ο τέταρτος άξονας επικεντρώνεται στη δημιουργία και τη συγκριτική αξιολόγηση κάποιων ενδεικτικών εφαρμογών που βασίζονται στην εφαρμογή του UbiChord.

Ένα εκτεταμένο σύνολο προσομοιώσεων και εξομοιώσεων έχουν διεξαχθεί στα πλαίσια της παρούσας διατριβής. Οι προσομοιώσεις αναφέρονται στο μοντέλο προσομοίωση που έχει αναπτυχθεί, ενώ η εξομοίωση αναφέρεται στην πραγματική εφαρμογή. Το UbiChord μπορεί να χρησιμοποιηθεί για τη δημιουργία υπηρεσιών σε ασύρματα δίκτυα όπως εφαρμογές κοινωνικής δικτύωσης, διανομής αρχείων κ.λ.π.

Λέξεις Κλειδιά: κινητό δίκτυο ομότιμων κόμβων, Κατανεμημένος Πίνακας Κατακερματισμού, υπερκείμενα δίκτυα

Ευχαριστίες

Η παρούσα διδακτορική διατριβή αποτελεί το επιστέγασμα μιας προσπάθειας πέντε ετών, στα πλαίσια του προγράμματος μεταπτυχιακών σπουδών του τμήματος Ηλεκτρολόγων Μηχανικών και Μηχανικών Η/Υ του Εθνικού Μετσόβιου Πολυτεχνείου. Η συναναστροφή με συναδέλφους, αλλά και το κλίμα δημιουργικότητας αποτέλεσαν βασικές πηγές έμπνευσης και συνέβαλλαν σημαντικά στη βελτίωση της προσωπικής αντιμετώπισης και επίλυσης ερευνητικών προκλήσεων.

Το αποτέλεσμα που παρουσιάζεται στις σελίδες αυτές οφείλεται στο μέγιστο βαθμό στη βοήθεια και στην καθοδήγηση που είχα από τον επιβλέποντα Καθηγητή κ. Γρ. Ν. Μέντζα. Του οφείλω ιδιαίτερες ευχαριστίες για τις ευκαιρίες που μου προσέφερε και την πίστη του σε μένα. Τα μαθήματα επιστημονικής κατάρτισης, ερευνητικού ζήλου, αλλά και ηθικής ακεραιότητας που πήρα από αυτόν αποτελούν τα σημαντικότερα εφόδια για τη μελλοντική μου πορεία.

Θα ήθελα να ευχαριστήσω τα άλλα δύο μέλη της τριμελούς εισηγητικής μου επιτροπής, τον Καθηγητή κ. Ι. Ψαρρά και τον Αναπληρωτή Καθηγητή κ. Δ. Ασκούνη, καθώς και τους Καθηγητές Μιχαήλ Θεολόγου, Γεώργιο Στασινόπουλο, Παναγιώτη Τσανάκα και τον Μάριο Δικαιάκο για την τιμή που μου έκαναν να συμμετάσχουν στην επιτροπή εξέτασης της διατριβής.

Θέλω επίσης να ευχαριστήσω τους συναδέλφους μου Γιάννη Βεργινάδη, Σπύρο Ντιούδη, Χαράλαμπο Μαγγούτα και Ευθύμιο Μπόθο που υπήρξαν αρωγοί και συμπαραστάτες σε όλη αυτή την πορεία. Ιδιαίτερες ευχαριστίες οφείλω στους Δημήτριο Αλεξάνδρου και Θανάση Μπούρα που εκτός από συνάδελφοι στα πλαίσια του εργαστηρίου είναι ιδιαίτεροι φίλοι και συνοδοιπόροι στον επαγγελματικό στίβο. Η συμβολή τους στην επίτευξη των στόχων μου είναι τεράστια και το κλίμα ειλικρίνειας και συνεργασίας μεταξύ μας αναντικατάστατο. Ιδιαίτερη μνεία οφείλω στους επιστημονικούς μου συνεργάτες και φίλους Αθανάσιο Λιακόπουλο, Σπύρο Μαντζουράτο και Αναστάσιο Ζαφειρόπουλο για την συμβολή τους στην διατριβή μου, στη Γιώτα Βελέντζα αλλά και στους αδελφικούς μου φίλους Σπύρο Μεταλλινό και Πάρη Λιάπη για την στήριξη που μου παρείχαν.

Ολοκληρώνοντας θα ήθελα να απευθύνω ένα μεγάλο ευχαριστώ στους γονείς μου Αγγελική και Αγαθοκλή, στην αδερφή μου Ελένη στην γιαγιά μου Μηλιά, για την αμέριστη αγάπη και ηθική υποστήριξη που μου προσέφεραν όλα αυτά τα χρόνια.

Γκουβάς Α. Παναγιώτης
Απρίλιος 2011

Table of Contents

Table of Figures

1. Introduction & Motivation

1.1 Evolution of Mobile Networks & Provided Services

The emergence of mobile networks has been tremendous during the last ten years. It is remarkable that while internet users have increased approximately 5 times the last decade, mobile cellular subscriptions have increased 60 times during the same period. This trend is illustrated at Figure 1.1 [ITU Report, 2010] where a comparative graph is provided.

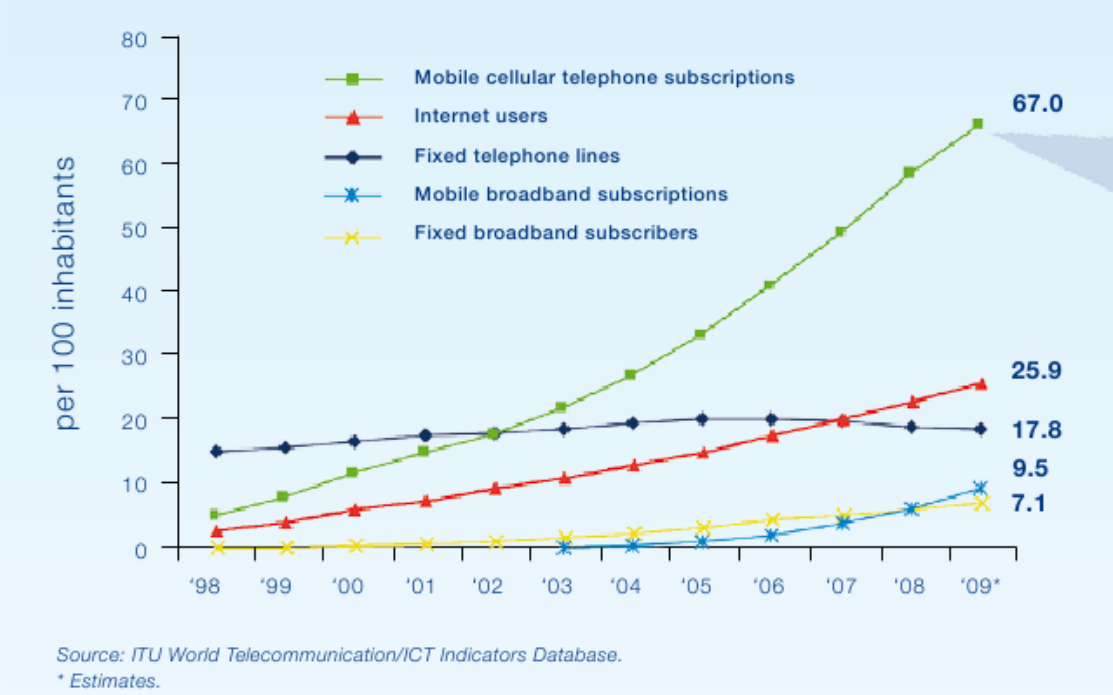


Figure 1.1 - Subscriptions per 100 inhabitants

However, the most significant observation from Figure 1.1 is that in a few years the majority of internet users will be mobile users. This conclusion can be derived by the facts that a) many operators have evolved their infrastructure to support High Speed Packet Access (HSDPA[1]) services to their customers and b) the problem of IP scarcity that is raised by IPv4's lack of free addresses is sufficiently solved by IPv6[2].

Indeed, today more than 151 countries and more than 373 operators haveadopted HSPA [3GAmericas, 2010]. Furthermore, according to the two most prominent predictive models[3] for IPv4 exhaustion, IPv4 address space will be exhausted between17-Jun-2011 (IANA prediction) and 19-Feb-2012 (RIR prediction). This trend is depicted at Figure 1.2 where the projection of reserved addresses based on data up to 2008 is provided.

[1] www.umtsworld.com/technology/hsdpa.htm
[2] www.ipv6.org
[3] http://www.potaroo.net/tools/ipv4/index.html

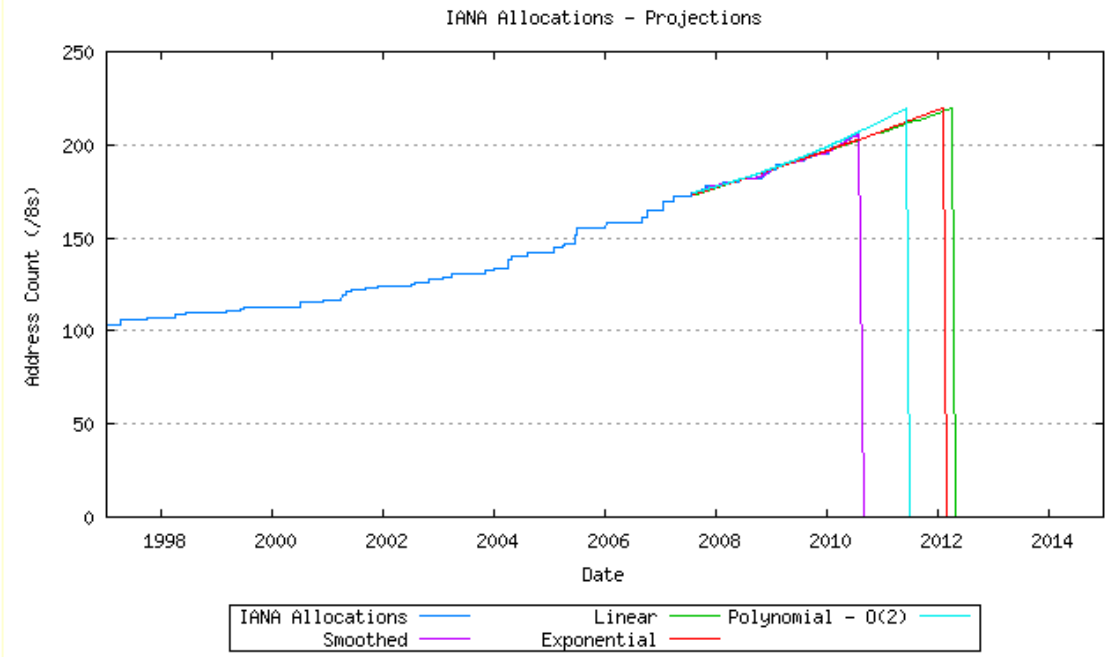


Figure 1.2 - IPv4 exhaustion prediction (IANA)

Conclusively, IPv6 will inevitably drive the Internet locomotive since it provides a robust solution for addressing (128 bit address space instead of 32bit address space of IPv4) and contains quite a few supplementary features that impose its adoption. Such indicative features are native Mobile IP support, stateless auto-configuration, extensible protocol stack etc. IPv6 adoption rate is reflected in Figure 1.3.

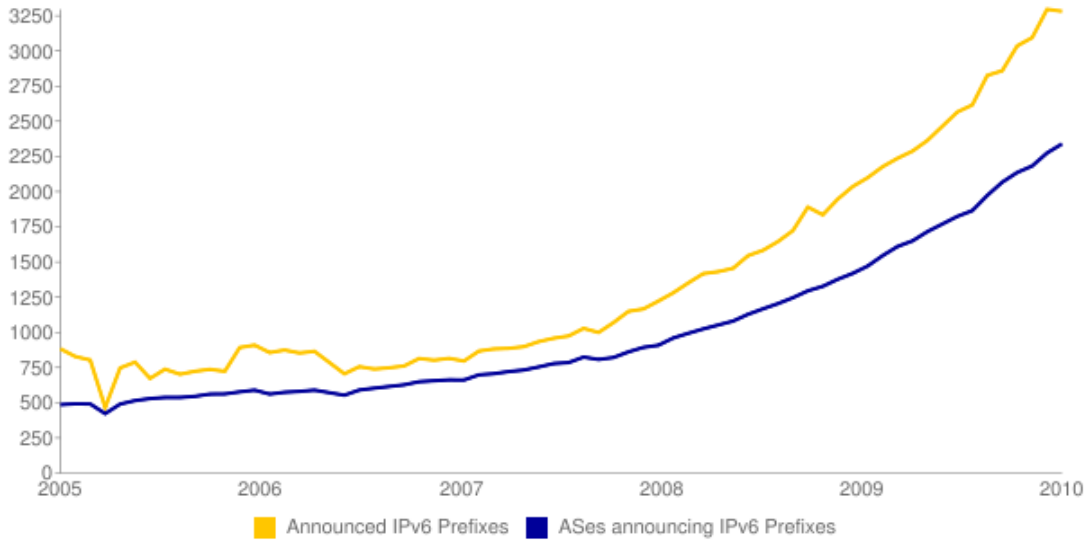


Figure 1.3 - IPv6 announced Prefixes

The graphs above represent the IPv6 address space that is actually in use on the Internet, as seen by the RIPE NCC's Routing Information Service (RIS)[4]. The data is near-real time, based on data gathered by the Internet Number Resource Database (INRDB)[5].This blue line in the graph represents the number of Autonomous Systems announcing IPv6 addresses, while the yellow one represents the total number of IPv6 prefixes announced to the Internet.

Emergence of mobile networks is also verified by the announced governmental expenditures of most developed countries. Indicatively, in Figure 1.4, it is shown that the scheduled US government expenditures will increase 4 times between 2007 and 2011 as far as wireless data infrastructure is concerned [ATT Report, 2007].

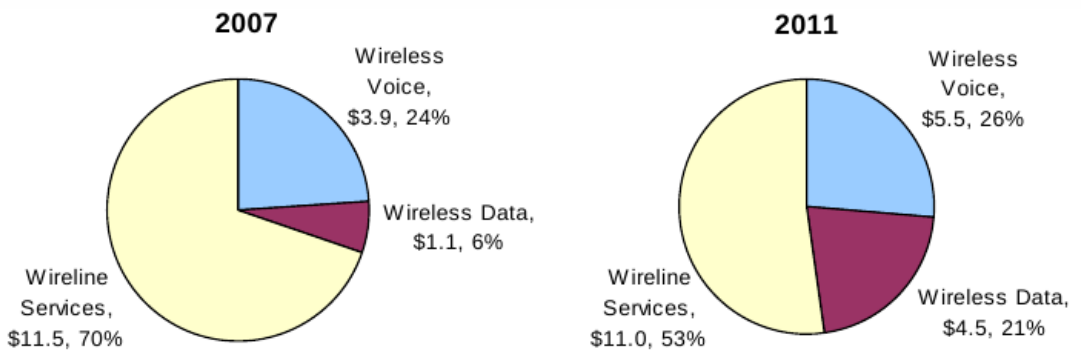


Figure 1.4 - US Government expenditures in Billion $

However, the most revolutionary aspect of the wireless networking era is Mesh Networking. Mesh networking is a type of networking where each node in the network may act as an independent router, regardless of whether it is connected to another network or not. Mesh networking implies multi-hop connectivity without static routing schemes. Mesh networking has been the main area of research for more than 10 years now. This is also reflected at the innovation Hype provided by Gartner at 2006 [Gartner, 2006] (see Figure 1.5).

It is remarkable that the hype's predictive trend has been affirmed since in less than 5 years (where Mesh Networking was in the Trough of Disillusionment) concrete Mesh Networking protocols have been formulated such as 802.11s[6], Zigbee[7] etc that serve diverse needs (low power consumption, mobility, security without centralization etc). Such protocols have already reference implementations by vendors and, consequently, they belong to the Plateau of Productivity regarding their Hype classification (see Figure 1.5).

[4]http://www.ripe.net/projects/ris/index.html
[5]http://labs.ripe.net/node/45
[6] www.ieee802.org/11
[7]www.zigbee.org

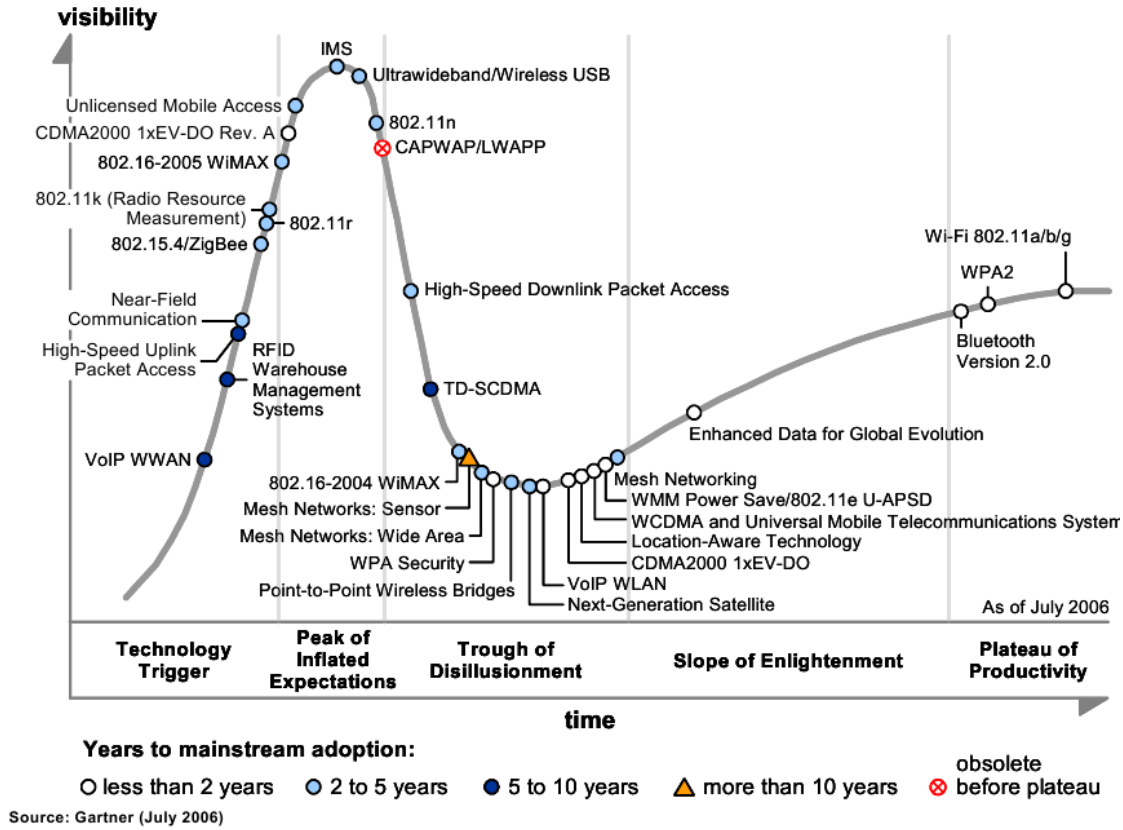
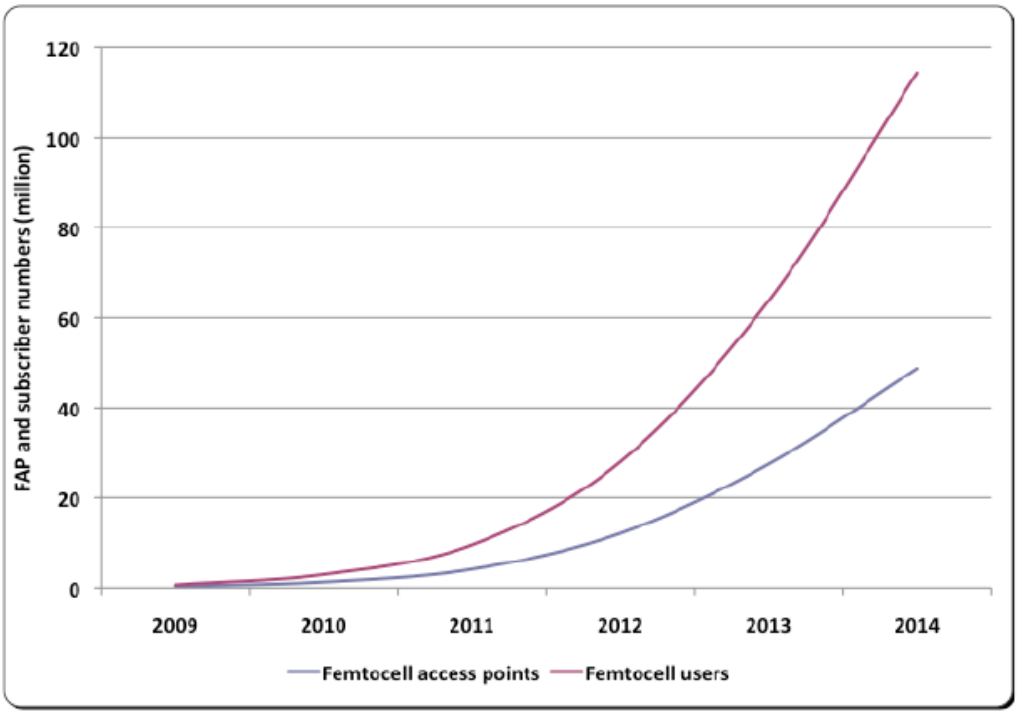


Figure 1.5 - Hype cycle for Wireless Networking

However, the most significant indication of the Mesh Networking emergence is the Mobile Operator's architectural choice to expand their network with femtocell endpoints. Femtocells are sold by Mobile Network Operators to their residential end-users or enterprise customers. A femtocell is typically the size of a residential gateway or smaller, and connects to the end-user's broadband line. Integrated femtocells (which include both a DSL router and femtocell) also exist. Once plugged in, the femtocell connects to the MNO's mobile network, and provides extra coverage in a range of typically 30 to 50 meters for residential femtocells (depending on the existing coverage and output power — usually 20mW which is five times less than a Wi-Fi router). From an end-users' perspective, it is plug and play since there is no specific installation or prerequisite technical knowledge.

The prediction of the femtocell-roll out is provided at Figure 1.6 where it could be argued that the tension is logarithmic since every year the expected amount of installed femtocell access points is doubled.



Source: Informa Telecoms & Media

Figure 1.6 - Femtocell roll-out prediction [Informa, 2010]

Consequently, while the infrastructural scenery is highly complex, a clear view about the capabilities of future networks and mobile nodes is perceptible even from today. Devices will be equipped with protocols that will allow Mesh communication and networks to be expanded under the femtocell approach. On the other hand, in the field of provided services the field is totally unclear.

The main reason for this uncertainty is related to the fact that the Mobile Network Operators (MNOs) and handset OEMs move their industries' structure from vertical to horizontal and vice versa [Visionmobile, 2009]. Therefore, the notion of service value is continuously changing. Differences between vertical and horizontal structures are depicted at Figure 1.7. A vertical industry structure implies the usage of already integrated components with proprietary interfaces. Barriers for entering the market are relatively high and the emphasis is given to the guaranteed performance.

In contradiction to the vertical structure, a horizontal industry structure emphasizes in the usage of modular components with standardized interfaces. Barriers for entering the market are low and the emphasis is given to flexibility and customization. In order to predict the nature of the provided mobile services in the near future, the helix for both Mobile Network Operators (MNOs) and OEM handset producers must be compared. These helixes are illustrated at Figure 1.8 and Figure 1.9 respectively.

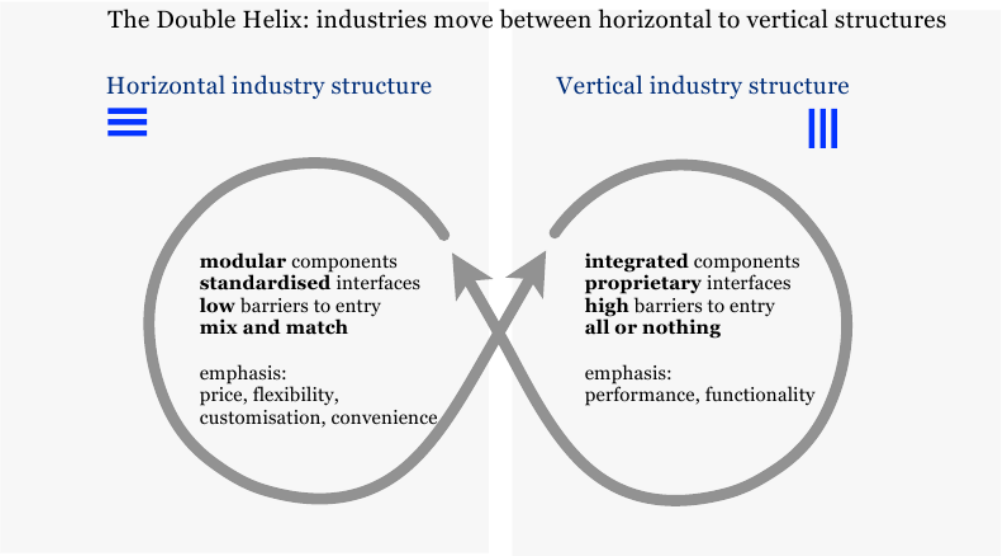


Figure 1.7 - Vertical/Horizontal structure helix

MNO's helix (see Figure 1.8) indicates that network operators are positioned at the horizontal industry structure. This practically means that their infrastructure up to 2010 is fully interoperable since all interfaces used are standardized. Specifically, the adoption of IP-Multimedia Subsystem (a.k.a. IMS[8]) of all MNOs was one of the biggest success stories for standardization and modularization according to 3GPP [9].

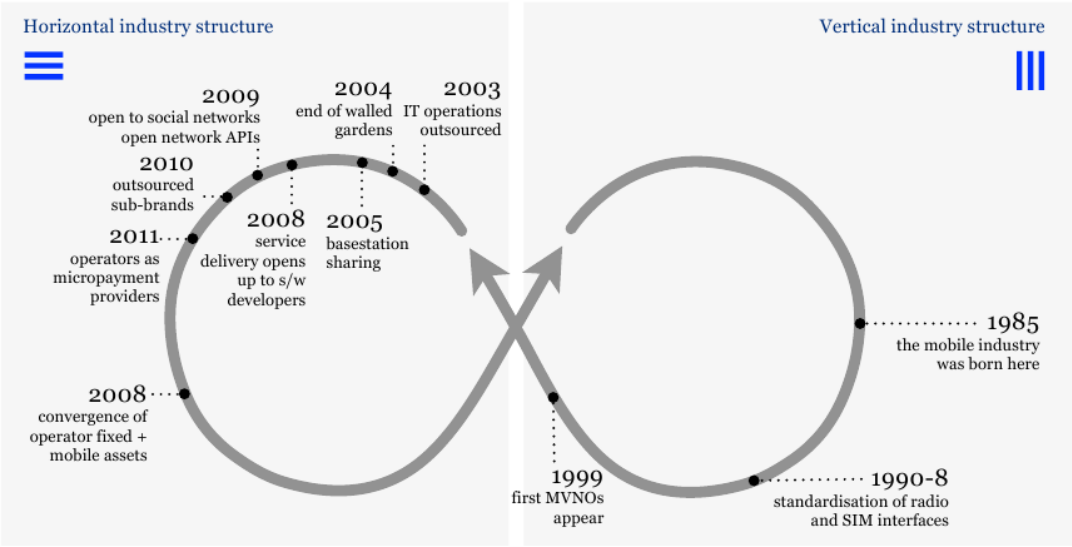


Figure 1.8 - MNO helix

[8] www.3gpp1.net/article/ims
[9] http://www.3gpp.org

Moreover, OEM producers have moved the last four years (i.e. 2006 - 2010) to the vertical helix and this can be easily inferred by the fact that all handset producers have created closed Application Repositories (a.k.a. AppStores) bounded to the capabilities of their products (see Figure 1.9).

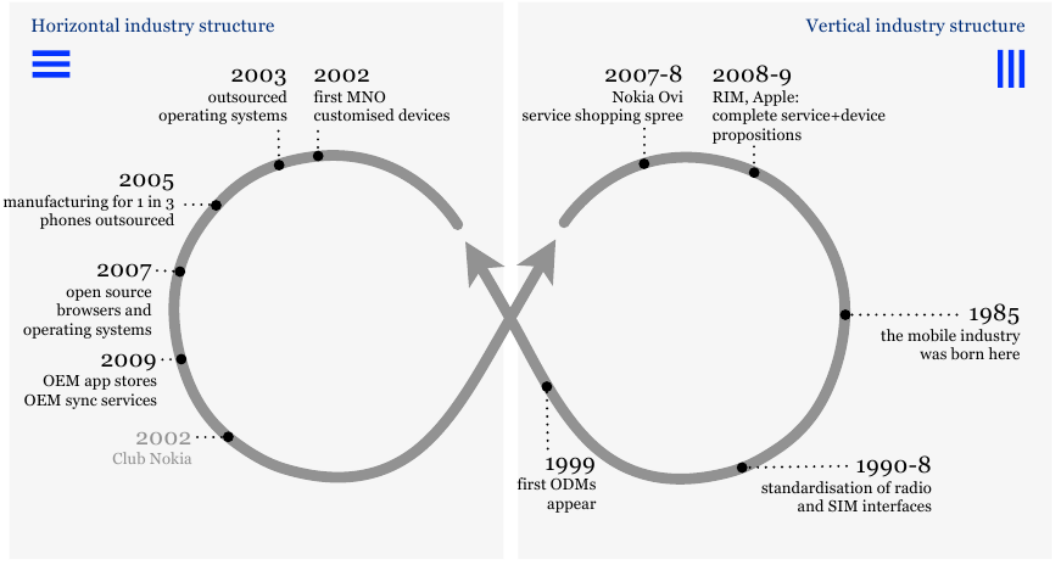


Figure 1.9 - OEM helix

This contradictive tension has already been identified. Additionally, there is an estimation that Handset OEMs move twice as fast as MNOs on the helix [Visionmobile, 2009]. Under this perspective, the notion of the delivered mobile service "value" has been radically redefined. As a result, today a service value can be classified in one of the quadrants that are formulated by the axis of *when* the service value is created and *where* the service value is created (see Figure 1.10). "*When*" refers to pre-sales or post-sales and "*where"* refers to the device itself or to cloud consisting of other devices. This service-value classification is extremely crucial because based on this classification someone can judge upon the viability or predict the sustainability of a new or existing service class.

The most emerging mobile service classes for 2010 are according to [ReadWrightWeb, 2010] the following: geo-location services, Internet of Things services, Augmented Reality services, and mobile social networking services. Additionally, at Gartner's press release regarding "Top 10 Consumer Mobile Applications for 2012"[10], the following additional service-classes are identified: Money Transfer & Mobile payment, Mobile Health Monitoring, Mobile Advertising and Mobile Instant Messaging.

[10]http://www.gartner.com/it/page.jsp?id=1230413

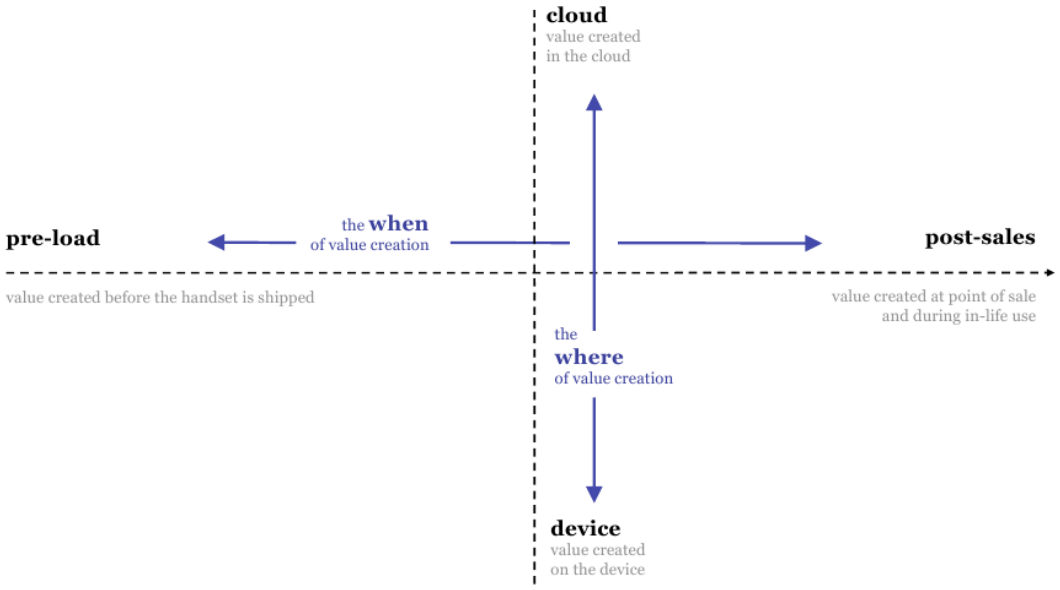


Figure 1.10 - Mobile-Service "value" redefined

It is remarkable that the majority of the emerging services are placed on the cloud quadrants (see Figure 1.10) which implies that the application value is derived by the interaction of a mobile node with other nodes in the network either fixed (in case of a Mobile Payment Gateway etc.) or mobile (in case of Mobile Gaming, Social Networking Applications etc.). This peer-to-peer nature of the emerging services can also be predicted by the already existing traffic distribution in fixed networks (see Figure 1.11).

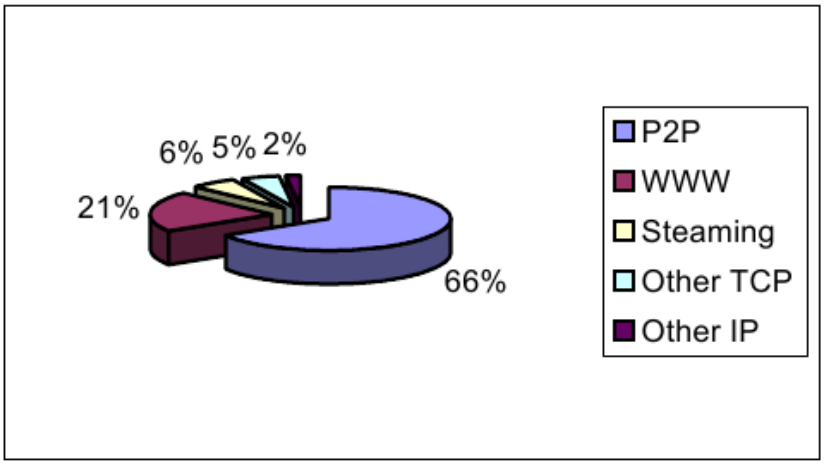


Figure 1.11 - Traffic Distribution in fixed Networks [Y. Raivio, 2005]

Therefore, in the mobile services domain there is a new emerging trend for Mobile Peer-to-Peer applications. This trend is empowered by a) the Mesh networking protocol evolution, b) the increasing mobile nodes capabilities and c) the infrastructural support by MNOs.

1.2 Motivation - Main Objectives

The trend towards the Future Internet is the transition to networks that provide highly distributed, pervasive and communication-intensive services. These services pose new requirements on the underlying network infrastructure [A. Galis et al., 2009] [R. Bless et al., 2008] that derive from the need to operate on top of dynamic, heterogeneous and complex networks. In order to cope with these requirements, such services will be expected to (i) be aware of the context and the environment in which they operate, (ii) self-configure and self-adapt according to the network conditions that they sense and (iii) require minimum feedback from the end-user avoiding any explicit human intervention.

These challenges can be addressed through the creation and maintenance of overlay networks [G. Kunzmann et al., 2008]. Overlay networks create a well-defined virtual topology above the underlying network infrastructure, where advanced services may be provided. However, new methodologies for implementing and operating overlays are needed. In particular new mechanisms are required that permit overlays to structure their topology, define their routing scheme, and manage their resources independently [K. Tutschkua et al., 2008].

The transition to overlay networking and the incorporation of autonomic characteristics in next generation networks is most crucial in ad-hoc and mesh networks since they present specific challenges, especially due to the following characteristics: (i) dynamic topology (links are established / torn down frequently), (ii) unreliable operation (nodes may be disconnected at any time), (iii) complexity (lack of centralized control or network hierarchy) and (iv) loose control of nodes (roles in the overlay network are automatically assigned).

Peer-to-peer (p2p) networks constitute the best candidate for addressing most of these challenges (unreliable operation, complexity, loose control of nodes) [E. K. Lua et al., 2005]. In p2p networks, a dynamic assignment of tasks among peers is realized and the provided services are based on their direct cooperation. Peer nodes share the available resources, provide redundancy due to distribution and replication of data and thus impose minimal requirements to the infrastructure. Ad-hoc deployment increases the self-organization level of the network and permits its operation also in an unstable environment with loosely connected nodes. Furthermore, despite the apparent chaos of periodic random changes to the membership of the network, p2p networks provide provable guarantees about performance.

However, the adoption of current p2p techniques necessitates the existence of a fixed network topology [M. Bisignano et al., 2007] [M. Caleffi et al., 2009]. With the term “fixed”, we refer to wired and wireless networks that have formulated their physical topology and do not present any topology changes. Over a formulated physical topology, specific p2p functions can be applied and a wide range of services can be deployed. This is

not the case in Mesh Environments. Each device in a Mesh Environment is free to move independently in any direction, and thus frequently change its connectivity to other devices.

Traditional Peer-to-Peer applications were considered revolutionary, as aforementioned, however all implemented Peer-to-Peer protocols base their operation to the reliability and centralization of the underlying fixed network infrastructure. Indicatively, regarding "(i) dynamic topology issues" in order for a node to identify its relative position in the overlay it consults a super-peer which has a public IP in the fixed network. Regarding, "(ii) unreliable operation" in order for a node to identify if another node is unreachable it relies on TCP timeout mechanism. Consequently, Mobile Peer-to-Peer protocols are very challenging domain since traditional protocols cannot be used or ported as-is in the Mesh Networking domain.

The cornerstone of most P2P systems is a Distributed Hash Table (a.k.a. DHT [H. Balakrishnan et al., 2003]). DHTs are a class of decentralized distributed systems that provide a lookup service similar to a hash table i.e. *(key, value)* pairs are stored in the DHT, and any participating node can efficiently retrieve the value associated with a given key. Responsibility for maintaining the mapping from keys to values is distributed among the nodes, in such a way that a change in the set of participants causes a minimal amount of disruption. This allows DHTs to scale to extremely large numbers of nodes and handle continual node arrivals, departures and failures.

The goal of this Thesis is to provide a Distributed Hash Table architecture and implementation that is able to bootstrap and operate in Mesh Environments without any form of centralization. This DHT will be addressed as UbiChord. UbiChord can be used as a cornerstone in order to create Mobile Peer-to-Peer applications. In the frames of this Thesis, the following objectives have been identified:

Objective 1: to identify the key problems for bootstrapping and maintaining a distributed key-value store in a Mesh Environment. Since DHTs are considered the cornerstone of p2p applications, it is highly important to analyze why traditional practices that are well established and applied in fixed networks cannot be ported as-is in mesh environments.

Objective 2: to propose and evaluate a protocol that overcomes the identified problems. More specifically, since there are specific problems that prevent existing DHTs operation, a new protocol that overcomes these problems must be proposed. This protocol will be tailored for mesh environments.

Objective 3: to provide Reference Implementation of the proposed protocol. This objective is the realization of objective-2 since in the course of this objective a real implementation of the afore-proposed protocol will be provided. This objective is related to the feasibility of the proposed pro tocol.

Objective 4: to create Indicative Mobile Peer-to-Peer Applications based on UbiChord. Taking under consideration the four main objectives, the contribution of this Thesis is described in the upcoming subsection.

1.3 Contribution of this Thesis

Taking under consideration the four main objectives of the Thesis we could highlight the assets that derive from the conducted research. In general there are three major assets:

Asset 1: The first asset is the formulation of a Distributed Hash Table Protocol capable of operating on top of Mesh Networks. Specific protocol mechanisms will be introduced to guarantee reliable operation of the structure, under the uncertainty of Mesh networking conditions. Various mechanisms that comprise the protocol are analytically described and evaluated. Furthermore, the protocol follows a layered approach, which allows straightforward protocol implementation in diverse platforms.

Asset 2: The second asset is a concrete reference implementation of the aforementioned protocol which is addressed as "UbiChord". In order for the protocol to be realized (or instantiated) specific technical choices have been followed. Moreover, during the evaluation of the implemented protocol, additional mechanisms have been created in order to tackle specific bottlenecks that were identified. These bottlenecks led to the creation of additional mechanisms that "bend" the non-linear performance of some layers (mainly routing layer). These mechanisms were grouped as a separate protocol addressed as "NEURON". However, NEURON mechanisms have not been integrated in the implemented protocol.

Asset 3: The third asset is related to the provision of a novel application-provision paradigm. More specifically, the existence of a DHT capable for storing and retrieving information (i.e. key-value sets) can introduce a new programming model regarding service provisioning. Up to now, collaborative applications in Mesh or PAN networks follow a centralized approach, in the sense that either all nodes that run the application are assumed to have connectivity with the central server or one node of the mesh network "acts" as a server. The utilization of UbiChord can alter this programming model by introducing native decentralization. Applications can share (i.e. store and retrieve) volatile data in UbiChord without caring who is responsible for the physical storage of data. This capability paves the way for the creation of a new set of applications that are in line with autonomic principles.

1.4 Scientific Publications & Contribution to Standards

The Thesis is the outcome of targeted research conducted in the domain of service provisioning in Mesh Environments. Research was targeting both theoretical and practical aspects of the domain. It is of major importance to emphasize on the continuity of the research in order to comprehend the maturity of the final results. First of all, the technology trigger at 2006 - 2007 (see Figure 1.5) was Near Field Communications technology (a.k.a. NFC) according to which, each internet-capable device could participate in a service-ecosystem. The open research issues included, among others, the formulation of semantic service models for sensors and actuators. In this specific field, the following publication [P. Gouvas et al., 2007]: "***Panagiotis Gouvas**, Thanassis Bouras, Gregoris Mentzas: An OSGi-Based Semantic Service-Oriented Device Architecture. OTM Workshops (2) 2007: 773-782*" contributed on establishing a meta-model for service-grounding in Service Oriented Device Architectures (a.k.a. SODA).

However, the vision of numerous inter-connected devices which collaborate, soon led to the emergence of the "broader" notion of Internet of Things (or equivalently the Internet of Connected Objects). This vision incorporates many challenges. Some of these challenges include creation of applications that operate on top of dynamic topologies, under uncertain and unreliable conditions with increased complexity. Under this perspective, the following publication [P. Gouvas et al. - 1, 2010]: "***Panagiotis Gouvas**, Anastasios Zafeiropoulos, Athanassios Liakopoulos, Gregoris Mentzas and Nikolas Mitrou: Integrating Overlay Protocols for Providing Autonomic Services in Mobile Ad-hoc Networks 2010 IEICE Transactions on Communications"* proposed an innovative way for creating services in Mesh Environments. The proposed approach relied on the adaptation of DHT structures that are a-priori non suitable for operating in such dynamic environments. A theoretical and practical description of the approach was provided. This publication is the foundational cornerstone of UbiChord.

Since a DHT structure could be the center of gravity as far as service creation in Mesh environments is concerned, UbiChord had to be exhaustively tested given that such environments are extremely challenging. Therefore, the aspect of scaling was thoroughly investigated. Specific bottlenecks that have been identified, especially in routing layer resulted in the creation of further adaptations. According to these adaptations, clustering and gossiping techniques can be used to minimize routing overhead. Indicative services have been created in sensor mesh environments. The outcome of this research was published at [A. Zafeiropoulos et al., 2010]: "*Anastasios Zafeiropoulos, **Panagiotis Gouvas**, Athanassios Liakopoulos, Gregoris Mentzas and Nikolas Mitrou, NEURON: Enabling Autonomicity in Wireless Sensor Networks, 2010, Sensors (ISSN 1424-8220)*".

The mechanisms of NEURON resulted in radical reduction of routing and signaling overhead required by the DHT structure in order to be fully operational. As a result, further

investigation of some mechanisms was conducted. Special emphasis was given in bio-inspired mechanisms such as gossiping. The goal of this research was to identify the implication of dynamicity in bio-inspired techniques. Conclusively, the following publication was accomplished [P. Gouvas et al. -2, 2010]: "***Panagiotis Gouvas**, Anastasios Zafeiropoulos, Athanassios Liakopoulos: Gossiping for Autonomic Estimation of Network-Based Parameters in Dynamic Environments. OTM Workshops 2010: 358-366*"

Furthermore, UbiChord can be used for the creation of several autonomic applications. Such applications include autonomic network monitoring and management which was discussed in the frames of the publication [A. Liakopoulos et al., 2010]: "*Athanassios Liakopoulos, Anastasios Zafeiropoulos, Constantinos Marinos, Mary Grammatikou, Nikolay Tcholtchev and **Panagiotis Gouvas**, Applying distributed monitoring techniques in autonomic networks, IEEE GLOBECOM 2010, Florida, USA*".

Finally, it could be argued that the need derived by the next generation of services is distributed reasoning. Distributed reasoning implies that inter-connected nodes (i.e. sensors or actuators) generate or consume information while interacting with the environment. UbiChord could be medium for information fusion which is the first step of distributed reasoning. Usage of domain dependent context models would optimize the reasoning procedure. Such a context model, on top of UbiChord has been evaluated in the frames of the publication [A. Zafeiropoulos et al., 2011]: "*Anastasios Zafeiropoulos, **Panagiotis Gouvas**, Athanassios Liakopoulos, A Context Model for Autonomic Management of Ad-hoc Networks, International Conference on Pervasive and Embedded Computing and Communication Systems, 2011, Portugal*" targeting the domain of network monitoring.

Regarding standardization efforts, concrete research results of this Thesis have been diffused in a Draft of European Telecommunications Standardization Institute (ETSI), in the working group of "Autonomic network engineering for the self-managing Future Internet" (a.k.a. AFI). More specifically, Draft "AFI 001: Scenarios, Use Cases, and Requirements for Autonomic/Self-Managing Future Internet" contains requirements for autonomic applications which are in-line with the requirements and the architectural principles of UbiChord. For ETSI members, the standardisation draft is also available here: http://portal.etsi.org/portal/server.pt/community/AFI/344.

A complete list of publications is provided as an annex at the end of the Thesis.

1.5 Structure of this Thesis

The Thesis is structured as follows:

"*Chapter 2 - Problem Statement*" introduces the reader to the key concepts of autonomic principles. Furthermore it explains why a Distributed Hash Table can be used as cornerstone for the realization of autonomic applications. DHTs have many reference implementations. However, DHT mechanisms can be abstracted at a certain level. Ignoring the low level implementation details of the diverse reference implementations, the abstracted DHT principles will be described. These principles are fundamental in order to understand the difficulties for porting an existing DHT implementation in a Mesh Network. DHT principles are provided at Chapter 2.1 while the aforementioned difficulties are thoroughly discussed at Chapter 2.2. Chapter 2.2 covers Objective-1.

"*Chapter 3 - State of The Art Analysis*" intends to familiarize the reader with existing DHT implementations in fixed networks (Chapter 3.1) and existing approaches/research initiatives that attempted to port Peer-to-Peer protocols in Mesh Environments (Vhapter 3.2). In parallel, limitations and correlation with UbiChord are discussed. Finally, one of the most critical aspects of Mesh Networks is routing since the absence of predefined routing scheme is a key problem as shown at Chapter 2.2. This fact leads to the presentation of a brief State of the Art (SoTA) analysis regarding Mesh routing (Chapter 3.3).

"Chapter 4 - Proposed Layered Approach" aims to analyze the UbiChord architecture that overcomes difficulties as identified at Chapter 2.2. The proposed approach is a layered approach and therefore Chapters 4.1 and 4.2 explain why this layered approach has been followed. Chapter 4.3 describes in-detail the mechanisms in each layer. These mechanisms have been implemented at a Simulator (PeerSim[11] Simulator). The results of simulations are described on Chapter 6.1. Based on these results, the four-layered approach has been enhanced in order to be more scalable. Specific adaptations have been introduced to address scalability. These adaptations are described at Chapter 4.4 and evaluated on Chapter 6.1.3. Finally, Chapter 4.5 provides a detailed analysis on how autonomic services can be created on-top of UbiChord. The entire Chapter 4 covers Objective-2.

"Chapter 5 - Prototype Implementation details" describes the prototype implementation of the layered approach that is described on Chapter 4. The layered approach is an architecture that can be realized, technically, in diverse ways. A prototype implementation of the architecture is described in the specific chapter. Chapter 5.1 analyses the technical choices that have been followed in order to realize UbiChord and provides and detailed analysis of UbiChord's internal architecture. Chapter 5.2 provides a bird's eye view of the developed Custom Emulation environment that was developed in the frames of this Thesis in order to assess the various layers of UbiChord. The necessity

[11] PeerSim Simulator - http://peersim.sourceforge.net/

for this environment and the environment itself are covered in this sub-chapter. Chapter 5 covers Objective-3 and partially Objective-4.

"Chapter 6 - Simulation & Emulation Results" has a threefold goal. First it provides the simulator results for the Proposed Layered Approach as described on Chapter 4 and an indicative service built on top of the layered approach (Chapters 6.1 and 6.2). Then it provides simulator results for the extensions that have been built in order to overcome scaling issues (Chapter 6.3). Finally, it provides the emulation results of the reference implementation as described on Chapter 5. Chapter 7 covers partially Objective-4.

"Chapter 7 - Conclusions and further research" contains an initial chapter which concludes on how UbiChord overcomes difficulties that were posed at the problem statement (Chapter 7.1). Then, an analysis on current limitations that have to be confronted, and ideas of possible improvements are provided on Chapter 7.2 along with description of the future steps of UbiChord.

Finally, "*Chapter 8 - References"* contains the references for the entire Thesis.

2. Problem Statement

2.1 Incorporation of Autonomic Principles in Mesh networking applications

As already stated in the introduction the trend regarding networks' evolution includes the incorporation of characteristics such as dynamic topologies, zero-configuration, mesh environments etc. In parallel, a plethora of devices is emerging supporting diverse networking capabilities. The combination of these trends leads to the realization of the so-called Internet of Things (IoT).

While there is no formal definition of the term Internet of Things, it could be argued that expresses the ability of everyday ordinary objects to connect to a network. When the ability of physical connection is augmented to ability of inter-operation among the participating objects then we refer to Internet of Connected Objects (ICO). There are diverse types of Objects which can be classified based on many characteristics e.g. objects can be characterized as sensors or actuators depending on the characteristic of information extraction/reaction or hardware/software depending on the physical nature of the object etc.

The instantiation/rollout of Internet Connected Objects paradigms follows a diverse scheme. Indicatively, Hewlett Packard launched at May 2010 CENSE[12] where millions of sensors will measure data such as vibration, rotation, sound, air flow, light, temperature, pressure and much more all around the globe. Furthermore, IBM's Smarter Planet campaign is about connecting objects to the Internet and applying intelligence and services on top of that. In January 2010, IBM shed light on Big Blue's sensor platform highlighting that IBM had developed 1,200 "smarter solutions" up till that time pinpointing that IBM has the ability to provide sensor systems to support city infrastructures[13].

Although the benefits for the applications that are based on IoT technology are a lot, there are problems that can be summarized in the estimation of HP's Peter Hartwell: "one trillion nanoscale sensors and actuators will need the equivalent of 1000 Internet Infrastructures (as deployed today): the next huge demand for computing!".

Consequently, ICO per se is not affected by the limitations and pathogens of existing Internet deployment but on the other hand it constitutes the basic reason why existing technology should change.

2.1.1 Problems regarding IoT applications

Moving one step further it is valuable to examine the nature of ICO deployment and applications and discuss on the competitive forces on the domain that formulate center of

[12] http://www.hpl.hp.com/research/intelligent_infrastructure/
[13] http://www.readwriteweb.com/archives/top_10_internet_of_things_developments_of_2010.php

gravities. First of all, the first set of competitive forces is complexity of operation and the requirement of a simplified administration (complexity VS simplicity).

Additionally, the decentralized manner of deployment is opposing to the need for centralized control (decentralization VS centralization). Each object is able to generate and consume local information (e.g. sensor data) but a context-aware incident can only be inferred by the combination of available information (information fusion VS distributed reasoning). Moreover, medium level communication between Objects is considered as granted, but the mobile nature of nodes prevents the operation of scalable static routing schemes (reactive routing VS scalability). Finally, the functionality exposed by Connected Objects is context-dependent while a developer would need a sound and simplified API in order to create an application utilising Connected Objects (expose Connected Object's functional profile VS transparency to application developer).

2.1.2 Autonomicity as a solution

Most of the aforementioned problems are already successfully tackled in existing biological systems, e.g. human body uses a hierarchically structured nervous system to coordinate a vast amount of sensors and actuators of diverse nature (i.e. different types of cells). Although each "sensor" and "actuator" seam to act independently (decentralized), they manage to achieve one (centralized) goal; "guarantee equilibrium between the body and the environment". Although there is no detailed view of the envisaged algorithms, there are many principles that can be ported from the domain of biology to the domain of Internet Connected Objects. Among all concepts, the most crucial one is the concept of autonomic systems.

Autonomic system is the part of the nervous system which controls the tissues, organs and systems without conscious thought. Lack of conscious and centralized control urged computer scientists to port autonomic principles in the networking domain to resolve growing complexity of network management. This philosophy is in-line with the emerging trend of bio-inspired computing according to which algorithms and principles applied at nature can provide elegant solutions to existing problems in computer science. As a result, Autonomic Computing Initiative introduced four principles (a.k.a. self-* principles) that any computational autonomic system should take under consideration:

- *Self-Configuration*: Automatic configuration of elements
- *Self-Healing*: Automatic discovery, and correction of faults
- *Self-Optimization*: Automatic monitoring and control of resources to ensure the optimal functioning with respect to the defined requirements
- *Self-Protection*: Proactive identification and protection from arbitrary attacks

The cornerstone concept applied in Autonomic Systems is the closed control loop. Such a loop in a self-managing system monitors some resource/variable of a software or

hardware component and autonomously tries to keep its parameters within a desired range. The theory behind the control loops is derived by the Process Control. According to IBM, hundreds or even thousands of these control loops are expected to work in a large-scale self-managing computer system.

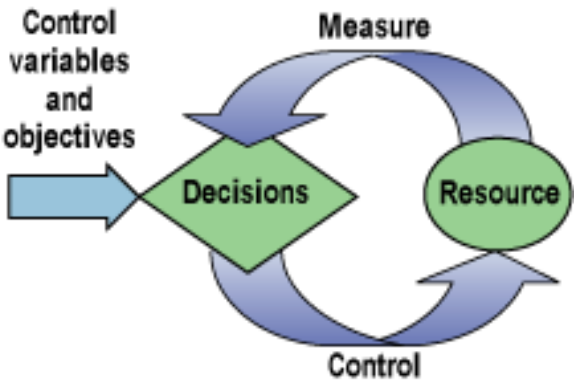


Figure 2.1 - Autonomic loop

The maintenance of an autonomic loop in a networking environment pre-assumes the ability to read and write variable values/states. Based on these values proactive or reactive decisions can be made. However, reading and writing in a shared data-space is not trivial since the usage of a centralized “database” is out of the question. A centralized solution would have a single point of failure and would impose extreme configuration cost in dynamic environments.

The most elegant solution for the realization of this shared ”data-space” is a Distributed Hash Table Structure which has zero-configuration cost and no point of failure. Although DHTs have significant properties regarding putting and getting key-value pairs, their principles of operation rely on network stability and reliability. Consequently, DHTs cannot be used, per se, in Mesh Environments. In the frames of this Thesis we will introduce mechanisms accompanied by theoretical analysis that overcome difficulties which are raised by Mesh environments.

2.2 DHTs at a glance

As already mentioned above a Distributed Hash Table is a class of decentralized distributed system that provides a lookup service similar to a hash table; (key, value) pairs are stored in the DHT, and any participating node can efficiently retrieve the value associated with a given key. Responsibility for maintaining the mapping from keys to values is distributed among the nodes, in such a way that a change in the set of participants causes a minimal amount of disruption. This allows DHTs to scale to extremely large numbers of nodes and to handle continual node arrivals, departures, and failures.

DHTs form an infrastructure that can be used to build more complex services, such as distributed file systems, peer-to-peer file sharing and content distribution systems, cooperative web caching, multicast, anycast, domain name services, and instant messaging, social applications etc. Notable distributed networks that use DHTs include BitTorrent's distributed tracker[14], the Kad network [M. Steiner et al, 2007], YaCy[15], and the Coral Content Distribution Network [M. J. Freedman et al., 2004].

2.2.1 History

DHT research was originally motivated, in part, by peer-to-peer systems such as Napster[16], Gnutella[17], and Freenet[18], which took advantage of resources distributed across the Internet to provide a single useful application. In particular, they took advantage of increased bandwidth and hard disk capacity to provide a file sharing service. These systems differed in how they found the data their peers contained:

Napster had a central index server: each node, upon joining, would send a list of locally held files to the server, which would perform searches and refer the "querier" to the nodes that held the results. This central component left the system vulnerable to attacks and lawsuits.

Gnutella and similar networks moved to a flooding query model—in essence, each search would result in a message being broadcasted to every other machine in the network. While avoiding a single point of failure, this method was significantly less efficient than Napster. Moreover, Freenet was also fully distributed, but employed a heuristic key-based routing in which each file was associated with a key, and files with similar keys tended to cluster on a similar set of nodes. Queries were likely to be routed through the network to such a cluster without needing to visit many peers. However, Freenet did not guarantee that data would be found.

[14] http://bitconjurer.org/BitTorrent
[15] http://yacy.net/Technology.html
[16] http://www.napster.com
[17] Gnutella Protocol Specification http://wiki.limewire.org/index.php?title=GDF
[18] http://freenetproject.org/

On the other hand, Distributed Hash Tables use a more structured key-based routing in order to attain both the decentralization of Gnutella and Freenet, and the efficiency and guaranteed results of Napster. One drawback is that like Freenet, DHTs only directly support exact-match search, rather than keyword search, although such functionality can be layered on top of a DHT.

From 2001 to 2004, six systems - CAN [S. Ratnasamy et al., 2001], Chord [I. Stoica et al., 2003], Pastry [A. Rowstron et al., 2001], Tapestry [B. Y. Zhao, 2004], Kademlia [P. Maymounkov et al., 2002] and Viceroy [D. Malkhi et al., 2002] - ignited DHTs as a popular research topic, and this area of research remains active. Outside academia, DHT technology has been adopted as a component of BitTorrent and in the Coral Content Distribution Network.

2.2.2 DHT Properties

DHTs characteristically emphasize the following properties:

- *Decentralization*: the nodes collectively form the system without any central coordination.
- *Scalability*: the system should function efficiently even with thousands or millions of nodes.
- *Fault tolerance*: the system should be reliable (in some sense) even with nodes continuously joining, leaving, and failing.

A key technique used to achieve these goals is that any node needs to coordinate with only a few other nodes in the system – most commonly, O(log n) of the n participants – so that only a limited amount of work needs to be done for each change in membership.

Some DHT designs seek to be secure against malicious participants [G. Urdaneta et al., 2011] and to allow participants to remain anonymous, though this is less common than in many other peer-to-peer (especially file sharing) systems; see anonymous P2P. Finally, DHTs also deal with more traditional distributed systems issues such as load balancing, data integrity, and performance (in particular, ensuring that operations such as routing and data storage or retrieval complete quickly).

2.2.3 DHT Principles

The structure of a DHT can be decomposed into several main components [Gurmeet, 2004]. The foundation is an *abstract keyspace*. A *keyspace partitioning* scheme splits ownership of this keyspace among the participating nodes. An overlay network then, connects the nodes, allowing them to find the owner of any given key in the keyspace.

Once these components are in place, a typical use of the DHT for storage and retrieval might proceed as follows. Suppose the keyspace is the set of 160-bit strings. To store a file with given *filename* and *data* in the DHT, the *SHA-1* [ANSI, 1997] hash of filename is generated, producing a 160-bit *key k*, and a message *put(k,data)* is sent to any node

participating in the DHT. The message is forwarded from node to node through the overlay network until it reaches the single node responsible for *key k* as specified by the *keyspace partitioning*. The appropriate node stores the key and the data. Any other client can retrieve the contents of the file by again hashing filename to produce *k* and asking any DHT node to find the data associated with *k* with a message *get(k)*. The message will again be routed through the overlay to the node responsible for *k*, which will reply with the stored data.

These principles are depicted at Figure 2.2 where the inner circle represents the physical topology of the mobile nodes while the outer circle represents the DHT overlay. The general idea is that every node that is registered to the DHT is able to publish and retrieve data. Please note that the same Hash function that is used for node registration in the overlay is used for data registration. This is very crucial since it is related to the keyspace partitioning.

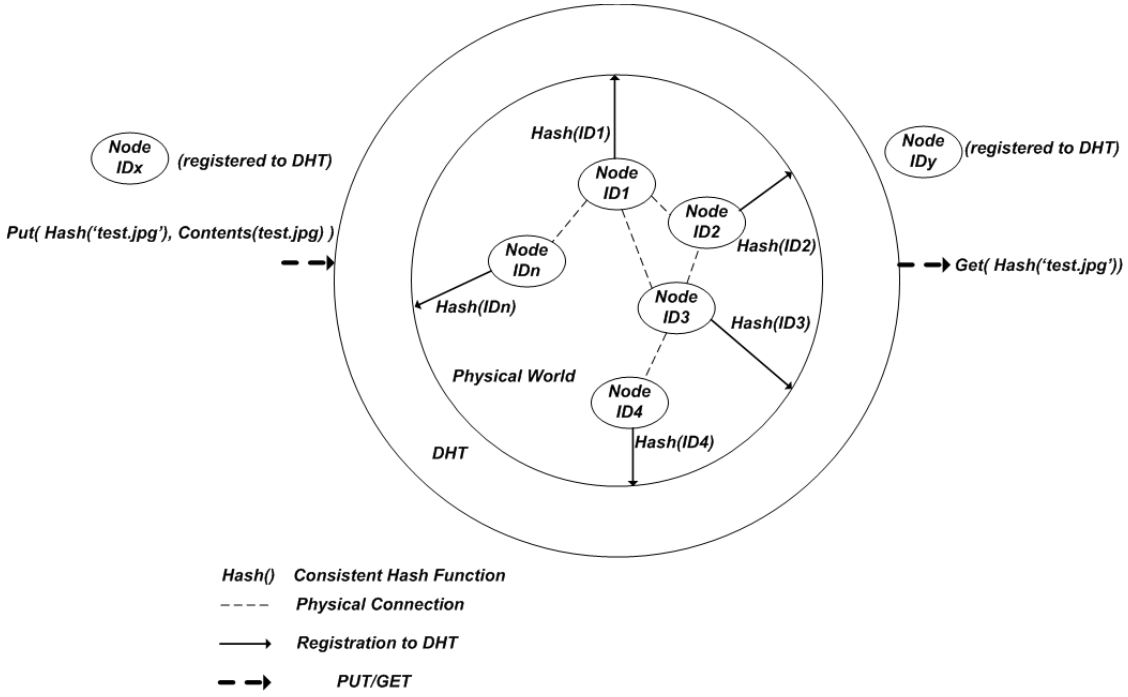


Figure 2.2 - DHT overview

The *keyspace partitioning* and *overlay network* components are described below with the goal of capturing the principal ideas common to most DHTs; many designs differ in the details.

2.2.3.1 Keyspace partitioning

Most DHTs use some variant of *consistent hashings* [D. Karger et al., 1997] to map keys to nodes. This technique employs a function $\delta(k_1,k_2)$ which defines an abstract notion of the

distance from key k_1 to key k_2, which is unrelated to geographical distance or network latency. Each node is assigned a single key called its identifier (***ID***). A node with ID i_x *owns all the keys* k_m for which i_x is the closest ***ID***, measured according to $\delta(k_m,i_n)$.

In order to make keyspace partitioning clearer, an example from a real DHT implementation will be provided. The Chord DHT treats keys **as points on a circle**, and $\delta(k_1,k_2)$ is the distance traveling clockwise around the circle from k_1 to k_2. Thus, the circular keyspace is split into contiguous segments whose endpoints are the node identifiers. If i_1 and i_2 are two adjacent ***ID***s, then the node with ID i_2 owns all the keys that fall between i_1 and i_2. This can be depicted at Figure 2.3 where a Chord DHT is bootstrapped. The DHT is configured to have replication factor equals to two. This practically means that every key-value pair that is assigned to the node-responsible is automatically assigned to the next two successors in the overlay. So, if a key-value pair with key:K is stored (e.g. by node-F) to the DHT and A<K<B then the authoritative physical node that must store this pair is the one that has Hash(NodeID)=B. Because of the replication factor, nodes B, C and D store keys in the range of A-B.

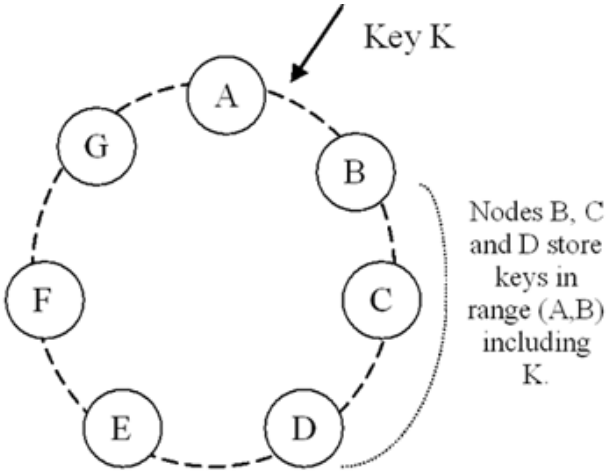


Figure 2.3 - Chord's keyspace partitioning

Consistent hashing is based on mapping items to a real angle (or equivalently a point on the edge of a circle). Each of the available machines (or other storage buckets) is also pseudo-randomly mapped on to a series of angles around the circle. The bucket where each item should be stored is then chosen by selecting the next highest angle which an available bucket maps to. The result is that each bucket contains the resources mapping to an angle between itself and the next smallest angle.

If a bucket becomes unavailable (e.g. because the computer it resides on is not reachable), then, the angles it maps to will be removed. Requests for resources that would

have mapped to each of those points now map to the next highest point. Since each bucket is associated with many pseudo-randomly distributed points the resources that were held by that bucket will now map to many different buckets. The items that mapped to the lost bucket must be redistributed among the remaining ones, but values mapping to other buckets will still do so and do not need to be moved.

A similar process occurs when a bucket is added. By adding an angle, we make any resources between that and the next smallest angle map to the new bucket. These resources will no longer be associated with the previous bucket, and any value previously stored there will not be found by the selection method described above. The portion of the keys associated with each bucket can be altered by altering the number of angles that bucket maps to.

Consistent hashing has the essential property of minimal disturbance of the network during removal or addition of nodes since topology-changes affect only the set of keys owned by the nodes with adjacent IDs, and leaves all other nodes unaffected. On the other hand, in traditional hash tables addition or removal of one bucket causes nearly the remapping of the entire keyspace. Since any change in ownership typically corresponds to bandwidth-intensive movement of objects stored in the DHT from one node to another, minimizing such reorganization is required to efficiently support high rates of churn (node arrival and failure). The most common consistent Hashing function is SHA-1.

2.2.3.2 Overlay Network

Each node maintains a set of links to other nodes (its neighbors or routing table) and together these links form the overlay network. A node picks its neighbors according to a certain structure, called the network's topology.

All DHT topologies share some variant of the most essential property: for any *key* ***k***, each node either has a node ***ID*** which owns ***k*** or has a link to a node whose node ***ID*** is closer to ***k***, in terms of the keyspace distance defined above. It is then easy to route a message to the owner of any key ***k*** using the following greedy algorithm (that is not necessarily globally optimal): at each step, forward the message to the neighbor whose ***ID*** is closest to ***k***. When there is no such neighbor, then we must have arrived at the closest node, which is the owner of ***k*** as defined above. This style of routing is sometimes called key-based routing.

Beyond basic routing correctness, two important constraints on the topology are to guarantee that the maximum number of hops in any route (route length) is low, so that requests complete quickly; and that the maximum number of neighbors of any node (maximum node degree) is low, so that maintenance overhead is not excessive. Of course, having shorter routes requires higher maximum degree. Some common choices for

maximum degree and route length are as follows, where ***n*** is the number of nodes in the DHT, using Big O notation:

- Degree O(1), route length O(*n*)
- Degree O(log*n*), route length O(log*n* / loglog*n*)
- Degree O(log*n*), route length O(log*n*)
- Degree O($\sqrt{n}$), route length O(1)

The third choice is the most common even though it is not quite optimal in terms of degree/route length tradeoff, because such topologies typically allow more flexibility in choice of neighbors. Many DHTs use that flexibility to pick neighbors which are close in terms of latency in the physical underlying network.

Maximum route length is closely related to diameter: the maximum number of hops in any shortest path between nodes. Clearly the network's route length is at least as large as its diameter, so DHTs are limited by the degree/diameter trade off which is fundamental in graph theory. Route length can be greater than diameter since the greedy routing algorithm may not find shortest paths [Gurmeet et al., 2004].

2.2.3.3 Variations of diverse Implementations

The most notable differences encountered in practical instances of DHT implementations are discussed below. First of all, several real world DHTs use 128 bit or 160 bit keyspace. Furthermore, some real-world DHTs use hash functions other than SHA1. Additionally, in the real world the key ***k*** could be a hash of a file's content rather than a hash of a file's name, so that renaming of the file does not prevent users from finding it.

Moreover, some DHTs may also publish objects of different types. For example, key ***k*** could be node ***ID*** and associated data could describe how to contact this node. This flexibility allows publication of presence information and is often used in Instant Messaging applications, etc. In simplest case ***ID*** is just a random number which is directly used as key ***k*** (so in a 160-bit DHT ***ID*** will be a 160 bit number, usually randomly chosen). In some DHTs publishing of nodes ***ID***s is also used to optimize DHT operations.

Redundancy can be added to improve reliability. The ***(k,data)*** key pair can be stored in more than one node corresponding to the key. Usually, rather than selecting just one node, real world DHT algorithms select ***i*** suitable nodes, with ***i*** being an implementation-specific parameter of the DHT. In some DHT designs, nodes agree to handle a certain keyspace range, the size of which may be chosen dynamically, rather than hard-coded.

Some advanced DHTs like Kademlia [P. Maymounkov et al., 2002] perform iterative lookups through the DHT first, in order to select a set of suitable nodes and send put(***k***,***data***) messages only to those nodes, thus drastically reducing useless traffic, since published messages are only sent to nodes which seem suitable for storing the key ***k***; and iterative lookups cover just a small set of nodes rather than the entire DHT, reducing

useless forwarding. In such DHTs forwarding of *put*(***k***,***data***) messages may only occur as part of a self-healing algorithm: if a target node receives a *put*(***k***,***data***) message but believes that k is out of its handled range and a closer node (in terms of DHT keys pace) is known, the message is forwarded to that node. Otherwise, data are indexed locally. This leads to a somewhat self-balancing DHT behavior. Of course, such an algorithm requires nodes to publish their presence data in the DHT so the iterative lookups can be performed.

2.3 Difficulties in maintaining a distributed key-value store in Mesh Environments

In Chapter 2.1 the principles of DHTs and the merits that are derived from them were presented. However, DHTs are designed to operate on the Internet environment and not on a Mesh environment. More specifically, the assumptions that are made by the majority of DHT implementations which are met by the Internet environment are the following:

Efficient underlay routing & efficient connection establishment: e.g. assume that a *newnode* enters the overlay in a Chord DHT implementation between *node1* and *node2* (we remind the reader that Chords uses a circular overlay topology). Since new segments have been formulated i.e. *node1-newnode*&*newnode-node2*, specific key-value entries have to be transferred from *node2* to *newnode*. This transfer has to be done efficiently without a big communication start-up cost since a possible query for a specific key *k* in the structure may end-up at *node1*. If *newnode* is the correspondent node according to the consistent hashing function, *node1* will consult its finger table and will propagate the query to *newnode.* Consequently, *newnode* should be ready to respond to the query performer as quick as possiblein order to prevent blocking issues.

Long lasting connections & stationary peers: assume that in the example described above, *newnode* enters and leaves the topology instantly. This would not be catastrophic for the DHT structure's coherency (since mechanisms that handle timeouts during key-value transfers exist) but it would generate a signaling cost both in the network and the overlay layer. This cost may be insignificant in fixed networks since a predefined routing scheme exists (see below) but in a Mesh Environment it would be too expensive.

Hierarchical routing scheme: assume that in the example described above, *node2* is notified that specific *key-value* pairs must be transferred to *newnode*. However, *node2* is not aware on how *new node* will be reached; *but it does not care also*, since Internet's hierarchical structure implies that the transferable key-value entry(ies) will be routed to *node2*'s default gateway and through TCP/IP entries will reach their destination.

Dedicated peers: as explained above, the task of overlay topology construction and maintenance is undertaken by low level mechanisms which in most of the cases are *centralized or semi-centralized*. Indicatively, such mechanisms are used in the Gnutella network [Y. Wang et al., 2007], in which topology creation may be achieved by using a pre-defined address-list of working nodes included within a compliant client or by using web caches of known nodes, a.k.a. Gnutella web caches. Similarly, Chord pre-assumes that nodes are ordered in a ring and are aware of their successor and predecessor in the overlay ring topology. Chord also relies on underlying mechanisms for the overlay network bootstrapping [I. Stoica et al., 2001]. In conclusion, p2p protocols are able to react to topology changes (and automatically re-assign key-value pairs) but are not responsible for

creating and maintaining the overlay topology. This indicative issue, among others, is delegated to some super-peers that are also accessible due to the hierarchical routing scheme (see bullet -c-).

Stable network: refers to the backbone network and not to the endpoints that constitute the DHT peers. A stable backbone network reassures the efficiency of the hierarchical routing scheme and prevents island formulation. In order to clarify this issue, assume that in the example of bullet -a- *newnode* is connected to *node2* trough a unique valid route e.g. *newnode-nodex-nodey-node2* and the connection between *nodex* and *nodey* during key-value pair transfer is lost. This would result in the formulation of two islands consisting of i) island 1: *newnode*, *nodex* and all their local neighbors and ii) island 2: *nodey*, *node2* and all their local neighbors.

It is obvious that none of the aforementioned assumptions can be considered as granted in a Mesh environment. Nodes are not stationary and links are considered unreliable (see Figure 2.4). Moreover, there is no form of centralization and no dedicated peer in order to coordinate overlay construction. Consequently, the construction and maintenance of the overlay must be accomplished in an ad-hoc mode.

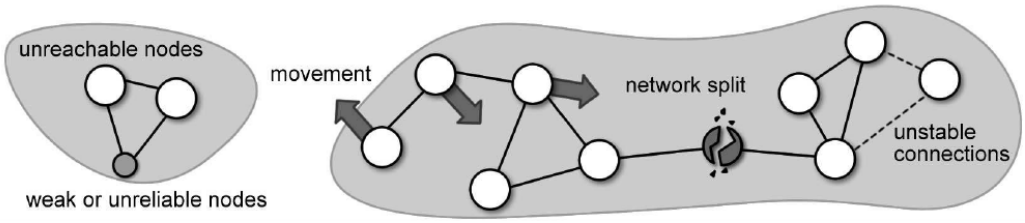


Figure 2.4 - Mesh Networking Environment

Furthermore, low level routing among nodes is an important issue since no predefined routing scheme can be taken for granted. Therefore, alternative routing policies have to be adopted. Such routing protocols will be described at Chapter 3.3.

Finally, the most critical problem is the lack of topological hierarchy which results in loosely temporarily created graphs. Such graphs (a.k.a. islands) of interconnected nodes may merge and split according to the current topology. This affects significantly the DHT operation. E.g. assume that in two existing separate islands, as depicted at Figure 2.5, the Chord protocol has bootstrapped. The mechanisms that have been used by Chord to bootstrap are out of the scope of the example. According to Chord all participant nodes are directed to circular overlay topology for both islands. Assume that one node from one island committed *put(key_1,$value_x$)* and another node from the other island committed in his circle/DHT *put(key_1,$value_y$)*. In the next step, the nodes that comprise the two islands come closer and formulate one big island (see Figure 2.6).

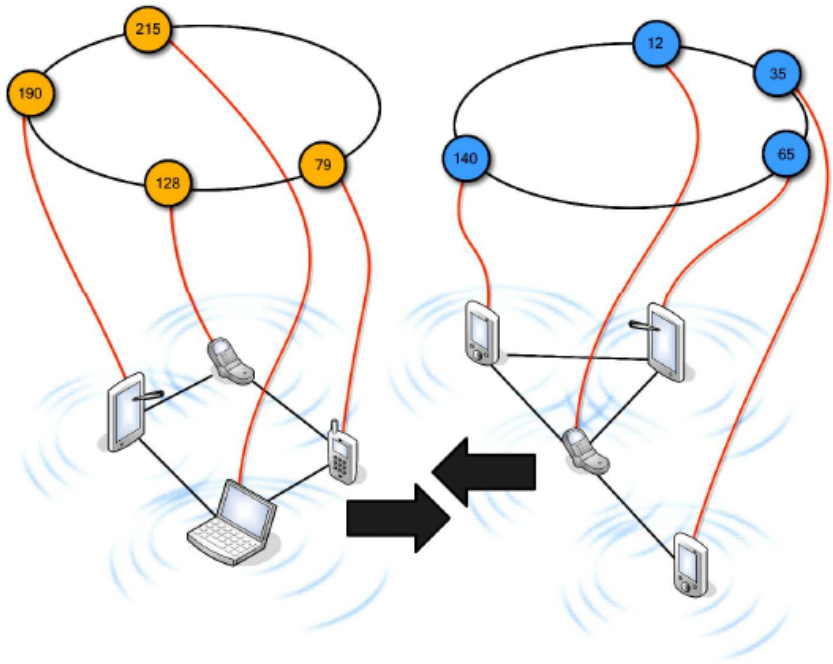


Figure 2.5 - Two islands before they merge

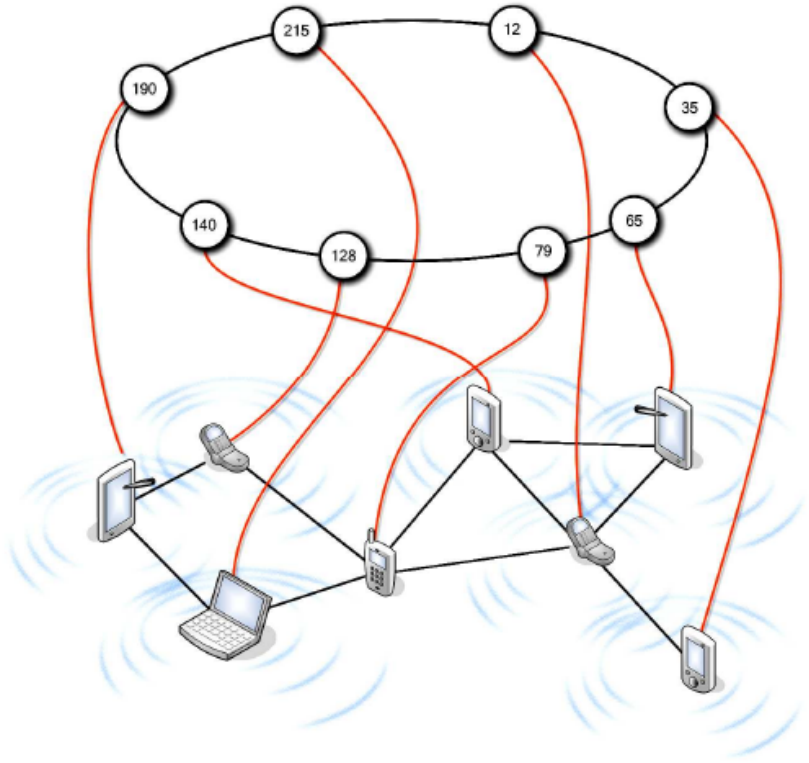


Figure 2.6 - Two islands already merged

The goal that has to be achieved after the merging is the efficient identification of the responsible node that maintains the value of key_1 after the query of a third-party node.

Normally, according to the DHT consistency principle, values should be merged and a possible *get(key_1)* should result in the multi-value response *$value_x$-$value_y$*. However, a successful response pre-assumes that DHT signaling has been completed. On the other hand signaling is of minor importance for DHTs that are bootstrapped on fixed networks but it is considered too expensive for Mesh Environments.

Moreover, node-splitting is a scenario where DHT consistency is not guaranteed. E.g. assume that the big island of Figure 2.6 splits in two sub-islands of Figure 2.5. Suppose that a specific key_1-$value_x$ pair is published, the DHT protocol is Chord and the *HashOf(key_1)=130*. According to Chord algorithm the responsible for storing this pair is the successor node of 130 i.e. node 140 in our case. It must be clarified here, that node 140 actually means node with *HashOf(NodeID)=140* where NodeID is something common among the nodes e.g. their MAC address. According to the splitting scenario, the physical topology is split and node 140 belongs to the "blue" island. Now the question is what will the result be when node "79" from the orange islands commits *get(key_1)*? The answer is *null* since in the "orange" island the authoritative node for providing the response is node 190 (i.e. the successor of 130). Node 190 has no info about key $_1$.

Similarly, assume that during *put(key_1,$value_x$)* (and before splitting up) there was a redundancy policy and the key-value pair was stored to its successor and to the next node (one node for redundancy). After the split-up, when node "79" from the orange islands commits *get(key_1),* the result would be $value_x$. So, redundancy is the key parameter as far as network splitting is concerned. Consequently, merging and splitting, in general, results in significant signaling cost on Mesh environments.

3. State Of The Art Analysis

3.1 Overview of Structured and Unstructured P2P Networks in fixed Networks

Peer-to-peer overlay networks are distributed systems in nature, without any hierarchical organization or centralized control. Peers form self-organizing overlay networks that are bootstrapped and operated on top of IP-networks, offering a mix of various features such as robust wide-area routing architecture, efficient search of data items, selection of nearby peers, redundant storage, permanence, hierarchical naming, trust and authentication, anonymity, massive scalability and fault tolerance.

Peer-to-peer overlay systems go beyond services offered by client-server systems by having symmetry in roles where a client may also be a server. They allow access to resources of participating systems and support resource-sharing, which requires fault-tolerance, self-organization and massive scalability properties.

We can view P2P overlay network models spanning a wide spectrum of the communication framework, which specifies a fully-distributed, cooperative network design with peers building a self-organizing system. Figure 3.1 shows an abstract P2P overlay hierarchy, illustrating the components in the overlay communications framework. The Network Communications layer describes the network characteristics of nodes connected over the Internet or small wireless or sensor-based devices that are connected in an ad-hoc manner. The dynamic nature of peers poses challenges in communication paradigm.

The Overlay Nodes Management layer covers the management of peers, which include discovery of peers and routing algorithms for optimization. The Features Management layer deals with the security, reliability, fault resiliency and aggregated resource availability aspects of maintaining the robustness of P2P systems. The Services Specific layer supports the underlying P2P infrastructure and the application-specific components through scheduling of parallel and computation-intensive tasks, content and file management. Meta-data describes the content stored across the P2P peers and the location information. The Application-level layer is concerned with tools, applications and services that are implemented with specific functionalities on top of the underlying P2P overlay infrastructure. There are two classes of P2P overlay networks: Structured and Unstructured.

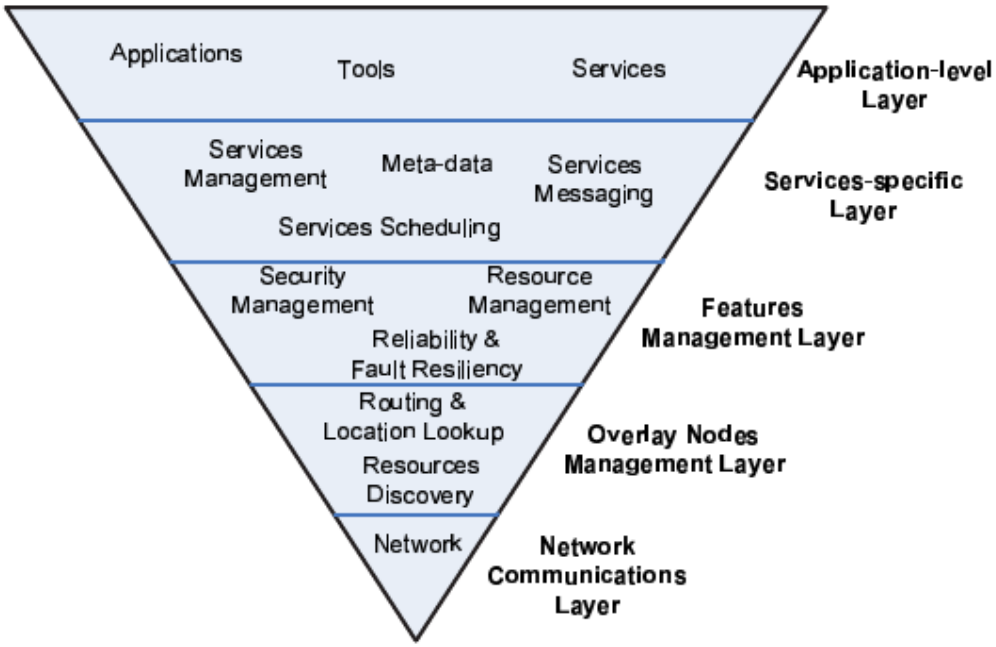


Figure 3.1 - Abstract Peer-to-Peer Overlay Architecture

The technical meaning of Structured is that the P2P overlay network topology is tightly controlled and content is placed not at random peers but at specified locations which results inefficient querying. Such Structured P2P systems use the Distributed Hash Table (DHT) as a substrate, according to which the location information of data-object is placed deterministically at a specific peer identified by the data object's unique key. DHT-based systems have the advantage of consistent assignment of data-objects to the nodes that constitute the network.

Data objects are assigned unique identifiers called keys, chosen from the same identifier space. Keys are mapped by the overlay network protocol to a unique live peer in the overlay network. The P2P overlay networks support the scalable storage and retrieval of *{key,value}* pairs on the overlay network, as illustrated in Figure 3.2. Given a key, a store operation *put(key,value)*and a lookup retrieval operation *value=get(key)* can be invoked to store and retrieve the data object corresponding to the key, which involves rou ting requests to the peer corresponding to the key. Each peer maintains a small routing table consisting of its neighboring peers' NodeIDs and network addresses. In the case of MESH networks, IP cannot be used since no TCP/IP routing is used. Lookup queries or message routing requests are forwarded across overlay paths to peers in a progressive manner utilizing the NodeIDs that are closer to the key in the identifier space.

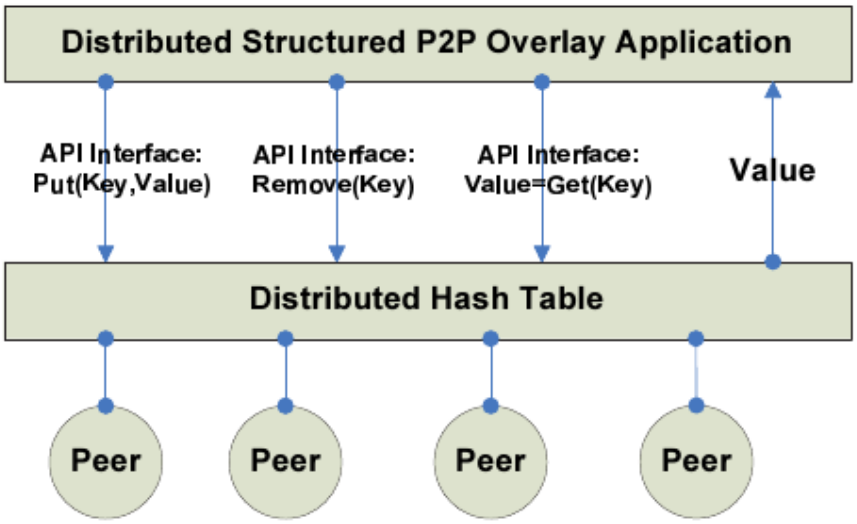


Figure 3.2 - Application Interface for Structured DHT-based P2P Overlay Systems

Different DHT-based systems have different organization schemes for the data objects and their key space and routing strategies. In theory, DHT-based systems can guarantee that any data object can be located in *O(logN)* overlay hops on average, where *N* is the number of peers in the system. The underlying network path between two peers can be significantly different from the path on the DHT-based overlay network. Therefore, the lookup latency in DHT-based P2P overlay networks can be quite high and could adversely affect the performance of the applications running over it. [C. Plaxton et al., 1997] provides an elegant algorithm that achieves nearly optimal latency on graphs that exhibit power-law expansion [L. Breslau et al., 1999], at the same time, preserving the scalable routing properties of the DHT-based system. However, this algorithm requires pair-wise probing between peers to determine latencies and it is unlikely to scale to a large number of peers in the overlay.

DHT-based systems [D. R. Karger et al., 1997] are an important class of P2P routing infrastructures. They support the rapid development of a wide variety of Internet-scale applications ranging from distributed file and naming systems to application-layer multicasting. They also enable scalable, wide-area retrieval of shared information. In 1999, Napster pioneered the idea of a peer-to- peer file sharing system supporting a centralized file search facility. It was the first system to recognize that requests for popular content need not to be sent to a central server but instead it could be handled by many peers that have the requested content. Such P2P file-sharing systems are self-scaling in that as more peers join the system, they add to the aggregate download capability. Napster achieved this self-scaling behavior by using a centralized search facility based on file lists provided by each peer, thus, it does not require much bandwidth for the centralized search. Such a system has the issue of a single point of failure due to the centralized search mechanism.

However, a lawsuit filed by the Recording Industry Association of America (RIAA) forced Napster to shut down the file-sharing service of digital music — literally, its killer application.

However, the paradigm caught the imagination of platform providers and users alike. Gnutella is a decentralized system that distributes both search and downloads' capabilities, establishing an overlay network of peers. It is the first system that makes use of an Unstructured P2P overlay network. An Unstructured P2P system is composed of peers joining the network with some loose rules, without any prior knowledge of the topology. The network uses flooding as the mechanism to send queries across the overlay with a limited scope. When a peer receives the flood query, it sends a list of all content matching the query to the originating peer. While flooding-based techniques are effective for locating highly replicated items and are resilient to peers joining and leaving the system, they are poorly suited for locating rare items. Clearly this approach is not scalable as the load on each peer grows linearly with the total number of queries and the system size. Thus, Unstructured P2P networks face one basic problem: peers readily become overloaded, therefore, the system does not scale when handling a high rate of aggregate queries and sudden increase in system size.

Although Structured P2P networks can efficiently locate rare items since the key-based routing is scalable, they incur significantly higher overheads than Unstructured P2P networks for popular content. Consequently, over the Internet today, the decentralized Unstructured P2P overlay networks are more commonly used. However, there are recent efforts on Key-based Routing (KBR) API abstractions [F. Dabek et al., 2003] that allow more application-specific functionality to be built over this common basic KBR API abstractions, and OpenHash (Open publicly accessible DHT service) [B. Karp et al., 2004] that allows the unification platform of providing developers with basic DHT service models that runs on a set of infrastructure hosts, to deploy DHT-based overlay applications without the burden of maintaining a DHT and with ease of use to spur the deployment of DHT-based applications.

In contrast, Unstructured P2P overlay systems are Ad-Hoc in nature, and do not present the possibilities of being unified under a common platform for application development. In the sections below, we will describe the key features of Structured P2P and Unstructured P2P overlay networks and their operational functionalities. After providing a basic understanding of the various overlays schemes in these two classes, an evaluation of these schemes is provided followed by some comparative results based on the following attributes:

- Decentralization: examine whether the overlay system is distributed.
- Architecture: describe the overlay system architecture with respect to its operation.

- Lookup Protocol: the lookup query protocol adopted by the overlay system.
- System Parameters: the required system parameters for the overlay system operation.
- Routing Performance: the lookup routing protocol performance in overlay routing.
- Routing State: the routing state and scalability of the overlay system.
- Peers Join and Leave: describe the behavior of the overlay system when churn and self-organization occurred.
- Security: look into the security vulnerabilities of overlay system.
- Reliability and Fault Resiliency: examine how robust the overlay system when subjected to faults.

3.1.1 Structured Networks

In this category the overlay network assigns keys to data items and organizes its peers into a graph that maps each data key to a peer. This structured graph enables efficient discovery of data items using the given keys. However, in its simple form, this class of systems does not support complex queries and it is necessary to store a copy or a pointer to each data object (or value) at the peer responsible for the data object's key. In this section, we survey and compare the following Structured P2P overlay networks: Content Addressable Network (CAN) [S. Ratnasamy et al., 2001], Tapestry [B. Y. Zhao, 2004], Chord [I. Stoica et al., 2003], Pastry [A. Rowstron et al., 2001], Kademlia [P. Maymounkov et al., 2002] and Viceroy [D. Malkhi et al., 2002].

3.1.1.1 Content Addressable Network (CAN)

The Content Addressable Network (CAN) [S. Ratnasamy et al., 2001] is a distributed decentralized P2P infrastructure that provides hash-table functionality on Internet-like scale. CAN is designed to be scalable, fault-tolerant, and self-organizing. The architectural design is a virtual multi-dimensional Cartesian coordinate space on a multi-torus. This d-dimensional coordinate space is completely logical. The entire coordinate space is dynamically partitioned among all the peers (*N* number of peers) in the system so as every peer possesses its individual, distinct zone within the overall space. A CAN peer maintains a routing table that holds the IP address and virtual coordinate zone of each of its neighbors in the coordinate space. A CAN message includes the destination coordinates. Using the neighbor coordinates, a peer routes a message towards its destination using a simple greedy forwarding to the neighbor peer that is closest to the destination coordinates. CAN has a routing performance of $O(d.N^{\frac{1}{d}})$ and its routing state is of 2.d bound.

As shown in Figure 3.3 [S. Ratnasamy et al., 2001], the virtual coordinate space is used to store *(key,value)* pairs as follows: to store a pair *{K,V}*, key *K* is deterministically mapped onto a point *P* in the coordinate space using a uniform hash function. According to the lookup protocol in order to retrieve an entry corresponding to key *K*, any peer can apply the same deterministic hash function to map *K* into point *P* and then retrieve the corresponding value *V* from the point *P*. If the requesting peer or its immediate neighbors do not own the point *P*, the request must be routed through the CAN infrastructure until it reaches the peer where *P* lays. A peer maintains the IP addresses of those peers that hold coordinate zones adjoining its zone. This set of immediate neighbors in the coordinate space serves as a coordinate routing table that enables efficient routing between points in this space.

A new peer that joins the system must have its own portion of the coordinate space allocated. This can be achieved by splitting existing peer's zone in half; retaining half for the peer and allocating the other half to the new peer. CAN has an associated DNS domain name which is resolved into IP address of one or more CAN bootstrap peers (which maintains a partial list of CAN peers). For a new peer to join CAN network, the peer looks up in the DNS a CAN domain name to retrieve a bootstrap peer's IP address, similar to the bootstrap mechanism in [P. Francis et al., 2000]. The bootstrap peer supplies the IP addresses of some randomly chosen peers in the system.

The new peer randomly chooses a point *P* and sends a JOIN request destined for point *P*. Each CAN's peer uses the CAN routing mechanism to forward the message until it reaches the peer in which zone *P* lies. The current peer in zone P then splits its zone in half and assigns the other half to the new peer.

For example, in a 2-dimensional (2 − d) space, a zone would first be split along the X-dimension, then the Y-dimension, and so on. The {K,V} pairs from the half zone to be handed over are also transferred to the new peer. After obtaining its zone, the new peer learns of the IP addresses of its neighbor set from the previous peer in point *P*, and adds to that previous peer itself.

When a peer leaves the CAN network, an immediate takeover algorithm ensures that one of the failed peer's neighbors takes over the zone and starts a takeover timer. The peer updates its neighbor set to eliminate those peers that are no longer its neighbors. Every peer in the system then sends soft-state updates to ensure that all of their neighbors will learn about the change and update their own neighbor sets.

The number of neighbors a peer maintains depends only on the dimensionality of the coordinate space (i.e. 2d) and it is independent of the total number of peers in the system. Figure 3.3 illustrates a simple routing path from peer X to point E when a new peer Z joins the CAN network.

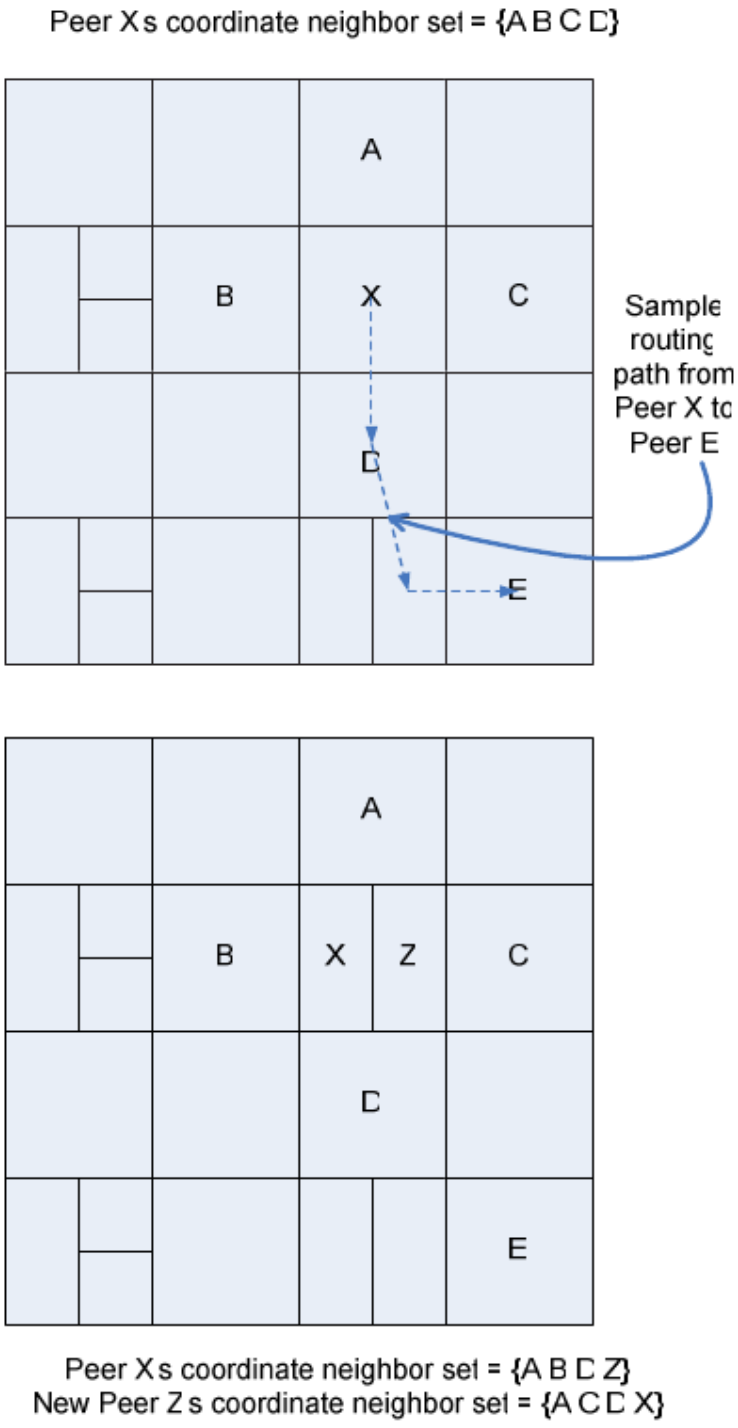


Figure 3.3 - Example of 2-d space CAN before and after peer-Z joins

For a d-dimensional space partitioned into *n* equal zones, the average routing path length is $(d/4)$ x$\left(n^{\frac{1}{d}}\right)$ hops and individual peers maintain a list of 2d neighbors. Thus,

growth of peers (or zones) can be achieved without increasing per peer state while the average path length grows as $O(n^{\frac{1}{d}})$.

Since there are many different paths between two points in the space, when one or more of a peer's neighbors fail, this peer can still route along the next best available path. Improvement to the CAN algorithm can be achieved by maintaining multiple, independent coordinate spaces with each peer in the system being assigned a different zone in each coordinate space, called reality. For a CAN with *"r"* realities, a single peer is assigned *"r"* coordinate zones, one on each reality available, and this peer holds *"r"* independent neighbor sets.

The contents of the hash table are replicated on every reality, thus improving data availability. For further data availability improvement, CAN could use *"k"* different hash functions to map a given key onto *"k"* points in the coordinate space. This results in the replication of a single *{key,value}* pair at *"k"* distinct peers in the system. A *{key,value}* pair is then unavailable only when all the *"k"* replicas are simultaneously unavailable. Thus, queries for a particular hash table entry could be forwarded to all *"k"* peers in parallel thereby reducing the average query latency, and reliability and fault resiliency properties are enhanced.

CAN could be used in large scale storage management systems such as the OceanStore [J. Kubiatowicz et al., 2002], Farsite [W. J. Bolosky et al., 2000], and Publius[M. Waldman et al., 2000]. These systems require efficient insertion and retrieval of content in a large distributed storage network with a scalable indexing mechanism. Another potential application for CANs is in the construction of wide-area name resolution services that decouple the naming scheme from the name resolution process. This enables an arbitrary and location-independent naming scheme.

3.1.1.2 Chord

Chord [I. Stoica et al., 2003] uses consistent hashing [D. Karger et al., 1997] to assign keys to its peers. Consistent hashing is designed to let peers enter and leave the network with minimal interruption. This decentralized scheme tends to balance the load on the system, since each peer receives roughly the same number of keys, and there is little movement of keys when peers join and leave the system. In a steady state, for N peers in the system, each peer maintains routing state information for about only O(logN) other peers (N number of peers in the system). This may be efficient but performance degrades gracefully when that information is out-of-date.

The consistent hash functions assign peers and data keys an m-bit identifier using SHA-1 [NIST, 1995] as the base hash function. A peer's identifier is chosen by hashing the peer's IP address, while a key identifier is produced by hashing the data key. The length of the

identifier "*m*" must be large enough to make the probability of keys hashing to the same identifier negligible. Identifiers are ordered on an identifier circle modulo *2m*.Key *k* is assigned to the first peer whose identifier is equal to or follows k in the identifier space. This peer is called the successor peer of key k, denoted by successor(k). If identifiers are represented as a circle of numbers from 0 to 2m − 1, then successor(k) is the first peer clockwise from k.

The identifier circle is termed as the Chord ring. To maintain consistent hashing mapping when a peer n joins the network, certain keys previously assigned to *n*'s successor now need to be reassigned to *n*. When peer *n* leaves the Chord system, all of its assigned keys are reassigned to n's successor. Therefore, peers join and leave the system with $(\log N)^2$ performance. No other changes of keys assignment to peers need to occur. In Figure 3.4 (adapted from [I. Stoica et al., 2003]), the Chord ring is depicted with m = 6. This particular ring has ten peers and stores five keys. The successor of the identifier 10 is peer 14, so key 10 will be located at NodeID 14. Similarly, if a peer were to join with identifier 26, it would store the key with identifier 24 from the peer with identifier 32.

Each peer in the Chord ring needs to know how to contact its current successor peer on the identifier circle. Lookup queries involve the matching of key and NodeID. For a given identifier, queries could be applied around the circle via these successor pointers until they encounter a pair of peers that include the desired identifier; the second peer in the pair is the peer the query maps to. An example is presented in Figure 3.4, whereby peer 8 performs a lookup for key 54. Peer 8 invokes the find successor operation for this key, which eventually returns the successor of that key, i.e. peer 56. The query visits every peer on the circle between peer 8 and peer 56. The response is returned along the reverse of the path.

As m is the number of bits in the key/NodeID space, each peer n maintains a routing table with up to m entries, called the finger table. The i^{th} entry in the table at peer n contains the identity of the first peer s that succeeds n by at least $2i-1$ on the identifier circle, i.e. $s = \text{successor}(n + 2^{i-1})$, where $1 \leq i \leq m$. Peer s is the i^{th} finger of peer n (n.finger[i]).

A finger table entry includes both the Chord identifier and the IP address (and port number) of the relevant peer.

Figure 3.4 shows the finger table of peer 8, and the first finger entry for this peer points to peer 14, as the latter is the first peer that succeeds (8+20) mod 26 = 9. Similarly, the last finger of peer 8 points to peer 42, i.e. the first peer that succeeds (8 + 25) mod 26 = 40. In this way, peers store information about only a small number of other peers, and know more about peers closely following it on the identifier circle than other peers. Also, a peer's finger table does not contain enough information to directly determine the

successor of an arbitrary key k. For example, peer 8 cannot determine the successor of key 34 by itself, as successor of this key (peer 38) is not present in peer 8's finger table.

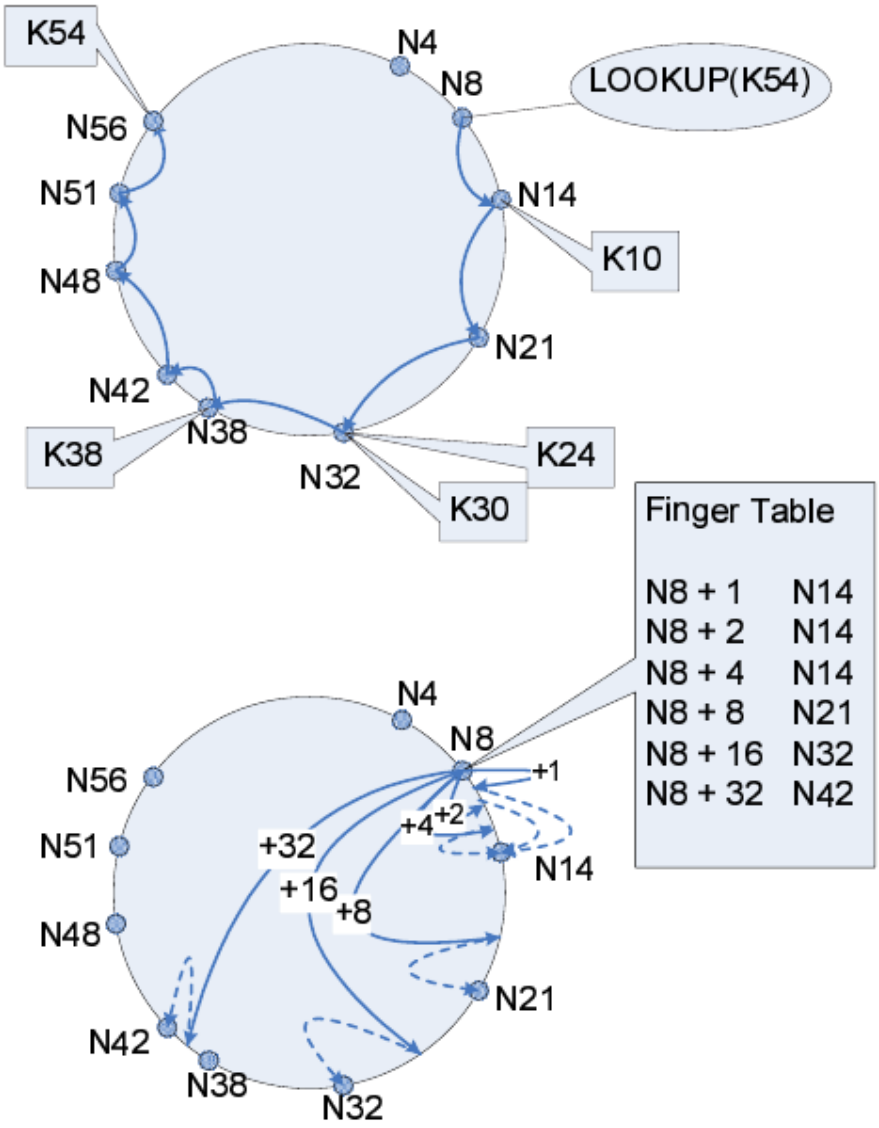


Figure 3.4 - Chord Ring of 10 peers and 5 key-value pairs.

When a peer joins the system, the successor pointers of some peers need to be changed. It is important that the successor pointers are up to date at any time because the correctness of lookups is not guaranteed otherwise. The Chord protocol uses a stabilization protocol [I. Stoica et al., 2003] running periodically in the background to update the successor pointers and the entries in the finger table. The correctness of the Chord protocol relies on the fact that each peer is aware of its successors. When peers fail, it is possible that a peer does not know its new successor and it has no chance to learn about

it. To avoid this situation, peers maintain a successor list of size *r*, which contains the peer's first *r* successors.

When the successor peer does not respond, the peer simply contacts the next peer on its successor list. Assuming that peer failures occur with a probability p, the probability that every peer on the successor list will fail is p^r. By increasing *r* makes the system more robust. By tuning this parameter, any degree of robustness with good reliability and fault resiliency may be achieved. The following applications are examples of how Chord could be used:

- Cooperative mirroring or Cooperative File System (CFS) [F. Dabek et al., 2001], in which multiple providers of content cooperate to store and serve each other's data. Spreading the total load evenly over all participant hosts lowers the total cost of the system, since each participant needs to provide capacity only for the average load, not for the peak load. There are two layers in CFS. The DHash (Distributed Hash) layer performs block fetches for the peer, distributes the blocks among the servers, and maintains cached and replicated copies. The Chord layer distributed lookup system is used to locate the servers responsible for a block.
- Chord-based DNS [R. Cox et al, 2002] provides a lookup service, with host names as keys and IP addresses (and other host information) as values. Chord could provide a DNS-like service by hashing each host name to a key [D. Karger et al., 1997]. Chord-based DNS would require no special servers, while ordinary DNS systems rely on a set of special root servers. DNS also requires manual management of the routing information (DNS records) that allows clients to navigate the name server hierarchy; Chord automatically maintains the correctness of the analogous routing information. DNS only works well when host names are hierarchically structured to reflect administrative boundaries; Chord imposes no naming structure. DNS is specialized to the task of finding named hosts or services, while Chord can also be used to find data object values that are not tied to particular machines.

3.1.1.3 Pastry

Pastry [A. Rowstron et al., 2001] makes use of Plaxton-like prefix routing to build a self-organizing decentralized overlay network, where each peer routes client requests and interacts with local instances of one or more applications. Each peer in Pastry is assigned a 128-bit peer identifier (NodeID). The NodeID is used to give a peer's position in a circular NodeID space, which ranges from 0 to $2^{128} - 1$. The NodeID is assigned randomly when a peer joins the system and it is assumed to be generated such that the resulting set of NodeIDs is uniformly distributed in the 128-bit space. For a network of N peers, Pastry routes to the numerically closest peer to a given key in less than $\log_B N$ steps under normal operation (where $B = 2^b$ is a configuration parameter with typical value of b = 4). The

NodeIDs and keys are considered as a sequence of digits with base B. Pastry routes messages to the peer whose NodeID is numerically closest to the given key. A peer normally forwards the message to a peer whose NodeID shares with the key a prefix that is at least one digit (or b bits) longer than the prefix that the key shares with the current peer NodeID.

As shown in Figure 3.5, each Pastry peer maintains a routing table, a neighborhood set and a leaf set. A peer routing table is designed with $\log_B N$ rows, where each row holds B – 1 number of entries. Each of the B − 1 number of entries at row n of the routing table refers to a peer whose NodeID shares the current peer's NodeID in the first n digits, but whose $(n+1)^{th}$ digit has one of the B − 1 possible values other than the $(n+1)^{th}$ digit in the current peer's NodeID. Each entry in the routing table contains the IP address of peers whose NodeID have the appropriate prefix, and it is chosen according to close proximity metric. The choice of b involves a trade-off between the size of the populated portion of the routing table [approx. $(\log_B N) \times (B-1)$ entries] and maximum number of hops required to route between any pair of peers ($\log_B N$).

With a value of b = 4 and 10^6 peers, a routing table contains on average 75 entries and the expected number of routing hops is 5. The neighborhood set, M contains the NodeIDs and IP addresses of the |M| peers that are closest in proximity to the local peer. The network proximity that Pastry uses is based on a scalar proximity metric such as the number of IP routing geographic distance. The leaf set L is the set of peers with |L|/2 numerically closest larger NodeIDs and |L|/2 peers with numerically smaller NodeIDs, in relation to the current peer's NodeID. Typical values for |L| and |M| are B or 2 x B. Even in case of concurrent peers' failure, eventual delivery is guaranteed with good reliability and fault resiliency unless |L|/2 peers with adjacent NodeIDs fail simultaneously, (|L| is a configuration parameter with a typical value of 16 or 32).

When a new peer (NodeID is X) joins the network, it needs to initialize its state table and inform other peers of its presence. This joining peer needs to know the address of a contact peer in the network. A small list of contact peers, based on a proximity metric (e.g. the RTT to each peer) to provide better performance, could be provided as a service in the network, and the joining peer could select at random one of the peers to be its contact peer. So, this new peer knows initially about a nearby Pastry peer A, according to a proximity metric, from the list of contact peers. Peer X asks A to route a special join message with the key equal to X. Pastry routes the join message to the existing peer Z whose NodeID is numerically closest to X. In response to receiving the join request, peers A, Z and all peers encountered on the path from A to Z send their state tables to X. Finally, X informs any peers that need to be aware of its arrival. This ensures that X initializes its state with appropriate values and that the state in all other affected peers is updated.

Routing Table of a Pastry peer with NodeID 37A0x, b = 4, digits are in hexadecimal, x is an arbitrary suffix

0x	1x	2x	**3x**	4x	...	Dx	Ex	Fx
30x	31x	32x	...	**37x**	38x	...	3Ex	3Fx
370x	371x	372x	...	**37Ax**	37Bx	...	37Ex	37Fx
37A0x	37A1x	37A2x	...	37ABx	37ACx	37ADx	37AEx	37AFx

Example: Routing State of a Pastry peer with NodeID 37A0F1, b = 4, L=16, M=32

NodeID 37A0F1			
Leaf Set (Smaller)			
37A001	37A011	37A022	37A033
37A044	37A055	37A066	37A077
Leaf Set (Larger)			
37A0F2	37A0F4	37A0F6	37A0F8
37A0FA	37A0FB	37A0FC	37A0FE
Neighborhood Set			
1A223B	1B3467	245AD0	2670AB
3612AB	37890A	390AF0	3912CD
46710A	477810	4881AB	490CDE
279DE0	290A0B	510A0C	5213EA
11345B	122167	16228A	19902D
221145	267221	28989C	199ABC

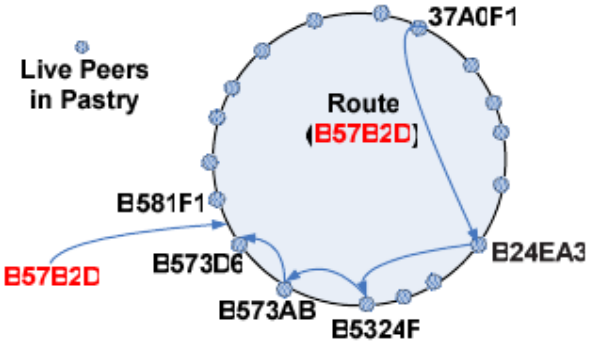


Figure 3.5 - An example of routing path for a Pastry peer

As peer A is assumed to be topologically close to the new peer X, A's neighborhood set initializes X's neighborhood set. Considering the general case, where NodeIDs of A and X share no common prefix, let A_i denote peer A's row of the routing table at level i. A0 contains appropriate values for X0 , since the entries in row 0 of the routing table are independent of a peer's NodeID. Other levels of A's routing table are of no use to X, since A's and X's NodeIDs share no common prefix. Appropriate values for X1 can be taken from

B1, when B is the first peer along the route path from A to Z. The entries in B1 and X1 share the same prefix, because X and B have the same first digit in their NodeID. Finally, X transmits a copy of its resulting state to each of the peers found in its neighborhood set, leaf set and routing table. These peers then update their own state based on the information received.

A Pastry peer is considered failed when its immediate neighbors in the NodeID space can no longer communicate with the peer. To replace a failed peer in the leaf set, its neighbor in the NodeID space contacts the live peer with the largest index on the side of the failed peer, and requests its leaf table. For example, if L_i failed for $|L|/2 < i < 0$, it would request the leaf set from $L - |L|/2$. Let the received leaf set be L', which overlaps the current peer's leaf set L, and it contains peers with nearby NodeIDs not present in L. The appropriate one is then chosen to insert into L, verifying that the peer is actually still alive by contacting it. To repair the failed routing table entry $R_{d(levell)}$, a peer contacts first the peer referred to by another entry $R_{i(levell)}$, $i \neq d$ of the same row, and asks for that peer's entry for Rd. If none of the entries in row l has a pointer to a live peer with appropriate prefix, the peer contacts an entry $R_{i(levell+1)}$, $i \neq d$, thereby casting a wider coverage. The neighborhood set is not used in the routing of messages, but it is still kept fresh/update because the set plays an important role in exchanging information about nearby peers. Therefore, a peer contacts each member of the neighborhood set periodically to see if it is still alive.

If the peer is not responding, the peer asks other members for their neighborhood sets and checks for the closest proximity of each of the newly discovered peers and updates its own neighborhood set. Pastry is being used in the implementation of an application-level multicast, called SplitStream [M. Castro et al., 2003].

Instead of relying on a multicast infrastructure in the network which is not widely available, the participating peers route and distribute multicast messages using only unicast network services. SplitStream allows a cooperative environment where peers contribute resources in exchange for using the service. The key idea is to split the content into *k* stripes and to multicast each stripe using a separate tree. Peers join as many trees as there are stripes they wish to receive and they specify an upper bound on the number of stripes that they are willing to forward. The challenge is to construct this forest of multicast trees such that an interior peer in one tree is a leaf peer in all the remaining trees and the bandwidth constraints specified by the peers are satisfied. This ensures that forwarding load can be spread across all participating peers. For example, if all peers wish to receive k stripes and they are willing to forward k stripes, SplitStream will construct a forest such that the forwarding load is evenly balanced across all peers while achieving low delay and link stress across the network.

Scribe [A. Rowstron et al., 2001], [M. Castro et al., 2002] is a scalable application-level multicast infrastructure that supports a large number of groups with large number of members per group. Scribe is built on top of Pastry which is used to create and manage groups and to build efficient multicast trees for the dissemination of messages to each group. Scribe builds a multicast tree formed by joining Pastry routes from each group member to a rendezvous point associated with a group. Membership maintenance and message dissemination in Scribe leverages the robustness, self-organization, locality and reliability properties of Pastry.

Squirrel [S. Iyer et al., 2002] uses Pastry as its data object location service to identify and route to peers that cache copies of a requested data object. It facilitates mutual sharing of web data objects among client peers and enables the peers to export their local caches to other peers in the network, thus creating a large shared virtual web cache. Each peer then performs both web browsing and web caching without the need of expensive and dedicated hardware for centralized web caching. Squirrel faces a new challenge whereby peers in a decentralized cache incur the overhead of having to serve each other requests, and this extra load must be kept low.

PAST [P. Druschel et al, 2001], [A. Rowstron et al., 2001] is a large scale P2P persistent storage utility, based on Pastry. The PAST system is composed of peers connected to the Internet where each peer is capable of initiating and routing client requests to insert or retrieve files. Peers may also contribute storage to the system. A storage system like PAST is attractive because it exploits the multitude and diversity of peers in the Internet to achieve strong persistence and high availability. This eradicates the need for physical transport of storage media to protect lookup and archival data, and the need for explicit mirroring to ensure high availability and throughput for shared data. A global storage utility also facilitates the sharing of storage and bandwidth, thus permitting a group of peers to jointly store or publish content that would exceed the capacity or bandwidth of any individual peer.

Pastiche [L. P. Cox et al., 2002] is a simple and inexpensive backup system that exploits excess disk capacity to perform P2P backup with no administrative costs. The cost and inconvenience of backup are unavoidable, and often prohibitive. Small-scale solutions require significant administrative efforts. Large-scale solutions require aggregation of substantial demand to justify the capital costs of a large, centralized repository. Pastiche builds on three architecture: Pastry which provides the scalable P2P network with self-administered routing and peer location; Content-based indexing [A. Muthitacharoen et al., 2001], [U. Manber et al., 1994] which provides flexible discovery of redundant data for similar files; and Convergent encryption [W. J. Bolosky et al., 2000] which allows hosts to use the same encrypted representation for common data without sharing keys.

3.1.1.4 Tapestry

Sharing similar properties as Pastry, Tapestry [B. Y. Zhao, 2004] employs decentralized randomness to achieve both load distribution and routing locality. The difference between Pastry and Tapestry is the handling of network locality and data object replication, and this difference will be more apparent, as described in Pastry section. Tapestry's architecture uses variant of the [Plaxton et al., 1997] distributed search technique, with additional mechanisms to provide availability, scalability, and adaptation in the presence of failures and attacks. [Plaxton et al., 1997] proposes a distributed data structure, known as the Plaxton mesh, optimized to support a network overlay for locating named data objects which are connected to one root peer. On the other hand Tapestry uses multiple roots for each data object to avoid a single point of failure. In the Plaxton mesh, peers can take on the roles of servers (where data objects are stored), routers (forward messages), and clients (entity of requests). It uses local routing maps at each peer to incrementally route overlay messages to the destination ID digit by digit, for instance, $***7 \Rightarrow ** 97 \Rightarrow *297 \Rightarrow 3297$, where "$*$" is the wildcard, similar to the longest prefix routing in the CIDR IP address allocation architecture [Y. Rekhter et al., 1993]. The resolution of digits from right to left or left to right is arbitrary. A peer's local routing map has multiple levels, where each of them represents a matching of the suffix up to a digit position in the ID space. The n^{th} peer that a message reaches shares a suffix of at least length n with the destination ID. To locate the next router, the $(n + 1)^{th}$ level map is examined to locate the entry matching the value of the next digit in the destination ID. This routing method guarantees that any existing unique peer in the system can be located within at most $\log_B N$ logical hops, in a system with N peers using NodeIDs of base B. Since the peer's local routing map assumes that the preceding digits all match the current peer's suffix, the peer needs only to keep a small constant size (B) entry at each route level, yielding a routing map of fixed constant size: (entries/map) x no.of maps = $B\log_B N$.

The lookup and routing mechanisms of Tapestry is similar to Plaxton, which are based on matching the suffix in NodeID as described above. Routing maps are organized into routing levels, where each level contains entries that point to a set of peers closest in distance that matches the suffix for that level. Also, each peer holds a list of pointers to peers referred to as neighbors. Tapestry stores the locations of all data object replicas to increase semantic flexibility and allow application level to choose from a set of data object replicas based on some selection criteria, such as date. Each data object may include an optional application-specific metric in addition to a distance metric; e.g. OceanStore [J. Kubiatowicz et al., 2002] global storage architecture finds the closest cached document replica which satisfies the closest distance metric. These queries deviate from the simple

"find first" semantics, and Tapestry will route the message to the closest k distinct data objects.

Tapestry handles the problem of a single point of failure due to a single data object's root peer by assigning multiple roots to each object. Tapestry makes use of surrogate routing to select root peers incrementally, during the publishing process to insert location information into Tapestry. Surrogate routing provides a technique by which any identifier can be uniquely mapped to an existing peer in the network. A data object's root or surrogate peer is chosen as the peer which matches the data object's ID, X. This is unlikely to happen, given the sparse nature of the NodeID space. Nevertheless, Tapestry assumes peer X exists by attempting to route a message to it. A route to a non-existent identifier will encounter empty neighbor entries at various positions along the way. The goal is to select an existing link, which can act as an alternative to the desired link; i.e. the one associated with a digit X. Routing terminates when a map is reached where the only non-empty routing entry belongs to the current peer. That peer is then designated as the surrogate root for the data object. While surrogate routing may take additional hops to reach a root if compared with Plaxton algorithm, the additional number of hops is small. Thus, surrogate routing in Tapestry has minimal routing overhead relative to the static global Plaxton algorithm.

Tapestry addresses the issue of fault adaptation and maintains cached content for fault recovery by relying on TCP timeouts and UDP periodic heartbeats packets to detect link, server failures during normal operations, and rerouting through its neighbors. During fault operation, each entry in the neighbor map maintains two backup neighbors in addition to the closest/primary neighbor. On a testbed of 100 machines with 1000 peers simulations, the results in [Y. Shavitt et al., 2004] indicate that good routing rates and maintenance bandwidths under instantaneous failures and continuing churn have been achieved.

A variety of different applications have been designed and implemented on Tapestry. Tapestry is self-organizing, fault-tolerant, resilient under load, and it is a fundamental component of the OceanStore system [J. Kubiatowicz et al., 2002], [S. Rhea, 2001]. The OceanStore is a global-scale, highly available storage utility deployed on the PlanetLab [L. Peterson, 2003] testbed. OceanStore's servers use Tapestry to disseminate encoded file blocks efficiently, and clients can quickly locate and retrieve nearby file blocks by their ID, despite server and network failures. Other Tapestry applications include the Bayeux [S. Q. Zhuang, 2001], an efficient self-organizing application-level multicast system, and SpamWatch [F. Zhou et al., 2003], a decentralized spam-filtering system that uses a similarity search engine implemented on Tapestry.

3.1.1.5 Kademlia

The Kademlia [P. Maymounkov et al., 2002] P2P decentralized overlay network takes the basic approach of assigning each peer a NodeID in the 160-bit key space, and {key,value} pairs are stored on peers with IDs close to the key. A NodeID-based routing algorithm will be used to locate peers near a destination key. One of the key architecture of Kademlia is the use of a novel XOR metric for distance between points in the key space. XOR is symmetric and it allows peers to receive lookup queries from precisely the same distribution of peers contained in their routing tables. Kademlia can send a query to any peer within an interval, allowing it to select routes based on latency or send parallel asynchronous queries. It uses a single routing algorithm throughout the process to locate peers near a particular ID.

Every message being transmitted by a peer includes its peer ID, permitting the recipient to record the sender peer's existence. Data keys are also 160-bit identifiers. To locate *{key,value}* pairs, Kademlia relies on the notion of distance between two identifiers. Given two 160-bit identifiers, a and b, it defines the distance between them as their bitwise exclusive OR (XOR, interpreted as $d(a, b) = a \oplus b = d(b, a)$ for all a, b), and this is a non-Euclidean metric. Thus, $d(a, b) = 0$, $d(a, b) > 0$(if $a \neq b$), and for all a, b: $d(a, b) = d(b, a)$.

XOR also offers the triangle inequality property: $d(a, b) + d(b, c) \geq d(a, c)$, since $d(a, c) = d(a, b) \oplus d(b, c)$ and $(a+b \geq a \oplus b)$ for all a, b = 0.

Similarly to Chord's clockwise circle metric, XOR is unidirectional. For any given point x and distance $d > 0$, there is exactly one point y such that $d(x, y) = d$. The unidirectional approach makes sure that all lookups for the same key converge along the same path, regardless of the originating peer. Hence, caching {key,value} pairs along the lookup path alleviates hot spots. The peer in the network stores a list of {IP address, UDP port, NodeID} triples for peers of distance between 2^i and 2^{i+1} from itself. These lists are called k-buckets. Each k-bucket is kept sorted by last time seen; i.e. least recently accessed peer at the head, most-recently accessed at the tail.

The Kademlia routing protocol consists of the following set of functions:

- PING that probes a peer to check if it is active.
- STORE that instructs a peer to store a {key,value} pair for later retrieval.
- FIND NODE that provides to the node a 160-bit ID and returns {IP address, UDP port, NodeID} triples for the k peers it knows that are closest to the target ID.
- FIND VALUE that is similar to FIND NODE and returns {IP address, UDP port, NodeID} triples. Kademlia's peer must locate the k closest peers to some given NodeID. This lookup initiator starts by picking X peers from its closest non-empty k-bucket, and then sends parallel asynchronous FIND NODE to the X peers it has chosen.

3.1.1.6 Viceroy

The Viceroy [D. Malkhi et al., 2002] P2P decentralized overlay network is designed to handle the discovery and location of data and resources in a dynamic butterfly fashion. Viceroy employs consistent hashing [D. Karger et al., 1997] to distribute data so that it is balanced across the set of servers and resilient to servers joining and leaving the network. It utilizes the DHT to manage the distribution of data among a changing set of servers and allowing peers to contact any server in the network to locate any stored resource by name. In addition to this, Viceroy maintains an architecture that is an approximation to a butterfly network [H. J. Siegel et al., 1979], as shown in Figure 3.6 (adapted from diagram in [D. Malkhi et al., 2002]), and uses links between successors and predecessors – ideas that were based on [J. Kleinberg et al., 2000] and [L. Barriere et al., 2001]-on the ring (a key is mapped to its successor on the ring) for short distances. Its diameter of the overlay is better than CAN and its degree is better than Chord, Tapestry and Pastry.

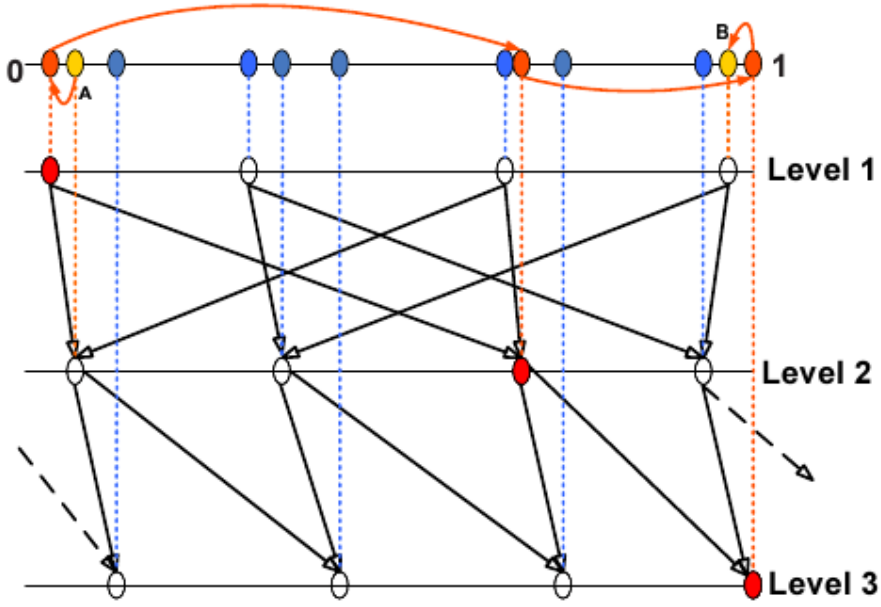


Figure 3.6 - Simplified Viceroy Network

When N peers are operational, one of logN levels is selected with near equal probability. Level l peer's two edges are connected to peers at level l + 1. A down-right edge is added to a long-range contact at level l + 1 at a distance about $\frac{1}{2^l}$ away, and a down-left edge at a close distance on the ringto the level l + 1. The up edge to a nearby peer at level l – 1 is included if l > 1. Then, level-ring links are added to the next and previous peers of the same level l. Routing is done by climbing, using up connections to a level l − 1 peer. Then, it proceeds down the levels of the tree using the down links, and moving from level l to level l + 1. It follows either the edge to the nearby down link or the further down link, depending on distance > $\frac{1}{2^l}$. This continues recursively until a peer is

reached with no down links, and it is in the vicinity of the target peer. So, a vicinity lookup is performed using the ring and level-ring links. For reliability and fault resiliency, when a peer leaves the overlay network, it hands over its key pairs to a successor from the ring pointers and notifies other peers to find a replacement. It is formalized and proved [D. Malkhi et al., 2002] that the routing process requires only O(logN) messages, where N is the number of peers in the network.

3.1.2 Comparative View of Structured (DHT) implementations

The algorithm of Plaxton was originally devised to route web queries to nearby caches and this influenced the design of Pastry, Tapestry and Chord. The method of Plaxton has logarithmic expected join/leave complexity. Plaxton ensures that queries never travel further in network distance than the peer where the key is stored. However, Plaxton has several disadvantages: it requires global knowledge to construct the overlay; an object's root peer is the single point of failure; no insertion or deletion of peers; no avoidance to hotspots congestion. Pastry and Tapestry schemes relied on DHT to provide the substrate for semantic-free and data-centric references, through the assignment of a semantic-free NodeID, such as a 160-bit key, and performed efficient request routing between lookup peers using an efficient and dynamic routing infrastructure, whereby peers leave and join. Overlays that perform query routing in DHT-based systems have strong theoretical foundations, guaranteeing that a key can be found if it exists and they do not capture the relationships between the object name and its content. However, DHT-based systems have a few problems in terms of data object lookup latency.

For each overlay hop, peers route a message to the next intermediate peer that can be located very far away with regard to the physical topology of the underlying IP network. This can result in high network delay and unnecessary long-distance network traffic, from a deterministic short overlay path of O(logN), (where N is the number of peers). DHT-based systems assume that all peers equally participate in hosting published data objects or their location information. This would lead to a bottleneck at low-capacity peers.

Pastry and Tapestry routing algorithms are a randomized approximation of a hypercube and routing towards an object is done by matching longer addresses suffixes, until either the object's root peer or another peer with where a nearby copy is found. [S. Rhea et al., 2003] makes use of FreePastry implementation to discover that most lookups fail to complete when there is excessive churn. They claimed that short-lived peers leave the overlay with lookups that have not yet timed out. They outlined design issues pertaining to DHT-based performance under churn: lookup timeouts, reactive versus periodic recovery of peers; and the choice of nearby neighbors.

Since the reactive recovery will increase traffic to congested links, they make use of periodic recovery and for lookup, they suggested for an exponential weighted moving average of each neighbor's response time instead of alternative fixed timeout. They discovered that selection of nearby neighbors, required global sampling which is more effective than simply sampling neighbor's neighbors. However, [Castro et al., 2004] uses the MSPastry implementation to show that it can cope with high churn rates by achieving shorter routing paths and less maintenance overhead. Pastry exploits network locality to reduce routing delays by measuring the delay Round-Trip-Time (RTT) to a small number of peers when building the routing tables. For each routing table entry, it chooses one of the closest peers in the network topology whose NodeID satisfies the constraints for that entry. The average IP delay of each Pastry hop increases exponentially until it reaches the average delay between two peers in the network. Chord's routing protocol is similar to Pastry's location algorithm in PAST. However, Pastry is a prefix-based routing protocol and differ in other details from Chord. Chord maps keys and peers to an identifier ring and guarantees that queries make a logarithmic number hops and that keys are well balanced. It uses Consistent Hashing to minimize disruption of keys when peers leave and join the overlay network. Consistent Hashing ensures that the total number of caches responsible for a particular object is limited and when these caches change the minimum number of object-references will move to maintain load balancing. Since the Chord lookup service presents a solution where each peer maintains a logarithmic number of long-range links, it gives a logarithmic join/leave update. In Chord, the network is maintained appropriately by a background maintenance process, i.e. a periodic stabilization procedure that updates predecessor and successor pointers to cater for newly joined peers.

[Liben-Nowell et al., 2002] asks the question of how often the stabilization procedure needs to run to determine the success of Chord's lookups and determining the optimum involves the measurements of peers' behavior. [Stoica et al., 2003] demonstrates that the advantage of recursive lookups over iterative lookups, but future improvement work is proposed to improve resiliency to network partitions using a small set of known peers, and to reduce the amount of messages in lookups by increasing the size of each step around the ring with larger fingers in each peer. [Alima et al., 2003] proposes a correction-on-use mechanism in their Distributed K-ary Search (DKS), which is similar to Chord, to reduce the communication costs incurred by Chord's stabilization procedure. The mechanism makes correction to the expired routing entries by piggybacking lookups and insertions.

The work on CAN is related to the creation of a constant degree network for routing lookup requests. It organizes the overlay peers into a d-dimensional Cartesian coordinate space, where each peer undertakes the ownership of a specific hyper-rectangular shape in the space. The key motivation of CAN design is based on the argument that Plaxton-based schemes would not perform well under churn, given peer departures and arrivals would

affect a logarithmic number of peers. It maintains a routing table with its adjacent immediate neighbors. Peers joining the CAN cause the peer owning the region of space to split, giving half to the new peer and retaining half. Peers leaving the CAN will pass its NodeID, neighbors' NodeID, IP addresses and its {key,value} pairs to a takeover peer.

CAN has a number of tunable parameters to improve routing performance: Dimensionality of the hypercube; network-aware routing by choosing the neighbor closest to the destination in CAN space; multiple peers in a zone, allowing CAN to deliver messages to anyone of the peers in the zone in an anycast manner; uniform partitioning allowed by comparing the volume of a region with the volumes of neighboring regions when a peer joins; and landmark-based placement that causes peers, at join time, to probe a set of well-known landmark hosts, estimating each of their network distances. There are open research questions on CAN's resiliency, load balancing, locality and latency/hopcount costs.

The Kademlia's XOR topology-based routing resembles very much the first phase in the routing algorithms of Pastry, Tapestry and Plaxton. For these three algorithms, there is a need for an additional algorithmic structure for discovering the target peer within the peers that share the same prefix but differ in the next b-bit digit. In [P. Maymounkov et al., 2002], it is argued that Pastry and Tapestry algorithms require secondary routing tables of size $O(2^b)$ in addition to the main tables of size $O(2^b \log_{2^b} N)$, which increases the cost of bootstrapping and maintenance.

Kademlia resolves in their distinctive ways by the use of XOR metric for the distance between 160-bit NodeID and each peer maintains a list of contact peers, which longer-lived peers are given preference on this list. Kademlia can easily be optimized with a base other than 2, by configuring the bucket table so that it approaches the target b bits per hop. This needs having one bucket for each range of peers at a distance $[j.2^{160-(i+1)b}, (j + 1).2^{160-(i+1)b}]$, for each $0 < j < 2^b$ and $0 \leq i < \frac{160}{b}$. This expects no more than $(2^b - 1)(\log_{2^b} N)$ buckets.

The Viceroy's overlay network (butterfly) presents an efficient network construction proved formally in [D. Malkhi et al., 2002] in order to maintain constant degree networks in a dynamic environment, similar to CAN. Viceroy has logarithmic diameter, similar to Chord, Pastry and Tapestry. Viceroy's diameter is proven to be better than CAN and its degree is better than Chord, Pastry and Tapestry. Its routing achieved is O(logN) hops (where N is the number of peers) and with nearly optimal congestion.

Peers joining and leaving the system induce O(logN) hops and require only O(1) peers to change their states. [X. Li et al., 2002] pinpoint that limited degree may increase the risk of network partition or limitations in the use of local neighbors. However, its advantage is the constant-degree overlay properties. [F. Kaashoek et al., 2003] highlights about its fault-tolerant blind spots and its complexity.

Further work was done by Viceroy's authors with proposal of a two-tier, locality-aware DHT [I. Abraham et al., 2004] which gives lower degree properties in each lower-tier peer, and the bounded-degree P2P overlay using de Bruijn graph [N. D. de Bruijn et al., 1946]. Since de Bruijn graphs give very short average routing distances and high resilience to peer failure, they are well suited for structured P2P overlay networks. The P2P overlays discussed above are "greedy", and for a given degree, the algorithms are suboptimal because the routing distance is longer. There are increasing de Bruijn P2P overlay proposals [F. Kaashoek et al, 2003], [M. Naor et al., 2003], [D. Loguinov et al, 2003], [M. Naor et al., 2003], [P. Fraigniaud et al., 2003]. The de Bruijn graph of degree k (k can be varied) could achieve an asymptotically optimum diameter (maximum hop counts between any two peers in the graph) of $\log_k N$, where N is the total number of peers in the system. Given O(logN) neighbors in each peer, the de Bruijn graphs' hop count is $O(\frac{logN}{loglogN})$. A good comparison study is done by [D. Loguinov et al, 2003], where instances of Chord, CAN and de Bruijn are used in order to study routing performance and resilience of P2P overlay networks, including graph expansion and clustering properties. They confirmed that de Bruijn graphs for a given degree k offer the best diameter and average distance between all pairs of peers (this determines the expected response time in number of hops), optimal resilience (k-peer connectivity), large bisection width (bisection width of a graph provides tight upper bounds on the achievable capacity of the graph), and good node (peer) expansion that guarantees little overlap between parallel paths to any destination peer (if there is a peer failure, very few alternative paths to a destination peer are affected).

P2P DHT-based overlay systems are susceptible to security breach from malicious peers' attacks. One simple attack on DHT-based overlay system could be the malicious return of wrong data objects to lookup queries. The authenticity of the data objects can be handled by using cryptographic techniques through some cost-effective public keys and/or content hashes to securely link together different pieces of data objects. Such techniques can neither prevent undesirable data objects from polluting the search results, nor prevent denial of attacks. Malicious peers may still be able to corrupt, deny access or response to lookup queries of replicas of a data object, and impersonate so that replicas may be stored on illegitimate peers.

[E. Sit et al., 2002] provides a very clear description of security considerations that involve the adversaries which are peers in the DHT overlay lookup system that do not follow the protocol correctly: malicious peers are able to eavesdrop the communication between other nodes; malicious peer can only receive data objects addressed to its IP address, and thus, IP address can be a weak form of peer identity; malicious peers can collude together giving believable false information. They presented a taxonomy of possible attacks involving routing deficiencies due to corrupted lookup routing and updates; vulnerability to partitioning and virtualization into incorrect networks when a

new peer joins and contacts a malicious peer; lookup and storage attacks; inconsistent behaviors of peers; denial of service attacks preventing access by overloading victim's network connection; and unsolicited responses to a lookup query. Defenses design principles can be classified as defining verifiable system invariants for lookup queries, NodeID assignment, peers selection in routing, cross checking using random queries, and avoid single point of responsibility.

[M. Castro et al., 2002] relates the problems of secure routing for structured P2P overlay networks, in terms of the possibilities that a small number of peers could compromise the overlay system, if peers are malicious and conspire with each other (this is also termed as Eclipse attack [A. Singh et al., 2004]). They presented a design and analysis of techniques for secure peer joining, routing table maintenance, and robust message forwarding in the presence of malicious peers in Structured P2P overlays. The technique can tolerate up to 25% of malicious peers while providing good performance when the number of compromised peers is small. However, this defense restricts the flexibility necessary to implement optimizations such as proximity neighbor selection and only works in Structured P2P overlay networks. So, [A. Singh et al., 2004] proposes a defense that prevents Eclipse attacks for both Structured and Unstructured P2P overlay networks, by bounding degree of overlay peers, i.e. the in-degree of overlay peers is likely to be higher than the average in-degree of legitimate peers and legitimate peers choose their neighbors from a subset of overlay peers whose in-degree is below a threshold. Having done the in-degree bounding, it is still possible for the attacker to consume the in-degree of legitimate peers and prevent other legitimate peers from referencing to them, therefore, bounding the out-degree is necessary so that legitimate peers choose neighbors from the subset of overlay peers whose in-degree and out-degree are below some threshold. An auditing scheme is also introduced to prevent incorrect information of the in-degree and out-degree.

Another good survey on security issues in P2P is from [D. S.Wallach et al., 2002], which describes that secured routing primitives: how to assign NodeIDs, maintain routing tables, and forward messages securely. He also suggested looking at distributed auditing the sharing of disk space resources in a P2P overlay network as a barter economy, and the mechanism to implement such an economy. The work on BarterRoam [E. K. Lua et al., 2004] sheds light on a formal computational approach that is applicable to P2P overlay systems towards exchanging resources so that higher level functionality, such as incentive-compatible economic mechanisms can be layered at the higher layers.

Formal game theoretical approach and models [C. Buragohain et al., 2003], [P. Golle et al., 2001], [K. Lai et al., 2003] could be constructed to analyze equilibrium of user strategies to implement incentives for cooperation. The ability to overcome free-rider problems in P2P overlay networks will definitely improve the system's reliability and its value Sybil

attack termed by Douceur [J. R. Douceur et al., 2002]. [J. R. Douceur et al., 2002] described the situation whereby there are a large number of potentially malicious peers in the system and without a central authority to certify peers' identities. It becomes very difficult to trust the claimed identity. [R. Dingledine et al., 2001] proposes puzzles schemes, including the use of micro-cash, which allow peers to build up reputations. Although this proposal provides a degree of accountability, this still allows a resourceful attacker to launch attacks. Many P2P computational models of trust and reputation systems have emerged to assess trustworthiness behavior through feedback and interaction mechanisms. The basic assumption of these computational trust and reputation models is that the peers engage in bilateral interactions and evaluations done on a globally agreed scale. However, most of such trust and reputation systems suffer from two problems, as highlighted by [Z. Despotovic et al., 2004]: extensive implementation overhead and vague trust related model semantics. The causes lie in the aggregation of the feedback about all peers in the overlay network in order to assess the trustworthiness of a single peer, and also the anti-intuitive feedback aggregation strategies resulting in outputs that are difficult to interpret.

Lastly, since each of the basic Structured P2P DHT- based systems defines different methods in the higher level DHT abstractions to map keys to peers and other Structured P2P application-specific systems such as cooperative storage, content distribution and messaging; there is effort [F. Dabek et al., 2003] in defining basic common API abstractions for the common services they provide which they called key-based routing API (KBR) at lower tier of the abstractions. At higher tier, more abstractions can be built upon this basic KBR. In addition to DHT abstraction which provides the same functionality as the hash table in Structured P2P DHT-based systems by mapping between keys and objects, the groups anycast and multicast (CAST) (provides scalable group communication and coordination) and decentralized object location and routing (DOLR) (provides a decentralized directory service) are also defined. However, [B. Karp et al., 2004] points out that the above mentioned bundled library model where the applications read the local DHT state and receive upcalls from the DHT, requires the codes for the same set of applications to be available at all DHT hosts. This prevents the sharing of a single DHT deployment by multiple applications and generates maintenance traffic from running the DHT on its underlying infrastructure. Thus, they proposed OpenHash with ReDiR, a distributed rendezvous service model that requires only put()/get() interfaces and shares a common DHT routing platform. Table 3-1 summarizes the characteristics of Structured P2P overlay networks which have been already discussed.

Algorithm Taxonomy	Structured P2P Overlay Network Comparisons					
	CAN	Chord	Tapestry	Pastry	Kademlia	Viceroy
Decentralization	DHT functionality on Internet-like scale					
Architecture	Multi-dimensional ID coordinate space.	Uni-directional and Circular NodeID space.	Plaxton-style global mesh network.	Plaxton-style global mesh network.	XOR metric for distance between points in the key space.	Butterfly network with connected ring of predecessor and successor links, data managed by servers.
Lookup Protocol	key,value pairs to map a point P in the coordinate space using uniform hash function.	Matching Key and NodeID.	Matching suffix in NodeID.	Matching Key and prefix in NodeID.	Matching Key and Node-ID based routing.	Routing through levels of tree until a peer is reached with no downlinks; vicinity search performed using ring and level ring links.
System Parameters	*N*-number of peers in network *d*-number of dimensions.	*N*-number of peers in network.	*N*-number of peers in network *B*-base of the chosen peer identifier.	*N*-number of peers in network *b*-number of bits ($B = 2^b$) used for the base of the chosen identifier.	*N*-number of peers in network *b*-number of bits (($B = 2^b$) of NodeID.	*N*-number of peers in network.
Routing Performance	$O(d.N^{\frac{1}{d}})$	$O(logN)$	$O(\log_B N)$	$O(\log_B N)$	$O(\log_B N) + c$ where c = small constant	$O(logN)$
Routing State	$2d$	$logN$	$\log_B N$	$B \log_B N + B \log_B N$	$B \log_B N + B$	$logN$
Peers Join/Leave	$2d$	$(logN)^2$	$\log_B N$	$\log_B N$	$\log_B N + c$ where c = small constant	$logN$
Security	Low level. Suffers from man-in-middle and Trojan attacks.					
Reliability/Fault Resiliency	Failure of peers will not cause network wide failure. Multiple peers responsible for each data item. On failures, application retries.	Failure of peers will not cause network-wide failure. Replicate data on multiple consecutive peers. On failures, application retries.	Failure of peers will not cause network-wide failure. Replicate data across multiple peers. Keep track of multiple paths	Failure of peers will not cause network-wide failure. Replicate data across multiple peers. Keep track of multiple paths to each peer.	Failure of peers will not cause network wide failure. Replicate data across multiple peers.	Failure of peers will not cause network wide failure. Load incurred by lookups routing evenly distributed among participating lookup servers.

Table 3-1 - Comparison of Structured P2P approaches

3.1.3 Unstructured Networks

In this category, the overlay networks organize peers in a random graph in flat or hierarchical manners (e.g. Super-Peers layer) and use flooding or random walks or expanding-ring Time-To-Live (TTL) search etc. on the graph to query content stored by overlay peers. Each visited peer will evaluate the query locally on its own content, and will support complex queries. This is inefficient because queries for content that are not widely replicated must be sent to a large fraction of peers and there is no coupling between topology and data items' location. In this section, we shall survey and compare some of the more seminal Unstructured P2P overlay networks: Freenet [I. Clarke et al., 1999], Gnutella[19], FastTrack[20]/KaZaA[21], BitTorrent[22], Overnet/eDonkey2000[23].

3.1.3.1 Freenet

Freenet is an adaptive P2P network of peers that make query to store and retrieve data items, which are identified by location-independent keys. This is an example of loosely.

Structured decentralized P2P network with placement of files based on anonymity. Each peer maintains a dynamic routing table, which contains addresses of other peers and the data keys that they are holding. The key features of Freenet are the ability of maintaining locally a set of files in accordance to the maximum disk space allocated by the network operator, and providing security mechanisms against malicious peers. The basic model is that requests for keys are passed along from peer to peer through a chain of proxy requests in which each peer makes a local decision about the location to send the request next, similar to the Internet Protocol (IP) routing.

Freenet also enables users to share unused disk space, thus allowing a logical extension to their own local storage devices. The basic architecture consists of data items being identified by binary file keys obtained by applying the 160-bit SHA-1 hash function [ANSI, 1997]. The simplest type of file key is the Keyword-Signed Key (KSK), which is derived from a short descriptive text string chosen by the user, e.g. /music/Group1. The descriptive text string is used as the input to deterministically generate a public/private key pair, and the public half is then hashed to yield the data file key. The private half of the asymmetric key pair is used to sign the data file, thus, providing a minimal integrity check that a retrieved data file matches its data file key. The data file is also encrypted using the descriptive string itself as a key so as to perform an explicit lookup protocol to access the contents of their data-stores.

[19] http://groups.yahoo.com/group/the_gdf
[20] http://developer.berlios.de/projects/gift-fasttrack
[21] http://www.kazaa.com
[22] www.bittorrent.com
[23] http://en.wikipedia.org/wiki/EDonkey2000

However, nothing prevents two users from independently choosing the same descriptive string for different files. These problems are addressed by the Signed -Subspace Key (SSK), which enables personal namespaces. The public namespace key and the descriptive string are hashed independently, XORed together and hashed to yield the data file key. For retrieval, the user publishes the descriptive string together with the user subspace's public key. Storing data requires the private key, so that only the owner of a subspace can add files to it, and owners have the ability to manage their own namespaces. The third type of key in FreeNet is the Content-Hash Key (CHK), which is used for updating and splitting of contents. This key is derived from hashing the contents of the corresponding file, which gives every file a pseudo-unique data file key. Data files are also encrypted by a randomly generated encryption key.

For retrieval, the user publishes the content-hash key itself together with the decryption key. The decryption key is never stored with the data file but is only published with the data file key, so as to provide a measure of cover for operators. The CHK can also be used for splitting data files into multiple parts in order to optimize storage and bandwidth resources. This is done by inserting each part separately under a CHK and creating an indirect file or multiple levels of indirect files to point to the individual parts. The routing algorithm for storing and retrieving data is designed to adaptively adjust routes over time and to provide efficient performance while using local knowledge, since peers only have knowledge of their immediate neighbors. Thus, the routing performance is good for popular content. Each request is given a Hops-To-Live (HTL) limit, similar to the IP Time-To-Live (TTL) which is decremented at each peer to prevent infinite chains.

Each request is also assigned a pseudo-unique random identifier so that peers can avoid loops by rejecting requests they have seen before. If this happens, the preceding peer chooses a different peer to forward to. This process continues until the request either is satisfied or has exceeded its HTL limit. The success or failure signal (message) is returned back up the chain to the sending peer. Joining the network will rely on discovering the address of one or more of the existing peers through out-of-band means, and no peer is privileged over any other peer, so no hierarchy or centralized point of failure can exist. This intuitive resilience and decentralization enhances the performance and scalability, thus, giving a constant routing state while peers join and leave the overlay. In addition, as described in [I. Clarke et al., 1999], Freenet uses its data-store to increase system performance. When an object is returned (forwarded) after a successful retrieval (insertion), the peer caches the object in its data-store, and passes the object to the upstream (downstream) requester which then creates a new entry in its routing table associating the object source with the requested key. So, when a new object arrives from either a new insert or a successful request, this would cause the data-store to exceed the

designated size and Least Recently Used (LRU) objects are ejected until there is space. LRU policy is also applied to the routing table entries when the table is full.

Figure 3.7 depicts a typical sequence of request messages. The user initiates a data request at peer A, which forwards the request to peer B, and then forwards it to peer C. Peer C is unable to contact any other peer and returns a backtracking failed request message to peer B. Peer B tries its second choice, peer E, which forwards the request to peer F, which then delivers it to peer B. Peer B detects the loop and returns a backtracking failure message. Peer F is unable to contact any other peer and backtracks one step further back to peer E. Peer E forwards the request to its second choice, peer D, which has the data. The data is returned from peer D, via peers E, B and A. The data is cached in peers E, B and A, therefore, a routing short-cut for the next similar queries is created. This example shows that the overlay suffers from security problems such as man-in-the-middle and Trojan attacks, and the failure of peers will not cause network-wide failure, because of its lack of centralized structure. This gives good reliability and fault resiliency.

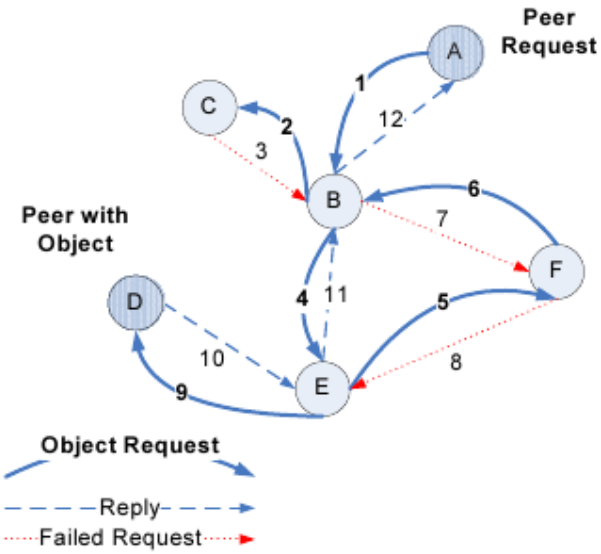


Figure 3.7 - Request sequence in Freenet

3.1.3.2 Gnutella

Gnutella is a decentralized protocol for distributed search on a flat topology of peers (servents). Gnutella is widely used and there has been a large amount of work on improving Gnutella [P. Ganesan et al., 2003], [Q. Lv et al., 2002], [Y. Chawathe et al., 2003]. Although the Gnutella protocol supports a traditional client/centralized server search paradigm, Gnutella's distinction is its peer-to-peer, decentralized model for document location and retrieval, as shown in Figure 3.8. In this model, every peer is a server or client. This system is neither a centralized directory nor does it possess any precise control over the network topology or file placement.

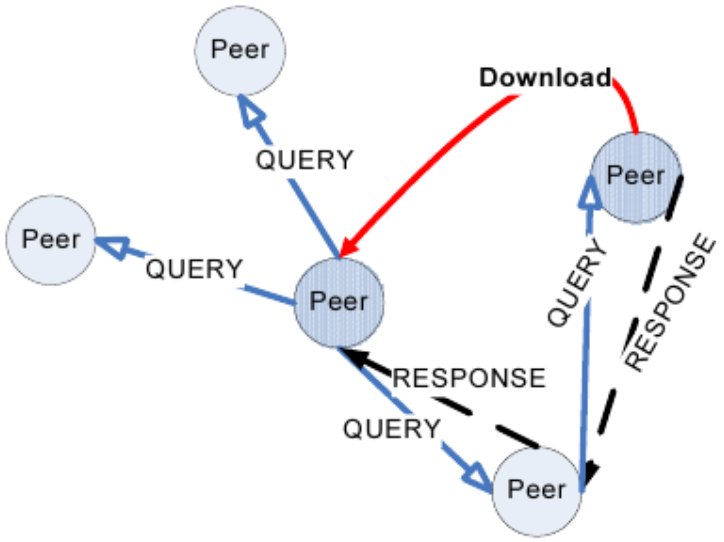


Figure 3.8 - Gnutella's decentralized architecture

The network is formed by peers joining the network following some loose rules. The resultant topology has certain properties, but the placement of data items is not based on any knowledge of the topology, as in the Structured P2P designs. To locate a data item, a peer queries its neighbors, and the most typical query method is flooding. The lookup query protocol is flooded to all neighbors within a certain radius. Such design is extremely resilient to peers entering and leaving the system.

However, the current search mechanisms are not scalable and generate unexpected loads on the network. The so-called Gnutella servents (peers) perform tasks normally associated with both clients and servers. They provide client-side interfaces through which users can issue queries and view search results while, at the same time, they also accept queries from other servents, check for matches against their local data set, and respond with applicable results. These peers are responsible for managing the background traffic that spreads the information used to maintain network integrity.

Due to the distributed nature, a network of servents that implement the Gnutella protocol is highly fault-tolerant, as operation of the network will not be interrupted if a subset of servents goes offline. To join the system, a new servent (peer) initially connects to one of several known hosts that are almost always available, e.g. list of peers available from http://gnutellahosts.com. Once connected to the network, peers send messages to interact with each other. These messages are broadcasted, (i.e. sent to all peers with which the sender has open TCP connections), or simply back-propagated, (i.e. sent on a specific connection on the reverse of the path taken by an initial, broadcasted message). First, each message has a randomly generated identifier. Second, each peer keeps a short memory of the recently routed messages, used to prevent re-broadcasting and to implement back-

propagation. Third, messages are flagged with TTL and "hops passed" fields. The messages that are allowed in the network are:

- Group Membership (PING and PONG) Messages. A peer joining the network initiates a broadcasted PING message to announce its presence. The PING message is then forwarded to its neighbors and initiates a back-propagated PONG message, which contains information about the peer such as the IP address, number and size of the data items.
- Search (QUERY and QUERY RESPONSE) Messages. QUERY contains a user specified search string that each receiving peer matches against locally stored file names and it is broadcasted. QUERY RESPONSEs are back-propagated replies to QUERY messages and include information necessary to download a file.
- File Transfer (GET and PUSH) Messages. File downloads are done directly between two peers using these types of messages.

Therefore, to become a member of the network, a servent (peer) has to open one or many connections with other peers that are already in the network. In order to cope with the unreliability, under such a such a dynamic network environment, after joining the network a peer periodically PINGs its neighbors to discover other participating peers. Peers decide where to connect in the network based only on local information. Thus, the entire application-level network has servents as its peers and open TCP connections as its links, forming a dynamic, self-organizing network of independent entities.

The latest version of Gnutella uses the notion of super-peers or ultra-peers (peers with better bandwidth connectivity) to help improve the routing performance of the network. However, it is still limited by the flooding mechanism used for communications across ultra-peers. Moreover, the ultra-peer approach makes a binary decision about a peer's capacity (ultra-peer or not) and to our knowledge, it has no mechanism to dynamically adapt the ultra-peer-client topologies as the system evolves. Ultra-peers perform query processing on behalf of their leaf peers. When a peer joins the network as a leaf, it selects a number of ultra-peers, and then it publishes its file list to those ultra-peers.

A query for a leaf peer is sent to an ultra-peer which floods the query to its ultra-peer neighbors up to a limited number of hops. Dynamic querying is a search technique whereby queries that return fewer results are re-flooded deeper into the network. [S. Saroiu et al., 2002] examines the bandwidth, latency, availability and file sharing patterns of the peers in Gnutella and Napster, and highlighted the existence of significant heterogeneity in both systems. [B. Krishnamurthy et al., 2001] proposes a cluster-based architecture for P2P systems (CAP), which uses a network-aware clustering technique (based on a central clustering server) to group peers into clusters. Each cluster has one or more delegate peers that act as directory servers for objects stored at peers within the same cluster. [Y. Chawathe et al., 2003] proposes a model called Gia, by modifying

Gnutella's algorithms to include flow control, dynamic topology adaptation, one-hop replication and careful attention to peer heterogeneity. The simulation results suggest that these modifications provide three to five orders of magnitude improvement in the total capacity of the system while retaining significant robustness to failures. Thus, making a few simple changes to Gnutella's search operations would result in dramatic improvements in its scalability.

3.1.3.3 FastTrack/KaZaA

FastTrack P2P is a decentralized file-sharing system that supports meta-data searching. Peers form a structured overlay of super-peers architecture to make search more efficient, as shown in Figure 3.9. Super-peers are peers with high bandwidth, disk space and processing power and have volunteered to get elected to facilitate search by caching the meta-data. The ordinary peers transmit the meta-data of the data files they are sharing to the super-peers. All the queries are also forwarded to the super-peer. Then, Gnutella-typed broadcast based search is performed in a highly pruned overlay network of super-peers. The P2P system can exist, without any super-peer but this result in worse query latency. However, this approach still consumes bandwidth so as to maintain the index at the super-peers on behalf of the peers that are connected.

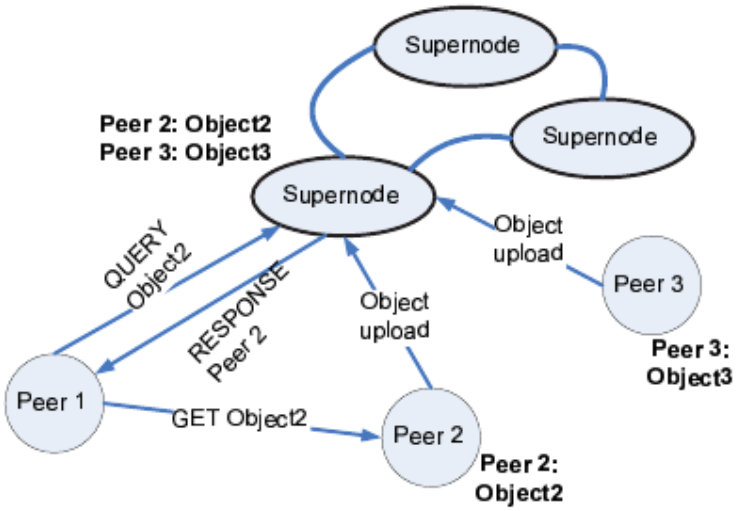


Figure 3.9 - FastTrack peers connected to super-peers

The super-peers still use a broadcast protocol for search and lookup queries are routed to peers and super-peers that have no relevant information for the query. Both KaZaA and Crokster are FastTrack applications. As mentioned, KaZaA is based on the proprietary FastTrack protocol which uses specially designated super-peers that have higher bandwidth connectivity. Pointers to each peer's data are stored on an associated

super-peer, and all queries are routed to the super-peers. Although this approach seems to offer better scaling properties than Gnutella, its design has not been analyzed. There have been proposals to incorporate this approach into the Gnutella network. The KaZaA peer to peer file sharing network client supports a similar behavior, allowing powerful peers to opt-out of network support roles that consume CPU and bandwidth.

KaZaA file transfer traffic consists of unencrypted HTTP transfers; all transfers include KaZaA-specific HTTP headers (e.g., X-KaZaA-IP). These headers make it simple to distinguish between KaZaA activity and other HTTP activity. A power-law topology, commonly found in many practical networks such as WWW [A. Barabasi et al., 2000], [R. Albert et al., 1999], has the property that a small proportion of peers have a high out-degree (i.e. have many connections to other peers), while the vast majority of peers have a low out-degree (i.e. have connections to few peers). Formally, the frequency f_d of peers with out-degree d exhibits a power-law relationship of the form $f_d \propto d^a$, a <0. This is the Zipf property with Zipf distributions looking linear when plotted on a log-log scale. [M. Faloutsos et al., 1999] has found that Internet routing topologies follow this power-law relationship with a ≈ −2. However, [S. Saroiu et al., 2002], [K. P. Gummadi et al., 2003] observes that the KaZaA measured popularity of the file-sharing workload does not follow a Zipf distribution.

The popularity of the most requested objects (mostly large, immutable video and audio objects) is limited since clients typically fetch objects at most once, unlike the Web. Thus the popularity of KaZaA's objects tends to be short-lived, and popular objects tend to be recently born. There is also significant locality in the KaZaA workload, which means that there is substantial opportunity for caching to reduce wide-area bandwidth consumption.

3.1.3.4 BitTorrent

BitTorrent is a centralized P2P system that uses a central location to manage users' downloads. This file distribution network uses tit-for-tat (peer responds with the same action that its other collaborating peer performed previously) as a method of seeking. The protocol is designed to discourage free-riders, by having the peers choose other peers from which the data has been received. Peers with high upload speed will probably also be able to download with a high speed, thus achieving high bandwidth utilization. The download speed of a peer will be reduced if the upload speed has been limited.

This will also ensure that content will be spread among peers to improve reliability. The architecture consists of a central location which is a tracker that is connected to when you download a torrent file, which contains information about the file, its length, name, and hashing information, and URL of a tracker, as illustrated in Figure 3.10.

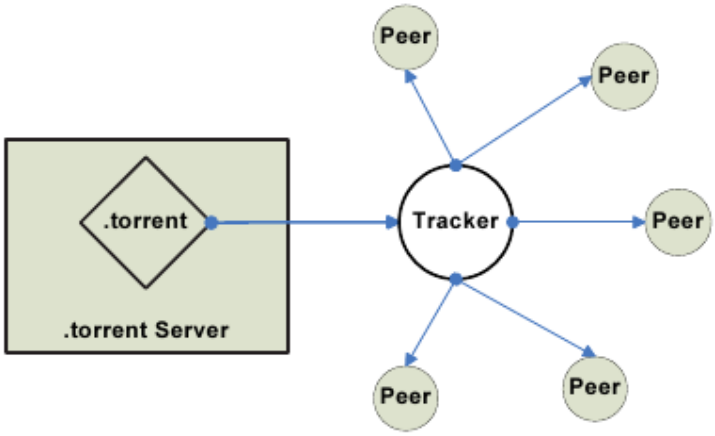


Figure 3.10 - BitTorrent architecture

The tracker keeps track of all the peers who have the file (both partially and completely) and lookup peers to connect with one another for downloading and uploading. The trackers use a simple protocol layered on top of HTTP in which a downloader sends information about the file it is downloading and the port number. The tracker responds with a random list of contact information about the peers which are downloading the same file. Downloaders then use this information to connect to each other. A downloader which has the complete file, known as a seed, must be started, and send out at least one complete copy of the original file.

BitTorrent cuts files into pieces of fixed size (256 Kbytes) so as to track the content of each peer. Each downloader peer announces to all of its peers the pieces it has, and uses SHA-1 to hash all the pieces that are included in the torrent file. When a peer finishes downloading a piece and checks that the hash matches, it announces that it has that piece to all of its peers. This is to verify data integrity. Peer connections are symmetrical. Messages sent in both directions look the same, and data can flow in either direction. When data is being transmitted, downloader peers keep several requests (for pieces of data) queued up at once in order to get good TCP performance. This is known as pipelining. Requests which cannot be written out to the TCP buffer immediately are queued up in memory rather than kept in an application-level network buffer, so they can all be thrown out when a choke happens.

Choking is a temporary refusal to upload; downloading can still happen and the connection does not need to be renegotiated when choking stops. Choking is done for several reasons. TCP congestion control behaves very poorly when sending over many connections at once. Additionally, choking lets each peer use a tit-for-tat-like algorithm to ensure that they get a consistent download rate. There are several criteria that a good choking algorithm should meet. It should cap the number of simultaneous uploads for

good TCP performance. It should avoid choking and unchoking quickly, known as fibrillation. It should reciprocate service access to peers who let it download.

Finally, it should try out unused connections once in a while to find out if they might be better than the currently used ones, known as optimistic unchoking. The currently deployed BitTorrent choking algorithm avoids fibrillation by only changing the peer that is choked once every ten seconds. It does reciprocation and the number of uploads are capped by unchoking the four peers with the best download rates. Peers which have a better upload rate but are not interested to get unchoked and if they become interested, the worst uploader gets choked. If a downloader has a complete file, it uses its upload rate rather than its download rate to decide which to unchoke. For optimistic unchoking, at any one time there is a single peer which is unchoked regardless of its upload rate. If this peer is interested, it counts as one of the four allowed downloaders. Peers which are optimistically unchoked rotate every 30 seconds.

3.1.3.5 Overnet/eDonkey

Overnet/eDonkey is a hybrid two-layer P2P information storage network composed of client and server, which are used to publish and retrieve small pieces of data by creating a file-sharing network. This architecture provides features such as concurrent download of a file from multiple peers, detection of file corruption using hashing, partial sharing of files during downloading and expressive querying methods for file search. To join the network, the peer (client) needs to know the IP address and port of another peer (server) in the network. It then bootstraps from the other peer. The clients connect to a server and register the object files that they are sharing by providing the meta-data describing the object files.

After registration, the clients can either search by querying the meta-data or request a particular file through its unique network identifier, thus providing guarantee service to locate popular objects. Servers provide the locations of object files when requested by clients, so that clients can download the files directly from the indicated locations.

3.1.4 Vis-à-vis of Structured-Unstructured Peer-to-Peer approaches

The Unstructured P2P centralized overlay model was first popularized by Napster. This model requires some managed infrastructure (the directory server) and is subjected to some scalability limits. A flooding-request model for decentralized P2P overlay systems such as Gnutella, according to which each peer keeps a user-driven neighbor table to locate data objects are quite effective to locate popular data objects, thanks to the power-law property of user-driven characteristics. However, it can lead to excessive network

bandwidth consumption, and remote or unpopular data objects may not be found due to the limit of lookup horizon, typically imposed by TTL.

The argument is that DHT-based systems while more efficient at many tasks and having strong theoretical fundamentals to guarantee a key to be found if it exists, are not well-suited for mass-market file sharing. They do not capture the semantic object relationships between its name and its content or metadata. In particular, DHT-based ability to find exceedingly rare objects is not required in a mass-market file sharing environment, and their ability to efficiently implement keyword search is still not proven. In addition, they use precise placement algorithms and specific routing protocols to make searching efficient. However, these Structured P2P overlay systems have not been widely deployed, and their ability to handle unreliable peers has not been tested. Thus, in the research community, efforts are being made in improving the lookup properties of Unstructured P2P overlays to include flow control, dynamic geometric topology adaptation, one-hop replication, peer heterogeneity etc.

The Freenet, like Chord, does not assign responsibility for data to specific peers, and its lookups take the form of searches for cached copies. This prevents it from guaranteeing retrieval of existing data or from providing low bounds on retrieval costs. But Freenet provides anonymity and it introduces a novel indexing scheme where files are identified by content-hash keys and by secured signed-subspace keys to ensure that only one object owner can write to a file and anyone can read it. P2P overlay designs using DHTs share similar characteristics as Freenet — an exact query yields an exact response. This is not surprising since Freenet uses a hash function to generate keys. Recent research in [A. Goel et al., 2004] shows that, that changing Freenet's routing table cache replacement scheme from LRU to enforcing clustering in the key space, can significantly improve performance. This idea is based on the intuition from the small-world models [J. Kleinberg et al., 2000] and theoretical results by [J. Kleinberg et al., 2000].

Version 0.6 of the Gnutella protocol adopted the concept of ultra-peers which are high capacity peers that act as proxies for lower capacity peers. One of the main enhancements is the Query Routing Protocol (QRP), which allows the leaf peers to forward index of object name keywords to its ultra-peers. This allows the ultra-peers to have their leaves receive lookup queries when they have a match, and subsequently, it reduces the lookup query traffic at the leaves.

A shortcoming of QRP is that the lookup query propagation is independent of the popularity of the objects. The Dynamic Query Protocol addressed this by letting the leaf peers send single queries to high-degree ultra-peers which adjust the lookup queries' TTL bounds in accordance to its number of received lookup query results.

As described in the Gnutella section, [Y. Chawathe et al., 2003] improves the Gnutella design using their Gia system, by incorporating adaptation algorithm so that peers are

attached to high-degree peers and providing a receiver-based token flow control for sending lookup queries to neighbors. Instead of flooding, they make use of random walk search algorithm and also the system keep pointers to objects in neighboring peers. However, in a Gnutella UDP extension, it has been proposed that Unstructured P2P overlay like Gnutella can be built on top of Structured P2P overlay to help reduce the lookup queries overhead and overlay maintenance traffic. They used the collapse point lookup query rate (defined as the per node query rate at which the successful query rate falls below 90%) and the average hop counts prior to collapse. However, the comparison was done in static network scenario with the older Gnutella and not the enhanced Gnutella version.

BitTorrent, a second generation P2P overlay system, achieves higher level of robustness and resource utilization based on its incentives cooperation techniq ue for file distribution. The longest and most comprehensive measurement study of BitTorrent P2P system [J. A. Pouwelse et al., 2004] provides a more insight by comparing a detailed measurement study of BitTorrent with other popular P2P file-sharing systems, such as FastTrack/KaZaA, Gnutella, Overnet/eDonkey, and DirectConnect, based on five characteristics:

- Popularity - Total number of users participating over a certain period of time.
- Availability - System availability depending on contributed resources.
- Download Performance - Contrast between size of data and the time required for download.
- Content Lifetime - Time period when data is injected into the system till no peers is willing to share the data anymore.
- Pollution Level - Fraction of corrupted content spread throughout the system.

FastTrack/KaZaA has the largest file sharing community, with Overnet/eDonkey and BitTorrent are gaining popularity. The popularity of BitTorrent system is influenced by the availability of the central components in terms of its number of downloads and technical faults in the system. Availability has a significant influence on popularity. FastTrack/KaZaA being more architecturally advanced, achieve significant availability because of its Super-Peers that allow the network to scale very well, by creating indexing. Gnutella and Overnet/eDonkey provide full and partial distribution of the responsibility for shared files respectively. The availability of content in BitTorrent is unpredictable and vulnerable to potential failures, due to its lack of decentralization.

BitTorrent is well-suited for download performance due to its advanced download distribution protocol. Overnet/eDonkey takes an opposite approach by offering powerful searching capabilities and queue-based scheduling of downloads, which can take longer waiting times. The lack of archive functionality in BitTorrent results in relatively short

content lifetimes. FastTrack/KaZaA which uses directory-level sharing policy allows data files to be located as long as the peer holding the data file stays connected.

FastTrack/KaZaA system does not limit the number of fake files in the overlay but it allows user to identify correct files based on hash-code verification. BitTorrent prevents fake files from floating in the system. The arising use of firewalls and NATs are growing problems for P2P systems because of the effect of reducing the download speed. This proposal [K. A. Skevik et al., 2004] tries to solve the firewall problems by designing a hybrid CDN structure with a P2P based streaming protocol in the access network based on an empirical study of BitTorrent which found the need for additional freeloader protection and the potential negative effect of firewall on download speeds. A fluid model for BitTorrent-like networks was proposed [D. Qiu et al., 2004] to capture the behavior of the system when the arrival rate is small, and to study the steady-state network performance. The study also provided expressions for the various parameters, such as average number of seeds, the average number of downloaders and the average downloading time, and proved that Nash equilibrium exists for each peer that chooses its uploading bandwidth to be equal to the actual uploading bandwidth.

It is also interesting to note that most of these Unstructured P2P networks (such as KaZaA and Gnutella) are not pure power-law networks with Zipf distribution properties; for example, analysis in [M. A. Jovanovic et al., 2001] shows that Gnutella networks have topologies that are power-law random graphs, and later measurement shows that there are too few peers with a low number of connectivity. This may be attributed to the behaviors of the users of these P2P networks. Research on power-law networks [A. Barabasi et al., 2000], [M. Faloutsos et al., 1999], [A. Broder et al., 2000], [A. Barabasi et al., 1999] shows that networks as diverse as the Internet, organize themselves so that most peers have few links while a small number of peers have a large number of links. The interesting paper by [L. Adamic et al., 2001] studies random-walk search strategies in power-law networks, and discovers that by changing walkers to seek out high degree peers, the search performance can be optimized greatly.

Several search techniques for Unstructured P2P networks are discussed in [B. Yang et al., 2002]: iterative deepening, directed BFS and local indices. Networks with power-law organizational structure, display an unexpected degree of robustness [R. Albert et al., 2000], i.e. the ability of the peers to communicate unaffectedly even by extremely high failure rates. But these networks are prone to attacks.

Thus, Unstructured P2P networks reduce the network dependence on a small number of highly connected, easy to attack peers. Instead of using DHT as building blocks in distributed applications, SkipNet [A. Harvey et al., 2003] is a new overlay based on Skip Lists that organizes peers and data primarily by their sorted string names, as Skip Lists do, rather than by hashes of those names. In this way, Skip Net supports useful locality

properties as well as the usual DHT functionality. In addition, some recent research, e.g. [B. T. Loo et al., 2004] proposes the design of a hybrid model by using Structured DHT techniques to locate rare object items, and Unstructured flooding to locate highly replicated contents.

All of the security issues discussed in the Structured P2P overlay networks section apply to Unstructured P2P overlay networks. It is worthwhile highlighting the work of [S. Bellovin et al., 2001] in the field of overcoming the difficult to limit Napster and Gnutella use via firewalls and how information can be leaked through search queries in the overlay network. The work highlighted concerns over Gnutella's push feature, intended to work around firewalls, which might be useful for distributed denial of service attacks. Napster's centralized architecture might be more secure towards such attacks due to a centralized trusted server.

Algorithm Taxonomy	Unstructured P2P Overlay Network Comparisons				
	Freenet	Gnutella	FastTrack/KaZaA	BitTorrent	Overnet/eDonkey2000
Decentralization	Loosely DHT functionality.	Topology is flat with equal peers.	No explicit central server. Peers are connected to their Super-Peers.	Centralized model with a Tracker keeping track of peers.	Hybrid two-layer network composed of clients and servers.
Architecture	Keywords and descriptive text strings to identify data objects.	Flat and Ad-Hoc network of servents (peers). Flooding request and peers download directly.	Two-level hierarchical network of Super-Peers and peers.	Peers request information from a central Tracker.	Servers provide the locations of files to requesting clients for download directly.
Lookup Protocol	Keys, Descriptive Text String search from peer to peer.	Query flooding.	Super-Peers.	Tracker.	Client-Server peers.
System Parameters	None	None	None	*.torrent* file.	None
Routing Performance	Guarantee to locate data using *Key* search until the requests exceeded the Hops-To-Live (HTL) limits.	No guarantee to locate data; Improvements made in adapting Ultrapeer client topologies; Good performance for popular content.	Some degree of guarantee to locate data, since queries are routed to the Super-Peers which has a better scaling; Good performance for popular content.	Guarantee to locate data and guarantee performance for popular content.	Guarantee to locate data and guarantee performance for popular content.
Routing State	Constant	Constant	Constant	Constant but choking (temporary refusal to upload) may occur.	Constant
Peers Join/Leave	Constant	Constant	Constant	Constant	Constant with bootstrapping from other peer and connect to server to register files being shared.
Security	Low; Suffers from man-in-middle and Trojan attacks.	Low; Threats: flooding, malicious content, virus spreading, attack on queries, and denial of service attacks.	Low; Threats: flooding, malicious or fake content, viruses, etc. Spywares monitor the activities of peers in the background.	Moderate; Centralized Tracker manage file transfer and allows more control which makes it much harder faking IP addresses, port numbers, etc.	Moderate; Similar threats as the FastTrack and Bit-Torrent.
Reliability/Fault Resiliency	No hierarchy or central point of failure exists.	Degradation of the performance; Peer receive multiple copies of replies from peers that have the data; Requester peers can retry.	The ordinary peers are reassigned to other Super-Peers.	The Tracker keeps track of the peers and availability of the pieces of files; Avoid Choking by fibrillation by changing the peer that is choked once every ten seconds.	Reconnecting to another server; Will not receive multiple replies from peers with available data.

Table 3-2 Comparison of Unstructured P2P approaches

3.2 Mobile Peer-to-Peer Approaches

As mentioned in Chapter 1, client-server networks are being transformed to distributed peer-to-peer networks. Lessons learned from fixed networks have been applied in cellular networks. But the special requirements of mobile devices and networks necessitate the elaboration and the adoption of different solutions in order to fulfill the expectations which arise with the use of mobile peer-to-peer technology.

Many researchers are currently proposing and developing new P2P schemes for mobile environments. Recent advances in mobile devices and wireless communications have enabled the development of mobile P2P applications for mobile phones. These new mobile P2P systems seem promising in a new domain of applications based on physical location and context together with the possibility of using a wide variety of wireless radio access technologies (see Figure 3.11).

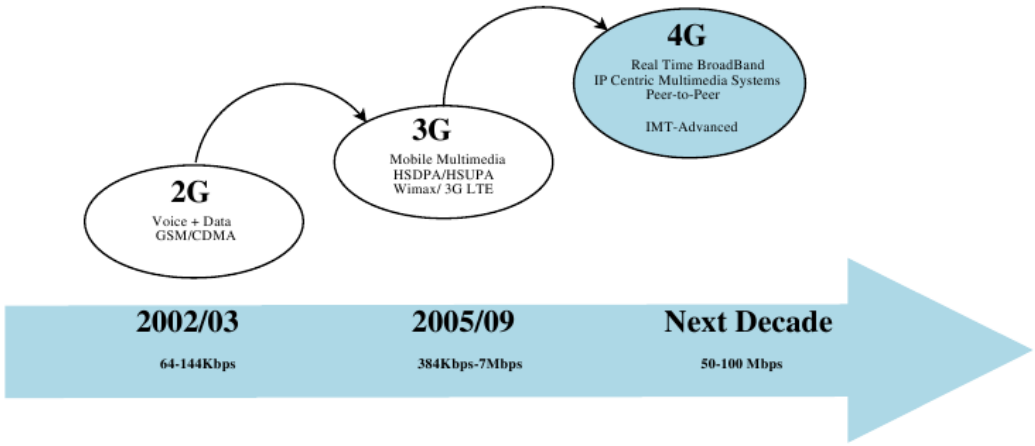


Figure 3.11 - Wireless Radio access Technology Evolution

In this context, new applications, different from traditional file sharing applications are expected to appear. For example, in all cell environments radio resources are very limited and multicast architectures are not recommendable. In this sense peer-to-peer video streaming applications appear as a real alternative to services based on multicast.

Peer-to-peer systems allow users to distribute their own content by means of the Internet or between their friends without using costly and centralized servers with high bandwidth requirements.

Peer-to-peer content distribution is a very interesting paradigm in cellular environments because in p2p networks bandwidth available to the content server is regulated according to the demand. This feature ensures a wide range of applications for peer-to-peer systems in mobile environments. For the development of these applications new techniques need to be adopted in order to deal with the limitations present in mobile

devices and in cellular networks. Current P2P applications and architectures are mainly designed to work in fixed and wired networks as analyzed in Chapter 2. Recently developed wireless communication technologies and the new available capacities presented in mobile devices have allowed novel peer to peer paradigms to emerge which focus mainly on mobile devices such as PDA and mobile phones.

These paradigms involve new challenges due to constraints present in mobile devices and wireless networks such as limited memory/processing power, network accessibility issues such as low bit rates, high latency, packet losses, temporal disconnections etc. and mobility issues such as roaming between different radio access technologies. Regarding mobility issues, there are some standards that are trying to resolve this problem, such as GAN (Generic Access Network), known before as UMA (unlicensed mobile access) [3GPP, 2007] or MIH (Multiple Independent Handover) IEEE 802.21 [M. G. Williams et al., 2005].

Additional problems are Operator-Control problems. Operators want to control traffic and services offered in their networks. In the available literature, it has been proposed to use a hybrid architecture where super-peers are located in the core network (which is the logical location for sharing resources in a mobile network) under the control of the operator [Q. Hofstatter et al., 2008]. Finally, other existing issues are related with firewall and NATs existing in cellular networks.

3.2.1 Proposed Protocols & Architectures

The most prominent proposed solutions in order to bridge wired and wireless P2P networks is the idea of using proxies as gateways between both domains. This model can be extended to let mobile devices act as peers to other mobiles in the same network, firstly allowing direct connections, and then enabling indirect connections through several mobile nodes.

Following in this line, another work investigates using the already existing eDonkey architecture to let mobile users directly participate in the network. The index server would be hosted by the mobile operator. Also, the protocol would be extended with enhanced signaling information about the mobile network domain and some infrastructure could be added to deal with the needs of the mobile users. These new elements are caching peers, servers that act as regular peers but are under the control of the mobile operator. They would provide popular content, eliminating the need to connect to nodes outside the mobile network; crawlers, nodes that bring information of the wired network to index server and proxies that act as bridges between mobile and regular peers.

However, the main focus of research in overlay networks is in the adaptation of already existing structured P2P architectures to the peculiarities of a mobile environment, namely frequent disconnections, node mobility issues and scarce bandwidth and resources. For example, DynaMO [R. Winter et al., 2004] is a modified Pastry [A. Rowstron

et al., 2001] system that exploits physical proximity between nodes, trying to make the overlay network similar to the underlying ad-hoc network. To this end, DynaMO adds two new mechanisms to form clusters of related nodes, in which neighbors in the overlay network are, probably, physically near nodes. Another example of this kind of architecture is MobiGrid [A. Datta, 2003]. This system, like the previous one, is based on an already existing structured P2P network targeted at fixed networks. Specifically, P-Grid [K. Aberer et al., 2003] is used as a base, adding mechanisms to provide replication, security and self-organization in ad-hoc networks.

The JXTA[24] project defines a set of open protocols that should allow devices connected to the network, ranking from cell phone and wireless PDAs to PCs and servers, to communicate and collaborate in a P2P manner.

Java version of these protocols are the most widely known, but there are projects in other languages and areas such as C++[25] or Symbian OS[26]. The Java version of these protocols for embedded devices (JXME[27]) such as mobile phones is not completely functional due to limitations in the MIDP profile present in these kinds of terminals.

Another example of protocol is the Mobile Peer-to-Peer Protocol (MPP) [R. Schollmeier et al., 2003]. MPP has been developed for P2P networking in mobile ad hoc environments. MPP implements an efficient signaling messages mechanism and cross layer communication between the network layer and application layer. Current results are based on NS-2 simulations and show that the MPP-protocol stack copes with node failures and link breaks, typical issues associated with wireless networks.

3.2.2 Implementations

With the prior constraints in mind there are many available projects and solutions focused on offering and developing mobile P2P applications in a straightforward manner. JXME (a limited version of JXTA) and Microsoft P2P framework are the best known solutions for the development of mobile P2P applications.

Mobile Peer to Peer Content Sharing Application [M. Matuszewski et al., 2006] is an innovative proposal of architecture of mobile peer-to-peer content sharing services in cellular networks developed by the Nokia Research Center and Helsinki University of Technology. This approach uses the SIP protocol as a basis for the deployment of mobile P2P services (Figure 3.12). The implementation consists of a peer-to-peer client application in the mobile phone and an application server in the network. The mobile peer-to-peer client was implemented on the Nokia Series 60 (Symbian platform). This solution presents a hybrid architecture with peers and super-peers.

[24] http://www.jxta.org
[25] http://jxta-c.jxta.org
[26] https://symbianjxta.dev.java.net
[27] http://jxme.jxta.org

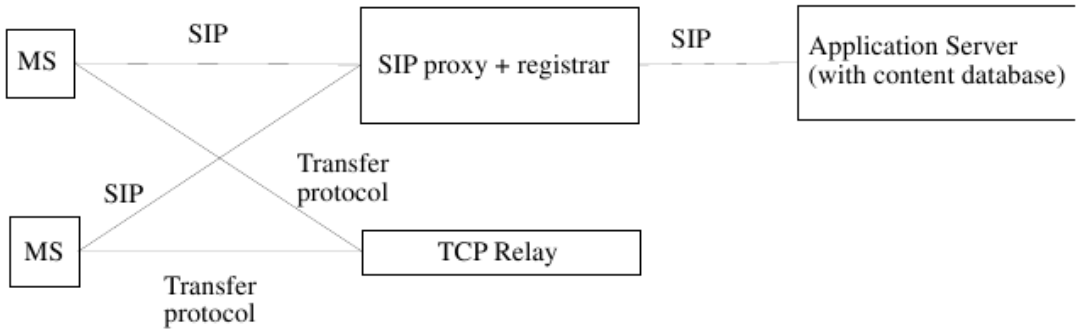


Figure 3.12 - Mobile Peer-to-Peer Content Sharing Prototype

The Generic Engine [T. Hakkarainen et al., 2005] uses mechanisms introduced in SIP in order to make a terminal globally discoverable. Collaborative networks created with this engine maintain a circular topology. Entering a network relies on the invite mechanism provided by SIP, in which the communication initiator must provide a remote node SIP identifier. Symbian has been chosen as the OS platform. This engine facilitates quick and robust data modeling by providing a meta- model-based generative mechanism.

JMobipeer [M. Bisignano et al., 2005] is a framework designed to work on J2ME enabled mobile devices on mobile ad-hoc networks (Figure 3.13). This framework uses a reactive routing algorithm. Although due to the modularity of its architecture, the routing algorithm can be replaced with any other algorithm. Interpretability with JXTA is supported, and the increased network load is the main cost to be accepted if this interpretability is to be maintained.

Proem [G. Kortuem et al., 2001] is a platform for the development of P2P collaborative applications in mobile ad-hoc networking environments. Proem provides a complete SDK, which includes a collection of Java interfaces and classes for rapid development of mobile peer-to-peer applications called peerlets. Proem also provides a runtime environment for the execution of peerlets. The Proem middleware consists of three main components: an application runtime environment, a set of middleware services and a protocol stack. Proem defines four protocols, one low level transport protocol and three higher-level protocols. Proem differs from previous platforms by focusing on the requirements of face-to-face applications.

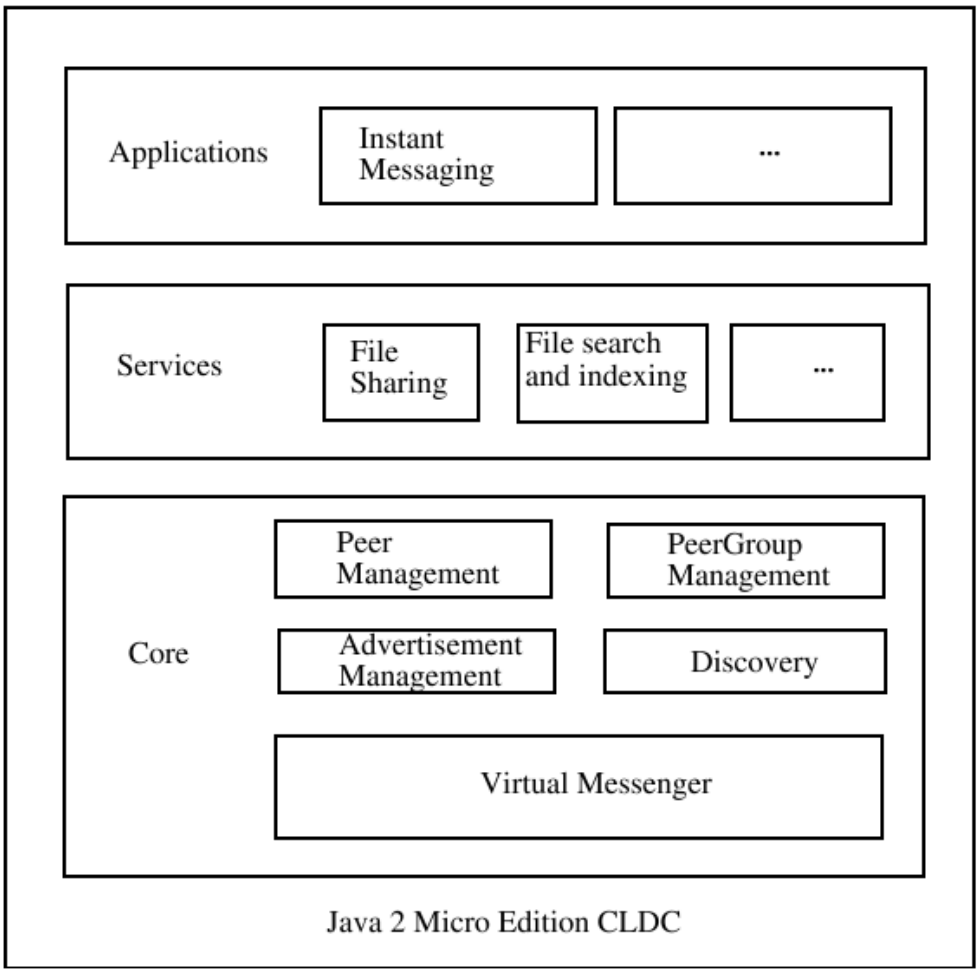


Figure 3.13 - JMobilePeer Architecture

Peer2Me [A. I. Wan et al., 2007] is an open source project developing a framework for mobile collaborative applications on mobile phones. Peer2Me enables developers to create collaborative applications for mobile phones using a network technology such as Bluetooth. The architecture and concepts of this framework are independent of the kind of PAN technology supported in the mobile device. The Peer2Me project was initiated to enable rapid development of proximity-based peer-to-peer applications for mobile devices on the Java 2 Micro Edition (J2ME) platform. Peer2Me is based on a hybrid peer-to-peer model.

Mobile Chedar [N. Kotilainen et al., 2005] is an extension to the Chedar peer-to-peer network allowing mobile devices to access the Chedar network and also to communicate with other Mobile Chedar peers. Chedar (CHEap Distributed ARchitecture) is a peer-to-peer middleware designed for peer-to-peer applications. In this project, Chedar has been extended to the mobile platform as Mobile Chedar. Mobile Chedar is implemented using Java 2 Micro Edition (J2ME) and uses Bluetooth as a transmission technology for

connecting to other peers. Current Bluetooth implementations have a restriction that nodes can be connected to only one piconet at a time. Therefore, the only topology available for constructing Bluetooth network is star-shaped. One device functions as a master and others as slaves (hybrid architecture).

Symella[28] is a Gnutella [J. Miller, 2004] file-sharing client for Symbian smartphones. It is capable of searching and downloading, but does not upload any data in its current release. It supports multi-threaded downloads which means that if multiple users have a particular file, then Symella can download the file from several locations simultaneously. Gnutella is a flooded request model. Each request from a peer is flooded to directly connected peers. This solution consumes a lot of bandwidth and consequently this mobile client does not support data uploading.

SymTorrent[29] is a complete BitTorrent [L. Guo et al., 2007] client for Symbian OS. It supports downloading multiple torrent files at the same time, is capable of both downloading and uploading and can save the status of unfinished torrents so that downloading can be resumed after restarting the application. BitTorrent organizes peers sharing the same file into a P2P network and focuses on fast and efficient replication to distribute the file. WizBit[30] is another BitTorrent client for mobile phones. It does not fully work yet and is only suitable for alpha release.

Finally, Bedd[31] is a commercial application which runs on Symbian Series 60 smartphones. Bedd, currently, uses GPRS and Bluetooth wireless technology. Bedd is an end user application which enables ad-hoc mobile communication between mobile phones. A classification of the current approaches is depicted at Figure 3.14.

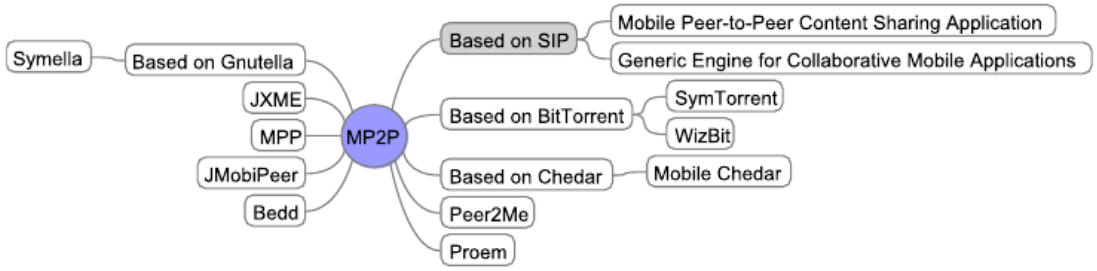


Figure 3.14 - Current Approaches

After the analysis carried out, we can conclude that there is a huge fragmentation in the mobile peer-to-peer field. Current approaches focus on operator-driven P2P applications while UbiChord focuses on pure Mesh Networking.

[28] http://symella.aut.bme.hu
[29] http://symtorrent.aut.bme.hu
[30] http://dave1010.googlepages.com/wizbit
[31] http://www.bedd.com

3.3 Mesh Routing Protocols

3.3.1 Proactive Routing Protocols

Routing is the exchange of information (in our case typical term “packets”) from one station of the network to another. The major goals of routing are to find and maintain routes between nodes in a dynamic topology with possibly unidirectional links, using minimum resources. A protocol is a set of standard-rules to exchange data between two devices. Routing protocols are classified in unicast routing protocols, multicast routing protocols and broadcast routing protocols.

Unicast forwarding means a one-to-one communication, i.e. one source transmits data packets to a single destination. This is the largest class of routing protocols found in ad hoc networks. Multicast routing protocols come into play when a node needs to send the same message, or stream of data, to multiple destinations. Broadcast is the basic mode of operation over a wireless channel; each message transmitted on a wireless channel is generally received by all neighbors located within one-hop from the sender.

The simplest implementation of the broadcast operation to all network nodes is by naive flooding, but this may cause the broadcast storm problem due to redundant re-broadcasting. There are several unicast protocols such as proactive, reactive and hybrid routing protocols (see Figure 3.15).

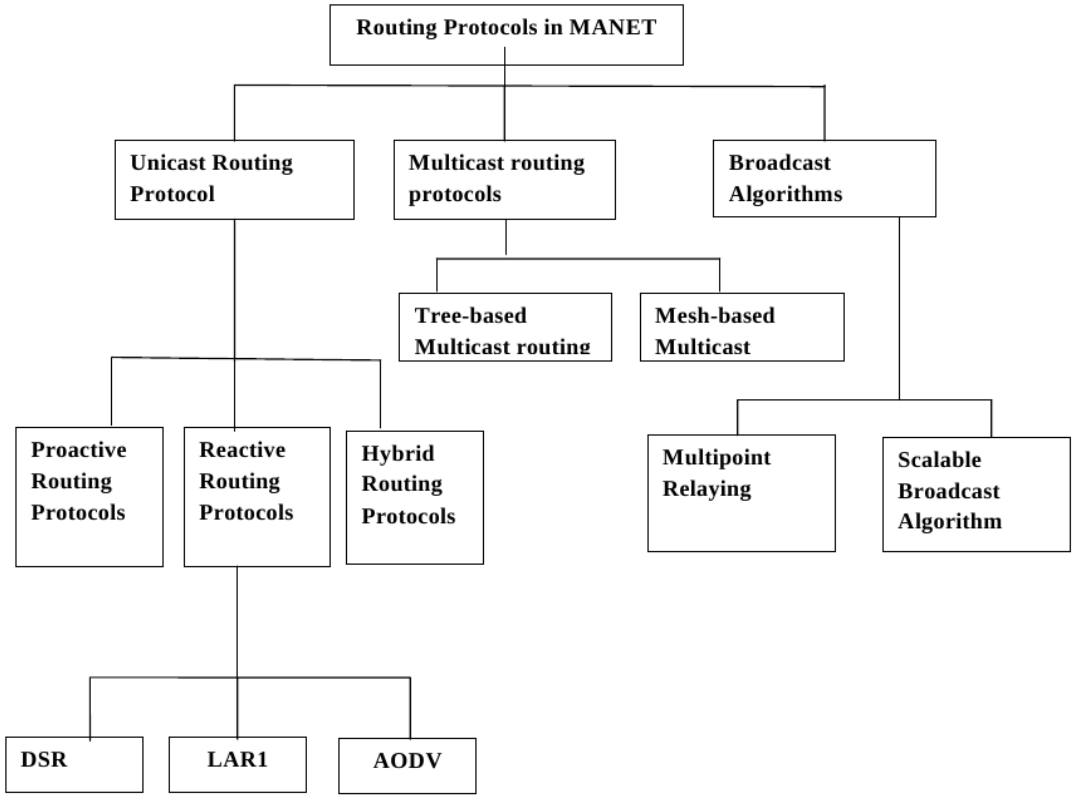


Figure 3.15 - Routing Protocols classification

Proactive Protocols keep track of routes for all destinations in the ad-hoc network and are also called Table-driven Protocols, as the routes can be assumed to exist in the form of tables.

The main advantage is that communications with arbitrary destinations experience minimal initial delay from the point of view of the application. The disadvantage of proactive protocols is that additional control traffic is needed to continually update stale route entries. Indicatives Proactive Routing Protocols are:

- AWDS (Ad-hoc Wireless Distribution Service) [32]
- CGSR (Cluster-head Gateway Switch Routing Protocol) [Ching-Chuan et al., 1997]
- DFR (Direction Forward Routing) [Gerla, 2006]
- DBF (Distributed Bellman-Ford Routing Protocol) [Awerbuch, B. et al., 1991]
- HSR (Hierarchical State Routing Protocol) [Pei et al., 1999]
- IARP (Intrazone Routing Protocol) [Zygmunt et al., 2002]

Reactive Protocols acquire routing information only when it is actually needed. The advantage is that due to the high uncertainty in the position of the nodes, the reactive protocols are much suited and perform better for ad-hoc networks.

The disadvantages of reactive protocols include high latency time in route finding and excessive flooding leading to network clogging. Indicative Reactive Routing Protocols are:

- Admission Control Enabled On Demand Routing (ACOR) [Noureddine et al., 2006]
- Associativity Based Routing (ABR) [Toh, 1997]
- AODV (Ad-hoc On-demand Distance Vector) [Perkins et al., 1991]
- DSR (Dynamic Source Routing) [D. B. Johnson et al., 1996]
- CHAMP (CacHing And Multi-Path Routing) [Valera et al., 2003]
- LAR1 (Location Aided Routing – Scheme 1) [Ko et al., 2000]

Hybrid routing are protocols in which the routing is initially established with some proactively prospected routes and then serves the demand from additionally activated nodes through reactive flooding. The disadvantages of hybrid protocols are that success depends on amount of nodes activated and reaction to traffic demand depends on gradient of traffic volume. Some of the most known Hybrid Routing Protocols are:

- HRPLS (Hybrid Routing Protocol for Large Scale Mobile Ad-hoc Networks with Mobile Backbone) [Pandey et al., 2006]
- HSLS (HAZY Sighted Link State Routing Protocol) [Santivanez et al., 2003]
- HWMP (Hybrid Wireless Mesh Protocol) [Bahr, 2007]
- ZRP (Zone Routing Protocol) [Haas, 1997]

[32] http://awds.berlios.de/about.html

It could be argued that proactive and hybrid methods are unsuitable for UbiChord, our developed platform for mobile p2p communications, since signaling cost would cause excessive flooding. Therefore, reactive routing should be adopted. In the next chapter, we focus on these protocols.

3.3.2 Reactive Routing Protocols

Reactive Routing Protocols, otherwise known as on demand routing protocols, take a lazy approach to routing which differs from proactive routing protocols by identifying and maintaining routes only when needed which results in reduced overhead. Routes are identified and maintained for nodes that require sending data to a known destination, this is typically done by invoking route discovery mechanisms to find path to the destination. The most prominent representative protocols in this category are DSR and AODV. A brief description of these protocols will be provided below .

DSR is an on demand routing protocol in which a sender determines the exact sequence of nodes through which a packet is propagated. The packet header contains a list of intermediate nodes for routing. Route cache is maintained by each node which caches the source route that it has learned. The major mechanisms of DSR are “Route Discovery” and “Route Maintenance” which work together for determining and maintaining routes to arbitrary destinations [D. B. Johnson et al., 1996]. It is designed to restrict the bandwidth consumed by control packets in ad hoc wireless networks by eliminating the periodic table-update messages required in the table-driven approach. A route is established by flooding “Route Request” packets in the network [Gergely et al., 2005].

DSR protocol has many advantages. A route is established only when it is required. It allows the sender to select and control routes there by reducing load. The other advantage includes loop-free routing in networks containing unidirectional links. However, the source route has to be included with each packet causing significant overheads. The other disadvantage includes aggressive use of caching and lack of any mechanism to detect freshness of routes which causes delay and throughput reduction. The route maintenance mechanism does not locally repair a broken link. The connection setup delay is higher than in table-driven protocols.

AODV is a reactive routing protocol which is basically a combination of DSR and DSDV algorithms. It uses the advantageous feature of both these algorithms. Dynamic, self-starting and multi-hop routing is allowed between participating mobile nodes. The demand routing mechanism of route discovery and route maintenance of DSR and the use of hop by hop routing sequencing number and periodic update packets of DSDV are both available in AODV. It employs destination sequence numbers to identify the most recent path. In AODV, the source node and the intermediate nodes store the next-hop information

corresponding to each flow for data packet transmission [Perkins et al., 1991], [N. Moghim et al., 2002], [Gergely et al., 2005].

The main advantages of AODV include its adaptability to highly dynamic networks and its reduced overhead. Other advantages include lower setup delay for connections and detection of latest route to the destination. As far as disadvantages are concerned, AODV requires periodic updates. The distinguishing feature is the use of a destination sequence number for each route entry. If the source sequence number is very old, it leads to inconsistent routes. Unnecessary bandwidth consumption occurs in response to periodic beaconing.

There are a lot of comparative studies that evaluate Mesh routing protocols. In one of these studies that emphasizes only at reactive routing protocols [M. Uma et al., 2009] the average end-to-end delay for three protocols are presented (see Figure 3.16).

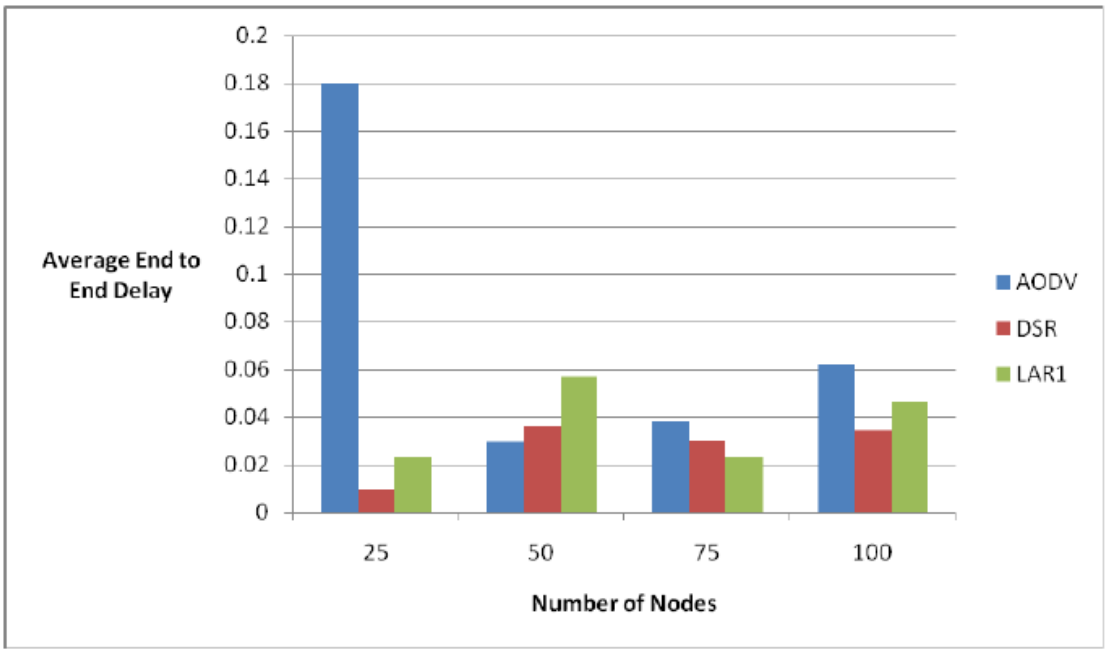


Figure 3.16 - Routing Protocols end-to-end delay

In the frames of UbiChord, the DSR protocol is adopted. The reason for this choice is first of all the simplicity of the protocol's mechanisms and its performance regarding channel setup and end-to-end delay.

3.4 Positioning of our approach to the SOTA

In the frames of this Thesis a protocol and a reference implementation of a Distributed Hash Table that is able to operate on Mesh environments is introduced. The goal of this structure is to facilitate the creation of emerging peer-to-peer mesh applications. In this chapter we examined two different approaches regarding peer-to-peer applications; the structured and the unstructured approach. The comparative analysis indicated that the unstructured approaches are more loosly-coupled since they do not impose signaling for overlay creation and maintenance. However, none of the existing unstructured techniques are operational without any centralized "coordination" server (e.g. super-peer etc.). Consequently, the techniques adopted in our approach are based on structured approaches.

As already explained on Chapter 2, existing structured approaches (DHTs) can only be used in order to create peer-to-peer applications in fixed networks. In the current Chapter we presented various representative DHTs such as Chord, Pastry etc. All of them suffer from the same barrier. The effort that has been made in order to re-use existing DHTs for the creation of peer-to-peer applications has been thoroughly analyzed on Chapter 3.2. However, these techniques rely on maintaining one server-side overlay network at the Mobile Network Operator's side through some proxies. Our approach follows a completely different policy. The maintenance of the overlay in UbiChord is achieved with the utilization of self-healing and self-adapting gossiping techniques which allows operation of a DHT without the involvement of the MNO's infrastructure. UbiChord's approach is agnostic to a specific DHT implementation. However, for implementation purposes we selected Chord.

Finally, the traditional communication pattern (i.e. sockets) for signaling and exchange of data objects at the application level is in-adequate for our approach since no static routing infrastructure can be considered as granted. Therefore, a MANET-like routing scheme has to be incorporated. In the frames of this Chapter we examined the basic MANET routing protocols. UbiChord will integrate a reactive routing scheme and specifically Data Source Routing (DSR). This choice is made based on the linear behavior of DSR regarding the end-to-end delay that is introduced as an overhead.

4. Proposed Layered Approach

4.1 Generic Principles

The proposed layered-approach aims at the provision of a generic framework that will facilitate the design and development of autonomic and decentralized services in Mesh networks (see Figure 4.1). The introduction of the different layers of the proposed approach is necessary due to the need to address the following challenges: a) efficiently utilize available network resources in a dynamic environment, b) provide services independently from the underlying topology, c) ensure reliability of services in case of network topology changes and d) reduce the management complexity and increase flexibility to application developers. In order to address these challenges, autonomic functionalities have to be incorporated. The following self-* properties have been defined [B. Jennings et al., 2007] and should be supported by an autonomic system: self-configuration, self-optimization, self-awareness and self- healing.

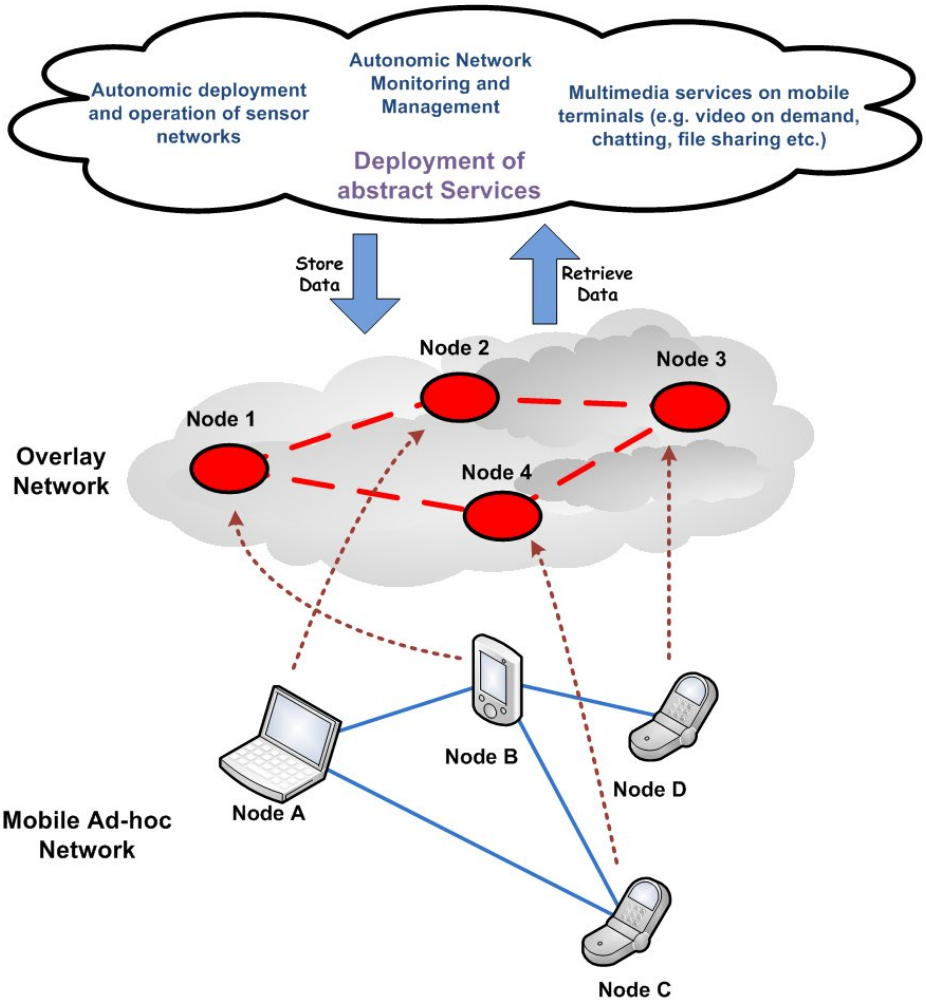


Figure 4.1 - Autonomic Services in Mesh Environments

Existing protocols that satisfy partially the challenges described above were considered during the design of the proposed approach. There is no existing work on how to combine existing protocols for achieving autonomic service provisioning and how different protocols could interact with using predefined interfaces. Taking into account these considerations, the proposed approach is focusing on a) defining concrete layering for enabling autonomic service provisioning in Mesh networks, b) specifying the discrete functionality of each layer and the interfaces between them and c) resolving conflicts between existing protocols, specifically in the field of the overlay t opology construction.

The creation and maintenance of an overlay topology that logically interconnects all the participating nodes in the physical network is critical in our approach. Any node that connects to the ad-hoc network has to join to the overlay network. The overlay network is formulated during the topology stabilization phase in an autonomic manner and hides any details of the underlying physical infrastructure, e.g. link establishment or torn down, node failures, node mobility etc. In case of multiple changes in the physical topology, the overlay network is able to adapt quickly to the new environment (re-stabilization). Furthermore, recovery from failures can be easily achieved based on information that it is available in the network. All these tasks are realized without the intervention of the network administrator.

After the overlay network is established, participating nodes are able to store and retrieve data using typical p2p protocols. Every node that wishes to store a key-value pair or query a value based on a key can achieve it by using a Distributed Hash Table (DHT) [E. Kang et al, 2008] that operates on-top of the overlay topology. In a similar way, several applications can be built taking under consideration the existence of a high level API put(key,value) and get(key) that would interact with a DHT protocol that operates on-top of a non-reliable Mesh environment.

Provided services are designed based on assumption of collaboration and dissemination of information among the participating nodes. These services can be fully decentralized as data and functionality is allocated in different nodes at the overlay network. Some functions may be delegated to more than one node for higher reliability. In case of changes or failures, roles may be re-assigned autonomously and performance guarantees may be assured for the services provision.

4.2 Overview of the four-layered approach

We propose a four-layered scheme based on the functionality requirements imposed by the provided services and the underlying physical networking environment. As shown in Figure 4.2, the following four layers are defined: i) *Neighbor-to-Neighbor layer*, ii) *Routing layer*, iii) *Topology Maintenance layer* and iv) *DHT layer*. Each layer has a discrete role, implements different mechanisms and specifies its messages types. The proposed layered approach is independent from the selection of p2p protocols, topology formulation mechanisms and routing protocols. Therefore, any combination of different protocols may be selected and proper adaptations may be proposed.

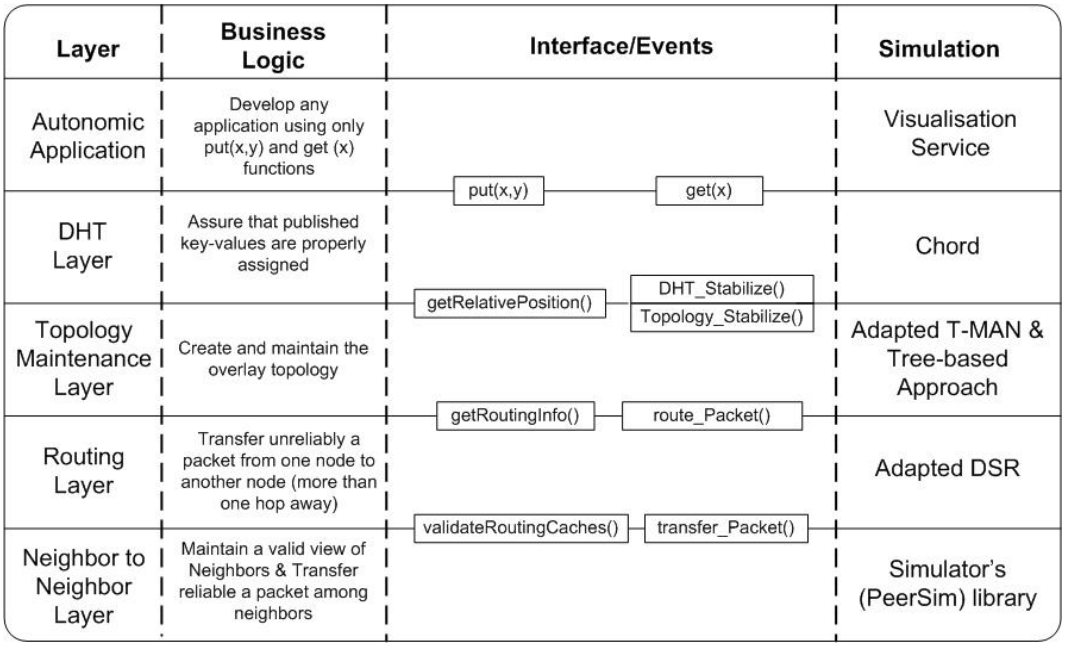


Figure 4.2 - Four-layered Approach

The Neighbor-to-Neighbor layer is responsible for delivering an upper-layer frame from a neighbor to another neighbor. No information from the upper layer is necessary for the delivery. Two types of messages are used: i) *MAC_SEND* in order to achieve one way frame delivery from neighbor X to neighbor Y and ii) *MAC_ACK* in order to achieve acknowledgment for successful message-delivery from neighbor Y back to neighbor X. Also, this layer is responsible for maintaining (i.e. initializing and keeping up-to-date) the routing cache of the Routing layer since, when neighbor-to-neighbor links are created or destroyed, the related routing information has to be updated.

The Routing layer is responsible for delivering an upper-layer frame from a node X to another node Z. It is assumed that node X is not aware how node Z can be reached. The layer is also agnostic of the reason that node X wants to communicate with node Z. This

layer relies on routing protocol for frame forwarding across the network. As we stated in section 3.3, in case of Mesh Environments it is suggested the use of a re-active routing protocol.

The Topology Maintenance layer is responsible for formulating a virtual topology of the participating nodes. In our case, the desired topology is a ring (imposed by the use of Chord). Consequently, this layer undertakes the task of identifying the relative position of each node in the overlay topology without being based in centralized or semi-centralized techniques.

The DHT layer is responsible for maintaining a distributed hash table that is bootstrapped over the stabilized overlay topology. For this purpose any existing DHT protocol may be used. These protocols are (semi or fully) decentralized and -in addition to storage and retrieval functionality- may succeed load balancing, reduce bandwidth consumption and improve data reliability across the network. The following interfaces have been defined for the communication among the different layers:

- The Neighbor-to-Neighbor layer provides to the Routing layer routing information for existing neighbors that is stored in the routing cache of each node, through the *validateRoutingCaches()* function. The Neighbor-to-Neighbor layer provides also medium-level acknowledgments to the Routing layer for neighbor-to-neighbor communication, through the *transfer_Packet()* function.
- The Routing layer provides routing functionality to upper layers through the *routePacket()* function. Additionally, it exposes topology information derived directly from the routing caches to the Topology Maintenance layer, through the *getRoutingInfo()* function. It is up to the Topology Maintenance layer to utilize this information for optimizing its mechanisms or not.
- The Topology Maintenance layer provides information to the DHT layer regarding the relative position of a node in the overlay network (e.g. the predecessor and successor in case of a ring topology) through the *getRelativePosition()* function. In case of changes in the network topology, stabilization procedures take place in both layers. The *Topology_Stabilize()* function is used for re-ordering the overlay topology (e.g. ring in our case) and triggers the *DHT_Stabilize()* function that is used for the re-assignment of key-value pairs that are assigned in the overlay network nodes.

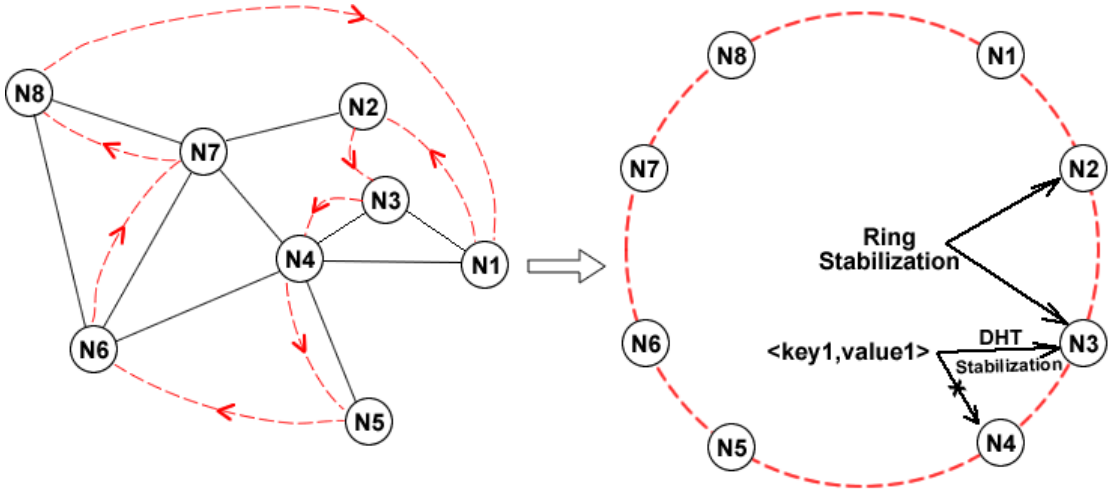


Figure 4.3 - Overlay topology stabilization & DHT entries stabilization

In Figure 4.3, a snapshot of the physical network topology (solid lines) and the logical overlay topology (dashed lines with arrows) is depicted. Initially, node 3 does not exist in the network and the key-value pairs have already been assigned to the network nodes by applications that run on the existing nodes (i.e. applications that use DHT). Then, node 3 is physically connected with node 1 and node 4 and the corresponding overlay topology is updated. It is the responsibility of Topology Maintenance layer to find thesuccessor for each node. However, it is not the Topology Maintenance layer's responsibility to reassign key-values according to the DHT's assignment algorithm.

The Topology Maintenance layer must inform the DHT layer that the relative position for the node in the overlay topology (e.g. ring in case of Chord) has changed. Then it is up to DHT layer to re-assign key-value pairs. This re-assignment will be addressed as DHT re-stabilization while the updated knowledge for the relative position in the overlaytopology is called Topology stabilization.

4.3 Protocol Analysis

In this section, a reference implementation of the proposed approach is described in detail. Specific mechanisms and protocols are selected for each independent layer. In the DHT layer we have selected Chord [W. Ding et al., 2007] as a p2p protocol, in the Topology Maintenance layer we follow gossiping and exhaustive tree-based techniques, in the Routing layer we have adapted the Data Source Routing (DSR) [D.B. Johnson et al., 1996] protocol and in the Neighbor-to-Neighbor layer communication is established through a simulation environment.

It is important to note that in addition to adopting specific techniques and mechanisms, several adaptations - that were considered useful for our approach - are provided. Specifically we describe in detail how gossiping techniques for topology formulation and the DSR routing protocol can facilitate each other. These adaptations are stated in detail in the following sub-sections.

4.3.1. Adapted DSR Routing Protocol

Prior to analyzing the overlay topology algorithms that have been implemented we briefly refer to the reactive routing protocol that has been adopted and customized. The Data Source Routing (DSR) is chosen for our reference implementation. Firstly, we describe the basic characteristics of DSR and the implementation details and secondly the proposed adaptations that are correlated with the Topology Maintenance layer.

DSR is relied on two mechanisms: a) *Route Identification* and b) *Route Maintenance*. The protocol if fully reactive; hence every time one node wants to communicate with another node, it initiates a route request mechanism. Note that it is not DSR's responsibility to know why one node wants to communicate with another node. In general, this is upper layer's responsibility (Topology Maintenance layer in our case). The Route Maintenance mechanism is used for route identification (see Figure 4.4). A specific message, called DSR_Route_Request, is broadcasted. Each DSR_Route_Request contains a request-id that is given by the initiator and the desired-destination node. This information remains intact all across the flooding procedure. The message also contains a header that logs the routing path of a route-request. Thus, a route-request message that is initiated by one node ends up to many route-request messages with the same request-id but with different headers (because different pathways are followed). Each node that receives a DSR_Route_Request can perform three tasks:

If a node is the destination-node, it creates a route-reply message, called *DSR_Route_Reply*. This message, in parallel with the request, contains a *reply-id* (which is directly correlated with the *request-id*) and a header (which is actually the reversed route-request header), that remain intact until the reply reaches its destination. In case that a

route-request reaches a node that knows a-priori the route to the destination node, it appends to the header the remaining route and creates a *DSR_Route_Reply*.

If a node knows nothing about the destination node, it just forwards the *DSR_Route_Request* to its neighbors (excluding the neighbor that initially sent the request). In both replying and forwarding cases the node adds the request-id to a "served" list. The "served" list is a temporary list of all the request/response ids that are served by a specific node, after the processing of a DSR_Route_Request (the same applies in route replies). If a node has the request-id of the DSR_Route_Request in its "served" list, the node does nothing at all. This is very critical for avoiding loops.

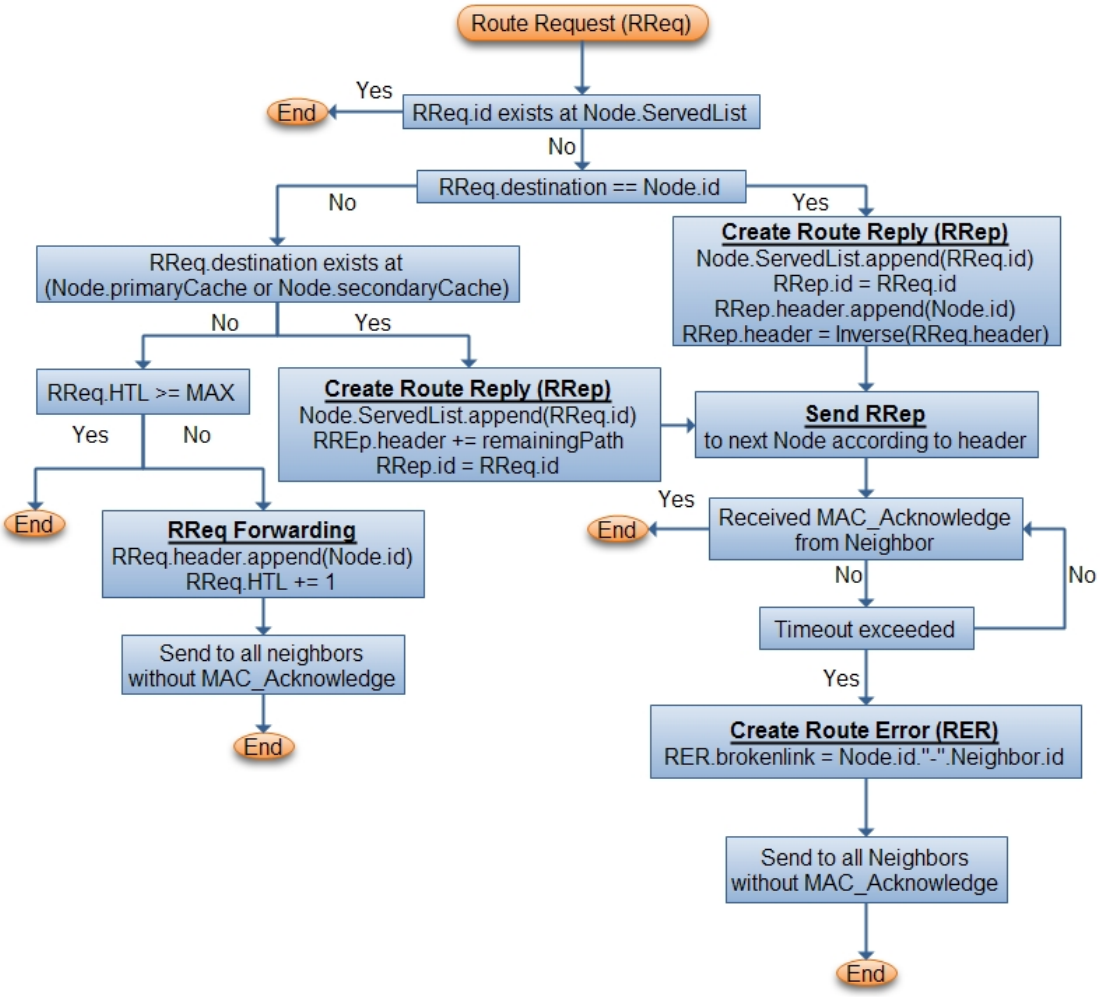


Figure 4.4 - Adapted *DSR_Route_Request* handling mechanism

When a *DSR_Route_Reply* reaches its destination, the node that initially initiated the route-request is "informed" for the route to a specific node. This information is stored in the nodes' routing cache. According to our implementation each node maintains a primary

routing cache and a secondary routing cache. The primary routing cache contains only one route for every known node and specifically the route which is considered to be optimum (for our implementation optimum means shortest but in general optimum may refer to other network parameters such as low latency etc.). The secondary routing cache contains multiple alternative routes without loops for known nodes. In case that a broken link is identified (through *DSR_Route_Error* solicitation mechanism) both routing caches are revalidated. If an invalid route -that contains this link- is identified in the primary routing cache, the route is removed and substituted by a possible alternative valid route, requested from the secondary routing cache.

As explained above, the Routing layer is necessary so as two nodes can exchange an upper layer frame. In order to do so, a message called *DSR_Message_Transfer* is used. When a route to a destination is known then a *DSR_Message_Transfer* is initiated. This message contains a packet-id (which is directly correlated with the sender's address), the destination node, the route that must be followed in the Mesh network in order to reach the destination and a flag that informs the destination node whether it should respond with a confirmation (acknowledgement). Finally, the *DSR_Message_Transfer* contains encapsulated upper layer data (topology formulation, DHT etc.).

When a *DSR_Message_Transfer* has to be transmitted from node A to node B and from node B to node C, the intermediate nodes are responsible to complete the transfer. Practically, if the *DSR_Message_Transfer* packet has reached node B and the header of the *DSR_Message_Transfer* is A → B → C, assuming that node B has direct connection with node C (because in the past a route_reply informed node A about that) and the connection between node B and node C is no longer available, then it is in the responsibility of node B to identify a new route to node C. Identification means usage of another route (consulting the routing cache) or initiation from scratch of a new route-request.

Whenever during a *DSR_Message_Transfer* a broken link is identified, then a message called *DSR_Route_Error* is initiated by the node that declares the broken link. The *DSR_Route_Error* contains a route-error-id and information about the broken link. This message is flooded all across the Mesh network. When a node receives a *DSR_Route_Error* it automatically removes from the primary and secondary cache all the occurrences of routes that contain this link. Afterwards each node re-evaluates its primary and secondary cache in order for the optimum (shortest) paths to be pushed in the primary cache.

At this point, we have to emphasize in some adaptations that we propose. The first adaptation is regarded to the *DSR_Route_Reply* messages. Each *DSR_Route_Reply* contains a valid up-to-date path that is routed from the destination back to the route-request initiator. This information that is extremely valuable also for the intermediate nodes. In our implementation all intermediate nodes overhear the route-replies that pass through them and enrich their routing cache. The second adaptation regards the introduction of the

Hopes-To-Live (HTL) parameter. As we already mentioned, the route-request mechanism uses the request-id in order to prevent infinite loops. The *HTL* parameter is added as a field in the *DSR_Route_Request*. Every time a route-request is forwarded then the *HTL* parameter is increased by one. When the maximum *HTL* is reached the node does not forward the *DSR_Route_Request* messages any more.

The idea behind the *HTL* is the following: it is useless for a *DSR_Route_Request* to be forwarded more times than the diameter (in hops) of the network. As a diameter we define the maximum non cyclic route that can be accomplished. However, the problem for a Mesh network is that it does not have a stable diameter and even if the nodes are stationary the diameter has to be collaboratively inferred by a respective protocol. In our implementation we take in account the worst case as we state in the following paragraphs.

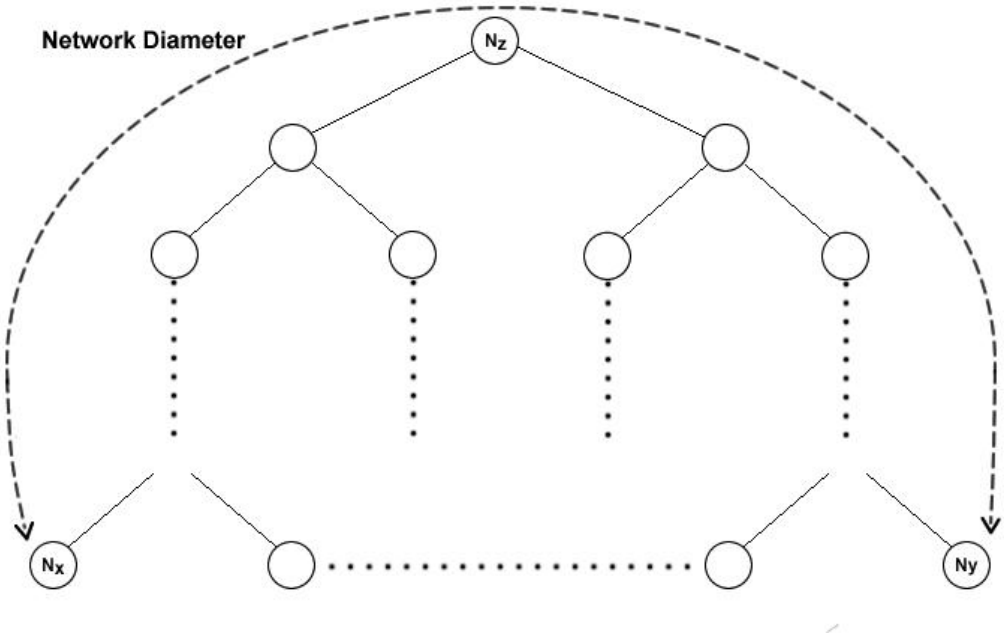


Figure 4.5 - Tree-based representation

We assume that we have a network where each node has approximately N_d neighbors (***d*** refers to network density). Without loss of generality we can represent this network as a tree with tree-degree: $T_d = N_d - 1$. As someone can notice from Figure 4.5, the worst case scenario for a route request is to go from one leaf to the root of the tree and back to another leaf; i.e. twice the tree depth. Consequently the maximum HTL that a route request must have in order to find one node is tightly bound to the depth of the virtual tree, which represents our network. We refer to this depth as ***k*** and we identify the correlation of k with the network size ***N*** and the network degree N_d. The number of leafs of

a tree with degree T_d and depth k is T_d^k. The total amount of tree nodes (or network nodes) is N and the depth of the virtual tree is k, as it is shown in the following equations:

$$N = T_d{}^0 + T_d{}^1 + \ldots + T_d{}^k = \frac{T_d{}^{(k+1)} - 1}{T_d - 1}$$

$$\Rightarrow \quad N = \frac{(N_d - 1)^{(k+1)} - 1}{N_d - 2}$$

Equation 1

$$k = \frac{\log\left(N(N_d - 2) + 1\right)}{\log\left(N_d - 1\right)} - 1$$

Equation 2

In Figure 4.6, it is shown that the depth of the virtual tree that represents our network is correlated with the network size for several network degrees. As we are going to see in the simulation results the introduction of the maximum HTL in the DSR, results in radical reduction of the *DSR_Route_Request* messages. It is important to note that in order to estimate the current depth of the network, based on the network size and degree, averaging techniques may be applied [M. Jelacity-2 et al., 2005]. In these techniques, each node interacts with its neighbors in order to calculate the mean value of a parameter and convergence is succeeded after a small number of iterations.

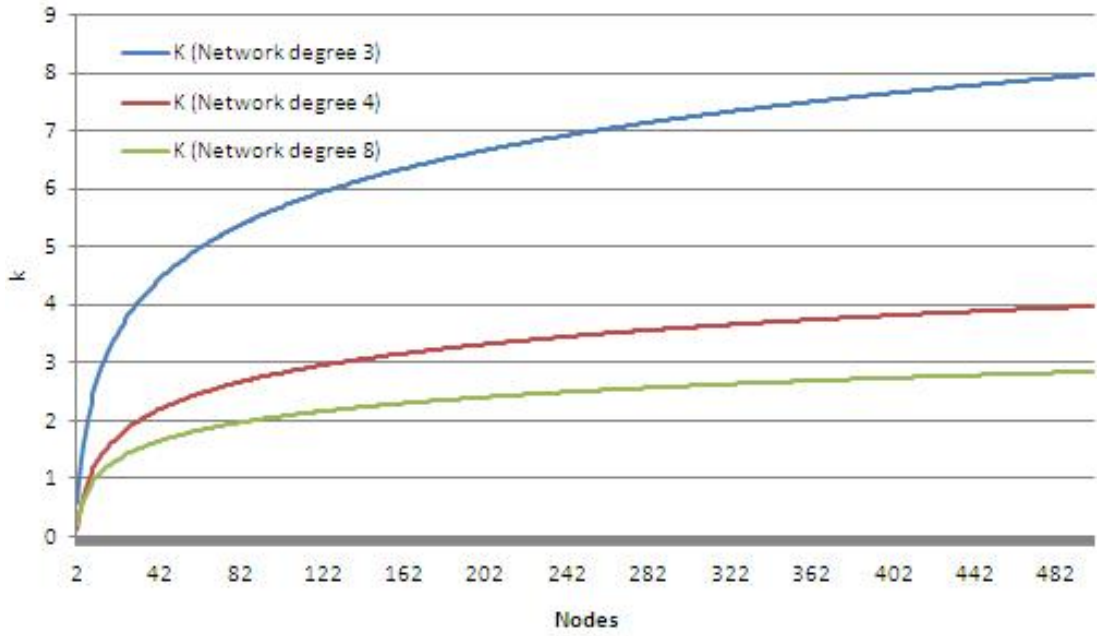


Figure 4.6 - Virtual tree depth vs network size

4.3.2 Overlay Creation and Maintenance

4.3.2.1 Adapted T-MAN Protocol

Up to now we have examined the Routing layer that is the cornerstone regarding message exchanges for the upper layers. Such an upper layer is the Topology Maintenance layer. The topology formulation mechanism is responsible for identifying the relative position of a node among all the network nodes. This is essential for providing DHT functionality. As stated earlier, the challenge is to apply a topology formulation algorithm for topology creation and maintenance to a Mesh environment.

One of the most prominent algorithms that have been proposed for general purpose topology formulation is T-MAN [M. Jelacity-1 et al., 2005]. The T-MAN approach is a gossip-based approach where each node refines its view about its relative position in the overlay based on the "knowledge" of the "closest" nodes that exist in a buffer. Each node that participates in a network uses a ranking function to evaluate its distance from another node. The evaluation parameters can vary. Some indicative examples are the network address, proximity, latency etc. The usage of different ranking function results in the creation of different topologies.

In our implementation we are based on the creation of an overlay ring topology. In our experiments we use as evaluation parameter the network address and as a distance function: $d(a, b) = \min(N - |a - b|, |a - b|)$.

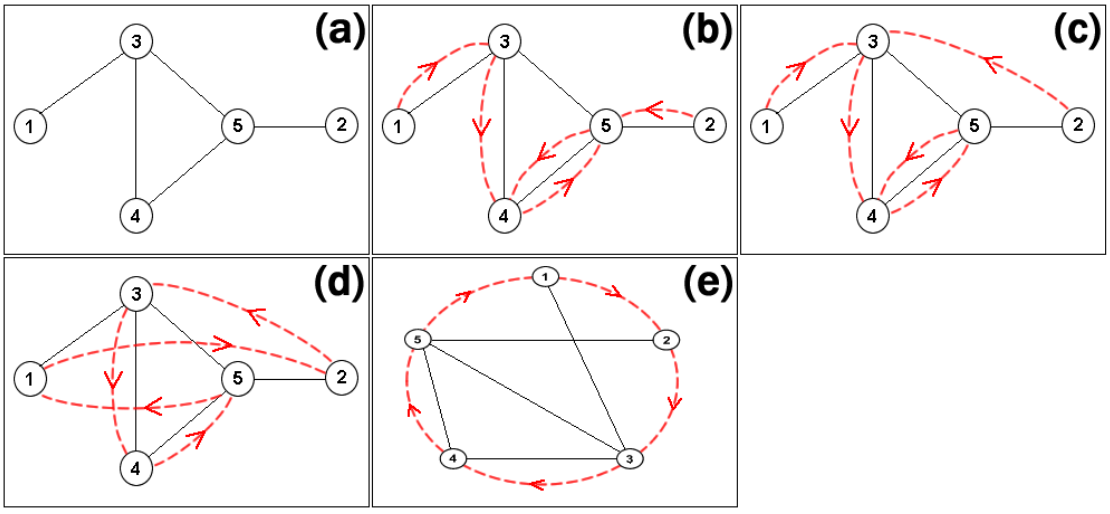


Figure 4.7 - T-MAN stabilization process

For example, we consider the topology where we have a network with 5 nodes and we assume that the network addresses of the participating nodes are 1,2, 3, 4 and 5 respectively (see Figure 4.7(a)). For the 3rd node, nodes 2 and 4 have score 1, while nodes 1 and 5 have score 2. In order to succeed successor identification as fast as possible, T

MAN applies a gossiping technique. Specifically, each node maintains a view with the nodes that are, up to a specific time, known and scored. Periodically, each node communicates with the most "closer" node solicits its view and requests the current view of the "closer" node (see Figure 4.7(b)). After this mutual exchange, nodes re-evaluate their views ((see Figure 4.7(c) and Figure 4.7(d)). This iterative procedure leads to extremely fast convergence (or stabilization), where convergence refers to the state that each node knows its successor and any further exchange of messages leads to no further refinement of the views (see Figure 4.7(e)).

do forever{
Node_To_Sentp ← selectCloserNode()
buffer ← merge(view,{myNodeDescriptor})
send buffer to Node_To_Sentp
receive bufferp from Node_To_Sentp
buffer ← merge(bufferp ,view)
view ← Reevaluate(buffer)
}
	do forever{
	receive bufferq from Senderq
	buffer ←
	merge(view,{myNodeDescriptor})
	send buffer to Senderq
	buffer ← merge(bufferq ,view)
	view ← Reevaluate (buffer)
	}

active thread	passive thread

Table 4.1 - T-MAN parallel threads

T-MAN is relied on two parallel threads as it is shown on Table 4.1. Since T-MAN is proposed for routed networks, the only variable parameters that exist in the above Table are the size of the view and the size of the buffer. The operation "send buffer to $Node\_To\_Sent_p$" of the active thread and "$send\ buffer\ to\ Sender_q$" of the passive thread in our layered approach is equivalent of initiating a *DSR_Message_Transfer* between these nodes. As mentioned above, a *DSR_Message_Transfer* consults the routing cache to find out if a route to the destination exists. If not, the route-request mechanism is initiated. Method "*merge*" returns a list that contains all the elements that exist in the sub-lists that are passed as arguments. Finally, method "*Revaluate*" applies the scoring function to all the elements of the argument-list and returns only the first *n* occurrences where *n* is the size of the view.

Taking under consideration that DSR is continuously "working" and contains up-to-date routing information, an adaptation (adapted T-MAN) of the initial T-MAN is proposed as it is shown in Table 4.2. In the adapted T-MAN approach there are the three following major changes.

Before the active thread sends its view to the closest node(s) it consults the DSR cache. Consulting means actually that all nodes that exist in the primary and secondary cache are scored (using the Ring-scoring function in our case). This "consultation" results in

major reduction of the T-MAN messages that have to be exchanged (before the ring is stabilized). Send-to-node is substituted by *DSR_Message_Transfer* as explained above.

Per each "cycle", instead of sending one message to the closest node that is in the node's current view, we send multiple messages (as defined by the parameter gossip factor). Multiple messages facilitate the fast dissemination of information regarding the network topology, something that is very important in case of dynamic networks. It is obvious that the gossip factor can be between one and buffer size. Proper selection of this option may accelerate the convergence while keeping the routing overhead low.

do forever{
buffer ← merge(view,{myNodeDescriptor})
routingcachelist←ExtractNodesFromCache()
buffer ← merge(view, routingcachelist)
for (i=0;i<gossipfactor;i++){
Node_To_Sentp ← selectCloserNode(i)
DSRMessageTransfer to Node_To_Sentp
receive bufferp from Node_To_Sentp
buffer ← merge(bufferp ,view)
view ← Reevaluate(buffer)
}}
	do forever{
	receive bufferq from Senderq
	buffer ←
	merge(view,{myNodeDescriptor})
	DSRMessageTransfer to Senderq
	buffer ← merge(bufferq ,view)
	view ← Reevaluate (buffer)
	}

active thread	passive thread

Table 4.2 - Adapted T-MAN parallel threads

4.3.2.2 An alternative-exhaustive tree-based approach

In order to have a comparative view of T-MAN we have also implemented an exhaustive tree-based approach (see Figure-4.8). According to this approach each node of the network is responsible to find its successor by using an excessive flooding mechanism. Each node in the network solicits a Find_Successor_Request message after assigning a request-id to that message (see Figure 4.9a and 9b). This message flows across the nodes using a Neighbor-to-Neighbor communication scheme (and not DSR node-to-node message transfer scheme as it happens at adopted T-MAN). After sending the message, a "served" list is enriched in each node with the request id. In this way each node keeps track of the request-ids that have been forwarded by the node. Before the *Find_Successor_Request* is forwarded to the next neighbor, each node enriches its header with all the nodes that are "visible" as neighbors to that node. Consequently when the next node receives the *Find_Successor_Request*, it has to check if all of its neighbors exist in the header of the request. If not, it enriches the header and forwards the message to the neighbors that did not exist in the header (see Figure 4.9c and 9d). The critical part of

the process is that if one node receives a *Find_Successor_Request* and the request-id is not in the served list and the header contains all of its current neighbors, then this node is considered to be a "virtual leaf" of a tree that has as root node the initiator of the Find_Successor_Request (see Figure 9e).

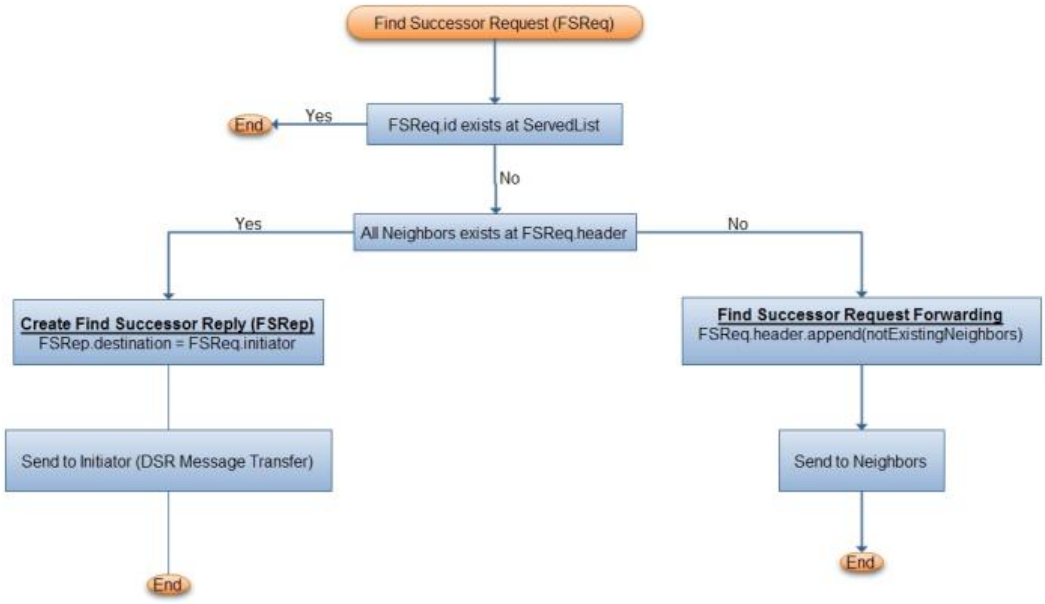


Figure 4.8 - Exhaustive tree-based flooding mechanism

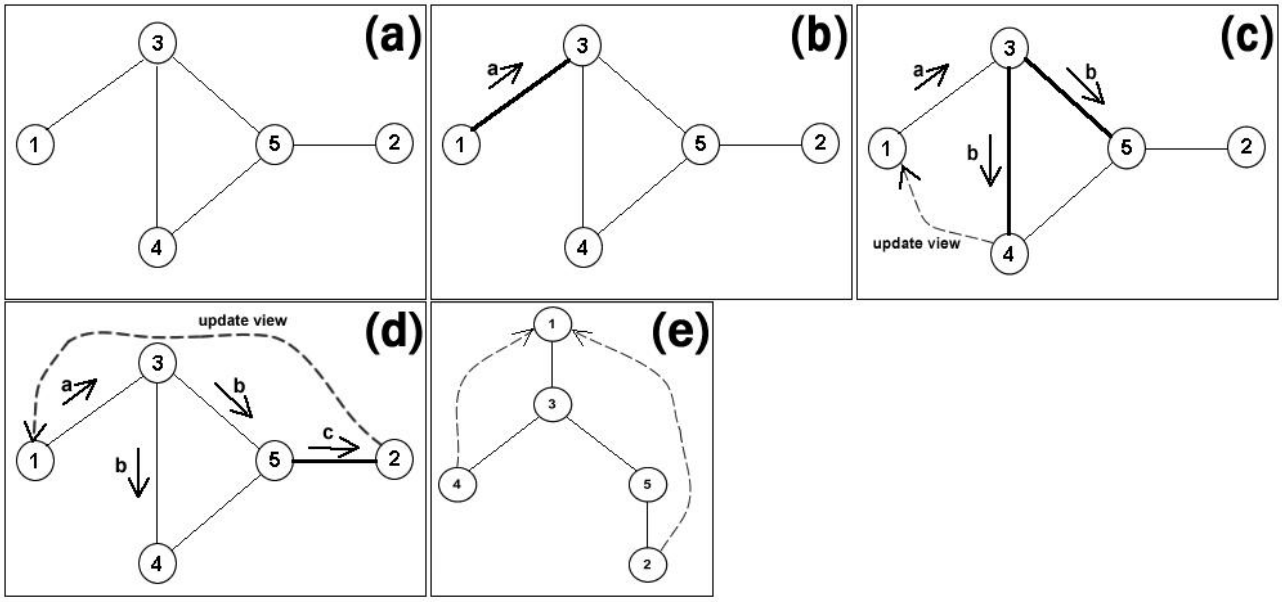


Figure 4.9 - Tree-based stabilization process

The virtual leaf uses DSR's message transfer mechanism to send a Find_Successor_Response back to the initiator (root). This message contains the request-id of the Find_Successor_Request and the header that was gradually built up to the point that a node inferred that is a leaf. Consequently a *Find_Successor_Request* initiated by a

node will result in as many *Find_Successor_Responses* as the number of the virtual leafs that exist. Also DSR's message transfers are equal to the amount of *Find_Successor_Responses*. Each root that receives the *Find_Successor_Responses* updates its view as far as its successor is concerned.

The number of *Find_Successor_Responses* in order to succeed stabilization of the overlay ring topology is the sum of the *Find_Successor_Responses* that receives each node in the tree. The root node receives Td^k responses (the number of the virtual leafs). In the first level of the tree there are T_d children where each one receives T_d^k responses. In the second level of the tree there are T_d^2 children where each one receives T_d^k responses. In the (k-1) level of the tree there are T_d^{k-1} children where each node receives T_d^k responses. Finally, in the k level of the tree there are Td^k children where each node receives $T_d^k \cdot (T_d^k - 1)$ responses. Thus, the number of Find_Successor_Responses that are required for stabilization is:

$$Routing\_Cost = \frac{(T_d^{\,k} - 1)T_d^{(k+1)}}{T_d - 1} = \frac{((N_d - 1)^k - 1)(Nd - 1)^{(k+1)}}{N_d - 2} \quad \textit{Equation 3}$$

$$\Rightarrow Routing\_Cost = \frac{(N_d - 1)^{(2k+1)} - N_d + 1}{N_d - 2}$$

4.3.3 Chord P2P Protocol Adjustment

For implementation purposes Chord has been selected. Chord is a simple but powerful protocol, which solves the problem of efficient data storage and retrieval on-top of fixed networks. It is an efficient distributed lookup protocol based on consistent hashing. Its basic principle is to map a key of a key-value pair to a responsible node. Each node maintains routing information about O(logN) other nodes, and lookups are feasible via O(logN) messages. Therefore, Chord scales well with a number of nodes and, thus, it can be applicable to large systems. Chord continues to operate correctly even if the system undergoes major changes or if the routing information is partially correct.

Chord does not implement services directly but rather provides a flexible, high-performance lookup primitive, upon which such functionality may be efficiently layered. Its design philosophy is to separate the lookup problem from additional functionality. By layering additional features on top of a core lookup service, overall systems gain robustness and scalability [A. Rowstron et al., 2001]. The DHT layer uses three types of messages: i) *PUT(k,v)* in order to associate the value v with the key k, ii) *GET(k)* in order to retrieve the value that is associated with the key k and, iii) *STABILISE()* in order to move parts of the keys that are stored in the local cache to another node due to changes in the topology (ring).

A thorough analysis of the Chord protocol has been provided at the State Of the Art analysis (Chapter 3.2). The adaptations that have to take place in order to operate on-top of Mesh networks are: a) the bootstrapping procedure which is based on gossiping (i.e. every node joins the overlay after its relative position is identified), b) the asynchronous maintenance operation of figure-table-fixing is becoming trigger-based (i.e. when gossiping procedure stabilizes) c) successor/predecessor maintenance is delegated to the low-level mechanisms and d) all socket based communications are substituted by reactive routing mechanisms.

A precise simulation model for all three layers that are analyzed above has been implemented in PeerSim [M. Jelacity et al., 2010]. On the other hand, for implementation purposes an already existing Chord implementation (OpenChord) has been adjusted (http://open-chord.sourceforge.net) which implies that only the first three layers and the adjustments have been implemented from scratch. The architectural analysis of UbiChord is provided on Chapter 5. Simulation model is also available at www.autonomicity.net. Additionally, simulation results for various aspects of the layered approach are presented at 6.1.1 and 6.1.2.

4.4 Optimizations based on Feedback from Simulation Results (NEURON)

The proposed four layer approach has been implemented in PeerSim in order to evaluate its scaling and performance. According to the simulation results (see Section 6.1) the four-layered approach behaves linearly as far as the signaling cost is concerned, for mesh networks that do not exceed ≈120 nodes. After this point, the routing layer introduces a significant messaging cost that has to be suppressed. Suppression of this cost will be achieved by some additional mechanisms that undertake the tasks of network-parameters' identification, clustering and hierarchical routing. The following sections provide a thorough analysis of these mechanisms.

4.4.1 Autonomic Estimation Algorithm (AEA)

The knowledge of network-based parameters is crucial for achieving high efficiency and optimizing mechanisms in a Mesh environment. The estimation of such parameters is challenging in dynamic environments, especially when there is no information available at the initial deployment or after a topology change. Routing and Topology maintenance layer should adapt their mechanisms based on network-wide parameters without presuming a priori knowledge of their values. Therefore an Autonomic Estimation Algorithm is introduced.

According to this algorithm, an estimation of the parameters' values is autonomously produced and updated regularly without imposing significant network overhead in terms of messages exchanged. The mechanism is activated in each mobile node when the network bootstraps or the topology changes significantly. It converge s in a short number of cycles and is applicable to large scale Mesh networks. This mechanism is presented in this section based on the principles of neighbor-based gossiping and specifically using averaging techniques based on neighbor-based gossiping [Lee et al., 2008].

```
converged=false;
neighbconverged=false;
do forever{
if (cycle mod resetcycle ==0){
converged=false;
neighbconverged=false;
paramvalue = CountNeighbors();
ReceivedMSG=0;
}
if (ReceivedMSG!=0){
oldparamvalue= paramvalue;
paramvalue=(paramvalue) /
ReceivedMSG;
if (Abs(oldparamvalue-
```

```
On_Message_Receive_Event {
paramvalue+=ReceivedFrame[paramvalue];
if (ReceivedFrame[conver ged]==false)
neighbconverged=false;
ReceivedMSG++;
}
```

paamvalue)<threshold){
converged=true;
}
}
if (neighbconverged==false){
for (i=0;i<Neighbors.size();i++){
Send Frame[paramvalue,coverged] to Node$_i$
}
neighbconverged=true;
}
}	

active thread	passive thread

Table 4.2 - Averaging through Neighbor-gossiping

According to the AEA, each node interacts with its neighbors in order to calculate the mean value of a parameter (Table 4.2). Each node calculates its initial value for a parameter and sends this value to its neighbors. In parallel each node receives from its neighbors their calculation about this parameter. After each "cycle" of mutual exchanges, each node revises its calculation using a weighting average:

$$\boldsymbol{Updated}_{\boldsymbol{Value}} = \frac{\boldsymbol{Value}_{\boldsymbol{from}_{\boldsymbol{neighbor}_1}} + \cdots + \boldsymbol{Value}_{\boldsymbol{from}_{\boldsymbol{neighbor}_n}}}{\boldsymbol{n}} \qquad \textit{Equation 4}$$

When the updated value after the completion of a cycle differ less than a specified threshold from the previous one, the parameter is considered as converged on this node. In this case, the node, in the next cycle, sends its converged value *along* with a flag that indicates that the node considers the parameter-estimation as precise. Only when all messages that are received during a message-exchange-cycle contain this convergence flag, a node decides to stop broadcasting its current value about a parameter. This procedure is repeated periodically in order to calculate the updated values of the network parameters.

This technique presents many advantages since there are no preconditions during the network bootstrapping, the estimation is conducted in an ad-hoc manner and the algorithm converges quickly, even for large scale networks, while communication overhead is kept low. The frequency of the periodic estimation mechanism is related to the application dynamicity. According to the AEA, this technique is used for size and average density estimation. The knowledge of these parameters is indicative for the possible network topology scheme and facilitates the good distribution of cluster heads in the clustering mechanism, as we explain in detail in Section 4.4.2. However, the same

technique may be used for the autonomic calculation of other parameters within a Mesh (e.g. variance in the cluster sizes, available energy percentage).

For the density estimation, each mobile node calculates the number of its neighbors and thus the converged parameter is considered to be the average network density. For the size estimation, one or more predefined nodes in the network initialize the parameter *Network_Size* to *1* (at *step 0*) while the rest nodes initialize the parameter *Network_Size* to *0*.

When the averaging protocol converges, the estimated value is $1/(N \cdot k)$ where N is the network size and k the number of nodes that initialized the parameter *Network_Size* to *1* [M. Jelacity-2 et al., 2005]. The following adaptation is proposed in order to estimate the *Network_Size* autonomically (without the need to predefine specific nodes that initialize the parameter *Network_Size* to 1). All nodes have a random number generator and express their initiative to initialize their *Network_Size* parameter to *1* with a certain probability. We address this probability as P_{init}. The critical part of the adaptation is that all nodes respect the same probability. P_{init} varies from *0.1* to *0.3*. When the Network_Size parameter converges the converged value is approximately $1/(N \cdot P_{init})$. By inversing the converged value, an approximation of the network size is available.

However, this adaptation provides a parameter's estimation with some variance. In case that we desire to have more precise estimations, each node that chooses to initialize its *Network_Size* parameter to *1* has to accompany the broadcasted message with an additional field called *SolicitatedGroup*. In this field the MAC address of the node is placed. Each node maintains a cache that contains all the solicitated MACs and in parallel, during each exchange of messages, solicits the contents of its cache. Then, in addition to the convergence criteria that were formulated previously, each node is not meant to be converged if the number of MACs that exist in its cache is not equal to the number of MACs that are solicited by its neighbors. Following this adaptation the converged value is exactly (not approximately) $[1/(N \cdot NoMacs)]$ where *NoMacs* stands for the number of solicited MACs. The advantage of this adaptation is that it generates extremely precise results, albeit at the expense of a larger amount of messages exchanged until convergence is achieved.

4.4.2. Cluster Formulation, Maintenance and Update Mechanism

The cluster formulation mechanism in NEURON has a significant impact on the Mesh Network from multiple perspectives. It improves the energy efficiency and accelerates the deployment of higher layer protocols and specifically routing layer. Clusters are autonomously formulated, maintained and updated based on neighbor to neighbor communication among the network nodes. The Autonomic Estimation Algorithm provides information necessary for the optimization of the Cluster Head (a.k.a. CH) selection and distribution. Routing information acquired during the clustering process is stored in nodes' local caches and used by the routing algorithm applied. Controlled flooding is also utilized in order to avoid traffic forwarding outside the cluster zone.

NEURON allows each node to become a CH according to specific criteria. A node that is selected as CH acts as a proxy for the rest of the members in its cluster. Each node may be in two states; either belonging to a cluster or being in the process of joining to a cluster. The clustering process starts only after the Autonomic Estimation Algorithm has been converged and, thus, the size *N* and density *d* of the Mesh network is estimated. Based on these two parameters, each node decides to become a CH with a probability P_{clust} given by the following equation:

$$P_{clust} = \frac{\log(N)}{10 * d} * KPI \quad \textit{Equation 5,}$$

where: $KPI = 0.5 * \frac{available\_battery}{total\_battery} + 0.5 * \frac{available\_memory}{total\_memory}$ *Equation 6*

The Key Performance Indicator (a.k.a. KPI) in Equation 5 refers to the capabilities of each node. Nodes with better KPI present higher probability of becoming CHs and remaining in this status for a longer period of time until their resources are reduced significantly. In our case, the KPI is related with the available battery and memory of each node and is given in Equation 6. These parameters were considered crucial for a mobile node deployment and operation in a Mesh Network. However, any other parameter that better addresses application-specific requirements (for the KPI) may be selected.

According to Equation 5, a smaller number of CHs is expected to be elected in dense networks than in sparse ones. In addition, a higher number of CHs is anticipated in larger networks compared to smaller ones. Equation 5 was selected to cover a wide set of possible Mesh topologies. However, if the network size or density is known a priori, more optimal possibilities P_{clust} may be selected.

The P_{clust} is updated regularly based on the current parameters *N* and *d* of the network. This allows NEURON to adapt to changes in the network topology or conditions and optimise the clustering formulation process. For example, in case of a node transitioning from a CH operation mode to normal operation mode, the nodes of the

cluster decide to become CHs taking into consideration the latest estimations of the parameters *N* and *d*. In this case, if the average density is reduced, more than one CH may be elected. This process, combined with the cluster formulation update mechanism described below, facilitates the better distribution of CHs among the Mesh network and the extension in the network lifetime.

Each CH is responsible for periodically broadcasting its existence utilizing a controlled flooding mechanism. According to this approach, at a certain time interval the CH broadcasts one *MSolicitateCH* message. This message contains its MAC address (which is also the group identifier), its updated *KPI* and an auto increment number that is used for cycle prevention. The goal for this solicitation message is threefold.

Cluster Formulation & Maintenance: Upon the receipt of a MSolicitateCH message by a node not registered to a CH, a MRegisterNode2CH response message is send to the CH. The latter updates its routing cache and then the node automatically becomes a member of the broadcasted cluster. If the node belongs to the cluster controlled by the CH generating the message, the node forwards it to its neighbors. In addition, it stores the message to a local cache in order to avoid serving the same message again. Otherwise, if the node does not belong to the cluster controlled by the CH generating the message, the node belongs to the borderline between two clusters, as shown in Figure 4.10. In this case, the node does not forward the message to its neighbor but instead forwards it directly to its CH. This process is very critical as it prevents the unnecessary flooding out of the scope of a cluster and allows CHs to be aware of their neighboring CHs. The routing information, collected to the CHs' caches during the clusters formulation, facilitates the hierarchical routing mechanism, as discussed in Section 4.4.3. The cluster formulation and maintenance process is also shown in Figure 4.11.

Routing Cache Maintenance: Each *MSolicitateCH* message - broadcasted by a CH and flooded within the cluster - contains a list of the MAC addresses of the nodes that have already forwarded it since each node that serves this message appends its MAC to this list (only once due to cycle preventive mechanism). This information allows cluster nodes to learn (or update) the shortest path towards their CH and store this information in their local routing cache.

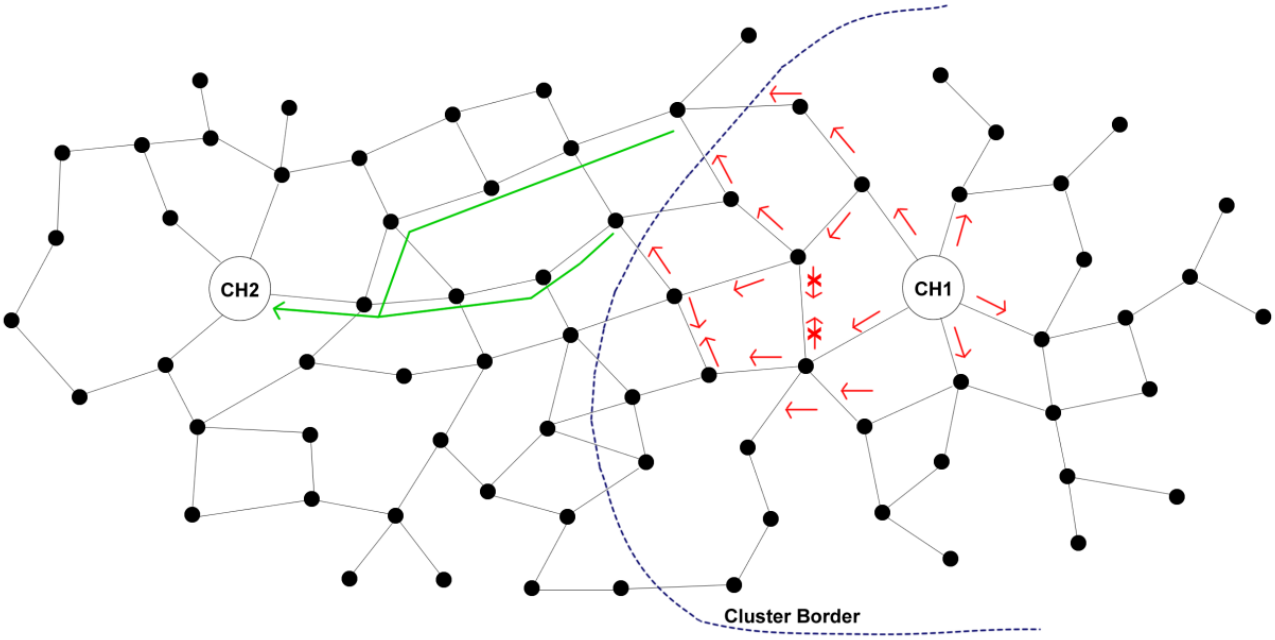


Figure 4.10 - Controlled Flooding Mechanism

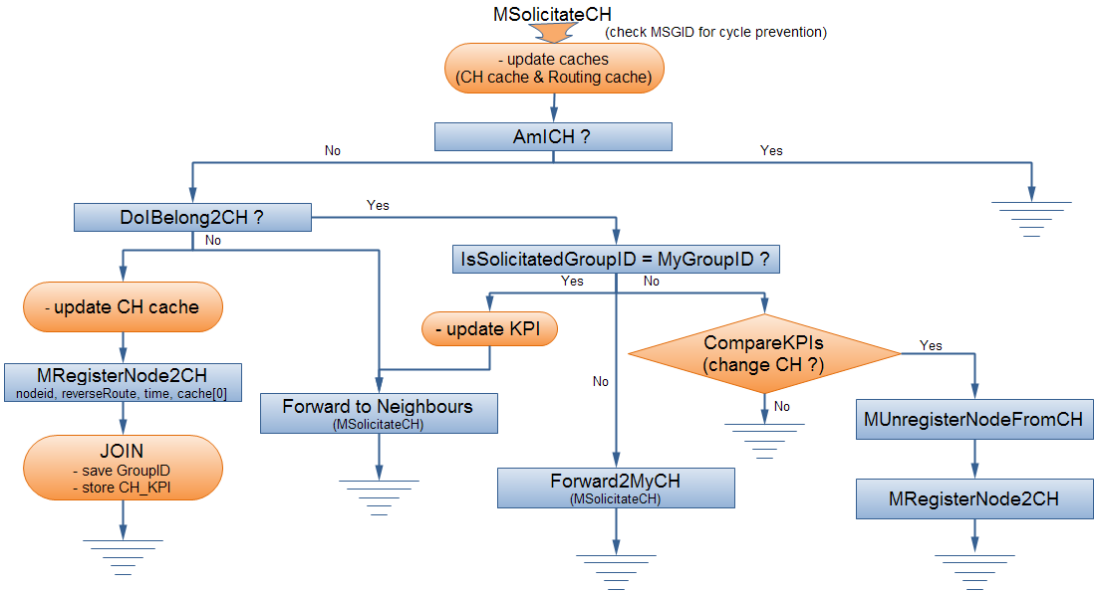


Figure - 4.11 Solicitation Mechanism

Cluster Formulation Update: The *MSolicitateCH* message allows nodes to be dynamically distributed among the existing clusters according to the CHs KPIs. In case that a node receives a *MSolicitateCH* message from a neighboring CH, it compares the received *KPI* with the *KPI* of its current CH. When this comparison overcomes a specified threshold, the node unregisters from its current CH and registers to the new CH. These tasks are accomplished with the usage of *MRegisterNode2CH* and *MUnRegisterNodefromCH* message, accordingly. This approach allows CHs to extend their lifetime since the load is re-distributed among the more powerful CHs.

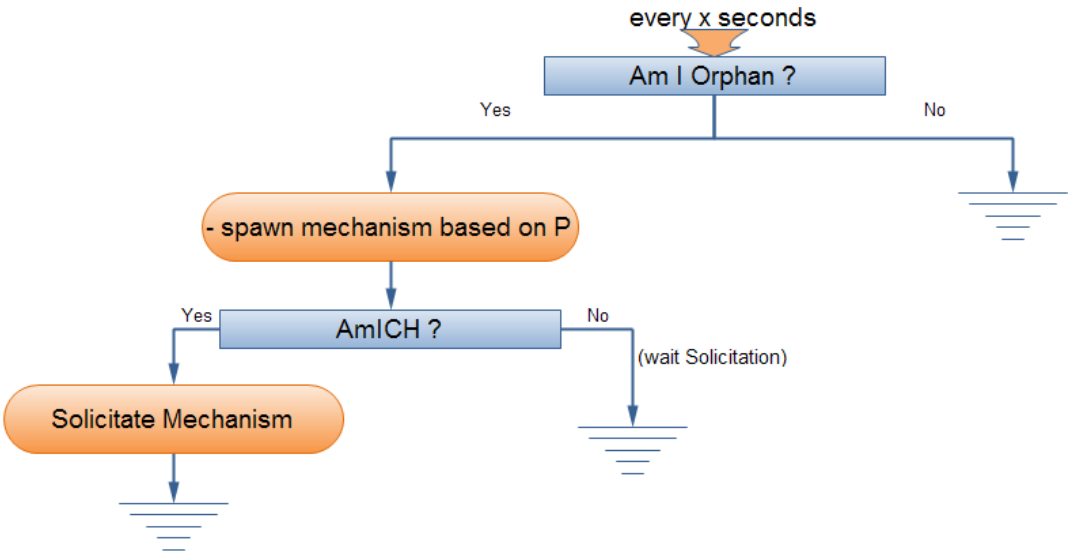


Figure 4.12 - Periodic Mechanism for CH election

NEURON clustering mechanism allows the autonomic re-formulation of clusters and enables the adaptation of the clusters' number and formation to the existing network conditions. This mechanism is initiated either because a CH decides to return tothe normal operation mode or because the CH leaves the network due to an unforeseen situation. In the first case, when the KPI of the CH passes below a specified threshold, the node stops to undertake the role of CH and a *MSolicitateCHDOWN* message is flooded within its cluster. Nodes update their routing cache and inform neighboring CHs, provided that routing information for them is available in their routing cache.

The re-clustering mechanism is then invoked and the cluster members decide to become a CH with the current probability P_{clust}. If no CH is elected, the nodes join a neighboring cluster after receiving a neighboring *MSolicitateCH* message. In the second case, if the *MSolicitateCH* message is not received within a period, the node updates the routing cache and initiates the re-clustering mechanism as previously (Figure 4.12). It

should be noted that the CHs remove any entries from their routing cache related with neighboring CHs if no relevant *MSolicitateCH* message is received within a specific period.

4.4.3. Hierarchical Routing

Routing and clustering mechanisms are interrelated in Mesh networks, both of them targeting to minimize energy consumption. It is desirable that packet forwarding and routing protocol overhead is distributed among all the mobile nodes according to their KPI values. This approach preserves scarce mobile node recourses and, thus, extends the network lifetime.

Energy efficiency in NEURON is achieved by hierarchical reactive routing. Nodes are organized into a hierarchy of clusters based on network proximity to the CHs. There is no proactive mechanism to build and maintain a valid routing table as the network topology continuously evolves. In addition, routing mechanism takes advantage of the routing cache entries generated during the clustering process, as presented in Section 4.4.2.

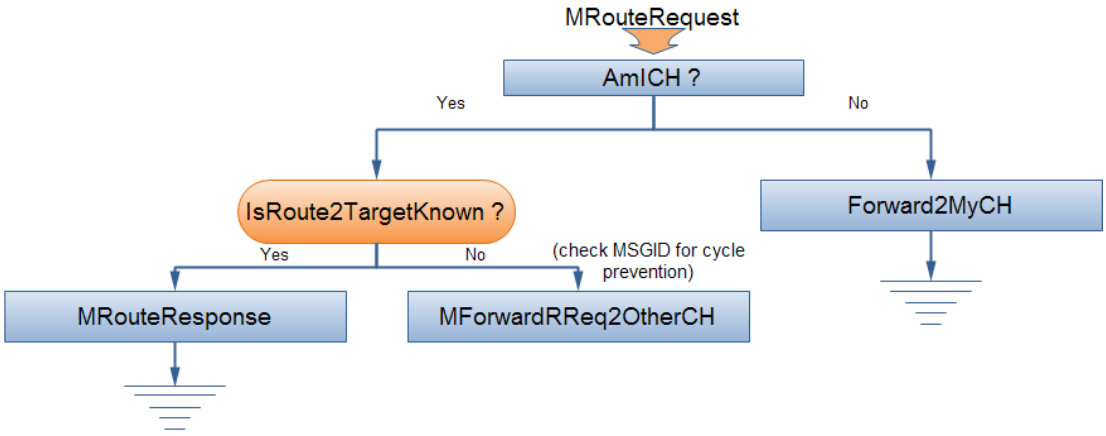


Figure 4.13 - Routing in NEURON

The routing algorithm in NEURON presents similarities with the DSR routing protocol. NEURON adopts some mechanisms from DSR for communication among CHs, while intra-cluster communication is designed independently. A *RouteRequest* message is used for detecting a valid route to a destination node, in accordance to the DSR protocol. In NEURON, however, the RouteRequest message is not flooded but directly forwarded to the CH of the transmitting node. The exact path to the CH is known via the *MSolicitateCH* messages broadcasted by CHs in regular intervals.

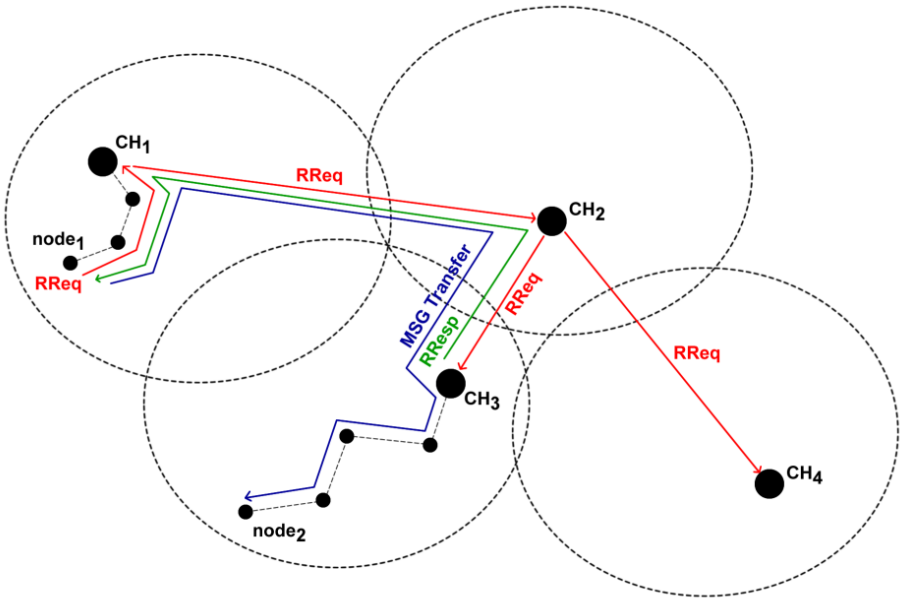


Figure 4.14 - Route Request Mechanism

When a node desires to establish communication with another node, it initially sends a RouteRequest message to its CH (Figure 4.13). The message contains the exact path towards the CH (source routing). If a node receives a *RouteRequest* message, it initially checks whether it operates as a CH. This is necessary because the election of new CHs is a dynamic process and thus new CHs may be present. If the node is not a CH, then it forwards the message to the next hop towards the CH according to the disseminated path from the initiator node. In case that a node along a path is unreachable, a *RouteError* message is generated and broadcasted within the scope of the cluster that contains the broken link. If the message is delivered to a CH during its path, the CH queries its local cache for the requested route towards the destination. If the valid entry is found, a *RouteResponse* message is sent to the initiating node. Otherwise, the CH forwards the *RouteRequest* message to its known CHs, exactly as routing is implemented in DSR. The CH of the destination node will directly reply to the initiator node with the correct end-to-end path (Figure 4.14).

When a route to a destination is known then a *Message_Transfer* message is initiated. This message contains the source node, the destination node, the route that must be followed in the Mesh network in order to reach the destination and a flag that informs the destination node whether it should respond with a confirmation (acknowledgement). The *Message_Transfer* message is used for transferring upper layer data, e.g. overlay topology formulation.

4.5 UbiChord as a medium for realization of Autonomic Applications

As already mentioned at Chapter 2.1, the realization of the Internet Connected Objects Architecture can be achieved as an autonomic service. UbiChord is designed to operate without any form of centralization. This practically means that any node can participate in the DHT upon establishment of medium-level connectivity with a participant node. Joining the UbiChord DHT implies additional low-level signaling cost, than original Chord, since a node must dynamically identify its relative position in the overlay topology. Participation in the DHT implies that any node can commit (i.e. put(key, value)) and retrieve (i.e. get[key]) a set of values. We refer to set of values since multiple values may be associated with a single key.

Also these values can be committed by different nodes. Interaction of the participant nodes to this multi-value hashmap can be the basis of a brand new service development and delivery paradigm. Developers may use UbiChord as an instant storage structure where data can be consistently stored and successfully retrieved by any node in the network without broadcasts at the application layer.

4.5.1 Service Principles

Indicatively, a topology-visualization service is considered as an indicative service that requires that any node in the ad-hoc network must have the ability to obtain at any time (upon request) a valid snapshot of the network topology. In order to develop such a service without any form of centralization and without rendezvous-based solutions (sink points of aggregated data), UbiChord can be exploited. In this case, all nodes have to publish in a predefined key their current list of neighbors. The notion of "predefined key" is the cornerstone of any service development and delivery according to the proposed layered approach [P. Gouvas et al. -1, 2010]. For example, assuming that the key "topology" is the predefined key for storing and retrieving topology related data, then each $node_i$ should call the function $put_{node(i)}(key_{topology}, neighbor\_view_{node(i)})$ in order to store the current view of its neighbors. Consequently, every node that would like to have an up-to-date snapshot of the topology, has to invoke the function $get(key_{topology})$. The result would be a set such as $\{neighbor\_view_{node(i)}, neighbor\_view_{node(j)}, \ldots\}$ where $node_i$ and $node_j$ are nodes in the ad-hoc network running the service over UbiChord (see Figure 4.15).

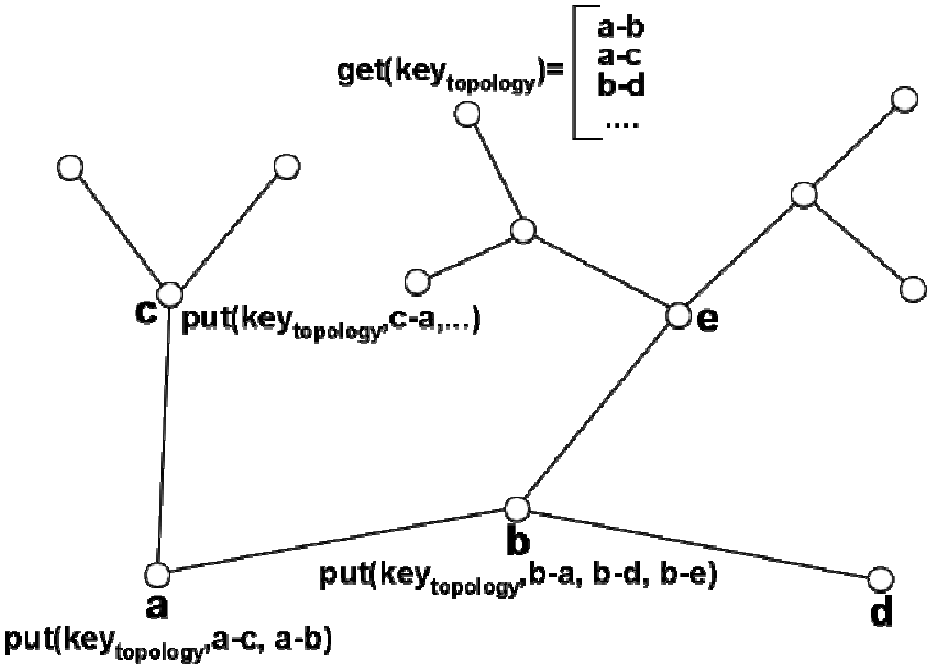


Figure 4.15 - Keys for Topology Visualization Service

UbiChord may be used following an on demand push-pull model. The application developer must take under consideration the following three aspects:

- Publication policy is regulated by the service developer and is service-dependent. Policy may be periodic or event-driven. For instance, in the aforementioned visualization scenario it is more reasonable for a node to publish its neighbor view only if a change occurs (i.e. neighbor added or removed).
- Retrieval policy is also service-dependent. It depends on how often the published value-set that exists in the DHT has to be "consumed" by the business logic of a service running on a node. Total messages required for both publishing and retrieval (i.e. pushing and pulling) have a logarithmic behavior [I. Stoica et al., 2001] on a stabilized Chord ring.
- All interactions that occur with the UbiChord structure are asynchronous. Moreover no central notion of time exists. This practically means that a stale-data prevention policy must be incorporated to ensure valid datasets in the DHT.

4.5.2 Data validity Problem

It should be stressed that stale (out-of-date) data is a two-fold problem. As global time does not exist among the participating nodes, several entries to the DHT for the same key

that are committed from the same node must be discriminated. In other words, a clear indication about which is the latest valid data must exist. Moreover, a DHT entry from one node is considered to be valid only if the node is part of the DHT. However, a node that receives a multi-value set that is already announced by many nodes, is not aware if a node that committed a key-value pair is still in the network (physical or overlay). It would be unfeasible for this node to check the presence of each node separately due to the exponential behavior of reactive routing protocols.

An elegant solution to the stale-data problem is the modification of the basic Chord algorithm so as each entry in the hashmap would require the specification of an additional volatility parameter. The volatility parameter would be initialized by the publi shing node to a default value such as $put_{node(i)}(key,\ volatility\_value)$. Volatility parameter could vary according to the key-publication policy for a specific service. In other words it would not be reasonable to initialize the volatility parameter to 10 seconds when the business logic of a service imposes re-publication of a value every 60 seconds. Upon a key-value publication, the authoritative node for the insertion of this set in the overlay, would initiate a thread per entry that would decrease -in terms of absolute time- the volatility of the key-value entry until it expires. Every key-value transfer that could occur due to overlay re-stabilization would be accompanied by the latest absolute volatility parameter. Although the existence of one thread per entry would make the DHT inappropriate for resource limited devices, it is in our future work plans to introduce this volatility parameter. In our current implementation, we use a more simplified approach for stale-data prevention based on versioning parameters that accompany the key-value entries. Specifically each key-value publication from a specific node (for a specific service) is accompanied by a version that is increasing per each key. Consequently each publication is as follows:

$$put_{node(i)}(key_{service(k)},\ [value_{t(n)},\ version_{t(n)service(k)},\ node_i])\ \ Equation\ 7$$

where $node_i$ is the publishing node, $service_k$ is the service that utilizes a specific predefined key $(key_{service(k)})$ in order to operate, $value_{t(n)}$ is the value of the specific key during publication time t_n and version $version_{t(n)service(k)}$ is an auto-increment versioning number associated with the specific key $(key_{service(k)})$. In each node that $service_k$ is bootstrapped, a publication similar to that described in (1) must be repeated containing the new value and the new version (increased by one) every time period (t_{period}). The parameter t_{period} is defined by the developer and is dependent to the business logic of the required service. For example, an instant chatting/social application has completely different needs than a network topology visualization application as far as t_{period} is concerned. In this point, it must be clarified that although t_{period} corresponds to a absolute time duration, publication time is different since there are overheads introduced by the routing layer, the topology maintenance layer and the DHT layer. Also it should be noted that the value operand of

the *put(key,value)*operation does not contain only the value of the key (for the specified service) but also the node that committed the value and the version that accompanies the commit. Consequently, the multi-value set that exists after several time periods for a specific $key_{service(k)}$ is as follows (as retrieved by $node_i$):

$$get_{node(i)}(key_{service(k)})=\{..., [value_x, version_x, node_j], [value_y, version_y, node_k], ...\} \quad Equation\ 8$$

In order for a node to identify if an entry is considered invalid it maintains a local list for every key after each retrieval (i.e. invocation of getnode(i)(keyservice(k))) that is structured as follows:

$$validList(node(i), key, service(k))=\{[Abs(max(version_{node(j)})-version_{node(i)}), value(max(version_{node(j)}))], ...\} \quad Equation\ 9$$

Consequently, the list contains the relative version that is considered as a starting bias for each node when compared with the retrieval node's version. After each retrieval a new *validList* is created (based on *Equation 9*). By calculating the derivative between two successive retrievals from the DHT, we can infer if a node is absent by setting a threshold to the acceptable range of changes.

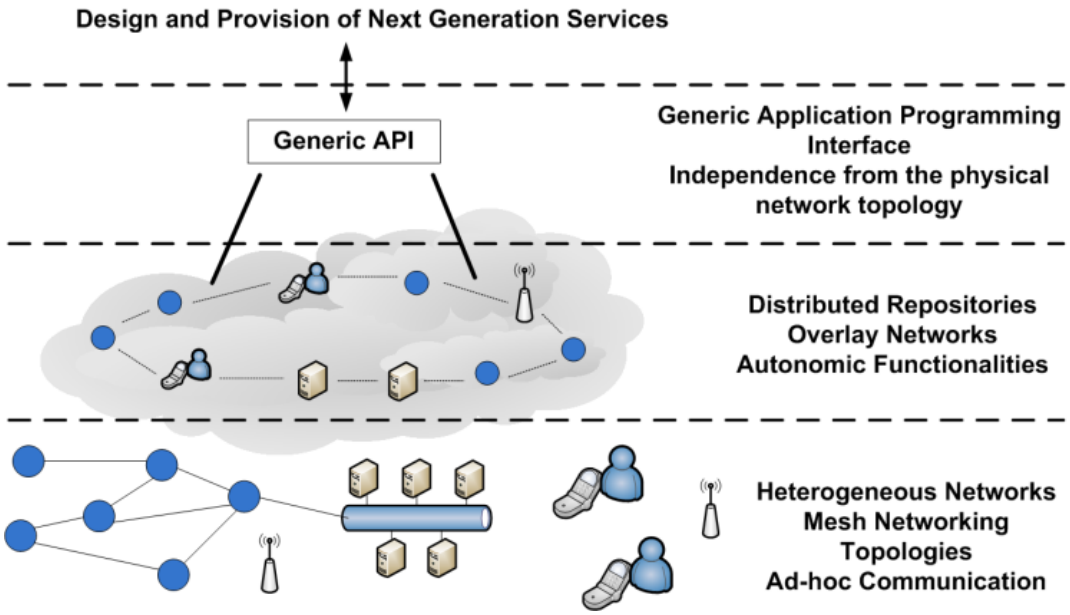


Figure 4.16 - API for next-generation of services

The usage of the aforementioned technique can guarantee valid data in a seamless way for the application developer. Conclusively, a developer can focus on the usage of a put/get API (Figure 4.17) ignoring network heterogeneity and data consistency.

5. Prototype Implementation details

5.1 Architectural Design

UbiChord implementation comprises of eight parallel asynchronous threads that operate independently in order to achieve DHT functionality (see Figure 5.1). Synchronization between threads is achieved through the concurrent-structures available at Java Framework.

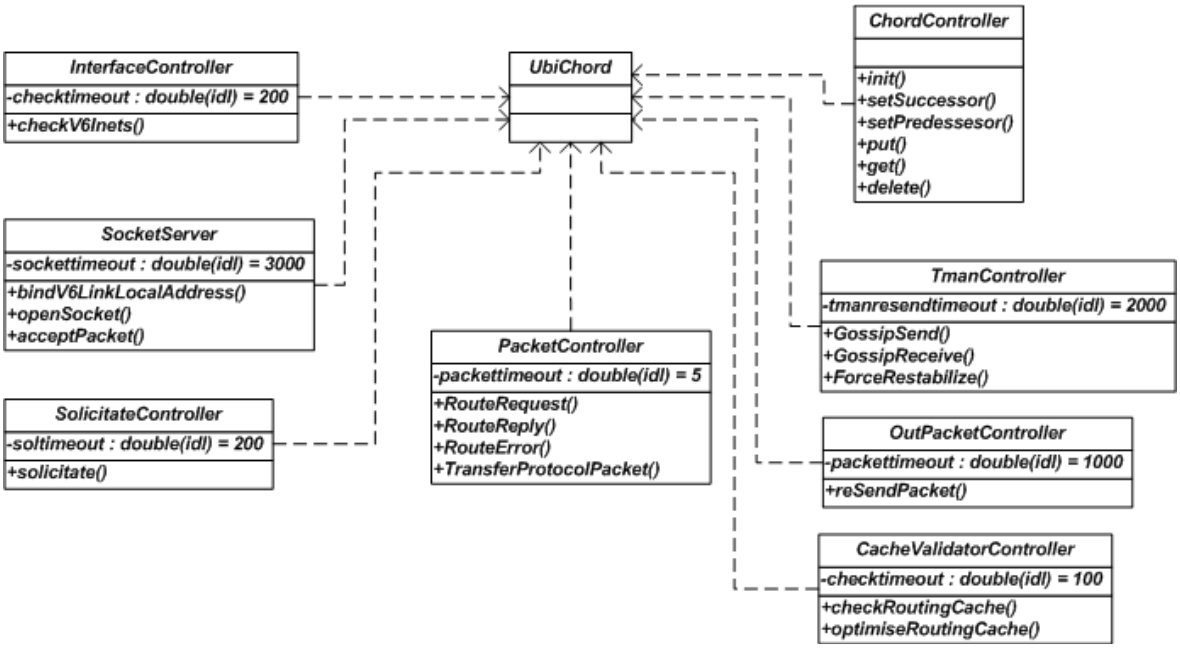


Figure 5.1 - UbiChord's threading model

In the following sections each of these threads are separately analyzed.

5.1.1 InterfaceController-Thread

The InterfaceController-Thread is responsible for monitoring the physical interfaces of the host that runs the software in order to identify if an IPv6 interface is present. The presence of an IPv6 interface is imperative since in order to achieve a zero-configuration DHT, the problem of network layer auto-configuration must be solved. It is not in the scope of UbiChord to provide a solution for this problem. Instead UbiChord relies on the robustness of IPv6 in order to confront this issue.

When a node "physically" joins a network (wired or wireless) then this node must acquire a network address that is resolvable and meaningful all across network. This is essential for direct communication between the nodes. In IPv4 based networks, IP assignment is delegated to DHCP servers that require manual configuration in order to provide IP addresses that are compatible with the address space of the subnet mask.

Beyond manual configuration, the IPv4 based assigned address is only meaningful in the specific LAN which prevents mobility of the node from one network to another network.

The incorporation of IPv6 networking in UbiChord's networking layer is imperative. It could be argued that the competitive advantage of IPv6 when compared to IPv4 is the extremely large address space, which allows a vast amount of nodes to be connected. However, the existence of large address space consists only one of IPv6' merits. Other benefits include support for mobile devices, simplified address auto-configuration, improved address management, built-in security with end-to-end IPSec etc.

In parallel, IPv6 enables more levels of hierarchy for route aggregation compared to IPv4. On the IPv6 era, billions of new devices such as cell phones, PDAs, appliances and vehicles can be IPv6 enabled; therefore, the Internet can extend its reach to billions of new users in densely populated regions of the world.

Regarding mobility, mobile access of the Internet becomes simplified to a great extent, as it incorporates a specific protocol, called Mobile IP to support mobility. In an IPv6 environment, there is support for roaming between different networks. Moreover, security is a matter of increasing concern. IPv6 incorporates Internet Protocol security (IPsec), which provides authentication, encryption, and compression of IP traffic. IPSec is a set of open standards meant for the protection of data communicated over Internet Protocol (IP) networks.

It achieves this protection through the use of cryptographic security services. IPsec supports network-level peer authentication, data origin authentication, data integrity, and data confidentiality (encryption), and replay protection. The Internet Engineering Task Force (IETF) IPsec Working Group has developed the standards for IPSEc.

Finally, regarding IPv6 adoption, the most significant feature is extensibility. IPv6 has been designed to be extensible and offers optimized support for new options and extensions. UbiChord inherits auto-configuration and mobility features of IPv6 and can even be bundled as an IPv6 extension that will facilitate the upper layers of the approach (topology creation and distributed storage) by proposing the appropriate IPv6 extension headers.

The InterfaceController-Thread checks periodically (the initial timeout is 200msec) the physical interfaces to identify if a new IPv6 enabled interface has been activated or de-activated (method: checkv6Inets()). In case of activation, the link-local address of the specific interface is propagated to the MulticastServer-Thread in order to "bind" it. The usage of link-local-address is the cornerstone of zero-configuration. A link-local address is an IP address that is intended only for communications within the segment of a local network (a link) or a point-to-point connection that a host is connected to. Link-local

addresses allow addressing hosts without using a globally-routable address prefix that must be obtained from a local or regional Internet registry.

Link-local addresses are used in stateless address auto-configuration procedures for IPv6 to assign IP addresses to network interfaces when no external, stateful mechanism of address configuration exists, such as DHCP, or another primary configuration method has failed. In IPv6, link-local addresses are always assigned, automatically or by configuration, and are required for the internal functioning of various protocol components. Link-local addresses follow the fe80::/10 notation prefix. The Data-Flow-Diagram of the InterfaceController-Thread is presented at Figure 5.2.

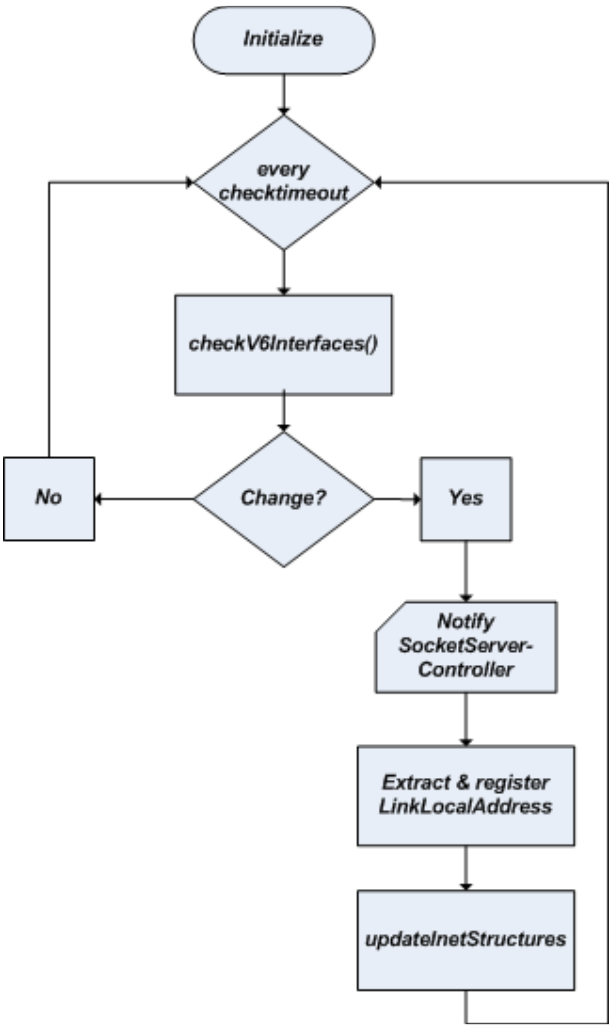


Figure 5.2 - InterfaceController flow

5.1.2 SocketServer-Thread

The SocketServer-Thread is responsible for "accepting" all packets that are transmitted in the medium. In the frames of UbiChord implementation Java Socket technology has been used. The maturity of IPv6 Socket stack allows the creation of socket-based communication among IPv6 enabled devices. In fact, during the transition period from IPv4 to IPv6 Java offers a homogenized API for both IPv4 and IPv6. In other words, an application developer can develop a socket-based application in an IP-agnostic way (see Figure 5.3).

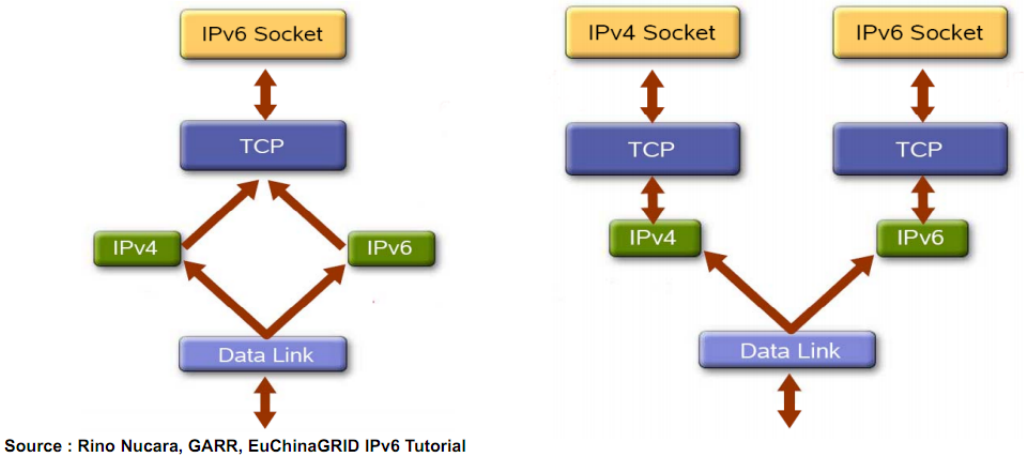


Figure 5.3 - Java's IPv4/6 homogenized API

UbiChord is fully functional, out of the box, only in IPv6 environments since it heavily relies on IP auto-configuration mechanism of IPv6. However, UbiChord can operate also in IPv4 networking environments assuming that a third party has undertaken the IP assignment. Moreover the dual-stack development guarantees that UbiChord can interact with IPv4 & IPv6 UbiChord clients (see Figure 5.4).

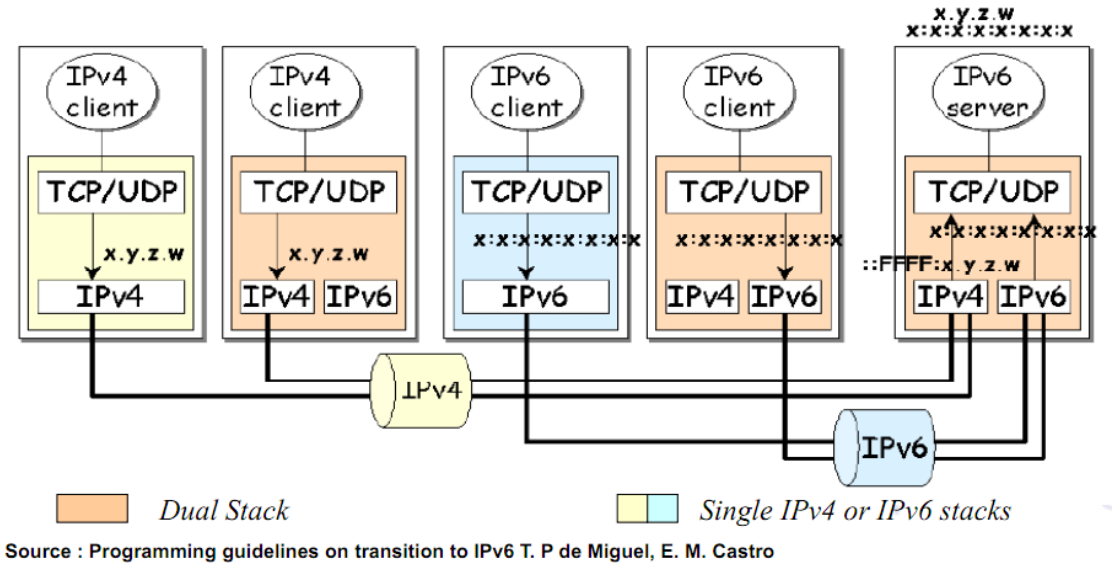


Figure 5.4 - Dual-stack socket development paradigm

The Data-Flow-Diagram for the SocketServer-Thread is presented at Figure 5.5.

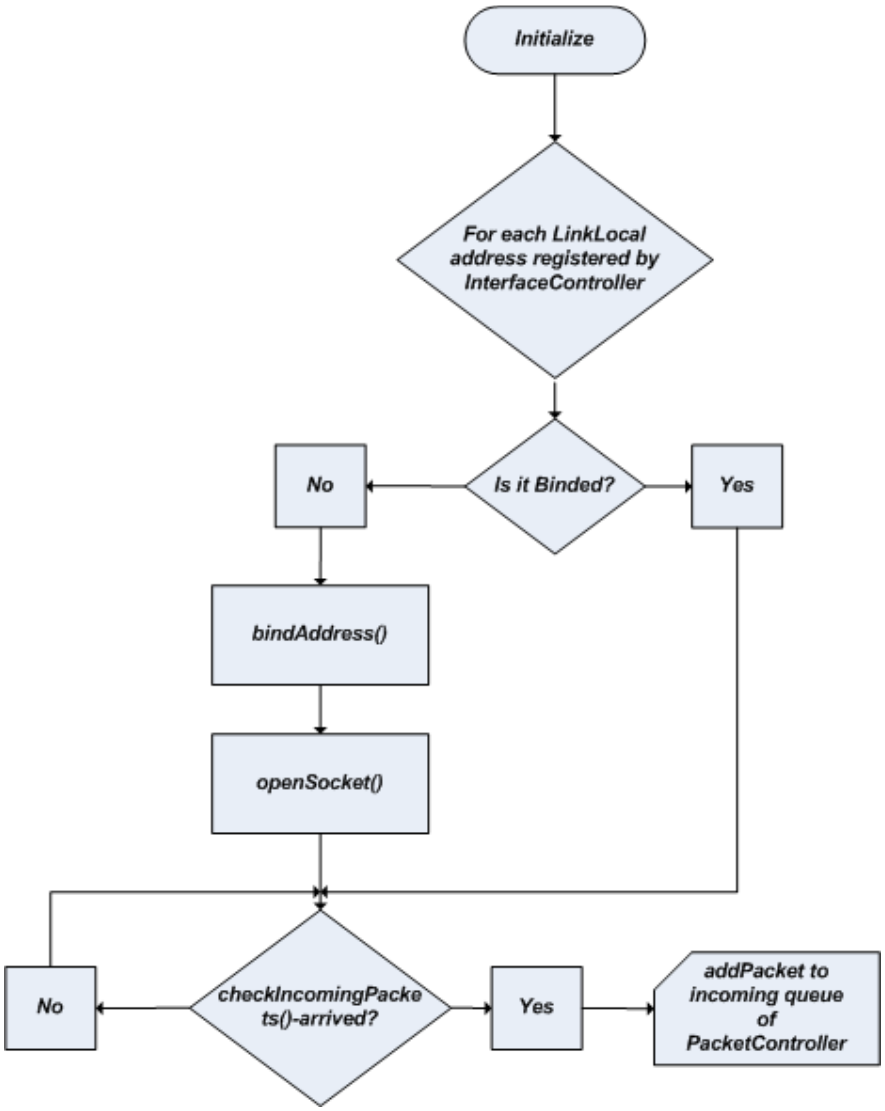


Figure 5.5 - SocketServer flow

5.1.3 SolicitateController-Thread

The SolicitateController-Thread is responsible for announcing the presence of a node to its neighbor. This mechanism can be considered as a substitute of the available Neighbor Discovery Protocol (a.k.a. NDP) that already exists in IPv6. However, since UbiChord has been developed following the dual-stack pattern a custom Neighbor discovery mechanism has been introduced. According to this mechanism every soltimeout milliseconds (initially set to 200msec) a solicitation packet is broadcasted to a default IPv4 and IPv6 group using the solicitate() method.

Neighbor identification is an extremely critical procedure because neighbors constitute the 1-hop entries of UbiChord's routing-cache as we will see in the

PacketController-Thread analysis. Moreover the addition of one neighbor may result in shortest routing paths as we will also examine at the CacheValidatorController-Thread. On the other hand, the removal of one node may invalidate a set of existing routing paths which contains the following chain-part: ... → MyNodeID → MyNeighborID → .... The Data-Flow-Diagram for the SolicitateController-Thread is presented at Figure 5.6.

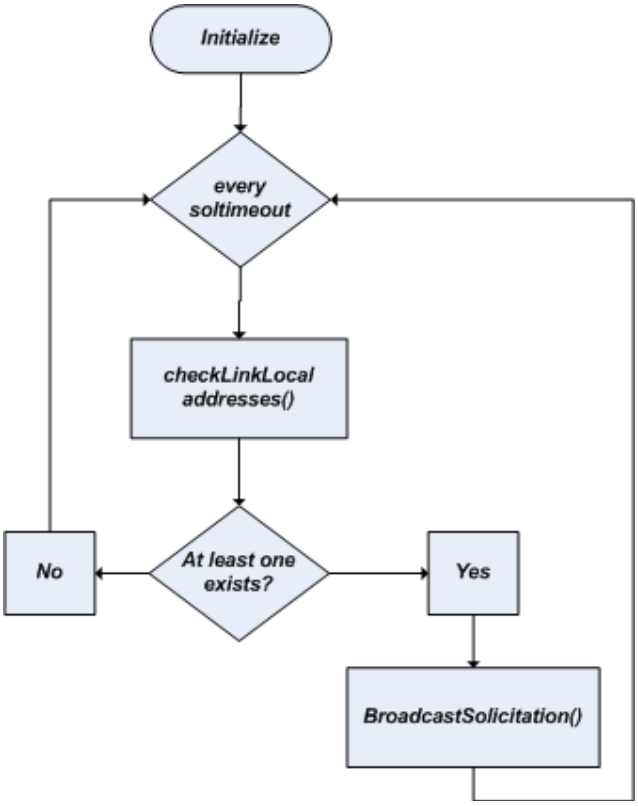


Figure 5.6 - SolicitateController flow

5.1.4 OutPacketController-Thread

The OutPacketController-Thread is responsible for sending outgoing packets and keeping track of the transfers in case of communication that requires acknowledgement (at the application layer). An out_packet_hashmap (implemented as ConcurrentHashMap) is filled by high level protocols that require the communication between two nodes. This controller examines the timestamp of an outgoing packet. If the packet has never been sent in the past and the route to the destination is unknown the normal route-request procedure is followed. After the route-discovery the packet is sent. If the application layer imposes an acknowledgement then the OutPacketController keeps track of the number of re-sends. If this number exceeds a limit the packet is dropped and the destination is considered un-reachable. The Data-Flow-Diagram of the packet queuing strategy as implemented in the frames of UbiChord is presented at Figure 5.7.

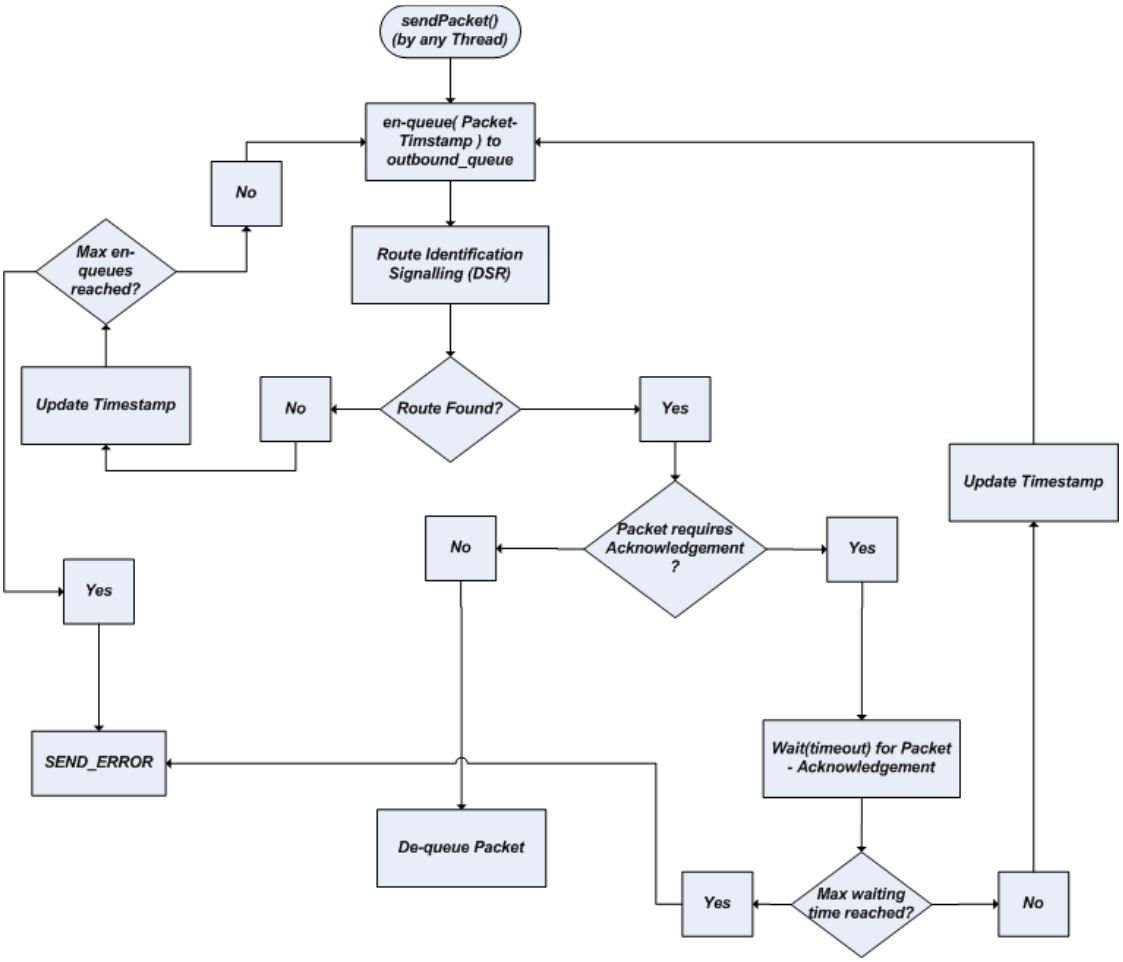


Figure 5.7 - UbiChord's outbound queuing strategy

5.1.5 PacketController-Thread

The PacketController-Thread is responsible for handling all incoming packets from the SocketServer-Thread. Every packet that arrives is persistently stored at an inpacketqueue which is instantiated as a ConcurrentLinkedQueue provided by Java 1.6. While the SocketServer-Thread is responsible for en-queuing packets, the current thread is responsible for de-queuing packets and handling them based on their protocol type. There are specific packet-types that reflect the protocols that have been implemented and thoroughly analyzed in the chapter above. In the following table the packet-object members are provided.

```
public class Packet implements Serializable {
   //Transmit types
   public static int BROADCAST = 0;
   public static int UNICAST   = 1;
   //Protocol types
   public static int SOLEXIST_TYPE  = 0;    // Neighbor Discovery protocol
   public static int ROUTE_REQUEST  = 1; // Route Discovery protocol
```

```
    public static int ROUTE_REPLY    = 2;    // Route Discovery protocol
    public static int ROUTE_ERROR    = 3;   // Route Maintenance protocol
    public static int MSG_TRANSFER   = 4;   / Route Maintenance protocol

    //Application types
    public static int PING   = 0;   //echo service
    public static int PONG = 1;  //echo service
    public static int MSG   = 2;   //Messaging
    public static int FILE    = 3;     //Messaging
    public static int SOCKET        = 4;   //Messaging
    public static int NEIGHBORGOSSIP = 5;  //Neighbor Gossiping
    public static int TMANGOSSIP     = 6;       //Node2Node Gossiping

    // init parameters
    public static int HTL = 3;

    //Type of messages
    public int  transmittype=0;        //e.g. BROADCAST-UNICAST
    public int  prottype=-1;  //e.g. SOLEXIST_TYPE-ROUTE_REQUEST
    public int  applicationtype=-1;     //e.g. PING or PONG
    public boolean requiresAck = false; //flag for Acknowledgement

    //Common attributes
    public String packetid = "";
    public    String sender;
    public String target;              //ONLY for UNICAST
    public String macsender="";        //macsender(e.g. atlas)
    public String mactarget="";        //ONLY for UNICAST messages
    public String routeHeader=""; //Used: for MessageTransfer to identify the route
    public int htl = -1;               //HTL  -1=infinite
    public long transmittimestamp;

//Protocol specific attributes
   public byte[] binmsg;              //for currying binary data
.....
```

Table 5.1 - Packet Object Members

First of all, a packet is characterized by its' transmit type. The available transmit types are BROADCAST and UNICAST. This attribute, as expected, affects the routing layer of UbiChord. Beyond transmit-type, each packet contains a protocol type. There are three types of protocol types: Neighbor-Discovery, Routing and Application-Level type. Neighbor discovery type is represented by one message-type which is SOLEXIST_TYPE while Routing type is represented by ROUTE_REQUEST, ROUTE_REPLY and ROUTE_ERROR message. Only the SolicitateController-Thread generates SOLEXIST_TYPE messages. Finally, as Application-Level type we characterize all packets that are "generated" and "consumed" by upper protocols such as Gossiping, Messaging, Ping etc.

Since there are different Packet types there is a need for different application-specific attributes. However there is a common set of attributes that are mandatory for all types of Packets. These are sender, target, macsender, mactarget, routeHeader and htl. Sender and target represent the sending node and the final destina tion node while macsender and mactarget represent the 1-hop away communication parameters during the routing

procedure. E.g. if a Gossiping Packet has to be routed from node A to B through nodes X ,Y (i.e. A → X → Y → B) then when the PacketController-Thread processes the packet at node X, the attributes of the packet would be: Transmit-type: UNICAST, Protocol-type: MSG_TRANSFER, Application-type: TMANGOSSIP, sender: A, target: B, macsender: X , macsender: Y, htl = at_leaset_3. Hops-To-Live parameter is defined initially by packet sender and can be optimised if a node is aware of the network size and density. The Data-Flow-Diagram for the PacketController-Thread is presented at Figure 5.8.

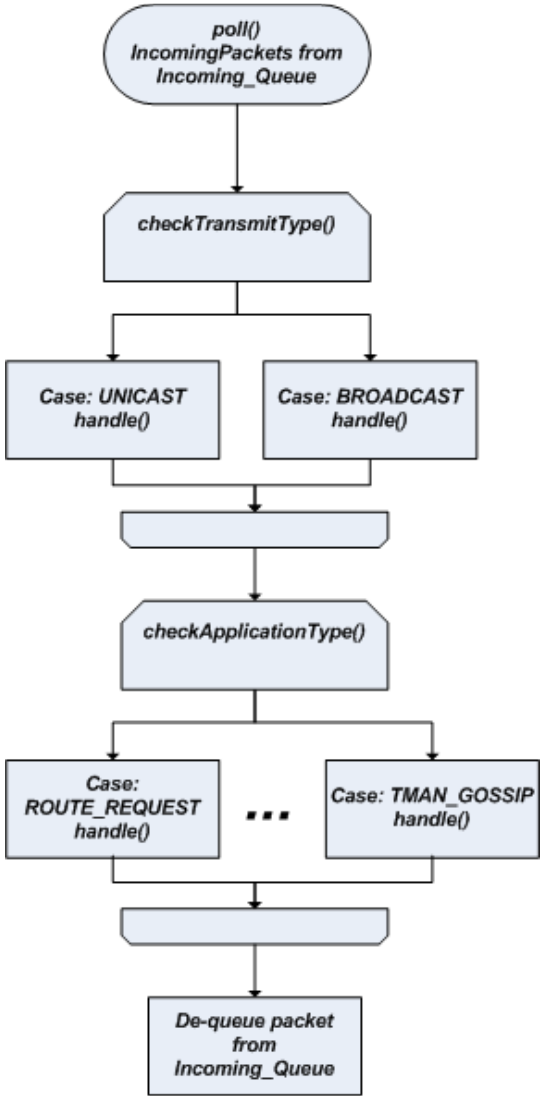


Figure 5.8 - PacketController flow

5.1.5 CacheValidatorController-Thread

The CacheValidatorController-Thread is responsible for validating the routing cache. Each node, as mentioned in the chapter above, maintains a multi-value map that contains all possible routes for a specific node. The ConcurrentHashMap Java object is used for

implementation purposes. UbiChord uses two levels of Cache; one primary and one secondary. Primary cache contains the shortest route to a node.

The roles of the CacheValidatorController are bilateral; in case of the discovery of a new node or route the RoutingCache is updated and validated. “Update” refers to the process where an alternative route of an existing destination (node) has already been identified and is shorter than the existing one. In this case the primary routing Cache has to be updated. It must clarified that “shorter” refers to hops which is not always representative. Discovery mechanisms for the “new route” vary from Route_Request announcements to Solicitation announcements.

On the other hand, when a Route_Error message is intercepted the RouteCache has to be validated again since all routes that contain the broken link have to be removed. Further revalidation is required is case of removal since some links in the secondary Cache may have to be propagated in the primary Cache. The Data-Flow-Diagram for the CacheValidatorController-Thread is presented at Figure 5.9.

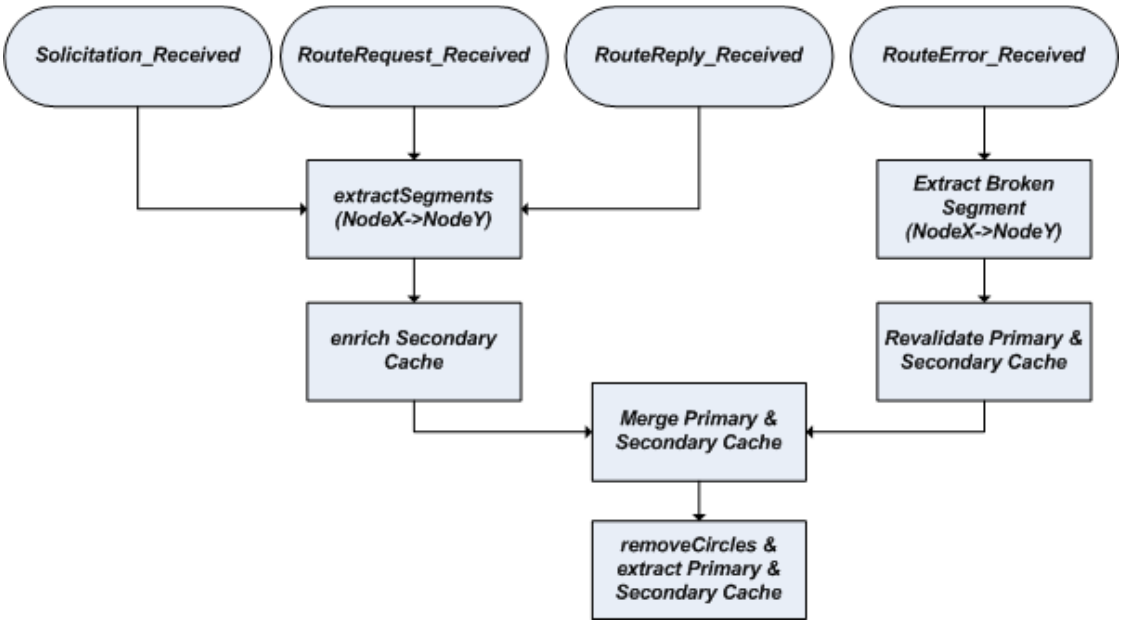


Figure 5.9 - CacheValidatorController flow

5.1.7 TmanController-Thread & ChordController-Thread

The tasks and flows of TmanController-Thread and ChordController-Thread are thoroughly described in Chapter 4.2. The development was based on an existing Chord implementation called OpenChord. The adaptations that took place in order to operate on-top of Mesh networks are: a) the bootstrapping procedure which is now based on gossiping (i.e. every node joins the overlay after its relative position is identified), b) the

asynchronous maintenance operation of figure-table-fixing that has become trigger-based (i.e. when gossiping procedure stabilizes) c) successor/predecessor maintenance that is now delegated to low-level mechanisms and d) all socket based communication is substituted by reactive routing mechanisms. The pre-existing OpenChord architecture is depicted at Figure 5.10.

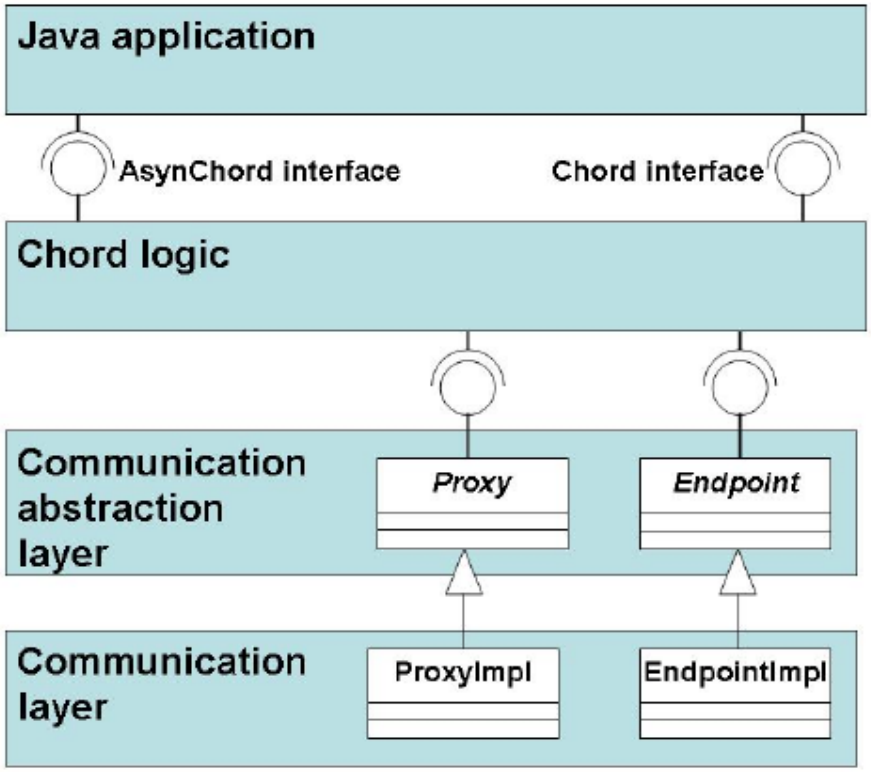


Figure 5.10 - OpenChord layering

The architecture of OpenChord is divided into three layers. On the lowest layer the implementation of the employed communication protocol based on a network communication protocol (e.g. Java Sockets) is located. On top of this communication layer a communication abstraction layer resides, that provides interfaces, which abstract from the actually used communication protocol.

For this purpose two abstract classes have been developed, which represent the communication abstraction layer, and provide factory methods, to create instances of them for a specific communication protocol. The communication abstraction layer provides interfaces for synchronous communication between peers. Instances of class represent chord.com.Proxy (Proxy) references to remote peers participating in an OpenChord overlay network. For each node in an OpenChord network an instance of chord.com.Endpoint (Endpoint) provides a connection point for remote peers with help of

Proxy for a specific communication protocol. The concrete implementations for a communication protocol are determined with help of the URL of a peer.

Based on the communication abstraction layer the logic of the Chord overlay network such as how to find the successor of a peer has been already implemented. This layer provides two interfaces to Java applications, which abstract from the implementation of routing within the Chord DHT. One interface chord.service.Chord (Chord) provides synchronous methods to retrieve, store and remove values within the DHT. The other interface chord.service.AsynChord (AsynChord) can be used for asynchronous retrieval, storage, and removal of data from the DHT. The Chord logic layer also is responsible for replication of data and maintenance of the properties that are necessary to keep the DHT running. These processes are transparent to the application using OpenChord, but can be configured. The properties to configure maintenance and replication of the Chord DHT are:

• Chord.successors: This property must be set to an integer value and represents the number of replicas that are created from a data value.

• Chord.StabilizeTask.start: This property must be set to an integer value and specifies the number of seconds to wait until the task to stabilize the Chord network is started, after Chord has been initialized.

• Chord.StabilizeTask.interval: This property specifies the timespan in seconds between successive executions of the task to stabilize the Chord network.

• Chord.FixFingerTask.start: This property must be set to an integer value and specifies the number of seconds to wait until the task to fix the routing table of a Chord peer is started, after Chord has been initialized.

• Chord.FixFingerTask.interval: This property specifies the timespan in seconds between successive executions of the task to fix the routing table of a Chord peer.

• Chord.CheckPredecessorTask.start: This property must be set to an integer value and specifies the number of seconds to wait until the task to check the predecessor of a Chord peer is started, after Chord has been initialized.

• Chord.CheckPredecessorTask.interval: This property specifies the timespan in seconds between successive executions of the task to check the predecessor of a Chord peer.

5.2 Emulation Environment

Since UbiChord is "shipped" as a standalone Java application, its extensive testing in real environments is limited to the resources of a laboratory, since no physical large scale experiments can be setup. Therefore, an emulation environment has been created to emulate large and diverse topologies. The emulation environment comprises of two software components: one "emulation filter" that is activated on-top of UbiChord and one graphical tool for topology and experiment editing. The way UbiChord is "tapped" by the emulation filter is depicted at Figure 5.11.

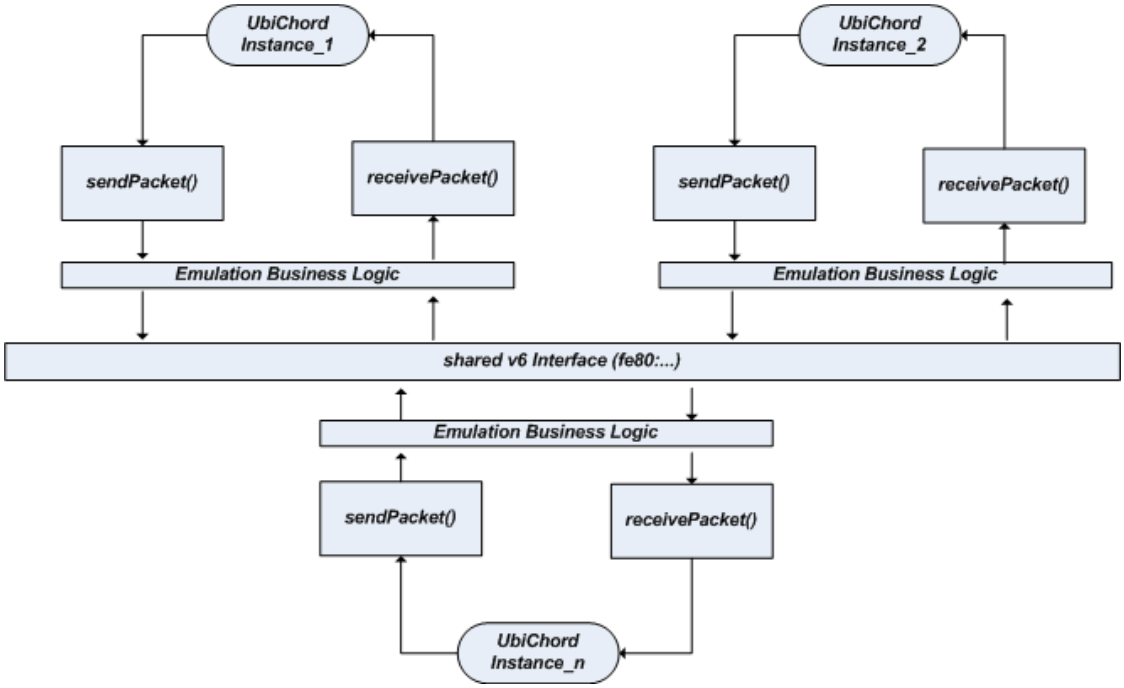


Figure 5.11 - Emulation Business Logic

More specifically, the filter is applied at the MulticastServer-Thread. During UbiChord's bootstrapping a specific variable (i.e. *boolean is_emulation_mode_activated*) indicates whether the emulation filter should be applied or not. If the filter is applied, every packet that is transmitted is filtered based on the source and destination address. The filtering business logic is based on a topology file which is common for all nodes and provides information per each node about which node is considered to be a neighbor.

The topology file can be created manually, however a specific interface has been developed in order to create and manage these topology-files. The interface of the UI tool is depicted at Figure 5.12.

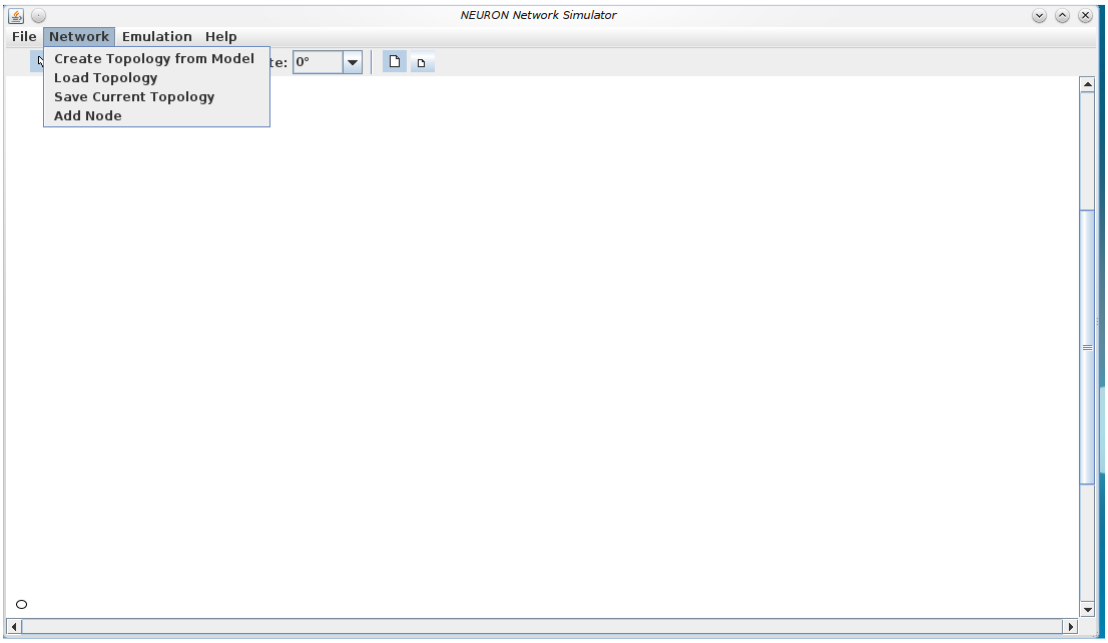


Figure 5.12 - Emulation Environment User Interface

Through the "Network" menu-item the user can create a topology, load an existing topology for editing, save an edited topology and add an additional virtual node to a topology. However, from the research perspective, there are specific topologies that are interesting such as circular topologies, social-networking topologies etc.

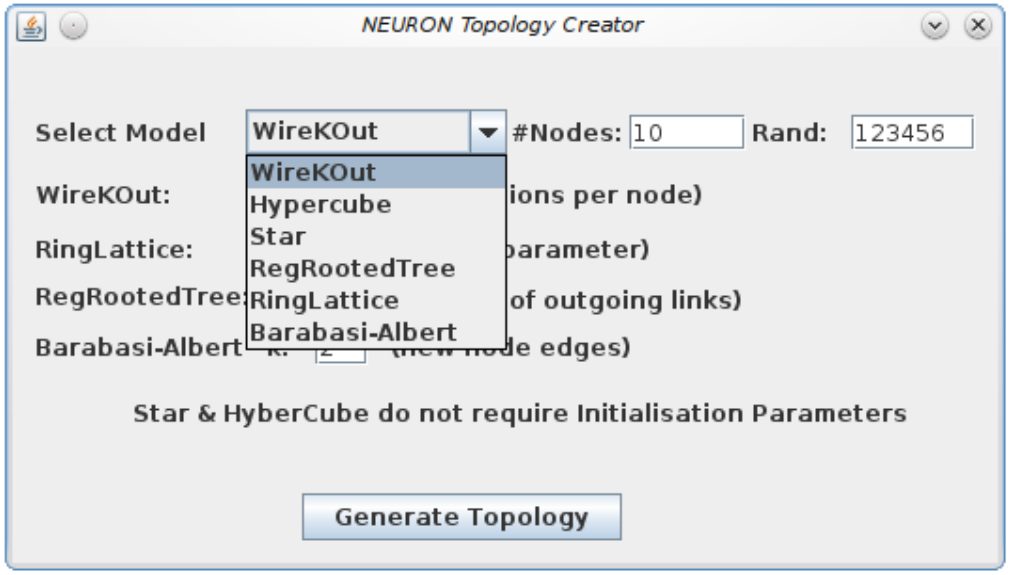


Figure 5.13 - Topology creation form

In order to increase the level of automation for the topology creation a topology-generation module has been developed and integrated in the Emulation Environment. Through the "Emulation" menu-item specific overlays are automatically generated based

on configuration parameters that are graphically prompted as illustrated at Figure 5.13. An auto-generated hypercube-overlay initialized for 12 nodes is depicted at Figure 5.14.

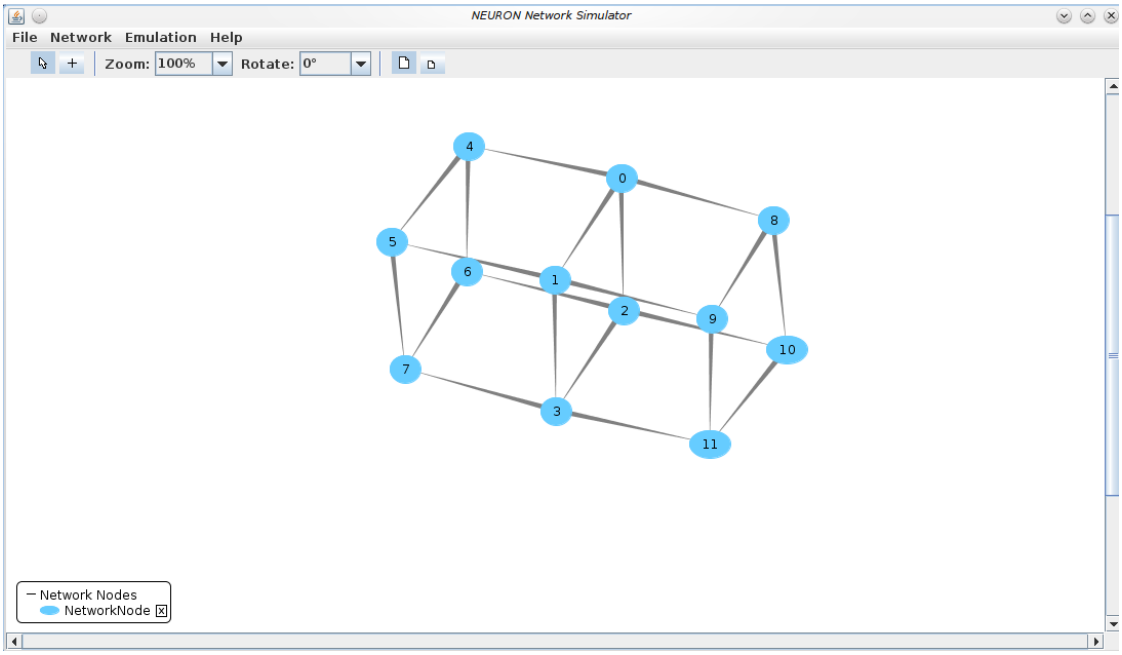


Figure 5.14 - Hypercube topology visualized

A user can change the visualization mode of the auto-generated topology. Furthermore, the topology can be edited through the addition or removal of nodes. However, an emulation experiment requires more input-parameters than the initial topology. Additional mandatory parameters are the duration of the experiment and the Node_failure_rate. These mandatory parameters can be inserted through the "Create Experiment" menu-item as illustrated at Figure 5.15.

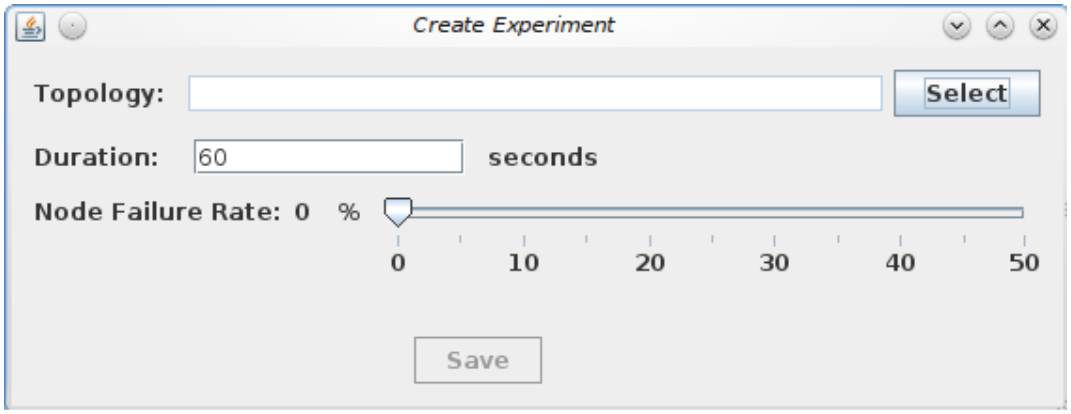


Figure 5.15 - Experiment creation

Moreover, these mandatory parameters are essential for the static part of an experiment. There is also a dynamic part of configuration which affects the application-level events that have to be generated e.g. dhtput(v1,k1) at time: 10s from: node1. These application-level triggers are also included in an experiment descriptor (i.e. a file with .experiment suffix). It must be highlighted that application-level triggers can only be manually injected in the .experiment file since no UI has been developed for such triggers.

When a node bootstraps a Console Shell is available to the end-user in order to monitor and control UbiChord's functionality (Figure 5.16).

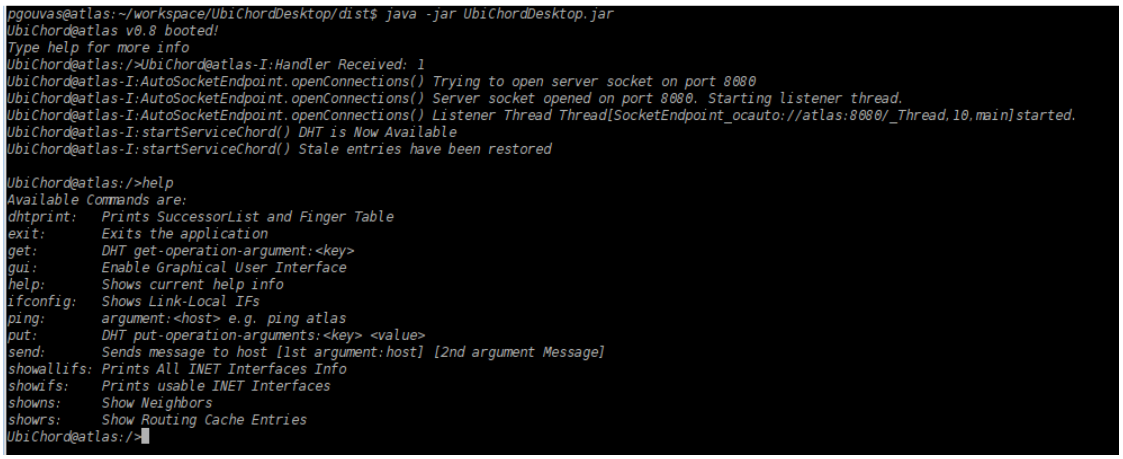


Figure 5.16 - Console menu

The DHT-related available choices are:

- put: inserts in the DHT a key-value pair,
- get: retrieves a value based on a key,
- dhtprint: it prints the successor list and the finger-table of the DHT.

6. Simulation & Emulation Results

6.1 Simulation Results

In this section we investigate the behavior of the proposed implementation, as it is described in Section 4.3. Each layer that was presented -except Neighbor-to-Neighbor layer- was implemented separately in PeerSim [M. Jelacity et al., 2010]. We assessed the performance of each layer in terms of messages exchanged, generated errors and convergence capability in diverse network sizes. Each simulation was executed multiple times and average values were considered in our analysis.

6.1.1 Layered Approach Performance

In the following scenarios, multiple nodes are simultaneously activated in the simulated ad-hoc network. Key parameters of the topology are the number of nodes (N) (size of the network) and the average number of neighbors (d) (density degree). In all simulations, any operational node automatically recognizes its neighbors and joins the overlay network. Each node keeps state related to its successor in the ring topology. In case of node failures, established connections are torn down and re-stabilization procedures take place.

Initially, we compare the performance of the proposed adaptations in terms of total number of messages exchanged until the overlay ring topology is formulated. Simulations are taken for practically sized and degree (density) networks. The Hops-To-Live parameter in DSR protocol was set to infinite and optimum respectively.

In Figure 6.1, the performance of the proposed approach with the use of the adapted DSR compared to the pure DSR is depicted. It is evident that the use of pure DSR underperforms significantly compared to the adapted DSR. This is reasonable, since the exploitation of routing cache information –generated by DSR– in adapted T-MAN makes easier the identification of the successor nodes in topology formulation phase and thus requires less messages.

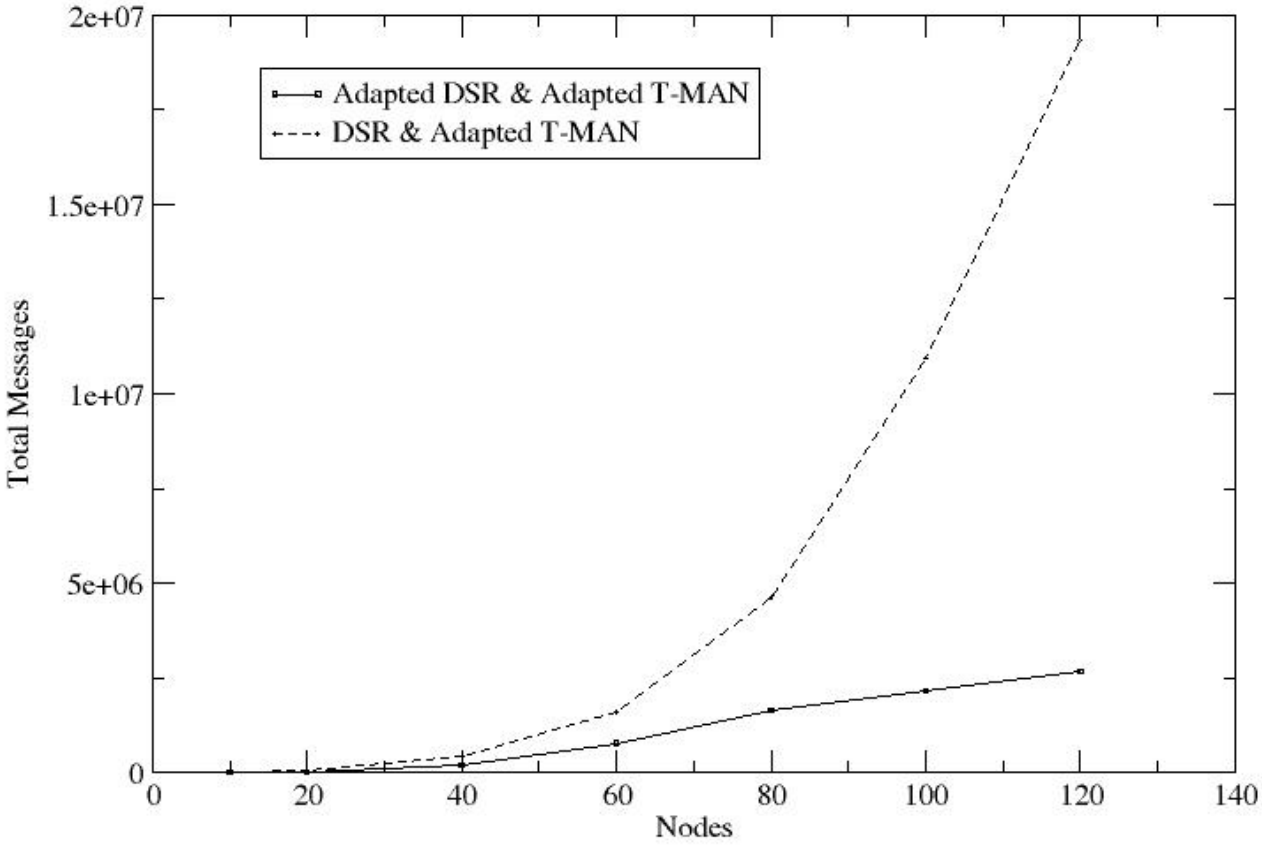


Figure 6.1 - Adapted DSR VS DSR messages

At this point, it is important to note that the selection of T-MAN instead of adapted T-MAN was not considered a priori as a viable solution, since T-MAN is not able to converge faster than the adapted T-MAN. This is due to the fact that T-MAN does not consult the existing routing caches of the nodes and therefore more cycles are required for overlay topology stabilization [M. Jelacity-1 et al., 2005].

In Figure 6.2, we compare the performance of adapted T-MAN and exhaustive tree-based approach. It can be shown that the number of messages for topology formulation in the tree-based approach is significant larger compared to the adapted T-MAN. This is due to the fact that in the tree-based approach, exhaustive neighbor to neighbor communication takes place and also that in the adapted T-MAN approach the information, that is available in the nodes' routing caches is exploited.

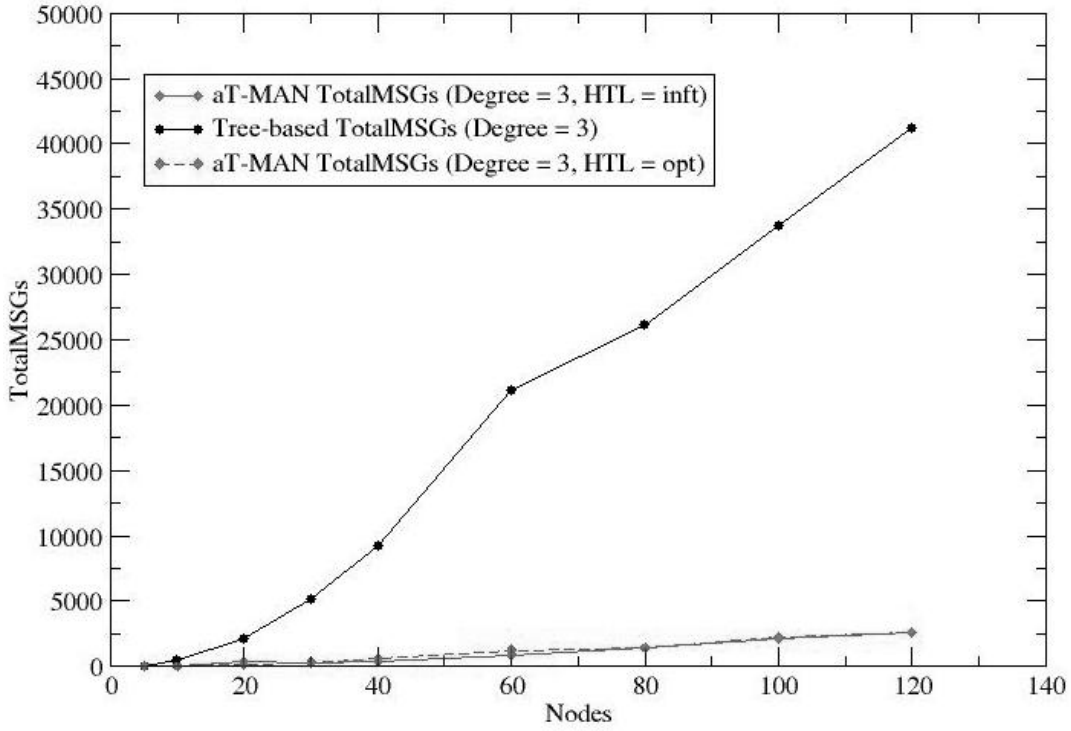


Figure 6.2 - Adapted T-MAN VS tree-based messages

These simulation results led us to focus on the adapted T-MAN algorithm and further investigate the impact of other parameters, such as the routing HTL parameter or the network degree in the overlay topology formulation and maintenance process. In Figure 6.3 we can observe that as the number of nodes in the network increases, the number of adapted T-MAN messages also linearly increases. In addition, fewer messages are required for topology formulation in dense networks (high degree) as each node exchanges more precise gossiping messages. Precision is achieved due to the scoring of larger number of neighbors. Finally, the number of messages remains almost stable in large degree networks.

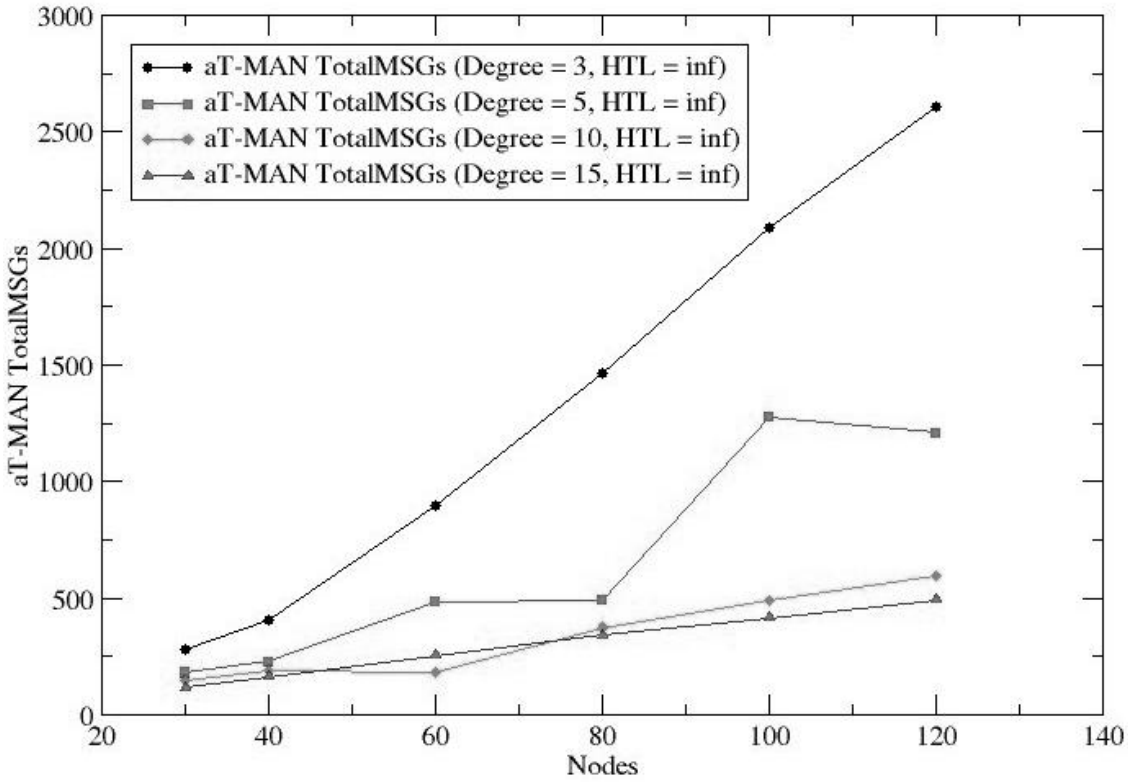


Figure 6.3 - Adapted T-MAN performance with HTL infinite

In Figure 6.4, the effect of the Hops-To-Live (HTL) parameter to the total number of adapted T-MAN messages is depicted. In case the HTL parameter is set to the optimal value according to the Equation 2 in Section 4.1, the number of T-MAN messages is slightly increased compared to the previous case. As expected, by limiting the HTL parameter we reduce the flooding of routing messages across the network -to the absolute minimum for addressing routing needs- and, thus, smaller routing caches in each node are exploited by adapted T-MAN stabilization phase.

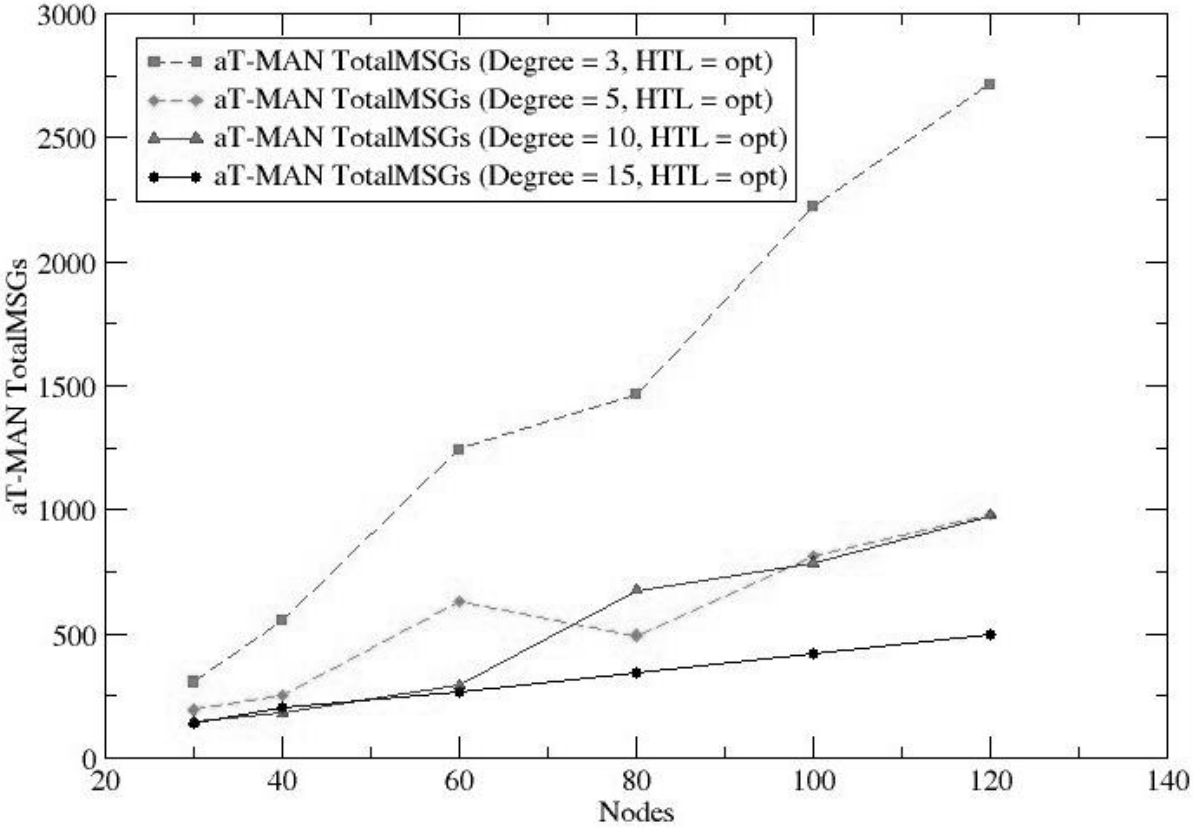


Figure 6.4 - Adapted T-MAN performance with HTL optimum

Furthermore, we aim to observe the amount of DSR's route-request messages that is exchanged until the overlay ring topology is formulated. In Figure 6.5 and 6.6, the HTL parameter is set to infinite and to optimum respectively. As shown in both figures, more route-request messages are generated as the size of the network increases. This is expected as DSR messages are generated in response to T-MAN messages, which are analogous to the number of nodes (Figure 6.3 & 6.4). Furthermore, sparse networks require more routing messages than dense networks, especially as the number of nodes increase due to smaller number of entries generated via neighbor-exchanges. The adapted T-MAN affects the flooding of routing messages in the network and, thus, the routing overhead is significantly reduced by approximately an order of magnitude.

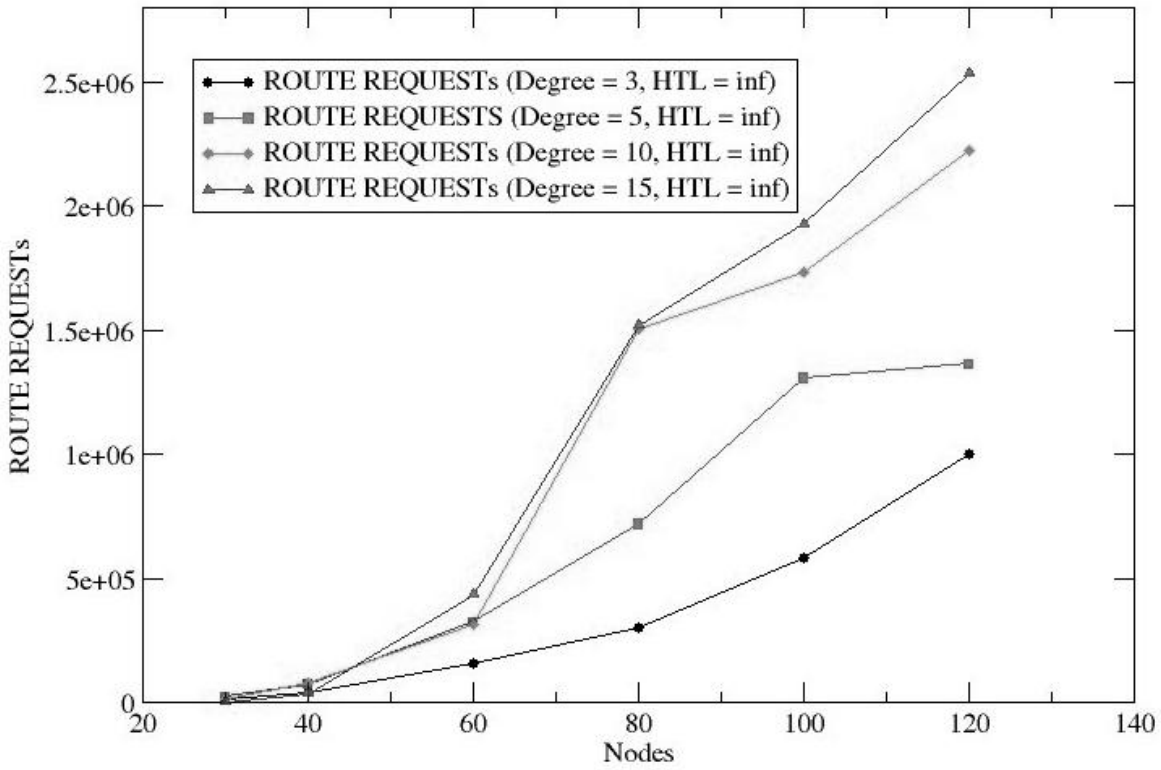


Figure 6.5 - DSR Route Requests with HTL infinite

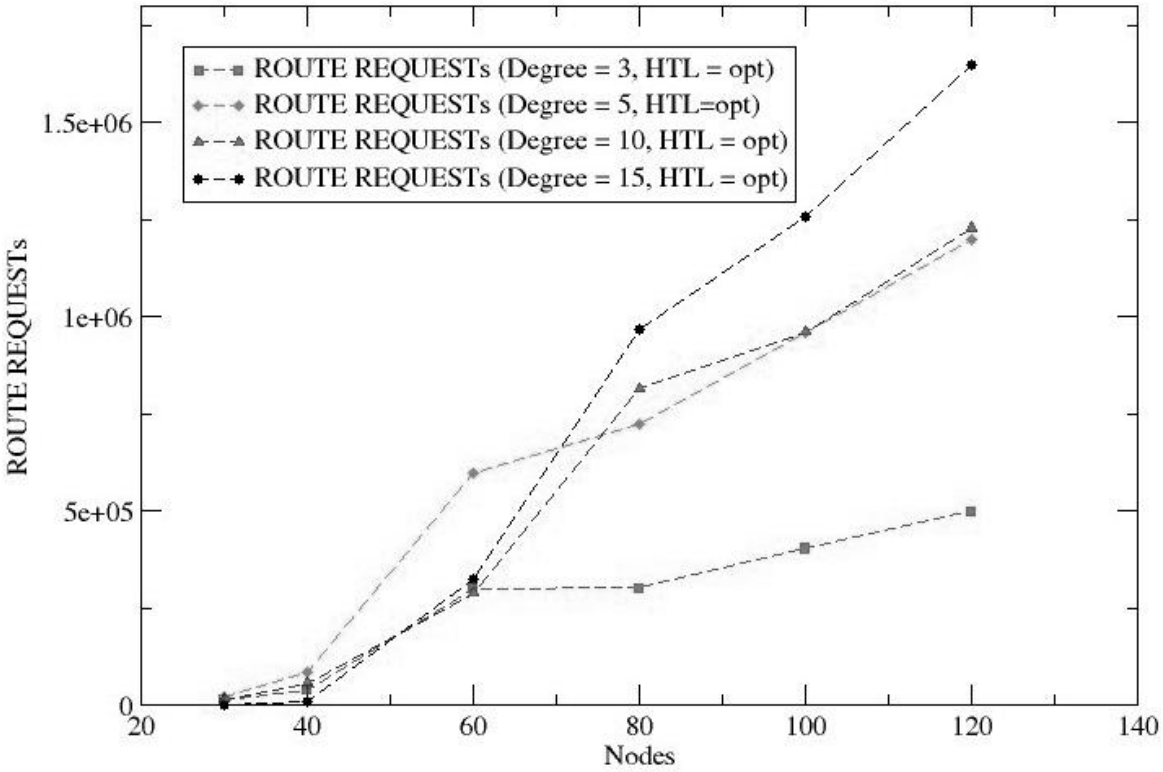


Figure 6.6 - DSR Route Requests with HTL optimum

Thus, we conclude that the effect of the HTL parameter is extremely important because, when the parameter is set to its theoretical optimum value, adapted T-MAN messages were slightly increased while in parallel the routing overhead was reduced by an order of magnitude.

Figure 6.7 depicts the number of adapted T-MAN messages for different network sizes and variable density. An important observation is that the number of messages converges quickly for different network density degrees. This observation is further exploited in building clustering algorithms, where the network density is used to estimate the best clustering size.

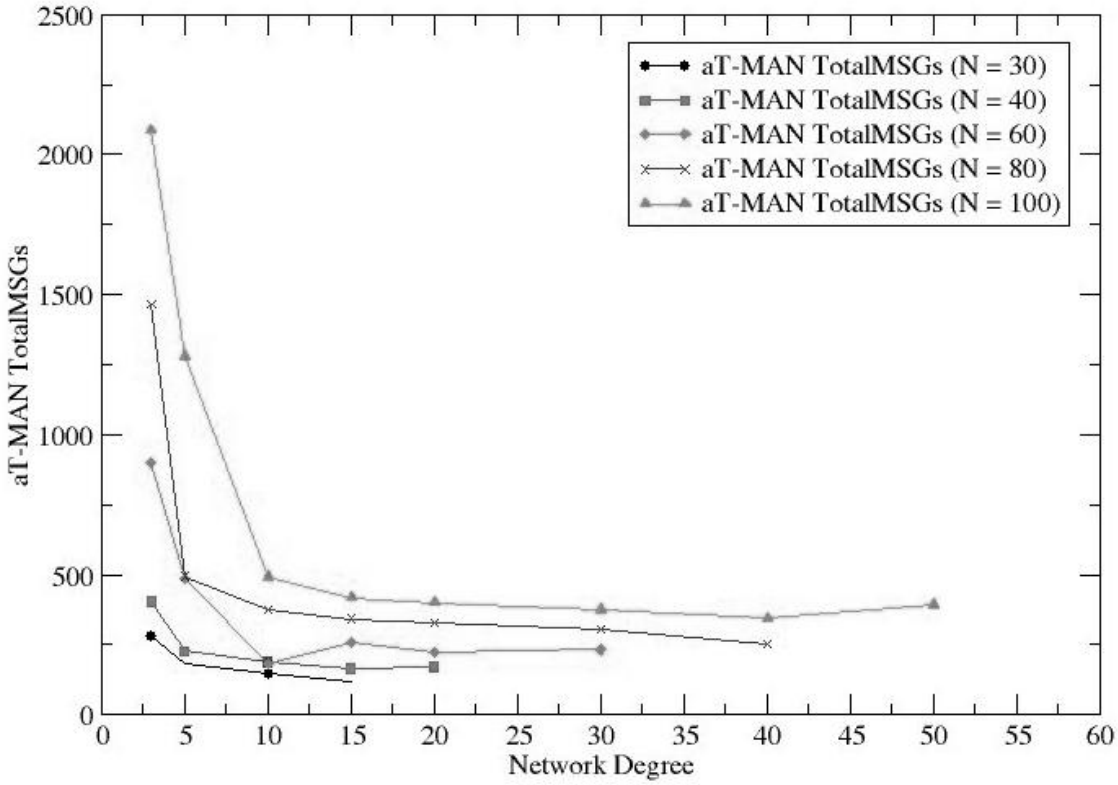


Figure 6.7 - Adapted T-MAN messages for various sizes and densities

6.1.2 Indicative Service Bootstrapping

Upon stabilization of the overlay ring, an indicative visualization service is provided. Each node stores topology data that is available through information acquired from neighbor discovery messages and also from information existing in the routing cache of each node. All nodes use a predefined key (e.g. "network_topology") in order to identify the responsible node for the storage of the visualization data. The visualization cost, i.e. the total number of messages exchanged for the visualization data to be stored, is calculated.

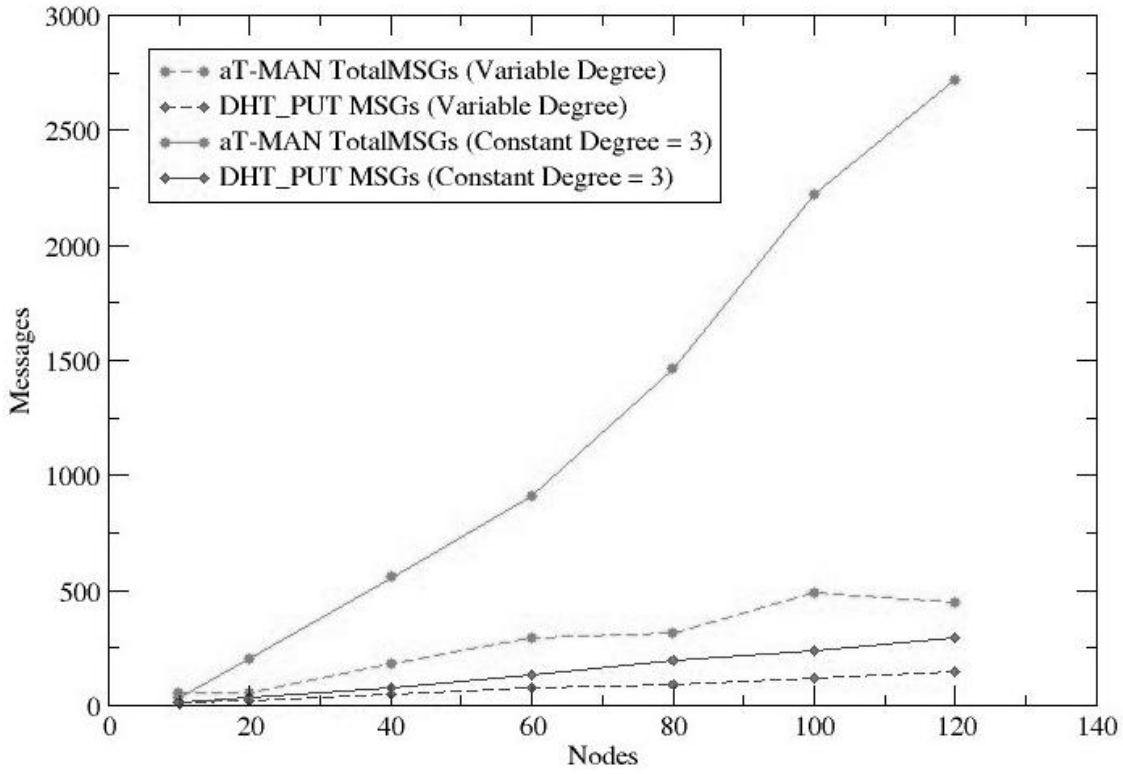


Figure 6.8 - Visualization Cost

Figure 6.8 depicts the number of adapted T-MAN and DHT_PUT messages for constant and variable network density. In the first case, the network expanses without any change in its density while in the latter case the density is proportional increased as the network sizes increases. We notice that the number of DHT_PUT messages required for the provision of the visualization service, is slightly affected by the network size and density. It is important to note that in case that the proposed adaptations in the T-MAN and DSR algorithm were not applied, the visualization cost would be significantly higher, as already shown in Figure 6.1.

In the last set of simulations, we investigated whether the visualization service is robust to multiple node failures. Under the unfavorable condition that 20% - 30% of

network nodes become simultaneously non-operational and for various network sizes, we measure the number of DHT put(key, value) calls that generate *DSR_Route_Error* messages for any broken link identified. Such messages are flooded across the network in order to update the routing cache for future requests (see Section 4.3.1). As shown in Figure 6.9, the majority of the put(key, value) requests are successfully completed without any delay (error). For example, for a 40-node network and 30% node failure, 17 messages were successful while another 10 failed. Even if a put(key, value) request initially fails due to routing inconsistencies, the requests will be eventually be successful after the routing tables are updated. Therefore, there is only a time penalty for the service delivery after a major network failure. It could be argued, therefore, that the visualization service remains operational even when a significant portion of nodes (and attached links) fails. Similar trends should be noticed for other distributed services, which have to be verified with further simulation experiments.

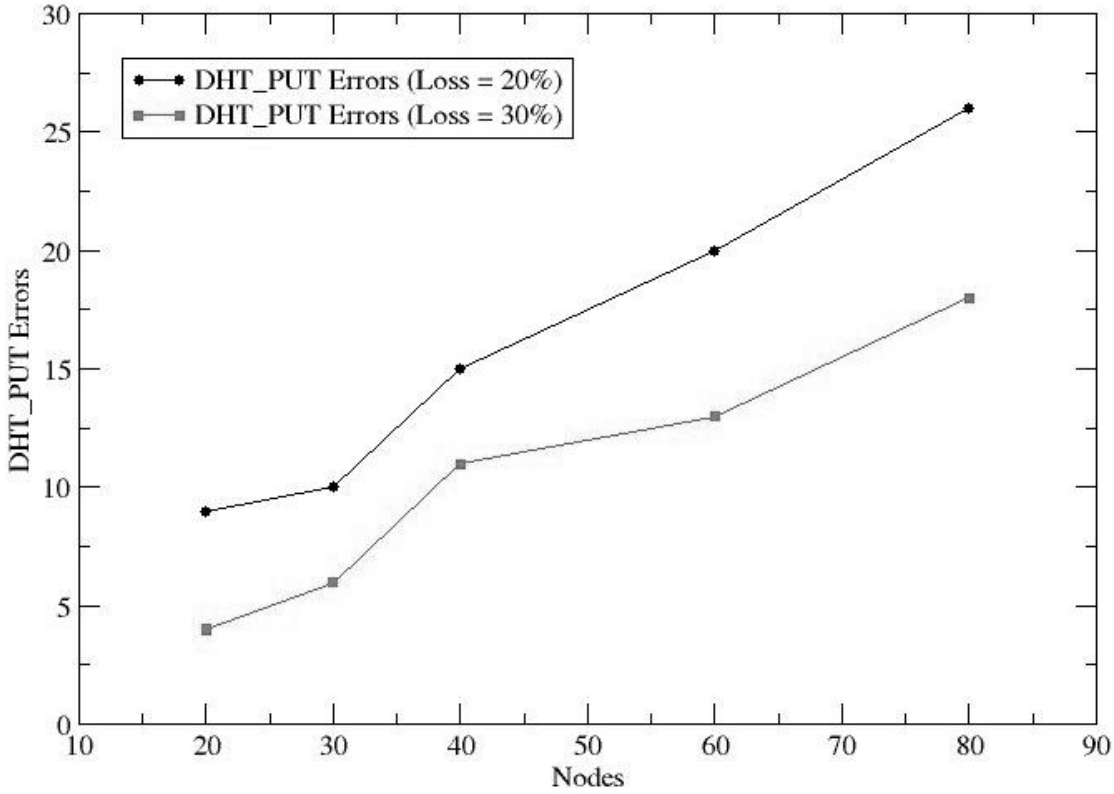


Figure 6.9 - DHT_PUT failures

6.1.3 Simulation of Optimization Mechanisms – NEURON evaluation

In this section the performance of NEURON for a wide set of topologies is evaluated. NEURON has also been developed in the Peersim simulator. A visualization module is also developed as a Peersim extension that provides a view of the clusters with their CHs that are formulated in the Mesh network at each cycle period of the simulation. In order to simulate the limited resources of the participant nodes of the Mesh network, a custom dynamic model is incorporated that imposes penalties according to the nodes operations.

In the simulations multiple nodes are simultaneously activated without any preconfigured state information. Each simulation lasts 2,000 cycles, while every node is initialized with 100,000 battery units and 1,000 routing cache memory units. Battery and memory penalties are defined for serving each message in the network. Each entry in the routing cache occupies one memory unit, each packet transmission or reception drains the available battery by three units, while each packet processing action (e.g. protocol encapsulation) that is accomplished by a node drains the battery by one unit. The periodic broadcasting of *MSolicitateCH* messages is set to five cycles, the KPI threshold (see Section 4.4.2) for transition to a new CH is set to two, and the threshold where a CH switches to normal mode is set to 50%. All the nodes are considered with equal battery and memory capabilities at their initial deployment. The number of nodes varies from 50 to 12,800 while the density varies from 3 to 36.

The performance of each mechanism is assessed using multiple criteria, such as messages exchanged for the operation in steady state, convergence capability, precision in the estimation of parameters, behaviour of the probabilistic techniques, energy efficiency and quality of distribution of CHs in diverse network sizes and densities. Simulation results that are related with the creation of the overlay network and NEURON's suitability for deployment of advanced services in the Mesh network are also presented. Each simulation is executed five times and average values are considered in our analysis.

6.1.3.1 Evaluation of the Autonomic Estimation Algorithm

The Autonomic Estimation Algorithm, aiming to estimate two network parameters, is activated right after the mobile nodes become operational. In each node, the algorithm converges if the estimated parameters' values, between two consecutive cycles, are less than 5%.

Figure 6.10(a) shows the number of messages that are exchanged until the algorithm converges to the estimation of the density and the size parameter for various sizes and densities. Figure 6.10(b) shows how these messages are distributed between nodes. There is a linear relationship between the total number of messages exchanged and the network size. It derives that the algorithm convergences without imposing significant overhead since the average number of messages per node remains small and stable even for large-scale networks. The autonomic estimation process may be repeated periodically in predefined number of cycles, related with the dynamicity that is present in the Mesh network.

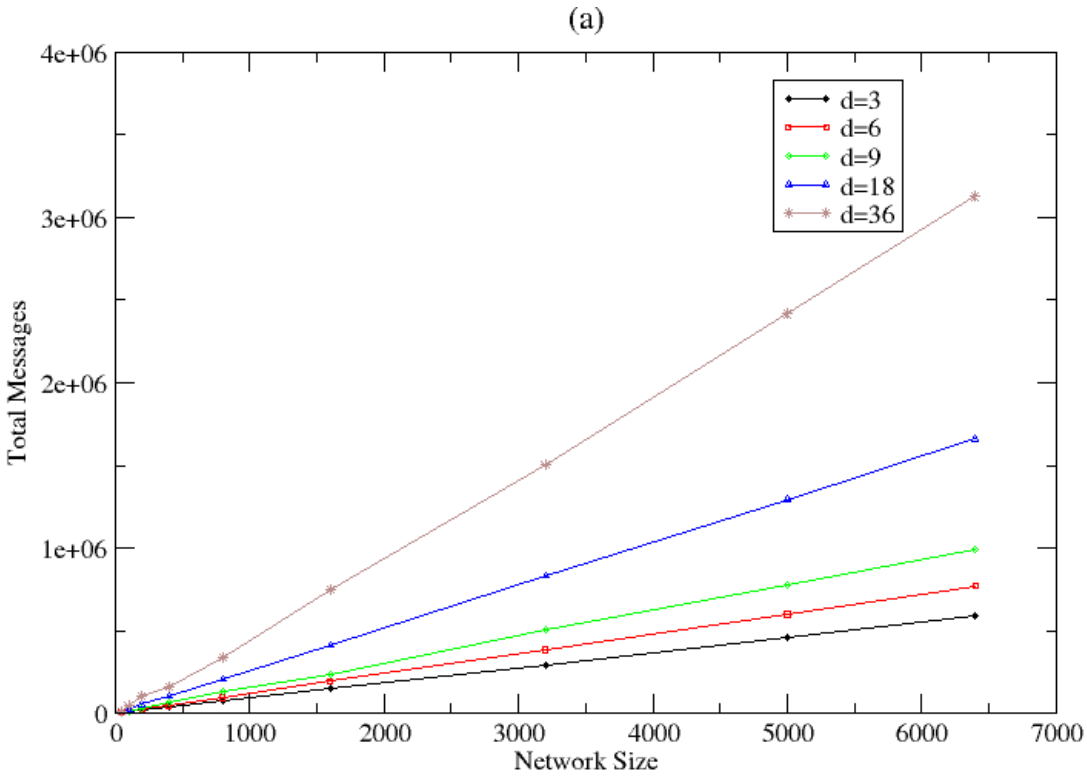


Figure 6.10(a) - Messages for Network Density and Size Estimation

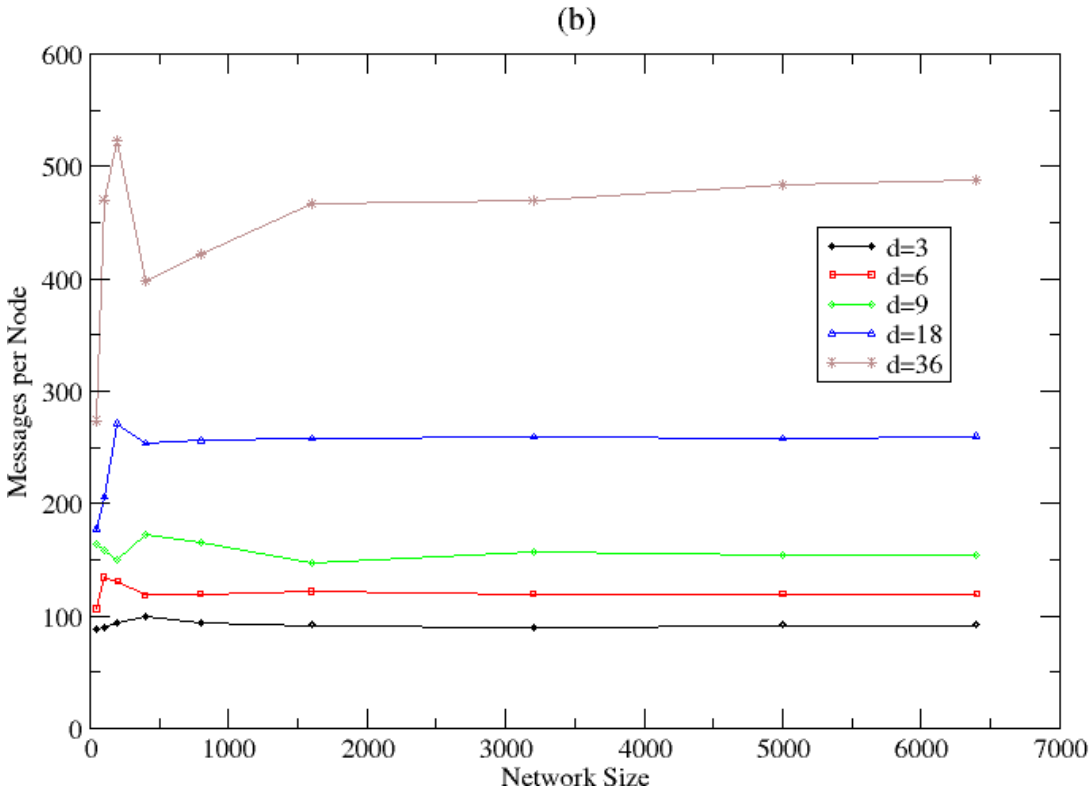


Figure 6.10(b) - Messages for Network Density and Size Estimation per node

Figure 6.11 presents the cycles that are necessary for the algorithm to converge. Stable behavior is achieved for each network density independently from the network size. The algorithm converges in less than 20 cycles in sparse networks and in less than 10 cycles in dense networks, independently of the network size. In dense networks, convergence is faster since more messages are exchanged at each cycle.

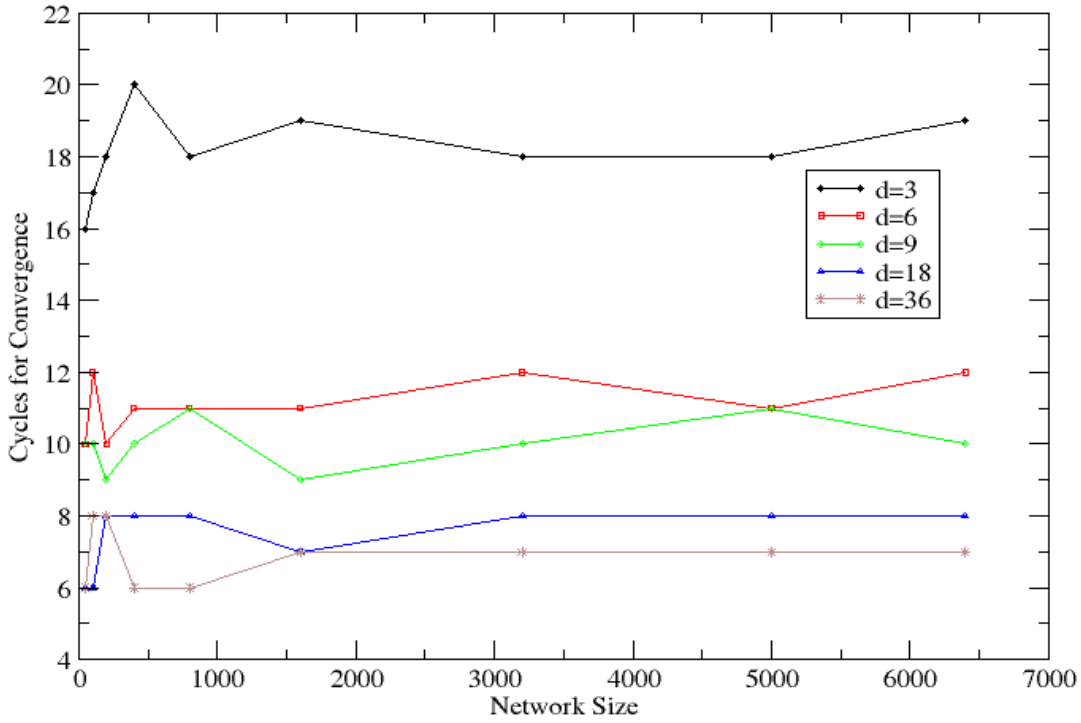


Figure 6.11 - Cycles for convergence of the estimated parameters

The Autonomic Estimation Algorithm achieves adequate precision for the estimation of the network parameters, as presented in Figure 6-12(a) for the density estimation and Figure 6-12(b) for the size estimation. For the density estimation, the deviation from the real values is less than 9% in all cases and remains approximately constant for a given network density. For the size estimation, the deviation is less than 20% for sparse netw orks and less than 10% in dense networks. In both cases, higher precision is noticed in dense networks since averaging is performed between multiple neighbors in each cycle.

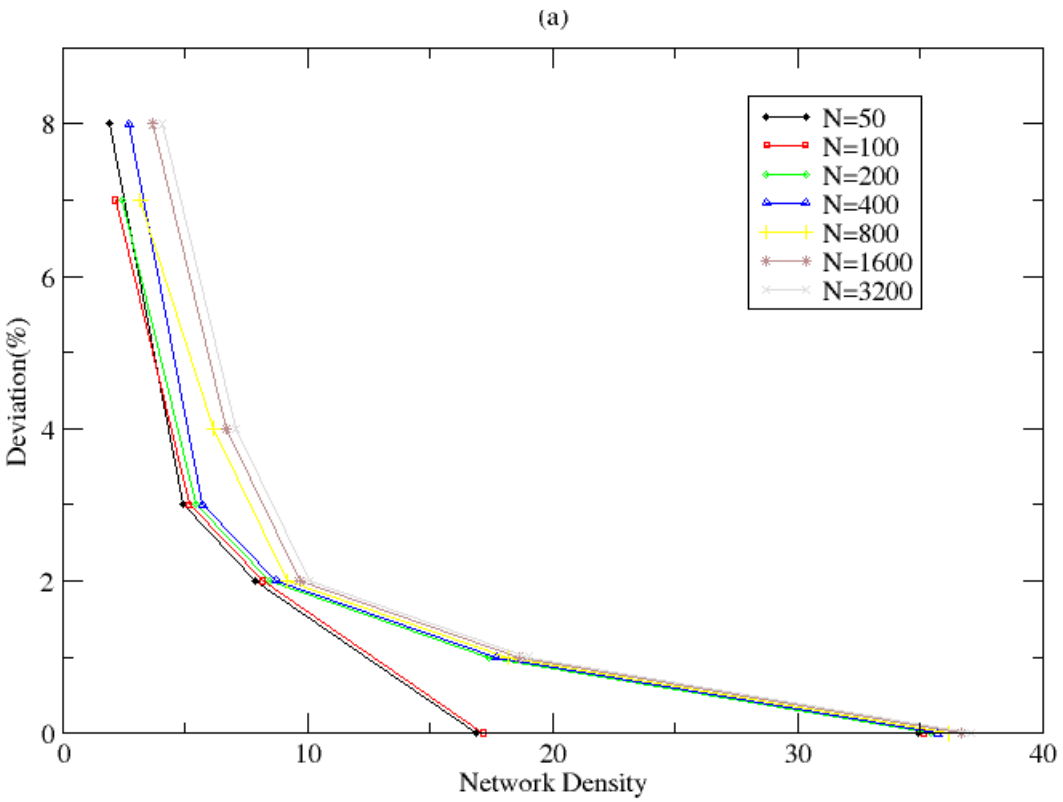


Figure 6.12(a) - Deviation in the estimation of density

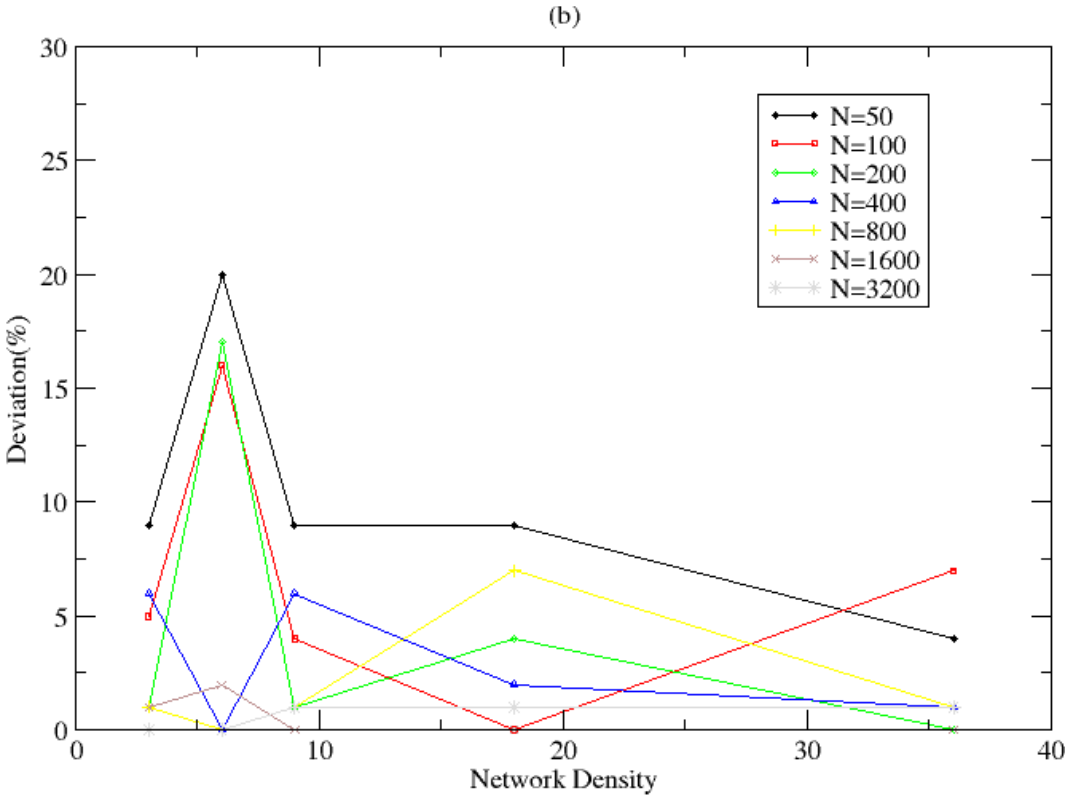


Figure 6.12(b) - Deviation in the estimation of the size of the Mesh

6.1.3.2. Clustering and Routing Mechanism Evaluation

A visualization module is developed for the dynamic illustration of the clustering process and the distribution of the CHs within the WSN. It is noticed that the clusters' distribution improves (qualitative metric) over time since the probabilistic techniques used tend to homogenize the size and form of the created clusters and thus distribute the clustering overhead among the elected CHs. Two indicative screenshots are presented in Figure 6.13 where clusters are distinguished.

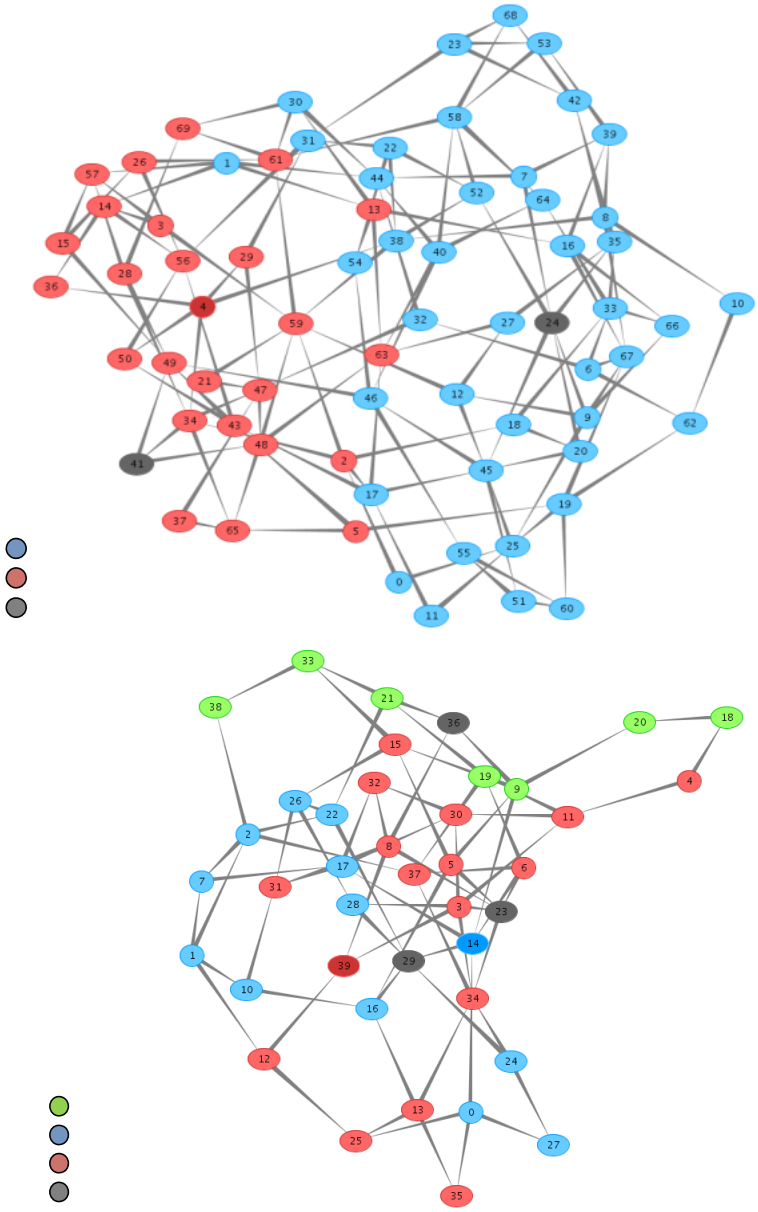


Figure 6.13 - Cluster Visualization

In Figure 6.14, the comparison of the number of CHs that are elected in the simulation environment with the theoretical ones according to the P_{clust} (refer to Equation 5) is shown. As it is expected, the theoretical and simulation results are closely related.

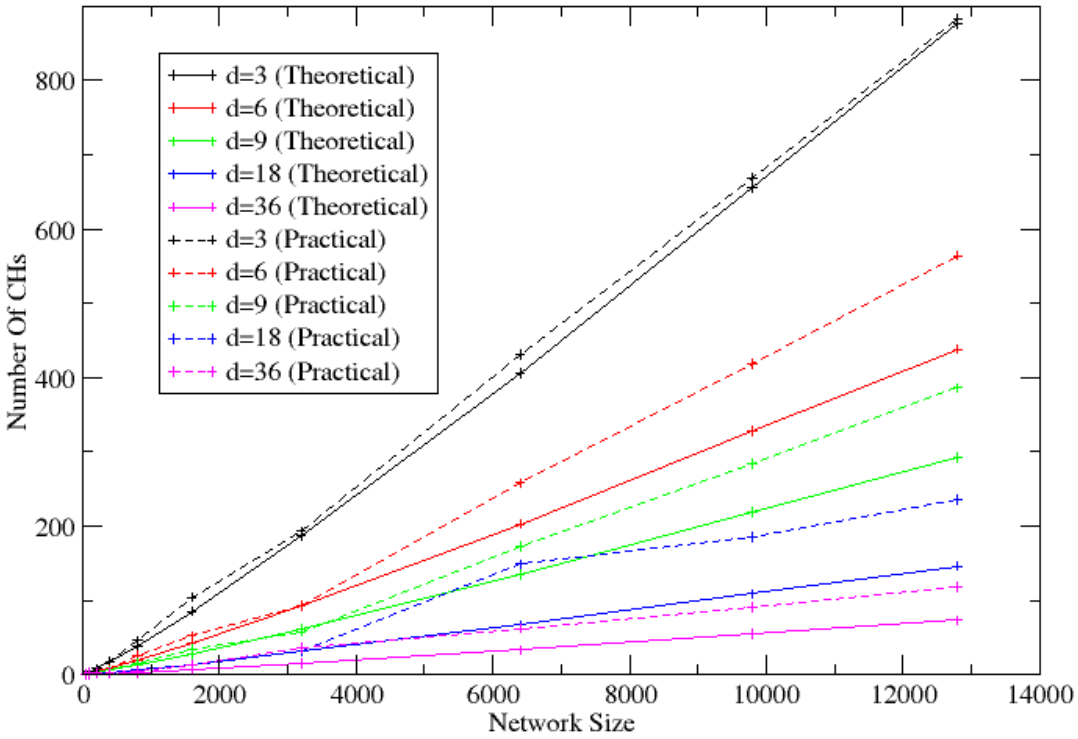


Figure 6.14 - Theoretical and Practical number of elected CHs

An important characteristic for the optimization of the clustering process is the adaptation of the clusters' size according to the changes in the network topology. In cases of more dense networks, it is desirable the creation of larger (in size) clusters since nodes are close (in number of hops) to each other. The existence of less CHs with small distances from their members improves the energy efficiency of the WSN (refer also to Section 6.1.3.3).

In Figures 6.15(a) – (c), the average size of the clusters that are created is presented for fixed and variable probability (refer to Equation 5). In the latter case, the trend is the creation of larger in size clusters in dense networks in opposition to the first case where the cluster size remains stable. Self-optimization of the clustering process is therefore achieved. However, more optimal equation for the P_{clust} probability may be selected in case that, smaller clusters are desirable in dense networks. Furthermore, great variation is present in small-scale networks [Figure 6.15(c)] due to the impact that has the probability in the cluster size, as the number of the elected CHs significantly affects the average cluster size. This variation is decreased as the period that NEURON is applied in the WSN increases since probabilistic techniques follow an optimal behavior.

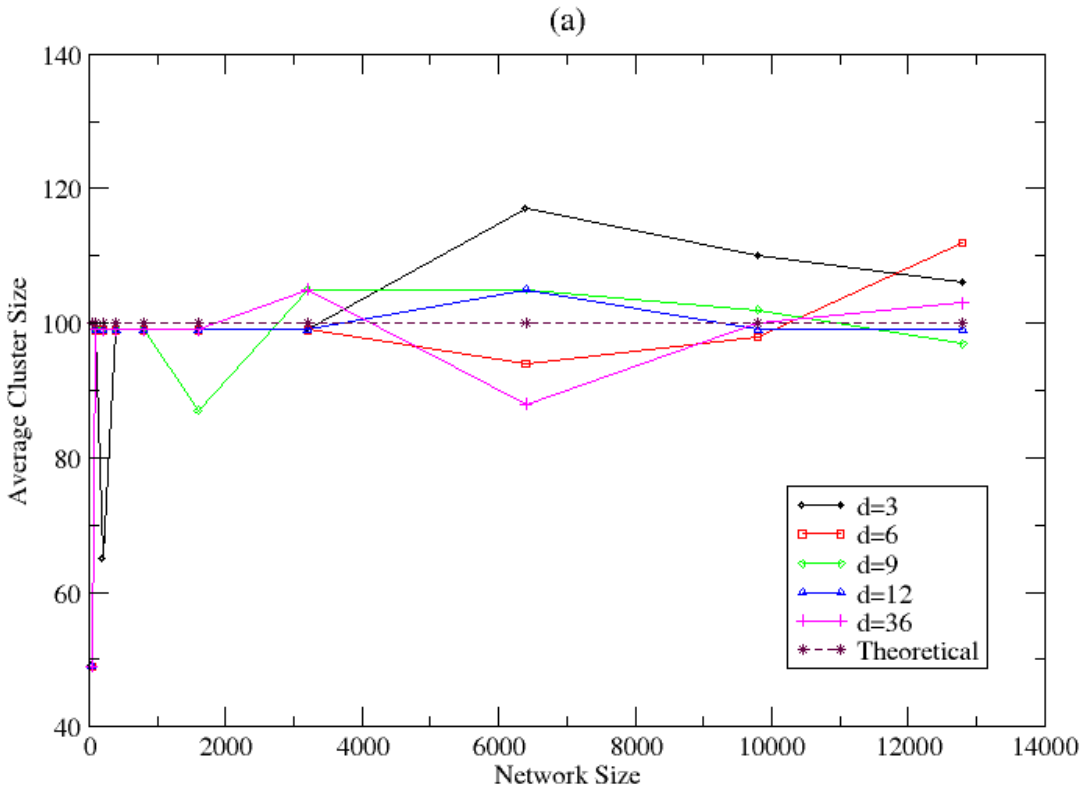


Figure 6.15(a) - Cluster Size for P_{clust} = 1%

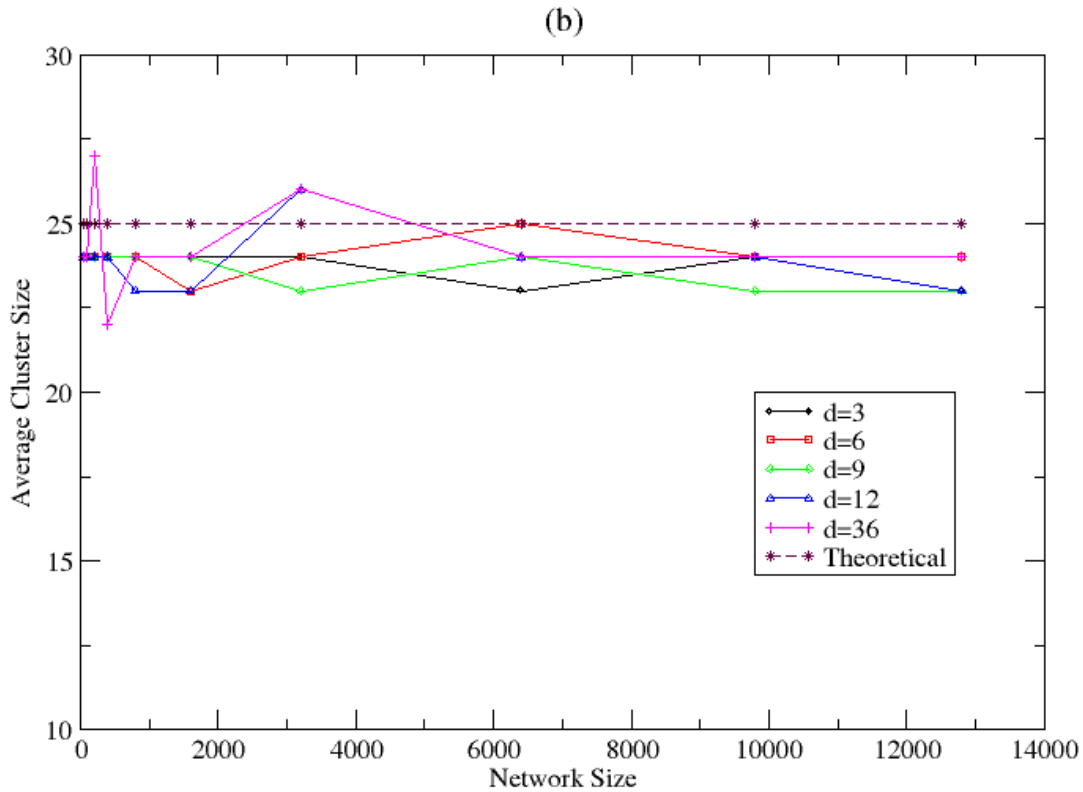


Figure 6.15(b) - Cluster Size for P_{clust} = 4%

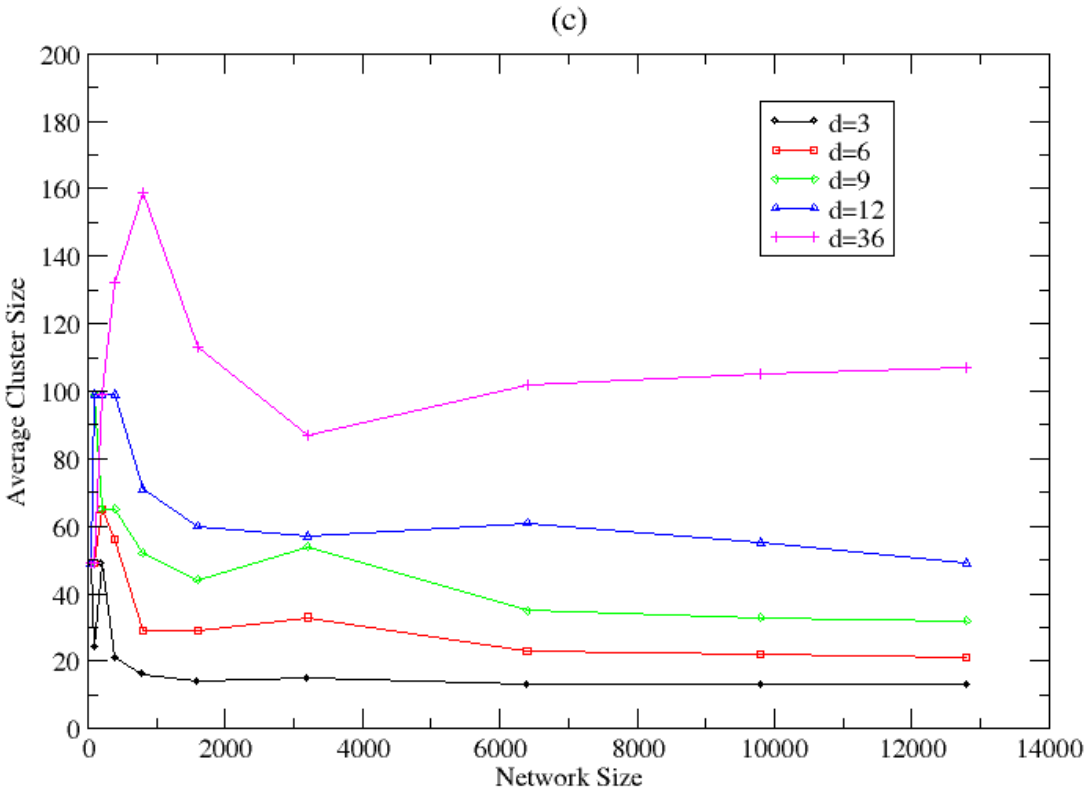


Figure 6.15(c) - Cluster Size for variable P_{clust}

The average number of route entries that are stored on each node's routing cache after the cluster formulation process is shown in Figure 6.16. This number is critical since mobile nodes may present memory constraints. P_{clust} is variable according to Equation 5. It is shown that the number of route entries increases slightly as the size and the density of the network increases. However, this number remains bounded, even for large networks. In sparse networks, an average routing cache has 50 entries while in dense networks an average routing cache with 150 entries is needed. Since the maximum size of a routing entry is 45 bytes, 15kB of routing cache size is adequate for all the mobile nodes.

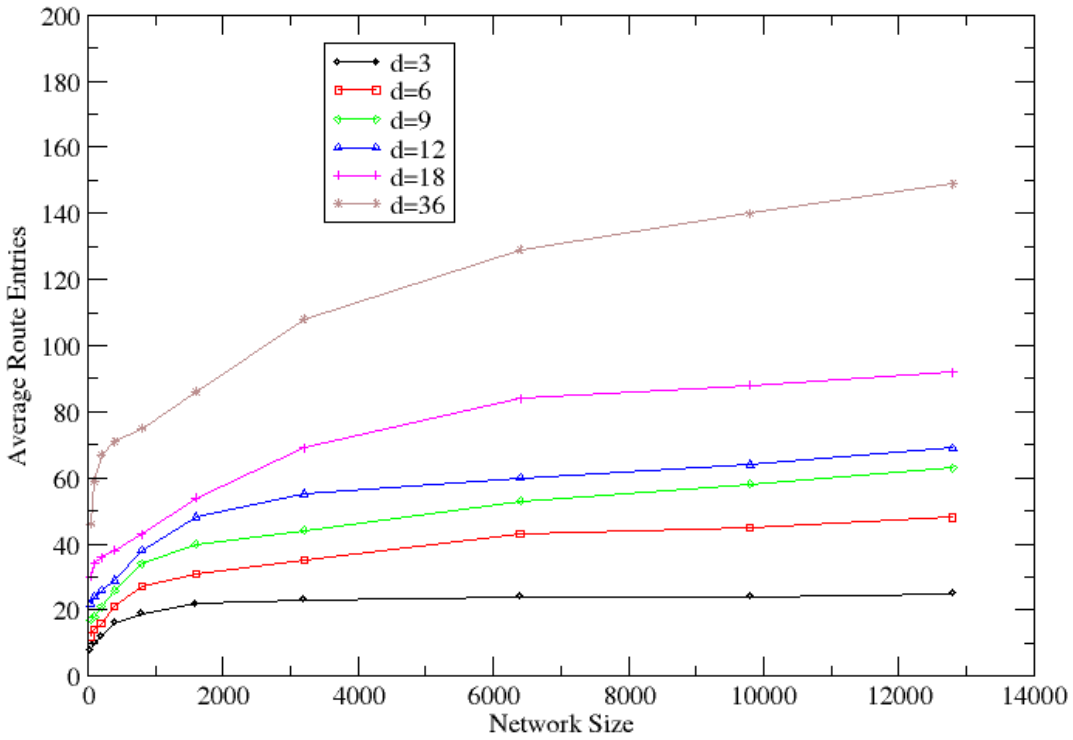


Figure 6.16 - Average entries in each node's routing cache

In addition to the number of route entries, a qualitative metric is the percentage of the total route entries that exists in the CH's routing caches since these entries are used from the routing functionality in NEURON. This metric is depicted in Figures 6.17(a) - (c) for fixed and variable probability. As the number of CHs increases in the network, the percentage of the total routing entries that exist in their routing caches also increases. This is shown in Figures 6.17(a, b), where the selection of larger value for the stable probability results to higher percentages of routing cache entries in the CHs.

When this percentage is smaller, intra-cluster communication is facilitated since the nodes that are not CHs have available routes for other nodes within the cluster and do not need to communicate with their CH for establishing a route towards them. This percentage is smaller for dense networks due to the greater overlapping among the cluster zones and the existence of multiple paths toward a CH. Nodes in the overlapping regions store routing entries towards more than one CH. The percentage also increases as the size of the network increases while when a variable probability is used [Figure-6.17(c)] the percentage is small for dense networks and large for sparse networks.

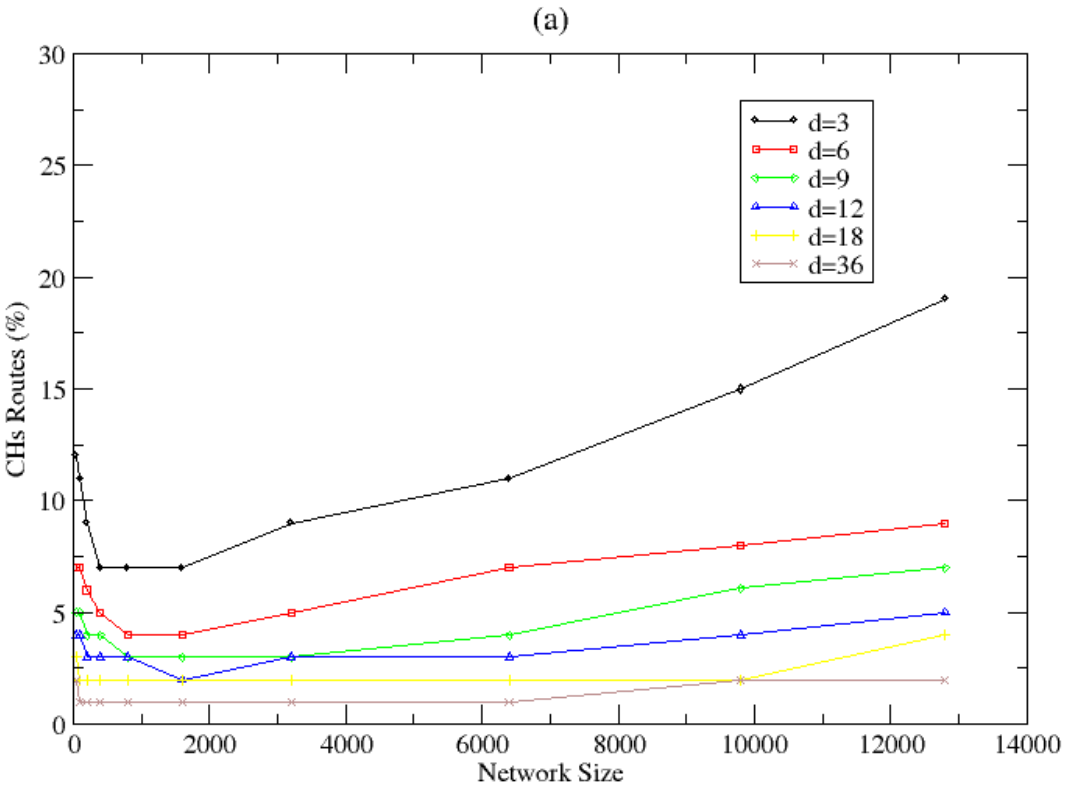


Figure 6.17(a) - Percentage of total routes in the CHs routing caches for P_{clust} = 1%

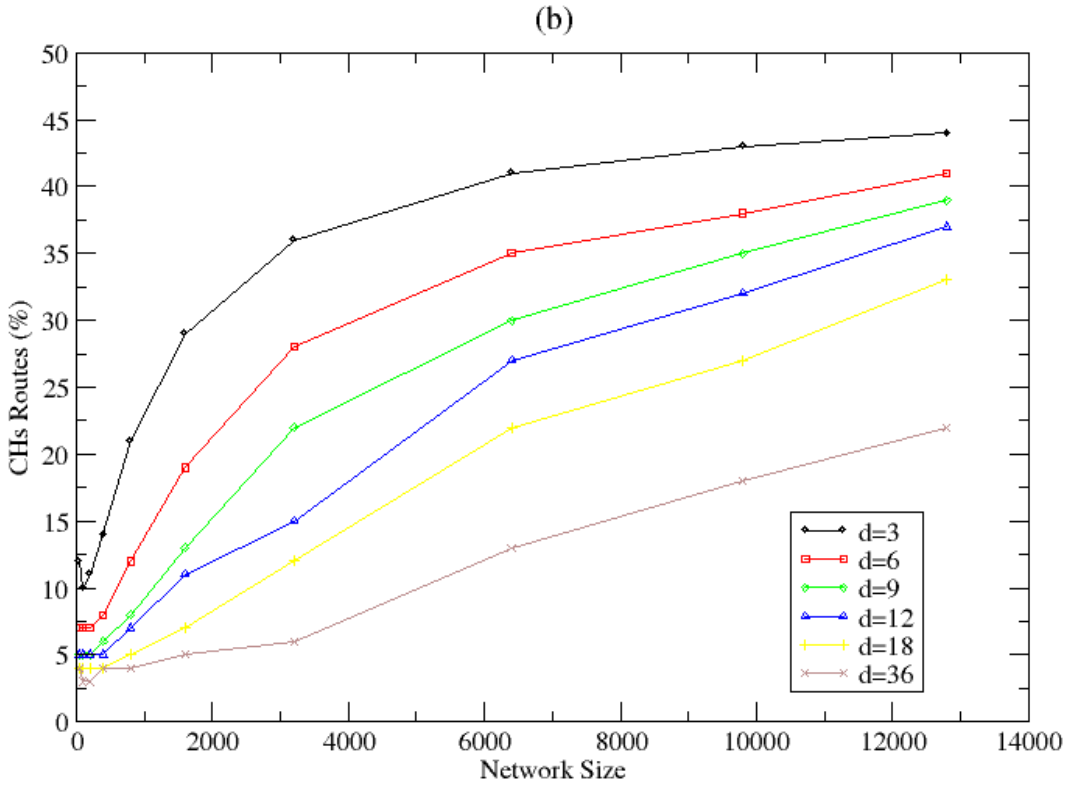


Figure 6.17(b) - Percentage of total routes in the CHs routing caches for P_{clust} = 4%

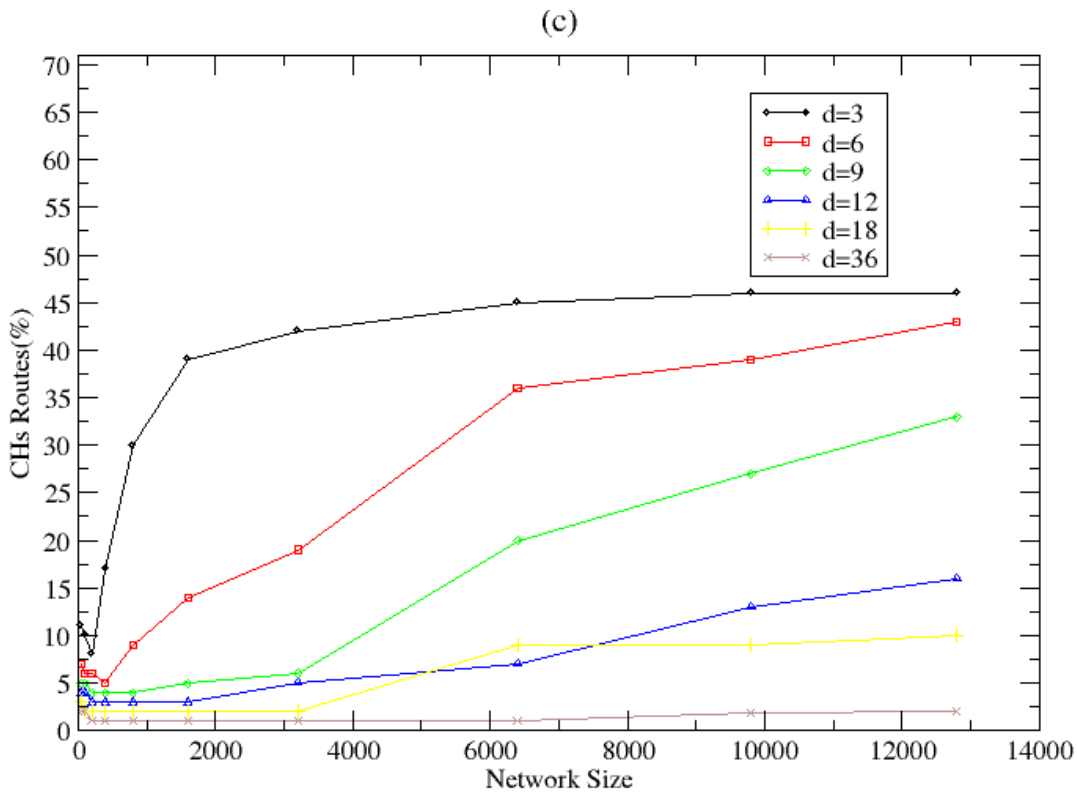


Figure 6.17(c) - Percentage of total routes in the CHs routing caches for variable P_{clust}

In Figure 6.18(a), the total number of routing messages that are exchanged until the clustering formulation is completed is presented, while in Figure 6.18(b), the same number per node in the WSN is shown. More routing messages are exchanged in dense networks, due to the nature of the controlled flooding mechanism that has been adopted from the *MSolicitateCH* message. Although cluster formulation messages are confined and cycle-prevented as discussed earlier, the existence of multiple connections for each node creates an analogous routing overhead that is avoided in sparse networks. Furthermore, in Figure 6.18(b) it is shown that the clustering formulation mechanism is scalable since the number of messages per node for different densities remains either stable or slightly increases as the network size increases.

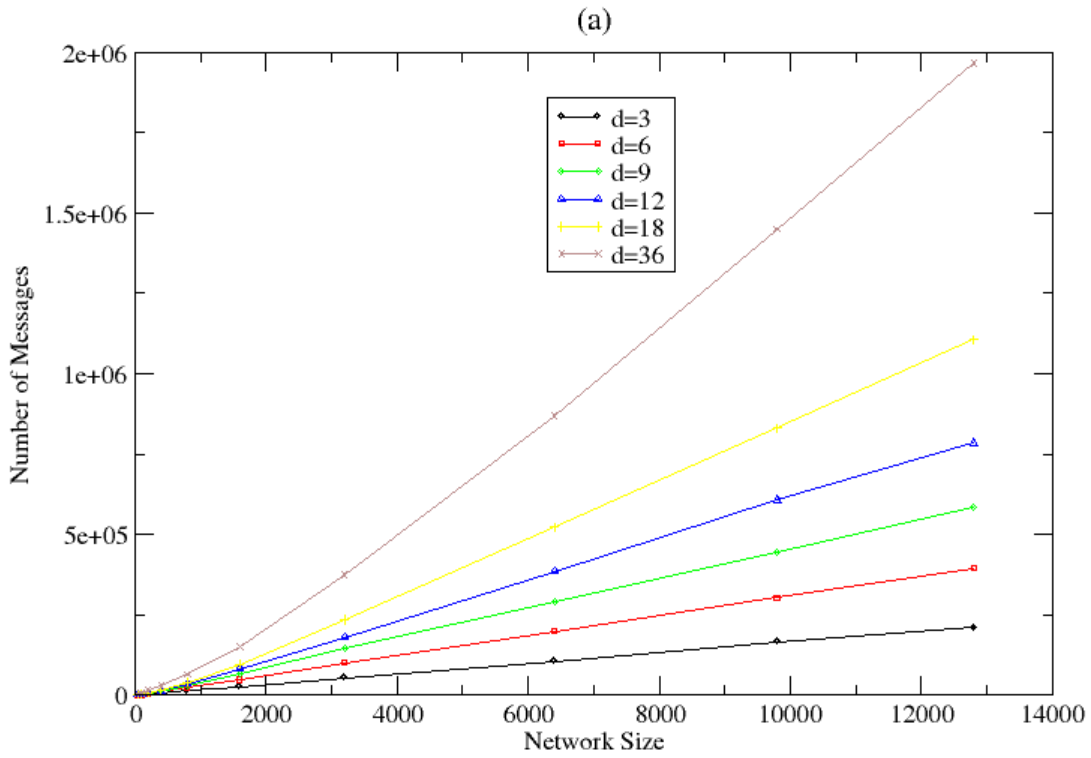


Figure 6.18(a) - Total number of messages exchanged for cluster formulation

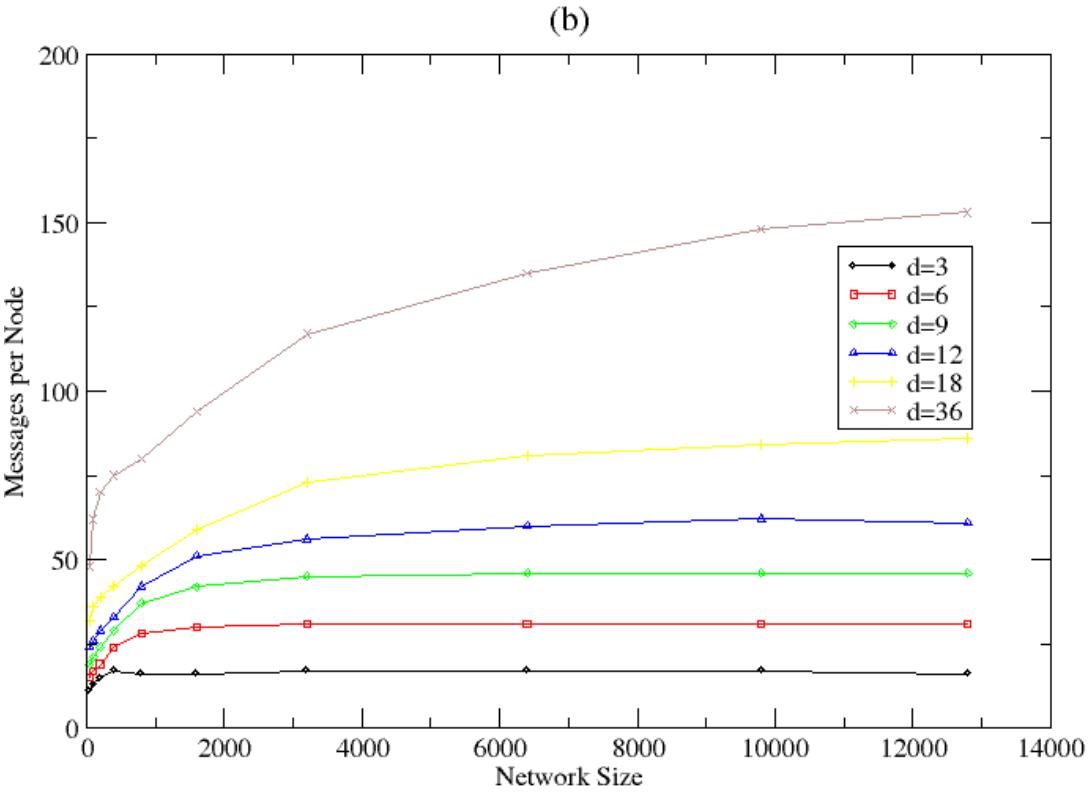


Figure 6.18(b) - Total number of messages exchanged for cluster formulation/ node

Upon completion of the clustering process, routing functionality exploits the available information in the routing caches of the mobile nodes. In Figure 6.19(a) and Figure 6.19(b) the total number of *RouteRequest* and *RouteResponse* messages that are exchanged in order a node to identify a valid route, are depicted. The number of the generated *RouteRequest* and *RouteResponse* messages is radically reduced as the network density increases since more routing information is already available in the nodes. Thus, NE URON's scalability is addressed as the routing overhead (in number of messages) is considered low.

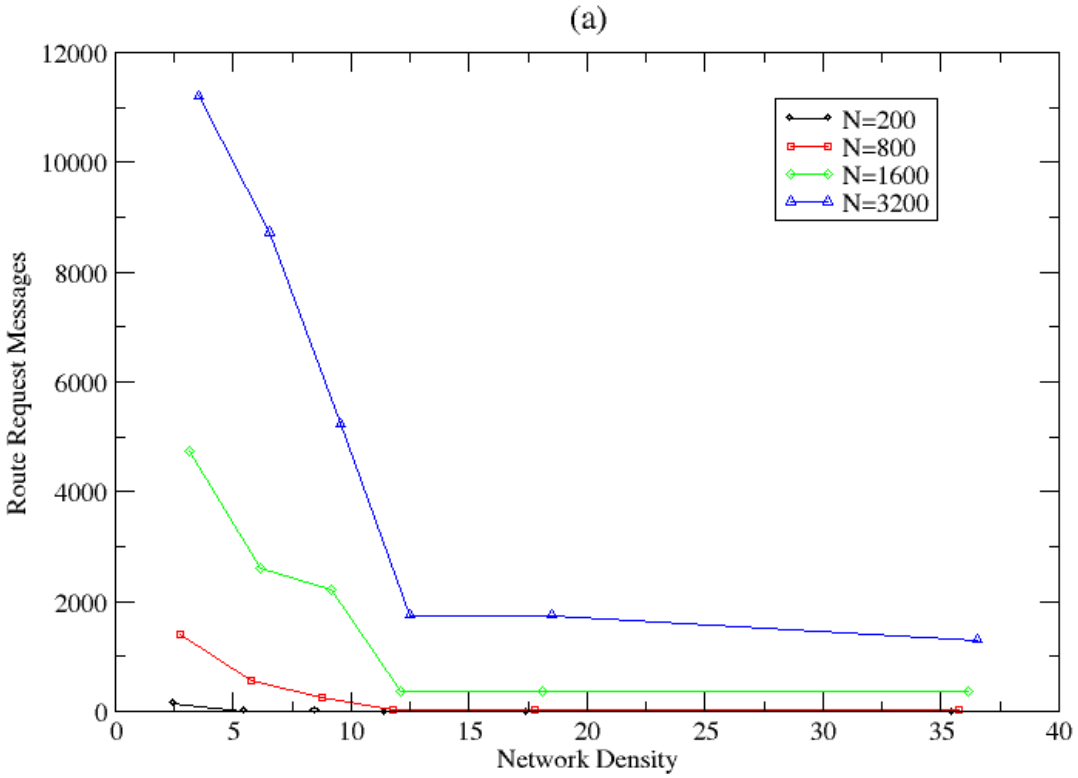


Figure 6.19(a) - Route Request Messages Cost

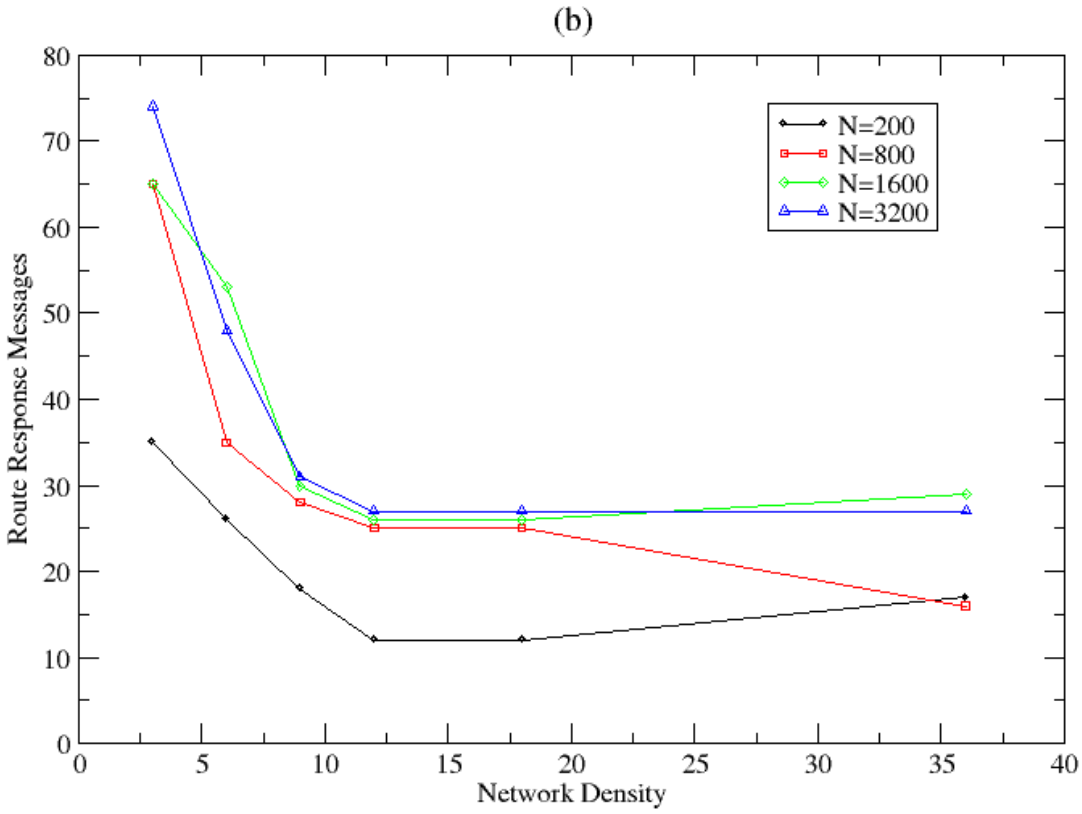


Figure 6.19(b) - Route Response Messages Cost

6.1.3.3. Energy Efficiency & Topology Formulation Evaluation in NEURON

Simulations are performed in order to assess the energy efficiency of NEURON mechanisms and their impact to the network lifetime. Network lifetime refers to the time period where all the nodes of the network (or a very high percentage of them) are operational. The network size is set to 1,000 while the initial energy of each node is set to 100,000 units. Simulations are terminated when the existing Mesh is split into two or more isolated groups as nodes leave the Mesh network when their battery is exhausted.

In Figure 6.20(a) the number of nodes that are alive while the number of cycle increases is shown for fixed and variable probability and different densities. The threshold where a CH switches to normal mode – when the KPI of the CH goes below it – is set to 50%. It is noticed that the network tends to extend its network lifetime as the number of alive nodes reduces steeply after a certain number of cycles. This means that the available power of the mobile nodes is almost the same and thus they run out of power in a few cycles. Furthermore, it is shown that the network lifetime is longer in the case of applying the variable probability and in case of more sparse networks. This is reasonable since fewer messages have to be exchanged in sparse networks for cluster formulation and maintenance. In Figure 6.20(b), the residual energy in the network is shown. The threshold

where a CH switches to normal mode is set to 50% and 75% while the density is set to 15 and 30, respectively. The residual energy is higher in case of sparse networks and in the case where the threshold is set to 50%. A high threshold reduces the rotation in the CHs in the Mesh and causes high energy consumption in each CH, causing them to run out of energy earlier than the other nodes. In this case, therefore, energy consumption is not homogeneously distributed among the mobile nodes. In Figure 6.20(c), the consumed energy is depicted in case of the fixed and the variable probability, while the density is set to 15 and the threshold where a CH transits to normal mode is set to 50%. It is clear that the application of the autonomic mechanism in the election of clusters is more energy efficient compared to the application of a stable probability.

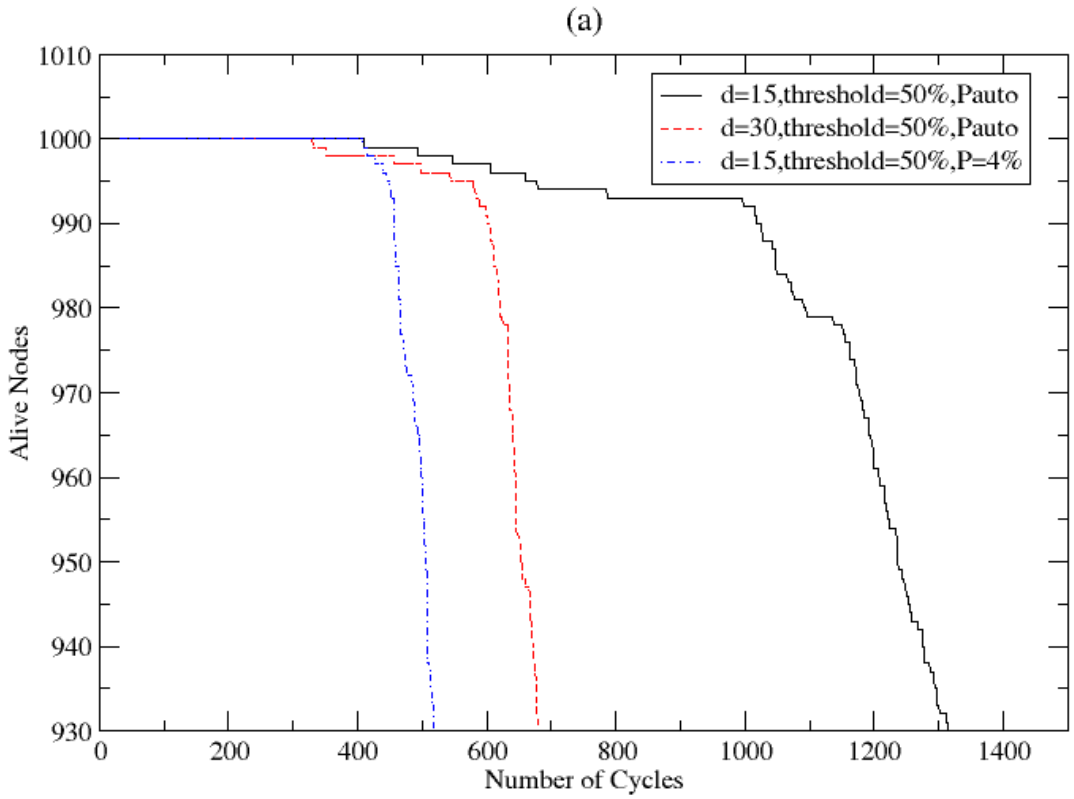


Figure 6.20(a) - Alive Nodes in the Mesh

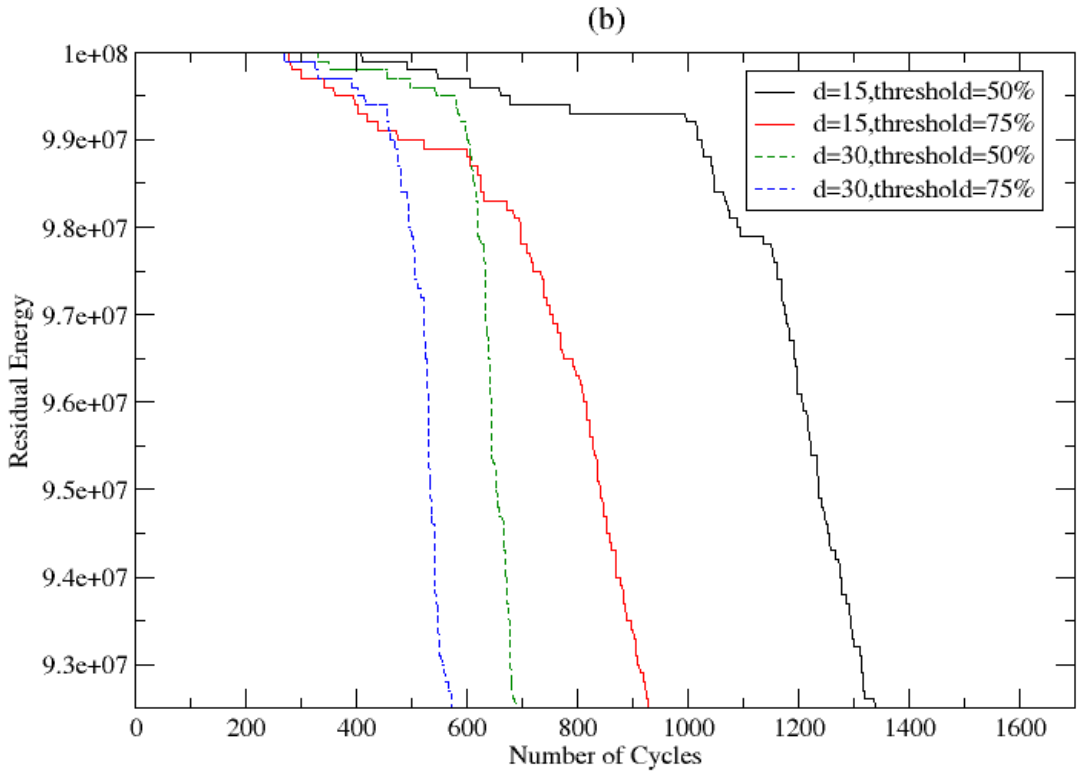


Figure 6.20(b) - Residual Energy in the Mesh

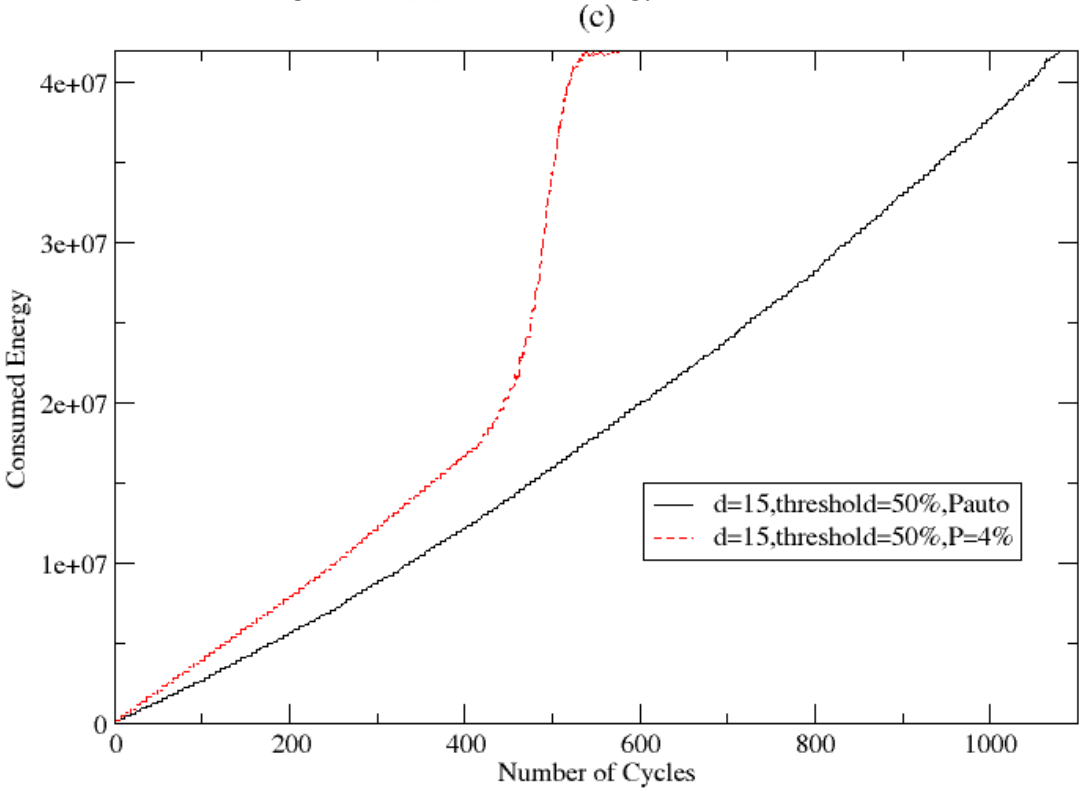


Figure 6.20(c) - Consumed Energy in the Mesh

As described in Section 4, the NEURON protocol facilitates the creation of an overlay topology and consequently the deployment and provision of autonomic services over it. In order to show NEURON's suitability for this purpose, we compare the messages that are generated for the overlay topology formulation using DSR, i.e. another reactive routing protocol. In Figure 6.21, it is shown that the logarithmic behaviour of routing cost in DSR imposes extreme overhead to the network in comparison to NEURON. Furthermore, this overhead is much greater in dense networks.

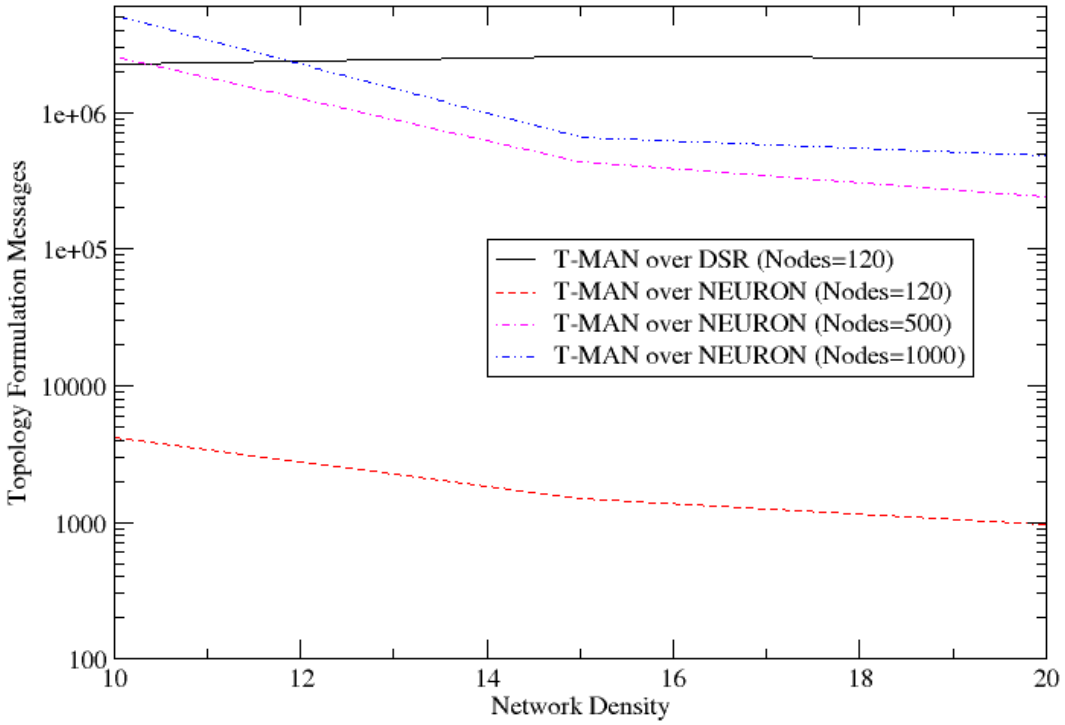


Figure 6.21 - Routing Messages Cost for Topology Formulation (logarithmic scale)

6.2 Emulation Results

Beyond PeerSim models that have been implemented in order to evaluate UbiChord's performance, an extensive set of emulations have been conducted using the real software . As mentioned at Chapter 5, the prototype implementation is developed in Java and supports the bootstrapping of a (multi-hop) ad-hoc network and the communication among the participant nodes. In the bootstrapped network, the autonomic estimation algorithm is applied for the estimation of various parameters. It is important to note that multiple instances of the prototype implementation may run to the same node for experimental purposes.

A topology editor is also implemented that permits the creation of an experimental topology and the emulation of the algorithm in the specified topology. The following

topologies that are representative of global characteristics of real networks are supported: Hypercube, Star, Rotated Tree, RingLattice and Barabasi-Albert. These topologies are helpful for understanding the behaviour of the gossiping mechanism in various graphs models. An indicative screenshot with the form of each topology is shown in Figure 6.22.

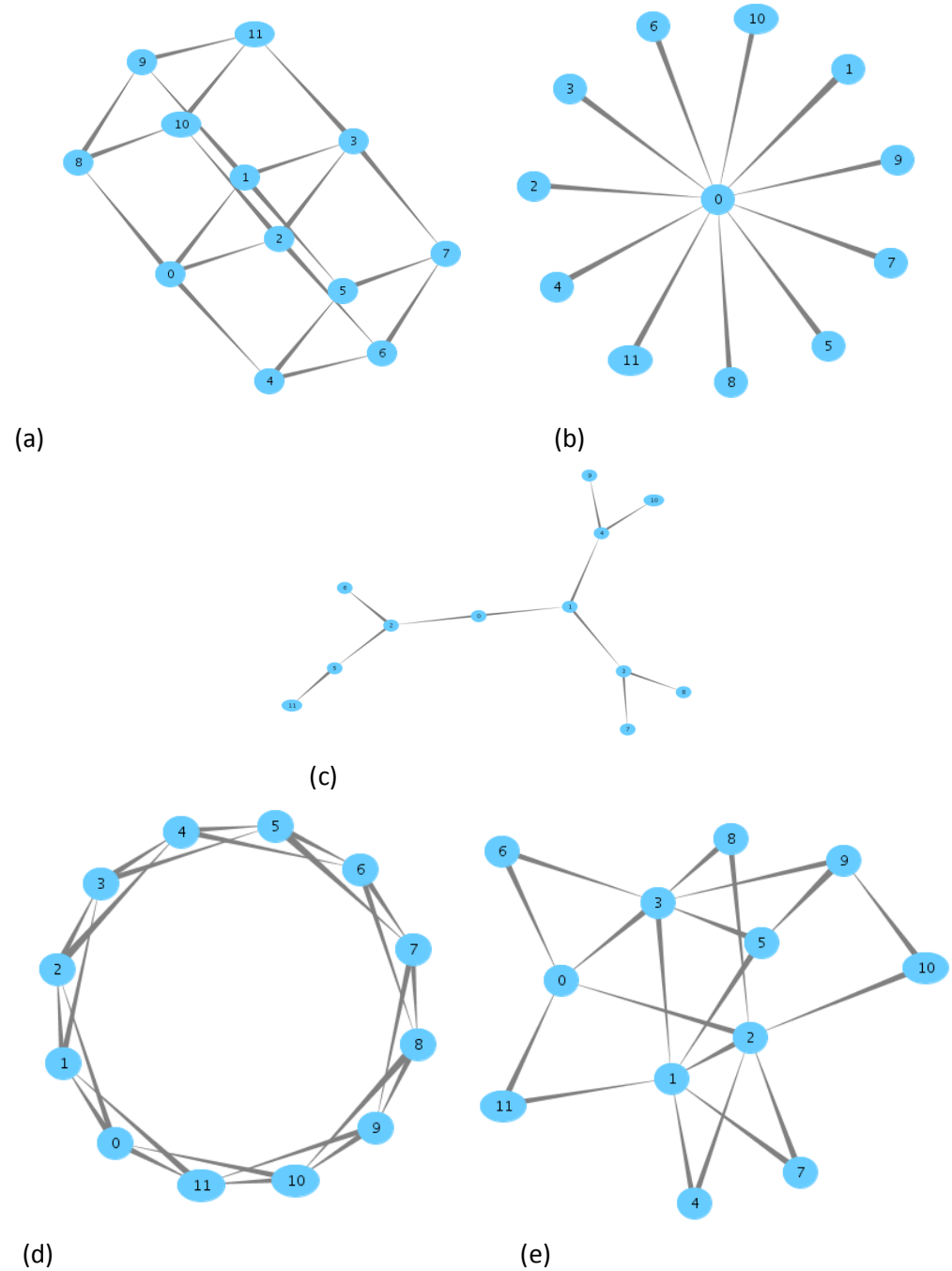


Figure 6.22 - Various Topologies: (a) Hypercube, (b) Star, (c) Rotated Tree, (d) Ring, (e) Barabasi

According to each experiment, multiple nodes are simultaneously activated without any preconfigured state information. In Figure 6.23, the total number of messages that are exchanged - until convergence of the estimated value is succeeded in the emulated network with threshold set to 10% - is depicted for various network sizes. The threshold refers to the difference in the estimated value between two consecutive cycles.

In topologies where the density is almost stable for all the nodes (i.e. the Tree and Circular topologies) the total messages required are less than those for topologies that present non uniform distribution of the network density (i.e. Barabasi-Albert, HyperCube and Star). Especially in the Star topology, the total messages required were more than double from the other topologies (Circular and Tree topologies) since the centralized structure is not suitable for neighbor to neighbor gossiping (inconsistency is present due to many mutual exchanges of messages with the central node). The proposed algorithm is claimed to be scalable since there is almost linear behavior in all the types of topologies. However, further study is necessary in larger networks in order to validate that the number of messages is proportional to the size of the network.

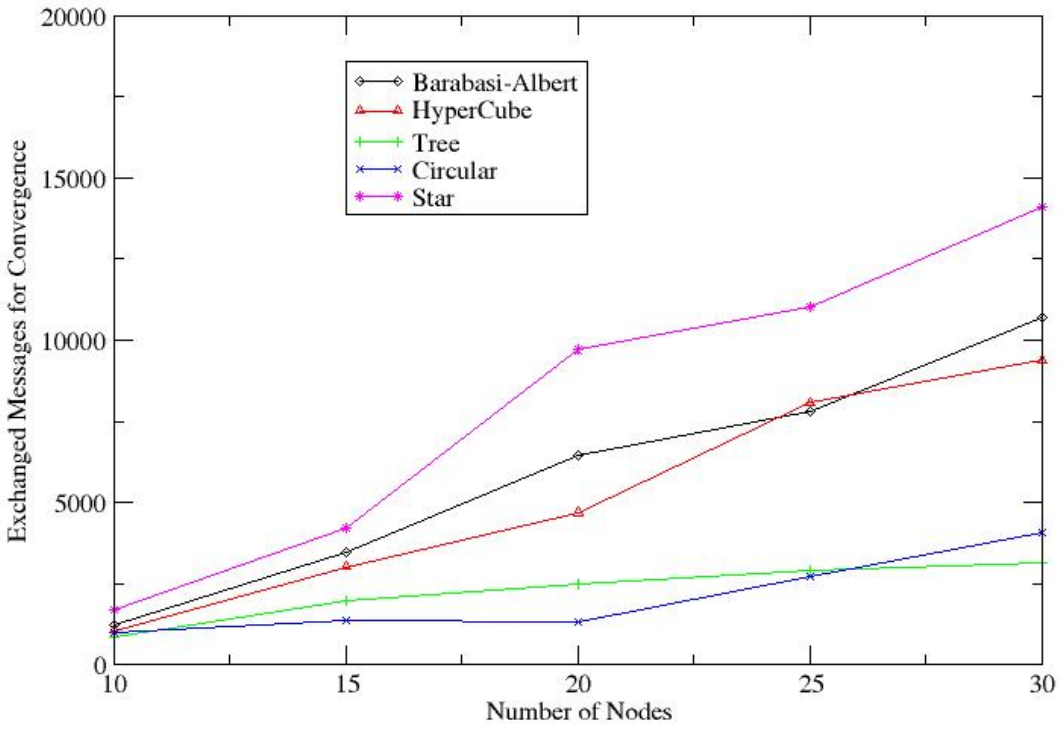


Figure 6.23 - Number of messages for convergence

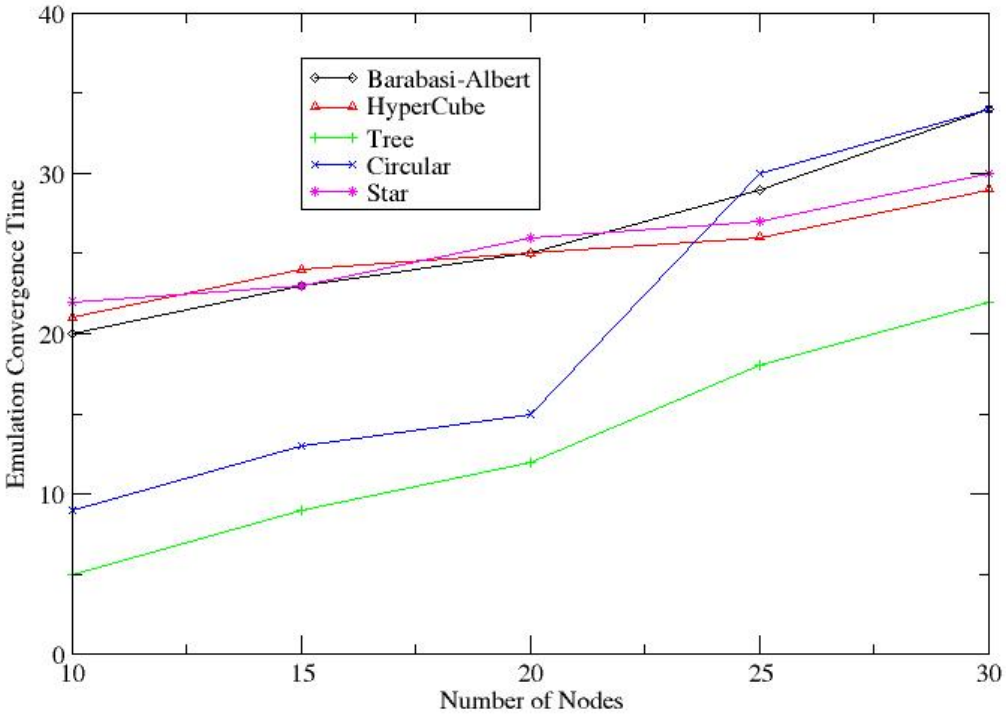


Figure 6.24 - Emulation time for convergence

In Figure 6.24, the emulation time - until convergence of the estimated value is succeeded in the emulated network - is depicted for various network sizes. In all types of topologies, the emulation time is increasing linearly with the increase in the network size. The smallest time is needed in case of Tree topologies. Moreover, in Figure 6.25, the number of messages required for convergence is depicted in case of threshold 1% and 10% for difference between two consecutive values. The emulation is performed in case of Tree topology. It is clear that an increase in the accuracy of the estimated value necessitates more messages for convergence.

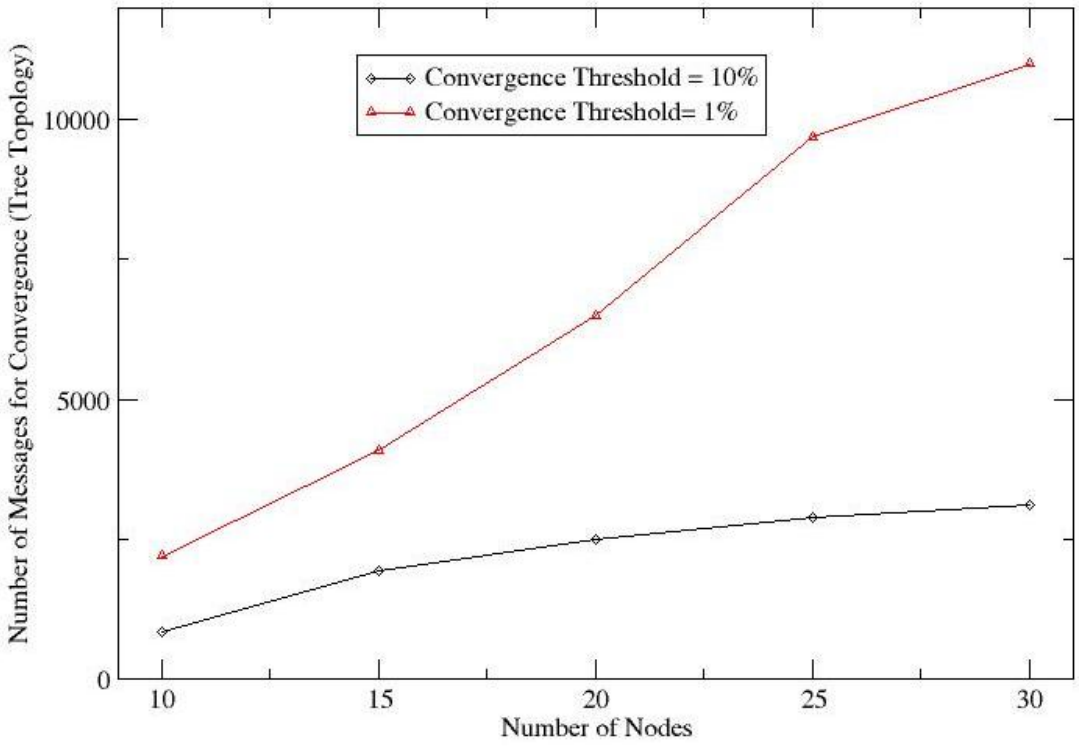


Figure 6.25 - Number of Messages for convergence

In the examined scenarios, multiple nodes are simultaneously activated in the emulated ad-hoc network. Emulations are conducted for networks with size from 5 to 20 nodes. Each experiment was executed 5 times and average values were considered in our analysis. An indicative visualization service is created on top of UbiChord. Each node periodically publishes its neighbors' view to the overlay network that is created and maintained over a mesh physical topology.

A final set of experiments on a complete autonomic visualization service has been conducted. In the first experiment, all network nodes were simultaneously bootstrapped. The total messages required for the provision of the visualization service are measured. In Figure 6.26, the total messages exchanged for routing purposes, for creation and maintenance of the overlay network and for the visualization service provision are depicted. It is shown that the number of messages increases following an almost linear pattern as the size of the network increases.

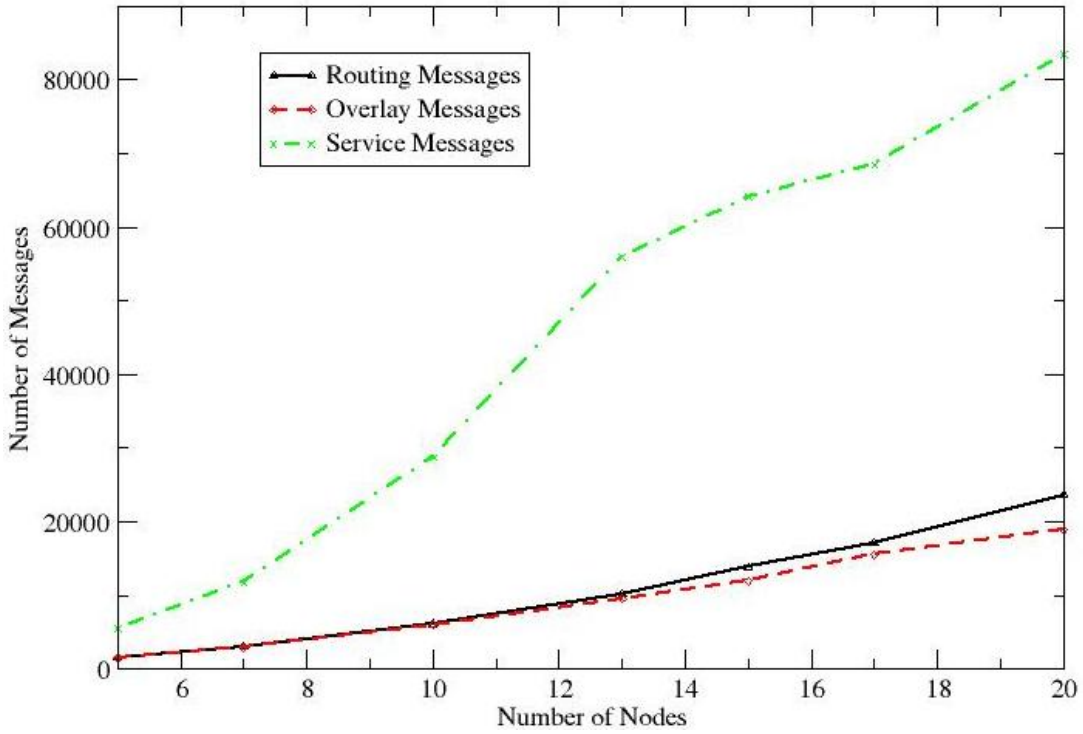


Figure 6.26 - Total messages required for services provision

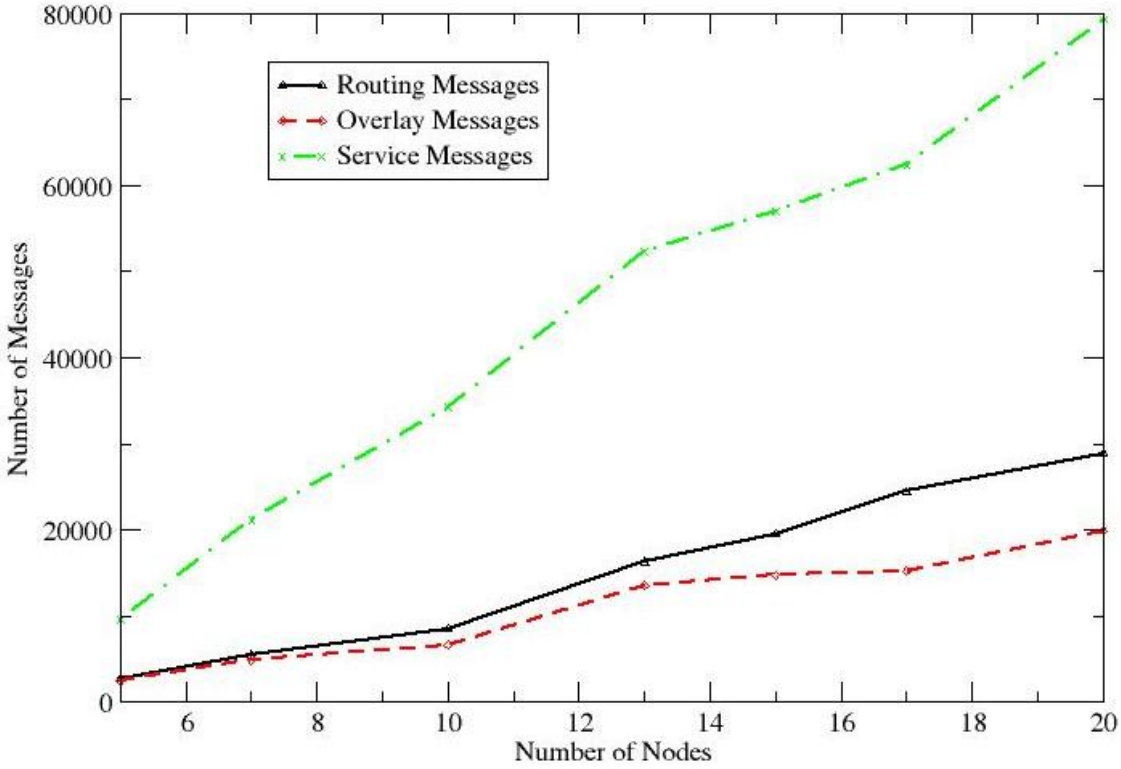


Figure 6.27 - Total messages required for services provision

In the second experiment, the size of the mesh network was progressively increased ranging from 1 to 20 nodes. In each entrance of a new node, re-stabilization procedures took place for the overlay network topology as well as the re-assignment of the key-value pairs. In Figure 6.27, the total messages exchanged for routing purposes, for creation and maintenance of the overlay network and for the visualization service provision are depicted. In comparison with the previous experiment, it is shown that the number of messages for routing and overlay topology maintenance is slightly increased due to the messages exchanged for re-stabilization purposes.

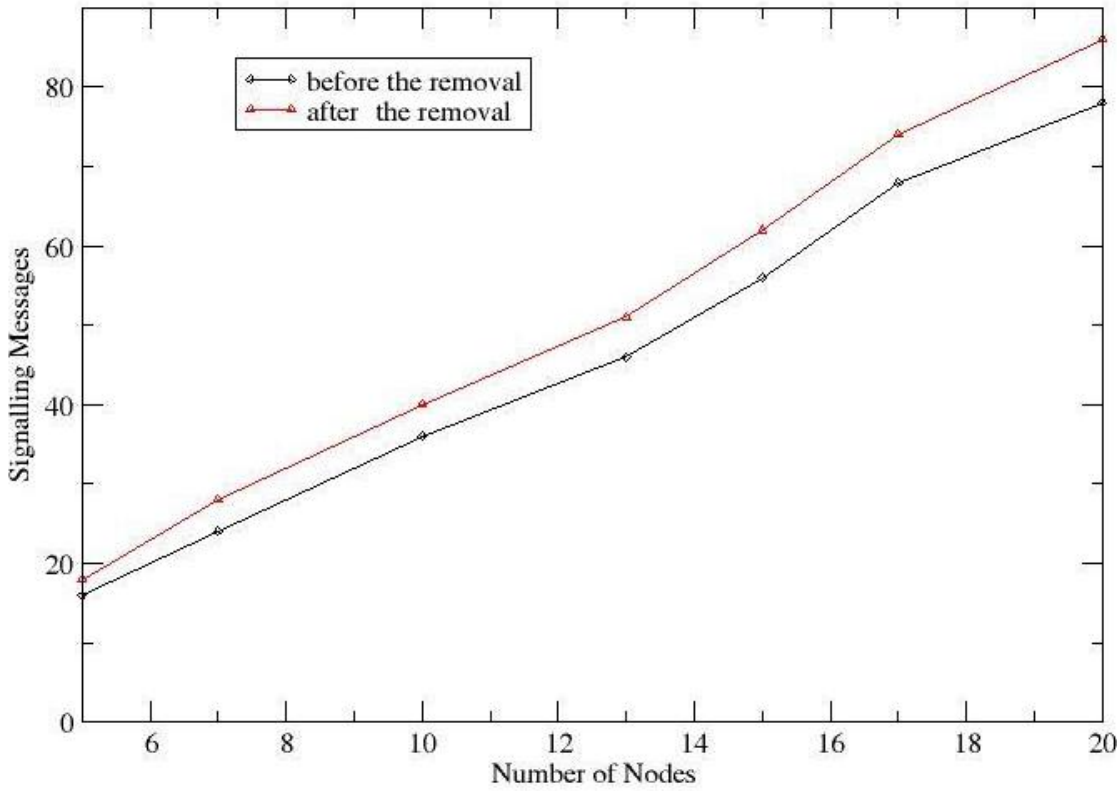


Figure 6.28 - Total number of signalling messages

In the third experiment, the size of the mesh network was progressively increased ranging from 1 to 20 nodes. In each case, a node was randomly removed from the network, while the provision of service was not interrupted. Re-stabilization messages were necessary in case of each topology change in order to make the service provision stable and reliable. In Figure 6.28, the total number of signaling messages exchanged, in case of stable network (before the removal) and in case of removal of one node (after the removal) is depicted. It is shown that dynamicity (removal of nodes) in the network slightly affects the total number of messages required for provision of stable services over the overlay network, and thus the extra overhead imposed is small.

7. Conclusions and further research

7.1 Conclusions

In the mobile services domain there is a new emerging trend for Mobile Peer-to-Peer applications. This trend is derived the increased capabilities of mobile devices as far as Mesh networking is concerned and the infrastructural support by Mobile Network Operators. Such applications are demanding since they must be aware of the context and the environment, in which they operate, self-configuration and self-adaptation according to the network conditions that they sense and require minimum feedback from the end-user avoiding any explicit human intervention.

These challenges can be addressed through the incorporation of p2p techniques. Such techniques base their operation on Distributed Hash Table structures maintained on top of overlay networks. However, the adoption of current p2p techniques necessitates the existence of a fixed network topology. Under this perspective, the major objectives of this Thesis were the analysis of the key problems that are raised during bootstrapping and operation of DHTs in Mesh Environments, the proposal and evaluation of a protocol that overcomes the identified problems and the provision of a reference implementation of the proposed protocol accompanied by a real Mobile Peer-to-Peer application.

As a result, the Thesis produced three concrete assets. First of all, a Distributed Hash Table Protocol capable of operating on top of Mesh Networks was introduced. In order to cope with the dynamicity that exists in mesh environments specific mechanisms had to be created. These mechanisms are organized in a layered scheme to undertake tasks such as unreliable node-to-node communication, absence of static routing scheme, ability to join and maintain an overlay etc. All these mechanisms where thoroughly analyzed since a representative simulation Peersim model has been created.

Furthermore, the second asset is related to a concrete reference implementation of UbiChord protocol. In order the protocol to be realized (or instantiated) specific technical choices have been followed. During the emulation of the implemented protocol, additional mechanisms have been introduced in order to tackle specific bottlenecks that were identified, especially at the routing layer. In order to resolve these issues, additional mechanisms that “bend” the non-linear performance issues have been incorporated. These mechanisms were grouped as a separate protocol addressed as “NEURON”. NEURON mechanisms have not been integrated in the implemented protocol however they have been exhaustively analyzed through Peersim simulations.

Furthermore, in the frames of this Thesis, a novel application-provision paradigm targeting mobile peer-to-peer services has been introduced. More specifically, the existence of a DHT capable for storing and retrieving information (i.e. key-value sets) comprises the cornerstone of a new programming model regarding service provisioning.

Up to now, collaborative applications in Mesh or PAN networks follow a centralized approach, in the sense that either all nodes that run an application are assumed to have connectivity with the central server or one node of the mesh network "acts" as a server. The utilization of UbiChord can alter this programming model by introducing native decentralization. Applications can share (i.e. store and retrieve) volatile data in UbiChord without caring who is responsible for the physical storage of data. This capability paves the way for the creation of a new set of applications that are in line with autonomic principles in the era of Internet of Things (a.k.a. Internet of Connected Objects).

The instantiation/rollout of Internet Connected Objects paradigms follows a diverse scheme by industrial stakeholders (e.g. IBM, HP etc.). Although the benefits for the applications that are based on IoT technology are a lot, there are problems that are raised by this diversity and are leveraged by the scaling limitation of existing Internet centralized infrastructure, which is not capable of "serving" the load of millions of sensors/actuators that have to collaborate. These specific problems can be tackled with the incorporation of autonomic techniques. This philosophy is in-line with the emerging trend of bio-inspired computing according to which algorithms and principles applied at nature can provide elegant solutions to existing problems in computer science.

As a result, theoretical and practical assets of this thesis can be used to create applications that can achieve Self-Configuration, Self-Healing, Self-Optimization and Self-Protection. These autonomic principles can be achieved using UbiChord as the medium for storing and retrieving information.

7.2 Limitations, Possible Improvements & Future Work

In the frames of this Thesis, theoretical and practical aspects of Distributed Hash structures that are able to operate in Mesh environments have been provided. The concrete outcome of this research was a proposed DHT protocol (i.e. UbiChord), a concrete implementation of this protocol (i.e. Java Library of UbiChord) and a set of mechanisms that resolve scaling issues of UbiChord (i.e.NEURON). It is meaningful, to refer to limitations that exist in each of the three different assets of the Thesis.

Regarding UbiChord protocol, it has been exhaustively simulated based on Data Source Routing protocol. It would be valuable to integrate a different reactive routing protocol (e.g. AODV) in order to examine changes regarding message-signaling overhead. Examining the simulation-results in Chapter 7, it could be argued that the routing layer introduces the majority of signaling, especially under dynamic conditions such as node failures, node entries, island formulations etc. The introduction of NEURON optimized the routing overhead since it introduced a dynamic clustering mechanism that is able to hierarchically propagate routing requests/replies. This policy minimizes the restricted flooding that is essential for any reactive routing protocol, while it introduces additional signaling for cluster formulation and cluster maintenance. This additional signaling has been examined also thoroughly. In general, the optimal solution would be the creation of a policy model which would control the mechanisms of each layer. For example, since every node is self-aware of the network size, a policy rule could regulate the routing protocol which is, during run-time, activated. For example in case of 10 to 50 nodes DSR performs extremely well even under the worst dynamic conditions. However, if the network size changes suddenly to 100 or 1000 then a hierarchical routing mechanism should be activated.

This policy model could be generalized, in the sense that usage of global information such as network size, density etc. could affect several mechanisms of UbiChord. Indicatively, in order to achieve neighbor identification, a periodic solicitation is made by each node (just as 802.11b sends 5 packets per second). Since a node can be aware of the dynamicity of its neighborhood, it could regulate the frequency of the announced packet by applying an exponential back off. This would result to energy saving which would be preferable in resource-constrained nodes such as nodes of a Wireless Sensor Network (WSN).

Moving from the routing layer and the various optimizations that can be applied there, many open research space exists at the overlay maintenance and DHT layer. Probably the most challenging issue that UbiChord has to undertake is the fast formulation of the overlay during startup and the minimum disruption of the overlay upon various topology changes. To achieve that UbiChord is relying to the unique IPv6 address of the node's physical network adapter. This technique implies that every node enters the overlay having

an already assigned identifier. However, in the DHT research domain another approach also exists; the assignment approach. According to this approach, each node that enters a physical network, where an overlay is already bootstrapped, is assigned with a virtual identifier. This approach minimizes the bootstrapping cost, however it introduces several other issues such as uniqueness of identifiers, maintenance of generated identifiers etc.

Another issue which is valuable to investigate is the correlation between the DHT redundancy-ratio and the signaling overhead that is generated under dynamic conditions. As described at Chapter 4, every key-value entry is physically stored to a delegated node according to the protocol. However, if the physical existence of the entry is confined in only one node, this node would be a single-point of failure. Consequently, UbiChord follows the traditional practice of replication for a specific entry. Nevertheless, the overhead of applying traditional practice of replication on DHTs that are bootstrapped on fixed networks is minimal. In the case of UbiChord this is not true, because the disturbance of the physical topology (which is frequent) causes significant signaling cost for key-value transferring. Part of the future work is to identify the golden ratio between the replication ratio (i.e. how many times every entry has to be replicated), the size of the network and the dynamicity of the network (i.e. frequency of removals/entries/link-failures etc.).

Regarding the limitations and possible improvements of the Java library that encapsulates the UbiChord functionality, there are a set of low-level improvements that could be accomplished. First of all, DSR mechanism has been implemented from scratch, but the maximum payload that can be transferred is limited at TCP MTU. In other words, routing layer can route messages that are less than the TCP MTU. This issue can be resolved with the creation of a Packetiser/Depacketiser that would disassemble and reassemble large amount of transferable information. Furthermore, the emulation layer that has been created is monolithic with means that a specific amount of memory is reserved per node-instance. A possible improvement here is to enable parallelization in order to emulate large number of nodes. Moreover, an improvement that would leverage the utilization of the library is its porting to the Android programming model.

Finally, regarding NEURON mechanisms, they have to be also integrated at the core UbiChord library, while in parallel the clustering and hierarchical routing mechanisms have to be tested under some context dependent mobility model. Such indicative models are applied on Vehicular Networks or Virtual Sensor Networks.

8. References

[A]

[A. Barabasi et al., 1999] A. Barabasi and R. Albert, "Emergence of scaling in random networks," Science, vol. 286, no. 509, 1999

[A. Barabasi et al., 2000] A. Barabasi, R. Albert, H. Jeong, and G. Bianconi, "Power-law distribution of the world wide web," Science, vol. 287, 2000

[A. Broder et al., 2000] A. Broder, R. Kumar, F. Maghoul, P. Raghavan, S. Rajagopalan, R. Stat, A. Tomkins, and J. Wiener, "Graph structure in the web," in Processings of the 9th Int. WWW Conference, Amsterdam, May 15-19 2000

[A. Datta, 2003] A. Datta, "Mobigrid: P2P overlay and MANET rendezvous — a data management perspective," in CAiSE 2003 Doctoral Symposium", 2003

[A. Galis et al., 2009] A. Galis, et al, "Management and Service-aware Networking Architectures (MANA) for Future Internet", Communications and Networking in China (ChinaCOM), pp.1-13, 26-28 Aug. 2009

[A. Goel et al., 2004] A. Goel and R. Govindan, "Using the small-world model to improve freenet performance," Computer Networks Journal, vol. 46, no. 4, pp. 555–574, November 2004

[A. Harvey et al., 2003] A. Harvey, M. B. Jones, S. S. amd M. Theimer, and A. Wolman, "Skipnet: A scalable overlay network with practical locality properties", in Proceedings of the 4th USENIX Symposium on Internet Technologies and Systems (USITS), Seattle, WA, USA, March 2003

[A. I. Wan et al., 2007] A. I. Wang, T. Bjørnsg˚ard, and K. Saxlund, "Peer2Me - rapid application framework for mobile peer-to-peer applications", in The 2007 International Symposium on Collaborative Technologies and Systems (CTS 2007), May 21-25 2007

[A. Liakopoulos et al., 2010] Athanassios Liakopoulos, Anastasios Zafeiropoulos, Constantinos Marinos, Mary Grammatikou, Nikolay Tcholtchev and Panagiotis Gouvas, Applying distributed monitoring techniques in autonomic networks, IEEE GLOBECOM 2010, Florida, USA

[A. Muthitacharoen et al., 2001] A. Muthitacharoen, B. Chen, and D. Mazi`eres, "A low-bandwidth network file system", in Proceedings of the eighteenth ACM symposium on Operating systems principles, 2001, pp. 174 –187

[A. Rowstron et al., 2001] A. Rowstron and P. Druschel, "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems", in Proceedings of the Middleware, 2001

[A. Rowstron et al., 2001] A. Rowstron and P. Druschel, "Storage management and caching in past, a large-scale, persistent peer-to-peer storage utility", in Proceedings of the eighteenth ACM symposium on Operating systems principles, October 2001, pp. 188 –201

[A. Rowstron et al., 2001] A. Rowstron, A. M. Kermarrec, M. Castro, and P. Druschel, "SCRIBE:The design of a large-scale event notification infrastructure", in Proceedings of the Third International Workshop on Networked Group Communications (NGC2001), Londo n, UK, November 2001, pp. 30 –43

[A. Rowstron el al., 2001] A. Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems," Lecture Notes in Computer Science, vol. 2218, 2001

[A. Rowstronet al., 2001] A. Rowstron and P. Druschel, "Scalable, distributed object location and routing for large-scale peer-to-peer systems", In Proceedings of the 18th IFIP/ACM International Conference on Distributed Systems Platforms, Heidelberg, Germany, November 2001

[A. Singh et al.,2004] A. Singh, M. Castro, P. Druschel, and A. Rowstron, "Defending against eclipse attacks on overlay networks", in Proceedings of the SIGOPS European Workshop, Leuven, Belgium, September 2004

[A. Zafeiropoulos et al., 2010] Anastasios Zafeiropoulos, Panagiotis Gouvas, Athanassios Liakopoulos, Gregoris Mentzas and Nikolas Mitrou, NEURON: Enabling Autonomicity in Wireless Sensor Networks, 2010, Sensors (ISSN 1424-8220)

[A. Zafeiropoulos et al., 2011] Anastasios Zafeiropoulos, Panagiotis Gouvas, Athanassios Liakopoulos, A Context Model for Autonomic Manage ment of Ad-hoc

Networks, International Conference on Pervasive and Embedded Computing and Communication Systems, 2011, Portugal

[ANSI, 1997] "Public key cryptography for the financial services industry - part 2: The secure hash algorithm (sha-1)", American National Standards Institute, Tech. Rep. American National Standard X9.30.2-1997, 1997

[ATT Report, 2007] Wireless Trends and Expenditures: US Government, www.corp.att.com/stateandlocal/docs/wirelessgov.pdf, Report, 2007

[B]

[B. Jennings et al., 2007]B. Jennings, S. Van der Meer, S. Balasubramaniam, D. Botvich, M.O. Foghlu, W. Donnelly and J. Strassner, "Towards Autonomic Management of Communications Networks", IEEE Communications Magazine, vol.45, no.10, pp.112-121, October 2007

[B. Karp et al., 2004] B. Karp, S. Ratnasamy, S. Rhea, and S. Shenker, "Spurring adoption of dhts with openhash, a public dht service", in Proceedings of the 3rd International Workshop on Peer-to-Peer Systems (IPTPS 2004), Berkeley, California, USA, February 26-27 2004

[B. Krishnamurthy et al., 2001] B. Krishnamurthy, J. Wang, and Y. Xie, "Early measurement of a cluster-based architecture for p2p systems", in Proceedings of the ACM SIGCOOM Internet Measurement Workshop, San Francisco, USA, November 2001

[B. T. Loo et al., 2004] B. T. Loo, R. Huebsch, I. Stoica, and J. M. Hellerstein, "The case for a hybrid p2p search infrastructure", in Proceedings of the 3rd International Workshop on Peer-to-Peer Systems (IPTPS), San Diego, California, USA, February 26-27 2004

[B. Y. Zhao, 2004] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. D.Kubiatowicz, "Tapestry: A resilient global-scale overlay for service deployment", IEEE Journal on Selected Areas in Communications, vol. 22, no. 1, pp. 41–53, January 2004

[B. Yang et al., 2002] B. Yang and H. Garcia-Molina, "Efficient search in peer-to-peer networks", in Proceedings of the 22nd IEEE International Conference on Distributed Computing Systems (ICDCS), July 2002

[C]

[C. Buragohain et al., 2003] C. Buragohain, D. Agrawal, and S. Suri, “A game-theoretic framework for incentives in p2p systems”, in Proceedings of the IEEE P2P 2003, Linkoping, Sweden, September 1-3 2003

[C. Plaxton et al., 1997] C. Plaxton, R. Rajaraman, and A. Richa, “Accessing nearby copies of replicated objects in a distributed environment”, in Proceedings of the 9th Annual ACM Symposium on Parallel Algorithms and Architectures, 1997

[D]

[D. Karger et al., 1997] D. Karger, E. Lehman, T. Leighton, R. Panigrahy, M. Levine, and D. Lewin, “Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the world wide web”, in Proceedings of the twenty-ninth annual ACM symposium on Theory of computing, May 1997, pp. 654–663

[D. Liben-Nowell et al., 2002] D. Liben-Nowell, H. Balakrishnan, and D. Karger, “Analysis of the evolution of peer-to-peer systems”, in Proceedings of the Annual ACM Symposium on Principles of Distributed Computing, Monterey, California, USA, 2002

[D. Loguinov et al, 2003] D. Loguinov, A. Kumar, and S. Ganesh, “Graph-theoretic analysis of structured peer-to-peer systems: routing distances and fault resilience”, in Proceedings of the ACM SIGCOMM, Karlsruhe, Germany, August 25-29 2003, pp. 395–406

[D. Malkhi et al., 2002] D. Malkhi, M. Naor, and D. Ratajczak, “Viceroy: a scalable and dynamic emulation of the butterfly”, in Proceedings of the ACM PODC’02, Monterey, CA, USA, July 2002, pp. 183–192

[D. Qiu et al., 2004] D. Qiu and R. Srikant, “Modeling and performance analysis of bittorrent-like peer-to-peer networks”, in Proceedings of the ACM SIGCOMM, Karlsruhe, Germany, August 30 - September 3 2004

[D. R. Karger et al., 1997] D. R. Karger, E. Lehman, F. T. Leighton, R. Panigrahy, M. S. Levine, and D. Lewin, “Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web”, in Proceedings of the ACM Symposium on Theory of Computing, May 1997, pp. 654–663

[D. S. Wallach, 2002] D. S. Wallach, “A survey of peer-to-peer security issues”, in Proceedings of the International Symposium on Software Security, Tokyo, Japan, November 2002

[D.B. Johnson et al., 1996] D. B. Johnson and D.A. Maltz, “Dynamic Source Routing in Ad Hoc Wireless Networks”, Kluwer Academic, 1996

[E]

[E. K. Lua et al., 2004] E. K. Lua, A. Lin, J. Crowcroft, and V. Tan, “Barterroam: A novel mobile and wireless roaming settlement model”, in Proceedings of the QofIS, 2004, pp. 348–357

[E.K. Lua et al., 2005] E .K. Lua, J. Crowcroft, M. Pias, R. Sharma and S.Lim, “A Survey and Comparison of Peer-to-Peer Overlay Network Schemes”, IEEE Communications Survey and Tutorial, vol.7, no.2, pp.72-93, 2005

[E. Kang et al, 2008] DHT-Based Mobile Service Discovery Protocol for Mobile Ad Hoc Networks, Proceedings of the 4th international conference on Intelligent Computing, pp.610 – 619, 2008

[E. Sit et al., 2002] E. Sit and R. Morris, “Security considerations for peer-to-peer distributed hash tables”, in Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS), Cambridge, MA 2002

[F]

[F. Dabek et al., 2001] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica, “Wide-area cooperative storage with cfs”, in Proceedings of the eighteenth ACM symposium on Operating systems principles, 2001, pp. 202–215

[F. Dabek et al., 2003] F. Dabek, B. Zhao, P. Druschel, and I. Stoica, “Towards a common api for structured peer-to-peer overlays”, in Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS 2003), Berkeley, California, USA, February 20-21 2003

[F. Kaashoek et al., 2003] F. Kaashoek and D. Karger, "Koorde: A simple degree-optimal hash table", in Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS 03), Berkeley, CA, USA, February 20-21 2003

[F. Zhou et al., 2003] F. Zhou, L. Zhuang, B. Y. Zhao, L. Huang, A. D. Joseph, and J. D. Kubiatowicz, "Approximate object location and spam filtering on peer-to-peer systems", in Processings of the Middleware, June 2003

[G]

[G. Kortuem et al., 2001] G. Kortuem, J. Schneider, D. Preuitt, T. Thompson, S. Fickas, and Z. Segall, "When Peer-to-Peer comes face-to-face: collaborative Peer-to-Peer computing in mobile ad-hoc networks", in Peer-to-Peer Computing, 2001, Proceedings. First International Conference on, 27-29 Aug. 2001, pp. 75–91

[G. Kunzmann et al., 2008] G. Kunzmann, A. Binzenhöfer, A. Stäber, "Structured Overlay Networks as an Enabler for Future Internet Services", it - Information Technology, vol.50, pp.376-382, no.6, November 2008

[G. Urdaneta et al., 2011] Guido Urdaneta, Guillaume Pierre and Maarten van Steen, A Survey of DHT Security Techniques, ACM Computing Surveys 43(2), January 2011

[Gartner, 2006] Hype Cycle for Wireless Networking, Report, 2006

[Gergely et al., 2005] Gergely Acs, Levente Buttyan and Istvan Vajda " Provably Secure on-demand Source routing in Mobile Ad-Hoc Networks", Accepted for publication in the IEEE transactions on Mobile Computing, November 28, 2005

[H]

[H. J. Siegel, 1979] H. J. Siegel, "Interconnection networks for simd machines", Computer, vol. 12, no. 6, pp. 57–65, 1979

[I]

[I. Abraham et al., 2004] I. Abraham, D. Malkhi, and O. Dubzinski, "Land: Stretch (1+epsilon) locality aware networks for dhts", in Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA'04), New Orleans, LA., USA, 2004

[I. Clarke et al., 1999] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong. (1999) Freenet: A distributed anonymous information storage and retrieval system, Freenet White Paper, [Online], Available: http://freenetproject.org/freenet.pdf

[I. Stoica et al., 2001] I. Stoica, R. Morris, D. Karger, M.F. Kaashoek and H. Balakrishnan, "Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications", In Proceedings of SIGCOMM 2001, San Deigo, CA, August 2001

[I. Stoica et al., 2003] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan,"Chord: A scalable peer-to-peer lookup protocol for internet applications", IEEE/ACM Transactions on Networking, vol. 11, no. 1, pp. 17–32, 2003

[Informa, 2010] Femtocell Market Status Q1 2010, http://www.femtoforum.org/femto/Files/File/Informa%20Femtocell%20Market%20Status%20Q1%202010%20F.pdf, Report, 2010

[ITU Report, 2010] The World in 2009: ICT Facts and Figures, http://www.itu.int/ITU-D/ict/material/Telecom09_flyer.pdf, Report, 2010

[J]

[J. A. Pouwelse et al., 2004] J. A. Pouwelse, P. Garbacki, D. Epema, and H. Sips, "A measurement study of the bittorrent peer-to-peer file sharing system", Delft University of Technology Parallel and Distributed Systems Report Series, Tech. Rep. Technical Report PDS-2004-007, 2004

[J. Kleinberg, 2000] J. Kleinberg, "The small-world phenomenon: an algorithm perspective", in Proceedings of the thirty-second annual ACM symposium on Theory of computing, 2000, pp. 163–170

[J. Kubiatowicz et al., 2002] J. Kubiatowicz, D. Bindel, Y. Chen, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao, "Oceanstore: An architecture for global-scale persistent storage", in Processings of the ACM ASPLOS, November 2002

[J. Miller, 2004] J. Miller, "Characterization of data on the Gnutella Peer-to-Peer network", in Consumer Communications and Networking Conference, 2004, CCNC 2004, First IEEE, 5-8 Jan. 2004, pp. 489–494

[J. R. Douceur et al., 2002] J. R. Douceur, "The sybil attack", in Proceedings of the First International Workshop on Peer-to-Peer Systems, March 7-8 2002, pp. 251–260.

[Johnson et al., 2001] Johnson D., Maltz D., Broch J., Ad Hoc Networking, Addison-Wesley: Upper Saddle River, NJ, USA, 2001, pp. 139–172

[K]

[K. Aberer et al., 2003] K. Aberer, P. Cudr´ -Mauroux, A. Datta, Z. Despotovic, M. Hauswirth, M. Punceva,eR. Schmidt, and J. Wu, "Advanced peer-to-peer networking: The p-grid system and its applications", PIK Journal - Praxis der Informationsverarbeitung und Kommunikation, Special Issue on P2P Systems, 2003

[K. Lai et al., 2003] K. Lai, M. Feldman, J. Chuang, and I. Stoica, "Incentives for cooperation in peer-to-peer networks", in Proceedings of the Workshop on Economics of Peer-to-Peer Systems, Linkoping, Sweden, June 2003

[K. P. Gummadi et al., 2003] K. P. Gummadi, R. J. Dunn, S. Saroiu, S. D. Gribble, H. M. Levy, and J. Zahorjan, "Measurement, modeling, and analysis of a peer-to-peer file sharing workload", in Processings of the SOSP, Bolton Landing, New York, USA, October 19-22 2003

[K.-A. Skevik et al., 2004] K. A. Skevik, V. Goebel, and T. Plagemann, "Analysis of bittorrent and its use for the design of a p2p based streaming protocol for a hybrid cdn" , Delft University of Technology Parallel and Distributed Systems Report Series, Tech. Rep. Technical Report, June 2004

[K. Tutschkua et al., 2008] K. Tutschkua, P. Tran-Giaa and F.U. Andersenb, "Trends in network and service operation for the emerging future Internet", International Journal of Electronics and Communications, vol.62, pp.705-714, October 2008

[L]

[L. Adamic et al, 2001] L. Adamic, R. Lukose, A. Puniyani, and B. Huberman, "Search in power-law networks", Physical Review E, vol. 64, 2001

[L. Alima et al., 2003] L. Alima, S. El-Ansary, P. Brand, and S. Haridi, "Dks(n,k,f): a family of low communication, scalable and fault-tolerant infrastructures for p2p

applications", in Proceedings of the 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid, Monterey, California, USA, 2003, pp. 344–350

[L. Barriere et al., 2001] L. Barriere, P. Fraigniaud, E. Kranakis, and D. Krizanc, "Efficient routing in networks with long range contacts", in Proceedings of the 15th International Conference on Distributed Computing, vol. 2180, 2001, pp. 270–284

[L. Breslau et al., 1999] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web caching and zipf-like distribution: Evidence and implications", in Proceedings of the IEEE INFOCOM, 1999

[L. Guo et al., 2007] L. Guo, S. Chen, Z. Xiao, E. Tan, X. Ding, and X. Zhang, "A performance study of BitTorrent-like peer-to-peer systems", Selected Areas in Communications, IEEE Journal on,vol. 25, no. 1, pp. 155–169, Jan. 2007

[L. P. Cox et al., 2002] L. P. Cox, C. D. Murray, and B. D. Noble, "Pastiche: making backup cheap and easy", SIGOPS Oper. Syst. Rev., vol. 36, no. SI, pp. 285–298, 2002

[L. Peterson, 2003] L. Peterson, T. Anderson, D. Culler, T. Roscoe, "A blueprint for introducing disruptive technology into the internet", SIGCOMM Computers Communications Rev., vol. 33, no. 1, pp. 59–64, 2003

[Lee et al., 2008] Lee, H., Lee, K., Lee, Y., A Self-Organized Head Selection for Hierarchical Routing in Wireless Sensor Networks, In Proceedings of the 3rd international Workshop on Self-Organizing Systems, Vienna, Austria, December 2008

[M]

[M. A. Jovanovic et al., 2001] M. A. Jovanovic, F. Annexstein, and K. Berman, "Scalability issues in large peer-to-peer networks - a case study of gnutella", Delft University of Technology Parallel and Distributed Systems Report Series, University of Cincinnati, Tech. Rep. Technical Report, 2001, [Online], Available: http://www.ececs.uc.edu/~mjovanov/Research/paper.html

[M. Bisignano et al., 2005] M. Bisignano, G. Di Modica, and O. Tomarchio, "JMobiPeer a middleware for mobile Peer-to-Peer computing in MANETs", in Distributed Computing Systems Workshops, 2005, 25th IEEE International Conference on, 6-10 June 2005, pp. 785–791

[M. Bisignano et al., 2007] M. Bisignano, G. Di Modica, O. Tomarchio and L. Vita, "P2P over Manet: a comparison of cross-layer approaches", 18th International Conference on Database and Expert Systems Applications, DEXA '07, pp.814-818, 3-7 Sept. 2007

[M. Caleffi et al., 2009] M. Caleffi,L. Paura, "P2P over MANET: Indirect Tree-based Routing", IEEE International Conference on Pervasive Computing and Communications, 2009

[M. Castro et al., 2002] M. Castro, P. Druschel, A. Ganesh, A. Rowstron, and D. S. Wallach, "Secure routing for structured peer-to-peer overlay networks", SIGOPS Oper. Syst. Rev., vol. 36, no. SI, pp. 299–314, 2002

[M. Castro et al., 2002] M. Castro, P. Druschel, A. M. Kermarrec, and A. Rowstron, "SCRIBE: A large-scale and decentralized application-level multicast infrastructure", in IEEE Journal on Selected Areas in Communications (JSAC) (Special issue on Network Support for Multicast Communications),October 2002

[M. Castro et al., 2004] M. Castro, M. Costa, and A. Rowstron, "Performance and dependability of structured peer-to-peer overlays", in Proceedings of the 2004 International Conference on Dependable Systems and Networks, Palazzo dei Congressi, Florence, Italy, June 28 - July 1 2004

[M. Castro, 2003] M. Castro, P. Druschel, A. M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh, "Splitstream: high-bandwidth multicast in cooperative environments", in Proceedings of the nineteenth ACM symposium on Operating systems principles, October 19-20 2003, pp. 298–313

[M. Faloutsos et al., 1999] M. Faloutsos, P.Faloutsos, and C. Faloutsos, "On power-law relation-ships of the internet topology", in Processings of the SIGCOMM 1999

[M. G. Williams, 2005] M. G. Williams, "Directions in Media Independent Handover", IEICE Trans. Fundam. Electron. Commun. Comput. Sci., vol. E88-A, no. 7, pp. 1772–1776, 2005

[M. Jelacity et al., 2010] M. Jelacity, A. Montresor, G. Jesi and S. Voulgaris, "The Peersim simulator", http://peersim.sf.net, accessed March 2010

[M. Jelacity et al.-1, 2005] M. Jelasity and O. Babaoglu, "T-Man: Gossip-Based Overlay Topology Management", In Proceedings of Engineering Self-Organising Systems, ISBN: 978-3-540-33342-5, July 2005

[M. Jelacity et al. -2, 2005] M. Jelasity, A. Montresor and O.Babaoglu, "Gossip-based aggregation in large dynamic networks", ACM Transactions on Computer Systems, vol.23, no.3, pp.219-252, 2005

[M. Matuszewski et al., 2006] M. Matuszewski, N. Beijar, J. Lehtinen, and T. Hyyrylainen, "Mobile Peer-to-Peer content sharing application", in Consumer Communications and Networking Conference, 2006, CCNC 2006, 2006 3rd IEEE, vol. 2, 8-10 Jan. 2006, pp. 1324–1325

[M. Naor et al., 2003] M. Naor and U. Wieder, "A simple fault tolerant distributed hash table", in Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS 03), Berkeley, California, USA, February 20-21 2003

[M. Naor et al., 2003] M. Naor and U. Wieder, "Novel architectures for p2p applications: the continuous-discrete approach", in Proceedings of the 15th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA 2003), San Diego, California, USA, June 7-9 2003, pp. 50–59

[M. Uma et al., 2009] M. Uma, G. Padmavathi, A Comparative study and performance evaluation of reactive quality of service routing protocols in Mobile Ad-hoc Networks, Journal of Theoretical and Applied Information Technology, 2009

[M. Waldman et al., 2000] M. Waldman, A. D. Rubin, and L. F. Cranor, "Publius: a robust, tamper-evident, censorship-resistant, web publishing system", in Processings of the Ninth USENIX Security Symposium, Denver, CO, USA, 2000

[N]

[N. D. de Bruijn, 1946] N. D. de Bruijn, "A combinatorial problem", Koninklijke Netherlands: Academe Van Wetenschappen, vol. 49, pp. 758–764, 1946

[N. Kotilainen et al., 2005] N. Kotilainen, M. Weber, M. Vapa, and J. Vuori, "Mobile Chedar - a Peer-to-Peer middleware for mobile devices", in Pervasive Computing and

Communications Workshops, 2005, PerCom 2005 Workshops, Third IEEE International Conference on, 8-12 March 2005, pp.86–90

[N. Moghim et al., 2002] N. Moghim, F. Hendessi, and N. Movehhedinia, "An improvement on adhoc wireless network routing based on aodv", in the 18th International Conference on Communication Systems ICCS, vol.2. IEEE, 2002, pp 1068 -1070

[NIST, 1995] "Secure hash standard", NIST, U.S. Dept. of Commerce, National Technical Information Service FIPS 180-1, April 1995

[P]

[P. Druschel et al, 2001] P. Druschel and A. Rowstron, "PAST: A large-scale, persistent peer-to-peer storage utility", in Proceedings of the 8th Workshop on Hot Topics in Operating Systems (HotOS-VIII). Schloss Elmau, Germany: IEEECompSoc, May 2001

[P. Fraigniaud et al., 2003] P. Fraigniaud and P. Gauron, "The content-addressable networks d2b", Laboratoire de Recherche en Informatique, Universit'e de Paris Sud, Tech. Rep. Technical Report 1349, Jan. 2003

[P. Francis et al., 2000] "Yoid: Extending the internet multicast architecture", unpublished, April 2000. http://www .aciri.org/yoid/docs/index.html

[P. Ganesan et al., 2003] P. Ganesan, Q.Sun, and H. Garcia-Molina, "Yappers: A peer-to-peer lookup service over arbitrary topology," in Proceedings of the IEEE Infocom 2003, San Francisco, USA, March 30 - April 1 2003

[P. Golle et al., 2001] P. Golle, K. Leyton-Brown, I. Mironov, and M. Lillibridge, "Incentives for sharing in peer-to-peer networks," Lecture Notes in Computer Science, vol. 2232, pp. 75+, 2001

[P. Gouvas et al., 2007] Panagiotis Gouvas, Thanassis Bouras, Gregoris Mentzas: An OSGi-Based Semantic Service-Oriented Device Architecture. OTM Workshops (2) 2007: 773-782

[P. Gouvas et al. -1, 2010] Panagiotis Gouvas, Anastasios Zafeiropoulos, Athanassios Liakopoulos, Gregoris Mentzas and Nikolas Mitrou: Integrating Overlay Protocols for

Providing Autonomic Services in Mobile Ad-hoc Networks 2010 IEICE Transactions on Communications

[P. Gouvas et al.- 2, 2010] Panagiotis Gouvas, Anastasios Zafeiropoulos, Athanassios Liakopoulos: Gossiping for Autonomic Estimation of Network-Based Parameters in Dynamic Environments. OTM Workshops 2010: 358-366

[P. Maymounkov et al., 2002] P. Maymounkov and D. Mazi` res, "Kademlia: A peer-to-peer information system based on the xor metric," in Processings of the IPTPS, Cambridge, MA, USA, February 2002, pp. 53–65

[Perkins et al., 1991] Perkins, C.E., & Royer, E.M. (1991). Adhoc on-demand distance vector routing.Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications (WMCSA) (pp.90-100)

[Q]

[Q. Hofstatter et al., 2008] Q. Hofstatter, S. Zols,M. Michel,Z. Despotovic,W. Kellerer, Chordella - A Hierarchical Peer-to-Peer Overlay Implementation for Heterogeneous, Mobile Environments, Eighth International Conference on Peer-to-Peer Computing , 2008

[Q. Lv et al., 2002] Q. Lv, S. Ratnasamy, and S. Shenker, "Can heterogeneity make gnutella scalable?" in Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS), Cambridge, MA, USA, February 2002

[R]

[R. Albert et al., 1999] R. Albert, H. Jeong, and A. Barabasi, "Diameter of the world wide web," Nature, vol. 401, pp. 130–131, 1999

[R. Albert et al., 2000] R. Albert and A. Barabasi, "Attack and tolerance in complex a networks," Nature, vol. 406, no. 378, 2000

[R. Bless et al., 2008] R. Bless, C. Hiibsch, S. Mies and O.P. Waldhorst, "The Underlay Abstraction in the Spontaneous Virtual Networks (SpoVNet) Architecture", Next Generation Internet Networks, NGI 2008 , pp.115-122, 28-30 April, 2008

[R. Cox et al, 2002] R. Cox, A. Muthitacharoen, and R. Morris, “Serving dns using chord,” in Proceedings of the First International Workshop on P eer-to-Peer Systems, March 2002

[R. Dingledine et al., 2001] R. Dingledine, M. J. Freedman, and D. Molnar, “Accountability measures for peer-to-peer systems,” in Peer-to-Peer: Harnessing the Power of Disruptive Technologies, D. Derickson, Ed. O’Reilly and Associates, November 2001

[R. Schollmeier et al., 2003]R. Schollmeier, I. Gruber, and F. Niethammer, “Protocol for Peer-to-Peer Networking in Mobile Environments,” in Computer Communications and Networks, 2003. ICCCN 2003, Proceedings, the 12th International Conference on, 20-22 Oct. 2003, pp. 121–127

[R. Winter et al., 2004] R. Winter, T. Zahn, and J. Schiller, “Dynamo: A topology-aware p2p overlay network for dynamic, mobile ad-hoc environments,” Telecommunication Systems, vol. 27, no. 2, p. 321, 2004

[ReadWrightWeb, 2010] Top 10 Mobile Trends of 2010, Report, 2010

[S]

[S. Bellovin, 2001] S. Bellovin, “Security aspects of napster and gnutella,” in Proceedings of the 2001 Usenix Annual Technical Conference, Boston , Massachusetts, USA, June 2001

[S. Iyer et al., 2002] S. Iyer, A. Rowstron, and P. Druschel, “Squirrel: A decentralized peer-to-peer web cache,” in Proceedings of the 21st Symposium on Principles of Distributed Computing (PODC), Monterey, C alifornia, USA, July 21-24 2002

[S. Q. Zhuang, 2001] S. Q. Zhuang, B. Y. Zhao, A. D. Joseph, R. H. Katz, and J. D. Kubiatowicz, “Bayeux: an architecture for scalable and fault-tolerant wide-area data dissemination,” in Proceedings of the 11th international workshop on Network and operating systems support for digital audio and video, 2001, pp. 11 –20

[S. Ratnasamy et al., 2001] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, “A scalable content addressable network,” in Processings of the ACM SIGCOMM, 2001, pp. 161–172

[S. Rhea et al., 2003] S. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz, "Handling churn in a dht," in Proceedings of the 2nd International Workshop on Peer-to-Peer (IPTPS'03), February 2003

[S. Rhea, 2001] S. Rhea, C. Wells, P. Eaton, D. Geels, B. Zhao, H. Weatherspoon, and J. Kubiatowicz, "Maintenance-free global storage in oceanstore," in IEEE Internet Computing, 2001

[S. Saroiu et al., 2002] S. Saroiu, P. K. Gummadi, and S. D. Gribble, "A measurement study of peer-to-peer file sharing systems," in Proceedings of the Multimedia Computing and Networking (MMCN), San Jose, California, USA, January 2002

[T]

[T. Hakkarainen et al., 2005] T. Hakkarainen, V. Savikko, and A. Lattunen, "Generic engine for collaborative mobile applications," in Proceedings of the IADIS International Conference WWW/Internet 2005 (ICWI2005), 2005, pp. 243 – 246

[U]

[U. Manber, 1994] U. Manber, "Finding similar files in a large file system," in Proceedings of the USENIX Winter 1994 Conference, January 1994, pp. 1–10

[V]

[Visionmobile, 2009] Megatrends2010, http://www.visionmobile.com/rsc/researchreports/Megatrends-2010-VisionMobile-(Rutberg-version).pdf, Report, 2010

[W]

[W. Ding et al., 2007] W. Ding and S.S Iyengar, "Bootstrapping Chord over MANETs - All Roads Lead to Rome", IEEE Wireless Communications and Networking Conference, 2007

[W. J. Bolosky et al., 2000] W. J. Bolosky, J. R. Douceur, D. Ely, and M. Theimer, "Feasibility of a serverless distributed file system deployed on an existing set of desktop

pcs," in Proceedings of the 2000 ACM SIGMETRICS international conference on Measurement and modeling of compute r systems, June 2000, pp. 34–43

[X]

[X. Li et al., 2002] X. Li and C. Plaxton, "On name resolution in peer to peer networks,"in Proceedings of the 2nd ACM International workshop on principles of mobile computing, Monterey, California, USA, 2002, pp. 82–89

[Y]

[Y. Chawathe et al., 2003] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker,"Making gnutella-like p2p systems scalable," in Proceedings of the ACM SIGCOMM, Karlsruhe, Germany, August 25-29 2003

[Y. Raivio, 2005] Mobile Peer-to-Peer in Cellular Networks, Proceedings of HUTT-110.51 Seminar on Internetworking, Helsinki Institute of Technology, 2005

[Y. Rekhter et al., 1993] Y. Rekhter and T. Li. (1993) An architecture for ip address allocation with cidr. IETF Internet draft RFC1518. [Online]. Available: http://www.isi.edu/in-notes/rfc1518.txt/

[Y. Shavitt et al., 2004] Y. Shavitt,T. Tankel, On the curvature of the internet and its usage for overlay construction and distance estimation, in Proceedings of the IEEE INFOCOM '04 Conference, Hong Kong, March 7-11 2004

[Y. Wang et al., 2007] Y. Wang, X. Yun, and Y. Li, "Analyzing the Characteristics of Gnutella Overlays", Fourth International Conference on Information Technology, pp.1095 - 1100, April 2007

[Z]

[Z. Despotovic et al., 2004] Z. Despotovic and K. Aberer, "A probabilistic approach to predict peers' performance in p2p networks," in Proceedings of the Eighth International Workshop on Cooperative Information Agents (CIA 2004),Erfurt , Germany, September 27-29 2004

[3]

[3GAmericas, 2010] 3G Global Status Update July 27 2010, http://www.3gamericas.org/documents/Global%20Status%20Update%20July%2027%202010.pdf, Report, 2010

[3GPP, 2007] 3GPP, "TS 43.328 Generic Access to the A/Gb Interface , Stage 2," 3GPP, Tech. Rep., 2007

Publications

International Journals:

[**J1**] **Panagiotis Gouvas**, Anastasios Zafeiropoulos, Athanassios Liakopoulos, Gregoris Mentzas and Nikolas Mitrou: Integrating Overlay Protocols for Providing Autonomic Services in Mobile Ad-hoc Networks 2010 IEICE Transactions on Communications (Science Citation Index Expanded, Impact Factor 2009: 0.7)

[**J2**] Anastasios Zafeiropoulos, **Panagiotis Gouvas**, Athanassios Liakopoulos, Gregoris Mentzas and Nikolas Mitrou , NEURON: Enabling Autonomicity in Wireless Sensor Networks ,2010 , Sensors (ISSN 1424-8220) (Science Citation Index Expanded, Impact Factor 2009: 1.8)

International Conferences:

[**C1**] **Panagiotis Gouvas**, Thanassis Bouras, Anastasios Zafeiropoulos, Athanssios Ch. Liakopoulos, UbiChord: Services Provision in Dynamic Networks based on p2p Protocols, 18th International Conference on Telecommunication , ICT 2011

[**C2**] Anastasios Zafeiropoulos, Athanassios Liakopoulos, **Panagiotis Gouvas**: A context model for autonomic management of ad-hoc networks, Int'l Conf. on Pervasive and Embedded Computing and Communication Systems 2011

[**C3**] Athanassios Liakopoulos, Anastasios Zafeiropoulos, Constantinos Marinos, Mary Grammatikou, Nicolay Tcholtchev and **Panagiotis Gouvas**, "Applying distributed monitoring techniques in autonomic networks", 2nd IEEE International Workshop on Management of Emerging Networks and Services (IEEE MENS 2010), in conjunction with IEEE GLOBECOM 2010, Miami, Florida, USA (2010)

[**C4**] **Panagiotis Gouvas**, Anastasios Zafeiropoulos and Athanassios Liakopoulos, "Gossiping for Autonomic estimation of Network based parameters in dynamic environments", Fifth International Workshop on MObile and NEtworking Technologies for social applications (MONET´10), Crete, Greece (2010)

[**C5**] Giannis Verginadis, **Panagiotis Gouvas**, Thanassis Bouras, Gregoris Mentzas: Conceptual modelling of service-oriented programmable smart assistive environments. PETRA 2008: 11

[**C6**] Athanasios Bouras, **Panagiotis Gouvas**, Gregoris Mentzas: ENIO: An Enterprise Application Integration Ontology. DEXA Workshops 2007: 419 -423

[**C7**] **Panagiotis Gouvas**, Thanassis Bouras, Gregoris Mentzas: An OSGi-Based Semantic Service-Oriented Device Architecture. OTM Workshops (2) 2007: 773 -782

[**C8**] Athanasios Bouras, **Panagiotis Gouvas**, Dimitrios Kourtesis, Gregoris Mentzas: Semantic Integration Of Business Applications Across Collaborative Value Networks. Virtual Enterprises and Collaborative Networks 2007: 539 -546

[**C9**] Dimitrios Kourtesis, Iraklis Paraskakis, Andreas Friesen, **Panagiotis Gouvas**, Athanasios Bouras: Web Service Discovery In A Semantically Extended Uddi Registry: The Case Of Fusion. Virtual Enterprises and Collaborative Networks 2007: 547 -554

[**C10**] Giannis Verginadis, **Panagiotis Gouvas**, Gregoris Mentzas: An Hybrid Intermediation Architectural Approach for Integrating Cross-Organizational Services. OTM Workshops 2005: 452-460

[**C11**] **Panagiotis Gouvas**, G. Magiorkinis, Athanasios Bouras, Dimitris Paraskevis, Dimitrios Alexandrou, A. Hatzakis, Gregoris Mentzas: Web Service-Enabled Grid-Based Platform for Drug Resistance Management. Panhellenic Conference on Informatics 2005: 469 -479

[**C12**] Christos Tatsiopoulos, G. Vardangalos, Dimitrios Alexandrou, **Panagiotis Gouvas**: A Distributed and Multilingual Search System for Exploring Large Image Databases. ICWI 2003: 501-507

Book Chapters:

[**B1**] Bouras **A., Gouvas P**. & Mentzas G. (2009). Dynamic Data Mediation in Enterprise Application Integration Scenarios. In G. Mentzas and A. Friesen (Eds) Semantic Enterprise Application Integration for Business Processes: Service-Oriented Frameworks , ISBN: 978-1-60566-804-8, pages 347, Publ. by IGI Global, 2009.

[**B2**] Bouras **A., Gouvas P**. & Mentzas G. (2006). A Semantic Service-Oriented Architecture for Business Process Fusion. In Salam, A. F., & Stevens, J. (Eds.), Semantic Web Technologies and eBusiness: Virtual Organization and Business Process Automation, Hershey PA, Idea Group Inc.

Standardization Bodies:

[**S1**] European Telecommunications Standardization Institute (ETSI)-Draft, in the working group of "Autonomic network engineering for the self-managing Future Internet" (a.k.a. AFI): "AFI 001:Scenarios, Use Cases, and Requirements for Autonomic/Self-Managing Future Internet". For ETSI members, the standardisation draft is also available at: http://portal.etsi.org/portal/server.pt/community/AFI/344

[**S2**] ITF - Interconnecting Smart Objects Workshop: Proposal-58: 58.Interconnecting smart objects through an overlay networking architecture by Anastasios Zafeiropoulos, Athanassios Liakopoulos and **Panagiotis Gouvas**

Σύντομο Βιογραφικό Σημείωμα – Γκουβά Παναγιώτη

Ο κ. Παναγιώτης Γκουβάς είναι Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών του Εθνικού Μετσόβιου Πολυτεχνείου (ΕΜΠ) με περίοδο φοίτησης 1999-2004. Παράλληλα με την φοιτητική του ιδιότητα ο κ. Γκουβάς από το Σεπτέμβριο του 2001 εργαζόταν ως προγραμματιστής στην εταιρεία European Profiles (www.europeanprofiles.gr) συμμετέχοντας στην αποπεράτωση έργων πληροφορικής στην Ελλάδα αλλά και στην Ευρωπαϊκή Ένωση. Στα πλαίσια της εργασίας του, συνέβαλε ενεργά στον σχεδιασμό και την ανάπτυξη ερευνητικών και εμπορικών έργων αποκτώντας την αντίστοιχη εμπειρία (ενδεικτικά WEIGHT-INFO (IST-2000-2625), ML-IMAGES (EDC-22046)).

Από τον Νοέμβρη του 2004 ο κ. Γκουβάς εργάστηκε ως ερευνητής στο Εργαστήριο Διοίκησης Πληροφοριακών Συστημάτων της Σχολής Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών του Εθνικού Μετσόβιου Πολυτεχνείου (www.imu.ntua.gr). Παράλληλα, ο κ. Γκουβάς ξεκινά την ίδια χρονική περίοδο την εκπόνηση της διδακτορικής διατριβής του.

Στα πλαίσια της διατριβής συμμετείχε ενεργά στις ερευνητικές δραστηριότητες του εργαστηρίου μέσω ερευνητικών έργων που χαρακτηρίζονται από σημαντική πρωτοτυπία και υψηλή επιστημονική στάθμη. Ενδεικτικά έργα είναι τα ακόλουθα:

- 2005 – 2008: “FUSION: Business process fusion based on Semantically-enabled Service-oriented Business Applications” Στόχος του έργου ήταν η δημιουργία κατάλληλου πληροφοριακού συστήματος για την αντιμετώπιση προβλημάτων διαλειτουργικότητας μεταξύ ετερογενών πληροφοριακών συστημάτων.
- 2006-2008: “SOPRANO: Service-oriented Programmable Smart Environments for Older Europeans” Σκοπός του έργου ήταν η δημιουργία μιας πλατφόρμας συνεργαζόμενων υπηρεσιών στα πλαίσια ενός έξυπνου οικιακού περιβάλλοντος.

Παράλληλα ο κ. Γκουβάς εργάστηκε ως εξωτερικός συνεργάτης του Εθνικού Δικτύου Έρευνας και Τεχνολογίας στο ακόλουθο έργο:

- 2007-2010: “EFIPSANS: Exposing the Features in IP version Six protocols that can be exploited/extended for the purposes of designing/building autonomic Networks and Services” Το έργο στόχευε να χρησιμοποιήσει τα ιδιαίτερα χαρακτηριστικά της τεχνολογίας IPv6 για να βελτιώσει το σχεδιασμό και να διευκολύνει την υλοποίηση αυτόνομων δικτύων. Στα πλαίσια του έργου σχεδιάστηκαν επεκτάσεις στο πρωτόκολλο IPv6 που στοχεύουν στην υποστήριξη αυτόνομων λειτουργιών στο δίκτυο.

Τέλος ο κ. Γκουβάς συμμετείχε ενεργά κατά τη διάρκεια της διατριβής του και στο εκπαιδευτικό έργο της Σχολής Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών του ΕΜΠ ενώ από το 2007 είναι ιδρυτικό και ενεργό στέλεχος της εταιρείας UBITECH (www.ubitech.eu) που δραστηριοποιείται στον τομέα της πληροφορικής και των επικοινωνιών.

Δημοσιευμένα Άρθρα σε Διεθνή Επιστημονικά Περιοδικά και Συνέδρια :

[J1] Panagiotis Gouvas, Anastasios Zafeiropoulos, Athanassios Liakopoulos, Gregoris Mentzas and Nikolas Mitrou: Integrating Overlay Protocols for Providing Autonomic Services in Mobile Ad-hoc Networks 2010 IEICE Transactions on Communications (Science Citation Index Expanded, Impact Factor 2009: 0.7)

[J2] Anastasios Zafeiropoulos, **Panagiotis Gouvas**, Athanassios Liakopoulos, Gregoris Mentzas and Nikolas Mitrou , NEURON: Enabling Autonomicity in Wireless Sensor Networks ,2010 , Sensors (ISSN 1424-8220) (Science Citation Index Expanded, Impact Factor 2009: 1.8)

[A1] Anastasios Zafeiropoulos, Athanassios Liakopoulos, **Panagiotis Gouvas**: A context model for autonomic management of ad-hoc networks, Int'l Conf. on Pervasive and Embedded Computing and Communication Systems 2011 (to-appear)

[A2] Giannis Verginadis, **Panagiotis Gouvas**, Thanassis Bouras, Gregoris Mentzas: Conceptual modelling of service-oriented programmable smart assistive environments. PETRA 2008: 11

[A3] Athanasios Bouras, **Panagiotis Gouvas**, Gregoris Mentzas: ENIO: An Enterprise Application Integration Ontology. DEXA Workshops 2007: 419-423

[A4] Panagiotis Gouvas, Thanassis Bouras, Gregoris Mentzas: An OSGi-Based Semantic Service-Oriented Device Architecture. OTM Workshops (2) 2007: 773-782

[A5] Athanasios Bouras, **Panagiotis Gouvas**, Dimitrios Kourtesis, Gregoris Mentzas: Semantic Integration Of Business Applications Across Collaborative Value Networks. Virtual Enterprises and Collaborative Networks 2007: 539-546

[A6] Dimitrios Kourtesis, Iraklis Paraskakis, Andreas Friesen, **Panagiotis Gouvas**, Athanasios Bouras: Web Service Discovery In A Semantically Extended Uddi Registry: The Case Of Fusion. Virtual Enterprises and Collaborative Networks 2007: 547-554

[A7] Giannis Verginadis, **Panagiotis Gouvas**, Gregoris Mentzas: An Hybrid Intermediation Architectural Approach for Integrating Cross-Organizational Services. OTM Workshops 2005: 452-460

[A8] Panagiotis Gouvas, G. Magiorkinis, Athanasios Bouras, Dimitris Paraskevis, Dimitrios Alexandrou, A. Hatzakis, Gregoris Mentzas: Web Service-Enabled Grid-Based Platform for Drug Resistance Management. Panhellenic Conference on Informatics 2005: 469-479

[A9] Christos Tatsiopoulos, G. Vardangalos, Dimitrios Alexandrou, **Panagiotis Gouvas**: A Distributed and Multilingual Search System for Exploring Large Image Databases. ICWI 2003: 501-507