
ΔΗΜΟΚΡΙΤΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΡΑΚΗΣ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

# Αρχιτεκτονικές Μελλοντικού Διαδικτύου στο Διαδίκτυο των πραγμάτων

Διπλωματική εργασία

Συντίλα Δέσποινα

Επιβλέπων Καθηγητής

κ.Τσαουσίδης Βασίλειος

Ξάνθη, 2021

# Περίληψη

Το Ίντερνετ έχει εισέλθει σε νέα πεδία με την πρόσβαση σε αυτό να γίνεται ολοένα γρηγορότερη και φθηνότερη. Μέχρις στιγμής έχει σημειώσει αξιοσημείωτη πορεία στην ολοένα και αυξανόμενη ζήτηση, δεδομένου ότι τα θεμέλια του τέθηκαν πριν τρεις δεκαετίες. Ωστόσο, η μελλοντική αρχιτεκτονική του Διαδικτύου θα πρέπει να ανταπεξέλθει όχι μόνο στη ραγδαία αύξηση των χρηστών και των συσκευών, αλλά και σε μία πληθώρα νέων ποικίλων εφαρμογών και υπηρεσιών. Χαρακτηριστικό παράδειγμα αποτελεί η εμφάνιση του Διαδικτύου των Πραγμάτων (IoT) που άκμασε τα τελευταία χρόνια και συνεχίζει να εξελίσσεται αλματωδώς.

Εμπνευσμένη από το γεγονός ότι το Ίντερνετ χρησιμοποιείται κυρίως για τη διάδοση της πληροφορίας και όχι για την επικοινωνία μεταξύ των χρηστών, η αρχιτεκτονική Information Centric Networking (ICN) στοχεύει να καλύψει τις ανάγκες τόσο του σήμερα όσο και του αύριο καλύτερα από την υπάρχουσα αρχιτεκτονική του Ίντερνετ αναπτύσσοντας κατάλληλους μηχανισμούς για να εστιάσει στην άκρη του δικτύου. Το μοντέλο αυτό μπορεί, επιπλέον, να εκμεταλλευτεί τα πλεονεκτήματα της αρχιτεκτονικής Delay-Tolerant Networking, ώστε να αντιτίθεται στην ανάγκη για συνδεσιμότητα από άκρο σε άκρο. Ως εκ τούτου, αναπτύχθηκε η αρχιτεκτονική UMOBILE που εστιάζει στη κινητικότητα και είναι προσανατολισμένη στις υπηρεσίες, με σκοπό να παραδίδει αποτελεσματικά τόσο περιεχόμενο πληροφορίας όσο και υπηρεσίες στους χρήστες, ιδιαίτερα σε απομακρυσμένες περιοχές.

Στην προσπάθεια μας να οδηγήσουμε τις υπηρεσίες στα άκρα του δικτύου αξιοποιήσαμε “lightweight” τεχνολογίες εικονικοποίησης (virtualization), όπως η τεχνολογία των Docker Container που εκμεταλλεύεται καλύτερα τους διαθέσιμους πόρους του συστήματος σε σχέση με την τεχνολογία των Virtual Machines (VMs). Τα Docker Container είναι απομονωμένα περιβάλλοντα που δίνουν τη δυνατότητα ανάπτυξης εφαρμογών με μεγαλύτερη απλότητα, ταχύτητα και ασφάλεια.

Στην παρούσα διπλωματική εργασία μελετήσαμε αρχικά τις μελλοντικές αρχιτεκτονικές του Διαδικτύου καθώς και την τεχνολογία των Container. Χρησιμοποιήσαμε τα Docker Container για να αναπτύξουμε τις εφαρμογές των πρωτοκόλλων DTN και NDN, σχηματίζοντας μία βέλτιστη εικόνα για τη δημιουργία αυτών. Αποδείξαμε με την πραγματοποίηση πειραμάτων πως ο συνδυασμός αυτών μπορεί να βελτιώσει σημαντικά την ανάκτηση των δεδομένων σε ένα περιβάλλον που εμφανίζει διακοπτόμενες συνδέσεις.

# Ευχαριστίες

Θα ήθελα να ευχαριστήσω θερμά τον επιβλέποντα καθηγητή της διπλωματικής εργασίας μου κ. Τσαουσίδη Βασίλειο που μου έδωσε την ευκαιρία να ασχοληθώ με αυτό το θέμα. Η συνεργασία μας υπήρξε ιδιαίτερα εποικοδομητική και τον ευχαριστώ ιδιαίτερα για τις συμβουλές και την καθοδήγηση του. Επίσης θα ήθελα να ευχαριστήσω τους υποψήφιους διδάκτορες Σάρρο Χρήστο-Αλέξανδρο και Δεμίρογλου Βασίλειο για τις πολύτιμες οδηγίες και το χρόνο τους.

Τέλος, θα ήθελα να πω ένα μεγάλο ευχαριστώ στην οικογένεια μου και στους φίλους μου για όλη τη στήριξη που μου παρείχαν. Ιδιαίτερα θα ήθελα να ευχαριστήσω τον αδερφό μου στον οποίο και αφιερώνεται η εργασία αυτή.

# Περιεχόμενα

# Κεφάλαιο 1

# Εισαγωγή

Το Διαδίκτυο εξελίσσεται προς πολλές κατευθύνσεις και με μεγάλες ταχύτητες. Τα τελευταία χρόνια οι χρήστες κινητής τηλεφωνίας και η κίνηση των δεδομένων αποτελούν το κυρίαρχο κομμάτι του δικτύου, ενώ η ανάπτυξη ιδεών όπως το Διαδίκτυο των πραγμάτων (IoT) φέρνουν δισεκατομμύρια διασυνδεδεμένες ετερογενείς συσκευές στο προσκήνιο [1]. Η τεράστια αυτή ανέλιξη του Διαδικτύου και η εισαγωγή νέων εφαρμογών για την κάλυψη των αναγκών που προέκυψαν, έχει δημιουργήσει νέες απαιτήσεις από την αρχιτεκτονική του, όπως την κινητικότητα, την ασφάλεια,την εμπιστοσύνη και πολλές άλλες [2].

Ως εκ τούτου η ανάπτυξη νέων αρχιτεκτονικών καθίσταται απαραίτητη. Στην παρούσα εργασία μελετήθηκαν οι παρακάτω αρχιτεκτονικές:

- Delay Tolerant Networking (DTN)
- Named Data Networking (NDN)
- Umobile

Η αρχιτεκτονική Delay Tolerant Networking (DTN) [3] έχει σχεδιαστεί ειδικά για ετερογενή δίκτυα και μπορεί να εφαρμοστεί σε απομονωμένα περιβάλλοντα, σε δίκτυα που παρουσιάζουν μεγάλες καθυστερήσεις και μπορεί να χρησιμοποιηθεί και σε δίκτυα που παρουσιάζουν διακοπτόμενες συνδέσεις, όπως τα δίκτυα αισθητήρων. Πιο συγκεκριμένα, η αρχιτεκτονική DTN παρέχει μια μέθοδο για τη διασύνδεση ετερογενών δικτύων χρησιμοποιώντας την τεχνική αποθήκευσης και προώθησης ( store-and-forward) για τη δρομολόγηση μηνυμάτων, με σκοπό να αντιμετωπίσουν τις διακοπτόμενες συνδέσεις. Το πρωτόκολλο Bundle προωθεί τα πακέτα περίπου με τον ίδιο τρόπο όπως και το πρωτόκολλο IP, όμως τα εξερχόμενα πακέτα-δέσμες (bundles) αποθηκεύονται τοπικά στη μνήμη έως ότου εμφανιστούν διαθέσιμοι σύνδεσμοι προώθησης. Οι προορισμοί εκφράζονται με ονόματα και όχι με διευθύνσεις, καθώς η θέση του κόμβου προορισμού μπορεί να αλλάξει ενώ τα δεδομένα βρίσκονται εν κινήση [4].

Η αρχιτεκτονική Named Data Networking (NDN) [5] άκμασε τα τελευταία χρόνια και αποτελεί ίσως μία από τις πιο γνωστές αρχιτεκτονικές προσανατολισμένες στην πληροφορία (Information-Centric Networking). Στην ουσία η ικανοποίηση ενός αιτήματος του χρήστη δε βασίζεται στην αναζήτηση του περιεχομένου με βάση τη διεύθυνση (IP) στην οποία βρίσκεται η πληροφορία, αλλά με βάση μία περιγραφή-ταυτότητα που έχει δοθεί στην ίδια την πληροφορία. Το NDN είναι μία πολλά υποσχόμενη αρχιτεκτονική για τον κλάδο του IoT [6]. Οι καινοτόμες ιδέες της αρχιτεκτονικής αυτής, όπως η ονοματοδοσία του περιεχομένου, δρομολόγηση βάση ονόματος και η προσωρινή αποθήκευση εντός του δικτύου, συνάδουν με τα πρότυπα ανταλλαγής δεδομένων του IoT. Αν και το NDN προσφέρει πολλά οφέλη (π.χ. μείωση της καταναλισκόμενης ενέργειας, ελαφρύτερη στοίβα πρωτοκόλλου) [7], η προεπιλεγμένη

εφαρμογή του δε μπορεί να λειτουργήσει σε πολύ δυναμικά περιβάλλοντα IoT (συμπεριλαμβανομένων σεναρίων κινητικότητας, διακοπών, καθυστερήσεων, deep sleep κ.λπ.).

Το UMOBILE προσαρμόζει το μοντέλο επικοινωνίας έχοντας ως βάση την πληροφορία για να ικανοποιεί τις απαιτήσεις ευκαιριακών συνδέσεων, ενσωματώνοντας αυτές τις δύο προσεγγίσεις σε μία ενιαία αρχιτεκτονική. Οδηγώντας τις υπηρεσίες στην "άκρη" του δικτύου (edge computing), μια τέτοια αρχιτεκτονική μπορεί να λειτουργήσει ευρέως σε οποιοδήποτε περιβάλλον δικτύωσης και επιτρέπει την ανάπτυξη καινοτόμων εφαρμογών, παρέχοντας πρόσβαση σε δεδομένα ανεξάρτητα από τη διαθεσιμότητα σύνδεσης από άκρη σε άκρη (end-to-end) [8].

Ένας τρόπος για να εκμεταλλευτούμε καλύτερα το edge computing [9] είναι να αξιοποιήσουμε τα Linux Container ωστέ να φιλοξενούν τις υπηρεσίες και τις εφαρμογές κοντά στα "άκρα" του δικτύου (edge). Αποτελούν μία εναλλακτική της εικονικοποίησης (virtualization) που προσφέρει φορητότητα, υψηλή απόδοση και μείωση της χωρητικότητας. Το μέγεθος της εικόνας των container είναι αρκετά μικρότερο από το μέγεθος της εικόνας του Virtual Machine (VM). Μία αρκετά διαδεδομένη πλατφόρμα που η χρήση της ταυτίστηκε με τον όρο Container είναι το Docker.

Πρωταρχικός στόχος της παρούσας διπλωματικής εργασίας είναι η ανάπτυξη μιας εικόνας (image) η οποία θα καταλαμβάνει το ελάχιστο μέγεθος για τη λειτουργία των πρωτοκόλλων DTN και NDN-DTN. Βασιστήκαμε σε μία υπάρχουσα εικόνα με αρχικό μέγεθος 3.01 GB και εφαρμόσαμε μερικές τεχνικές στο Dockerfile για τη βελτιστοποίηση της, με κυριότερη τη μέθοδο multi-stage [10]. Έχοντας στη διάθεση μας την ελαχιστοποιημένη εικόνα ορίζουμε το δεύτερο στόχο που αφορά τη δημιουργία ενός εικονικού δικτύου με τη χρήση των Docker Containers. Το δίκτυο αυτό χρησιμοποιείται για την πραγματοποίηση πειραμάτων, τα οποία επιβεβαιώνουν τα οφέλη της προσέγγισης NDN-DTN για την ανάκτηση δεδομένων σε ένα περιβάλλον IoT που εμφανίζει διακοπτόμενες συνδέσεις. Τα αποτελέσματα συγκρίθηκαν με τα πειράματα που διεξήχθησαν σε πραγματικό δίκτυο IoT στο εργαστήριο, εφαρμόζοντας ένα σενάριο μεταφοράς δεδομένων από τον παραγωγό στον καταναλωτή μέσω μιας συσκευής μεταφοράς δεδομένων [5].

Στο Κεφάλαιο 2 παρουσιάζονται αναλυτικά οι αρχιτεκτονικές DTN, NDN και η UMOBILE που αξιοποιεί τις λειτουργίες και των δύων. Στο Κεφάλαιο 3 παρουσιάζεται η λειτουργία των Docker Container έναντι της εικονικοποίησης (virtualization) των Virtual Machine (VM), αναλύονται όλες οι δομές από τις οποίες αποτελείται το Docker. Στο Κεφάλαιο 4 παρουσιάζονται τα στάδια δημιουργίας της εικόνας που θα χρησιμοποιηθεί στο Κεφάλαιο 5, οι μέθοδοι ελαχιστοποίησης του μεγέθους της, καθώς και οι δύο τελικές εικόνες των πειραμάτων. Στο Κεφάλαιο 5 περιγράφεται αρχικά η τοπολογία του δικτύου που χρησιμοποιείται, η μεθοδολογία που ακολουθήθηκε για την εκτέλεση των πειραμάτων και τα αποτελέσματα αυτών. Στο κεφάλαιο 6 συνοψίζονται τα συμπεράσματα της παρούσας διπλωματικής εργασίας και αναφέρονται οι μελλοντικές επεκτάσεις της.

# Κεφάλαιο 2

# Αρχιτεκτονικές Μελλοντικού Διαδικτύου

Στο κεφάλαιο αυτό παρουσιάζονται τα πρωτόκολλα του μελλοντικού Διαδικτύου που εξετάστηκαν στην παρούσα εργασία. Αρχικά, το Delay Tolerant Networking (DTN) [11]είναι μια προσέγγιση στην αρχιτεκτονική δικτύου υπολογιστών, που επιδιώκει να αντιμετωπίσει τα τεχνικά ζητήματα σε ετερογενή δίκτυα, όπως είναι τα δίκτυα IoT. Αποτελεί μία αρχιτεκτονική τύπου overlay, δηλαδή λειτουργεί πάνω από υπάρχουσες στοίβες πρωτοκόλλων. Το σύνολο των μηνυμάτων της ονομάζονται bundle [12] και βασίζεται στη τεχνική αποθήκευσης και προώθησης αυτών (store-and-forward), προσφέροντας έτσι πρόσβαση ακόμα και σε απομακρυσμένες περιοχές. Επιπλέον, οι κόμβοι διαθέτουν ονόματα και όχι διευθύνσεις ώστε να γίνεται η παράδοση των bundle ακόμα και αν ο κόμβος προορισμού βρίσκεται εν κινήση.

Η αρχιτεκτονική Named Data Networking (NDN) [6] είναι μία προσέγγιση της αρχιτεκτονικής Information Centric Networking (ICN) [13]. Στόχος της είναι η επικοινωνία με βάση την ίδια την πληροφορία και όχι τη διεύθυνση IP. Η λειτουργία του δικτύου βασίζεται στην ανάκτηση δεδομένων που προσδιορίζονται από ένα καθορισμένο όνομα και στην αποθήκευση δεδομένων εντός του. Έτσι, ο συνδρομητής μπορεί να αιτηθεί ένα περιεχόμενο αποστέλλοντας ένα πακέτο Ενδιαφέροντος, το πακέτο αυτό προωθείται στο δίκτυο και μπορεί να ικανοποιηθεί, είτε από κάποιον παραγωγό είτε από κάποιο κόμβο που να έχει αποθηκευμένη την πληροφορία λόγω προηγούμενο αιτήματος, στέλνοντας ένα πακέτο Δεδομένων.

Η αρχιτεκτονική UMOBILE είναι μία προσέγγιση που αξιοποιεί τις λειτουργίες τόσο του NDN όσο και του DTN εισάγοντας βελτιώσεις αυτών. Έχει ως σκοπό την παράδοση πληροφοριών αλλά και υπηρεσιών ακόμα και σε περιοχές που είναι απομονωμένες και παρουσιάζουν διακοπτόμενες συνδέσεις.

## 2.1. Delay Tolerant Networking (DTN)

Το Delay Tolerant Networking (DTN) [11] είναι μια προσέγγιση στην αρχιτεκτονική δικτύου υπολογιστών, που επιδιώκει να αντιμετωπίσει τα τεχνικά ζητήματα σε ετερογενή δίκτυα, τα οποία ενδέχεται να μην έχουν συνεχή συνδεσιμότητα στο δίκτυο. Παραδείγματα τέτοιων δικτύων, είναι αυτά που λειτουργούν σε κινητά ή απομονωμένα χερσαία περιβάλλοντα, δίκτυα στο διάστημα ή δίκτυα αισθητήρων/ενεργοποιητών [14] όπως τα δίκτυα IoT. Τα δίκτυα αισθητήρων/ενεργοποιητών, που αφορούν τη παρούσα εργασία, χαρακτηρίζονται συχνά από περιορισμένη ισχύ, μνήμη και ικανότητα επεξεργαστή.

Επιπλέον, αναμένεται να ανήκουν σε δίκτυα μεγάλης κλίμακας, αποτελούμενα πιθανώς από χιλιάδες ή εκατομμύρια κόμβους.

Η αρχιτεκτονική DTN βασίζεται στην αφαιρετικότητα των μηνυμάτων μεταγωγής. Το σύνολο των μηνυμάτων είναι γνωστό ως “bundle”, ο όρος υιοθετήθηκε από [12]. Οι δρομολογητές στο DTN ονομάζονται “bundle forwarders” ή πύλες DTN.

Ως αρχιτεκτονική “overlay”, το DTN προορίζεται να λειτουργεί πάνω από τις υπάρχουσες στοίβες πρωτοκόλλων διαφόρων αρχιτεκτονικών και να παρέχει μια λειτουργία πυλών, αυτή της αποθήκευσης και προώθησης (store-and-forward), όταν ένας κόμβος συνδέεται φυσικά με δύο ή περισσότερα δίκτυα. Για παράδειγμα, στο Διαδίκτυο μπορεί να λειτουργήσει πάνω από το TCP/IP, για συνδέσεις με το διάστημα μπορεί να παρέχει μια υπηρεσία πύλης στο CFDP [15] και σε delay tolerant δίκτυα αισθητήρων/ενεργοποιητών μπορεί να παρέχει διασύνδεση με κάποιο πρωτόκολλο μεταφοράς αισθητήρων, που δεν είναι ακόμα τυποποιημένο.

Για κάθε ένα από αυτά τα δίκτυα έχουν αναπτυχθεί εξειδικευμένες στοίβες πρωτοκόλλων και ειδική σημασιολογία ονομάτων που αντικατοπτρίζουν τον τομέα εφαρμογής τους. Η επίτευξη της διαλειτουργικότητας μεταξύ τους επιτυγχάνεται με ειδικές πύλες DTN που βρίσκονται στα σημεία διασύνδεσης τους.

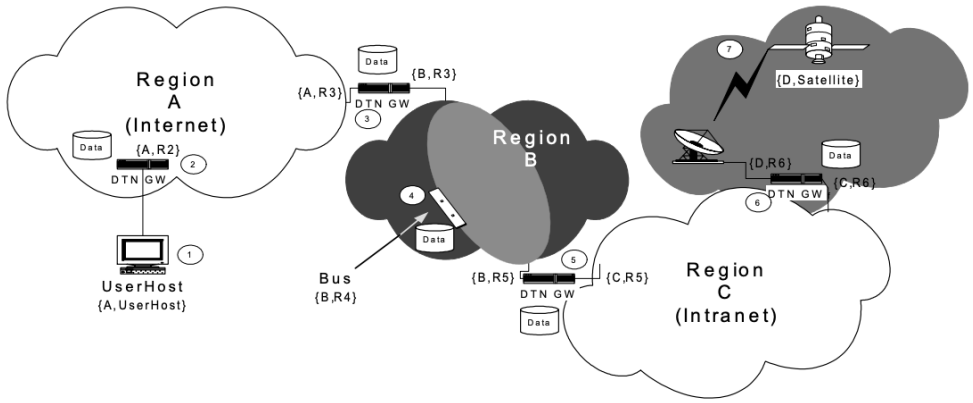
### 2.1.1. Περιοχές και πύλες DTN

Η αρχιτεκτονική DTN περιλαμβάνει τις έννοιες των περιοχών και των πυλών DTN, όπως αυτές αναπαρίστανται στο Σχήμα 2.1. Σε αυτό το παράδειγμα απεικονίζονται τέσσερις περιοχές (A, B, C, D). Η Περιοχή B περιλαμβάνει μια πύλη DTN, η οποία έχει τη μορφή λεωφορείου που μεταφέρει δεδομένα μεταξύ των πυλών DTN 3 και 5. Η Περιοχή D περιλαμβάνει ένα δορυφορικό σύνδεσμο low-earth- orbiting (LEO)[16] που παρέχει περιοδική σύνδεση με τη περιοχή C (αν και ίσως πιο τακτική από αυτή του λεωφορείου που μπορεί να υπόκεινται σε κυκλοφοριακή συμφόρηση ή άλλου είδους καθυστερήσεων).

Τα όρια των περιοχών χρησιμοποιούνται ως σημεία διασύνδεσης μεταξύ των ετερογενών δικτύων. Διαφορετικά θα μπορούσε κανείς να πει ότι δύο κόμβοι βρίσκονται στην ίδια περιοχή εάν μπορούν να επικοινωνήσουν μεταξύ τους χωρίς τη χρήση πυλών DTN, αλλά με τη χρήση των πρωτοκόλλων της περιοχής που ανήκουν. Οι πύλες DTN αντιστοιχούν τόσο στην έννοια "waypoint" του Metanet στο [17] όσο και στην έννοια των πυλών που περιγράφονται στην αρχική σχεδίαση ARPANET [18] [19]. Κάθε πύλη DTN η οποία κινείται μεταξύ δύο περιοχών αποτελείται από δύο "μισά", καθένα από τα οποία ανήκει σε μία από τις γειτονικές περιοχές .

Όταν απαιτείται αξιόπιστη επικοινωνία και αντιστοίχιση καθολικών ονομάτων σε τοπικά ονόματα για τη μεταφορά μηνυμάτων που προορίζονται για γειτονικές περιοχές, οι

πύλες είναι υπεύθυνες για την αποθήκευση μηνυμάτων σε μη πτητική μνήμη. Μπορούν επίσης να πραγματοποιήσουν έλεγχο ταυτότητας και πρόσβασης στην κυκλοφορία που εισέρχεται για να διασφαλίσουν ότι επιτρέπεται η προώθηση αυτής.




Σχήμα 2.1: Διασύνδεση ετερογενών περιοχών μέσω των πυλών DTN.

### 2.1.2. Tuple ονόματα

Για τη δρομολόγηση μηνυμάτων DTN, επιλέγεται να χρησιμοποιηθούν αναγνωριστικά για τα αντικείμενα ή τις ομάδες αντικειμένων που ονομάζονται tuple names και τα οποία περιλαμβάνουν δύο μεταβλητές. Στο Σχήμα 2.1 τα ονόματα των tuple για κάθε τερματική συσκευή (end-point) και για κάθε "μισό" του δρομολογητή παρουσιάζονται με τη μορφή {Region Name, Entity Name}. Η πρώτη μεταβλητή περιγράφει ένα ιεραρχικά δομημένο όνομα μιας περιοχής. Χρησιμοποιείται από τις πύλες DTN για το σχηματισμό πορείας προς μία ή περισσότερες πύλες DTN στην άκρη της καθορισμένης περιοχής. Τα ονόματα των περιοχών συμπληρώνονται στους πίνακες προώθησης είτε στατικά (από έναν διαχειριστή δικτύου), είτε από ένα ή περισσότερα δυναμικά πρωτόκολλα προώθησης του DTN .

Η ιεραρχική δομή των ονομάτων των περιοχών παρέχει τη δυνατότητα μείωσης του μεγέθους των πινάκων προώθησης του DTN, με τρόπο παρόμοιο με τη συγχώνευση διαδρομών του Διαδικτύου όπως στο CIDR [20]. Παρόλο που φαινομενικά υπάρχουν αρκετές ομοιότητες με τα ονόματα στο DNS [21], τα ονόματα περιοχών δεν χρειάζεται απαραίτητα να μεταφράζονται σε οποιοδήποτε μορφή διεύθυνσης ή να κατανέμονται ιεραρχικά όπως τα ονόματα DNS.

Η δεύτερη μεταβλητή προσδιορίζει ένα όνομα εντός της περιοχής που καθορίζει η πρώτη, το οποίο δε χρειάζεται να είναι μοναδικό εκτός αυτής της περιοχής. Όπως φαίνεται στο Σχήμα 2.1, μπορεί να είναι αυθαίρετης δομής και μπορεί να περιέχει ειδικά σύμβολα.

Στην περίπτωση του Διαδικτύου, για παράδειγμα, θα μπορούσαμε να έχουμε το ακόλουθο tuple:

{internet.icann.int, "http://www.ietf.org/oview.html"}

Το αναγνωριστικό αυτό αναφέρεται στην περιοχή του διαδικτύου και συγκεκριμένα σε ένα τοπικό αναγνωριστικό (στην περίπτωσή μας σε ένα URI [22]). Όταν ένα μήνυμα μεταφέρεται μέσα από περιοχές που παρουσιάζουν ετερογένεια, μόνο το πρώτο μέρος του αναγνωριστικού χρησιμοποιείται για τη δρομολόγηση του μηνύματος. Όταν φτάσει στα σύνορα της περιοχής τερματισμού, το όνομα της οντότητας μεταφράζεται σε τοπικό επίπεδο, σε ένα όνομα ή διεύθυνση κατάλληλη για την συγκεκριμένη περιοχή. Αυτή η μέθοδος επίλυσης ονομάτων οδηγεί σε μια μορφή *late binding* [23] για τα tuples, στην οποία χρησιμοποιείται μόνο η πρώτη μεταβλητή, το όνομα της περιοχής, του tuple από τις πύλες DTN.

Η επιλογή υιοθέτησης των ονομάτων έναντι των διευθύνσεων προκύπτει από την παρατήρηση των σύγχρονων τάσεων στη λειτουργία του Διαδικτύου. Οι διευθύνσεις χρησιμοποιούνται για τη δρομολόγηση και για την αναφορά σε έναν υπολογιστικό πόρο (π.χ end-point server) και προστίθεται η ονομασία για να διευκολύνει τη διευθυνσιοδότηση για τους ανθρώπους. Στις περισσότερες περιπτώσεις, ένα όνομα (με τη μορφή μιας διεύθυνσης URI ή μιας διεύθυνσης ιστού URL) αναφέρεται σε ένα αίτημα δεδομένων και όχι στην αναγνώριση ενός συγκεκριμένου υπολογιστικού συστήματος που τα παρέχει.

### 2.1.3. Υπηρεσίες τύπου ταχυδρομείου

Η έννοια ενός "challenged" δικτύου συνεπάγεται εγγενώς τον περιορισμό σε διάφορους πόρους. Επομένως, η κατανομή των πόρων με βάση την προτεραιότητα τους είναι σημαντικό να υιοθετηθεί στο τελικό μοντέλο. Η προσέγγιση που υιοθετεί το DTN αποτελεί υποσύνολο των υπηρεσιών που παρέχονται από την Ταχυδρομική Υπηρεσία των ΗΠΑ. Αυτό το σύστημα έχει αναπτυχθεί για να καλύψει τις ανάγκες εκατομμυρίων χρηστών που ανταλλάσσουν μηνύματα μη διαδραστικά και έχει το πρόσθετο πλεονέκτημα ότι είναι ήδη αρκετά γνωστό στους περισσότερους χρήστες.

Εκτός από τις βασικές κατηγορίες παράδοσης όπως για παράδειγμα η υπηρεσία πρώτης κατηγορίας και η υπηρεσία προτεραιότητας [11], μία μεγάλη ποικιλία ειδικών επιλογών παράδοσης είναι διαθέσιμη. Ορισμένοι συνδυασμοί αυτών των ειδικών επιλογών παράδοσης δεν υποστηρίζονται, ενώ άλλοι έχουν αμοιβαία αλληλεξάρτηση. Όμως οι υπηρεσίες ταχυδρομείου είναι αρκετά αποτελεσματικές λόγω της μακρόχρονης ιστορίας τους και της οικειότητας των χρηστών με αυτές. Επομένως, οι ακόλουθες βασικές ταχυδρομικές υπηρεσίες φαίνεται να είναι ελκυστικές: παράδοση χαμηλής, συνηθισμένης ή υψηλής προτεραιότητας, ειδοποιήσεις αποστολής, παράδοση στον παραλήπτη (απόδειξη επιστροφής) και καταγραφή δρομολόγησης (αρχείο παράδοσης). Το μοντέλο επεκτείνεται με τη πρόσθετη επιλογή της αξιόπιστης παράδοσης. Τα μηνύματα που απαιτούνται για

αυτήν την υπηρεσία αντιμετωπίζονται διαφορετικά από το σύστημα δρομολόγησης, καθώς απαιτούν μη πτητική αποθήκευση και custody transfer (αναφορά παρακάτω).

### 2.1.4. Αξιοπιστία και Custody transfer

Η αρχιτεκτονική DTN περιλαμβάνει δύο διαφορετικούς τύπους κόμβων δρομολογητών: πτητικοί (P) και μη-πτητικοί (NP). Οι κόμβοι P συμμετέχουν γενικά στο custody transfer χρησιμοποιώντας τα πρωτόκολλα μεταφοράς της περιοχής που ανήκουν, εκτός εάν δεν είναι σε θέση ή δεν θέλουν να αποθηκεύσουν ένα συγκεκριμένο μήνυμα. Το custody transfer αναφέρεται στη παράδοσης ενός μηνύματος, που έχει επιβεβαιωθεί, από ένα κόμβο DTN στον επόμενο και στη μεταφορά της ευθύνης για αξιόπιστη παράδοση στον επόμενο κόμβο.

Η έννοια του custody transfer είναι θεμελιώδης για την αρχιτεκτονική DTN με σκοπό να αντιμετωπίσει υψηλά ποσοστά απωλειών και να απαλλάξει ορισμένους κόμβους, που ενδέχεται να διαθέτουν λίγους πόρους, από τις ευθύνες που σχετίζονται με τη συνδεσιμότητα από άκρο σε άκρο. Ο μηχανισμός αυτός δεν είναι λιγότερο αξιόπιστος από τη χρήση του συνηθισμένου μηχανισμού αξιοπιστίας από άκρο σε άκρο, αν και είναι διαφορετικός. Το custody transfer μπορεί να θεωρηθεί ως βελτιστοποίηση της απόδοσης του μηχανισμού αξιοπιστίας από άκρο σε άκρο, που περιλαμβάνει την κίνηση τελικού σημείου.

### 2.1.5. Επιλογή διαδρομής και προγραμματισμός

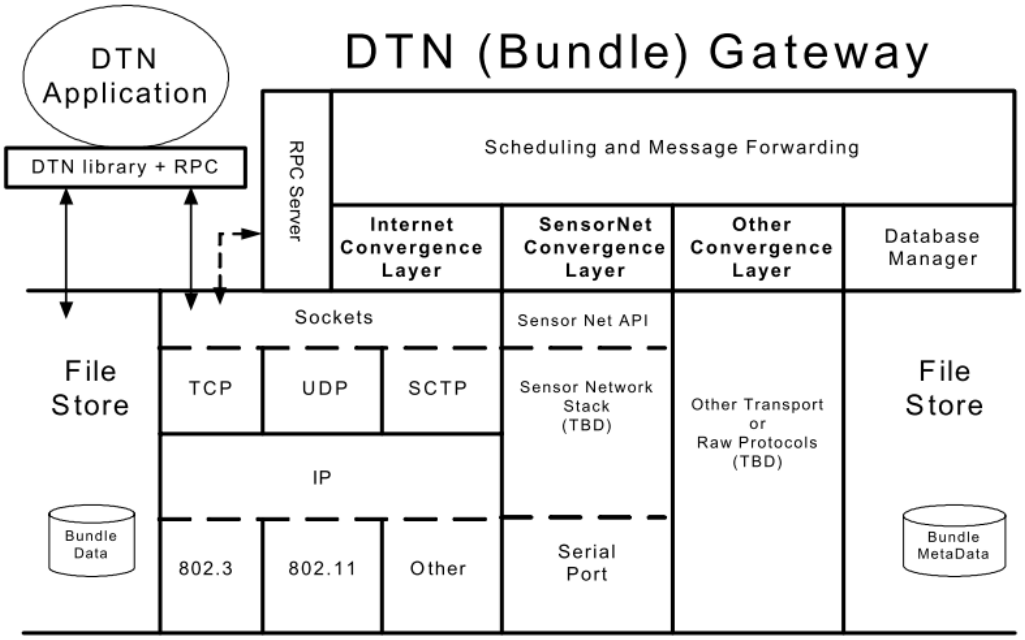
Η αρχιτεκτονική DTN στοχεύει σε δίκτυα όπου δεν μπορεί να θεωρηθεί ότι υπάρχει σύνδεση από άκρο σε άκρο. Οι διαδρομές αποτελούνται από ένα σύνολο επαφών (ευκαιρίες επικοινωνίας) που εξαρτώνται από το χρόνο και χρησιμοποιούνται για τη μεταφορά μηνυμάτων από το σημείο προέλευσή τους προς το σημείο προορισμού. Οι επαφές παραμετροποιούνται σε σχέση με το χρόνο έναρξης και λήξης αυτών, τη χωρητικότητα, την καθυστέρηση, τις τερματικές συσκευές και την κατεύθυνση. Επιπρόσθετα, ένα μέτρο της προβλεψιμότητας μιας επαφής μπορεί να βοηθήσει στην επιλογή των επόμενων κόμβων για τη μεταφορά των μηνυμάτων, καθώς και στην επιλογή του επόμενου μηνύματος που θα σταλεί.

Η προβλεψιμότητα μιας διαδρομής κυμαίνεται σε εύρος, από εντελώς προβλέψιμη (π.χ. ενσύρματη σύνδεση ή περιοδική σύνδεση της οποίας η φάση και η συχνότητα είναι γνωστά) έως εντελώς απρόβλεπτη (μια “opportunistic” επαφή στην οποία ένας δρομολογητής κινητής πλησιάζει έναν κόμβο DTN). Να ληφθεί υπόψη ότι το μέτρο της προβλεψιμότητας μιας επαφής επηρεάζεται από τη κατεύθυνση της. Για παράδειγμα, μια σύνδεση μέσω τηλεφώνου μπορεί να είναι εντελώς προβλέψιμη από την πλευρά του ατόμου που θα απαντήσει, ενώ είναι εντελώς απρόβλεπτη από την πλευρά του καλούντος.

Αυτές οι λεπτομέρειες για την επιλογή της διαδρομής και του προγραμματισμού των μηνυμάτων αναμένεται να επηρεαστούν σε μεγάλο βαθμό από τα πρωτόκολλα δρομολόγησης και τους αλγόριθμους της εκάστοτε περιοχής. Κατά την ανάπτυξη της DTN αρχιτεκτονικής, εντοπίστηκαν αρκετά προβλήματα που αποτέλεσαν πρόκληση για το σχεδιασμό της όπως:

- ο προσδιορισμός της ύπαρξης, αλλά και της προβλεψιμότητας των επαφών
- η απόκτηση πληροφοριών για την κατάσταση των μηνυμάτων που βρίσκονται σε εκκρεμότητα, δεδομένης και μιας μεγάλης καθυστέρησης
- η αποτελεσματική ανάθεση των μηνυμάτων στις επαφές και ο προσδιορισμός της σειράς μετάδοσης τους.

Στο [24] περιγράφεται μια γραμμική επίλυση των προβλημάτων δρομολόγησης/ προγραμματισμού.




Σχήμα 2.2: Δομή πύλης DTN.

### 2.1.6. Επίπεδα Convergence και Retransmission

Οποιοδήποτε πρωτόκολλο μεταφοράς μπορεί ή δε μπορεί να προσφέρει τις παρακάτω λειτουργίες: αξιόπιστη μεταφορά, συνδέσεις, έλεγχο ροής, έλεγχο συμφόρησης και όριο μηνυμάτων. Καθώς η λειτουργία προώθησης των bundle υποθέτει μια υποκείμενη αξιόπιστη ικανότητα παράδοσης με όρια μηνυμάτων κατά την εκτέλεση του custody transfer, τα πρωτόκολλα μεταφοράς που δε διαθέτουν αυτά τα χαρακτηριστικά πρέπει να

ενισχυθούν καταλλήλως. Το Σχήμα 2.2 απεικονίζει τη δομή υλοποίησης ενός δρομολογητή για τη προώθηση bundle (πύλη DTN), συμπεριλαμβανομένου ενός αριθμού ειδικών πρωτοκόλλων μεταφοράς επιπέδου convergence που χρησιμοποιούνται για να προσφέρουν αξιοπιστία, όρια μηνυμάτων και άλλα χαρακτηριστικά πάνω στα πρωτόκολλα μεταφοράς που απαιτούν ενίσχυση.

Σε περιπτώσεις όπου η αξιόπιστη παράδοση παρέχεται από μια υποκείμενη μεταφορά, το αντίστοιχο convergence επίπεδο χρειάζεται μόνο να διαχειριστεί την κατάσταση της σύνδεσης και να πραγματοποιήσει επανεκκίνηση σε περίπτωση απώλειας σύνδεσης. Σε περιπτώσεις χρήσης πρωτοκόλλων προσανατολισμένων στη σύνδεση, η ανίχνευση της απώλειας μιας σύνδεσης γίνεται αντιληπτή μέσω της διεπαφής της εφαρμογής. Σε περιπτώσεις όπου δεν παρέχεται άμεση υποστήριξη για την ανίχνευση σφαλμάτων, η ίδια η λειτουργία προώθησης των bundle μπορεί να ορίσει ένα χρονοδιακόπτη για να ξεκινήσει εκ νέου η μεταφορά μηνυμάτων εάν συμπεραίνεται ότι έχει αποτύχει η παράδοση τους. Αυτό έχει σχεδιαστεί ως εφεδρικό μέτρο σε περιπτώσεις υποκείμενης αποτυχίας σύνδεσης και δεν αναμένεται να είναι ένας αποτελεσματικός μηχανισμός για την έναρξη της αναμετάδοσης.

Η κατάλληλη επιλογή χρονικού ορίου για το μετρητή αναμετάδοσης ποικίλλει ανάλογα με την εκάστοτε περιοχή. Στα “challenged” δίκτυα, η γνώση ορισμένων ιδιοτήτων των διαδρομών στο επίπεδο προώθησης φαίνεται να είναι πολύ χρήσιμη για την επιλογή κατάλληλης πολιτικής ελέγχου σφαλμάτων.

### 2.1.7. Συγχρονισμός

Η αρχιτεκτονική DTN απαιτεί ένα αυστηρό βαθμό συγχρονισμού του χρόνου για τον εντοπισμό τμημάτων μηνύματος (fragments) (βλ. [14] ), καθώς και για την απομάκρυνση των μηνυμάτων που έχουν υπερβεί τη διάρκεια ζωής τους. Στις περισσότερες περιπτώσεις, ωστόσο, υπάρχουν πολλά επιπλέον οφέλη που προκύπτουν από την επιβολή ενός αυστηρότερου χρονικού περιορισμού στο συγχρονισμό (π.χ. της τάξεως των msec).

Παρατηρείται ότι ο συγχρονισμός του χρόνου απαιτείται από πολλές κατανεμημένες εφαρμογές, που χρησιμοποιούνται σε “challenged” περιβάλλοντα, και από την προσέγγιση του DTN για τον προγραμματισμό και την επιλογή διαδρομής (σε περιπτώσεις όπου η ώρα έναρξης και λήξης της επαφής είναι γνωστή εκ των προτέρων). Επιπλέον, δεδομένου του ακριβούς συγχρονισμού του χρόνου, οι τεχνικές διαχείρισης της συμφόρησης στο DTN μπορούν να προβλέψουν ποιες ώρες μπορεί να μειωθεί η συμφόρηση. Παρόλο που είναι πιο δύσκολο από τις απαιτήσεις συγχρονισμού του χρόνου στο Διαδίκτυο (που ουσιαστικά δεν υπάρχουν), πιστεύεται ότι το πρόβλημα του σωστού συγχρονισμού του χρόνου έχει αναπτυχθεί επαρκώς για να είναι μια προεπιλεγμένη πολιτική για τα περισσότερα δίκτυα.

### 2.1.8. Ασφάλεια

Το DTN επικεντρώνεται στην επαληθευμένη πρόσβαση στο φορτίο κίνησης μιας συγκεκριμένης κατηγορίας υπηρεσιών καθώς και στην αποφυγή μεταφοράς φορτίου κίνησης για ενδεχομένως μεγάλες αποστάσεις στις οποίες μπορεί να διαπιστωθεί αργότερα ότι απαγορεύεται.

Για την υλοποίηση του μοντέλου ασφάλειας, κάθε μήνυμα περιλαμβάνει μία αμετάβλητη «ταχυδρομική σφραγίδα» που περιέχει ένα εξακριβωμένο αναγνωριστικό του αποστολέα, μία έγκριση από την αιτούσα κατηγορία υπηρεσιών (Class of Service-CoS) που σχετίζεται με το μήνυμα, και ένα ακόμα κρυπτογραφικό υλικό για την επιβεβαίωση της ορθότητας του περιεχομένου του μηνύματος. Οι δρομολογητές ελέγχουν τα διαπιστευτήρια σε κάθε DTN κόμβο, και αν η επιβεβαίωση αποτύχει, απορρίπτουν την κίνηση όσο το δυνατόν γρηγορότερα.

Η συγκεκριμένη προσέγγιση χρησιμοποιεί κρυπτογραφία δημοσίου κλειδιού ως σημείο εκκίνησης. Στους δρομολογητές και χρήστες εκδίδονται ζεύγη δημοσίων/ιδιωτικών κλειδιών, και ο εντολέας που στέλνει ένα μήνυμα πρέπει να αποκτήσει ένα υπογεγραμμένο αντίγραφο των δημοσίων κλειδιών από μια αρχή ασφαλείας γνωστή στους DTN προωθητές. Στον πρώτο DTN δρομολογητή, το υπογεγραμμένο δημόσιο κλειδί χρησιμοποιείται για την επικύρωση του αποστολέα και την αιτούμενη κατηγορία υπηρεσιών (CoS) έναντι μιας λίστας πρόσβασης ελέγχου που είναι αποθηκευμένη στην πύλη. Τα αποδεκτά μηνύματα υπογράφονται εκ νέου από το κλειδί της πύλης για τη μεταφορά τους. Χρησιμοποιώντας αυτήν την προσέγγιση, μόνο οι first-hop πύλες θα πρέπει να φυλάσσουν στη μνήμη τους πιστοποιητικά ανά χρήστη και μετά μόνο για τους γειτονικούς χρήστες.

Αυτή η προσέγγιση θα βοηθήσει στη βελτίωση της διαχείρισης κλειδιών για αυτά τα δίκτυα, καθώς θα περιορίσει τον αριθμό των προσωρινά αποθηκευμένων πιστοποιητικών δημόσιου κλειδιού. Δεδομένου ότι οι πύλες DTN είναι πιθανό να αναπτυχθούν σε απομακρυσμένες περιοχές, ο περιορισμός του αριθμού και της συχνότητας ενημέρωσης των πιστοποιητικών θα πρέπει να παρέχει επιπλέον εξοικονόμηση ενέργειας.

### 2.1.9. Έλεγχος ροής και συμφόρησης

Όντας μια hop-by-hop αρχιτεκτονική, ο έλεγχος ροής και ο έλεγχος συμφόρησης για το DTN είναι στενά συνδεδεμένοι. Ο έλεγχος ροής αναφέρεται στον περιορισμό του ρυθμού αποστολής των μηνυμάτων από έναν DTN κόμβο στον επόμενο, ενώ ο έλεγχος συμφόρησης αναφέρεται στη διαχείριση των πόρων μιας πύλης DTN. Οι διαθέσιμοι μηχανισμοί για την αντιμετώπιση τέτοιων θεμάτων, μπορούν να κατηγοριοποιηθούν ως προληπτικοί(proactive) ή αντιδραστικοί (reactive).

Οι προληπτικοί μέθοδοι γενικά περιλαμβάνουν κάποια μορφή ελέγχου για την αποφυγή της συμφόρησης όταν αυτή βρίσκεται σε αρχικό στάδιο. Σε πολλές περιπτώσεις μια περιοχή μπορεί να βρίσκεται υπό το διαχειριστικό έλεγχο μιας οντότητας, τότε αυτή η προσέγγιση μπορεί να είναι αποτελεσματική. Εάν οι προληπτικοί μέθοδοι είναι ανεπαρκείς ή μη διαθέσιμοι, πρέπει να χρησιμοποιηθούν αντιδραστικοί μέθοδοι (πιθανότατα πραγματοποιούν άμεσα έλεγχο της ροής) που συνήθως οδηγούν στη μείωση της απόδοσης όταν οι καθυστερήσεις είναι πολύ μεγάλες.

Δύο πτυχές της αρχιτεκτονικής DTN καθιστούν το ζήτημα του ελέγχου συμφόρησης ιδιαίτερα δύσκολο:

- Εάν δεν πραγματοποιηθεί κάποια επαφή για ένα χρονικό διάστημα, τότε τα συσσωρευμένα δεδομένα δε θα μπορούν να απομακρυνθούν από τον εκάστοτε κόμβο για εκείνο το διάστημα.
- Τα ληφθέντα μηνύματα για τα οποία έχουν αναληφθεί ευθύνες (custody transfer) δεν μπορούν να απορριφθούν εκτός από ακραίες συνθήκες ή εάν λήξουν.

Λαμβάνοντας υπόψη αυτούς τους περιορισμούς, μπορούν να ληφθούν κάποια μέτρα για την αντιμετώπιση της συμφόρησης όπως:

- Δέσμευση χώρου αποθήκευσης ως συνάρτηση του CoS.
- Απόρριψη εισερχόμενων συνδέσεων για τη μεταφορά νέων μηνυμάτων όταν ο χώρος αποθήκευσης είναι πλήρης.
- Εξέταση μεταφοράς ευθυνών (custody transfer) σε άλλους πιθανούς κόμβους, που μπορεί να μην είναι οι καταλληλότερες επιλογές ως επόμενοι hop (hot potato routing [25])
- Απόρριψη μηνυμάτων για τα οποία δεν έχουν αναλάβει ευθύνες(custody transfer) ακόμα.
- Σε ασυνήθιστες και τρομερές περιστάσεις, ενδέχεται να υπάρχει δυνατότητα για την αφαίρεση μηνυμάτων που χρειάζονται φύλαξη, αλλά η απομάκρυνση τέτοιων πληροφοριών πρέπει να αποφεύγεται αν είναι δυνατόν, καθώς η διαγραφή αξιόπιστων πακέτων θα θεωρείται σφάλμα του συστήματος.

Η συγκεκριμένη προσέγγιση χρησιμοποιεί μια κοινόχρηστη ουρά προτεραιότητας για την κατανομή του αποθηκευτικού χώρου. Αρχικά, μηνύματα που έχουν λήξει χρονικά διαγράφονται. Στη συνέχεια, τα μηνύματα που καταφθάνουν και είναι πολύ μεγάλα απορρίπτονται. Μετά, τα μηνύματα κατηγοριοποιούνται με βάση την προτεραιότητα και το χρόνο ζωής (καθορίζεται από τον αποστολέα και μεταφέρεται σε κάθε μήνυμα).

Για την εφαρμογή του ελέγχου ροής, ένας δρομολογητής DTN θα προσπαθήσει να εκμεταλλευτεί οποιονδήποτε μηχανισμό ελέγχου ροής υπάρχει στα υποκείμενα πρωτόκολλα μεταφοράς κάθε περιοχής. Για τα περισσότερα δίκτυα, υπάρχουν ήδη μερικοί τέτοιοι μηχανισμοί (π.χ. TCP, X.25, RTS / CTS, XON / XOFF, κ.λπ.). Για άλλα δίκτυα, όμως, όπου εξακολουθούν να αναπτύσσονται τέτοιοι μηχανισμοί, μπορούν να κατασκευαστούν

τοπικά ειδικοί μηχανισμοί στα επίπεδα convergence των δρομολογητών DTN. Σε κάθε περίπτωση, οι κυριότερες λειτουργίες ενός DTN δρομολογητή υποθέτουν γενικά την ύπαρξη ελέγχου ροής, συνεπώς κάποιος τέτοιος μηχανισμός θα πρέπει να υπάρχει για να εξασφαλίσει αξιόπιστη παράδοση μηνυμάτων.

## 2.2. Information Centric Networking

Τα τελευταία χρόνια έχουν προκύψει νέες απαιτήσεις στη λειτουργία του Διαδικτύου, όπως η κλιμάκωση της διανομής περιεχομένου, η κινητικότητα, η εμπιστοσύνη, η ασφάλεια και ούτω καθεξής, στις οποίες δε μπορεί να ανταπεξέλθει με την υπάρχουσα δομή του. Για την κάλυψη των απαιτήσεων άρχισε να εμφανίζεται ένας φαύλος κύκλος ενημερώσεων του κώδικα λειτουργικότητας (patches), όπως το Mobile IP. Τα περισσότερα από αυτά τα patches αύξησαν την πολυπλοκότητα της συνολικής αρχιτεκτονικής και αποδείχθηκαν ότι αποτελούν προσωρινές λύσεις [26]. Αυτό εγείρει το ερώτημα εάν μπορεί να συνεχιστεί η «επιδιόρθωση μέσω επιδιορθώσεων» (patches-over-patches) ή εάν απαιτείται μια νέα αρχιτεκτονική προσέγγιση για το Διαδίκτυο [27].

Στο πλαίσιο αυτό, δημιουργήθηκε μια ερευνητική κοινότητα η οποία, έχοντας εντοπίσει τους περιορισμούς της τρέχουσας δομή του Διαδικτύου, συζητά τις βασικές απαιτήσεις και τους στόχους του Μελλοντικού Διαδικτύου, και προτείνει νέες αρχιτεκτονικές και λύσεις για την αντιμετώπισή τους [28]. Στις παρακάτω ενότητες μελετάτε μία από αυτές τις αρχιτεκτονικές, η λεγόμενη Named Data Networking (NDN).

Η προσέγγιση Information-Centric Networking (ICN) έχει αναδειχθεί ως μία πολλά υποσχόμενη αρχιτεκτονική του μελλοντικού Διαδικτύου. Εμπνευσμένη από το γεγονός ότι το Διαδίκτυο χρησιμοποιείται ολοένα και περισσότερο για τη διάδοση πληροφορίας, αντί για την επικοινωνία μεταξύ συσκευών, η ICN προσέγγιση αντικατοπτρίζει τις ανάγκες του Διαδικτύου καλύτερα από την υπάρχουσα αρχιτεκτονική του. Δίνοντας όνομα στην πληροφορία στο επίπεδο δικτύου, η ICN ευνοεί την ανάπτυξη μηχανισμών προσωρινής αποθήκευσης εντός δικτύου (ή αποθήκευσης γενικότερα) και μηχανισμών πολλαπλής διανομής, διευκολύνοντας έτσι την αποτελεσματική και έγκαιρη παράδοση πληροφοριών στους χρήστες. Επιπρόσθετα, η ICN εκτός από τη διανομή πληροφοριών καλύπτει όλο το φάσμα των μελλοντικών απαιτήσεων και στόχων του Διαδικτύου, όπως τη διαχείριση της κινητικότητας και της ασφάλειας.

Οι διάφορες αρχιτεκτονικές της ICN αντιμετωπίζουν ένα σύνολο βασικών λειτουργιών με διαφορετικές προσεγγίσεις η κάθε μια. Παρακάτω παρατίθενται αυτές οι λειτουργίες:

- **Ονοματοδοσία** : Η δομή του ονόματος είναι ένα από τα κύρια χαρακτηριστικά κάθε αρχιτεκτονικής προσέγγισης ICN. Σε όλες τις αρχιτεκτονικές ICN τα ονόματα που αποδίδονται στην πληροφορία είναι ανεξάρτητα της τοποθεσίας. Ανάλογα την προσέγγιση τα ονόματα μπορεί να είναι δομημένα ιεραρχικά και χωρισμένα σε τμήματα (hierarchical) ή δομημένα γραμμικά και ενιαία (flat) και μπορούν ή δε μπορούν να είναι αναγνώσιμα από τους ανθρώπους.

- **Ανάλυση ονόματος και δρομολόγηση δεδομένων** : Η ανάλυση του ονόματος της πληροφορίας (name resolution) ουσιαστικά αντιστοιχεί ένα όνομα σε έναν πάροχο ή μία πηγή που διαθέτει αυτή την πληροφορία. Η δρομολόγηση των δεδομένων περιλαμβάνει το σχηματισμό μιας διαδρομής κατά την οποία θα μεταφερθεί αυτή η πληροφορία από τον πάροχο στον αιτούντα. Ένα βασικό ζήτημα είναι εάν αυτές οι δύο λειτουργίες είναι σχετικές μεταξύ τους και συνδέονται (coupled approach), ή είναι ανεξάρτητες χωρίς καμία συσχέτιση (decoupled approach). Στη περίπτωση της προσέγγισης coupled το αίτημα της πληροφορίας κατευθύνεται σε έναν πάροχο, ο οποίος στη συνέχεια στέλνει την πληροφορία ακολουθώντας την ίδια διαδρομή με πριν. Ενώ στην προσέγγιση decoupled η συνάρτηση του name resolution δε καθορίζει ούτε περιορίζει τη διαδρομή που θα ακολουθήσουν τα δεδομένα από τον πάροχο στον συνδρομητή.

- **Αποθήκευση** : Διακρίνεται σε δύο κατηγορίες στην αποθήκευση εντός διαδρομής (on-path) και εκτός (off-path). Στην περίπτωση της αποθήκευσης εντός διαδρομής το δίκτυο εκμεταλλεύεται τις πληροφορίες που είναι αποθηκευμένες στη διαδρομή που έχει καθοριστεί από τη συνάρτηση name resolution ενός αιτήματος, ενώ στην περίπτωση της αποθήκευσης εκτός διαδρομής το δίκτυο εκμεταλλεύεται τις πληροφορίες που είναι αποθηκευμένες εκτός της διαδρομής που σχηματίζεται.

- **Κινητικότητα** : Η κινητικότητα των συνδρομητών υποστηρίζεται εγγενώς, καθώς οι συνδρομητές κινητής μπορούν να στέλνουν εκ νέου αιτήματα πληροφορίας μετά από κάθε παράδοση. Η κινητικότητα του παρόχου είναι πιο δύσκολο να υποστηριχθεί, καθώς το σύστημα name resolution και οι πίνακες δρομολόγησης στη decoupled προσέγγιση χρειάζονται ενημέρωση συνεχώς.

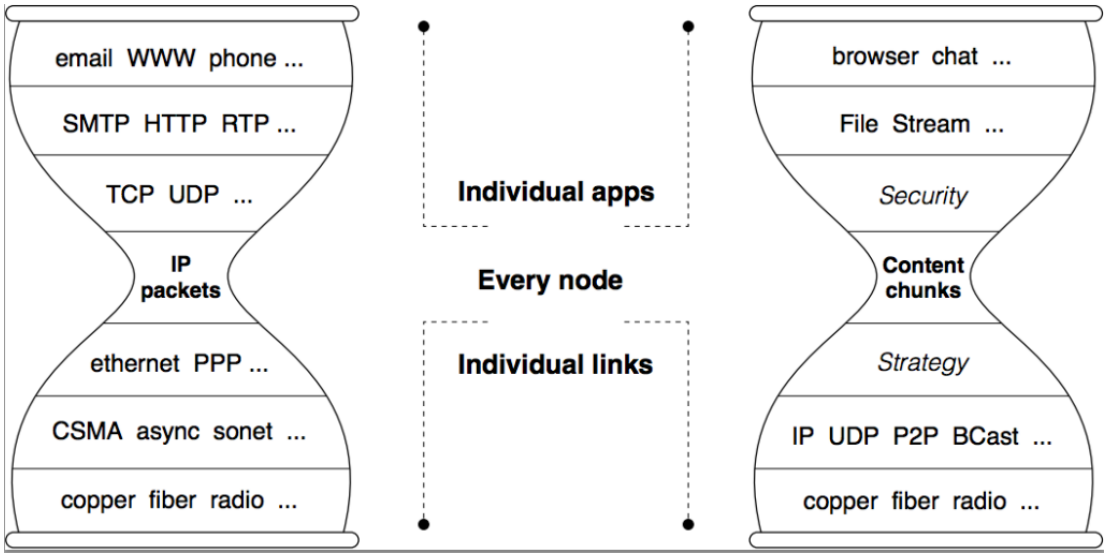
- **Ασφάλεια** : σχετίζεται ιδιαίτερα με τη δομή των ονομάτων [29]. Από τη μία πλευρά, τα αναγνώσιμα από τον άνθρωπο ονόματα απαιτούν έναν αξιόπιστο αντζέντη ή μια σχέση εμπιστοσύνης με το σύστημα name resolution για να εξακριβωθεί αν οι πληροφορίες που επιστρέφονται αντιστοιχούν στο όνομα που ζητήθηκε. Από την άλλη πλευρά, τα flat ονόματα υποστηρίζουν εγγενώς την αξιοπιστία τους, αλλά δεν είναι αναγνώσιμα από τον άνθρωπο, απαιτώντας έτσι ένα άλλο αξιόπιστο σύστημα για την αντιστοίχιση flat ονομάτων σε ονόματα αναγνώσιμα από τον άνθρωπο.

## 2.3. Προσέγγιση Named Data Networking

Η σημερινή αρχιτεκτονική του Διαδικτύου με τη μορφή της κλεψύδρας επικεντρώνεται σε ένα καθολικό επίπεδο δικτύου (π.χ δίκτυο IP διευθύνσεων) που εφαρμόζει την ελάχιστη λειτουργικότητα που απαιτείται για την παγκόσμια διασύνδεση. Αυτή η “thin-waist” προσέγγιση ευνόησε τη ραγδαία ανάπτυξη του Διαδικτύου, επιτρέποντας τόσο τις τεχνολογίες

των κατώτερων όσο και των ανώτερων επιπέδων να καινοτομήσουν ανεξάρτητα η μία από την άλλη. Ωστόσο, η αρχιτεκτονική IP είχε σχεδιαστεί για να δημιουργήσει ένα δίκτυο επικοινωνίας, όπου τα πακέτα αναφερόντουσαν μόνο στα τελικά σημεία επικοινωνίας (endpoint devices).

Το έργο Named Data Networking [6] διερευνά την εξέλιξη της σημερινής αρχιτεκτονικής του Ίντερνετ, με βάση τη χρήση IP διευθύνσεων( host-centric), σε μία αρχιτεκτονική δικτύου με βάση το ίδιο το περιεχόμενο. Το NDN έχει τις ρίζες του από ένα προηγούμενο έργο, το Content-Centric Networking(CCN) [30], για το οποίο μίλησε πρώτη φορά δημόσια ο Van Jacobson σε μία παρουσίαση του το 2006 [31]. Το πρωτόκολλο NDN είναι ένα από τα πέντε προγράμματα που χρηματοδοτούνται από το εθνικό επιστημονικό ίδρυμα των Ηνωμένων Πολιτειών Αμερικής για το πρόγραμμα «Μελλοντικές Αρχιτεκτονικές του Διαδικτύου».




Σχήμα 2.3: Στα αριστερά απεικονίζεται η δομή της αρχιτεκτονικής του IP ,ενώ στα δεξιά η δομή της αρχιτεκτονικής NDN.

Το NDN αλλάζει τη σημασιολογία της υπηρεσίας του δικτύου από την παράδοση του πακέτου σε μια δεδομένη διεύθυνση προορισμού σε ανάκτηση των δεδομένων τα οποία προσδιορίζονται από ένα δεδομένο όνομα. Αυτή η αλλαγή επιτρέπει στα δίκτυα NDN αρχιτεκτονικής να αξιοποιούν σχεδόν όλες τις ιδιότητες του Ίντερνετ. Έτσι βρίσκεται σε θέση να επιλύει ένα μεγαλύτερο φάσμα προβλημάτων συμπεριλαμβανομένων όχι μόνο προβλημάτων επικοινωνίας από άκρη-σε-άκρη αλλά και διανομής περιεχομένου και ελέγχου.

Η σχεδίαση της αρχιτεκτονικής ενσωματώνει γνωστούς μηχανισμούς ασφαλείας, μέσω της χρήσης των υπογραφών σε όλα τα ονομαζόμενα δεδομένα, αλλά και μηχανισμούς αυτορρύθμισης της κίνησης του δικτύου, μέσω της αντιστάθμισης της ροής μεταξύ των πακέτων Ενδιαφέροντος (Interest packet) και των πακέτων Δεδομένων (Data packet). Το όνομα ενός NDN πακέτου μπορεί να αναφέρεται σε οτιδήποτε, όπως σε ένα τελικό σημείο, ένα τμήμα δεδομένων μιας ταινίας ή ενός βιβλίου, μία εντολή για την ενεργοποίηση φώτων, κ.α. Η αρχιτεκτονική NDN διαθέτει λειτουργίες που είναι σχεδιασμένες ώστε να συμβάλλουν στην επιλογή των χρηστών και στον ανταγωνισμό καθώς αναπτύσσεται το δίκτυο, όπως προώθηση

δεδομένων προς πολλά μονοπάτια (multipath forwarding) και αποθήκευση δεδομένων μέσα στο ίδιο το δίκτυο (in-network caching).

Για την ανάκτηση αντικειμένων μεγάλου περιεχομένου, που μπορεί να χωρίζονται σε πολλαπλά πακέτα, τα πακέτα Ενδιαφέροντος παρέχουν παρόμοιο ρόλο στον έλεγχο της ροής της κυκλοφορίας όπως και τα ACK στο πρωτόκολλο TCP στη σημερινή λειτουργία του Διαδικτύου: ένα βρόχο ανατροφοδότησης που ελέγχεται από το συνδρομητή και περιγράφεται στην επόμενη ενότητα. Ούτε τα πακέτα Ενδιαφέροντος ούτε τα πακέτα Δεδομένων φέρουν διευθύνσεις κεντρικού υπολογιστή ή διεπαφών. Οι δρομολογητές προωθούν πακέτα Ενδιαφέροντος προς τους παραγωγούς με βάση τα ονόματα που μεταφέρουν τα πακέτα και προωθούν τα πακέτα Δεδομένων στους καταναλωτές με βάση τις πληροφορίες που έχουν καταχωρηθεί στον πίνακα PIT. Αυτή η συμμετρία ανταλλαγής πακέτων Ενδιαφέροντος/Δεδομένων (Interest/Data packet) προκαλεί έναν βρόχο ελέγχου hop-by-hop. Έτσι εξαλείφει την ανάγκη της έννοιας των κόμβων προέλευσης ή προορισμού στην παράδοση δεδομένων, σε αντίθεση με το μοντέλο παράδοσης πακέτων από άκρο σε άκρο της IP αρχιτεκτονικής.

### 2.3.1. Ονοματοδοσία

Τα ονόματα στην αρχιτεκτονική NDN είναι ιεραρχικά και μπορεί να μοιάζουν με URLs [32],για παράδειγμα ένα όνομα μπορεί να είναι /aueb.gr/ai/main.html. Ωστόσο τα ονόματα δεν είναι απαραίτητο να έχουν τη μορφή URL: το πρώτο τμήμα τους δεν είναι ένα όνομα DNS [21] ή μια διεύθυνση IP και δε χρειάζεται να είναι αναγνώσιμα από τον άνθρωπο. Αντ΄ αυτού στο NDN κάθε στοιχείο του ονόματος μπορεί να είναι οτιδήποτε, συμπεριλαμβανομένων διάστικτων συμβολοσειρών (dotted string) αναγνώσιμων από τον άνθρωπο ή “αποτύπωμα” μίας τιμής (hash [33]) .

Στο NDN ένα αίτημα για μία ονομαζόμενη πληροφορία θεωρείται ότι αντιστοιχεί σε οποιαδήποτε πληροφορία της οποίας το όνομα έχει το ζητούμενο όνομα ως πρόθεμα. Για παράδειγμα, το /aueb.gr/ai/main.html μπορεί να αντιστοιχιθεί με ένα αντικείμενο πληροφοριών που ονομάζεται /aueb.gr/ai/main.html/_v1/_s1, που θα μπορούσε να ερμηνευθεί ως το πρώτο τμήμα της πρώτης έκδοσης των ζητουμένων δεδομένων. Μετά τη λήψη αυτού του αντικειμένου πληροφοριών, ο συνδρομητής θα μπορούσε να ζητήσει το επόμενο τμήμα δεδομένων, είτε απευθείας ζητώντας το URL /aueb.gr/ai/main.html/_v1/_s2, ή απλά ζητώντας το αμέσως επόμενο τμήμα υπό αυτήν την έκδοση.

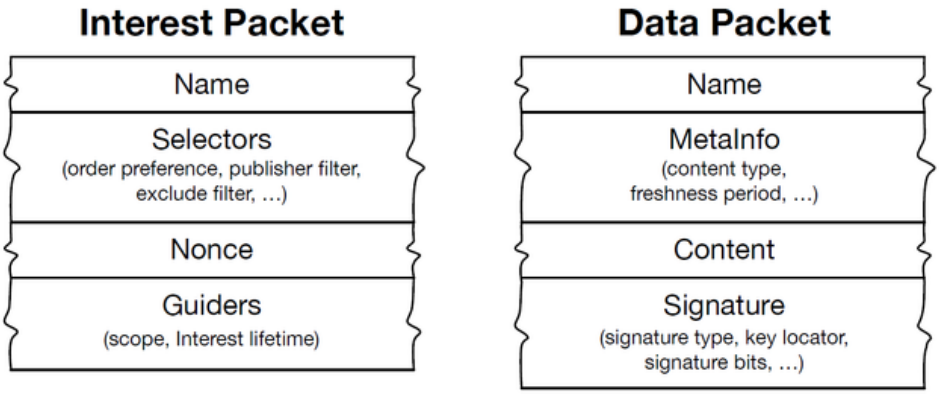
Ενώ ο τρόπος με τον οποίο τμηματοποιούνται τα αντικείμενα πληροφοριών αναμένεται να είναι γνωστός από την εφαρμογή του συνδρομητή, ο κανόνας αντιστοίχισης του προθέματος επιτρέπει σε μια εφαρμογή να ανακαλύψει τι είναι διαθέσιμο. Επιπλέον, επιτρέπει στον συνδρομητή να ζητήσει δεδομένα τα οποία δεν έχουν παραχθεί ακόμα: ένας πάροχος μπορεί να κοινοποιήσει ότι μπορεί να ικανοποιήσει αιτήματα για συγκεκριμένα προθέματα ονομάτων, και στη συνέχεια να επιστρέψει τα αντικείμενα πληροφορίας με τα πλήρη ονόματα NDN. Αυτό μπορεί να χρησιμοποιηθεί για την εφαρμογή διαφόρων εφαρμογών όπου τα αντικείμενα πληροφοριών δημιουργούνται δυναμικά, επομένως τα πλήρη ονόματά τους δεν είναι γνωστά εκ των προτέρων, όπως σε μία φωνητική διάσκεψη [34].

### 2.3.2. Ανάλυση ονόματος και δρομολόγηση δεδομένων

Η επικοινωνία στο NDN είναι καθοδηγούμενη από τους δέκτες, π.χ καταναλωτές δεδομένων (data consumers), μέσω της ανταλλαγής δύων τύπων πακέτων:

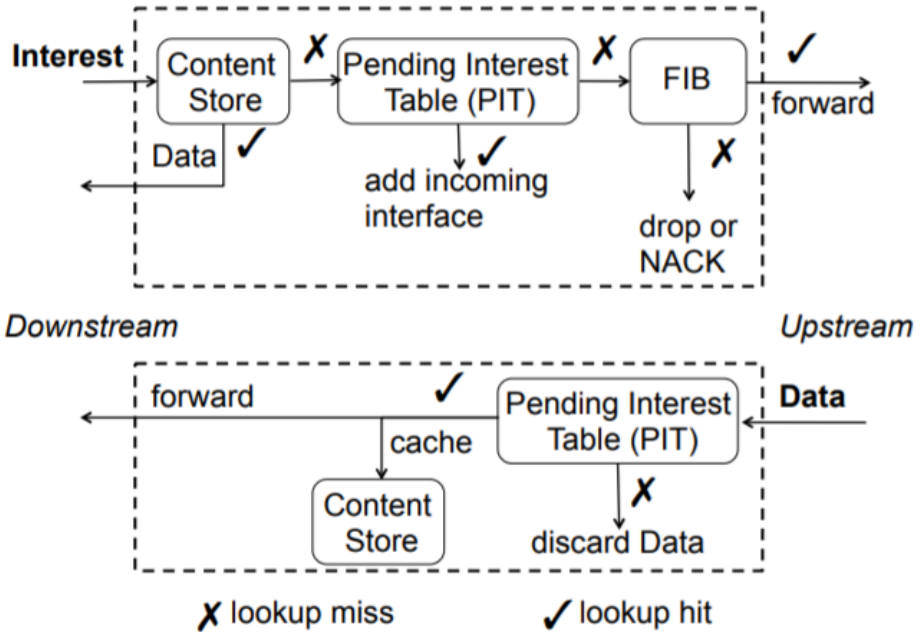
- Πακέτων Ενδιαφέροντος ( Interest packet)
- Πακέτων Δεδομένων (Data packet)

Και οι δύο τύποι πακέτων φέρουν ένα όνομα που προσδιορίζει ένα κομμάτι δεδομένων, το οποίο μεταδίδεται στο πακέτο Δεδομένων (Data Packet). Ένας συνδρομητής βάζει το όνομα ενός επιθυμητού τμήματος δεδομένων σε ένα πακέτο Ενδιαφέροντος (Interest Packet) και το στέλνει στο δίκτυο. Οι δρομολογητές χρησιμοποιούν αυτό το όνομα για να προωθήσουν το πακέτο στους παραγωγούς δεδομένων (producers). Μόλις το πακέτο Ενδιαφέροντος (Interest Packet) φτάσει σε έναν κόμβο που διαθέτει τα ζητούμενα δεδομένα, ο κόμβος θα επιστρέψει ένα πακέτο Δεδομένων (Data Packet) που περιέχει τόσο το όνομα όσο και το περιεχόμενο που ζητήθηκε, μαζί με μια υπογραφή από το κλειδί του παραγωγού που τα δεσμεύει (Σχήμα 2.4). Αυτό το πακέτο Δεδομένων (Data Packet) ακολουθεί αντίστροφα τη διαδρομή που ακολούθησε το πακέτο Ενδιαφέροντος (Interest Packet) για να επιστρέψει στον αιτούντα συνδρομητή.




Σχήμα 2.4: Δομή των πακέτων στο πρωτόκολλο NDN.

Για να πραγματοποιήσει τις λειτουργίες προώθησης πακέτων Ενδιαφέροντος (Interest Packet) και Δεδομένων (Data Packet), κάθε δρομολογητής (Content Router-CR) διατηρεί τρεις δομές δεδομένων: ένα Pending Interest Table (PIT), ένα Forwarding Information Base (FIB), και ένα Content Store (CS)(Σχήμα 2.4), καθώς και μία στρατηγική προώθησης (δεν φαίνεται στο σχήμα) που καθορίζει εάν, πότε και πού θα προωθήσει το κάθε πακέτο Ενδιαφέροντος.




Σχήμα 2.5: Διαδικασία προώθησης σε ένα κόμβο NDN.

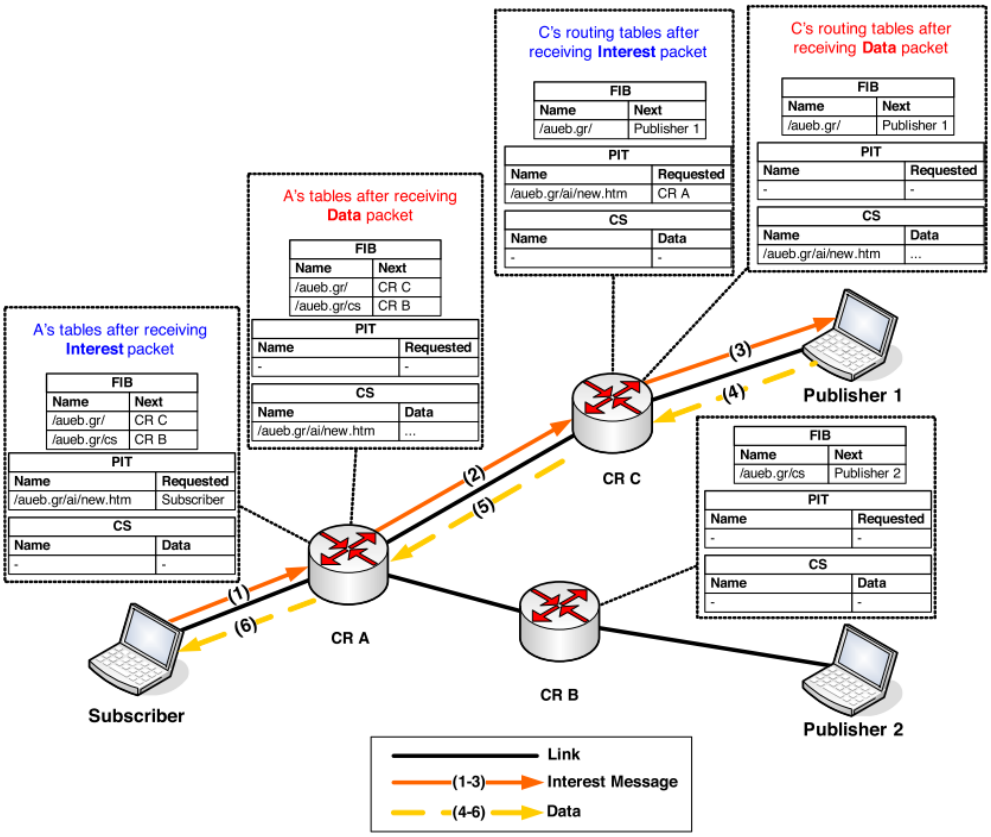
Το PIT αποθηκεύει όλα τα πακέτα Ενδιαφέροντος (Interest Packet) που έχει προωθήσει ένας δρομολογητής (Content Router) αλλά δεν έχουν ακόμα ικανοποιηθεί. Κάθε καταχώρηση του PIT καταγράφει τα ονόματα των δεδομένων που μεταφέρονται στο Διαδίκτυο, μαζί με τις διεπαφές από τις οποίες εισήλθαν στο δρομολογητή. Όταν φτάνει ένα πακέτο Ενδιαφέροντος (Interest Packet) σε έναν δρομολογητή CR αυτός ελέγχει πρώτα αν υπάρχουν αποθηκευμένα τα δεδομένα στο Content Store. Εάν βρεθεί το πακέτο που ζητείται στο CS ο δρομολογητής επιστρέφει το πακέτο Δεδομένων (Data Packet) από τη διεπαφή που εισήλθε το πακέτο Ενδιαφέροντος (Interest Packet) εξ αρχής. Διαφορετικά, ο δρομολογητής αναζητά το όνομα στο PIT του και αν υπάρχει αντίστοιχη καταχώρηση, καταγράφει απλώς την εισερχόμενη διεπαφή αυτού του Ενδιαφέροντος (Interest Packet) στην εκάστοτε καταχώρηση PIT. Εάν δεν υπάρχει αντίστοιχη καταχώρηση PIT, ο δρομολογητής θα προωθήσει το Ενδιαφέρον προς τους εκδότες (publisher) με βάση τις πληροφορίες που δίνονται στο πίνακα FIB και τη στρατηγική προώθησης του δρομολογητή (θεωρείται ότι αυτή προσαρμόζεται όποτε χρειάζεται).

Όταν ένας δρομολογητής λαμβάνει πακέτα Ενδιαφέροντος για το ίδιο όνομα από πολλούς κόμβους, προωθεί μόνο το πρώτο εξ αυτών προς τον εκδότη. Σε κάθε περίπτωση το FIB συμπληρώνεται από ένα πρωτόκολλο δρομολόγησης βάση του προθέματος του ονόματος και μπορεί να έχει πολλαπλές διεπαφές εξόδου για κάθε πρόθεμα.

Η στρατηγική προώθησης μπορεί να αποφασίσει να απορρίψει ένα πακέτο Ενδιαφέροντος σε συγκεκριμένες περιπτώσεις, π.χ. εάν όλοι οι σύνδεσμοι από το συνδρομητή προς τον εκδότη (upstream) βρίσκονται σε κατάσταση συμφόρησης ή υπάρχει υποψία ότι το πακέτο αυτό αποτελεί μέρος μιας επίθεσης DoS [35]. Για κάθε πακέτο Ενδιαφέροντος (Interest Packet), η στρατηγική προώθησης ανακτά την καταχώρηση με το μεγαλύτερο ποσοστό αντιστοιχίας του προθέματος από τον πίνακα FIB και αποφασίζει πότε και πού θα προωθήσει το πακέτο. Το Content Store είναι μια προσωρινή μνήμη των πακέτων Δεδομένων που έχει λάβει ο

δρομολογητής. Επειδή ένα πακέτο Δεδομένων (Data Packet) NDN δεν εξαρτάται από το πού προέρχεται ή προς τα που προωθείται, μπορεί να αποθηκευτεί στην κρυφή μνήμη για να ικανοποιήσει κάποιο μελλοντικό πακέτο Ενδιαφέροντος.

Όταν φτάνει ένα πακέτο Δεδομένων, ένας δρομολογητής NDN βρίσκει την αντίστοιχη καταχώρηση PIT και προωθεί τα δεδομένα σε όλες τις διεπαφές που βρίσκονται από την πλευρά κίνησης των δεδομένων από τον παραγωγό στον συνδρομητή(downstream) και αναφέρονται σε αυτήν την καταχώρηση PIT. Στη συνέχεια, διαγράφει αυτήν την καταχώρηση PIT και αποθηκεύει προσωρινά τα δεδομένα στο Content Store. Η ανάλυση ονόματος και η δρομολόγηση δεδομένων συνδέονται (coupled approach). Αυτό σημαίνει ότι τα πακέτα Δεδομένων (Data Packet) ακολουθούν πάντα την αντίστροφη πορεία των πακέτων Ενδιαφέροντος (Interest Packet) και αν δεν υπάρξει απώλεια πακέτου, τότε ένα πακέτο Ενδιαφέροντος οδηγεί σε ένα πακέτο Δεδομένων παρέχοντας ισορροπία ροής (Σχήμα 2.6).




Σχήμα 2.6: Ο συνδρομητής (subscriber) στέλνει ένα πακέτο Ενδιαφέροντος (Interest Packet) για το όνομα /aueb.gr/ai/new.htm (βέλη 1-3). Και στη συνέχεια ο εκδότης απαντάει με ένα πακέτο Δεδομένων (Data Packet)(βέλη 4-6).

Για την ανάκτηση αντικειμένων μεγάλου περιεχομένου, που μπορεί να χωρίζονται σε πολλαπλά πακέτα, τα πακέτα Ενδιαφέροντος παρέχουν παρόμοιο ρόλο στον έλεγχο της ροής της κυκλοφορίας όπως και τα ACK στο πρωτόκολλο TCP στη σημερινή λειτουργία του Διαδικτύου: ένα βρόχο ανατροφοδότησης που ελέγχεται από το συνδρομητή και περιγράφεται στην επόμενη ενότητα. Ούτε τα πακέτα Ενδιαφέροντος (Interest Packet) ούτε τα πακέτα Δεδομένων (Data Packet) φέρουν διευθύνσεις κεντρικού υπολογιστή ή διεπαφών. Οι δρομολογητές προωθούν πακέτα Ενδιαφέροντος (Interest Packet) προς τους παραγωγούς με βάση τα ονόματα που μεταφέρουν τα πακέτα και προωθούν τα πακέτα Δεδομένων (Data Packet) στους καταναλωτές με βάση τις πληροφορίες που έχουν καταχωρηθεί στον πίνακα PIT.

Αυτή η συμμετρία ανταλλαγής πακέτων Ενδιαφέροντος/Δεδομένων (Interest/Data Packet) προκαλεί έναν βρόχο ελέγχου hop-by-hop. Έτσι εξαλείφει την ανάγκη της έννοιας των κόμβων προέλευσης ή προορισμού στην παράδοση δεδομένων, σε αντίθεση με το μοντέλο παράδοσης πακέτων από άκρο σε άκρο της IP αρχιτεκτονικής.

### 2.3.3. Caching

Το NDN υποστηρίζει εγγενώς την προσωρινή αποθήκευση εντός διαδρομής (on-path). Κάθε δρομολογητής συμβουλεύεται πρώτα το CS όταν λαμβάνει ένα μήνυμα Interest και αποθηκεύει προσωρινά όλα τα αντικείμενα πληροφορίας που μεταφέρονται από τα πακέτα Δεδομένων (Data Packet). Το CS μπορεί να χρησιμοποιήσει LRU ή οποιαδήποτε άλλη πολιτική αντικατάστασης, αλλά, ρεαλιστικά, δε μπορεί να χρησιμοποιηθεί για μακροπρόθεσμη αποθήκευση αποθηκεύοντας όλα τα πακέτα που φτάνουν σε αυτό [36], [37]. Επομένως, είναι κυρίως χρήσιμο για ανάκτηση πληροφοριών σε περιπτώσεις απώλειας πακέτων και για το χειρισμό ενός κύματος χρηστών, όπου πολλοί χρήστες ζητούν τα ίδια δεδομένα διαδοχικά.

Η προσωρινή αποθήκευση εκτός διαδρομής (off-path) υποστηρίζεται αποστέλλοντας ένα πακέτο Ενδιαφέροντος (Interest Packet) σε οποιαδήποτε κόμβο μπορεί να φιλοξενεί το ζητούμενο αντικείμενο πληροφορίας. Για παράδειγμα, το επίπεδο στρατηγικής μπορεί να κατευθύνει το πακέτο Ενδιαφέροντος (Interest Packet) σε κάποιο δρομολογητή NDN που διαθέτει τη πληροφορία, παρά στον αρχικό εκδότη αυτής. Αυτό όμως δεν είναι προφανές για το NDN, τα ονόματα αυτών των αντιγράφων θα πρέπει να κοινοποιούνται από τους δρομολογητές NDN μέσω του πρωτοκόλλου δρομολόγησης που χρησιμοποιείται, με σκοπό να συμπληρωθούν σωστά οι πίνακες FIB με τους δείκτες που θα δείχνουν σε αυτά τα αντίγραφα.

### 2.3.4. Κινητικότητα

Όταν ένας συνδρομητής μετακινείται, μπορεί απλά να εκδώσει νέα πακέτα Ενδιαφέροντος (Interest Packet) από την τρέχουσα θέση του για τα αντικείμενα πληροφοριών που δεν έχει λάβει ακόμη. Ο PIT θα αποθηκεύσει τα αιτήματα στο πρώτο κοινό δρομολογητή. Φυσικά, τα αντίστοιχα πακέτα Δεδομένων (Data Packet) θα παραδοθούν επίσης στην παλιά τους θέση.

Από την άλλη πλευρά, όταν ένας εκδότης κινείται οι FIB που έχουν μία καταχώρηση γι αυτόν πρέπει να ενημερωθούν ξανά, κάτι που απαιτεί ξανά κοινοποίηση των ονομάτων των πληροφοριών που διαθέτουν μέσω του πρωτοκόλλου δρομολόγησης. Αυτό, όμως, αντιστοιχεί σε υψηλό κόστος. Γι' αυτό το NDN χρησιμοποιεί το πρωτόκολλο Listen First Broadcast Later (LFBL) [38] για να εφαρμόσει την κινητικότητα σε ad-hoc/opportunistic δίκτυα. Στο LFBL, τα πακέτα Ενδιαφέροντος (Interest Packet) υπερχειλίζουν. Όταν μια πιθανή πηγή (δρομολογητής ή εκδότης) των ζητούμενων πληροφοριών λάβει ένα πακέτο Ενδιαφέροντος (Interest Packet), "ακούει" το (ασύρματο) κανάλι για να ανακαλύψει εάν κάποιος άλλος κόμβος έχει ήδη στείλει το αντίστοιχο πακέτο Δεδομένων (Data Packet). Εάν όχι, στέλνει ως απάντηση το ζητούμενο πακέτο Δεδομένων (Data Packet) στο συνδρομητή.

### 2.3.5. Ασφάλεια

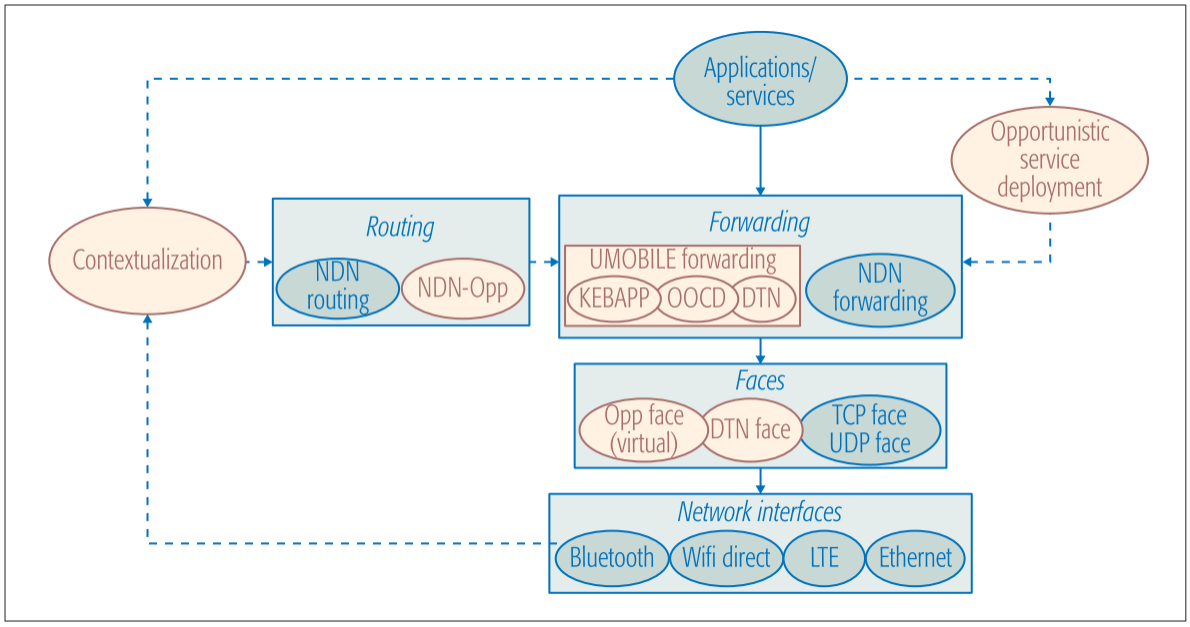
Το NDN υποστηρίζει την αντιστοίχιση των ιεραρχικών ονομάτων που είναι αναγνώσιμα από τον άνθρωπο με τα κατάλληλα αντικείμενα πληροφορίας με έναν επαληθεύσιμο τρόπο [39]. Κάθε πακέτο Δεδομένων (Data Packet) περιλαμβάνει ένα υπογεγραμμένο όνομα μαζί με την πληροφορία που περιλαμβάνει το μήνυμα, καθώς και πληροφορίες σχετικά με το κλειδί που χρησιμοποιείται για την παραγωγή της υπογραφής, π.χ. το δημόσιο κλειδί του υπογράφοντος, ένα πιστοποιητικό γι αυτό το δημόσιο κλειδί ή ένα δείκτη σε αυτά. Αυτό επιτρέπει σε οποιονδήποτε κόμβο, συμπεριλαμβανομένων των δρομολογητών CR, να εξακριβώνει τη σύνδεση μεταξύ του ονόματος του πακέτου (πιθανώς αναγνώσιμου από τον άνθρωπο) και της πληροφορίας που το συνοδεύει.

Προκειμένου να επαληθευτεί η προέλευση των πληροφοριών, ο συνδρομητής θα πρέπει να εμπιστεύεται τον κάτοχο του δημόσιου κλειδιού που χρησιμοποιείται για την υπογραφή του ονόματος. Η ιεραρχική δομή των ονομάτων διευκολύνει τη δημιουργία σχέσεων εμπιστοσύνης, για παράδειγμα, ο τομέας /aueb.gr/ai/main.html μπορεί να υπογραφεί από τον κάτοχο του τομέα /aueb.gr/ai, του οποίου το κλειδί μπορεί να πιστοποιηθεί από τον κάτοχο του τομέα /aueb.gr. Το NDN υποστηρίζει επίσης την ανώνυμη λειτουργία χρησιμοποιώντας μια προσέγγιση τύπου Tor που ονομάζεται ANDaNA [40].

## 2.4. Universal, Mobile-Centric, and Opportunistic Communications Architecture (UMOBILE)

Πρωταρχικός στόχος του UMOBILE είναι να αναπτύξει μία αρχιτεκτονική που προσανατολίζεται στη παροχή υπηρεσιών και βασίζεται στην κινητικότητα. Έτσι καταφέρνει να παραδώσει αποτελεσματικά τόσο περιεχόμενα πληροφορίας όσο και υπηρεσίες σε τελικούς χρήστες, ιδιαίτερα σε απομονωμένες περιοχές. Το UMOBILE διαχωρίζει τις υπηρεσίες από την προέλευση τους μεταβαίνοντας σε ένα πρότυπο που ενσωματώνει στοιχεία τόσο από την προσέγγιση Information Centric Networking και από το Opportunistic Networking, με απώτερο στόχο να οδηγήσει τις υπηρεσίες του δικτύου (υποστήριξη διακοπτόμενων συνδέσεων) και τις υπηρεσίες των χρηστών όσο πιο κοντά στα άκρα. Οδηγώντας αυτές τις υπηρεσίες κοντά στους χρήστες, στοιχεία όπως η διαχείριση των πόρων και η εκμετάλλευση του εύρου ζώνης μπορούν να χρησιμοποιηθούν με το καλύτερο τρόπο. Επιπλέον μπορεί να βελτιωθεί και η διαθεσιμότητα των υπηρεσιών σε απομονωμένα δίκτυα.

Η αρχιτεκτονική του UMOBILE παρουσιάζεται στο Σχήμα 2.7. Οι κόκκινες μονάδες αναπαριστούν τα νέα στοιχεία που αναπτύχθηκαν στην αρχιτεκτονική UMOBILE, ενώ οι μπλε αναπαριστούν τις μονάδες που υπήρχαν ήδη στο πλαίσιο του NDN. Αυτήν τη στιγμή, το UMOBILE εφαρμόζεται σε σταθερά συστήματα Linux τεχνολογίας και σε κινητές συσκευές Android. Οι κύριες τεχνικές αναβαθμίσεις του μοντέλου παρουσιάζονται στις επόμενες ενότητες, πιο αναλυτικά η αρχιτεκτονική παρουσιάζεται στο [41].




Σχήμα 2.7: Αρχιτεκτονική UMOBILE.

### 2.4.1. Προώθηση

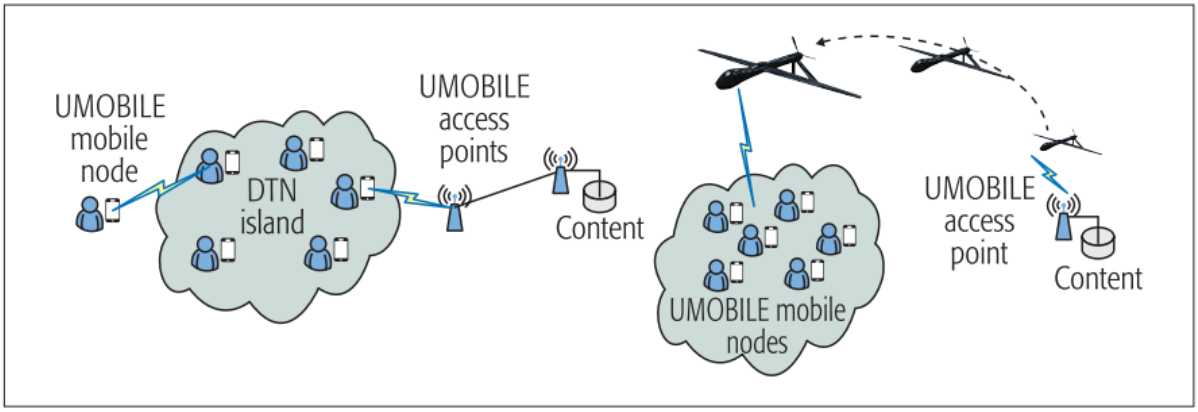
Η βασική ιδέα του μοντέλου επικοινωνίας με βάση την πληροφορία έγκειται στη χρήση των πακέτων Ενδιαφέροντος (Interest Packet) που εκδίδονται από τους πελάτες (pull-based μοντέλο) ή από τους παρόχους περιεχομένου (puss-based μοντέλο). Για μία επιτυχή παράδοση πρέπει να ληφθούν οι σωστές αποφάσεις σχετικά με το που, πότε και σε ποιον πρέπει να προωθηθεί ένα πακέτο.

Η προώθηση των πακέτων στο UMOBILE ακολουθεί το πρότυπο του NDN (best route, broadcast, client control, Ncc), και ενισχύεται επιπλέον με νέες στρατηγικές και μηχανισμούς προώθησης των πακέτων, ώστε να εξασφαλίζει επικοινωνία σε περιβάλλοντα delay-tolerant και opportunistic. Παρακάτω παρουσιάζονται δύο νέες επιλογές προώθησης πακέτων που αναπτύχθηκαν στα πλαίσια αυτού του project.

**DTN Tunneling:** Για την ανάπτυξη αυτής της μονάδας χρησιμοποιήθηκε η υλοποίηση IBR-DTN του πρωτοκόλλου Bundle, ενισχύοντας το NDN με τη δημιουργία μιας καινοτόμου επαφής DTN. Μεταφέροντας τα NDN πακέτα μέσω της νέας DTN επαφής, επεκτείνεται η λειτουργία του NDN και παρέχεται προσβασιμότητα σε απομακρυσμένες περιοχές που δεν υπάρχει συνδεσιμότητα στο Ίντερνετ, καθώς και αξιοπιστία των υπηρεσιών.

Σ’ αυτήν την προσέγγιση, η προώθηση των πακέτων γίνεται στο επίπεδο DTN μέσω των νησίδων DTN που υπάρχουν μεταξύ των κόμβων NDN. Ένας κόμβος UMOBILE (π.χ ένα σημείο πρόσβασης) διατηρεί μία είσοδο στα FIB/PIT προς τον επόμενο κόμβο, ενώ τα πακέτα Ενδιαφέροντος/Δεδομένων (Interest/Data Packet) ενθυλακώνονται και προωθούνται με διαφάνεια από τον ενδιάμεσο κινητό κόμβο DTN. Έτσι, δίνεται η δυνατότητα να προωθούνται τα δεδομένα μεταξύ απομακρυσμένων UMOBILE κόμβων που συνδέονται κατά διαστήματα μέσω των DTN κόμβων χωρίς να αλλάξει κάτι στο NDN. Για την ενίσχυση των υπηρεσιών σε περιβάλλοντα που παρουσιάζουν τέτοιες δυσκολίες

αξιοποιείται η ιδέα της μεταφορά όχι μόνο δεδομένων, αλλά και υπηρεσιών (π.χ εκτελέσιμος κώδικας που μπορεί να τρέχει σε ένα σημείο πρόσβασης) σε απομακρυσμένες περιοχές, δείτε το Σχήμα 2.8. Με αυτό το τρόπο βελτιώνεται τόσο η διαθεσιμότητα των υπηρεσιών όσο και η μείωση καθυστερήσεων, καθώς τα αιτήματα δε χρειάζεται να μεταφέρονται περιοδικά.




Σχήμα 2.8: Κύριες λειτουργίες του DTN. Αριστερά λειτουργία DTN tunneling, δεξιά η λειτουργία data muling.

**Opportunistic Off-Path Content Discovery (OOCD):** Όλες οι προτεινόμενες αρχιτεκτονικές, συμπεριλαμβανομένου και του NDN, επικεντρώνονται στη δρομολόγηση των αιτημάτων προς το πυρήνα του δικτύου. Στο [8] υποστηρίζεται ότι αυτή η μέθοδος είναι ενάντια στο αρχικό όραμα της δημιουργίας ενός εγγενούς δικτύου διανομής περιεχομένου και πως αποτρέπει τα αιτήματα να ανακαλύψουν δεδομένα που μπορεί να βρίσκονται σε γειτονικούς κόμβους, εκτός και αν αυτή η ανακάλυψη οδηγεί στην πιο σύντομη διαδρομή προς το πυρήνα του δικτύου.

Για την επίλυση του προβλήματος αυτού, αναπτύχθηκε μια βελτιωμένη δομή της δρομολόγησης του NDN. Δημιουργήθηκε ένας καινούργιος πίνακας D-FIB που αποθηκεύει όλα τα αιτήματα που έχουν ικανοποιηθεί επιτυχώς. Ο πίνακας D-FIB λειτουργεί παρόμοια με τον FIB, μόνο που αυτός καταχωρεί του κόμβους που έχουν λάβει πακέτα Δεδομένων (Data Packet) και άρα έχουν αποθηκευτεί τοπικά στη μνήμη τους. Οι πίνακες D-FIB χρησιμοποιούνται αποτελεσματικά για να ανιχνεύσει περιεχόμενο το οποίο είναι αποθηκευμένο εκτός διαδρομής (off-path). Έτσι αυξάνεται ο ρυθμός cache hit και μειώνεται το overhead [42].

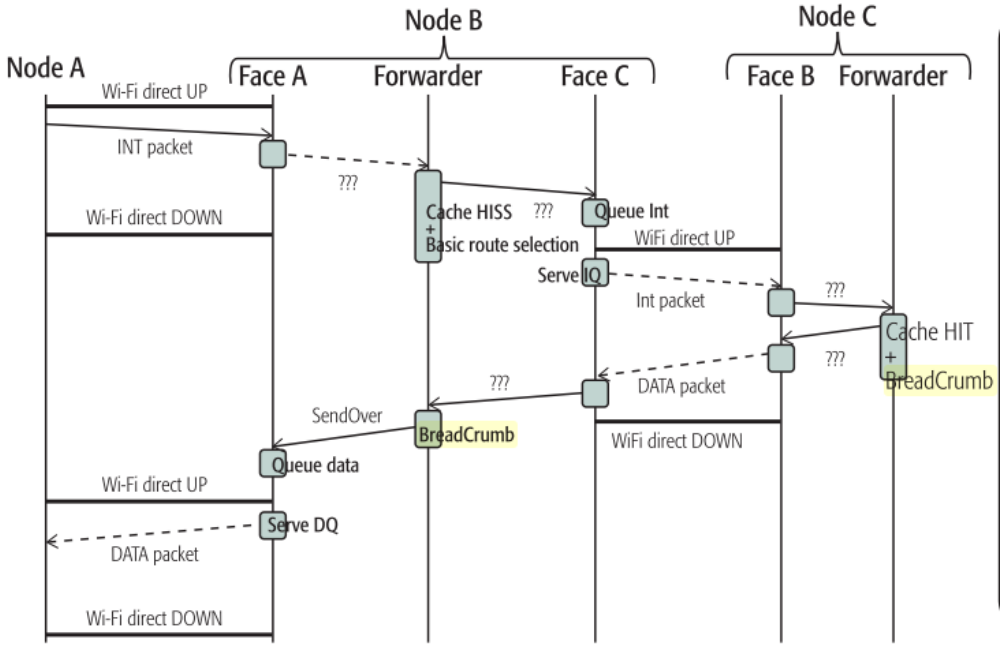
### 2.4.2. Δρομολόγηση

Η δρομολόγηση στο UMOBILE στοχεύει στην ασύρματη επικοινωνία σε opportunistic δίκτυα. Γι αυτό αναπτύχθηκε το NDN-Opp το οποίο εκμεταλλεύεται τις δυνατότητες της ασύρματης επικοινωνίας. Το NDN-Opp διαχειρίζεται τη δυναμική των opportunistic δικτύων μέσω της προώθησης πακέτων Ενδιαφέροντος σε γειτονικούς κόμβους δίνοντας τους πολλές πιθανότητες να συναντήσουν κάποιον κόμβο ο οποίος θα διαθέτει τα επιθυμητά δεδομένα. Για να εξασφαλίσει τη συμβατότητα με το NDN χρησιμοποιεί τη στρατηγική προώθησης βέλτιστης διαδρομής του NDN (best route forwarding strategy) για τη

παράδοση των πακέτων Ενδιαφέροντος (Interest Packet), αλλά και τη προσέγγιση “breadcrumb” [8] για τη παράδοση των πακέτων Δεδομένων (Data Packet) με βάση τη πληροφορία που είναι αποθηκευμένη στο PIT.

Για να διαχειριστεί τις διακοπτόμενες συνδέσεις διαθέτει διεπαφές (Opp Faces) που βασίζονται στη τεχνολογία Wifi-Direct [43]. Κάθε επαφή μπορεί να βρίσκεται σε δύο καταστάσεις ( ON-OFF) και διαθέτει δύο ουρές:

- Ουρά των Interest (IQ): αποθηκεύει τα πακέτα Ενδιαφέροντος (Interest Packet) που πρόκειται να στείλει.
- Ουρά των Data (DQ) : αποθηκεύει δείκτες προς το Content Store για τα μπλοκ Δεδομένων που πρόκειται να στείλει.

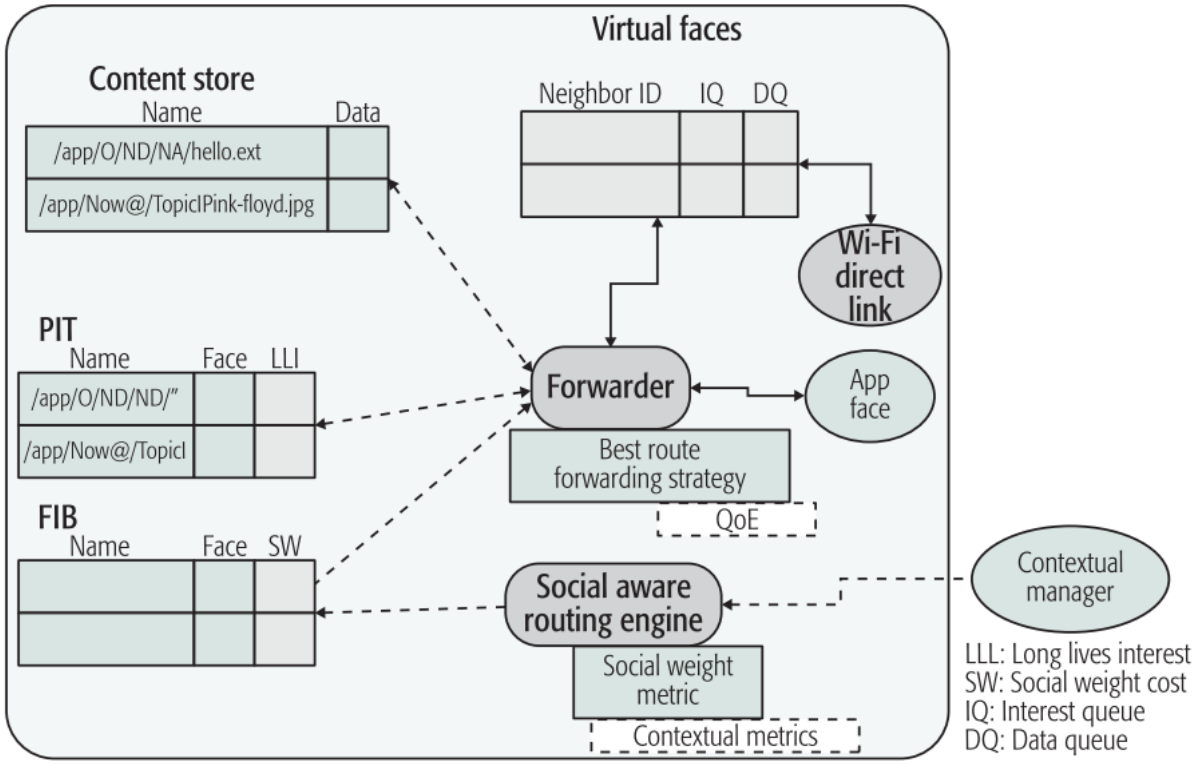



Σχήμα 2.9: Λειτουργία NDN-Opp.

Η λειτουργία της επαφής Opp είναι παρόμοια με τη λειτουργία του NDN, όπως φαίνεται στο Σχήμα 2.9, καθώς η κατάσταση του κόμβου αλλάζει κατά την άφιξη ενός πακέτου Ενδιαφέροντος ή Δεδομένων (Interest/Data packet). Η βασική διαφορά τους είναι ότι η λειτουργία του NDN τερματίζει με την αποστολή του πακέτου, ενώ η λειτουργία των Opp Faces τερματίζει με την αποθήκευση των πακέτων τοπικά στη μνήμη του. Όταν το Opp Face βρίσκεται στη κατάσταση ON τότε τα πακέτα μεταδίδονται μέσω των serving interest και serving data, Σχήμα 2.9.

Στο Σχήμα 2.10 παρουσιάζεται η σχεδίαση ενός NDN-Opp κόμβου, η οποία βασίζεται σε ένα καινοτόμο δρομολογητή και σε μία μηχανή δρομολόγησης. Κατά τη λήψη των πακέτων Ενδιαφέροντος (Interest Packet) αποθηκεύεται στον PIT μία επιπλέον πληροφορία, η διάρκεια ζωής του Ενδιαφέροντος (LLI). Τα πακέτα Ενδιαφέροντος (Interest Packet) προωθούνται με βάση τη στρατηγική προώθησης βέλτιστης διαδρομής του NDN, ενώ τα κόστη των ονομάτων που αποθηκεύονται στο FIB (SW) εξαρτώνται από μία παράμετρο που λαμβάνει υπόψη κοινωνικά κριτήρια (πχ. ομοιότητες σε κοινωνικό επίπεδο ή ομοιότητες ζήτησης δεδομένων) και υπολογίζεται από μία μηχανή δρομολόγησης “social

aware”. Η μηχανή αυτή υπολογίζει τα κόστη με βάση τη χρήση των Opp Faces που απαιτείται για να ανακτηθούν τα επιθυμητά δεδομένα. Αυτήν τη στιγμή η μηχανή που χρησιμοποιεί το NDN-Opp είναι η Time-Evolving Contact Duration algorithm (TECD) [44].Τα πακέτα Δεδομένων (Data Packet) ακολουθούν τους γνωστούς κανόνες προώθησης του NDN.



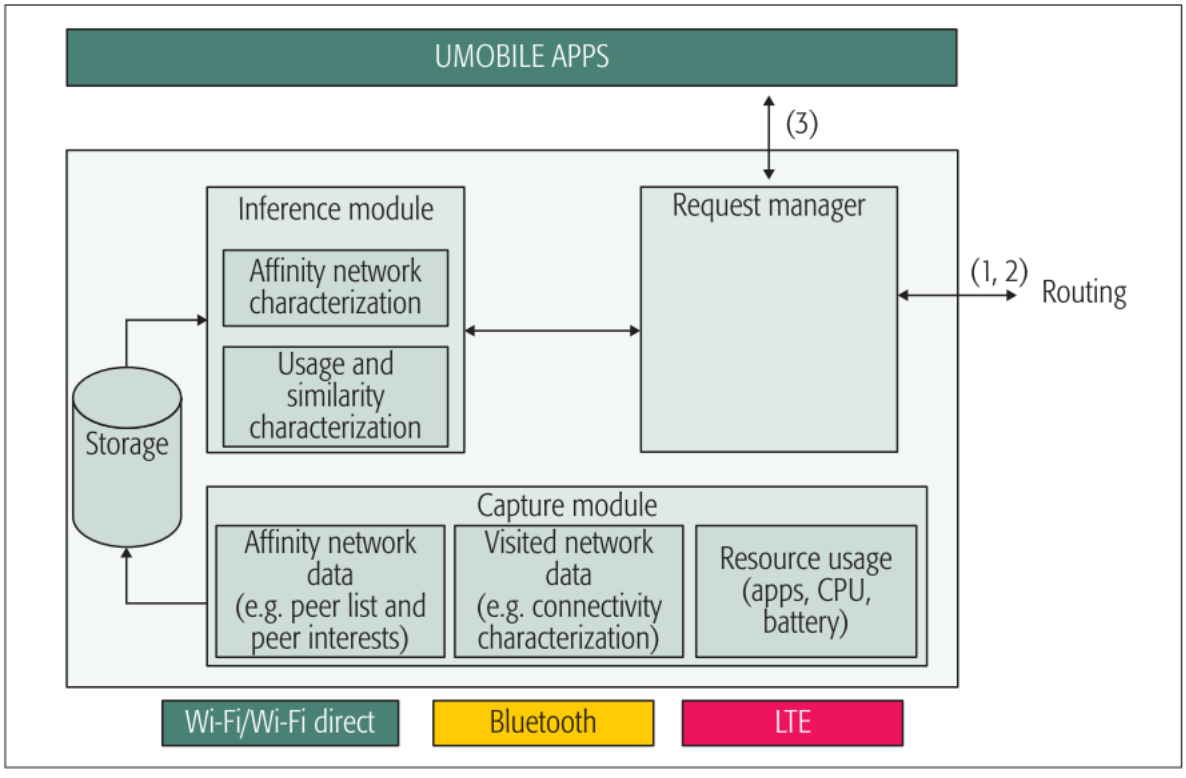

Σχήμα 2.10: Σχεδίαση NDN-Opp κόμβου.

### 2.4.3. Contextualization

Η λήψη κοινωνικών κριτηρίων που αναφέρθηκε προηγουμένως (social awareness) τοποθετείται και στο context plane με σκοπό τη βελτίωση της διάδοσης πληροφορίας. Το contextual plane προσδιορίζεται από το contextual manager (CM) του Σχήματος 2.11. Αποτελεί μία υπηρεσία που τρέχει στο παρασκήνιο σε ένα σημείο πρόσβασης ή σε μία τερματική συσκευή. Το CM λαμβάνει πληροφορίες σχετικά με το δίκτυο συσχέτισης των συσκευών (π.χ σχετικά με τους peers στο χρόνο και στο χώρο) καθώς και πληροφορίες σχετικά με τη χρήση των πόρων ( π.χ χρήση της CPU). Τα κόστη τα οποία παράγονται από το contextualization προωθούνται κατ’ απαίτηση είτε περιοδικά σε άλλα τμήματα του UMOBILE για να συνδράμουν σε διάφορες λειτουργίες του με σκοπό την αποδοτικότερη διάδοση της πληροφορίας.

Το CM περιλαμβάνει τρεις διεπαφές (δύο για το τμήμα της δρομολόγησης, (1), (2), και μία για τις εφαρμογές (3)) επιτρέποντας σε άλλα τμήματα και εφαρμογές να αιτούνται ή να λαμβάνουν πληροφορίες από αυτό. Οι διεπαφές παρέχουν δύο τύπους πληροφορίας:

- Δείκτες που προσδιορίζουν το δίκτυο συσχέτισης, όπως λίστες που περιλαμβάνουν τους peers, δείκτες που προσδιορίζουν την κατάσταση των peers(π.χ στάθμη μπαταρίας)
- Δείκτες που προσδιορίζουν χαρακτηριστικά της χρήσης και της ομοιότητας, όπως η γεω-τοποθεσία.




Σχήμα 2.11: Contextual manager

### 2.4.4. Northbound APIs

Οι εφαρμογές αλληλεπιδρούν με το δίκτυο με διάφορους τρόπους. Το UMOBILE παρέχει ένα πλαίσιο για να καλύψει διάφορες ανάγκες των εφαρμογών. Αυτό παρουσιάζεται παρακάτω.

**Υπηρεσίες push:** το NDN εγγενώς υποστηρίζει υπηρεσίες pull, όπου η μεταφορά επωνομαζόμενου περιεχομένου ξεκινά από το συνδρομητή. Αυτό το μοντέλο, όμως, δεν είναι ιδανικό για τη μετάδοση περιοδικά παραγόμενων εφήμερων δεδομένων [45]. Γι ’αυτό αναπτύχθηκαν το μοντέλο push υπηρεσιών, όπου οι παραγωγοί ξεκινούν τη μεταφορά των δεδομένων στους καταναλωτές [41].

**Δημοσκόπηση Interest**: Ιδανικό για εφαρμογές όπου ο συνδρομητής δεν γνωρίζει πότε θα διαθέσει ο παραγωγός ένα περιεχόμενο. Βασίζεται σε ένα μηχανισμό δημοσκόπησης που επιτρέπει στο συνδρομητή να εκδίδει περιοδικά αιτήματα προς τον παραγωγό. Το μειονέκτημα αυτής της υπηρεσίας είναι ότι αυξάνει το κόστος (overhead) του

δικτύου και το πόσο πρόσφατα θεωρούνται τα ληφθέντα δεδομένα εξαρτάται από τη συχνότητα της δημοσκόπησης.

**Ειδοποιήσεις Interest**: Χρήσιμο όταν το περιεχόμενο είναι ένα κομμάτι κειμένου που είναι αρκετά μικρό ώστε να επισυνάπτεται από το παραγωγό σε ένα όνομα ενός πακέτου Ενδιαφέροντος (Interest Packet). Κατά τη λήψη του μηνύματος Ενδιαφέροντος (Interest Packet), ο παραλήπτης μπορεί προαιρετικά να στείλει ένα "ack Data" για να επιβεβαιώσει τη λήψη του.

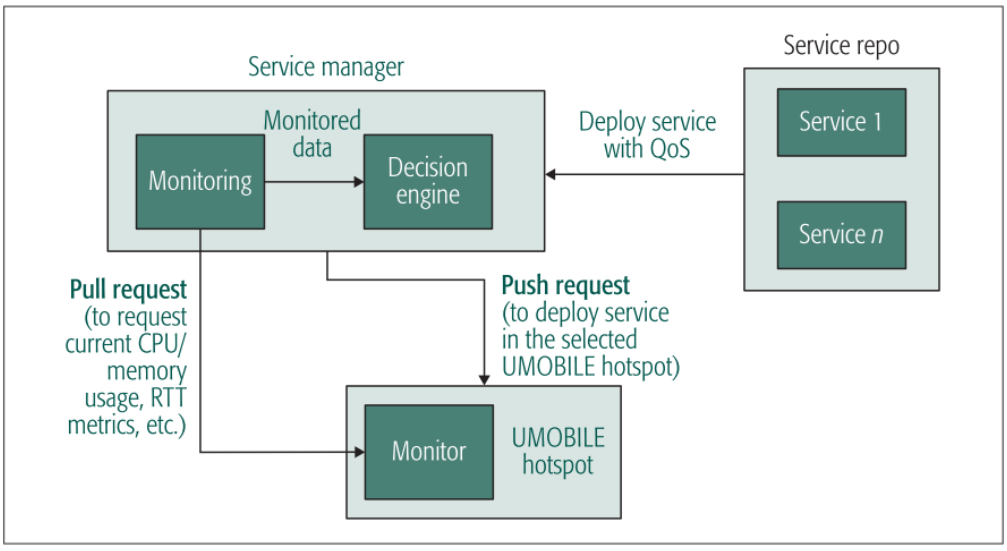
**Κοινοποίηση διάδοσης δεδομένων**: Όταν ο παραγωγός κάνει διαθέσιμο το περιεχόμενο στέλνει μία ειδοποίηση στο συνδρομητή ο οποίος με τη σειρά του απαντά με ένα μήνυμα Ενδιαφέροντος (Interest Packet) που ζητά το περιεχόμενο. Το περιεχόμενο χωρίζεται σε τμήματα. Κάθε τμήμα που αποστέλλεται περιλαμβάνει την πληροφορία καθώς και τον αριθμό τον εναπομείναντων τμημάτων, τα οποία στέλνονται στη συνέχεια.

**KEBAPP**: Το πλαίσιο αυτό [46] επιτρέπει την πρόσβαση στην επιθυμητή επεξεργασμένη και μη εξατομικευμένη πληροφορία μέσω της κοινής χρήσης των εφαρμογών, αξιοποιώντας αποτελεσματικά ένα σύνολο πόρων. Εισάγει την έννοια των λέξεων κλειδιών για να επιτρέψει την επίκληση εφαρμογών που μπορεί να βρίσκονται σε κινητές συσκευές σε κάποια κοντινή τοποθεσία.

### 2.4.5. Ποιότητα των υπηρεσιών

Στο UMOBILE δίνεται ιδιαίτερη σημασία στην ποιότητα των υπηρεσιών. Γι αυτό αναπτύχθηκε μία πλατφόρμα που χρησιμοποιείται για τη μεταφορά υπηρεσιών στα άκρα, μειώνοντας τη κίνηση στο πυρήνα του δικτύου και παρέχοντας υπηρεσίες σε απομακρυσμένες περιοχές.

Υποθέτει ένα επιχειρηματικό μοντέλο όπου ο πάροχος δικτύου έχει στη διάθεση του το δίκτυο, υπολογιστικούς και αποθηκευτικούς χώρους και προτίθεται να προσομοιώσει τις υπηρεσίες του host [47].




Σχήμα 2.12: Opportunistic Service Deployment

Αυτό το μοντέλο χρησιμοποιεί τις αφηρημένες έννοιες που προσφέρονται από το NDN και επιπλέον εκμεταλλεύεται την εποπτεία και την προσομοίωση με χρήση lightweight τεχνολογιών( dockerizing) για την αξιοποίηση υπηρεσιών κατ΄απαίτηση στα άκρα του δικτύου. Ο διαχειριστής των υπηρεσιών διαθέτει όλες τις απαραίτητες λειτουργίες.

Περιλαμβάνει ένα χώρο αποθήκευσης των υπηρεσιών (service repo) όπου αποθηκεύονται όλες οι εικόνες docker που αναπαριστούν υπηρεσίες (service1, …, servicen) (Σχήμα 2.12). Επιπλέον περιλαμβάνει μια μηχανή αποφάσεων, που αποφασίζει ποια υπηρεσία να εφαρμόσει και σε ποιο UMOBILE hotspot. Οι αποφάσεις λαμβάνονται με βάση την πληροφορία που δέχεται από τη μονάδα monitoring, που είναι υπεύθυνη για να υποβάλλει αιτήματα pull στη μονάδα monitor που χρησιμοποιεί κάθε UMOBILE hotspot για να συλλέξει μετρικές σχετικές με την κατανάλωση των πόρων του συστήματος.

# Κεφάλαιο 3

# Εικονικοποίηση (Virtualization) και Docker

Η εικονικοποίηση (virtualization) είναι ένα πλαίσιο ή μία μέθοδος κατανομής των υπολογιστικών πόρων μεταξύ πολλών εφαρμογών που τρέχουν ταυτόχρονα [48], εφαρμόζοντας μία ή περισσότερες τεχνολογίες, όπως κατανομή του hardware και software, κατανομή του χρόνου (time-sharing) μεταξύ πολλών εφαρμογών, μερική ή ολοκληρωτική εξομοίωση μηχανών, ποιότητα των υπηρεσιών κ.α [49]. Ιδιαίτερα για την εικονικοποίηση (virtualization) λειτουργικών συστημάτων αναπτύχθηκε η τεχνολογία των εικονικών μηχανών (Virtual Machine-VM).

Μια εικονική μηχανή αποτελείται από τον hypervisor και από τον Guest. Ο hypervisor είναι το λογισμικό που ενεργοποιεί την εικονικοποίηση (virtualization). Είναι υπεύθυνος για τη δημιουργία του εικονικού περιβάλλοντος μέσα στο οποίο θα λειτουργήσει ο Guest. Επιπλέον, επιβλέπει τη λειτουργία του συστήματος του Guest παρέχοντας τους απαραίτητους πόρους.

Μια μέθοδος που αναπτύχθηκε ιδιαίτερα τα τελευταία χρόνια έρχεται για να ξεπεράσει τις εικονικές μηχανές. Το Containerization [50] βοηθάει τους προγραμματιστές να αναπτύξουν εφαρμογές πιο γρήγορα, εύκολα και με ασφάλεια. Τα Container αναφέρονται συχνά ως “lightweight”, εννοώντας ότι μοιράζονται το λειτουργικό σύστημα Kernel και εξαλείφουν το κόστος της συσχέτισης ενός λειτουργικού συστήματος με κάθε εφαρμογή. Τα Container είναι μικρότερα σε χωρητικότητα από τα VM και απαιτούν μικρότερο χρόνο εκκίνησης, επιτρέποντας περισσότερα Container να τρέχουν σε ένα σύστημα.

Παρόλο που η ιδέα του Containerization υπάρχει εδώ και δεκαετίες, η εμφάνιση μιας πλατφόρμας λογισμικού ανοιχτού κώδικα, Docker Engine [51] το 2013, πυροδότησε την υιοθέτηση αυτής της τεχνολογίας. Το Docker διαχειρίζεται τα container μέσω της διαχείρισης του χώρου ονομάτων (namespaces). Χρησιμοποιεί 5 χώρους ονομάτων: Process ID (PID), Networking, Inter-Process Communication (IPC), Mount (MNT) namespace, UNIX Time-sharing System (UTS) και cgroups που ελέγχουν την κατανομή των διαθέσιμων πόρων (μνήμης, CPU, δικτύου, I/O) του συστήματος. Το Docker Engine αποτελείται από τρεις μονάδες: μία διεργασία daemon, τη γραμμή εντολών (CLI) και APIs. Επίσης, η αρχιτεκτονική Docker περιλαμβάνει τον docker daemon, το docker client, τα μητρώα docker, τα αντικείμενα docker (εικόνες, container, υπηρεσίες).

Πρακτικά το Docker χρησιμοποιεί το Dockerfile για να δημιουργήσει μία εικόνα με την οποία μπορεί να κατασκευάσει τα container. Χρησιμοποιώντας το Docker API ή CLI μπορεί κανείς να δημιουργήσει, να ξεκινήσει, να σταματήσει ή να διαγράψει ένα container. Τέλος, το Docker προσφέρει σταθερό και απομονωμένο περιβάλλον, φορητότητα, δυνατότητα επέκτασης και καλύτερη απόδοση.

## 3.1. Εικονικοποίηση (Virtualization)

Η εικονικοποίηση (virtualization) είναι μία διαδικασία κατά την οποία προσομοιώνεται ένα σύστημα σε αφηρημένο επίπεδο και δεν εκτελείται σε πραγματικό hardware. Συνήθως, αναφέρεται στην εκτέλεση πολλαπλών λειτουργικών συστημάτων ταυτόχρονα σε ένα μόνο σύστημα-μηχανή δίνοντας τη δυνατότητα σε κάθε εφαρμογή να λειτουργεί στο δικό της απομονωμένο περιβάλλον με το κατάλληλο λειτουργικό σύστημα, βιβλιοθήκες και άλλα απαραίτητα εργαλεία.

Η εικονικοποίηση (virtualization) είναι ένα πάρα πολύ σημαντικό εργαλείο. Για τους χρήστες υπολογιστών είναι σημαντικό, καθώς μία εφαρμογή της οποίας η εκτέλεση προορίζεται για ένα διαφορετικό λειτουργικό σύστημα μπορεί να εκτελείται χωρίς να χρησιμοποιηθεί διαφορετικός υπολογιστής ή να γίνει επανεκκίνηση σε άλλο λειτουργικό σύστημα. Για τους διαχειριστές διακομιστών, η εικονικοποίηση (virtualization) προσφέρει επίσης τη δυνατότητα εκτέλεσης διαφορετικών λειτουργικών συστημάτων. Όμως το πιο σημαντικό, προσφέρει ένα μέσο για την τμηματοποίηση ενός μεγάλου συστήματος σε μικρότερα μέρη επιτρέποντας στο διακομιστή να χρησιμοποιείται πιο αποτελεσματικά από διαφορετικούς χρήστες ή εφαρμογές με διαφορετικές ανάγκες. Επιτρέπει, επίσης, την απομόνωση διατηρώντας τα προγράμματα που εκτελούνται μέσα σε μια εικονική μηχανή ασφαλή από τις διαδικασίες που πραγματοποιούνται σε μια άλλη εικονική μηχανή στον ίδιο κεντρικό υπολογιστή.

Ένας hypervisor είναι ένα πρόγραμμα για τη δημιουργία και λειτουργία εικονικών μηχανών. Οι hypervisors είναι χωρισμένοι σε δύο κατηγορίες:

- "bare metal" hypervisors που εκτελούν εικονικές μηχανές απευθείας στο υλικό του συστήματος, ουσιαστικά συμπεριφέρονται ως λειτουργικό σύστημα.
- "host" hypervisors που συμπεριφέρονται περισσότερο σαν παραδοσιακές εφαρμογές που μπορούν να ξεκινήσουν και να σταματήσουν σαν ένα κανονικό πρόγραμμα.

Μια εικονική μηχανή είναι μία προσομοίωση συστήματος που λειτουργεί πάνω από ένα άλλο σύστημα. Οι εικονικές μηχανές ενδέχεται να έχουν πρόσβαση σε διάφορους πόρους:

1. υπολογιστική ισχύ
2. περιορισμένη πρόσβαση στη CPU και τη μνήμη του κεντρικού υπολογιστή
3. μία ή περισσότερες φυσικές ή εικονικές συσκευές δίσκου για αποθήκευση
4. εικονικό ή πραγματικό δίκτυο διεπαφών
5. διάφορες συσκευές όπως κάρτες βίντεο, συσκευές USB ή άλλο υλικό που είναι κοινόχρηστο με την εικονική μηχανή.

Εάν η εικονική μηχανή είναι αποθηκευμένη σε εικονικό δίσκο, αυτό αναφέρεται συχνά ως εικόνα δίσκου. Μια εικόνα δίσκου (image) μπορεί να περιέχει τα αρχεία για εκκίνηση μιας εικονικής μηχανής ή μπορεί να περιέχει άλλες ειδικές ανάγκες αποθήκευσης.

## 3.2. Linux Container (LXC)

Τα container είναι απομονωμένα περιβάλλοντα τα οποία δημιουργούνται για την εκτέλεση εφαρμογών και τα οποία μοιράζονται το υποκείμενο kernel [52] σύστημα. Το περιβάλλον αυτό εκτός από την ίδια την εφαρμογή περιλαμβάνει τις απαιτούμενες βιβλιοθήκες, τα δυαδικά εκτελέσιμα αρχεία (binaries) και τα αρχεία ρύθμισης των παραμέτρων για ένα εκτελέσιμο πρόγραμμα (configuration files).

Η πρώτη τεχνολογία υλοποίησης των Container είναι της Linux, LXC. Είναι μία μέθοδος εικονικοποίησης (virtualization) σε επίπεδο λειτουργικού συστήματος για την εκτέλεση πολλαπλών απομονωμένων συστημάτων Linux σε ένα μόνο εξυπηρετητή. Η LXC τεχνολογία έγινε εφικτή λόγω δύο πολύ σημαντικών χαρακτηριστικών των Linux :

- namespaces: ενσωματώνουν ένα σύνολο πόρων του συστήματος, ωστέ κάθε container να διαθέτει τους δικούς του πόρους, όπως σύστημα αρχείων, διαχείριση διεργασιών, διεύθυνση IP, κ.α.

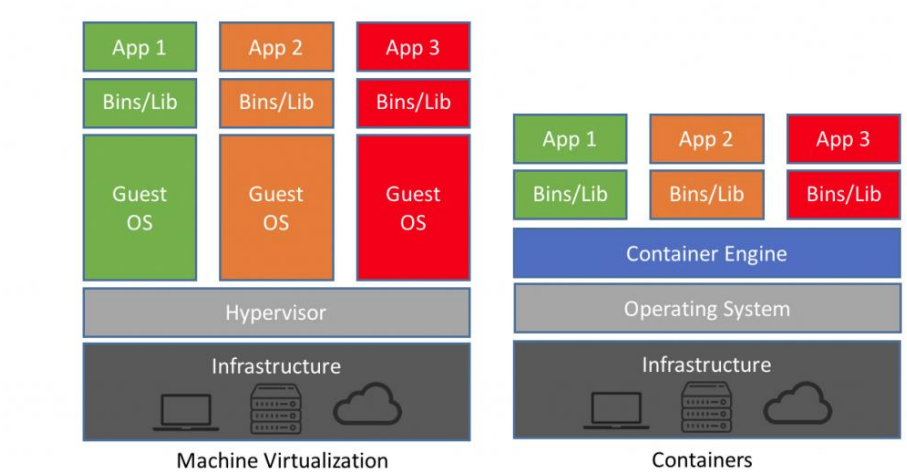
| Namespaces | Λειτουργία |
|---|---|
| pid namespace | Υπεύθυνο για την απομόνωση των διεργασιών (PID: Process ID) |
| net namespace | Προσομοιώνει τη στοίβα δικτύου. Διαχειρίζεται τις διεπαφές (NET: Networking). |
| ipc namespace | Διαχειρίζεται την πρόσβαση στους IPC πόρους [53] (IPC: InterProcess Communication) |
| mnt namespace | Υπεύθυνο για τη διαχείριση των σημείων τοποθέτησης των αρχείων (MNT: Mount) |
| uts namespace | Απομονώνει το kernel και διαθέτει αναγνωριστικά έκδοσης UTS: Unix Timesharing System) |

- cgroups: ελέγχουν το διαμοιρασμό των διαθέσιμων πόρων( μνήμης, CPU, δικτύου, I/O) του συστήματος θέτοντας όρια και περιορισμούς.

## 3.3. Container vs Virtual Machine

Η τεχνολογία της εικονικής μηχανής (Virtual Machine-VM) περιλαμβάνει ένα ολόκληρο λειτουργικό σύστημα, καθώς και την ίδια την εφαρμογή. Ένας διακομιστής ο οποίος τρέχει για παράδειγμα τρεις εικονικές μηχανές (VMs) θα έχει έναν επόπτη και τρία ξεχωριστά λειτουργικά συστήματα να τρέχουν πάνω από αυτόν Σχήμα 3.1.

Αντίθετα ένας διακομιστής ο οποίος τρέχει τρία container τρέχει ένα μοναδικό λειτουργικό σύστημα το οποίο διαμοιράζεται. Επιπλέον μοιράζονται και άλλους πόρους. Αυτό έχει ως αποτέλεσμα τα containers να έχουν σημαντικά πιο μικρό μέγεθος σε σχέση με τις εικονικές μηχανές και να ξεκινάνε πολύ πιο γρήγορα.



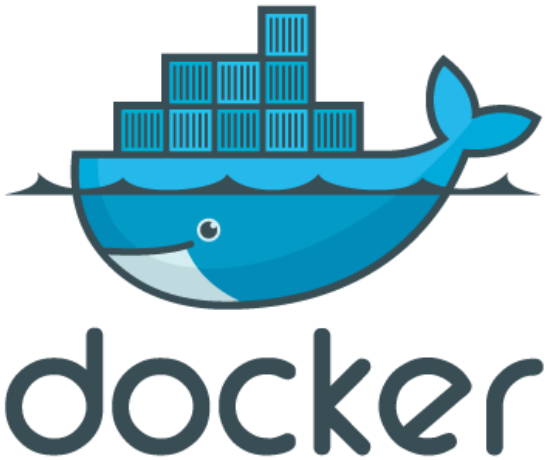

Σχήμα 3.1 : Στα αριστερά παρουσιάζεται η τεχνολογία των virtual machines και στα δεξιά η τεχνολογία των container.

Τα στοιχεία του λειτουργικού συστήματος τα οποία μοιράζονται είναι μόνο για ανάγνωση και το κάθε container έχει ένα ξεχωριστό επίπεδο για εγγραφή δεδομένων. Επιπλέον η χρήση των container δεν περιλαμβάνει hypervisor, άρα μειώνεται το overhead.

## 3.4. Docker

Το Docker είναι μια πλατφόρμα λογισμικού ανοιχτού κώδικα η οποία είναι γραμμένη σε γλώσσα GO που υλοποιεί την τεχνολογία LXC. Η επιτυχία αυτής της πλατφόρμας την καθιστά πλέον συνώνυμη με την τεχνολογία container. Αρχικά δημιουργήθηκε για τα Linux, σήμερα όμως τρέχει επίσης σε Windows και MacOS. Ουσιαστικά το Docker παρέχει τη δυνατότητα ανάπτυξης εφαρμογών σε container με τη χρήση των namespaces και cgroups. Η απομόνωση και η ασφάλεια που παρέχεται επιτρέπουν στο χρήστη να τρέχει ταυτόχρονα

πολλά container σε έναν εξυπηρετητή. Τα container είναι ελαφριά, χρησιμοποιούν ελάχιστους πόρους και περιλαμβάνουν ότι χρειάζεται για να λειτουργήσει η εφαρμογή.



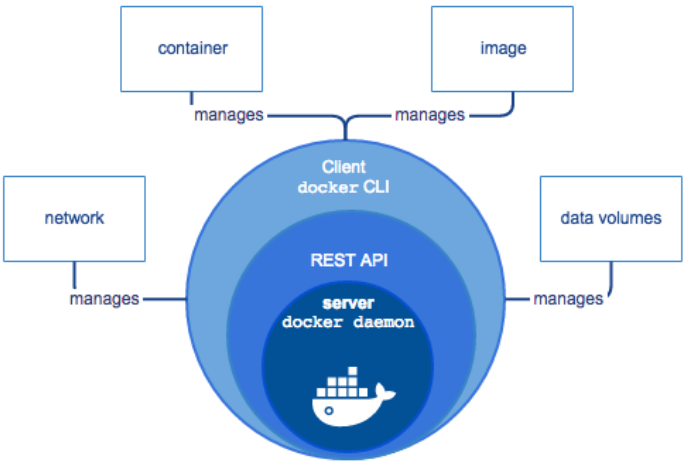

Σχήμα 3.2: Docker

Το Docker παρέχει μία πλατφόρμα και διάφορα εργαλεία ωστέ να διαχειρίζεται ο χρήστης τα container. Έτσι τα container αποτελούν τη μονάδα για τη διανομή και δοκιμή της εφαρμογής. Όταν πια η εφαρμογή είναι έτοιμη και δοκιμασμένη μπορεί να χρησιμοποιηθεί σε ένα περιβάλλον εφαρμογής, ως ένα container ή ως μια ενορχηστρωμένη υπηρεσία. Η λειτουργία του θα είναι η ίδια είτε έχουμε ένα τοπικό κέντρο δεδομένων, ένα πάροχο cloud υπηρεσιών ή ακόμα και ένα συνδυασμό αυτών των περιπτώσεων.

## 3.5. Docker Engine

Η μηχανή Docker είναι μια εφαρμογή πελάτη-διακομιστή (client-server) η οποία αποτελείται από τα παρακάτω στοιχεία Σχήμα 3.3:

- Ένα διακομιστή ο οποίος είναι ένας τύπος διεργασίας μεγάλης διάρκειας που ονομάζεται daemon.
- Ένα REST API που καθορίζει τις διεπαφές που τα προγράμματα μπορούν να χρησιμοποιήσουν για να επικοινωνήσουν με τον daemon και να τον καθοδηγήσουν ως προς το τι να κάνει.
- Μια διεπαφή γραμμής εντολών (CLI) εφαρμογής πελάτη, η εντολή docker.




Σχήμα 3.3: Δομή Docker Engine.

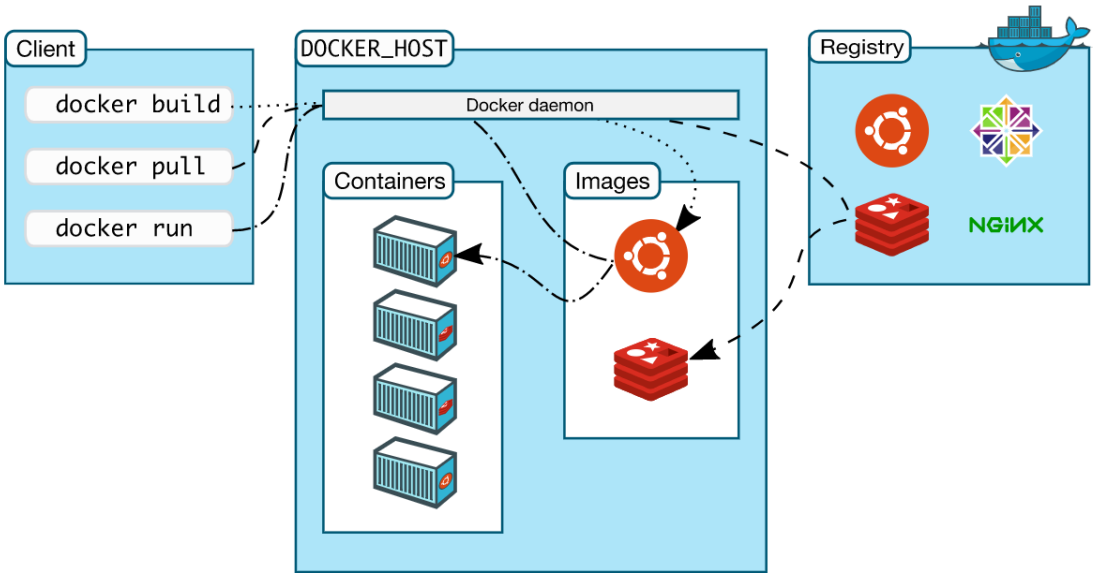
Το CLI χρησιμοποιεί το Docker REST API για τον έλεγχο ή την αλληλεπίδραση με τον Docker daemon μέσω αρχείων script ή άμεσα μέσω εντολών CLI. Πολλές άλλες εφαρμογές Docker χρησιμοποιούν το υποκείμενο API και CLI.

## 3.6. Αρχιτεκτονική Docker

Το Docker χρησιμοποιεί αρχιτεκτονική πελάτη-διακομιστή (Σχήμα 3.4). Ο πελάτης Docker επικοινωνεί με τον Docker daemon, ο οποίος πραγματοποιεί την κατασκευή, τη λειτουργία και τη διανομή των container του Docker. Ο πελάτης και ο Docker daemon μπορούν να εκτελούνται στο ίδιο σύστημα ή μπορεί να συνδεθεί ένας πελάτης Docker με έναν απομακρυσμένο Docker daemon. Ο πελάτης και ο δαίμονας Docker επικοινωνούν χρησιμοποιώντας ένα REST API, μέσω υποδοχών UNIX ή διασύνδεσης δικτύου.

### 3.6.1. Δαίμονας Docker (daemon)

Ο δαίμονας Docker (dockerd) ακούει σε αιτήματα API Docker και διαχειρίζεται αντικείμενα Docker όπως εικόνες (images), container, δίκτυα και σε μηχανισμούς διατήρησης των δεδομένων μετά το τερματισμού του container( volumes). Ο daemon μπορεί επίσης να επικοινωνήσει με άλλους για τη διαχείριση των υπηρεσιών Docker.




Σχήμα 3.4: Αρχιτεκτονική Docker.

### 3.6.2. Docker client

Ο Docker client (docker) είναι ο κύριος τρόπος αλληλεπίδρασης πολλών χρηστών του Docker με το εργαλείο Docker. Όταν χρησιμοποιούνται εντολές όπως *docker run*, ο client στέλνει αυτές τις εντολές στο δαίμονα ο οποίος τις εκτελεί. Η εντολή docker χρησιμοποιεί το Docker API. Ο Docker client μπορεί να επικοινωνήσει με περισσότερους από έναν daemon.

### 3.6.3. Μητρώα Docker (registries)

Το μητρώο Docker αποθηκεύει εικόνες Docker. Το Docker Hub είναι ένα δημόσιο μητρώο που μπορεί να χρησιμοποιήσει ο καθένας. Το Docker έχει ρυθμιστεί για αναζήτηση εικόνων στο Docker Hub από προεπιλογή. Μπορεί κανείς να εκτελέσει το δικό του ιδιωτικό μητρώο. Όταν χρησιμοποιούνται οι εντολές *docker pull* ή *docker run*, οι απαιτούμενες εικόνες λαμβάνονται από το διαμορφωμένο μητρώο. Όταν χρησιμοποιείται η εντολή *docker push* η εικόνα μεταφέρεται στο διαμορφωμένο μητρώο.

### 3.6.4. Αντικείμενα Docker (objects)

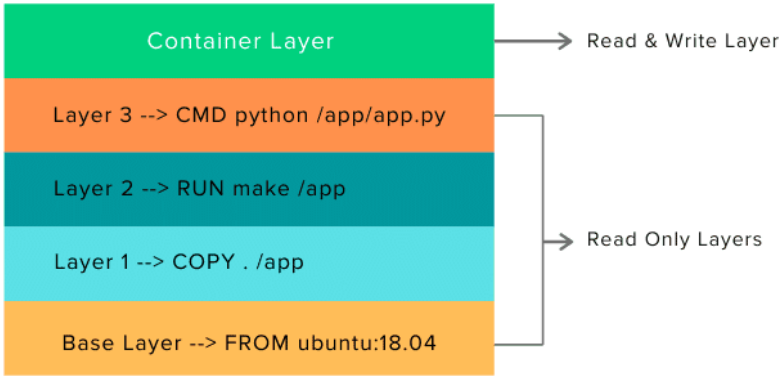
Μερικά από τα αντικείμενα που δημιουργούνται κατά τη χρήση του Docker παρουσιάζονται παρακάτω.

- Εικόνα (image)

Μια εικόνα είναι ένα πρότυπο μόνο-ανάγνωσης με οδηγίες για τη δημιουργία ενός container. Συχνά, μια εικόνα βασίζεται σε μία άλλη εικόνα (base image), με κάποιες

επιπλέον προσαρμογές. Για παράδειγμα, μπορεί μία εικόνα να βασίζεται στη βασική εικόνα (base image) μιας έκδοσης Ubuntu, αλλά εγκαθιστά τον διακομιστή ιστού Apache [54] και μία εφαρμογή, καθώς και τις ρυθμίσεις που απαιτούνται για την εκτέλεση της εφαρμογής.

Μπορεί ο καθένας να δημιουργήσει τη δική του εικόνα ή να χρησιμοποιήσει μόνο αυτές που δημιουργήθηκαν από άλλους και δημοσιεύθηκαν σε κάποιο μητρώο. Για τη δημιουργία μιας εικόνας, δημιουργείται ένα Dockerfile με μια απλή σύνταξη για τον καθορισμό των βημάτων που απαιτούνται για τη δημιουργία και την εκτέλεση της εικόνας. Κάθε εντολή σε ένα Dockerfile παρουσιάζεται σε μία γραμμή και δημιουργεί ένα επίπεδο (layer) στην εικόνα (Σχήμα 3.5). Το επίπεδο που βρίσκεται στην κορυφή χρησιμοποιείται από το τρέχον container και μπορεί να εκτελέσει λειτουργίες εγγραφής. Ενώ τα υπόλοιπα επίπεδα της εικόνας είναι μόνο για ανάγνωση.




Σχήμα 3.5: Κατασκευή εικόνας(image) μέσω των επιπέδων Dockerfile.

Όταν αλλάζει το Dockerfile και δημιουργείται ξανά η εικόνα, μόνο τα επίπεδα που έχουν αλλάξει ξανά δημιουργούνται. Αυτή η διαδικασία είναι αυτό που κάνει τις εικόνες τόσο μικρές σε μέγεθος και γρήγορες, σε σύγκριση με άλλες τεχνολογίες εικονικοποίησης (virtualization).

- Container

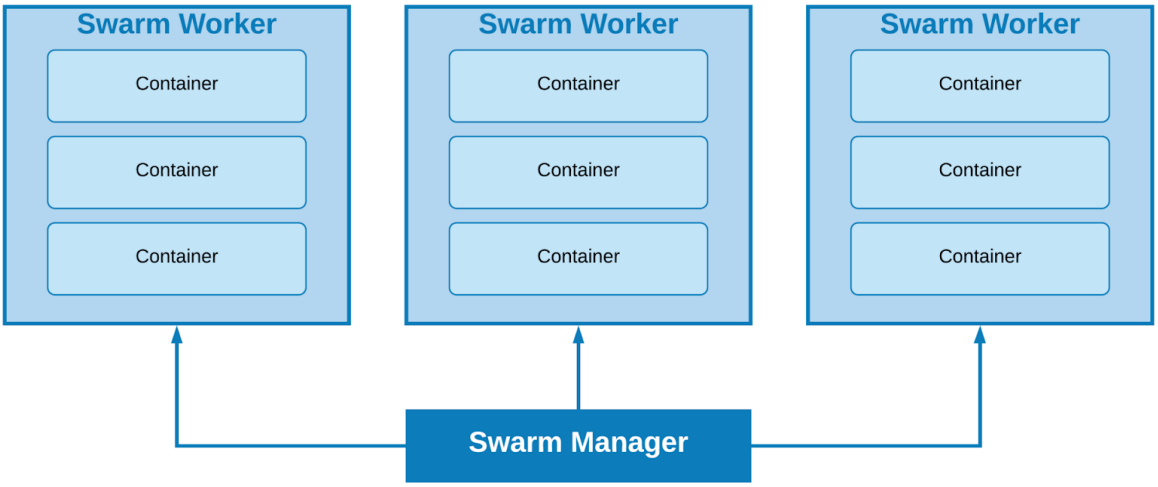
Είναι ένα τρέχον παράδειγμα μιας εικόνας. Χρησιμοποιώντας το Docker API ή CLI μπορεί κανείς να δημιουργήσει, να ξεκινήσει, να σταματήσει ή να διαγράψει ένα container. Ένα container μπορεί να συνδεθεί σε ένα ή περισσότερα δίκτυα, να αποκτήσει σύνδεση με κάποιο αποθηκευτικό χώρο του συστήματος που φιλοξενείται ή ακόμα και να δημιουργηθεί μια νέα εικόνα με βάση την τρέχουσα κατάστασή του.

Ένα container είναι σχετικά καλά απομονωμένο από άλλα container και το κεντρικό υπολογιστή. Ένα container ορίζεται από την εικόνα του καθώς και από τυχόν αρχεία

ρύθμισης παραμέτρων προγράμματος (configuration files) που επισυνάπτονται σε αυτό κατά τη δημιουργία ή εκκίνηση του. Όταν αφαιρείται ένα container, τυχόν αλλαγές στην κατάστασή του που δεν αποθηκεύονται σε μόνιμο χώρο αποθήκευσης και έτσι εξαφανίζονται.

- Υπηρεσίες (services)

Οι υπηρεσίες επιτρέπουν την ανάπτυξη των container σε πολλούς daemon, όπου όλοι μαζί μπορούν να λειτουργήσουν ως ένα σμήνος (swarm) με πολλούς διαχειριστές (manager) και εργάτες (workers). Κάθε μέλος του σμήνους (swarm) είναι ένας daemon και όλοι οι daemons επικοινωνούν χρησιμοποιώντας το Docker API. Η υπηρεσία επιτρέπει να οριστεί η επιθυμητή κατάσταση, όπως ο αριθμός των αντιγράφων της υπηρεσίας, τα οποία πρέπει να είναι διαθέσιμα ανά πάσα στιγμή. Προεπιλεγμένα, η υπηρεσία έχει οριστεί ωστέ να είναι ισορροπημένη με όλους τους κόμβους εργαζομένων. Για τον καταναλωτή, η υπηρεσία Docker φαίνεται να είναι μία μόνο εφαρμογή. Η Μηχανή Docker υποστηρίζει τη λειτουργία swarm στην έκδοση Docker 1.12 και άνω.




Σχήμα 3.6: Λειτουργία των υπηρεσιών. Μοντέλο Docker Swarm.

## 3.7. Πλεονεκτήματα Docker

Το Docker έχει γίνει αρκετά δημοφιλή τα τελευταία χρόνια, καθώς παρουσιάζει αρκετά πλεονεκτήματα [55] :

- Σταθερό και απομονωμένο περιβάλλον: τα container προσφέρουν απομονωμένα περιβάλλοντα για την ανάπτυξη εφαρμογών. Ανεξάρτητα από το που έχει αναπτυχθεί η εφαρμογή, τα πάντα παραμένουν σταθερά με αποτέλεσμα να υπάρχει μεγαλύτερη παραγωγικότητα: λιγότερο χρόνος για τον εντοπισμό σφαλμάτων

(debugging) και περισσότερος χρόνος για το λανσάρισμα νέων χαρακτηριστικών και λειτουργιών στους χρήστες.

- Φορητότητα: Μόλις δοκιμαστεί η λειτουργία της εφαρμογής στο container, μπορεί να αναπτυχθεί σε οποιοδήποτε άλλο σύστημα όπου εκτελείται το Docker έχοντας την ίδια ακριβώς απόδοση. Σε αυτό έχει συμβάλει πάρα πολύ και η δημοτικότητα τους. Πολλοί κορυφαίοι πάροχοι υπηρεσιών cloud έχουν υιοθετήσει τη τεχνική των docker containers. Επιπλέον έχουν αναπτυχθεί πολύ χρήσιμα εργαλεία διαχείρισης container, όπως το Kubernetes [56] , που ενισχύουν τη φορητότητα τους.

- Δυνατότητες επέκτασης (scaling): η δημιουργία νέων container είναι πολύ γρήγορη και εύκολη όταν απαιτείται. Επιπλέον έχουν αναπτυχθεί πολύ χρήσιμα εργαλεία για τη διαχείριση των containers, όπως αναφέρθηκε το Kubernetes.

- Καλύτερη απόδοση: Το γεγονός ότι τα container δε χρησιμοποιούν ένα ολοκληρωμένο λειτουργικό σύστημα, σε αντίθεση με τις εικονικές μηχανές, σημαίνει ότι τα container έχουν λιγότερες απαιτήσεις σε υπολογιστικούς πόρους σε σχέση με τις εικονικές μηχανές. Είναι πιο γρήγορα στη δημιουργία και στην εκκίνηση τους.

## 3.8. Εγκατάσταση Docker

Η εγκατάσταση του Docker έγινε σε λειτουργικό σύστημα της έκδοσης Linux Ubuntu: 20.04. Για την εγκατάσταση του Docker Engine απαιτείται η έκδοση 64-bit των Ubuntu.

Αρχικά πρέπει να βεβαιωθούμε ότι δεν υπάρχει κάποιο εγκατεστημένο πακέτο docker στο σύστημα γι αυτό χρησιμοποιείται η εντολή:

*sudo apt-get remove docker docker-engine docker.io containerd runc*

Στη συνέχεια και ενώ έχουμε διαπιστώσει ότι δεν υπάρχει οποιοδήποτε πακέτο docker στο σύστημα μας μπορούμε να εγκαταστήσουμε το docker με τον πιο απλό τρόπο με τη βοήθεια ενός αρχείου script ακολουθώντας τις εντολές:

*curl -fsSL https://get.docker.com -o get-docker.sh*
*sudo sh get-docker.sh*

Για πιο εύκολη χρήση του docker χωρίς να απαιτούνται δικαιώματα root user υπάρχει η επιλογή προσθήκης του ονόματος χρήστη στο docker group

*sudo usermod -aG docker your-user*

Περισσότερες πληροφορίες υπάρχουν στην επίσημη ιστοσελίδα με τις οδηγίες εγκατάστασης.

# Κεφάλαιο 4

# Δημιουργία Εικόνας

Σκοπός του πρώτου μέρους της πειραματικής διαδικασίας είναι η δημιουργία μιας εικόνας που θα υλοποιεί εφαρμογές του DTN και NDN-DTN με τη χρήση Docker Containers. Οι προσπάθειες μας επικεντρώθηκαν στην ελαχιστοποίηση του μεγέθους της ήδη υπάρχουσας εικόνας, το μέγεθος της οποίας είναι 3.01 GB αρκετά μεγάλο και καθόλου σύνηθες. Κύριο μέλημα μας είναι η κατασκευή μιας εικόνας που θα εξασφαλίζει ασφάλεια, αποδοτικότητα και ευέλικτη συντήρηση των εφαρμογών. Σ' αυτό το κεφάλαιο παρουσιάζονται αναλυτικά οι τεχνικές που αξιοποιήσαμε για τη κατασκευή ενός Dockerfile που θα μας εξασφαλίζει τη δημιουργία μιας εικόνας με τα χαρακτηριστικά που αναφέραμε, κατάλληλης για την εκτέλεση των πειραμάτων.

## 4.1. Πλεονεκτήματα ελαχιστοποίησης του μεγέθους της Εικόνας

Όπως παρουσιάστηκε στην προηγούμενη ενότητα η λειτουργία των Docker Containers, δεν υπάρχει αμφιβολία ότι η χρήση τους διευκολύνει την ανάπτυξη πολλών εφαρμογών απομονωμένων μεταξύ τους σε ένα αυτόνομο περιβάλλον. Το μέγεθος του Container επηρεάζει τον κύκλο ζωής της εφαρμογής σε πολλά επίπεδα. Αυτό φυσικά εξαρτάται από τη μείωση του μεγέθους της εικόνας (image) που χρησιμοποιούμε για τη δημιουργία Container. Ελαχιστοποιώντας το μέγεθος τους επιτυγχάνουμε καλύτερη ασφάλεια, απόδοση, αποδοτικότητα και συντήρηση της εφαρμογής σε ένα Container [57].

### Ασφάλεια

Η χρήση μιας εικόνας μικρότερης χωρητικότητας υποδηλώνει τη χρήση λιγότερων βιβλιοθηκών μειώνοντας τα πεδία στα οποία μπορεί να επιτεθεί κανείς στο Container. Επιπλέον, όταν δημιουργούμε εικόνες χρησιμοποιώντας αυτήν την προσέγγιση, το Container είναι πιο διαφανές στο τι περιλαμβάνει, άρα θεωρείται πιο ασφαλές στη χρήση του.

### Απόδοση και αποδοτικότητα

Τα Container μικρότερης χωρητικότητας καθιστούν τη φορητότητα πολύ πιο εύκολη και γρήγορη. Βελτιώνεται η απόδοση της κατασκευής και ανάπτυξης των Container, καθώς το μέγεθος της εικόνας που χρειάζεται να ληφθεί από ένα μητρώο (π.χ Docker Hub) για τη δημιουργία Container (pull image process) είναι το ελάχιστο δυνατό. Επιπλέον, εκμεταλλεύονται αποδοτικά τους διαθέσιμους υπολογιστικούς πόρους, αφού το μέγεθος που καταλαμβάνουν στο δίσκο και στη μνήμη είναι το ελάχιστο. Αυτό, βέβαια, εξαρτάται και από το αν οι εικόνες των διάφορων Container μοιράζονται αρκετά επίπεδα μεταξύ τους (τα επίπεδα καθορίζονται στο Dockerfile) και το μέγεθος των κοινών επιπέδων [58].

**Συντήρηση**

Δεδομένου ότι υπάρχουν λιγότερες βιβλιοθήκες εγκατεστημένες και dependencies απλοποιείται η διαχείριση των Container, η ενημέρωση τους σε σχέση με τις εκάστοτε αλλαγές του λειτουργικού συστήματος, κ.α.

## 4.2. Τεχνικές δημιουργίας αποδοτικής εικόνας

Μερικές από τις τεχνικές ελαχιστοποίησης του μεγέθους της εικόνας και οι οποίες θα εξεταστούν στις παρακάτω ενότητες είναι: η μείωση των εργαλείων εγκατάστασης (όπως debugging tools), η επιλογή βασικής εικόνας (base image) με το μικρότερο δυνατό μέγεθος, η ελαχιστοποίηση των επιπέδων (layers) του αρχείου Dockerfile συγχωνεύοντας εντολές, η κατασκευή εικόνας με τη χρήση της μεθόδου multi-stage [59]. Να επισημάνουμε ότι οι μέθοδοι αυτοί χρησιμοποιούνται κλιμακωτά, δηλαδή κάθε νέα μέθοδος συμπληρώνει τη προηγούμενη για να επιτευχθεί το επιθυμητό αποτέλεσμα.

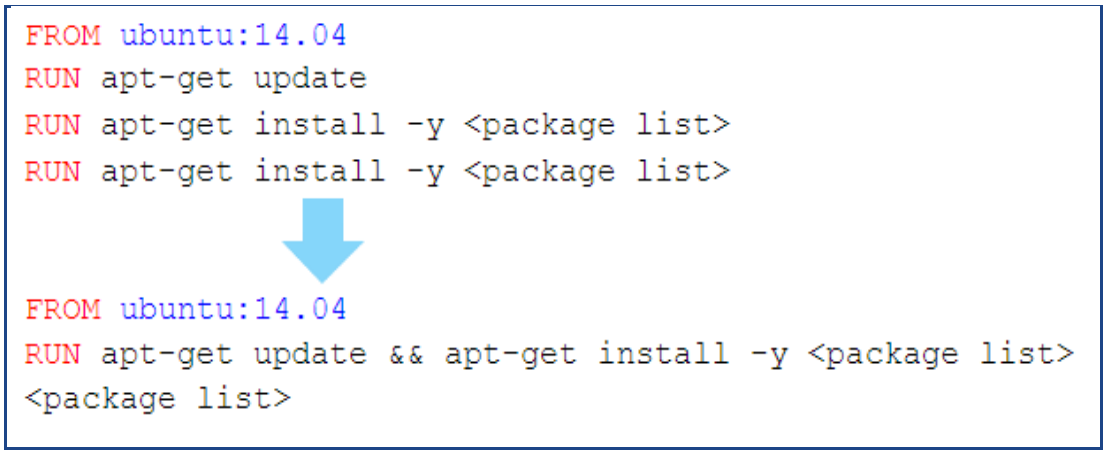
### 4.2.1. Μείωση των εργαλείων εγκατάστασης

Πολλές φορές παρατηρούμε ότι οι προγραμματιστές χρησιμοποιούν εργαλεία εντοπισμού και διόρθωσης σφαλμάτων (debugging tools), προγράμματα για τη διόρθωση κειμένων (text editors), καθώς και άλλα εργαλεία που μπορεί να μην είναι πάντα χρήσιμα, εκτός και αν η εφαρμογή εξαρτάται άμεσα από αυτά. Κατά συνέπεια, διαγράφηκαν από το Dockerfile όσα εργαλεία δεν ήταν απαραίτητα για την εγκατάσταση των βιβλιοθηκών IBR-DTN [60], NFD [61] και NDN-Cxx [62]. Πράγματι, μετά από δοκιμές διαπιστώσαμε ότι τα παρακάτω εργαλεία δεν εμπόδισαν την εγκατάσταση των βιβλιοθηκών και γι'αυτό διαγράφηκαν: cdbs,debhelper, autotools-dev, libssl-dev zlib1g-dev, libcurl4-openssl-dev, vim, doxygen, graphviz, python-sphinx.

### 4.2.2. Μείωση των επιπέδων του Dockerfile

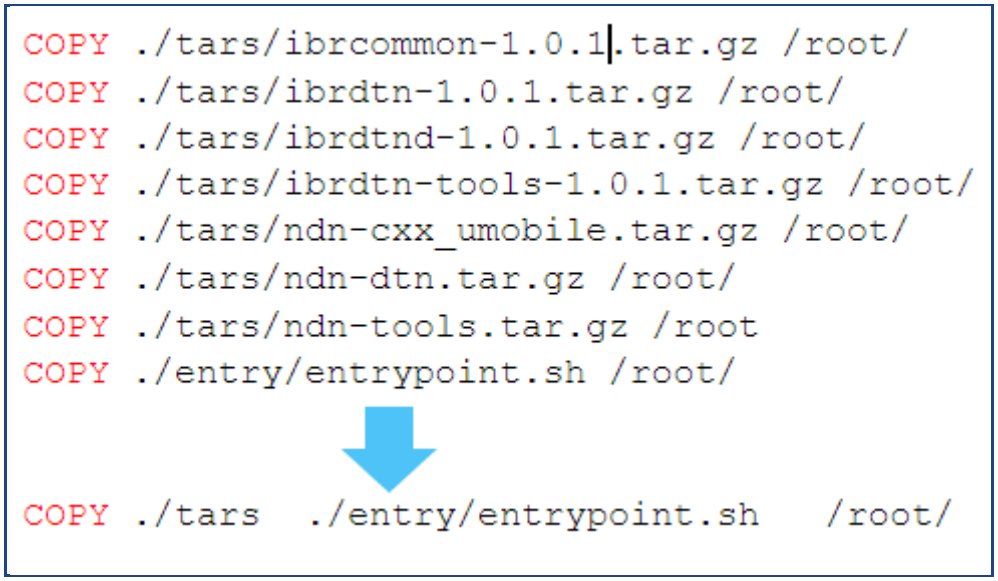
Ένας ακόμα παράγοντας που μπορεί να αυξάνει το μέγεθος της εικόνας είναι το πλήθος των επιπέδων που διαμορφώνονται κατά τη δημιουργία του Dockerfile. Κάθε εντολή RUN, COPY και ADD στο Dockerfile αποτελεί ένα επίπεδο. Οι υπόλοιπες εντολές του Dockerfile δε προκαλούσαν αύξηση του μεγέθους της εικόνας. Με αυτά τα κριτήρια εξετάσαμε ποια επίπεδα μπορούσαμε να συγχωνεύσουμε.

Αρχικά ελέγξαμε όλες τις εντολές RUN του Dockerfile. Διαπιστώσαμε ότι κάποιες από αυτές αφορούσαν την ενημέρωση του συστήματος και την εγκατάσταση των βιβλιοθηκών και μπορούσαμε να τις γράψουμε σε ένα επίπεδο όπως φαίνεται στο Σχήμα 4.1.




Σχήμα 4.1 : Συγχώνευση εντολών RUN του Dockerfile.

Στη συνέχεια ελέγξαμε τις εντολές COPY του Dockerfile. Παρατηρήσαμε ότι οι περισσότερες εντολές COPY αντέγραφαν αρχεία που βρισκόντουσαν στον ίδιο φάκελο και τα αποθήκευαν αντίστοιχα στον ίδιο προορισμό ( */root* ), άρα μπορούσαν να συγχωνευθούν (Σχήμα 4.2).




Σχήμα 4.2: Συγχώνευση εντολών COPY του Dockerfile.

Με τις δύο πρώτες μεθόδους βελτιστοποίησης η εικόνα απέκτησε μέγεθος 2.89GB, δηλαδή μείωση της τάξεως ~3,98%. Αυτό το ποσοστό δεν ήταν επαρκές και δεν έλυνε το πρόβλημα της χωρητικότητας. Αυτό είχε ως επακόλουθο να διερευνήσουμε περαιτέρω τις μεθόδους ελαχιστοποίησης του μεγέθους της εικόνας.

### 4.2.3. Multi-Stage Builds

Η μέθοδος αυτή μειώνει δραστικά το μέγεθος της εικόνας. Επειδή μία εικόνα δημιουργείται κατά το τελευταίο στάδιο της διαδικασίας κατασκευής της, μπορούμε να

ελαχιστοποιήσουμε τα επίπεδα της εκμεταλλεύομενοι σωστά τη μνήμη cache του docker. Για παράδειγμα, μπορούμε να ταξινομήσουμε τα επίπεδα της εικόνας (layers) από αυτά που πιστεύουμε ότι αλλάζουν σπάνια (για να εξασφαλίσουμε ότι επαναχρησιμοποιούμε τα δεδομένα που αποθηκεύονται στη cache) προς αυτά που πιστεύουμε ότι αλλάζουν πιο συχνά.

Στη μέθοδο πολλών σταδίων κατασκευής (multi-stage) μπορούμε να χρησιμοποιήσουμε πολλές εντολές *FROM*. Κάθε εντολή *FROM* ξεκινάει ένα νέο build stage της εικόνας και μπορεί να χρησιμοποιεί διαφορετική εικόνα ως βάση (base image). Επιλεκτικά μπορούμε να αντιγράψουμε όσα στοιχεία θεωρούμε χρήσιμα για την τελική εικόνα από το ένα stage στο επόμενο. Αφού η τελική εικόνα διαμορφώνεται από το τελευταίο stage, τότε όλα τα προηγούμενα stages διαγράφονται αυτόματα.

Εξ ορισμού τα stages δε διαθέτουν κάποια ονόματα και μπορούμε να αναφερθούμε σε αυτά χρησιμοποιώντας αύξοντα αριθμό για κάθε εντολή FROM ξεκινώντας από το 0. Ωστόσο, μπορούμε να ονομάσουμε κάποιο stage. Το μόνο που έχουμε να κάνουμε είναι να προσθέσουμε μια επέκταση *AS <όνομα stage>* στην εντολή *FROM* (Σχήμα 4.3). Επιπλέον έχουμε τη δυνατότητα να κατασκευάσουμε οποιοδήποτε stage επιθυμούμε χρησιμοποιώντας την επιλογή target της εντολής docker build. Για παράδειγμα: *docker build --target builder -t test . .*

```
from ubuntu:14.04 AS builder

#install the dependencies of build proces
RUN apt-get update && apt-get install -y
pkg-config libnl-3-dev libnl-genl-3-dev l
libnl-nf-3-dev libnl-cli-3-dev libssl-dev
automake autoconf pkg-config libtool libc
\
  && apt-get clean \
  && rm -rf /var/lib/apt/lists/*

          . . . .

from ubuntu:14.04

RUN apt-get update && apt-get install -y
```

Σχήμα 4.3: Multi-stage Dockerfile.

Σύμφωνα με τα παραπάνω, θα υλοποιήσουμε δύο stages στο συγκεκριμένο Dockerfile. Ονομάζουμε το πρώτο stage builder για να μπορεί να αναφέρεται σε αυτό το δεύτερο stage. Χρησιμοποιούμε ως εικόνα βάσης την έκδοση ubuntu:14.04 για να εξασφαλίσουμε τη συμβατότητα με τις εκδόσεις των εργαλείων Σχήμα 4.3. Συμπεριλαμβάνουμε όλες τις βιβλιοθήκες που θεωρήσαμε ότι είναι απαραίτητες, σύμφωνα με τα αποτελέσματα των προηγούμενων δοκιμών, και φυσικά εφαρμόζουμε τη μέθοδο συγχώνευσης των επιπέδων.

Στο δεύτερο stage αρχικά χρησιμοποιούμε ως εικόνα βάσης την έκδοση ubuntu: 14.04, δε θα δώσουμε κάποιο όνομα στο stage καθώς θα είναι το τελικό. Αντιγράφουμε στο δεύτερο stage τα εκτελέσιμα αρχεία (binaries) των υλοποιήσεων NDN και DTN από το stage: builder με τη χρήση της εντολής *COPY --from= builder <διεύθυνση πηγής> < διεύθυνση προορισμού>* (Σχήμα 4.4). Η διεύθυνση πηγής προσδιορίζει το που είναι αποθηκευμένα τα εκτελέσιμα αρχεία (binaries) στο πρώτο stage, ενώ η διεύθυνση προορισμού προσδιορίζει το που αποθηκεύονται τα εκτελέσιμα αρχεία (binaries) στο δεύτερο stage.

```
#copy binaries
#COPY --from=builder /root/ndn-cxx_umobile /ndn-cxx_umobile
COPY --from=builder /root/entrypoint.sh /root/entrypoint.sh
COPY --from=builder /usr/local/bin /usr/local/bin
COPY --from=builder /usr/local/sbin /usr/local/sbin
```

Σχήμα 4.4: Αντιγραφή των binaries από το πρώτο stage του Dockerfile στο δεύτερο.

Μπορεί να εξασφαλίσαμε τη δημιουργία των binaries με τη χρήση του πρώτου stage στο Dockerfile, αλλά διαπιστώνουμε στη συνέχεια ότι είναι δυναμικά εκτελέσιμα. Αυτό σημαίνει ότι η εκτέλεση τους εξαρτάται από κοινόχρηστες βιβλιοθήκες, οι κώδικες των οποίων είναι αποθηκευμένοι σε .so αρχεία. Για να βρούμε τα αρχεία που απουσιάζουν, ώστε να τα προσθέσουμε στη συνέχεια στο δεύτερο stage, χρησιμοποιούμε δύο βοηθητικές εικόνες.

Από το πρώτο stage και με τη χρήση της εντολής *docker build -t test:first_stage --target builder .* δημιουργήσαμε μία εικόνα ονόματι test:first_stage. Την εικόνα αυτή θα τη χρησιμοποιήσουμε παρακάτω για τη δημιουργία ενός Container (container_1) στο οποίο θα μπορούμε να εκτελούμε επιτυχώς όλα τα binaries. Στη συνέχεια από το δεύτερο stage δημιουργούμε την εικόνα ονόματι test:2. Με αυτή θα δημιουργήσουμε ένα Container (container_2) για την ανίχνευση των αρχείων που απουσιάζουν για να επιτευχθεί η σωστή εκτέλεση κάθε binary.

Χρησιμοποιήσαμε τις παρακάτω εντολές για τη δημιουργία και εκτέλεση των δύο container σε δύο διαφορετικούς terminal:

- *docker run --rm -dit --entrypoint bash --name container_1 -v "${PWD} /my_confs: /confs" test:2*
- *docker exec -it container_1 bin/bash*
- *docker run --rm -dit --entrypoint bash --name container_2 -v "${PWD} /my_confs: /confs" test:first_stage*
- *docker exec -it container_2 bin/bash*

Στο container_1 εκτελούμε την εντολή ldd για κάθε δυναμικά εκτελέσιμο binary του DTN και NDN. Η εντολή *ldd < file name>* τυπώνει τις βιβλιοθήκες (shared libraries) που φορτώνονται κατά την εκκίνηση ενός αρχείου, στην προκειμένη περίπτωση ενός εκτελέσιμου αρχείου (binaries). Όταν κάποια από τις βιβλιοθήκες απουσιάζει εμφανίζει και ανάλογο μήνυμα που την υποδεικνύει (Σχήμα 4.5). Αφού ανακαλύψουμε ποια αρχεία απουσιάζουν από το container_1 για την εκτέλεση των εκτελέσιμων αρχείων (binaries), θα

εκτελέσουμε το container_2 για να βρούμε που είναι αποθηκευμένα και να τα αντιγράψουμε στο δεύτερο stage του Dockerfile. Με αυτό τον τρόπο εξετάστηκε προσεκτικά κάθε δυναμικά εκτελέσιμο αρχείο (dynamic binary) των υλοποιήσεων του DTN και NDN για να προστεθούν τα απαραίτητα αρχεία.

```
root@d17a4248712b:~/ibrdtnd-1.0.1/src# ldd dtnd
        linux-vdso.so.1 (0x00007ffe24953000)
        libdaemon.so.0 => /usr/lib/x86_64-linux-gnu/libdaemon.so.0 (0x00007ff545f98000)
        libvmime.so.0 => not found
        libibrdtn-1.0.so.1 => not found
        libibrcommon-1.0.so.1 => not found
        libpthread.so.0 => /lib/x86_64-linux-gnu/libpthread.so.0 (0x00007ff545d79000)
        libcrypto.so.1.0.0 => not found
        libstdc++.so.6 => /usr/lib/x86_64-linux-gnu/libstdc++.so.6 (0x00007ff5459f0000)
        libm.so.6 => /lib/x86_64-linux-gnu/libm.so.6 (0x00007ff545652000)
        libgcc_s.so.1 => /lib/x86_64-linux-gnu/libgcc_s.so.1 (0x00007ff54543a000)
        libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007ff545049000)
        /lib64/ld-linux-x86-64.so.2 (0x00007ff54619f000)
```

Σχήμα 4.5: Εφαρμογή εντολής ldd στο δυναμικά εκτελέσιμο αρχείο (static binary) dtnd.

Κάθε κοινόχρηστη βιβλιοθήκη (shared library) αποτελείται από ένα soname αρχείο όπως φαίνεται και στο Σχήμα 4.5. Κάθε soname [63] αρχείο αποτελεί μία αναφορά σε άλλο αρχείο ή διεύθυνση (symbolic link), συγκεκριμένα στο real name αρχείο της κοινόχρηστης βιβλιοθήκης. Το real-name αρχείο περιλαμβάνει το πραγματικό κώδικα της βιβλιοθήκης, τον οποίο και θέλουμε να αντιγράψουμε. Έτσι, χρησιμοποιούμε την εντολή *ls -l <directory soname>* για να εξάγουμε το όνομα του αρχείου real-name στο οποίο αναφέρεται το symbolic link και όπως αποφαίνεται θα βρίσκεται στην ίδια διεύθυνση.

Αφού ολοκληρώσουμε την παραπάνω διαδικασία για κάθε binary, θα προσθέσουμε όλα τα αρχεία real-name στο δεύτερο stage χρησιμοποιώντας την εντολή *COPY --from= builder <διευθύνσεις αρχείων real name>* (Σχήμα 4.6). Αυτό όμως δεν αρκεί για την εγκατάσταση και τη σωστή λειτουργία των βιβλιοθηκών. Θα πρέπει, λοιπόν, να εκτελέσουμε την εντολή ldconfig η οποία εξετάζει την ύπαρξη των αρχείων real-name και στη συνέχεια δημιουργεί τα symbolic link, όπως επίσης και το αρχείο μνήμης /etc/ld.so.cache.

```
COPY --from=builder /usr/lib/x86_64-linux-gnu/libnl-route-3.so.200.16.1 /
0.0.19 /usr/lib/x86_64-linux-gnu/libgnutls.so.26.22.6 /usr/lib/x86_64-lin
x86_64-linux-gnu/libidn.so.11.6.11 /usr/lib/x86_64-linux-gnu/libk5crypto.
libkrb5.so.3.3 /usr/lib/x86_64-linux-gnu/libarchive.so.13.1.2 /usr/lib/x8
x86_64-linux-gnu/libkrb5support.so.0.1  /usr/lib/x86_64-linux-gnu/libxml2
libicuuc.so.52.1 /usr/lib/x86_64-linux-gnu/libicui18n.so.52.1 /usr/lib/x8
x86_64-linux-gnu/libicudata.so.52.1 /usr/lib/x86_64-linux-gnu/
RUN sudo ldconfig

COPY --from=builder   /lib/x86_64-linux-gnu/libnl-3.so.200.16.1 /lib/x86_
x86_64-linux-gnu/libssl.so.1.0.0 /lib/x86_64-linux-gnu/liblzo2.so.2.0.0 /
RUN sudo ldconfig
```

Σχήμα 4.6: Αντιγραφή των shared libraries στο δεύτερο stage του Dockerfile.

Η διαδικασία αυτή ήταν απαραίτητο να γίνει καθώς τα εκτελέσιμα αρχεία (binaries), όπως αναφέραμε, είναι δυναμικά εκτελέσιμα αρχεία και όχι σταθερά, πράγμα που καθιστά την εκτέλεση τους εξαρτώμενη από άλλους παράγοντες. Η τελική εικόνα που δημιουργείται έχει μέγεθος 542MB. Επιτεύχθηκε ραγδαία μείωση της τάξεως ~81.99%, όπως ακριβώς περιμέναμε από αυτή τη μέθοδο.

### 4.2.4. Κατάλληλη επιλογή εικόνας ως βάση (base image)

Η χρήση της κατάλληλης εικόνας ως βάση μπορεί να επιφέρει μεγάλες αλλαγές στο μέγεθος της. Έτσι επιλέξαμε να εξετάσουμε την εικόνα με έκδοση ubuntu:18.04 ως βάση για το δεύτερο stage του Dockerfile, το οποίο χτίζει και την τελική εικόνα. Η έκδοση αυτή αποτελεί την πιο ελαφριά έκδοση εικόνας της Ubuntu αυτήν τη στιγμή με αρχικό μέγεθος 63.3MB έναντι της ubuntu:14.04 που χρησιμοποιούμε στις προηγούμενες δοκιμές με αρχικό μέγεθος 197 MB. Αυτή η αλλαγή προκαλεί μείωση του τελικού μεγέθους της εικόνας, κατά ~85.68%, στα 431 MB. Η επιλογή μιας εικόνας της οικογένειας του λειτουργικού συστήματος Ubuntu έγκειται στο γεγονός ότι η αρχική εκτέλεση γινόταν σε λειτουργικό ubuntu και διασφαλίζει οπωσδήποτε τη συμβατότητα με το αρχικό μοντέλο.

Στη συνέχεια των δοκιμών, όσον αφορά την επιλογή εικόνας ως βάση, δοκιμάσαμε την εικόνα Alpine. Το αρχικό μέγεθος αυτής δεν ξεπερνά τα 6MB. Διαπιστώσαμε ότι η λειτουργική έκδοση Alpine δεν ενδείκνυται για τις υλοποιήσεις DTN και NDN. Ωστόσο, έγινε μία απόπειρα εγκατάστασης των απαραίτητων βιβλιοθηκών στο δεύτερο stage, αλλά αντιμετωπίσαμε πρόβλημα με την εντολή ldconfig η οποία δε διατίθεται από το λειτουργικό Alpine λόγω διαφορετικής δομής του λειτουργικού συστήματος. Το τελικό Dockerfile για την εικόνα test:2 παρουσιάζεται στο Σχήμα 4.7-4.8.

```
FROM ubuntu:14.04 AS builder

#install the dependencies of build process
RUN apt-get update && apt-get install -y  devscripts\
build-essential pkg-config libnl-3-dev libnl-genl-3-dev\
libnl-route-3-dev libnl-nf-3-dev libnl-cli-3-dev libssl-dev \
libsqlite3-dev openssl libdaemon-dev libvmime-dev libarchive-dev \
automake autoconf pkg-config libtool libcppunit-dev libcrypto++-dev \
libboost-all-dev  pkg-config libpcap-dev curl \
  && apt-get clean \
  && rm -rf /var/lib/apt/lists/*

# copy all files from tar folder to /root
COPY ./tars  ./entry/entrypoint.sh   /root/
COPY ./confs/nfd_template.conf /usr/local/etc/ndn/

#untar the files and install the components
RUN cd /root && \
    tar -xvf ibrcommon-1.0.1.tar.gz && \
    cd ibrcommon-1.0.1 && \
    ./configure --with-openssl && \
    make && \
    make install && \
    ldconfig

RUN cd /root && \
    tar -xvf ibrdtn-1.0.1.tar.gz && \
    cd ibrdtn-1.0.1 && \
    ./configure && \
    make && \
    make install && \
    ldconfig

RUN cd /root && \
    tar -xvf ibrdtnd-1.0.1.tar.gz && \
    cd ibrdtnd-1.0.1 && \
    ./configure --with-curl && \
    make && \
    make install && \
    ldconfig
```

Σχήμα 4.7: Πρώτο μέρος του Dockerfile της εικόνας test:2.

```
RUN cd /root && \
    tar -xvf ibrdtn-tools-1.0.1.tar.gz && \
    cd ibrdtn-tools-1.0.1 && \
    ./configure && \
    make && \
    make install && \
    ldconfig

RUN cd /root && \
    tar -xvf ndn-cxx_umobile.tar.gz && \
    cd ndn-cxx_umobile && \
    ./waf configure && \
    ./waf && \
    ./waf install  && \
    ldconfig


RUN cd /root && \
    tar -xvf ndn-dtn.tar.gz && \
    cd ./ndn-dtn && \
    mkdir websocketpp && \
    curl -L https://github.com/zaphoyd/websocketpp/archive/0.5
    tar zxf websocket.tar.gz -C websocketpp/ --strip 1 && \
    ./waf configure && \
    ./waf && \
    ./waf install && \
    ldconfig && \
    rm -rf /root/ndn-dtn/*.tar.gz


RUN cd /root && \
    tar -xvf ndn-tools.tar.gz && \
    cd ndn-tools-ndn-tools-0.3 && \
    ./waf configure && \
    ./waf && \
    ./waf install && \
    ldconfig


RUN rm -rf /root/*.tar.gz
FROM ubuntu:18.04

RUN apt-get update && apt-get install -y sudo


#import all dependencies to run the binaries

#copy binaries
COPY --from=builder /root/entrypoint.sh /root/entrypoint.sh
COPY --from=builder /usr/local/bin /usr/local/bin
COPY --from=builder /usr/local/sbin /usr/local/sbin

#copy libs ibrcommon, ibrdtn
COPY --from=builder /usr/local/lib /usr/local/lib
RUN sudo ldconfig

#other libs for dtn and ndn
COPY --from=builder /usr/lib/libdaemon.so.0.5.0
/usr/lib/libvmime.so.0.0.0 /usr/lib/libcrypto++.so.9.0.0
/usr/lib/libgsasl.so.7.9.6 /usr/lib/
RUN sudo ldconfig


COPY --from=builder /usr/lib/x86_64-linux-gnu/libnl-route-3.so.200.16.1
/usr/lib/x86_64-linux-gnu/libntlm.so.0.0.19 /usr/lib/x86_64-linux-gnu/libgnutls.so.26.22.6
/usr/lib/x86_64-linux-gnu/libgssapi_krb5.so.2.2 /usr/lib/x86_64-linux-gnu/libidn.so.11.6.11
/usr/lib/x86_64-linux-gnu/libk5crypto.so.3.1 /usr/lib/x86_64-linux-gnu/libkrb5.so.3.3
/usr/lib/x86_64-linux-gnu/libarchive.so.13.1.2 /usr/lib/x86_64-linux-gnu/libnettle.so.4.7
/usr/lib/x86_64-linux-gnu/libkrb5support.so.0.1 /usr/lib/x86_64-linux-gnu/libboost_*.so.1.54.0
/usr/lib/x86_64-linux-gnu/libxml2.so.2.9.1 /usr/lib/x86_64-linux-gnu/libsqlite3.so.0.8.6
/usr/lib/x86_64-linux-gnu/libicuuc.so.52.1 /usr/lib/x86_64-linux-gnu/libicui18n.so.52.1
/usr/lib/x86_64-linux-gnu/libpcap.so.1.5.3 /usr/lib/x86_64-linux-gnu/libicudata.so.52.1
/usr/lib/x86_64-linux-gnu/
RUN sudo ldconfig


COPY --from=builder /lib/x86_64-linux-gnu/libcrypto.so.1.0.0
/lib/x86_64-linux-gnu/libgcrypt.so.11.8.2 /lib/x86_64-linux-gnu/libnl-3.so.200.16.1
/lib/x86_64-linux-gnu/libkeyutils.so.1.4 /lib/x86_64-linux-gnu/libssl.so.1.0.0
/lib/x86_64-linux-gnu/liblzo2.so.2.0.0 /lib/x86_64-linux-gnu/
RUN sudo ldconfig
#copy nfd_conf for nfd
COPY --from=builder /usr/local/etc/ndn/ /usr/local/etc/ndn/

RUN chmod a+x /root/entrypoint.sh

ENTRYPOINT ["/root/entrypoint.sh"]
```

Σχήμα 4.8: Δεύτερο μέρος του Dockerfile της εικόνας test:2.

## 4.3. Δημιουργία εικόνας Πειραμάτων

Για την εκτέλεση των πειραμάτων θα δημιουργήσουμε μια ξεχωριστή εικόνα ονόματι exp, καθώς διαπιστώσαμε ότι δεν επαρκούσε η εικόνα test:2 ώστε να τρέχουν επιτυχώς οι πηγαίοι κώδικες των εφαρμογών consumer και producer στην περίπτωση των πειραμάτων του πρωτοκόλλου NDN-DTN που θα δούμε παρακάτω. Ουσιαστικά αυτό που θα κάνουμε είναι να τροποποιήσουμε το δεύτερο stage της εικόνας test:2, το οποίο χτίζει τη τελική εικόνα. Οι αλλαγές του δεύτερου stage παρουσιάζονται αναλυτικά παρακάτω.

Αρχικά, αντιγράφουμε τα binaries από το πρώτο επίπεδο μαζί και το φάκελο ndn-cxx_umobile. Ο φάκελος ndn-cxx_umobile κρίνεται απαραίτητος για τη διαδικασία του compile των εφαρμογών producer και consumer, αφού διαθέτει όλα τα απαραίτητα build tools. Για την εγκατάσταση του ndn-cxx_umobile καθώς και για τη διαδικασία του compile απαιτείται η εγκατάσταση των παρακάτω βιβλιοθηκών: build-essential libcrypto++-dev libsqlite3-dev libboost-all-dev libssl-dev.

```
from ubuntu:14.04

RUN apt-get update && apt-get install -y sudo build-essential \
libcrypto++-dev libsqlite3-dev libboost-all-dev

RUN sudo ldconfig

#copy binaries
#COPY --from=builder /root/ndn-cxx_umobile /ndn-cxx_umobile
COPY --from=builder /root/entrypoint.sh /root/entrypoint.sh
COPY --from=builder /usr/local/bin /usr/local/bin
COPY --from=builder /usr/local/sbin /usr/local/sbin
```

Σχήμα 4.9: Τροποποιημένο Dockerfile της εικόνας exp. Εγκατάσταση βιβλιοθηκών και εκτελέσιμων αρχείων (binaries).

Όπως αναφέραμε στην υποενοτήτα 4.2.3 η εκτέλεση των δυναμικά εκτελέσιμων αρχείων (dynamic binaries) απαιτεί την εγκατάσταση ορισμένων αρχείων (soname), οπότε θα προστεθούν και αυτά.

```
#copy libs ibrcommon, ibrdtn
COPY --from=builder /usr/local/lib /usr/local/lib
RUN sudo ldconfig

#other libs for dtn and ndn
COPY --from=builder /usr/lib/libdaemon.so.0.5.0 /usr/lib/libvmime.so.0.0.0 /usr/lib/libgsasl.so.7.9.6 /usr/lib/
RUN sudo ldconfig


COPY --from=builder /usr/lib/x86_64-linux-gnu/libnl-route-3.so.200.16.1 /usr/lib/x86_64-linux-gnu/libntlm.so.
0.0.19 /usr/lib/x86_64-linux-gnu/libgnutls.so.26.22.6 /usr/lib/x86_64-linux-gnu/libgssapi_krb5.so.2.2 /usr/lib/-
x86_64-linux-gnu/libidn.so.11.6.11 /usr/lib/x86_64-linux-gnu/libk5crypto.so.3.1 /usr/lib/x86_64-linux-gnu/-
libkrb5.so.3.3 /usr/lib/x86_64-linux-gnu/libarchive.so.13.1.2 /usr/lib/x86_64-linux-gnu/libnettle.so.4.7 /usr/lib/-
x86_64-linux-gnu/libkrb5support.so.0.1  /usr/lib/x86_64-linux-gnu/libxml2.so.2.9.1  /usr/lib/x86_64-linux-gnu/-
libicuuc.so.52.1 /usr/lib/x86_64-linux-gnu/libicui18n.so.52.1 /usr/lib/x86_64-linux-gnu/libpcap.so.1.5.3 /usr/lib/-
x86_64-linux-gnu/libicudata.so.52.1 /usr/lib/x86_64-linux-gnu/
RUN sudo ldconfig


COPY --from=builder   /lib/x86_64-linux-gnu/libnl-3.so.200.16.1 /lib/x86_64-linux-gnu/libkeyutils.so.1.4 /lib/-
x86_64-linux-gnu/libssl.so.1.0.0 /lib/x86_64-linux-gnu/liblzo2.so.2.0.0 /lib/x86_64-linux-gnu/
RUN sudo ldconfig
```

Σχήμα 4.10: Τροποποιημένο Dockerfile της εικόνας exp. Εγκατάσταση soname αρχείων.

Τέλος, αντιγράφουμε τα configuration αρχεία του NFD και θέτουμε την εντολή entrypoint [64].

```
#copy nfd_conf for nfd
COPY --from=builder /usr/local/etc/ndn/ /usr/local/etc/ndn/

RUN chmod a+x /root/entrypoint.sh

ENTRYPOINT ["/root/entrypoint.sh"]
```

Σχήμα 4.11: Τροποποιημένο Dockerfile της εικόνας exp. Αντιγραφή configuration files και εντολή entrypoint.

Σε πρώτο στάδιο δοκιμάζουμε ως εικόνα βάσης του δεύτερου stage την έκδοση ubuntu:18.04. Παρατηρούμε, όμως, πως για ορισμένες βιβλιοθήκες υπάρχουν εγκατεστημένες δύο διαφορετικές εκδόσεις αυτών το οποίο όχι μόνο καταλαμβάνει επιπλέον χώρο, αλλά και εμποδίζει την εκτέλεση ορισμένων εργαλείων που έχουν κατασκευαστεί με βάση την έκδοση ubuntu:14.04. Οπότε επιλέξαμε για το δεύτερο stage του Dockerfile ως εικόνα βάσης την έκδοση ubuntu:14.04 (Σχήμα 4.9). Το συνολικό μέγεθος της εικόνας exp των πειραμάτων είναι 1.57GB.

Αναφέραμε πως αποθηκεύουμε το αρχείο ndn_cxx-umobile μέσα στην εικόνα, αυτό σημαίνει ότι κατά τη δημιουργία ενός Container τα δεδομένα αυτά θα αποθηκεύονται στο επίπεδο εγγραφής (writable) [65]. Οπότε όταν θα διαγράφουμε το Container θα διαγράφονται και αυτά. Για να το αποφύγουμε αυτό χρησιμοποιούμε τα bind mounts [66]. Έτσι αποθηκεύουμε το αποσυμπιεσμένο αρχείο ndn_cxx-umobile σε ένα σημείο στο κεντρικό υπολογιστή και αυτό τοποθετείται μέσα στο Container μέσω της επιλογής bind mount κατά τη δημιουργία του.

Ως εκ τούτου, κάθε φορά που θα θέλουμε να κάνουμε οποιαδήποτε αλλαγή στο κώδικα των εφαρμογών, είτε του consumer είτε του producer, δε θα χρειάζεται να διαγράφουμε τα Container και να τα δημιουργούμε εκ νέου. Όπως φαίνεται και στο Σχήμα:4.9 τελικά δε χρησιμοποιήθηκε το ndn_cxx-umobile στο δεύτερο επίπεδο της εικόνας exp:1 αλλά όπως θα δούμε παρακάτω στην τοπολογία των πειραμάτων χρησιμοποιούμε bind mount. Μετά από την τροποποίηση αυτή, το μέγεθος της τελικής εικόνας exp:1 που θα χρησιμοποιήσουμε στα πειράματα μειώθηκε στα 964 MB, ~67.97%.

Στο πίνακα 2.1 παρουσιάζονται συγκριτικά τα αποτελέσματα όλων των δοκιμών. Οι εικόνες που θα χρησιμοποιήσουμε στη συνέχεια των πειραμάτων είναι οι test:2 και exp:1.

| Πείραμα | I. | II. | III. | IV. | V. |
|---|---|---|---|---|---|
| Εικόνα | ubuntu:14.04 | ubuntu:14.04 | ubuntu:18.04 | ubuntu:14.04 | ubuntu:14.04 |
| Όνομα εικόνας | Test | test:2 | test:2 | exp | exp:1 |
| Μέγεθος | 2.89 GB | 542 MB | 431 MB | 1.57 GB | 964 MB |
| Μείωση | 3.98% | 81.99% | 85.68% | 47.84% | 67.97% |

Πίνακας 4.1: Συγκριτικά αποτελέσματα μεγέθους εικόνας.

# Κεφάλαιο 5

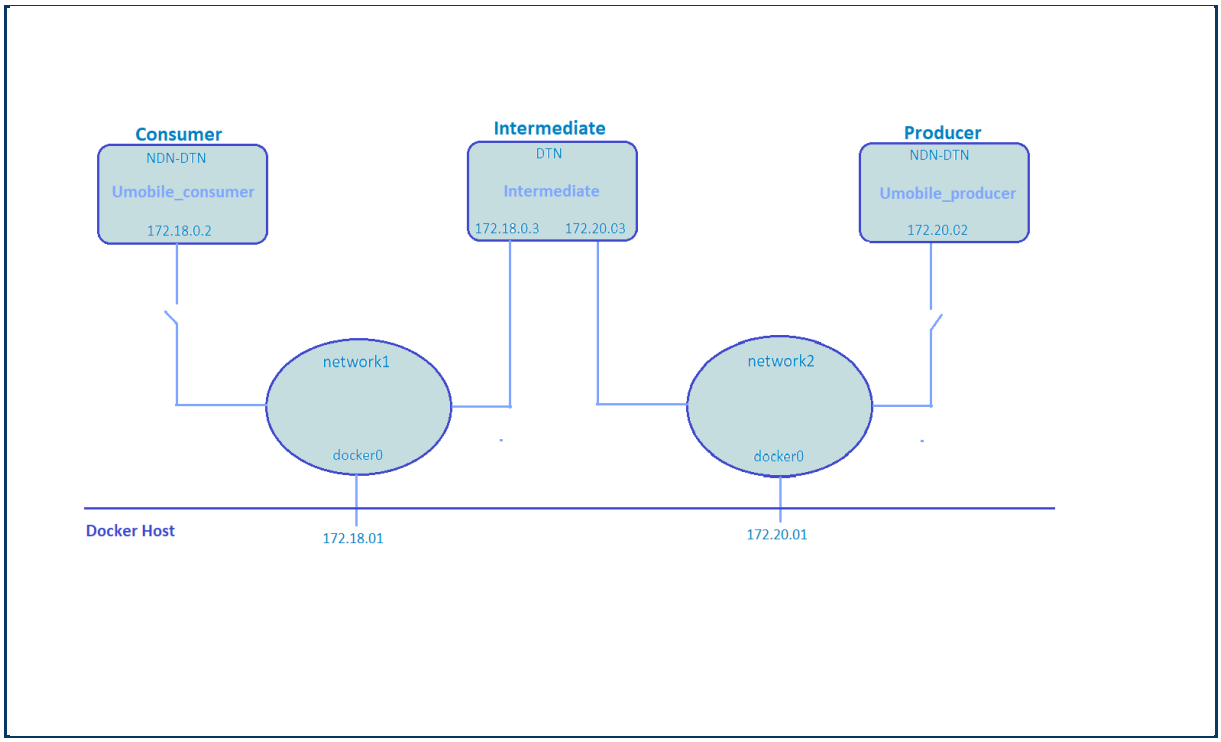
# Πειράματα

Σε αυτό το κεφάλαιο θα παρουσιάσουμε την τοπολογία του δικτύου που στήσαμε με τη χρήση των Docker Containers, τα πειράματα τα οποία πραγματοποιήσαμε καθώς και τα αποτελέσματα αυτών. Σκοπός των πειραμάτων ήταν να εξετάσουμε την απόδοση της στοίβας τόσο του DTN πρωτοκόλλου όσο και του NDN-DTN σε ένα περιβάλλον προσομοίωσης διακοπτόμενων συνδέσεων χρησιμοποιώντας το εργαλείο Docker για τη δημιουργία των Container με τη χρήση των εικόνων που δημιουργήσαμε στη προηγούμενη ενότητα. Επιπλέον, πραγματοποιήσαμε σύγκριση των αποτελεσμάτων μας με τα αποτελέσματα των πειραμάτων που διεξήχθησαν σε συσκευές IoT στο εργαστήριο ωστέ να επιβεβαιώσουμε την ακρίβεια του θεωρητικού μοντέλου [5] μέσω της προσομοίωσης του δικτύου.

Στα πειράματα αυτά υποθέσαμε ένα σενάριο μεταφοράς αισθητήριων δεδομένων (data muling scenario) από ένα απομονωμένο Container Producer σε ένα απομονωμένο Container Consumer μέσω ενός κόμβου μεταφοράς (Container Intermediate) που συνδεόταν άλλοτε στον έναν και άλλοτε στον άλλον (muling device). Τα αποτελέσματα στη παρούσα εργασία απέδειξαν με τη σειρά τους την πολύτιμη συμβολή του NDN-DTN για την ανάκτηση Δεδομένων σε περιπτώσεις διακοπτόμενων συνδέσεων και τη μείωση του μέσου χρόνου καθυστέρησης (Average Delay). Τέλος, σε ορισμένες περιπτώσεις είδαμε ότι η απόδοση του μοντέλου προσομοίωσης αντανακλά το θεωρητικό μοντέλο σε μεγαλύτερο βαθμό [5].

## 5.1. Τοπολογία Δικτύου

Η αναπαράσταση του δικτύου για την πραγματοποίηση των πειραμάτων βασίστηκε στο [5]. Η τοπολογία μας αποτελείται από ένα απομονωμένο Container NDN producer και ένα απομονωμένο Container NDN consumer. Η επικοινωνία μεταξύ τους επιτυγχάνεται μέσω ενός ενδιάμεσου DTN κόμβου (intermediate container) Σχήμα 5.1. Το container intermediate μιμείται ένα μοντέλο κινητικότητας ενός κόμβους μεταφοράς δεδομένων (mule device) μεταβαίνοντας από μία κατάσταση σε μία άλλη. Οι διαθέσιμες καταστάσεις είναι τρεις: σύνδεση στο Container consumer, αποσύνδεση, σύνδεση στο Container producer. Τα Container consumer και producer διαθέτουν μία διεπαφή DTN για να στέλνουν τα δεδομένα στο Container intermediate, η οποία βασίζεται στο DTN Bundle.




Σχήμα 5.1: Τοπολογία Δικτύου Προσομοίωσης.

Για το σχεδιασμό του δικτύου χρησιμοποιούμε ένα αρχείο bash script που περιλαμβάνει όλες τις απαραίτητες εντολές και αυτοματοποιεί τη διαδικασία. Για την εκτέλεση των πειραμάτων χρησιμοποιούμε δύο εικόνες (την υλοποίηση αυτών αναλύσαμε στις παραπάνω ενότητες). Στα πειράματα που θα εξετάσουμε την απόδοση του πρωτοκόλλου DTN χρησιμοποιούμε την εικόνα test:2 για τη δημιουργία των container, ενώ στα πειράματα που θα εξετάσουμε το πρωτόκολλο NDN-DTN χρησιμοποιούμε την εικόνα exp:1. Κάθε εικόνα εξασφαλίζει τη σωστή λειτουργία των Container και καταλαμβάνει το ελάχιστο μέγεθος, ώστε να αξιοποιεί σωστά τους διαθέσιμους πόρους σε κάθε περίπτωση.

Το πρώτο στάδιο είναι να δημιουργήσουμε τα δύο εικονικά δίκτυα (network1, network2) που στη συνέχεια συνδέουμε σε αυτά τα δυο container consumer και producer. Η διεύθυνση υποδικτύου (subnet) του network1 όπως φαίνεται στο Σχήμα 5.2 είναι 172.18.0.0/16 και η διεύθυνση υποδικτύου του network2 είναι 172.20.0.0/16.

```
#!/bin/bash

# check if containers networks exist and create them, if they do not.
docker network create --subnet=172.18.0.0/16 network1
docker network create --subnet=172.20.0.0/16 network2

# make sure our base image is already build.
#docker build . -t exp:1
```

Σχήμα 5.2: Δημιουργία εικονικών δικτύων.

Στη συνέχεια δημιουργούμε το container του consumer το οποίο συνδέουμε στο δίκτυο network1 και του δίνουμε τη διεύθυνση ip 172.18.0.2 (Σχήμα 5.3). Το ονομάσαμε umobile_consumer και τρέχει στο παρασκήνιο ένα bash session έχοντας πρόσβαση σε

ορισμένα αρχεία του κεντρικού υπολογιστή μέσω bind mounts. Έπειτα ξεκινάμε την εκτέλεση των εντολών που θέτουν σε λειτουργία το δαίμονα του DTN (dtnd) και το δαίμονα του NDN (nfd) για τα πειράματα NDN-DTN, ενώ για τα πειράματα DTN λειτουργεί μόνο ο δαίμονας του DTN (dtnd) .

```
#Run consumer dtn-ndn daemon running
docker run --rm -dit --net network1 --ip 172.18.0.2 --entrypoint bash --name umobile_consumer
-v "${PWD}/my_confs:/confs"  -v "${PWD}/ndn/ndn-cxx_umobile:/ndn-cxx_umobile" exp:1
docker exec -dit umobile_consumer /bin/bash -c 'dtnd -c /confs/my_dtn_consumer2.conf&
cp /confs/my_nfd_consumer.conf /usr/local/etc/ndn/nfd.conf && nfd-start &&
sh /confs/nfdc_register_consumer.sh& wait;'
```

```
#Run consumer dtn only
#docker run --rm -dit --net network1 --ip 172.18.0.2 --entrypoint bash  --name umobile_consumer
#-v "${PWD}/my_confs:/confs" test:2
#docker exec -dit umobile_consumer /bin/bash -c 'dtnd -c /confs/my_dtn_consumer2.conf& wait;'
```

Σχήμα 5.3: Δημιουργία container consumer.

Με τον ίδιο τρόπο δημιουργούμε το container του producer το οποίο όμως συνδέουμε με το δίκτυο network2 και του δίνουμε τη διεύθυνση ip 172.20.0.2 (Σχήμα 5.4). Αυτό ονομάστηκε umobile_producer και έχει επίσης πρόσβαση στα ίδια αρχεία μέσω bind mounts. Τα αρχεία αυτά, όπως αναφέραμε στην προηγούμενη ενότητα, είναι απαραίτητα για την εκτέλεση των πειραμάτων και για το compile των εφαρμογών (configuration files, build tools).

```
#Run producer dtn-ndn daemon running
docker run --rm -dit --net network2 --ip 172.20.0.2 --entrypoint bash  --name umobile_producer
-v "${PWD}/my_confs:/confs"  -v "${PWD}/ndn/ndn-cxx_umobile:/ndn-cxx_umobile" exp:1
docker exec -dit umobile_producer /bin/bash -c 'dtnd -c /confs/my_dtn_producer2.conf&
cp /confs/my_nfd_producer.conf /usr/local/etc/ndn/nfd.conf && nfd-start &&
sh /confs/nfdc_register_producer.sh& wait;'
```

```
#Run producer dtn only
#docker run --rm -dit --net network2 --ip 172.20.0.2 --entrypoint bash  --name umobile_producer
#-v "${PWD}/my_confs:/confs" test:2
#docker exec -dit umobile_producer /bin/bash -c 'dtnd -c /confs/my_dtn_producer2.conf&  wait;'
```

Σχήμα 5.4: Δημιουργία container producer.

Επιπρόσθετα, δημιουργούμε το container του intermediate (Σχήμα 5.5) στο οποίο η μόνη διαφορά που υπάρχει στις δύο κατηγορίες πειραμάτων(NDN-DTN, DTN) είναι η εικόνα που χρησιμοποιούμε. Η χρήση κοινής εικόνας σε όλα τα container για κάθε πείραμα συμβάλλει στο να διατηρήσουμε σταθερή τη χρήση των πόρων του συστήματος και όσο το δυνατόν σε χαμηλά επίπεδα. Αν χρησιμοποιήσουμε διαφορετικές εικόνες σε κάθε container, για παράδειγμα θα μπορούσε ο intermediate να χρησιμοποιεί την εικόνα test:2 σε κάθε πείραμα αφού τρέχει μόνο το δαίμονα (daemon) του DTN, τότε στη περίπτωση των πειραμάτων NDN-DTN οι δύο εικόνες test:2 και exp:1 θα διέφεραν σε ορισμένα επίπεδα (layers) και το τελικό μέγεθος που θα καταλάμβαναν τα container στο δίσκο θα ήταν μεγαλύτερο. Το container intermediate συνδέεται κατά τη δημιουργία του με το network2 και στη συνέχεια το συνδέουμε και με το network1.

```
# Run intermediate dtn &ndn daemon running
docker run --rm -dit --net network2 --entrypoint bash --name intermediate
-v "${PWD}/my_confs:/confs" exp:1
docker network connect network1 intermediate
docker exec -dit intermediate /bin/bash -c 'dtnd -c /confs/dtn_intermediate.conf& wait;'
```

```
#Run intermediate dtn only
#docker run --rm -dit --net network2 --entrypoint bash --name intermediate
#-v "${PWD}/my_confs:/confs" test:2
#docker network connect network1 intermediate
#docker exec -dit intermediate /bin/bash -c 'dtnd -c /confs/dtn_intermediate.conf& wait;'
```

Σχήμα 5.5: Δημιουργία container intermediate.

Τέλος, δημιουργούμε ένα script που αυτοματοποιεί τη διακοπτόμενη επικοινωνία των container κατά την υλοποίηση των πειραμάτων. Αρχικά, αποσυνδέουμε τα container consumer και producer, ώστε ο Intermediate να βρίσκεται στη κατάσταση αποσύνδεσης εξ αρχής. Στη συνέχεια συνδέουμε τον intermediate στο δίκτυο network1 για ένα συγκεκριμένο χρονικό διάστημα για να μπορεί να επικοινωνήσει με το container consumer, το αποσυνδέουμε, στη συνέχεια το συνδέουμε στο δίκτυο network2 για ένα συγκεκριμένο χρονικό διάστημα για να μπορεί να επικοινωνήσει με το container producer και το αποσυνδέουμε ξανά. Έτσι μεταβαίνει στις τρεις επιθυμητές καταστάσεις σύνδεση με το consumer, σύνδεση με το producer και αποσύνδεση από το δίκτυο (Σχήμα 5.6).

```
#!/bin/bash


docker network disconnect network1 umobile_consumer
sleep 2
docker network disconnect network2 umobile_producer
sleep 2

while true; do

    # State 1: Connected to consumer
    echo "Connected to consumer"
    docker network connect network1 umobile_consumer
    sleep 40
    # State 2: Not connected
    echo "Disconnected"
    docker network disconnect network1 umobile_consumer
    sleep 40

    # State 3: Connected to producer
    echo "Connected to producer"
    docker network connect network2 umobile_producer
    sleep 40


    # State 4: Not connected
    echo "Disconnected"
    docker network disconnect network2 umobile_producer
    sleep 40

done
```

Σχήμα 5.6: Διακοπτόμενη επικοινωνία του intermediate με τους consumer και producer .

## 5.2. Μεθοδολογία

Το Container του producer έχει στη διάθεση του αισθητήρια δεδομένα τα οποία στέλνει στο container του consumer μέσω της στοίβας NDN-DTN κάθε φορά που θα λαμβάνει ένα πακέτο Ενδιαφέροντος (Interest Packet). Όταν ο consumer θα λαμβάνει ένα πακέτο Δεδομένων (Data Packet) τότε θα στέλνει ένα νέο αίτημα στο producer επιλέγοντας τυχαία μεταξύ των διαθέσιμων αντικειμένων. Εάν κάποιο αίτημα αντικειμένου έχει ζητηθεί ξανά τότε αυτό θα είναι αποθηκευμένο στην κρυφή μνήμη (cache) του consumer, διαφορετικά θα πρέπει να σταλεί το αίτημα στον producer. Εδώ να τονίσουμε ότι στην περίπτωση των πειραμάτων του DTN όλα τα αιτήματα θα στέλνονται στο producer, αφού δεν αποθηκεύονται δεδομένα στην κρυφή μνήμη (cache).

Η παραπάνω διαδικασία θα επαναληφθεί για ένα καθορισμένο αριθμό ληφθέντων πακέτων Δεδομένων (Data Packet). Σε περίπτωση που λήξει η διάρκεια ζωής του πακέτου Ενδιαφέροντος (Interest Packet), τότε θα πραγματοποιηθεί επαναποστολή του πακέτου. Θα ορίσουμε τη διάρκεια ζωής των Bundles στη μία ώρα, ώστε να μη λήξει κάποιο Bundle κατά τη διάρκεια των πειραμάτων. Επιπλέον, θα επαναλάβουμε τα πειράματα 10-20 φορές για να εξασφαλίσουμε την ακρίβεια των αποτελεσμάτων μας.

Θα μελετήσουμε τέσσερα είδη πειραμάτων που το καθένα θα εξετάσει την επίδραση διαφορετικών παραμέτρων. Στα πειράματα 1-2 θα πραγματοποιήσουμε πειράματα στη στοίβα του πρωτοκόλλου DTN και NDN-DTN για να συγκρίνουμε την απόδοση τους. Αντιθέτως, στα πειράματα 3-4 θα επικεντρωθούμε στη στοίβα του πρωτοκόλλου NDN-DTN με σκοπό να μελετήσουμε την επιρροή των παραμέτρων που σχετίζονται μόνο με το πρωτόκολλο NDN.

Στο πείραμα 1 θα αξιολογήσουμε την απόδοση του συστήματος για διαφορετικές διάρκειες καταστάσεων (η χρονική περίοδος που ο intermediate κόμβος παραμένει σε κάθε κατάσταση σύνδεσης). Η διάρκεια ζωής (Lifetime) του πακέτου Ενδιαφέροντος (Interest Packet) και το χρονικό διάστημα που το πακέτο Δεδομένων θεωρείται “πρόσφατο” ωστέ να ληφθεί από τη μνήμη cache (Data Freshness) θα τεθούν σε υψηλές τιμές (1 ώρα και 2 ώρες αντίστοιχα) για να εξασφαλίσουμε ότι δε θα υπάρχει απώλεια πακέτων εξαιτίας αυτών των παραμέτρων. Σε κάθε επανάληψη ο consumer θα επιλέγει μεταξύ των έξι διαθέσιμων ονομάτων αντικειμένων. Στο συγκεκριμένο πείραμα, αντιμετωπίσαμε πρόβλημα στην εκτέλεση του πειράματος DTN, αφού ο consumer δε λάμβανε το bundle το οποίο ζητούσε κάθε φορά. Υπήρξαν περιπτώσεις που ζητούσε ένα bundle ξανά και ξανά. Αυτό όμως ικανοποιούνταν από ένα προηγούμενο bundle, καθώς δε περίμενε να σταλεί το κατάλληλο πακέτο Δεδομένων (Data Packet) από το producer. Αυτό αποτελούσε προβληματική συμπεριφορά. Έτσι αποφασίσαμε εκτός από το πακέτο Δεδομένων (Data Packet) να στέλνεται και η ταυτότητα του bundle. Ορίσαμε ως ταυτότητα του bundle το EID ( Endpoint Identifier) του producer, καθώς άλλαζε με την αποστολή ενός νέου πακέτου Δεδομένων (Data Packet). Με αυτό τον τρόπο ο consumer μπορούσε να ελέγξει κάθε φορά την τιμή της ταυτότητας, αν ήταν διαφορετική από την προηγούμενη να δέχεται τα δεδομένα, διαφορετικά θα έπρεπε να περιμένει για το σωστό bundle.

Στο πείραμα 2 θα διευρύνουμε το πλήθος των διαθέσιμων ονομάτων που θα μπορεί να ζητήσει ο consumer για να λάβει πακέτα Δεδομένων (Data Packet). Έτσι, θα μεταβληθεί ο

ρυθμός με τον οποίο λαμβάνονται δεδομένα από την κρυφή μνήμη του consumer (Cache Hit Ratio) της στοίβας NDN-DTN, αφού όσο πιο μικρός είναι ο αριθμός των διαθέσιμων αντικειμένων τόσο πιο μεγάλη είναι η πιθανότητα να επαναληφθεί ένα αίτημα δεδομένων μελλοντικά. Θα θέσουμε τη διάρκεια παραμονής σε μία κατάσταση στα 40 sec. Έτσι θα ελαχιστοποιήσουμε το χρόνο που απαιτείται ανά μέτρηση για να ικανοποιηθεί ένα αίτημα (Round Trip Time-RTT). Η διάρκεια ζωής των πακέτων Ενδιαφέροντος (Interest Packet) τέθηκε και εδώ στη 1 ώρα.

Στα πειράματα 3 και 4 θα εξετάσουμε την απόδοση της αρχιτεκτονικής NDN-DTN χρησιμοποιώντας διάφορες τιμές της διάρκειας ζωής των πακέτων Ενδιαφέροντος (Lifetime) και του και του χρονικού διαστήματος που το πακέτο Δεδομένων θεωρείται “πρόσφατο” ωστέ να ληφθεί από τη μνήμη cache (Data Freshness). Και στα δύο πειράματα η διάρκεια παραμονής σε μία κατάσταση θα τεθεί στα 40 sec και ο αριθμός των διαθέσιμων ονομάτων στην τιμή των δεκαπέντε (15). Στο πείραμα 3 το χρονικό διάστημα που το πακέτο Δεδομένων θεωρείται “πρόσφατο” ωστέ να ληφθεί από τη μνήμη cache (Data Freshness) θα τεθεί στις 2 ώρες, ενώ στο πείραμα 4 η τιμή της διάρκειας ζωής των πακέτων Ενδιαφέροντος (Lifetime) θα τεθεί στα 190 sec όπως και στο πείραμα του [5].

Η βασική παράμετρος που θα χρησιμοποιήσουμε για την αξιολόγηση των πειραμάτων ήταν η μέση τιμή του χρόνου καθυστέρησης λήψης των πακέτων Δεδομένων (Average Delay) [5]. Η χρονική αυτή διάρκεια καθορίζεται από τη στιγμή που θα υπάρξει ένα αίτημα περιεχομένου (π.χ αποστολή ενός πακέτου Ενδιαφέροντος) μέχρι τη στιγμή που θα ικανοποιηθεί το αίτημα, όταν δηλαδή ληφθεί το πακέτο Δεδομένων για το αντίστοιχο αίτημα. Σε περιπτώσεις επαναποστολής του πακέτου Ενδιαφέροντος η καθυστέρηση θα υπολογίζεται εκ νέου. Επίσης, θα χρησιμοποιήσουμε το ρυθμό με τον οποίο λαμβάνονται δεδομένα από τη κρυφή μνήμη (Cache Hit Ratio) και το ρυθμό με τον οποίο ικανοποιούνται τα πακέτα Ενδιαφέροντος (Interest Satisfaction Ratio-ISR).

Η παράμετρος *Cache Hit Ratio* (h) υπολογίζεται από την εξίσωση (1), όπου η μεταβλητή *Cache Hit* είναι ο συνολικός αριθμός των αιτημάτων που ικανοποιούνται από τη κρυφή μνήμη στο consumer και η μεταβλητή *Received Interests* είναι ο συνολικός αριθμός των πακέτων Ενδιαφέροντος για το οποία έχουμε λάβει αντίστοιχα πακέτα Δεδομένων (είτε από τον producer, είτε από τη κρυφή μνήμη στο consumer). Η παράμετρος Interest Satisfaction Ratio (ISR) υπολογίζεται από την εξίσωση (2), όπου η μεταβλητή *Satisfied Interest* είναι ο συνολικός αριθμός των πακέτων Ενδιαφέροντος τα οποία έχουν λάβει πακέτα Δεδομένων και η μεταβλητή *Sent Interest* είναι ο συνολικός αριθμός των απεσταλμένων πακέτων Ενδιαφέροντος.

$$\mathrm{h} = \frac{\text{Cache Hit}}{\text{ReceivedInterests}} \quad (1)$$

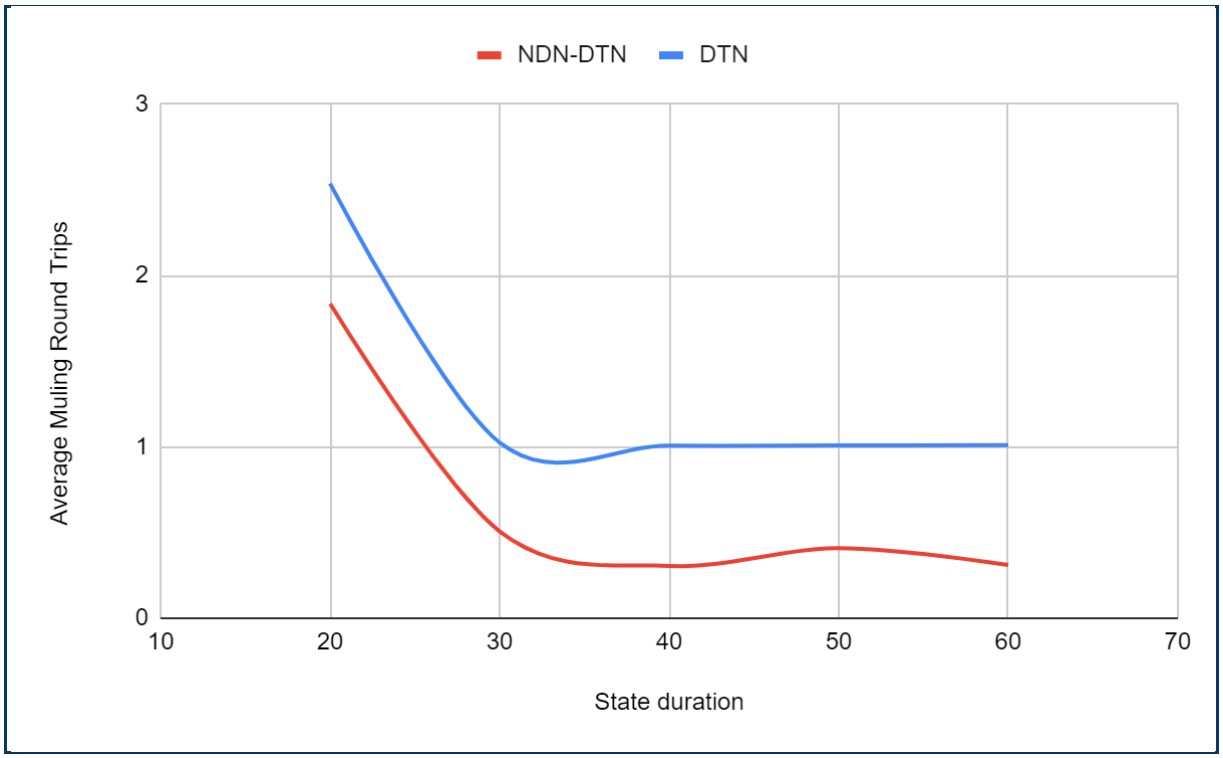
$$\mathrm{ISR} = \frac{\text{SatisfiedInterests}}{\text{SentInterests}} \quad (2)$$

## 5.3. Αποτελέσματα πειραμάτων

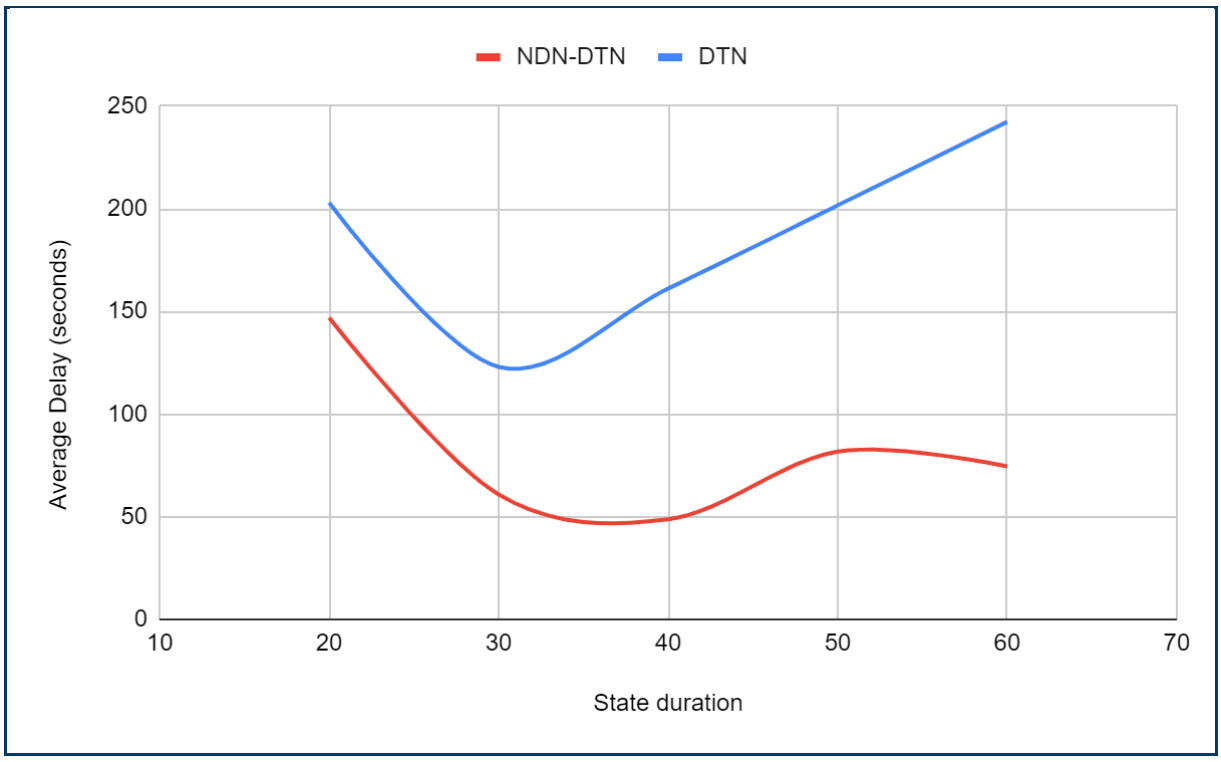
Πείραμα 1: Τα αποτελέσματα του πειράματος 1 παρουσιάζονται στο Σχήμα 5.7 και Σχήμα 5.8. Παρατηρούμε από το διάγραμμα ότι για διάρκεια παραμονής σε μία κατάσταση (state duration) ίση με 40 sec η μέση τιμή του απαιτούμενου χρόνου μετάδοσης μετ' επιστροφής (Average muling RTT) για την ικανοποίηση των αιτημάτων εμφανίζει τα καλύτερα αποτελέσματα. Για state duration μεγαλύτερο του 40 sec παρατηρούμε ότι το η μέση τιμή του απαιτούμενου χρόνου μετάδοσης μετ' επιστροφής (Average muling RTT) παραμένει σταθερή, καθώς ο χρόνος επαφής με τον κόμβο intermediate επαρκή για να γίνει η ανταλλαγή δεδομένων.

Για διάρκεια επαφής (state duration) μικρότερο από 40 sec παρατηρούμε ότι επηρεάζεται η ικανότητα του NDN-DTN να παραδίδει τα δεδομένα, καθώς ο διαθέσιμος χρόνος δεν επαρκεί για την ανταλλαγή μηνυμάτων. Αυτό το φαινόμενο μπορεί να οφείλεται στο γεγονός ότι η μεταβλητή discovery beacon [67] έχει τη προεπιλεγμένη τιμή της (5 sec) μειώνοντας έτσι το χρόνο που είναι διαθέσιμος για ανταλλαγή δεδομένων μεταξύ δύο κόμβων. Ένας επιπλέον παράγοντας μπορεί να είναι το μέγεθος των πακέτων του NDN-DTN που είναι μεγαλύτερο από το DTN, αφού συμβάλλουν και τα δύο πρωτόκολλα, με αποτέλεσμα να αυξάνεται το κόστος λειτουργίας.

Βλέπουμε πως τα αποτελέσματα της μέσης τιμής του χρόνου καθυστέρησης λήψης των πακέτων Δεδομένων (Average Delay) που σημειώνει το NDN-DTN έναντι του DTN είναι πολύ καλύτερα. Με τη χρήση του NDN αποθηκεύουμε δεδομένα στη κρυφή μνήμη (cache) του consumer, με αποτέλεσμα να μειώνεται ο χρόνος ανάκτησης των δεδομένων (σε σχέση με την ανάκτηση των δεδομένων από τον producer στο DTN). Έτσι αποδεικνύεται και με το εικονικό δίκτυο που δημιουργήσαμε το όφελος του NDN-DTN στα δίκτυα που εμφανίζουν διακοπτόμενες συνδέσεις.




Σχήμα 5.7: Πείραμα 1.Μέση τιμή του χρόνου μετάδοσης μετ' επιστροφής (Average mulling RTT) που απαιτούνται για τη μεταφορά των πακέτων από τον intermediate ανά μέτρηση για διάφορες τιμές της διάρκειας επαφής (state duration).

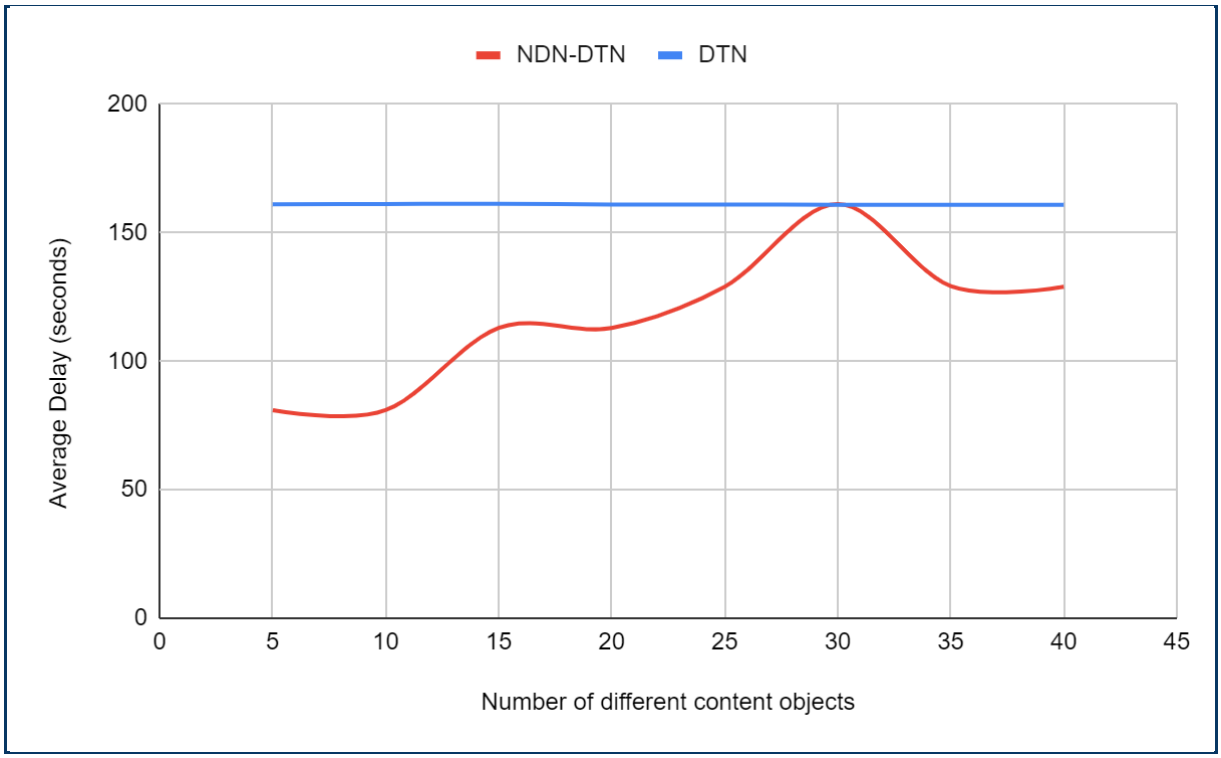



Σχήμα 5.8: Πείραμα 1.Μέσος χρόνος καθυστέρησης λήψης δεδομένων (Average Delay) για διάφορες τιμές της διάρκειας επαφής (state duration).

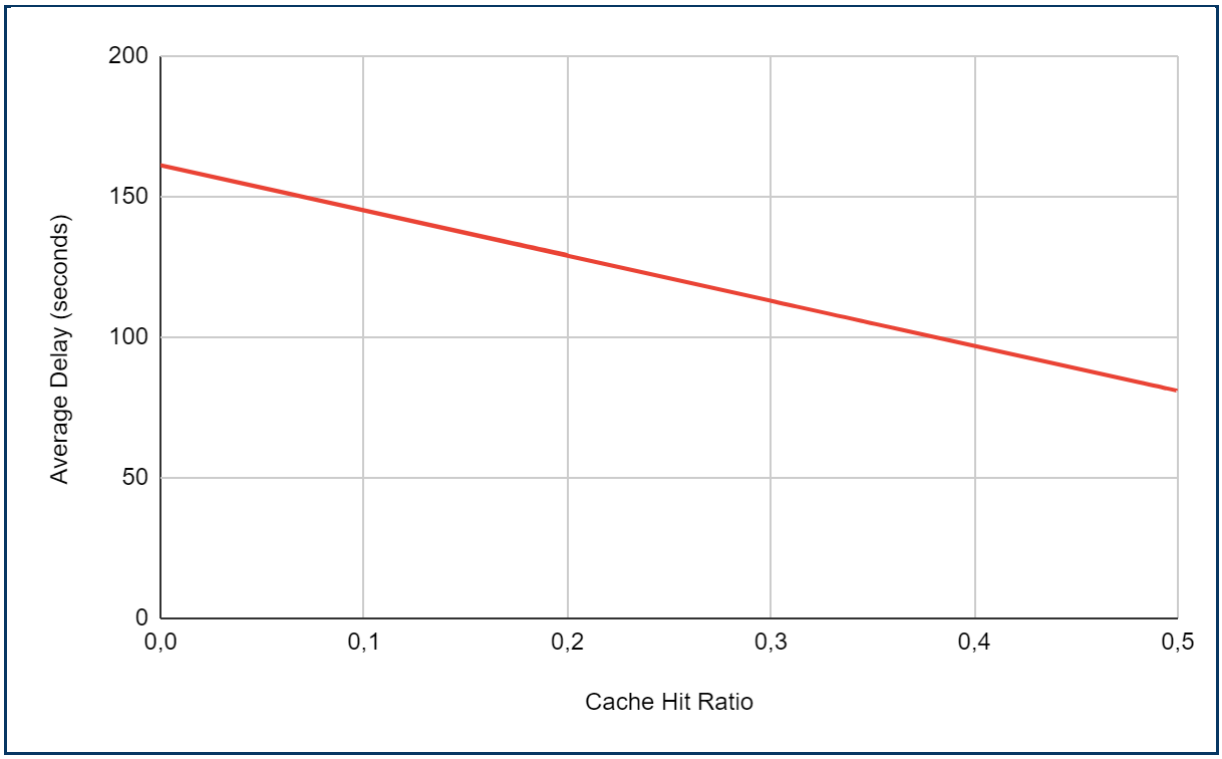
Πείραμα 2: Τα αποτελέσματα του πειράματος 2 παρουσιάζονται στο Σχήμα 5.9 και Σχήμα 5.10. Σε αυτό το πείραμα αυξάνονται σταδιακά τα διαθέσιμα αντικείμενα (αισθητήρια ονόματα) μειώνοντας έτσι τις πιθανότητες να ζητήσει ο consumer το ίδιο αντικείμενο στο ίδιο πείραμα. Αυτό έχει σαν αποτέλεσμα τη μείωση του ρυθμού με τον οποίο λαμβάνονται δεδομένα από τη κρυφή μνήμη (Cache Hit Ratio) στη περίπτωση του NDN-DTN όπως παρατηρούμε στο Σχήμα 5.10.

Γνωρίζουμε πως στο DTN δεν αποθηκεύονται δεδομένα στη κρυφή μνήμη, οπότε κάθε αίτημα ικανοποιείται από τον producer (ο intermediate μεταφέρει το αίτημα από τον consumer στο producer και στη συνέχεια τα δεδομένα από το producer στο consumer). Αυτό φαίνεται και από την απόδοση του DTN στο Σχήμα 5.9, όπου βλέπουμε ότι δεν εξαρτάται από τα διαθέσιμα ονόματα . Έτσι ο Μέσος χρόνος καθυστέρησης λήψης δεδομένων (Average Delay) παραμένει σταθερός στη περίπτωση του DTN, αφού και η διάρκεια επαφής (state duration) σε αυτό το πείραμα παραμένει σταθερό (40 sec). Είναι εμφανές ότι η αρχιτεκτονική NDN-DTN μπορεί να μειώσει το μέσο χρόνο καθυστέρησης (Average Delay) σε σχέση με το DTN. Αυτή η μείωση γίνεται όλο και πιο σημαντική όσο ο ρυθμός με τον οποίο λαμβάνονται δεδομένα από τη κρυφή μνήμη (Cache Hit Ratio) μεγαλώνει. Στην περίπτωση μας αυτό συμβαίνει για μικρό αριθμό διαθέσιμων αντικειμένων.

Παρατηρούμε ότι τα αποτελέσματα είναι συγκρίσιμα με τα αποτελέσματα στο [5]. Στην περίπτωση του NDN-DTN υπάρχει μία μικρή απόκλιση σε κάθε πείραμα, σε αυτό θα μπορούσαμε να πούμε ότι οφείλεται η τυχαιότητα των αιτημάτων. Στην περίπτωση του DTN (Σχήμα 5.9), όμως, το συγκεκριμένο μοντέλο παρουσιάζει μεγαλύτερη ακρίβεια. Πιθανόν να οφείλεται στην προσομοίωση του δικτύου και στη χρήση εικονικών συνδέσεων που αντικατοπτρίζουν μία ιδανική κατάσταση, χωρίς εμφάνιση προβλημάτων σύνδεσης που ενδεχομένως να προέκυψαν στο φυσικό δίκτυο. Τα αποτελέσματα στο Σχήμα 5.10 είναι σαφώς συγκρίσιμα με τα αποτελέσματα στο [5] και μάλιστα παρουσιάζουν μία συμπεριφορά όμοια με αυτήν του θεωρητικού αναλυτικού μοντέλου που αναπτύχθηκε.

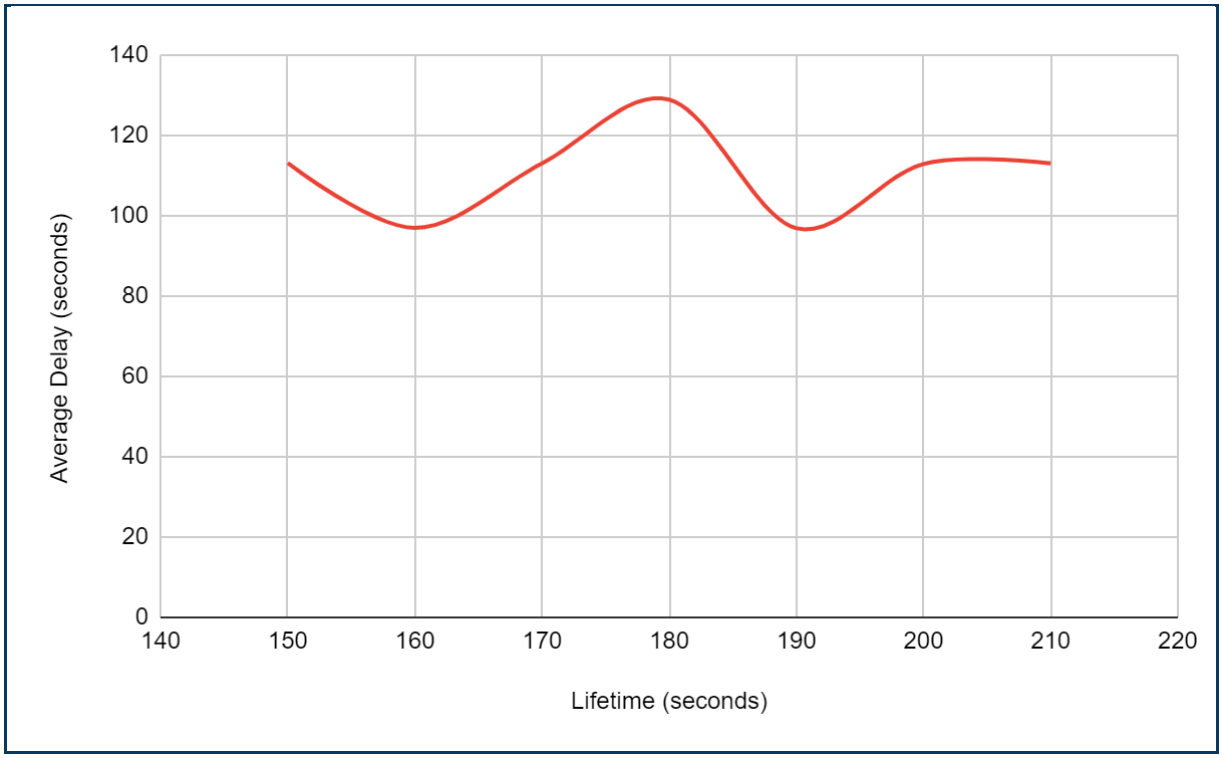



Σχήμα 5.9: Πείραμα 2.Μέσος χρόνος καθυστέρησης (Average Delay) σε σχέση με τα διαθέσιμα ονόματα.




Σχήμα 5.10: Πείραμα 2. Μέσος χρόνος καθυστέρησης (Average Delay) σε σχέση με το ρυθμό με τον οποίο λαμβάνονται δεδομένα από τη κρυφή μνήμη (Cache Hit Ratio).

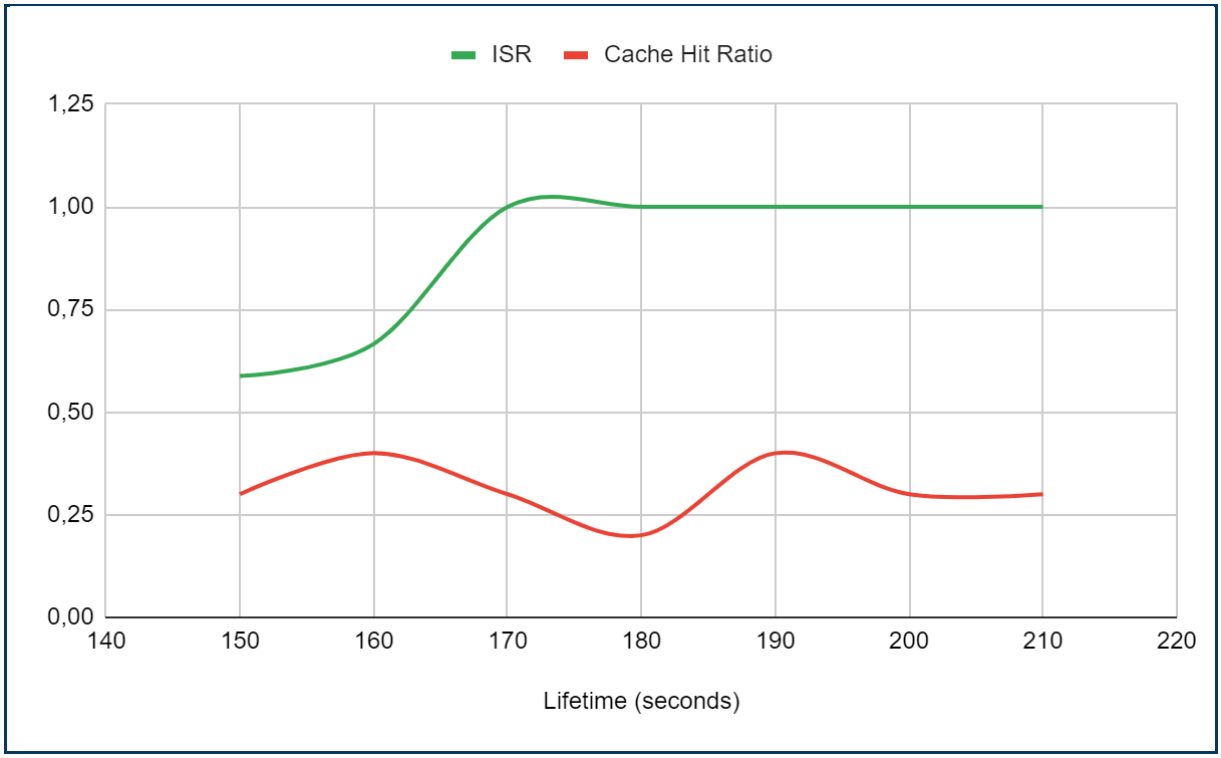
Πείραμα 3: Στο σχήμα 5.11 μπορούμε να παρατηρήσουμε ότι ο μέσος χρόνος καθυστέρησης (Average Delay) παρουσιάζει μικρές διακυμάνσεις, διότι επιδρά και η τυχαιότητα της επιλογής των αντικειμένων που αιτείται ο consumer, οι οποίες φαίνεται να είναι ανεξάρτητες της διάρκειας ζωής του πακέτου. Η απόδοση του NDN-DTN είναι σχετικά σταθερή, αφού έχουμε ρυθμίσει το consumer έτσι ώστε να πραγματοποιεί επαναποστολή του πακέτου Ενδιαφέροντος όταν αυτό λήξει. Συνεπώς, πάντα θα υπάρχει μία καταχώρηση στο πίνακα εκκρεμών πακέτων Ενδιαφέροντος (Pending Interest Table-PIT) για τη λήψη του πακέτου Δεδομένων. Να επισημάνουμε ότι τα πακέτα Δεδομένων μπορεί να αποστέλλονται ως απάντηση σε κάποιο προηγούμενο αίτημα.




Σχήμα 5.11: Πείραμα 3. Μέσος χρόνος καθυστέρησης (Average Delay) σε σχέση με τη διάρκεια ζωής των πακέτων Ενδιαφέροντος (Interest Lifetime).

Σε αντίθεση με τα αποτελέσματα στο [5] βλέπουμε στο Σχήμα 5.12 ότι για διάρκεια ζωής μεγαλύτερη των 170 sec η τιμή του ρυθμού με τον οποίο ικανοποιούνται τα πακέτα Ενδιαφέροντος (Interest Satisfaction Ratio-ISR) παραμένει σταθερή. Αυτό σημαίνει ότι για αυτές τις τιμές η διάρκεια ζωής του πακέτου Ενδιαφέροντος (Interest Lifetime) επαρκεί για να ικανοποιηθεί το αίτημα του. Για μικρότερες τιμές της διάρκειας ζωής του πακέτου Ενδιαφέροντος (Interest Lifetime) πυροδοτούνται περισσότερες αναμεταδόσεις πακέτων, καθώς η διάρκεια ζωής των πακέτων Ενδιαφέροντος (Interest Lifetime) δεν επαρκεί για να ληφθούν τα πακέτα Δεδομένων. Άρα μπορούμε να αποφανθούμε ότι η επιλογή του της διάρκεια ζωής του πακέτου Ενδιαφέροντος (Interest Lifetime) καθορίζει το ρυθμό με τον οποίο ικανοποιούνται τα πακέτα Ενδιαφέροντος (Interest Satisfaction Ratio-ISR). Οι διακυμάνσεις του ρυθμού με τον οποίο ικανοποιούνται τα πακέτα Ενδιαφέροντος (Interest Satisfaction Ratio-ISR) με τη σειρά τους δε φαίνονται να επηρεάζουν το μέσο χρόνο καθυστέρησης (Average Delay). Επιπλέον, η επιλογή της διάρκειας ζωής των πακέτων

Ενδιαφέροντος (Interest Lifetime) δε φαίνεται να επηρεάζει το ρυθμό με τον οποίο λαμβάνονται δεδομένα από τη κρυφή μνήμη (Cache Hit Ratio) σημαντικά, συνεπώς και του μέσου χρόνου καθυστέρησης (Average Delay), καθώς αυτά τα δύο όπως φάνηκε και στο πείραμα 2 είναι αλληλένδετα.




Σχήμα 5.12: Πείραμα 3. Αποτελέσματα του ρυθμού με τον οποίο ικανοποιούνται τα πακέτα Ενδιαφέροντος (Interest Satisfaction Ratio-ISR) και του ρυθμού με τον οποίο λαμβάνονται δεδομένα από τη κρυφή μνήμη (Cache Hit Ratio)σε σχέση με τη διάρκεια ζωής των πακέτων Ενδιαφέροντος (Interest Lifetime).

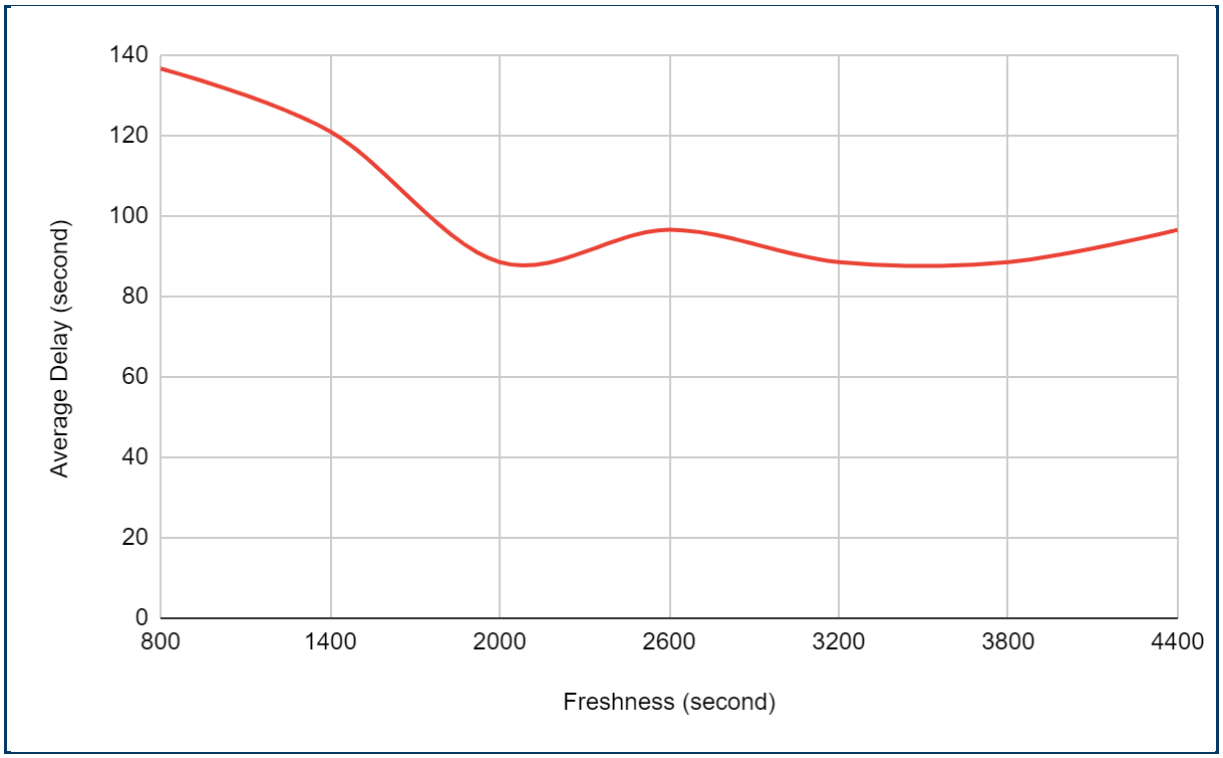
Με βάση τα αποτελέσματα μπορούμε να συμπεράνουμε ότι στην αρχιτεκτονική NDN-DTN η επιλογή της τιμής της διάρκεια ζωής του πακέτου Ενδιαφέροντος (Interest Lifetime) έχει τόσο πλεονεκτήματα όσο και μειονεκτήματα. Η επιλογή μιας μικρής τιμής μπορεί να προκαλέσει μεγαλύτερη κίνηση στο δίκτυο, αλλά τα πακέτα Ενδιαφέροντος παραμένουν για λιγότερο χρόνο στο πίνακα εκκρεμών πακέτων Ενδιαφέροντος (Pending Interest Table) και αποδεσμεύουν τη μνήμη πιο γρήγορα, ενώ το αντίθετο συμβαίνει για μεγάλες τιμές της διάρκειας ζωής του πακέτου Ενδιαφέροντος (Interest Lifetime). Συμπερασματικά, η επιλογή της διάρκειας ζωής του πακέτου Ενδιαφέροντος (Interest Lifetime) δεν επηρεάζει το μέσο χρόνο καθυστέρησης (Average Delay) δεδομένου ότι κατά τη λήξη των πακέτων Ενδιαφέροντος ο consumer προχωρά στην αναμετάδοση τους.

Πείραμα 4: Στο σχήμα 5.13 παρατηρούμε ότι καθώς αυξάνεται το χρονικό διάστημα που το πακέτο Δεδομένων θεωρείται “πρόσφατο” ωστέ να ληφθεί από τη μνήμη cache (Data Freshness) μειώνεται ο μέσος χρόνος καθυστέρησης (Average Delay). Αυτό οφείλεται, κυρίως, στην αύξηση του ρυθμού με τον οποίο λαμβάνονται δεδομένα από τη κρυφή μνήμη (Cache Hit Ratio) (Σχήμα 5.14), καθώς τα δεδομένα παραμένουν στο Content Store (CS) για

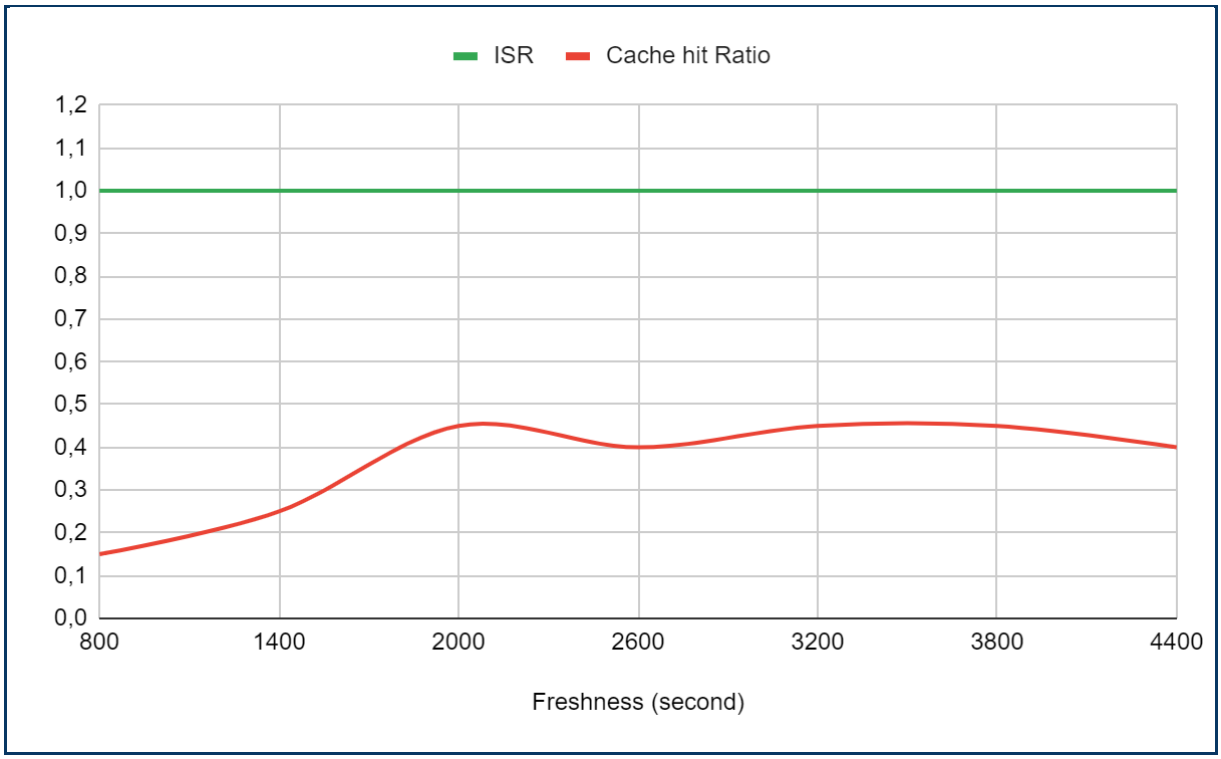
μεγαλύτερο χρονικό διάστημα και περισσότερα μεταγενέστερα πακέτα Ενδιαφέροντος μπορούν να ικανοποιηθούν από αυτά.

Παρόλα αυτά, για τις τιμές του χρονικού διαστήματος που το πακέτο Δεδομένων θεωρείται “πρόσφατο” ωστέ να ληφθεί από τη μνήμη cache (Data Freshness) που είναι μεγαλύτερες από 3000 sec παρατηρούμε τη διακοπή της αυξητικής πορείας του ρυθμού με τον οποίο λαμβάνονται δεδομένα από την κρυφή μνήμη (Cache Hit Ratio). Το αποτέλεσμα αυτό είναι άρρηκτα συνδεδεμένο με το χρόνο που απαιτείται ώστε ένα πακέτο Ενδιαφέροντος να μεταδοθεί από το consumer στο producer και στη συνέχεια να επιστραφεί το πακέτο Δεδομένων στο consumer (RTT). Ο χρόνος αυτός είναι περίπου ίσος με 4*40=160 sec.

Όταν ένα πακέτο Δεδομένων ληφθεί από ένα τοπικό δρομολογητή, μπορεί να ανακτηθεί από τη μνήμη cache όσο αυτό θεωρείται “φρέσκο”. Επαναλαμβάνουμε το συγκεκριμένο πείραμα έως ότου λάβουμε 20 αντικείμενα περιεχομένου. Θα μπορούσαμε να πούμε ότι μία εκτίμηση του χρόνου που τα πακέτα παραμένουν “φρέσκα” είναι 20*160=3200 sec. Αυτό δε μας εκπλήσσει, αφού όπως φαίνεται και από τα αποτελέσματα στο Σχήμα 5.14 αυτή είναι και η τιμή που εμφανίζει το μέγιστο ρυθμό με τον οποίο λαμβάνονται δεδομένα από την κρυφή μνήμη (Cache Hit Ratio). Σε αντίθεση με το [5] δεν είναι η μοναδική τιμή για την οποία έχουμε το καλύτερο αποτέλεσμα του μέσου χρόνου καθυστέρησης (Average Delay). Ίσως εδώ η τυχαιότητα να ήταν με το μέρος μας, καθώς εμφάνισε το ίδιο ικανοποιητικά αποτελέσματα και για τη τιμή 2000 sec. Τέλος, παρατηρούμε ότι οι αλλαγές στο χρονικό διάστημα που το πακέτο Δεδομένων θεωρείται “πρόσφατο” ωστέ να ληφθεί από τη μνήμη cache (Data Freshness) δεν επηρεάζουν το ρυθμό με τον οποίο ικανοποιούνται τα πακέτα Ενδιαφέροντος (Interest Satisfaction Ratio-ISR). Ο ρυθμός με τον οποίο ικανοποιούνται τα πακέτα Ενδιαφέροντος (Interest Satisfaction Ratio-ISR) όπως είδαμε στο πείραμα 3 καθορίζεται από τη διάρκεια ζωής των πακέτων Ενδιαφέροντος (Interest Lifetime) και εδώ παραμένει σταθερό αφού επιλέξαμε τη σταθερή τιμή 190 sec που παρουσιάζει ISR=1.




Σχήμα 5.13: Πείραμα 4. Μέσος χρόνος καθυστέρησης (Average Delay) σε σχέση με το χρονικό διάστημα που το πακέτο Δεδομένων θεωρείται “πρόσφατο” ωστέ να ληφθεί από τη μνήμη cache (Data Freshness).




Σχήμα 5.14: Πείραμα 4. Αποτελέσματα του ρυθμού με τον οποίο ικανοποιούνται τα πακέτα Ενδιαφέροντος (Interest Satisfaction Ratio-ISR) και του ρυθμού με τον οποίο λαμβάνονται δεδομένα από την κρυφή μνήμη (Cache Hit Ratio)σε σχέση με το χρονικό διάστημα που το πακέτο Δεδομένων θεωρείται “πρόσφατο” ωστέ να ληφθεί από τη μνήμη cache (Data Freshness).

# Κεφάλαιο 6

# Συμπεράσματα και Μελλοντική Εργασία

Η παρούσα διπλωματική εργασία παρουσιάζει τις αρχιτεκτονικές προσεγγίσεις DTN, NDN και UMOBILE ως λύσεις για την κάλυψη των σύγχρονων αναγκών του Διαδικτύου. Εξετάζει τεχνολογίες εικονικοποίησης (Virtualization) για την ανάπτυξη εφαρμογών στα άκρα του δικτύου. Αποδεικνύει ότι η τεχνολογία των Container αξιοποιεί καλύτερα τους πόρους του συστήματος. Συγκεκριμένα, επιλέγεται μία πλατφόρμα ευρέως διαδεδομένη και συνυφασμένη με τα Container, το Docker.

Στη συνέχεια αναπτύσσεται μία εικόνα για τη δημιουργία των Docker Container που αξιοποιούν τα πρωτόκολλα DTN και το NDN-DTN για την εκτέλεση εφαρμογών. Εφαρμόζονται ορισμένες τεχνικές βελτιστοποίησης της εικόνας αυτής για να επιτευχθεί καλύτερη ασφάλεια, απόδοση και αποδοτικότητα. Αφού δημιουργείται η κατάλληλη εικόνα, εξετάζεται η απόδοση των πρωτοκόλλων σε ένα εικονικό δίκτυο IoT που αναπτύσσεται με τη χρήση των Docker Container. Τα αποτελέσματα των πειραμάτων μας οδήγησαν σε μερικά συμπεράσματα:

A. Η στοίβα που αναπτύχθηκε στην προσέγγιση UMOBILE και αξιοποιεί τις ιδιότητες τόσο του NDN όσο και του DTN μπορεί να χρησιμοποιηθεί σε περιβάλλοντα προσομοίωσης δικτύων IoT, όπως αυτό που δημιουργήσαμε με το εργαλείο Docker, καταναλώνοντας τους ελάχιστους πόρους. Τα αποτελέσματα της στοίβας NDN-DTN, όπως επιβεβαιώνεται και εδώ, ξεπερνούν την απόδοση της στοίβας DTN στην ανάκτηση δεδομένων από απομονωμένα περιβάλλοντα. Το εικονικό μοντέλο που αναπτύχθηκε σε ορισμένες περιπτώσεις εμφανίζει αποτελέσματα συγκρίσιμα με το θεωρητικό αναλυτικό μοντέλο του [5].
B. Έπειτα από συγκρίσεις των αποτελεσμάτων μας με το [5], το δίκτυο προσομοίωσης που δημιουργήσαμε αποδεικνύει και αυτό σε εικονικό επίπεδο τη μείωση του μέσου χρόνου καθυστέρησης που επιτυγχάνεται από την κοινή στοίβα NDN-DTN έναντι του DTN.
C. Επαληθεύεται ότι ο ρυθμός με τον οποίο λαμβάνονται δεδομένα από την κρυφή μνήμη (Cache Hit Ratio) που επιτεύχθηκε στα πειράματα NDN-DTN αποτελεί σημαντική βελτίωση. Όσο αυξάνεται ο ρυθμός με τον οποίο λαμβάνονται δεδομένα από τη κρυφή μνήμη (Cache Hit Ratio), τόσο μειώνεται ο μέσος χρόνος καθυστέρησης λήψης των Δεδομένων (Average Delay).
D. Η διάρκεια ζωής των πακέτων Ενδιαφέροντος (Interest Lifetime) δεν επηρεάζει τη μέση τιμή του χρόνου καθυστέρησης λήψης δεδομένων (Average Delay), καθορίζει όμως το ρυθμό με τον οποίο ικανοποιούνται τα πακέτα Ενδιαφέροντος (Interest Satisfaction Ratio). Ωστόσο, η επιλογή κατάλληλης τιμής της διάρκειας ζωής των πακέτων Ενδιαφέροντος (Interest Lifetime) προκύπτει από τις απαιτήσεις του συστήματος (διαθέσιμος αποθηκευτικός χώρος, ενεργειακή επάρκεια).

Ε. Το χρονικό διάστημα που το πακέτο Δεδομένων θεωρείται “πρόσφατο” ωστέ να ανακτηθεί από τη κρυφή μνήμη ( Data Freshness) θα πρέπει να τεθεί ιδανικά σε μια υψηλή τιμή, όπως αυτή υπολογίζεται από την καθυστέρηση στα πειράματα DTN, για να εκμεταλλευτεί πλήρως την αποθήκευση των δεδομένων στην κρυφή μνήμη (cache). Η επιλογή χαμηλότερης τιμής θα οδηγήσει στην αύξηση του μέσου χρόνου καθυστέρησης (Average Delay) και στη μείωση του ρυθμού με τον οποίο λαμβάνονται δεδομένα από την κρυφή μνήμη (Cache Hit Ratio).

Μελλοντικά θα μπορούσε να δημιουργηθεί μια εικόνα για τη δημιουργία Container με καλύτερα χαρακτηριστικά από αυτήν που δημιουργήσαμε αξιοποιώντας νέες μεθόδους και πρακτικές για την ελαχιστοποίηση του μεγέθους της. Με τη χρήση των Docker Container τα πειράματα θα μπορέσουν να διευρυνθούν σε μέγεθος και να προστεθούν περισσότεροι κόμβοι για το σχηματισμό ενός μεγαλύτερου δικτύου και την αξιολόγηση των πρωτοκόλλων σε αυτό. Τέλος, μπορούν να εξεταστούν και άλλες μετρικές των πρωτοκόλλων για την αξιολόγηση της απόδοσης τους, όπως για παράδειγμα διαφορετικές στρατηγικές δρομολόγησης.

# Βιβλιογραφία


[1] E. Baccelli, C. Mehlis, O. Hahm, T. C. Schmidt, and M. Wählisch, “Information centric networking in the IoT: Experiments with NDN in the wild,” in *ICN 2014 - Proceedings of the 1st International Conference on Information-Centric Networking*, 2014, pp. 77–86, doi: 10.1145/2660129.2660144.

[2] M. Amadeo, C. Campolo, A. Iera, and A. Molinaro, “Named data networking for IoT: An architectural perspective,” *EuCNC 2014 - Eur. Conf. Networks Commun.*, pp. 1–5, 2014, doi: 10.1109/EuCNC.2014.6882665.

[3] S. Burleigh *et al.*, “Delay-tolerant networking: An approach to interplanetary internet,” *IEEE Commun. Mag.*, vol. 41, no. 6, pp. 128–136, 2003, doi: 10.1109/MCOM.2003.1204759.

[4] S. Burleigh, V. G. Cerf, J. Crowcroft, and V. Tsaoussidis, “Space for Internet and Internet for space,” *Ad Hoc Networks*, vol. 23, pp. 80–86, 2014, doi: 10.1016/j.adhoc.2014.06.005.

[5] C. Sarros, V. Demiroglou, and V. Tsaoussidis, “Intermittently-connected IoT devices : Experiments with an NDN-DTN architecture,” in *IEEE 18th Annual Consumer Communications & Networking Conference (CCNC)*, 2021, pp. 1–9, doi: 10.1109/CCNC49032.2021.9369578.

[6] L. Zhang *et al.*, “Named data networking,” *Comput. Commun. Rev.*, vol. 44, no. 3, pp. 66–73, 2014, doi: 10.1145/2656877.2656887.

[7] E. Borgia, R. Bruno, and A. Passarella, “Making opportunistic networks in IoT environments CCN-ready: A performance evaluation of the MobCCN protocol,” *Comput. Commun.*, vol. 123, pp. 81–96, 2018, doi: 10.1016/j.comcom.2018.03.005.

[8] C. A. Sarros *et al.*, “Connecting the Edges: A Universal, Mobile-Centric, and Opportunistic Communications Architecture,” *IEEE Communications Magazine*, vol. 56, no. 2, IEEE, pp. 136–143, 2018.

[9] B. I. Ismail *et al.*, “Evaluation of Docker as Edge computing platform,” in *ICOS 2015 - 2015 IEEE Conference on Open Systems*, 2016, pp. 130–135, doi: 10.1109/ICOS.2015.7377291.

[10] D. Documentation, “Use multi-stage builds.” https://docs.docker.com/develop/develop-images/multistage-build/.

[11] K. Fall, “A Delay-Tolerant Network Architecture for Challenged Internets,” *Comput. Commun. Rev.*, vol. 33, no. 4, pp. 27–34, 2003, doi: 10.1145/863956.863960.

[12] V. Cerf, S. Burleigh, A. Hooke, L. Torgerson, R. Durst, K. Scott, K. Fall, E.Travis,H. Weiss, “Interplanetary Internet (IPN): Architectural Definition,” 2001. https://tools.ietf.org/id/draft-irtf-ipnrg-arch-00.txt.

[13] M. Gritter and D. R. Cheriton, “An architecture for content routing support in the internet,” 2001.

[14] K. Fall, “A Delay-Tolerant Network Architecture for Challenged Internet,” 2003.

[15] “CFDP Protocol Specification, CCSDS 727.0-B-1,” 2002. http://www.ccsds.org.

[16] L. Boldt-Christmas, “Low Earth orbit,” *ESA*. https://www.esa.int/ESA_Multimedia/Images/2020/03/Low_Earth_orbit.

[17] J. Wroclawski, “The MetaNet: White Paper,” *Workshop on Research Directions for the Next Generation Internet*. http://www.cra.org/Policy/NGI/papers/wroklawWP.

[18] V. G. Cerf and R. E. Kahn, “A protocol for packet network intercommunication,” *IEEE Transactions on Communications*, vol. 22, no. 5, pp. 637–648, 1978.

[19] D. D. Clark, “The design philosophy of the DARPA internet protocols,” *Symp. Proc. Commun. Archit. Protoc. SIGCOMM 1988*, vol. 02139, pp. 106–114, 1988, doi: 10.1145/52324.52336.

[20] E. Chen and J. Stewart, “A Framework for Inter-Domain Route Aggregation,” *Internet RFC2519*, 1999.

[21] P. Mockapetris, “Domain Names - Concepts and Facilities,” *The Internet Society*, 1987.

https://tools.ietf.org/html/rfc1034.
[22] M. Mealling, R. Denenbers, and Eds., “Report from the Joint W3C/IETF URI Planning Interest Group: Uniform Resource Identifiers (URIs), URLs, and Uniform Resource Names (URNs): Clarifications and Recommendations,” *Internet RFC 3305*, 2002.
[23] J. L. W. Adgie-Winoto, E. Schwartz, H. Balakrishnan, “The design and implementation of an intentional naming system,” in *Proceedings SOSP*, 1999, pp. 186–201, doi: https://doi.org/10.1145/319151.319164.
[24] J. Alonso and K. R. Fall, “A Linear Programming Formulation of Flows over Time with Piecewise Constant Capacity and Transit Times,” 2003.
[25] R. Teixeira, A. Shaikh, T. Griffin, and J. Rexford, “Dynamics of hot-potato routing in IP networks,” *ACM SIGMETRICS Performance Evaluation Review*, vol. 32, no. 1, p. 307, 2004.
[26] M. Handley, “Why the Internet only just works,” *BT Technol. J. 24*, pp. 119–129, 2006.
[27] C. Dovrolis, “Future internet architecture: Clean-slate versus evolutionary research,” in *Communications of the ACM*, 2010, vol. 53, no. 9, pp. 36–40, doi: 10.1145/1810891.1810906.
[28] G. Xylomenos *et al.*, “A Survey of information-centric networking research,” in *IEEE Communications Surveys and Tutorials*, 2014, vol. 16, no. 2, pp. 1024–1049, doi: 10.1109/SURV.2013.070813.00063.
[29] A. Ghodsi, T. Koponen, J. Rajahalme, P. Sarolahti, and S. Shenker, “Naming in content-oriented,” in *in ACM SIGCOMM Workshop on Information-Centric Networking (ICN)*, 2011, pp. 1–6.
[30] “Content Centric Networking project.”
[31] “A New Way to Look at Networking.” https://www.youtube.com/watch?v=oCZMoY3q2uM.
[32] T. Berners-Lee, “Uniform Resource Locators (URL): A Syntax for the Expression of Access Information of Objects on the Network,” *World Wide Web Consortium*. 1994.
[33] I. F. Alshaikhli and M. A. AlAhmad, “Cryptographic Hash Function: A High Level View,” in *Handbook of Research onThreat Detection and Countermeasures in Network Security*, 2015, pp. 80–94.
[34] V. Jacobson, D. K. Smetters, N. H. Briggs, M. F. Plass, J. D. Thornton, and R. L. Braynard, “VoCCN : Voice-over Content-Centric Networks,” in *Proceedings of the 2009 workshop on Re-architecting the internet*, 2009, pp. 1–6.
[35] “Understanding Denial-of-Service Attacks,” *US-CERT*, 2019. https://us-cert.cisa.gov/ncas/tips/ST04-015.
[36] W. K. Chai, D. He, I. Psaras, and G. Pavlou, “Cache ‘less for more’ in information-centric networks,” 2012, doi: 10.1007/978-3-642-30045-5_3.
[37] I. Psaras, W. K. Chai, and G. Pavlou, “Probabilistic In-Network Caching for Information-Centric Networks,” 2012.
[38] M. Meisel, V. Pappas, and L. Zhang, “Ad Hoc Networking via Named Data Michael,” in *Proceedings of the fifth ACM international workshop on Mobility in the evolving internet architecture*, 2010, pp. 3–8.
[39] D. Smetters and V. Jacobson, “Securing network content,” *Palo Alto Res. Cent.*, pp. 1–7, 2009.
[40] S. DiBenedetto, P. Gasti, G. Tsudik, and E. Uzun, “ANDaNA: Anonymous Named Data Networking Application,” *Netw. Distrib. Syst. Secur. Symp.*, Dec. 2012.
[41] U. Project, “D3.2 UMOBILE architecture report (2).” http://umobile-project.eu/deliverables.
[42] O. Ascigil, V. Sourlas, I. Psaras, and G. Pavlou, “Opportunistic off-path content discovery in information-centric networks,” *IEEE Work. Local Metrop. Area Networks*, pp. 1–7, 2016, doi: 10.1109/LANMAN.2016.7548860.
[43] M. Conti, F. Delmastro, G. Minutiello, and R. Paris, “Experimenting opportunistic networks with WiFi Direct,” *IFIP Wirel. Days*, pp. 1–6, 2013, doi: 10.1109/WD.2013.6686501.
[44] W. Moreira and P. Mendes, “Impact of Human Behavior on Social Opportunistic Forwarding,” *Ad Hoc Networks*, vol. 25, no. Part B, pp. 293–302, 2015.
[45] A. Carzaniga, M. Papalini, and A. L. Wolf, “Content-Based ( Publish / Subscribe ) Networking,”

in *Proc. ACM SIGCOMM Wksp. Information-Centric Networking*, 2011, pp. 56–61.
[46] I. Psaras *et al.*, “Keyword-based mobile application sharing,” in *Proceedings of the Workshop on Mobility in the Evolving Internet Architecture*, 2016, pp. 1–6.
[47] B. Schwarz, “Content delivery networks 3.0,” *CTOiC White Pap. Tech. Rep.*, 2013, [Online]. Available: http://www.broadpeak.tv/upload/documentation/fichier/73-264-live_webtv_white_paper_b_schwarz_201212.pdf.
[48] C. D. Graziano, “A performance analysis of Xen and KVM hypervisors for hosting the Xen Worlds Project,” 2011.
[49] A. Singh, “An introduction to virtualization.,” 2004. http://www.kernelthread.com.
[50] IBM Cloud Education, “Containerization.” https://www.ibm.com/cloud/learn/containerization.
[51] Docker Inc., “Docker Engine.” https://docs.docker.com/engine/.
[52] D. A. Rusling, *The linux kernel.* 1999.
[53] R. J. Anthony, “The Process View,” in *Systems Programming*, Elsevier, 2016, pp. 21–106.
[54] Apache Software Foundation, “About the Apache HTTP Server Project.” http://httpd.apache.org/ABOUT_APACHE.html.
[55] M. Andrian, “The What and Why of Containers,” in *Using Docker: Developing and Deploying Software with Containers*, 2015, pp. 1–355.
[56] Kubernetes Authors, “What is Kubernetes?” https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/.
[57] T. Sappal, “Keep it Small : tips for optimising docker images for a faster development workflow,” 2019. https://medium.com/tech-bits/keep-it-small-tips-for-optimising-docker-images-for-a-faster-development-workflow-2d4e0d9d0ed2.
[58] P. Rakić, “Docker Image Size – Does It Matter?,” *Semaphore*, 2020. .
[59] Docker Inc., “Best practices for writing Dockerfiles.” https://docs.docker.com/develop/develop-images/dockerfile_best-practices/.
[60] “IBR-DTN:A modular and lightweight implementation of the bundle protocol.” https://github.com/ibrdtn/ibrdtn.
[61] “Named Data Networking Forwarding Daemon.” https://named-data.net/doc/NFD.
[62] “NDN C++ library with eXperimental eXtensions.” https://named-data.net/doc/ndn-cxx/current/.
[63] The Linux Documentation Project, “Shared Libraries.” https://tldp.org/HOWTO/Program-Library-HOWTO/shared-libraries.html.
[64] Docker Inc., “Dockerfile reference: Entrypoint.” https://docs.docker.com/engine/reference/builder/#entrypoint.
[65] Docker Inc., “Manage data in Docker.” https://docs.docker.com/storage/.
[66] Docker Inc., “Use bind mounts.” https://docs.docker.com/storage/bind-mounts/.
[67] A. Gladd, D. Brown, D. Ellard, R. Altmann, and R. . Velt, “DTN IP Neighbor Discovery (IPND),” 2015.