ΔΗΜΟΚΡΙΤΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΡΑΚΗΣ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΛΟΓΙΣΜΙΚΟΥ ΚΑΙ ΑΝΑΠΤΥΞΗΣ ΕΦΑΡΜΟΓΩΝ

ΕΡΓΑΣΤΗΡΙΟ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ ΚΑΙ ΕΠΕΞΕΡΓΑΣΙΑΣ ΠΛΗΡΟΦΟΡΙΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ ΜΕ ΘΕΜΑ

Αρχιτεκτονική UMOBILE - επεκτάσεις σε σχέση με το ευκαιριακό δίκτυο

Κωνσταντίνος Πρασόπουλος

Επιβλέπων: Αυγερινός Αραμπατζής, Επίκουρος Καθηγητής

Ξάνθη, Ιούνιος 2017

Θα ήθελα να ευχαριστήσω θερμά τον Καθηγητή κ. Βασίλη Τσαουσίδη για την ανάθεση του συγκεκριμένου θέματος, την έμπνευση, καθοδήγηση και πολύ καλή συνεργασία που είχα μαζί του, παρά τα εμπόδια που αντιμετωπίζει. Επίσης ευχαριστώ τους υποψήφιους Διδάκτορες Σωτήρη Διαμαντόπουλο και Αλέξανδρο Σάρρο για την πολύτιμη και απλόχερη βοήθεια που μου παρείχαν καθ' όλη τη διάρκεια της εκπόνησης αυτής της εργασίας.

Θέλω να αφιερώσω την εργασία αυτή στους γονείς μου, Παναγιώτη και Κατερίνα, στην αδελφή μου Ιωάννα και στην Ειρήνη.

Στο πλαίσιο της παρούσας διπλωματικής εργασίας ενσωματώθηκαν τα πρωτόκολλα ICN (Information Centric Networking) και DTN (Delay Tolerant Networking) σε ένα ενιαίο δικτυακό stack για κινητές συσκευές με το λειτουργικό σύστημα Android. Η σύνδεση των δύο πρωτοκόλλων προδιαγράφεται στην αρχιτεκτονική UMOBILE και έχει σκοπό την αξιόπιστη παροχή περιεχομένου και υπηρεσιών σε τελικούς χρήστες παρά τις ελλείψεις ή τις δυσκολίες, οι οποίες μπορεί να παρουσιάζονται στη δικτυακή υποδομή. Για την αξιολόγηση της υλοποίησης αναπτύχθηκε μία εφαρμογή, η οποία λειτούργησε ως πλατφόρμα για τη διεξαγωγή πειραμάτων και την καταγραφή αποτελεσμάτων σε πραγματικά δίκτυα. Τα πειράματα πραγματοποιήθηκαν σε κανονικές συνθήκες και σε συνθήκες διαλείπουσας συνδεσιμότητας του ασύρματου δικτύου Wi-Fi. Το νέο αυτό protocol stack παρέχει πλήρη αξιοπιστία πετυχαίνοντας, σε όλες τις δοκιμές που πραγματοποιήθηκαν, packet delivery ratio 100%. Τέλος, τα αποτελέσματα συγκρίθηκαν με μετρήσεις που έγιναν χρησιμοποιώντας τα πρωτόκολλα TCP και UDP.

The primary goal of this thesis is the integration of the ICN (Information Centric Networking) and DTN (Delay Tolerant Networking) architectures into a single protocol stack for use in Android devices. The unification of ICN and DTN is specified and will be used by the UMOBILE architecture with the objective to provide reliable services and content distribution in challenged network environments (i.e. intermittent connectivity, lack of infrastructure). An application has been developed in order to test and evaluate the new protocol stack in real network scenarios and compare it to TCP and UDP. It has been used as a platform for the collection of experimental data under normal or intermittent connectivity and in both cases the packet delivery ratio was 100%, achieving full reliability.

Περιεχόμενα

Κεφάλαιο 1 Εισαγωγή 1
Κεφάλαιο 2 Βασικά Στοιχεία Πληροφοριοκεντρικών Δικτύων – Το Πρωτόκολλο NDN 3
2.1 Βασικές Έννοιες ICN 3
2.1.1 Ονοματολογία 3
2.1.2 Δρομολόγηση και Ανάλυση Ονόματος 4
2.1.3 Προσωρινή Αποθήκευση (Caching) 4
2.1.4 Φορητότητα 5
2.1.5 Ασφάλεια 6
2.2 Η αρχιτεκτονική DONA 7
2.4 Η αρχιτεκτονική NDN 9
2.4.1 Ονοματολογία 12
2.4.2 Δρομολόγηση και Προώθηση 12
2.4.3 Ασφάλεια 13
2.4.4 Προσωρινή Αποθήκευση 13
Κεφάλαιο 3 Η Αρχιτεκτονική UMOBILE 14
Κεφάλαιο 4 Διασύνδεση NDN - DTN 20
4.1 NDN Forwarding Daemon 20
4.2 Υλοποίηση 23
Κεφάλαιο 5 Πειραματικά Αποτελέσματα 28
5.1 Μεθοδολογία 28
5.2 Αποτελέσματα Πειραμάτων 32
5.2.1 Συμπεριφορά του DTN face 33
5.2.2 Σύγκριση με UDP και TCP 40
5.2.3 Χαρακτηριστικά απόδοσης 49
Κεφάλαιο 6 Συμπεράσματα και Προοπτικές 59
Βιβλιογραφία 61
Παράρτημα 1 63
Παράρτημα 2 80

Κεφάλαιο 1

Εισαγωγή

Το Διαδίκτυο έχει αλλάξει το πρόσωπο της ανθρωπότητας τον 21ο αιώνα τροφοδοτώντας ραγδαίες αλλαγές στη δικτύωση, την επικοινωνία, την κατανάλωση περιεχομένου και γενικά, σε όλες τις εκφάνσεις της ανθρώπινης δραστηριότητας. Το σημερινό Διαδίκτυο εξυπηρετεί πολύ διαφορετικές και διευρυμένες ανάγκες σε σχέση με την αρχική του σύλληψη που περιέγραφε ένα δίκτυο τηλεπικοινωνιών, οι εφαρμογές του οποίου περιορίζονταν σε ανταλλαγή δεδομένων μεταξύ κόμβων. Σήμερα όμως, οι χρήστες του Διαδικτύου αναζητούν εικόνες, βίντεο, περιεχόμενο μέσων κοινωνικής δικτύωσης, live events και πολλά άλλα. Ουσιαστικά δεν ενδιαφέρονται για την τοποθεσία του παραγωγού αλλά για την ίδια την πληροφορία. Ασφαλώς, η αρχιτεκτονική του Διαδικτύου εξελίχθηκε ώστε να ανταποκριθεί στις προκλήσεις της επεκτασιμότητας (εξάντληση IP διευθύνσεων), της ασφάλειας, της φορητότητας κ.τ.λ. Παρ' όλα αυτά, η προσαρμογή αυτή έχει αυξήσει την πολυπλοκότητα και έχει δώσει προσωρινές λύσεις [2, 3].

Σήμερα έχει σχηματιστεί μία ερευνητική κοινότητα γύρω από την ανάγκη υπερκέρασης των υπαρχόντων προβλημάτων, στόχος της οποίας είναι ο προσδιορισμός της αρχιτεκτονικής του μελλοντικού Διαδικτύου. Η *Πληροφοριοκεντρική Δικτύωση* (*Information-Centric Networking* – ICN) έχει αναδυθεί ως ένας πολλά υποσχόμενος υποψήφιος σε αυτή την πορεία. Τα πληροφοριοκεντρικά δίκτυα λειτουργούν σαν δίκτυα διανομής. Σε αυτά, δεν ονοματίζονται οι κόμβοι αλλά τα ίδια τα δεδομένα. Έτσι, οι καταναλωτές δεν χρειάζεται να γνωρίζουν τη διεύθυνση του διανομέα αλλά το όνομα της πληροφορίας που επιθυμούν. Ένα πλεονέκτημα της δομικής αυτής αλλαγής είναι ότι καθίσταται δυνατή η προσωρινή αποθήκευση (caching) των δεδομένων σε οποιοδήποτε τμήμα του δικτύου, γεγονός που αυξάνει την αποδοτικότητα.

Η παρούσα διπλωματική εργασία εντάσσεται στο πλαίσιο της αρχιτεκτονικής UMOBILE [5]. Ο βασικός στόχος του UMOBILE (*universal, mobile-centric and opportunistic communications architecture*) είναι η ανάπτυξη μίας αρχιτεκτονικής με κέντρο τη φορητότητα για την αξιόπιστη παροχή περιεχομένου και υπηρεσιών σε τελικούς χρήστες παρά τις ελλείψεις ή δυσκολίες οι οποίες μπορεί να παρουσιάζονται στη δικτυακή υποδομή. Για την επίτευξη αυτού του στόχου το UMOBILE, μεταξύ άλλων, συνδυάζει την Πληροφοριοκεντρική Δικτύωση με τη δικτύωση η οποία είναι ανεκτική σε καθυστερήσεις (Delay Tolerant Networking – DTN). Έτσι, η πλατφόρμα

UMOBILE παρέχει μια νέα κλάση υπηρεσιών (less-than-best-effort) και αξιοπιστία σε συνθήκες προβληματικού δικτύου ή συμφόρησης [5].

Στο πλαίσιο της παρούσας εργασίας υλοποιήθηκε η ενσωμάτωση ICN και DTN για κινητές συσκευές με το λειτουργικό σύστημα Android. Η υλοποίηση δοκιμάστηκε και συγκρίθηκε σε συνθήκες προβληματικού ασύρματου δικτύου στο οποίο παρουσιαζόταν διαλείπουσα συνδεσιμότητα. Για το σκοπό αυτό αναπτύχθηκε μία εφαρμογή η οποία λειτούργησε ως πλατφόρμα για τη διεξαγωγή πειραμάτων και την καταγραφή αποτελεσμάτων σε πραγματικά δίκτυα. Το νέο αυτό protocol stack, σε όλες τις δοκιμές που πραγματοποιήθηκαν, παρέχει πλήρη αξιοπιστία με packet delivery ratio 100%.

Η δομή της εργασίας είναι η εξής:

Στο 2ο κεφάλαιο παρουσιάζονται τα κύρια χαρακτηριστικά της πληροφοριοκεντρικής δικτύωσης και αναπτύσσεται ως παράδειγμα η αρχιτεκτονική DONA. Ακόμη, αναλύεται η αρχιτεκτονική NDN η οποία χρησιμοποιείται στο πρόγραμμα UMOBILE.

Στο 3ο κεφάλαιο περιγράφεται η αρχιτεκτονική UMOBILE και ορίζονται οι προδιαγραφές που αυτή θέτει για τη διασύνδεση NDN και DTN καθώς και τα σενάρια στα οποία θα χρησιμοποιηθεί το νέο αυτό protocol stack.

Στο 4ο κεφάλαιο εξετάζεται ο τρόπος λειτουργίας του NDN Forwarding Daemon (NFD). Στη συνέχεια αναλύεται η υλοποίηση της διασύνδεσης NDN – DTN.

Στο 5ο κεφάλαιο παρουσιάζονται τα πειραματικά αποτελέσματα τα οποία εξήχθησαν. Αυτά περιλαμβάνουν τη συμπεριφορά του νέου protocol stack σε συνθήκες διακοπτόμενης σύνδεσης, τη σύγκρισή του με τα πρωτόκολλα TCP και UDP καθώς και την ανάλυση της απόδοσής του.

Τέλος στο 6ο κεφάλαιο συμπυκνώνονται τα συμπεράσματα που προέκυψαν από την υλοποίηση και τη διεξαγωγή των πειραμάτων ενώ αναφέρονται πιθανές μελλοντικές επεκτάσεις.

Στο παράρτημα 1 παρατίθεται ο κώδικας της διασύνδεσης NDN-DTN και στο παράρτημα 2 αντίστοιχα ο κώδικας της πλατφόρμας διεξαγωγής πειραμάτων.

Κεφάλαιο 2

Βασικά Στοιχεία Πληροφοριοκεντρικών Δικτύων – Το Πρωτόκολλο NDN

Σε αυτό το κεφάλαιο θα παρουσιαστούν τα κύρια χαρακτηριστικά της πληροφοριοκεντρικής δικτύωσης και θα περιγραφεί μία από τις πρώτες ICN υλοποιήσεις, η DONA. Ακόμη θα αναλυθεί η αρχιτεκτονική NDN η οποία χρησιμοποιείται στο πρόγραμμα UMOBILE.

2.1 Βασικές Έννοιες ICN

Οι βασικές επιλογές που έχουν γίνει κατά τη σχεδίαση των διάφορων ICN αρχιτεκτονικών αφορούν κυρίως: τη μορφή των ονομάτων, της δρομολόγηση και την προώθηση, τη λειτουργία της προσωρινής αποθήκευσης, την επίτευξη φορητότητας (mobility) και την ασφάλεια.

2.1.1 Ονοματολογία

Στην κλασσική δικτύωση IP ο παραλήπτης πρέπει να γνωρίζει το όνομα της πληροφορίας αλλά και τη διεύθυνση του κατόχου της. Έτσι, στη βασική της μορφή, η αίτηση για την πρόγνωση καιρού της εβδομάδας έχει τη μορφή 78.109.124.42/weather-forecast/week όπου το πρώτο τμήμα είναι η IP διεύθυνση του server και το δεύτερο είναι το όνομα της πληροφορίας. Ευτυχώς, το Domain Name System (DNS) μας έχει απαλλάξει από την ανάγκη να γνωρίζουμε τις διευθύνσεις των server με περιεχόμενο που μας ενδιαφέρει ενώ παράλληλα έχει εισάγει αναγνώσιμες από ανθρώπους διευθύνσεις.

Στις αρχιτεκτονικές ICN η πληροφορία είναι πλήρως αποσυζευγμένη [2] από την πηγή της. Έτσι ο παραλήπτης δεν έχει παρά να εκφράσει το ενδιαφέρον του για weather-forecast/week. Το ίδιο το δίκτυο είναι υπεύθυνο για τον εντοπισμό και την παράδοση των δεδομένων πίσω στο χρήστη. Στις διάφορες υλοποιήσεις ICN τα μοναδικά ονόματα των δεδομένων διαφέρουν ως προς τη δομή τους. Σε κάποιες είναι flat (π.χ. yf$db@n3dn) ενώ σε άλλες είναι ιεραρχικά (π.χ. weather-forecast/week/51) ενώ και το αν θα πρέπει αν είναι αναγνώσιμα από ανθρώπους είναι αντικείμενο μελέτης. Σε κάθε περίπτωση το όνομα δεν εξαρτάται από την τοποθεσία της πηγής γεγονός που επιτρέπει την εκ φύσεως αποθήκευση των δεδομένων οπουδήποτε στο δίκτυο. Μία άλλη συνέπεια της πληροφοριοκεντρικής δικτύωσης είναι ότι η επικοινωνία ξεκινάει και ελέγχεται από τον παραλήπτη αφού, εάν αυτός δεν εκφράσει

το ενδιαφέρον του (δηλαδή να στείλει ένα πακέτο τύπου *interest*) δεν μπορεί να λάβει δεδομένα (δηλαδή ένα πακέτο τύπου *data*) γεγονός που έχει διάφορες προεκτάσεις σχετικά με την ασφάλεια όπως θα δούμε στη συνέχεια.

2.1.2 Δρομολόγηση και Ανάλυση Ονόματος

Στις αρχιτεκτονικές ICN η δρομολόγηση λειτουργεί παρόμοια με αυτή του σημερινού Διαδικτύου, με τη διαφορά ότι στους πίνακες δρομολόγησης βρίσκονται ονόματα δεδομένων αντί για διευθύνσεις κόμβων. Το αν αυτή η λογική μπορεί να κλιμακωθεί στο μέγεθος του Διαδικτύου είναι ένα ανοιχτό ζήτημα αφού τα δεδομένα είναι σαφώς περισσότερα από τους κόμβους. Εκεί που οι αρχιτεκτονικές ICN διαφοροποιούνται σημαντικά μεταξύ τους είναι στην Ανάλυση Ονόματος. Ο όρος είναι μία προσπάθεια απόδοσης του όρου *Name Resolution* που περιλαμβάνει την αντιστοίχιση του ονόματος της πληροφορίας σε κάποια πηγή. Για την ανάλυση ονόματος υπάρχουν δύο διαφορετικές προσεγγίσεις, η συζευγμένη και η αποσυζευγμένη – σε σχέση με τη δρομολόγηση. Κατά τη συζευγμένη, το πακέτο δεδομένων ακολουθεί προς τα πίσω την ίδια διαδρομή που ακολούθησε το πακέτο το οποίο εκδήλωσε το ενδιαφέρον. Στην αποσυζευγμένη προσέγγιση δεν υπάρχει αυτός ο περιορισμός.

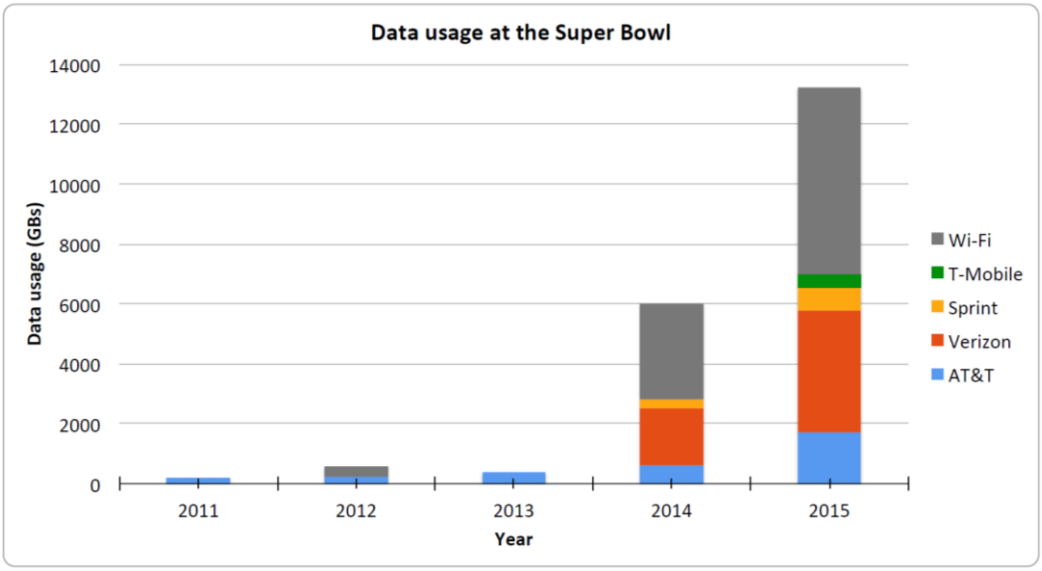
2.1.3 Προσωρινή Αποθήκευση (Caching)

Στις αρχιτεκτονικές ICN διευκολύνεται από σχεδιασμού η προσωρινή αποθήκευση δεδομένων οπουδήποτε στο δίκτυο χωρίς την ανάγκη χρήσης εξωτερικών λύσεων. Τα πακέτα δεδομένων μπορούν να αποθηκευτούν και να ανακληθούν εύκολα χωρίς την επιβάρυνση που συνεπάγονται τεχνικές που χρησιμοποιούνται σήμερα όπως το *Deep Packet Inspection.*

Το γεγονός αυτό επιτρέπει την ομαδοποίηση των αιτήσεων για την ίδια μονάδα δεδομένων. Ας πάρουμε το παράδειγμα ενός σταδίου με χιλιάδες θεατές που παρακολουθούν κάποιο άθλημα και χρησιμοποιούν τα κινητά τους τηλέφωνα και καταναλώνουν ολοένα και μεγαλύτερες ποσότητες δεδομένων όπως φαίνεται στο σχήμα 1. Συχνά οι δικτυακές υποδομές των εγκαταστάσεων δεν επαρκούν και η χρήση του Internet μέσω κινητού δικτύου είναι περιορισμένη λόγω έλλειψης εύρους ζώνης. Είναι πολύ πιθανό ένα μέρος αυτής της κίνησης να αφορά στα ίδια δεδομένα. Σε αντίθεση με άλλες λύσεις ([6]) οι αρχιτεκτονικές ICN μπορούν να περιορίσουν το φόρτο του δικτύου χάρη στην από σχεδιασμού υποστηριζόμενη προσωρινή αποθήκευση αφού τα δημοφιλή δεδομένα θα μπορούν να ανακαλούνται τοπικά. Μία λύση που χτίζει πάνω στις δυνατότητες του ICN είναι το KEBAPP Framework [7]

στο οποίο οι εφαρμογές ορίζουν ιεραρχικά πεδία ονομάτων (namespaces) ώστε να είναι δυνατή η ανάκτηση επεξεργασμένων πληροφοριών σε ευκαιριακά σενάρια.

Στις διάφορες αρχιτεκτονικές ICN διακρίνονται δύο τύποι caching, ο εντός και ο εκτός πορείας. Στην πρώτη περίπτωση το δίκτυο μπορεί να αξιοποιήσει μόνο δεδομένα που βρίσκονται εντός της πορείας που ακολουθεί η αίτηση για αυτά Στη δεύτερη περίπτωση είναι δυνατή η ανάκληση δεδομένων και εκτός διαδρομής. Για τις αποσυζευγμένες αρχιτεκτονικές, το εκτός-πορείας caching υποστηρίζεται με την αντιμετώπιση των προσωρινών μνημών ως κανονικούς παραγωγούς. Για τις δε συζευγμένες, πρέπει να υποστηρίζεται από το σύστημα δρομολόγησης.



Σχήμα 1: Η αυξανόμενη χρήση Wi-Fi και cellular δεδομένων μέσω κινητών συσκευών σε παιχνίδια του NFL Super Bowl στις ΗΠΑ. [6]

2.1.4 Φορητότητα

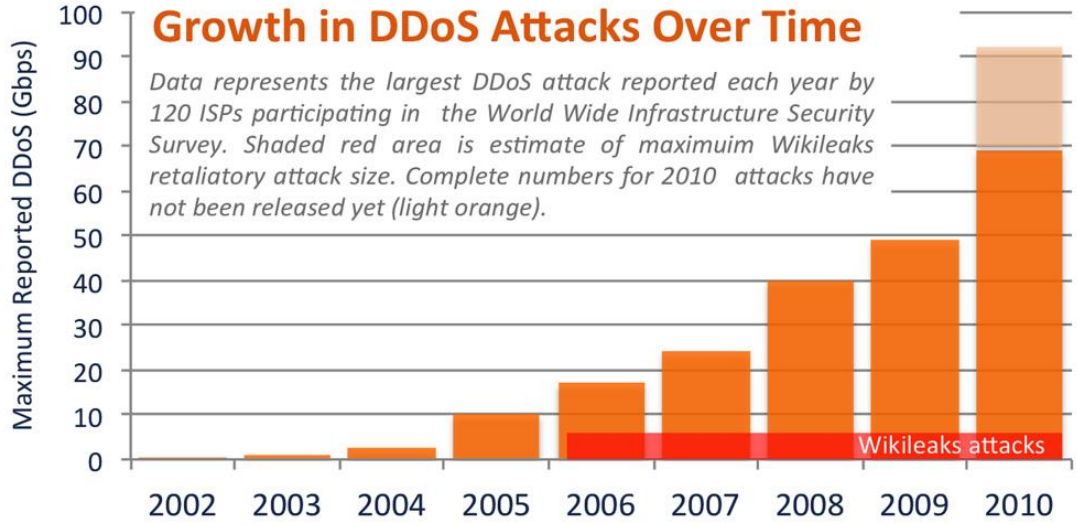
Το Διαδίκτυο στη σημερινή του μορφή υποθέτει τη σύνδεση από άκρο σε άκρο για να επιτευχθεί επικοινωνία. Όμως ο συνεχώς αυξανόμενος αριθμός των φορητών συσκευών που είναι συνδεδεμένες (οι οποίες το 2015 ξεπέρασαν τις σταθερές συσκευές [8]) δημιουργεί προβλήματα αφού αυτές οι συσκευές ενδέχεται να αλλάζουν συχνά δίκτυο, αρά και διεύθυνση IP. Το πρόβλημα αυτό έχει αντιμετωπιστεί με τη εισαγωγή του Mobile IP πρωτοκόλλου το οποίο όμως δεν είναι αποδοτικό.

Αντίθετα, η πληροφοριοκεντρική δικτύωση χρησιμοποιεί ένα μοντέλο publish/subscribe (δημοσίευση/εγγραφή). Με βάση αυτό το μοντέλο οι πάροχοι διαφημίζουν την πληροφορία που προσφέρουν και οι παραλήπτες εκφράζουν το ενδιαφέρον τους για αυτή. Η διαδικασία αυτή είναι ασύγχρονη. Δηλαδή ένας πάροχος μπορεί να διαφημίσει την πληροφορία που διαθέτει ακόμη και αν δεν έχει εκφραστεί

ενδιαφέρον για αυτή ενώ ο παραλήπτης μπορεί αιτηθεί πληροφορίας που έχει διαφημιστεί στο παρελθόν. Η μη ανάγκη συγχρονισμού των δύο άκρων επιτρέπει στους φορητούς κόμβους απλά να επανεκδίδουν την αίτηση ενδιαφέροντος η οποία μάλιστα ενδέχεται να μπορεί να εξυπηρετηθεί και τοπικά λόγω caching.

2.1.5 Ασφάλεια

Το Διαδίκτυο δεν σχεδιάστηκε με γνώμονα την ασφάλεια, καθώς στην αρχική του μορφή συνέδεε έμπιστους κόμβους μεταξύ τους (πανεπιστήμια, στρατιωτικές εγκαταστάσεις). Στη σημερινή, εμπορευματοποιημένη μορφή του, παρά τις προσθήκες και επεκτάσεις σχετικά με την ασφάλεια, το Διαδίκτυο είναι χώρος κλοπής προσωπικών δεδομένων, οικονομικής απάτης, επιθέσεων denial of service, ακόμη και πεδίο «μάχης» μεταξύ χωρών. Η αιτία είναι η μεγάλη ισχύς που διαθέτει ο αποστολέας σε σχέση με τον παραλήπτη. Αυτή η ανισορροπία είναι που επιτρέπει τη διεξαγωγή επιθέσεων denial of service οι οποίες έχουν συνεχώς αυξητικές τάσεις (σχήμα 2).

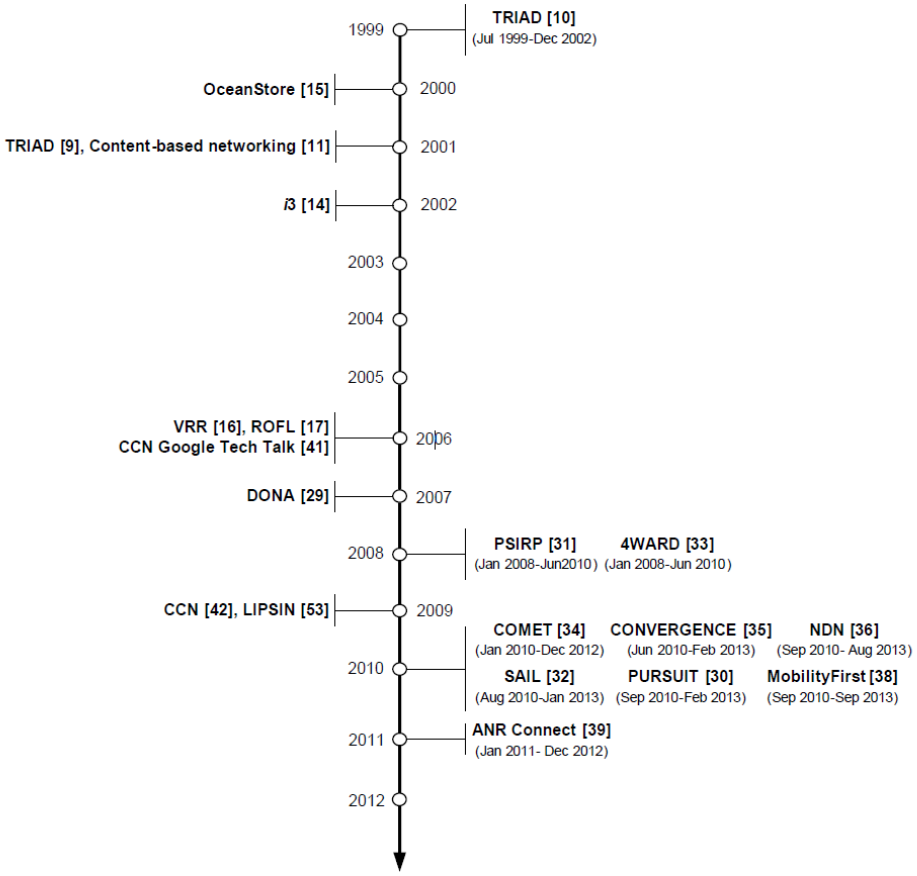


Σχήμα 2: Η εξέλιξη του μεγέθους των επιθέσεων τύπου denial of service [9]

Η πληροφοριοκεντρική δικτύωση λειτουργεί με βάση την έκφραση ενδιαφέροντος. Έτσι, ροή δεδομένων υπάρχει μόνο όταν έχει προηγουμένως εκφραστεί ενδιαφέρον γι' αυτά. Το χαρακτηριστικό αυτό αναμένεται να περιορίσει τις ανεπιθύμητες μεταφορές δεδομένων, όπως τα spam [2]. Ακόμη, οι αρχιτεκτονικές που χρησιμοποιούν επίπεδα (flat) ονόματα, τα οποία μπορούν να χρησιμοποιηθούν για την πιστοποίηση των περιεχομένων, μπορούν να βασίζονται σε αυτά ώστε να απορρίπτουν κακόβουλα δεδομένα σε επίπεδο δικτύου.

2.2 Η αρχιτεκτονική DONA

Σε αυτό το υποκεφάλαιο θα παρουσιαστεί η πληροφοριοκεντρική αρχιτεκτονική DONA ώστε να αποσαφηνιστούν οι έννοιες του κεφαλαίου2.1 και να τεθεί ένα σημείο αναφοράς για την αρχιτεκτονική NDN που χρησιμοποιείται στο UMOBILE και θα αναλυθεί στη συνέχεια.



Σχήμα 3: Το χρονοδιάγραμμα των ICN αρχιτεκτονικών.

Η *Data Oriented Network Architecture* (DONA) που δημιουργήθηκε στο UC Berkeley είναι από τις πρώτες ολοκληρωμένες αρχιτεκτονικές ICN. Η DONA διατηρεί την IP διευθυνσιοδότηση και δρομολόγηση ενώ προσθέτει τη δυνατότητα ανάλυσης ονόματος (name resolution), η οποία αντιστοιχεί τα flat ονόματα στην πληροφορία, πάνω από την υπάρχουσα υποδομή. [2]

Τα flat ονόματα στην αρχιτεκτονική DONA χρησιμοποιούνται και για τον έλεγχο της ακεραιότητας των δεδομένων που παρελήφθησαν. Έτσι τα ονόματα αποτελούνται από το κρυπτογραφικό hash του παραγωγού τους (P) και ένα μοναδικό αναγνωριστικό (L) που προσδιορίζει το πακέτο μοναδικά στο πλαίσιο του συγκεκριμένου παραγωγού.

Η ανάλυση ονόματος (name resolution) γίνεται από εξειδικευμένους servers που ονομάζονται *Resolution Handlers* (στο εξής RH). Όποιος επιθυμεί να

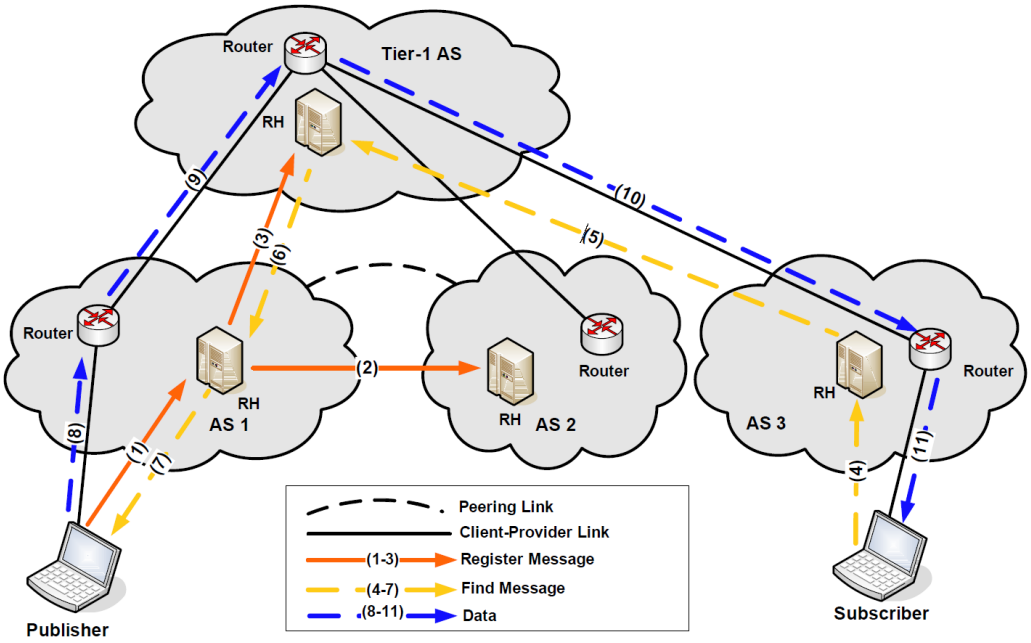
δημοσιεύσει δεδομένα αποστέλλει ένα μήνυμα REGISTER με το όνομα τους σε κοντινό του RH. Ο τελευταίος κοινοποιεί αυτή την καταχώρηση στους γειτονικούς του RH με σκοπό αυτή να διαδοθεί στα ανώτερα στρώματα της ιεραρχίας (Οι RH μπορούν να είναι τοπικοί ή μεγαλύτερης κλίμακας σχηματίζοντας έτσι μία ιεραρχία). Ο καταναλωτής αποστέλλει ένα μήνυμα FIND στον κοντινό του RH ο οποίος είτε προωθεί το μήνυμα προς τον κάτοχο της πληροφορίας (εάν τον έχει καταχωρημένο) είτε προς άλλους RH.

Η δρομολόγηση μπορεί να είναι συζευγμένη ή αποσυζευγμένη από την ανάλυση ονόματος. Στη δεύτερη περίπτωση τα δεδομένα μπορούν να αποσταλούν στον καταναλωτή από τον κάτοχο με τις κλασσικές μεθόδους του πρωτοκόλλου IP. Αυτή όμως η στρατηγική επηρεάζεται από το πρόβλημα της εξάντλησης των διευθύνσεων IPv4. Για το λόγο αυτό υπάρχει και η δυνατότητα της συζευγμένης δρομολόγησης. Σε αυτή, η διαδρομή του μηνύματος FIND καταγράφεται σε αυτό και μπορεί να χρησιμοποιηθεί για την προώθηση των δεδομένων μέσω της αντίστροφης διαδρομής. Έτσι μπορούν να χρησιμοποιηθούν διευθύνσεις IP που είναι τοπικές σε κάθε αυτόνομο σύστημα (σύννεφα στο σχήμα 4).

Η προσωρινή αποθήκευση (caching) βασίζεται στους RH οι οποίοι επιλέγουν τι θα αποθηκεύσουν. Εάν ένα μήνυμα FIND αναζητά πληροφορία η οποία βρίσκεται στον RH τότε αυτή μπορεί να αποσταλεί άμεσα πίσω στον ενδιαφερόμενο.

Όσον αφορά στη φορητότητα, ένας χρήστης που αλλάζει δίκτυο μπορεί να εκδώσει ένα καινούριο μήνυμα FIND. Όταν ένας παραγωγός πληροφορίας αλλάζει θέση, οι σχετικές καταχωρήσεις που έχουν οι RH πρέπει να ανανεωθούν το οποίο είναι αρκετά πιο πολύπλοκη και δαπανηρή διαδικασία.

Η DONA υποστηρίζει τον έλεγχο της ακεραιότητας των δεδομένων με βάση το όνομα. Εάν τα δεδομένα δεν είναι μεταβλητά (non-mutable), τότε το μοναδικό αναγνωριστικό L που βρίσκεται στο όνομα είναι το κρυπτογραφικό hash των δεδομένων. Άρα ο παραλήπτης μπορεί να παράξει το hash από τα δεδομένα που έλαβε και να το συγκρίνει με αυτό που βρίσκεται στο όνομα. Στην περίπτωση που τα δεδομένα είναι μεταβλητά (mutable), το μοναδικό αναγνωριστικό L δεν έχει προκύψει απ’ ευθείας από τα δεδομένα. Το δημόσιο κλειδί του ιδιοκτήτη των δεδομένων, το οποίο χρησιμοποιήθηκε για την παραγωγή του τμήματος P του ονόματος, περιλαμβάνεται στο πακέτο ως meta-data μαζί με ένα hash που έχει προκύψει από τα δεδομένα. Άρα μπορεί να ελεγχθεί και η ακεραιότητα και η προέλευση με βάση τα δύο αυτά στοιχεία.



Σχήμα 4: Η αρχιτεκτονική DONA διαγραμματικά. [2]

2.4 Η αρχιτεκτονική NDN

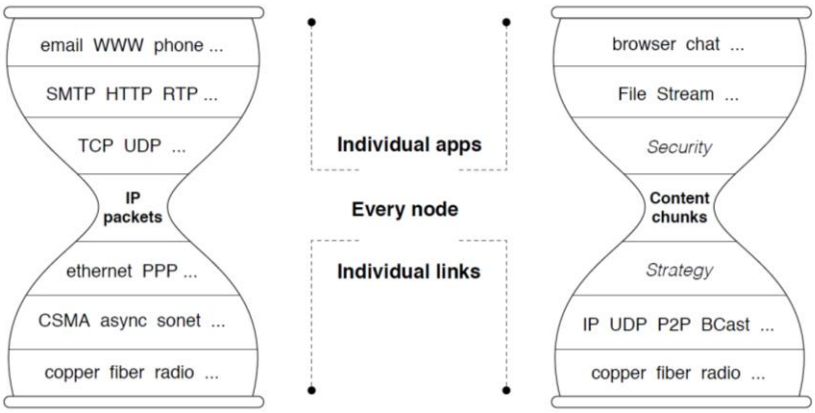
Η αρχιτεκτονική *Named Data Networking* (NDN[1]) βασίζεται στην αρχιτεκτονική *Content Centric Networking* (CCN[2]) και η ανάπτυξη της χρηματοδοτείται από το US Future Internet Architecture Program [2]. Το όραμα της αρχιτεκτονικής NDN είναι να γενικεύσει το ρόλο της «λεπτής μέσης» (narrow waist) της δικτυακής στοίβας του σημερινού Internet που καταλαμβάνεται από το πρωτόκολλο IP [1]. Η δομή αυτή περιγράφεται και από τον όρο *κλεψύδρα* (hourglass) διότι όλη η δικτυακή κίνηση διέρχεται μέσω της λεπτής μέσης (σχήμα 6).

Η λογική της αρχιτεκτονικής NDN είναι η αντικατάσταση του IP, που ορίζει ένα δίκτυο τηλεπικοινωνιών όπου η επικοινωνία διεξάγεται μεταξύ κόμβων, από μία εξελιγμένη μορφή του, σύμφωνα με την οποία τα πακέτα πληροφορίας φέρουν το όνομα αντικειμένων και όχι αυτό των ακραίων σημείων επικοινωνίας. Τα αντικείμενα αυτά μπορεί να είναι τμήματα δεδομένων, ένα βιβλίο, ένα τμήμα ταινίας κ.α. – ακόμη και κόμβοι. Με αυτό τον τρόπο το Διαδίκτυο μετατρέπεται σε δίκτυο διανομής (distribution network). Έτσι διατηρείται η τωρινή λειτουργικότητα και διευκολύνεται η επίλυση μίας ευρείας γκάμας προβλημάτων. Ακόμη η αρχιτεκτονική NDN

[1] https://named-data.net/

[2] https://en.wikipedia.org/wiki/Content_centric_networking

ενσωματώνει μεθόδους ασφαλείας (υπογραφές στα πακέτα δεδομένων) και έλεγχο συμφόρησης.



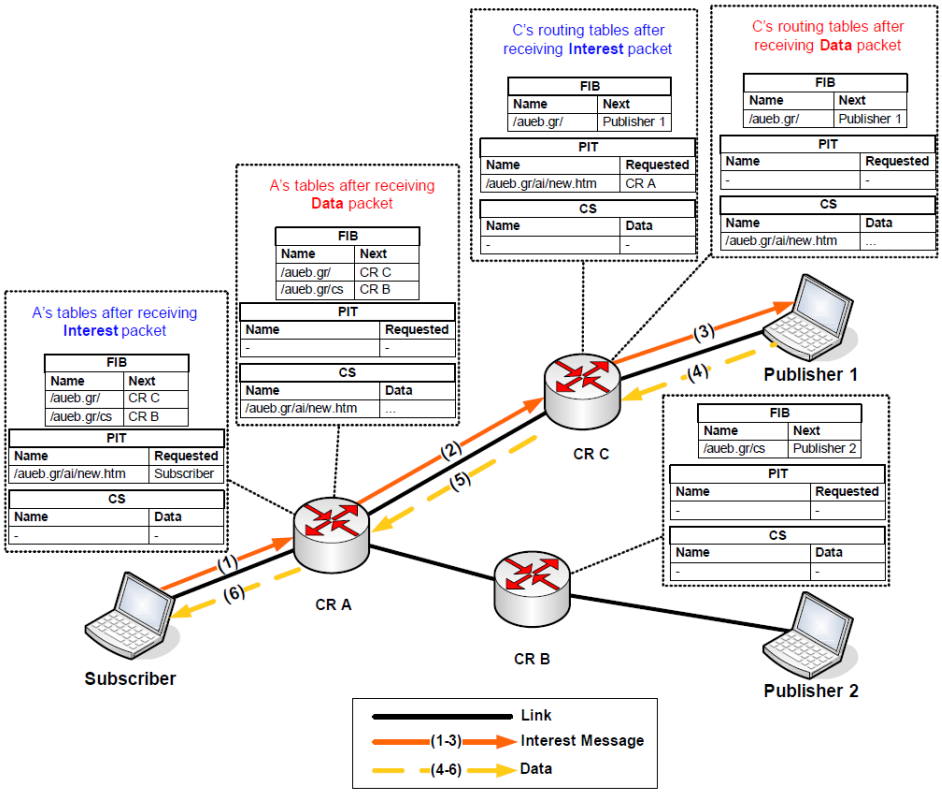
Σχήμα 6: Η γενίκευση της λεπτής μέσης της Διαδικτυακής στοίβας.

Μεταξύ του καταναλωτή και του διακομιστή ανταλλάσσονται δύο τύποι πακέτων: *Interest* και *Data* τα οποία φέρουν ένα όνομα που ταυτοποιεί δεδομένα τα οποία μπορούν να μεταφερθούν σε ένα πακέτο δεδομένων (Data). Ο καταναλωτής προωθεί ένα πακέτο Interest με το όνομα των δεδομένων που επιθυμεί στο δίκτυο. Όταν το πακέτο φτάσει σε κάποιο δρομολογητή αυτός ακολουθεί την εξής διαδικασία: Αρχικά ελέγχει εάν διαθέτει το πακέτο δεδομένων στην προσωρινή του μνήμη (cache) η οποία ονομάζεται *Content Store* (CS). Εάν το βρει, το επιστρέφει στον ενδιαφερόμενο. Διαφορετικά, εξετάζει εάν έχει καταχωρημένο ενδιαφέρον για τα ίδια δεδομένα στη δομή *Pending Interest Table* (PIT). Στη δομή αυτή ο δρομολογητής αποθηκεύει τα interests που δεν έχει εξυπηρετήσει τοπικά, έχει προωθήσει σε άλλους δρομολογητές ή διακομιστές και για τα οποία δεν έχει λάβει Data. Εάν υπάρχει ήδη τέτοια καταχώρηση, ο δρομολογητής απλά προσθέτει τον συγκεκριμένο καταναλωτή σε αυτή (με βάση το όνομα του Interest). Σε περίπτωση που κανένα από τα παραπάνω δεν ισχύει, ο δρομολογητής προωθεί το Interest προς το διακομιστή με βάση μία τρίτη δομή, την *Forwarding Information Base* (FIB), αλλά και σύμφωνα με τη στρατηγική προώθησής του (Forwarding Strategy). Ο πίνακας FIB δημιουργείται με τη χρήση ενός πρωτοκόλλου δρομολόγησης που βασίζεται σε τμήματα ονομάτων (name prefixes) και μπορεί να έχει πολλαπλές καταχωρήσεις για κάθε όνομα. Όταν λαμβάνονται πολλά Interests για τα ίδια δεδομένα προωθείται μόνο ένα προς την κατεύθυνση του παραγωγού ή κάποιου κατόχου.

Η στρατηγική προώθησης είναι ένα σημαντικό τμήμα της αρχιτεκτονικής NDN. Μεταξύ άλλων, είναι υπεύθυνη για την προσπέλαση του FIB και την εξεύρεση του καλύτερου ταιριάσματος με τη λογική του *longest prefix*. Μπορεί ακόμη να

υπαγορεύσει την απόρριψη ενός Interest σε περίπτωση συμφόρησης ή εάν υπάρχει υποψία κακόβουλης επίθεσης DoS.

Όταν ένας δρομολογητής λάβει ένα πακέτο δεδομένων (Data) το προωθεί σύμφωνα με τον πίνακα PIT (Pending Interest Table) προς την κατεύθυνση ενός ή περισσότερων παραληπτών ενώ ταυτόχρονα διαγράφει την καταχώρηση και αποθηκεύει τα δεδομένα στο CS (Content Store). Εάν δεν υπάρχει απώλεια πακέτων ο δρομολογητής θα λάβει ένα πακέτο Data για κάθε πακέτο Interest που προώθησε αφού τα δεδομένα ακολουθούν την αντίστροφη πορεία από αυτή των Interests. Στην περίπτωση μεγάλου όγκου δεδομένων, αυτός χωρίζεται σε πολλά ζευγάρια Interest-Data. Με αυτό το τρόπο επιτυγχάνεται και ο έλεγχος της ροής της κίνησης καθώς τα Interest λειτουργούν παρόμοια με τα ACKs στο TCP.



Σχήμα 7: Η αρχιτεκτονική NDN. CR: Content Router, FIB: Forwarding Information Base, PIT: Pending Interest Table, CS: Content Store. [1]

2.4.1 Ονοματολογία

Τα ονόματα στην αρχιτεκτονική NDN είναι ιεραρχικά. Για παράδειγμα /somenewsnetwork/economics/stockmarkets/today.html/1 όπου το «/1» αντιπροσωπεύει το πρώτο τμήμα της σελίδας και αντιστοιχεί σε ένα πακέτο Data. Η ιεραρχική δομή διευκολύνει την κλιμάκωση του συστήματος δρομολόγησης και επιτρέπει στις εφαρμογές να ορίζουν τις σχέσεις μεταξύ των δεδομένων [1]. Ακόμη, τα ονόματα είναι αδιαφανή για το δίκτυο, γεγονός που επιτρέπει την εξέλιξη των εφαρμογών ανεξάρτητα από το αυτό.

Οι καταναλωτές πρέπει να είναι σε θέση να κατασκευάζουν το όνομα για το τμήμα δεδομένων που επιθυμούν με ντετερμινιστικό τρόπο (*deterministically*). Αυτό μπορεί να επιτευχθεί με δύο τρόπους. Ο πρώτος είναι με τη χρήση ενός ντετερμινιστικού αλγορίθμου (*deterministic algorithm*) με βάση τον οποίο ο καταναλωτής και ο παραγωγός καταλήγουν στο ίδιο όνομα με βάση τις πληροφορίες που είναι διαθέσιμες και στους δύο. Η δεύτερη μέθοδος συνδυάζει τη λογική longest prefix και ένα πεδίο στο πακέτο Interest που ονομάζεται *Interest Selector*. Για παράδειγμα ο καταναλωτής εκδίδει Interest για το βίντεο welcome.mpg με όνομα duth/info/welcome.mpg και το συμπληρώνει με τον Interest Selector “leftmost child” [1] (ουσιαστικά ζητώντας το πρώτο τμήμα των δεδομένων). Ο παραγωγός των δεδομένων ενδεχομένως θα στείλει ένα πακέτο δεδομένων με όνομα duth/info/welcome.mpg/1. Με βάση αυτή την πληροφορία ο καταναλωτής μπορεί πλέον να εκδώσει Interest με όνομα duth/info/welcome.mpg/2.

Τέλος, η αρχιτεκτονική NDN επιτρέπει τη χρήση είτε τοπικών (local) είτε καθολικών (global) ονομάτων. Τα δεδομένα που πρέπει να είναι προσπελάσιμα παγκοσμίως απαιτούν μοναδικά ονόματα σε αυτό το πλαίσιο. Όμως η χρήση τοπικών ονομάτων απλοποιεί τη δημιουργία εφαρμογών και επαρκεί σε περιπτώσεις που αυτές περιορίζονται σε τοπική δρομολόγηση.

2.4.2 Δρομολόγηση και Προώθηση

Η δρομολόγηση στην αρχιτεκτονική NDN μπορεί να βασιστεί στην τροποποίηση κλασσικών πρωτοκόλλων δρομολόγησης (OSFP, BGP) ώστε να διαδίδουν ονόματα δεδομένων αντί για διευθύνσεις IP. Κατά τη δρομολόγηση συμπληρώνεται με τμήματα δεδομένων (name-prefixes) δομή FIB (Forwarding Information Base).

Η διαφορά της NDN από τη σημερινή μορφή του Διαδικτύου αφορά στην προώθηση. Οι NDN δρομολογητές διατηρούν στη δομή PIT (Pending Interest Table) τα Interest που εκκρεμούν και τα διαγράφουν όταν λάβουν τα σχετικά δεδομένα ή εάν περάσει ένα διάστημα χρόνου (Timeout). Άρα οι δρομολογητές διατηρούν

πληροφορία για την κατάσταση του κάθε πακέτου σε αντίθεση με το IP πρωτόκολλο το οποίο είναι *stateless*. Η πληροφορία αυτή αξιοποιείται από τη στρατηγική προώθησης (Forwarding Strategy) του δρομολογητή ώστε να λάβει αποφάσεις σχετικά με το ποια Interests να προωθήσει και προς ποια διεπαφή, τη σειρά προτεραιότητας μεταξύ των Interests, την επιλογή εναλλακτικών διαδρομών σε περίπτωση γνωστής αποτυχίας κάποιου κόμβου κ.α. Ο δρομολογητής έχει ακόμη τη δυνατότητα να απαντήσει με NACK σε ένα Interest εάν αποφασίσει ότι δεν θα το εξυπηρετήσει το οποίο μπορεί να το αξιοποιήσει ο προηγούμενος δρομολογητής για τον εντοπισμό συμφόρησης. Άλλα πλεονεκτήματα του stateful forwarding [10] είναι η έμφυτη υποστήριξη multicasting χάρη στην αντιστοίχιση πολλών διεπαφών με κάθε καταχώρηση στη δομή PIT και η δυνατότητα ελέγχου της ροής με τον έλεγχο του αριθμού των εκκρεμών Interest.

2.4.3 Ασφάλεια

Οι παραγωγοί πληροφορίας αναμένεται να υπογράφουν κάθε πακέτο δεδομένων. Έτσι σε αντίθεση με τη στοίβα TCP/IP που αφήνει την ευθύνη της ασφάλειας στους ακραίους κόμβους, η αρχιτεκτονική NDN παρέχει ασφάλεια σε επίπεδο δικτύου. Οι καταναλωτές μπορούν να ελέγχουν την προέλευση και την ακεραιότητα των δεδομένων που λαμβάνουν. Ακόμη, χάρη στο stateful forwarding οι δρομολογητές έχουν την δυνατότητα να εντοπίζουν ευκολότερα επιθέσεις τύπου denial of service. +++

2.4.4 Προσωρινή Αποθήκευση

Οι δρομολογητές μπορούν να αποθηκεύουν πακέτα δεδομένων αφού, σύμφωνα με την αρχιτεκτονική NDN, αυτά αποτελούν ολοκληρωμένες οντότητες. Η δομή που χρησιμοποιείται για αυτό το σκοπό είναι το *Content Store* (CS). Ο δρομολογητής συμπεριφέρεται στο CS όπως θα συμπεριφερόταν σε ένα οποιοδήποτε κανάλι όσον αφορά στην ανάκτηση δεδομένων (π.χ. ισχύει ο κανόνας για το longest-prefix). Η προσωρινή αποθήκευση λειτουργεί εξαιρετικά για στατικά αρχεία αλλά και τα δυναμικά δεδομένα μπορούν να επωφεληθούν χάρη στην έμφυτη υποστήριξη multicast (π.χ. οπτικοακουστική επικοινωνία σε πραγματικό χρόνο με πολλούς συμμετέχοντες) ή σε περίπτωση επαναμετάδωσης λόγω απώλειας πακέτων [1]. Τέλος, σε αντίθεση με τα IP πακέτα, τα πακέτα δεδομένων στην αρχιτεκτονική NDN δεν περιέχουν πληροφορία για τον παραλήπτη (άλλωστε σε διαφορετική περίπτωση θα ήταν αδύνατη η προσωρινή αποθήκευση) με συνέπεια την ενισχυμένη ιδιωτικότητα.

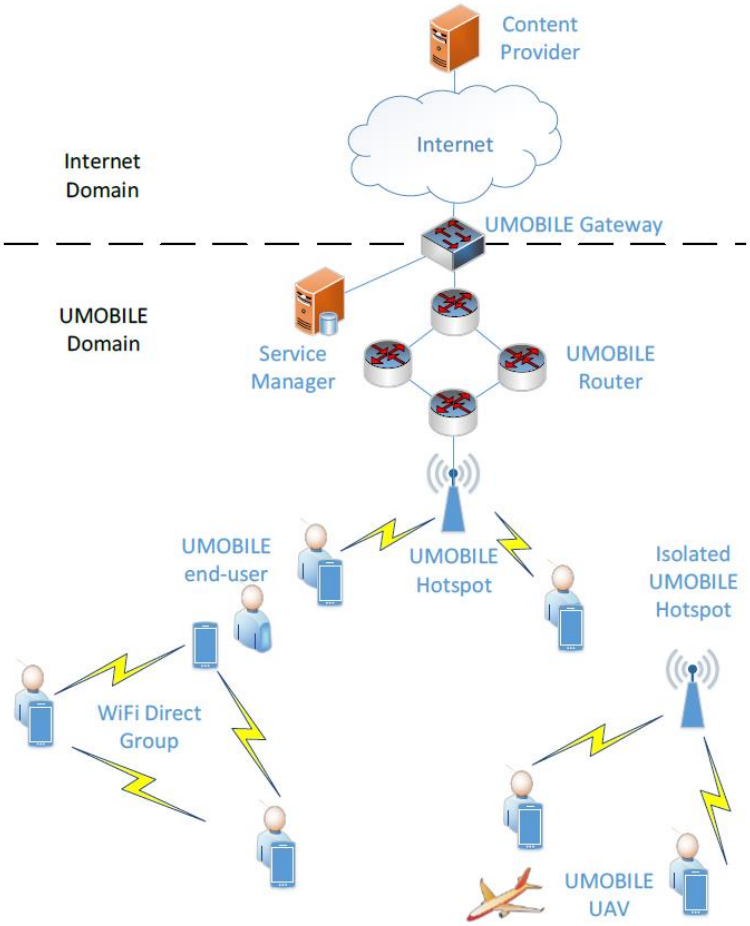
Κεφάλαιο 3

Η Αρχιτεκτονική UMOBILE

Ο κύριος στόχος του UMOBILE (Universal, mobile-centric and opportunistic communications architecture) είναι η ανάπτυξη μίας αρχιτεκτονικής με κέντρο τη φορητότητα η οποία παρέχει στους τελικούς χρήστες υπηρεσίες και περιεχόμενο με τρόπο αξιόπιστο και αποδοτικό ακόμη και σε περίπτωση αποτυχίας ή έλλειψης δικτυακής υποδομής [5]. Συγκεκριμένα, η αρχιτεκτονική εστιάζεται στη διεξαγωγή της κίνησης τοπικά για τη μείωση της καθυστέρησης και του κόστους και την επέκταση της εμβέλειας των υπηρεσιών σε περιοχές με περιορισμένη ή καθόλου υποδομή (π.χ. επείγουσες συνθήκες, απομονωμένες περιοχές).

Οι βασικοί συντελεστές της αρχιτεκτονικής και το πως διασυνδέονται φαίνεται στο σχήμα 8. Συγκεκριμένα η αρχιτεκτονική UMOBILE περιλαμβάνει:

- Φορητές συσκευές οι οποίες ανταλλάσσουν μηνύματα, φωτογραφίες κ.α. αλλά και ευκαιριακά δεδομένα (π.χ. θερμοκρασία, επίπεδα θορύβου, κ.α.).
- Hotspots τα οποία είναι σε θέση να συλλέξουν και να αναμεταδώσουν πληροφορία όπως μηνύματα κινδύνου ή οδηγίες από τις αρχές, να παρέχουν υπηρεσίες και να συλλέξουν και επεξεργαστούν δεδομένα.
- Μη επανδρωμένα – αυτόνομα οχήματα ικανά να συλλέξουν και να αναμεταδώσουν πληροφορία και να διασυνδέσουν δύο απομονωμένες περιοχές.
- Gateways τα οποία παρέχουν τη διασύνδεση μεταξύ του Διαδικτύου και του UMOBILE domain.
- Δρομολογητές οι οποίοι χρησιμοποιούν πρωτόκολλα του UMOBILE για την υποστήριξη της μεταφοράς υπηρεσιών (service migration) και την παροχή των μηχανισμών ποιότητας υπηρεσιών που ορίζονται από την αρχιτεκτονική.



Σχήμα 8: Αρχιτεκτονική UMOBILE [5]

Για την επίτευξη των παραπάνω στόχων, το UMOBILE αξιοποιεί την πληροφοριοκεντρική δικτύωση ώστε να αποδεσμεύσει τις υπηρεσίες από την αρχική τους τοποθεσία ενώ παράλληλα χρησιμοποιεί εργαλεία ευκαιριακής δικτύωσης και ανοχής στην καθυστέρηση. Τα παραπάνω συνδυάζονται για τη δημιουργία ενός νέου, ενιαίου επιπέδου αφαίρεσης.

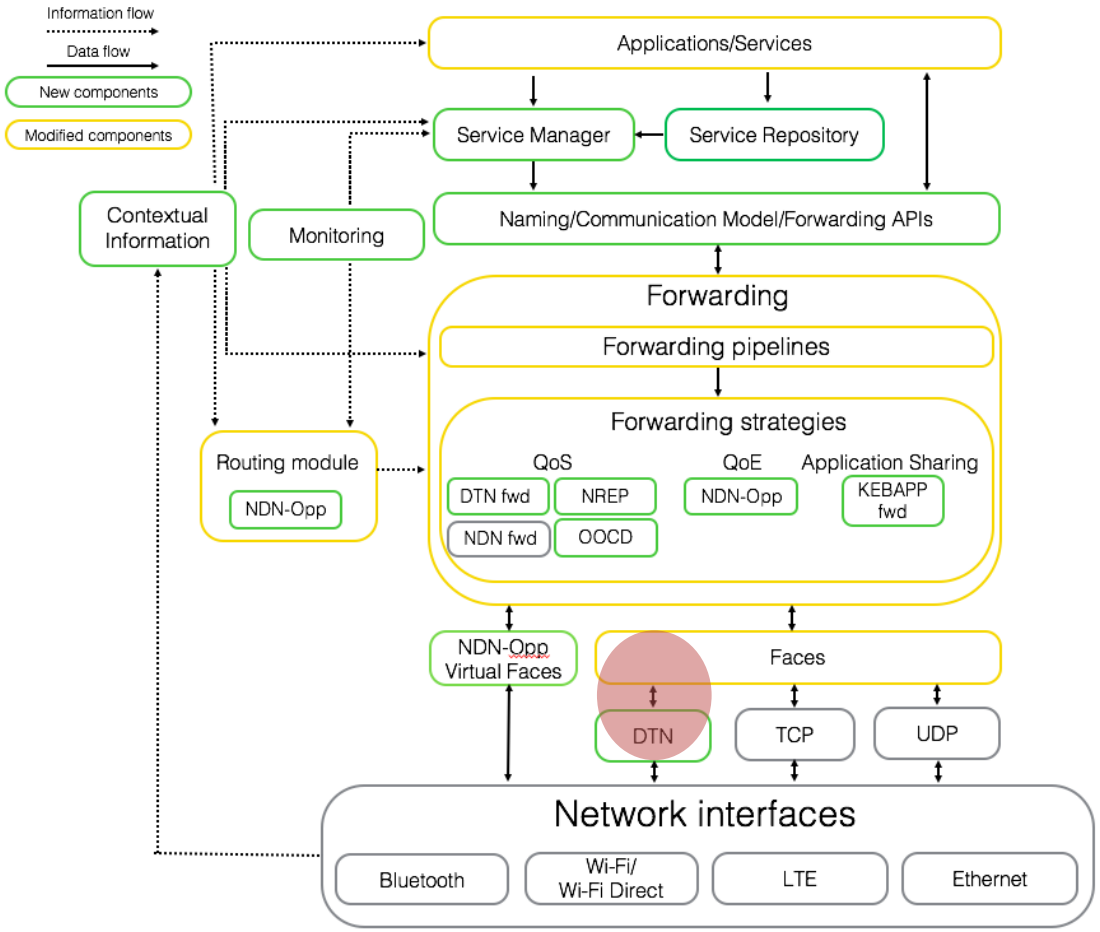
Το UMOBILE βασίζεται στην αρχιτεκτονική NDN και την επεκτείνει ως εξής (σχήμα 9):

- Δημιουργία ενός νέου, ανεκτικού στις καθυστερήσεις face (το face είναι γενίκευση του interface στην αρχιτεκτονική NDN, π.χ. TCP face). Αυτό το DTN face χρησιμοποιείται σε ευκαιριακά σενάρια ή σε περιπτώσεις προβληματικής συνδεσιμότητας και παραδίδει τα NDN πακέτα στο πρωτόκολλο IBR-DTN[3] το οποίο είναι μία υλοποίηση του Bundle Protocol [19] το οποίο και αναλαμβάνει τη μετάδοσή τους.

[3] https://www.ibr.cs.tu-bs.de/projects/ibr-dtn/

- Ανάπτυξη μίας νέας δομής (NDN-Opp) η οποία καθιστά την ευκαιριακή επικοινωνία μέσω NDN δυνατή. Για το σκοπό αυτό αναπτύσσονται ένα νέο Wi-Fi Direct face καθώς και δύο νέες εφαρμογές, μία εφαρμογή chat και μία ειδήσεων.
- Παροχή δύο νέων υπηρεσιών: Μία πλατφόρμα μεταφοράς υπηρεσιών (service migration) η οποία μπορεί να μεταφέρει υπηρεσίες πιο κοντά στα άκρα του δικτύου για ταχύτερη πρόσβαση και ένα push-service σε περίπτωση που ο χρήστης χρειάζεται να μεταφέρει δεδομένα (και όχι interests) στο δίκτυο π.χ. σε περίπτωση ανάγκης.
- Ανάπτυξη μίας application-centric δομής (KEBAPP) [7] στην οποία οι εφαρμογές μοιράζονται κοινά name-spaces και υποστηρίζεται η χρήση keywords.
- Υλοποίηση μηχανισμών για την αντιμετώπιση έκτακτων συνθηκών (π.χ. φυσικές καταστροφές): 1) Ένας μηχανισμός εντοπισμού και ανάκτησης προσωρινά αποθηκευμένου (cached) περιεχομένου από δικτυακούς χώρους αποθήκευσης ή συσκευές χρηστών και 2) μία λογική διάδοσης πληροφορίας (NREP) [11] με βάση την προτεραιότητα που ευνοεί τη διάδοση των πιο σημαντικών μηνυμάτων.
- Εισαγωγή ενός νέο αλγορίθμου για τον έλεγχο της συμφόρησης (INRPP) [12] ο οποίος συγκεντρώνει (pools) τούς πόρους (εύρος ζώνης και αποθηκευτική ικανότητα) ώστε να πετύχει δικαιοσύνη και τοπική σταθερότητα.
- Δημιουργία μίας δομής έξυπνης δρομολόγησης που λαμβάνει υπόψιν τη συμπεριφορά και το περιβάλλον του χρήστη.
- Ανάπτυξη ενός προγράμματος το οποίο αντλεί δεδομένα από το περιβάλλον του χρήστη (network mining) και παράγει μετρικές δρομολόγησης αλλά και προτάσεις χρήσιμες προς άλλες εφαρμογές.

Στην παρούσα εργασία υλοποιήθηκε η διεπαφή μεταξύ NDN και DTN για φορητές συσκευές Android με τη δημιουργία ενός DTN face (σχήμα 9). Έτσι, ο NFD (NDN Forwarding Daemon) έχει τη δυνατότητα μεταφοράς πακέτων με αυξημένη αξιοπιστία μέσω DTN είτε με μορφή tunneling (ενδιάμεσοι κόμβοι που δεν διαθέτουν τον NFD είτε σε hop-by-hop διάταξη.



Σχήμα 9: Τα στοιχεία της αρχιτεκτονικής UMOBILE

Η αρχιτεκτονική UMOBILE αξιοποιεί το DTN face με νέες στρατηγικές προώθησης οι οποίες επιτρέπουν επικοινωνία που είναι ανεκτική σε καθυστερήσεις και διαταραχές. Η απόφαση για προώθηση μέσω DTN λαμβάνεται είτε προληπτικά είτε ως αντίδραση. Στην πρώτη περίπτωση τον λόγο έχει η εφαρμογή η οποία ορίζει τον τύπο face που θα χρησιμοποιήσει. Στη δεύτερη περίπτωση γίνεται χρήση του DTN face αφού εμφανιστούν σημάδια συμφόρησης ή απώλειας σύνδεσης.

Η ανεκτική σε καθυστερήσεις δικτύωση επεκτείνει την επικοινωνία στο πεδίο του χρόνου λειτουργώντας με λογική *αποθήκευσε-και-προώθησε* (*store-and-forward*) αντί να υποθέτει την ύπαρξη συνεχούς επικοινωνίας από άκρο σε άκρο. Κάθε κόμβος αναλαμβάνει προσωρινά την κηδεμονία των δεδομένων που πρέπει να προωθήσει μέχρι να του δοθεί η ευκαιρία να το πράξει. Τα δεδομένα που είναι αποθηκευμένα σε ένα κόμβο μπορούν να μεταφερθούν ακόμη και στο πεδίο του χώρου ώστε να επιτευχθεί επικοινωνία (π.χ. με ένα αεροπλάνο [13]).

Ο κορμός της αρχιτεκτονικής DTN είναι το Bundle Protocol (BP) [19]. Το Bundle αποτελείται από μία σειρά ομάδων δεδομένων τα οποία δρομολογούνται με τη λογική *αποθήκευσε-και-προώθησε* μέσω διάφορων transport επιπέδων (π.χ. TCP,

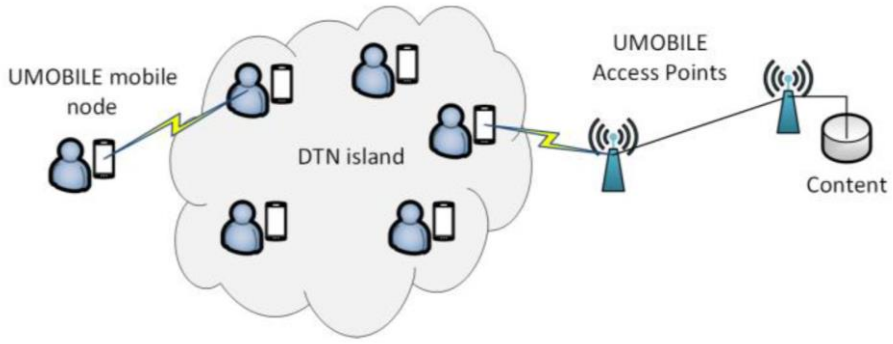
UDP). Το DTN αναλαμβάνει ακόμα την διάσπαση (fragmentation) και την επανασύνθεση των Bundles με σκοπό την αποφυγή άσκοπων επαναμεταδόσεων αλλά και για να είναι συμβατό με πρωτόκολλα χαμηλότερου επιπέδου. Η επανασύνθεση μπορεί αν πραγματοποιηθεί είτε στον τελικό προορισμό είτε σε κάποιον ενδιάμεσο κόμβο.

Για τις ανάγκες του UMOBILE χρησιμοποιείται η DTN υλοποίηση IBR-DTN η οποία είναι από τις πιο δημοφιλείς υλοποιήσεις του Bundle Protocol. Το IBR-DTN είναι σχεδιασμένο για ενσωματωμένα συστήματα και συντηρείται ενεργά. Το DTN face στέλνει και λαμβάνει πακέτα μέσω του IBR-DTN daemon γεγονός που καθιστά δυνατά τα εξής:

- Ένα νέο επίπεδο ποιότητας υπηρεσιών (quality of service), το less-than-best-effort το οποίο μπορεί να υποστηρίξει μη απαιτητικές εφαρμογές .
- Αξιοπιστία σε περιβάλλοντα στα οποία υπάρχει συμφόρηση.
- Έλεγχο ροής (congestion control) με την διοχέτευση της κίνησης μέσω του IBR-DTN εάν εντοπισθεί συμφόρηση.

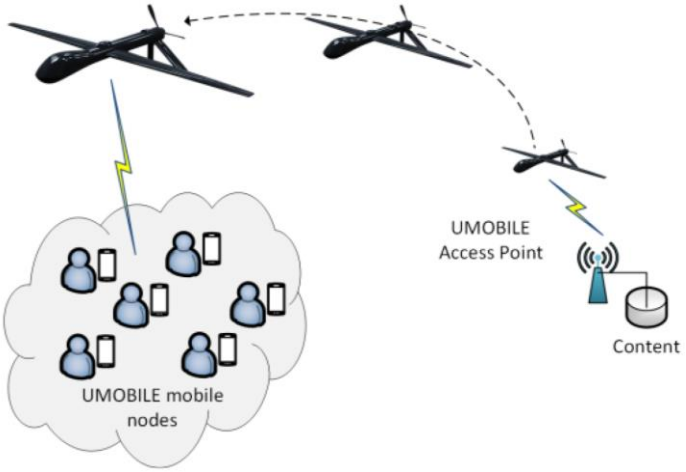
Το DTN face χρησιμοποιείται στα παρακάτω σενάρια [5] :

- Ως στρατηγική προώθησης (forwarding strategy) σύμφωνα με την οποία το DTN face επιλέγεται αυτόματα ώστε ένας χρήστης ο οποίος βρίσκεται εκτός δικτυακής υποδομής να μπορέσει να επικοινωνήσει με κάποιο κοντινό access point μέσω ενός ενδιάμεσου DTN-Tunnel (σχήμα 10) σε περίπτωση που οι ενδιάμεσοι κόμβοι εμφανίζουν έντονη μεταβλητότητα (volatility).



Σχήμα 10: Προώθηση μέσω DTN Tunneling σε ασταθείς συνθήκες δικτύου [5]

- Ως στρατηγική προώθησης που εφαρμόζεται εκ των προτέρων όταν η καθυστέρηση είναι αναμενόμενη ή και σκόπιμη (π.χ. μεταφορά δεδομένων μέσω UAV) σε περιοχές εκτός δικτυακής υποδομής (σχήμα 11).



Σχήμα 11: Προώθηση DTN όταν είναι γνωστό εξ' αρχής ότι θα υπάρξει καθυστέρηση [5]

Στη συνέχεια θα παρουσιαστεί η υλοποίηση της διασύνδεσης NDN – DTN.

Κεφάλαιο 4

Διασύνδεση NDN - DTN

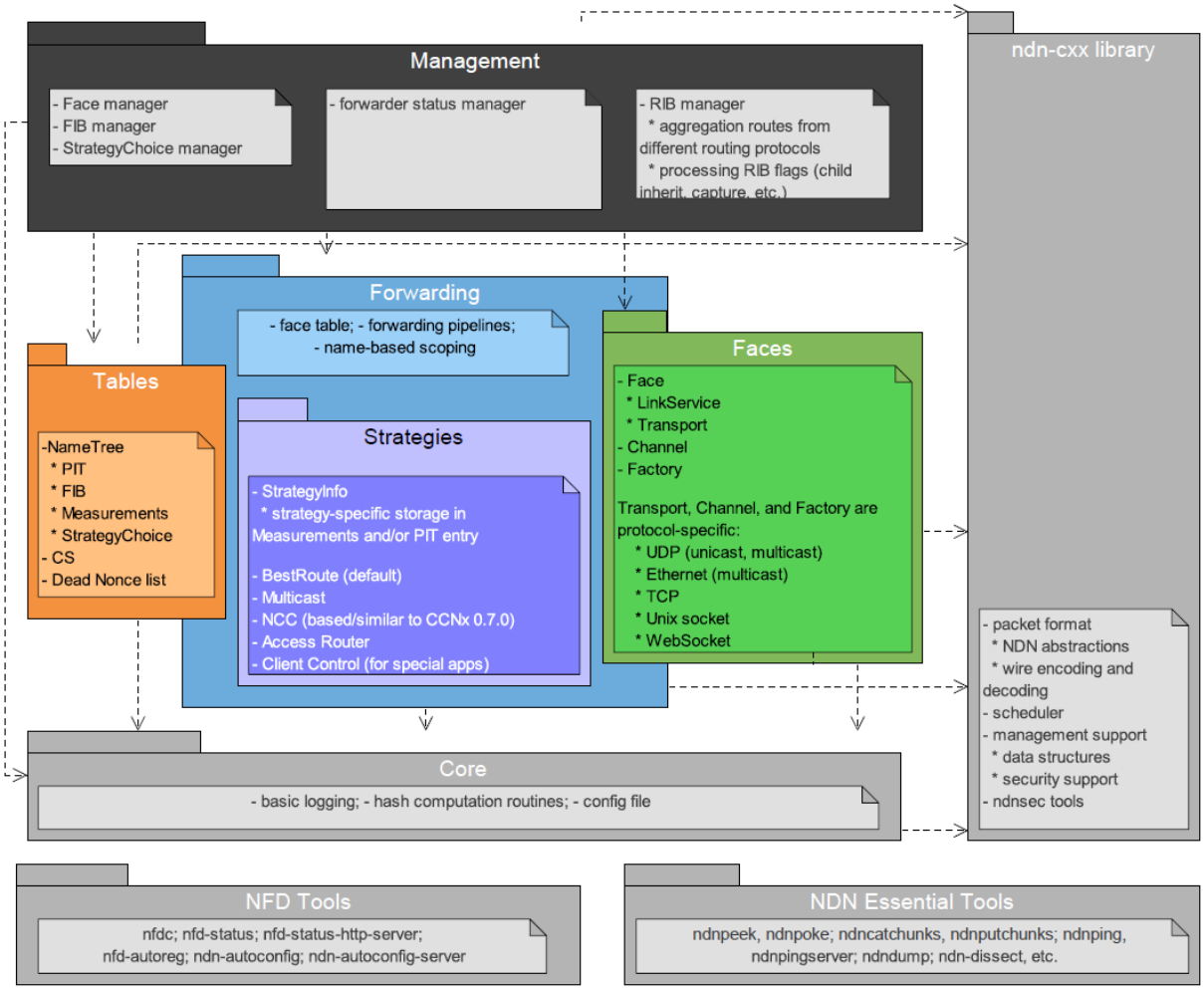
Ο κύριος στόχος της παρούσας εργασίας είναι η δημιουργία μίας διεπαφής μεταξύ του NFD (NDN Forwarding Daemon) και του IBR-DTN έτσι ώστε να μπορεί ο πρώτος να προωθήσει πακέτα μέσω του DTN face. Απαραίτητα εργαλεία για τον παραπάνω σκοπό ήταν το Android Studio IDE, η γνώση της λειτουργίας του NFD, το IBR-DTN API και οι γλώσσες C++ και Java. Ο κώδικας της διεπαφής βρίσκεται στο παράρτημα 1. Για να γίνουν κατανοητές οι αποφάσεις που ελήφθησαν κατά την υλοποίηση είναι αναγκαία μία σύντομη περιγραφή της λειτουργίας του NFD.

4.1 NDN Forwarding Daemon

Ο NFD [14] εξελίσσεται παράλληλα με το πρωτόκολλο NDN το οποίο και υλοποιεί με βασική του λειτουργία την προώθηση πακέτων Interest και Data [15]. Στο σχήμα 13 φαίνονται τα επιμέρους τμήματα του NFD. Συνοπτικά αυτά είναι:

- Faces: Η γενίκευση των χαμηλότερου επιπέδου δικτυακών διεπαφών (UDP/TCP/Ethernet κ.α.).
- Tables: Εδώ περιλαμβάνονται τα Content Store (CS – caching), Pending Interest Table (PIT), Forwarding Information Base (FIB), διάφορες μετρήσεις καθώς και άλλες δομές για την υποστήριξη της προώθησής.
- Forwarding: Εδώ πραγματοποιείται η επεξεργασία και προώθηση των πακέτων σε συνεργασία με τα τμήματα Faces, Tables και Strategies.
- Strategies: Ένα σημαντικό κομμάτι του τμήματος προώθησης. Υποστηρίζει την ύπαρξη πολλών στρατηγικών προώθησης, όπως η χρήση του DTN face σε συνθήκες συμφόρησης (κεφάλαιο 3), με τη μορφή αγωγών προώθησης (forwarding pipelines).
- Management: Η υλοποίηση του NFD Management Protocol [16] το οποίο επιτρέπει τη διαμόρφωση (configuration) και τον ορισμό ή την γνώση της κατάστασης του NFD από εφαρμογές. Η επικοινωνία μεταξύ NFD και εφαρμογών γίνεται με ανταλλαγή πακέτων Interest και Data.
- RIB Management: Ο διαχειριστής της Routing Information Base η οποία μπορεί να τροποποιηθεί με διάφορους τρόπους όπως από πρωτόκολλα δρομολόγησης, από εφαρμογές και χειροκίνητα. Στόχος είναι η δημιουργία ενός αξιόπιστου πίνακα προώθησης και ο συγχρονισμός του με το FIB, το οποίο περιέχει την λιγότερη δυνατή πληροφορία που χρειάζεται για την προώθηση.

- ndn-cxx Library, Core και Tools: Εδώ περιλαμβάνονται χρήσιμες υπηρεσίες (π.χ. υπολογισμός hash, DNS resolution, παρακολούθηση των faces κ.α.) οι οποίες είναι κοινές για τα διάφορα τμήματα.



Σχήμα 13: Τα επιμέρους τμήματα του NFD [15]

Τα πακέτα Interest και Data φτάνουν στον NFD μέσω των faces τα οποία ονομάστηκαν έτσι, γενικεύοντας τον όρο interface, ώστε να περιλαμβάνουν πρωτόκολλα στο φυσικό επίπεδο (π.χ. Ethernet) αλλά και πρωτόκολλα επιπέδου μεταφοράς (π.χ. TCP/UDP). Κάθε face αποτελείται από την LinkService και το Transport. Η πρώτη παρέχει υπηρεσίες υψηλού επιπέδου όπως διάσπαση (fragmentation) και επανασύνθεση, μετρητές επιπέδου δικτύου και ανίχνευση αποτυχιών ενώ το δεύτερο γενικεύει το χαμηλότερου επιπέδου πρωτόκολλο μετάδοσης (TCP, UDP, Ethernet κ.τ.λ.). Το κάθε face διαβάζει τα εισερχόμενα δεδομένα από το API του λειτουργικού συστήματος και τα παραδίδει στο τμήμα Forwarding με μορφή Interest, Data ή Nack.

Ειδικότερα, ένα εισερχόμενο Interest καταχωρείται αρχικά στον πίνακα PIT μαζί με τα υπόλοιπα Interest τα οποία εκκρεμούν ή ικανοποιήθηκαν πρόσφατα. Στη συνέχεια τα αντίστοιχα δεδομένα αναζητούνται στην προσωρινή μνήμη (CS). Εάν εντοπιστούν επιστρέφονται στον ενδιαφερόμενο, διαφορετικά το Interest θα προωθηθεί σε άλλο κόμβο. Η απόφαση σχετικά με το πως θα προωθηθεί το Interest λαμβάνεται από τη Στρατηγική Προώθησης (Forwarding Strategy) με βάση το όνομα και το FIB. Το Interest τελικά προωθείται μέσω του κατάλληλου face.

Στην περίπτωση ενός εισερχόμενου πακέτου Data, αρχικά γίνεται έλεγχος στον πίνακα PIT για το εάν ικανοποιεί κάποιο Interest που εκκρεμεί. Εάν ναι, τότε το πακέτο προωθείται προς τον κατάλληλο προορισμό (ή προς πολλούς) και αποθηκεύεται στο Content Store ενώ σε διαφορετική περίπτωση απορρίπτεται.

Για την υλοποίηση του DTN face τροποποιήθηκαν τα τμήματα Face, Management, ndn-cxx και Core. Το τελευταίο περιλαμβάνει το configuration file στο οποίο προστέθηκε το πρωτόκολλο DTN ώστε ο τροποποιημένος Face Manager (τμήμα Management) να υποστηρίζει τη δημιουργία DTN faces. Στη βιβλιοθήκη ndn-cxx προστέθηκε η δυνατότητα δημιουργίας και ελέγχου του μοναδικού αναγνωριστικού (URI) των DTN faces. Στη συνέχεια θα περιγραφεί αναλυτικότερα η δομή και η λειτουργία των faces.

Η κλάση nfd::Face παρέχει ποιότητα υπηρεσιών επιπέδου best-effort για τα NDN πακέτα επιπέδου δικτύου. Μέσω των faces το τμήμα forwarding λαμβάνει και στέλνει Interest, Data και Nacks. Ακόμη, με τη χρήση faces επιτυγχάνεται η επικοινωνία μεταξύ NFD και εφαρμογών γεγονός το οποίο απλοποιεί το σχεδιασμό του NFD (σε σχέση με τη χρήση κλήσεων συστήματος). Κάθε face παρέχει τις συναρτήσεις sendInterest, sendData και sendNack στο τμήμα forwarding ενώ μια άλλη κλάση, η FaceTable, διατηρεί μία λίστα με τα ενεργά faces και επιτρέπει στο forwarding να λάβει πακέτα μέσω των συναρτήσεων afterReceiveInterest, afterReceiveData και afterReceiveNack. Κάθε face έχει ορισμένα δημόσια χαρακτηριστικά:

- FaceId: Η ταυτότητα που δίνει στο face η κλάση FaceTable για όσο αυτό είναι ενεργό.
- LocalURI: Το FaceUri το οποίο αντιπροσωπεύει τον τοπικό κόμβο.
- RemoteURI: Το FaceUri που αντιστοιχεί τον απομακρυσμένο κόμβο.
- Persistency: Καθορίζει τη συμπεριφορά του face σε περίπτωση σφάλματος στο κανάλι επικοινωνίας ή μετά από ένα διάστημα απραξίας.
    - on-demand: Το face καταστρέφεται εάν παραμείνει ανενεργό για κάποιο διάστημα ή σε περίπτωση σφάλματος.
    - persistent: Για να καταστραφεί το face πρέπει αν δοθεί ρητή εντολή ή να υπάρξει σφάλμα στο κανάλι επικοινωνίας.
    - Permanent: Το face δεν καταστρέφεται εκτός εάν δοθεί ρητή εντολή.

- LinkType: Καθορίζει το εάν το face είναι point-to-point ή multi-access.
- State: Δείχνει την κατάσταση του face.
    - UP: Κανονική λειτουργία
    - DOWN: Προσωρινό κλείσιμο – σε διαδικασία ανάκτησης.
    - CLOSING: Φυσιολογικός τερματισμός του face
    - FAILED: Τερματισμός λόγω αποτυχίας
    - CLOSED: Το face έχει τερματιστεί
- Counters: Στατιστικά για τον αριθμό και το μέγεθος των Interests, Data, Nacks αλλά και πακέτων χαμηλότερου επιπέδου τα οποία έχουν διέλθει μέσω του face.

Εσωτερικά, το face αποτελείται από τις κλάσεις LinkService και Transport. Το τελευταίο είναι το κατώτερο επίπεδο του face και διαχειρίζεται πακέτα TLV (Type-Length-Value) [18]. Η LinkService είναι το άνω τμήμα του face και μετατρέπει πακέτα επιπέδου δικτύου σε χαμηλότερου και αντίστροφα. Η κλάση nfd::face::Face δέχεται τα αντικείμενα LinkService και Transport κατά την κατασκευή του και αυτά ορίζουν πλήρως τη συμπεριφορά του.

Όταν λαμβάνονται πακέτα από το Transport, αυτά μεταφέρονται στο LinkService μέσω της συνάρτησης LinkService::receivePacket. Η αντίστροφη πορεία ακολουθείται κατά την αποστολή πακέτων όπου ανάλογα με το Type (πρώτο πεδίο του TLV) αυτά προωθούνται προς το LinkService από τις συναρτήσεις Face::sendInterest, Face::sendData και Face::sendNack και μετά προς το Transport καλώντας τη συνάρτηση Transport::send.

4.2 Υλοποίηση

Τμήματα της υλοποίησης βασίστηκαν στη δομή της υπάρχουσας ενσωμάτωσης NDN-DTN [17] η οποία υλοποιήθηκε στο πλαίσιο του UMOBILE για σταθερούς κόμβους. Ο NFD για Android[4] είναι port της αρχικής έκδοσης, είναι γραμμένος σε C++ και χρησιμοποιεί το Java Native Interface (JNI) για την αλληλεπίδραση του native κώδικα με τον κώδικα γραμμένο σε Java, η οποία είναι η βασική γλώσσα προγραμματισμού στο Android. Μάλιστα, χρησιμοποιείται το third-party CrystalX Android NDK[5] (Native Development Kit) το οποίο, σε αντίθεση με το Google NDK,

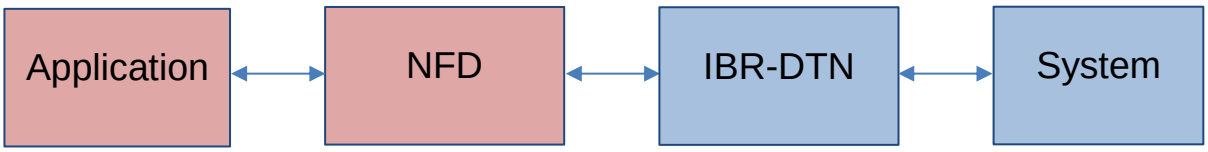
[4] https://github.com/named-data-mobile/NFD-android

[5] https://www.crystax.net/en

υποστηρίζει, μεταξύ άλλων, τις βιβλιοθήκες Boost C++[6] τις οποίες χρησιμοποιεί ο NFD.

Η υλοποίηση του IBR-DTN βρίσκεται στο Google Play Store[7]. Είναι επίσης διαθέσιμη μία βιβλιοθήκη API[8] (Application Programming Interface), γραμμένη σε Java, για την επικοινωνία με τον IBR-DTN.

Αφαιρετικά, τα δεδομένα διέρχονται από το εξής stack:



Σχήμα 14: NFD – IBR-DTN stack. Με κόκκινο τα τμήματα που τροποποιήθηκαν/αναπτύχθηκαν.

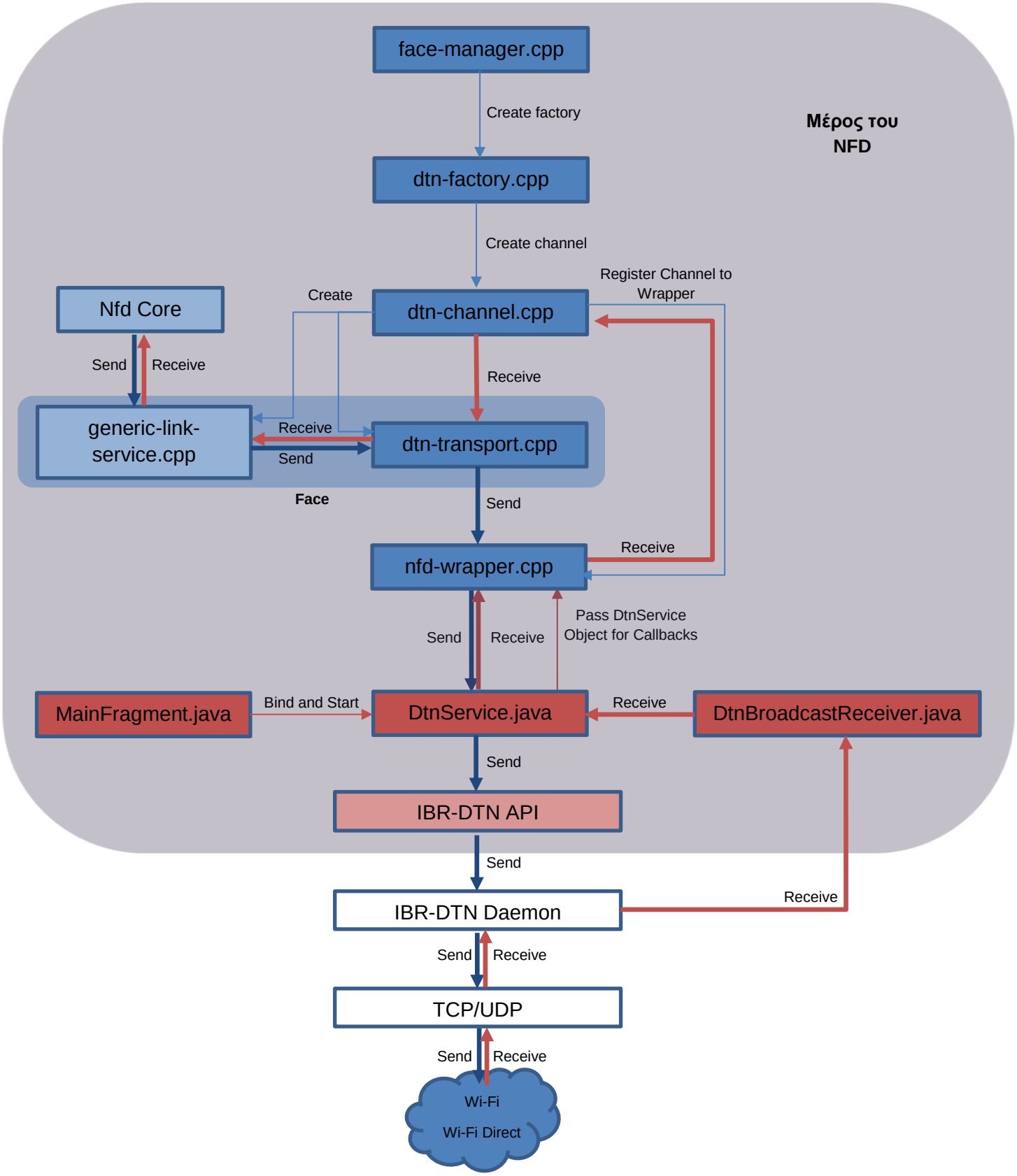
Η δομή που ακολουθήθηκε φαίνεται στο σχήμα 15. Συνοπτικά, όταν η εφαρμογή στέλνει πακέτα ακολουθείται η εξής διαδρομή: Το τμήμα Forwarding καλεί την αντίστοιχη συνάρτηση (για Interest, Data ή Nack) στο generic-link-service.cpp το οποίο με τη σειρά του καλεί τη συνάρτηση DtnTransport::doSend. Από το αρχείο dtn-transport.cpp καλείται η συνάρτηση sendToIBRDTN η οποία βρίσκεται στο αρχείο nfd-wrapper.cpp. Το τελευταίο είναι αυτό που φορτώνεται από την Java Virtual Machine ως native library και μέσω αυτού συντελείται το σύνολο της επικοινωνίας μεταξύ Java και C++. Ύστερα καλείται, μέσω του Java Native Interface, η συνάρτηση DtnService.sendMessage η οποία εκκινεί την υπηρεσία DtnService με σκοπό την παράδοση των δεδομένων στον IBR-DTN.

Κατά τη λήψη δεδομένων η κλάση DtnBroadcastReceiver λαμβάνει το σχετικό broadcast που εκπέμπει ο IBR-DTN και εκκινεί την υπηρεσία DtnService με σκοπό τη λήψη δεδομένων. Ο DataHandler εντός της κλάσης DtnService καλεί μέσω του JNI τη συνάρτηση queueBundleJNI η οποία βρίσκεται στο αρχείο nfd-wrapper.cpp. Στη συνέχεια, με την κλήση της συνάρτησης DtnChannel::queueBundle, το πακέτο προωθείται στο κανάλι (dtn-channel.cpp) όπου καταχωρείται ως ασύγχρονο task στον scheduler του NFD. Τέλος ο scheduler καλεί τη συνάρτηση DtnTransport::receiveBundle, το πακέτο στη συνέχεια προωθείται στην κλάση GenericLinkService από όπου καταλήγει στο τμήμα Forwarding

[6] http://www.boost.org/

[7] https://play.google.com/store/apps/details?id=de.tubs.ibr.dtn&hl=en

[8] https://github.com/ibrdtn/ibrdtn-android-library



Σχήμα 15: Διαγραμματική απεικόνιση της διασύνδεσης NDN - IBR-DTN. Με έντονο μπλε ή κόκκινο τα αρχεία που δημιουργήθηκαν/τροποποιήθηκαν.

Σε αυτό το σημείο είναι σημαντικό να σημειωθεί ότι η παραπάνω υλοποίηση δεν είναι η πιο αποδοτική αφού η παρεμβολή του Java Native Interface εισάγει αρκετό overhead. Εντούτοις, στα σενάρια όπου, σύμφωνα με τις προδιαγραφές του UMOBILE, το DTN face χρησιμοποιείται, η επιπρόσθετη καθυστέρηση είναι αμελητέα όπως θα δούμε στο κεφάλαιο 5.

Τα περισσότερα από τα modules, των οποίων η λειτουργία θα περιγραφεί συνοπτικά, βρίσκονται στο σχήμα 15. Για την επεξήγηση της λειτουργίας ακολουθείται κυρίως η ροή των δεδομένων.

- DtnService.java
  Η κλάση DtnService είναι το πιο κοντινό τμήμα στον IBR-DTN. Κληρονομεί την κλάση DtnIntentService του IBR-DTN API, η οποία επιτρέπει το registration ενός prefix στον IBR-DTN για την επικοινωνία με αυτόν. Όταν δημιουργείται το αντικείμενο της κλάσης, καλείται η συνάρτηση initializeNativeInterface η οποία βρίσκεται στο αρχείο nfd-wrapper.cpp και δίδεται στο native κομμάτι ένας δείκτης για το αντικείμενο DtnService ώστε να είναι δυνατή η αποστολή πακέτων από τον NFD μέσω αυτού. Η DtnService είναι μία Bound Service. Κατά την εκκίνηση του NFD η κλάση mainFragment προσδένεται στην κλάση DtnService και έτσι δεν καλείται η συνάρτηση onDestroy της τελευταίας κάθε φορά που ολοκληρώνεται μία αποστολή ή λήψη.
- DtnBroadcastReceiver.java
  Κληρονομεί την κλάση android.content.BroadcastReceiver και μπορεί και λαμβάνει τα broadcasts που εκπέμπει ο IBR-DTN όταν αυτός θέλει να παραδώσει ένα πακέτο στον NFD. Τότε καλείται η συνάρτησή onReceive, η οποία δημιουργεί ένα intent που προορίζεται για την DtnService (τα intents αποτελούν περιγραφές εργασιών που πρόκειται να συμβούν) στο οποίο τοποθετεί τα δεδομένα που ελήφθησαν. Η κλάση DtnService λαμβάνει το Intent χάρη στη συνάρτηση onHandleIntent, εξάγει τα δεδομένα και τη διεύθυνση του αποστολέα και τα προωθεί στο αρχείο nfd-wrapper.cpp μέσω της συνάρτησης DtnService.queueBundleJNI.
- nfd-wrapper.cpp
  Το αρχείο nfd-wrapper τροποποιήθηκε ώστε να επιτρέπει την μεταφορά πακέτων προς και από την κλάση DtnService. Κατά τη φόρτωσή του αποθηκεύονται δείκτες για την DtnService και τη συνάρτησή της sendMessage, ώστε να βελτιωθεί η απόδοση, καθώς και για τη Java Virtual Machine. Όταν ο NFD επιθυμεί να στείλει ένα πακέτο μέσω του DtnFace καλεί τελικά τη συνάρτηση sendToIBRDTN η οποία στη συνέχεια καλεί την DtnService.sendMessage. Ακόμη, το αρχείο nfd-wrapper διαθέτει τη

συνάρτηση registerChannelToWrapper η οποία καλείται από το dtn-channel.cpp ώστε να μπορούν να προωθηθούν ληφθέντα πακέτα σε αυτό. Τέλος, εδώ βρίσκεται hard-coded το configuration file του nfd, το οποίο επιτρέπει στον face-manager να δημιουργήσει τα κανάλια ή/και τα faces των πρωτοκόλλων.

- dtn-transport.cpp
  Αποτελεί το κάτω μέρος του DTN face και διαθέτει τις συναρτήσεις DtnTransport::doSend και DtnTransport::receiveBundle. Η πρώτη καλείται από το generic-link-service.cpp όταν ο NFD επιθυμεί να προωθήσει κάποιο πακέτο μέσω του DTN face το οποίο στη συνέχεια προωθείται στον nfd-wrapper.cpp. Η δεύτερη καλείται από το dtn-channel.cpp για την παραλαβή εισερχόμενου πακέτου το οποίο στη συνέχεια παραδίδεται στο generic-link-service.cpp.
- dtn-channel.cpp
  Δημιουργείται από τον dtn-factory.cpp με εντολή του FaceManager, κατά την εκκίνηση του nfd, σύμφωνα με το configuration file. Η κλάση DtnChannel: 1) λαμβάνει εισερχόμενα πακέτα από τον wrapper στη συνάρτηση queueBundle και στη συνέχεια τα παραδίδει στον scheduler του NFD ώστε αυτός να καλέσει ασύγχρονα [17] τη συνάρτηση DtnTransport::receiveBundle, 2) Δημιουργεί το DTN face κατόπιν εντολής από το dtn-factory.cpp.
- dtn-factory.cpp
  Δημιουργείται από τον FaceManager κατά την εκκίνηση του NFD. Κατασκευάζει και διαχειρίζεται το DtnChannel.
- Face-manager.cpp [17]
  Δημιουργεί το DtnFactory με βάση το configuration file. Επίσης ζητά από το dtn-channel να δημιουργήσει ένα κατάλληλο DTN face (το channel είναι ένα, όμως τα face μπορούν να είναι πολλά) όταν εισάγεται ένα route στο GUI που παρέχει ο NFD για android κατά την πειραματική διαδικασία (κεφάλαιο 5.1).
- Βοηθητικά Αρχεία
  - Face-uri.cpp (βιβλιοθήκη ndn-cxx) [17]
    Τροποποιήθηκε κατάλληλα ώστε να μπορεί να διαχειριστεί τα URIs των DTN faces.
  - nfd.mk (αρχείο για το Android Studio IDE)
    Προστέθηκαν τα νέα .cpp αρχεία.
  - build.Gradle (αρχείο του Android Studio IDE)
    Προστέθηκε η βιβλιοθήκη του IBR-DTN API.
  - local.properties (αρχείο του Android Studio IDE)
    Προστέθηκε το third-party CrystalX ndk.

Κεφάλαιο 5

Πειραματικά Αποτελέσματα

5.1 Μεθοδολογία

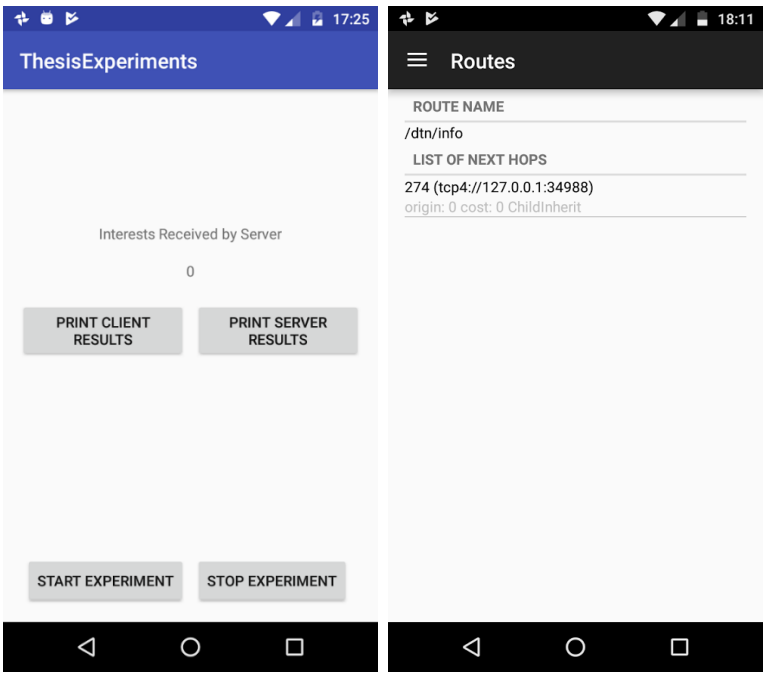
Για την διεξαγωγή των πειραμάτων αναπτύχθηκε μία πλατφόρμα (στο λειτουργικό Android) ή οποία επιτρέπει την λήψη και την αποστολή Interest και Data, την παραμετροποίηση χαρακτηριστικών όπως τον ρυθμό και το τρόπο αποστολής, το πρωτόκολλο που θα χρησιμοποιηθεί από τον NFD (UDP, TCP, DTN), το μέγεθος των πακέτων κ.α. αλλά και την εξαγωγή χρήσιμων μετρήσεων όπως goodput, per packet delay κ.α. Για την επικοινωνία με τον NFD χρησιμοποιείται το API jndn[9] το οποίο επιτρέπει την αποστολή δεδομένων από και προς τον NFD με τη δημιουργία του αντίστοιχου face. Η πλατφόρμα είναι γραμμένη σε Java και αποτελείται από τις εξής κλάσεις:

- MainActivity: Πρόκειται για την κλάση η οποία αλληλεπιδρά με το GUI (εικόνα 1α) και αποτελεί το σημείο εκκίνησης του προγράμματος. Τα Activities είναι μία βασική ομάδα κλάσεων στο Android και συνήθως αλληλεπιδρούν με τον χρήστη (π.χ. μία εφαρμογή χρονόμετρο μπορεί να είναι ένα activity). Κατά την κλήση της μεθόδου onCreate (η μέθοδος η οποία καλείται πρώτη κατά τη δημιουργία του Activity) η MainActivity ξεκινάει το thread InterestReceiverThread. Τέλος, η MainActivity, μετά το τέλος του πειράματος, εξάγει και τυπώνει τα δεδομένα για τις διάφορες μετρικές.
- InterestReceiverThread: Κληρονομεί την κλάση java.lang.Thread και ο σκοπός της είναι η λήψη Interest από τον NFD. Για να είναι αυτό δυνατό, πρέπει πρώτα να καταχωρηθεί ένα NDN name με βάση το οποίο ο NFD θα δημιουργήσει το κατάλληλο route για την επικοινωνία του με την πλατφόρμα (εικόνα 1β). Η δομή του ονόματος που επελέγη είναι /«πρωτόκολλο»/info/«x». Έτσι, εάν το πείραμα χρησιμοποιεί το πρωτόκολλο UDP και πρόκειται να σταλούν 1000 interests τότε το όνομα των Interests θα είναι /udp/info/1…1000 με το κάθε interest να προσδιορίζει ένα συγκεκριμένο τμήμα δεδομένων από το 1 έως το 1000. Το thread κάνει sleep ανά κάποια millisecond ώστε να μην σπαταλούνται οι πόροι του συστήματος. Όταν ληφθεί κάποιο Interest (μέσω της συνάρτησης onInterestCallback), δημιουργείται ένα thread για την αποστολή δεδομένων.

[9] https://github.com/named-data/jndn/blob/master/README.md

- DataSenderThread: Το thread αυτό δημιουργείται, στέλνει στον NFD ένα πακέτο με τυχαία δεδομένα και καταστρέφεται. Τα τυχαία δεδομένα είναι τύπου byte[] και το μέγεθός τους είναι παραμετροποιήσιμο.
- InterestSenderThread: Τα προηγούμενα thread χρησιμοποιούνται από τον server ώστε να ανταποκρίνεται στα Interests που λαμβάνει. Το InterestSenderThread εκκινεί όταν πατηθεί το κουμπί “Start Experiment” και σταματά όταν έχουν σταλεί όλα τα Interest (ο αριθμός τους είναι παραμετροποιήσιμος). Υποστηρίζονται δύο τρόποι λειτουργίας. Ο πρώτος είναι η αποστολή των Interest *back-to-back* με βάση μία περίοδο αποστολής (π.χ. 100 ms). Στο δεύτερο, το Interest στέλνεται όταν ληφθούν τα δεδομένα που αντιστοιχούν στο προηγούμενο (στο εξής *conversation mode*).
- GoodputVsTimeThread: Το συγκεκριμένο thread ενεργοποιείται σε τακτά διαστήματα, μετράει τον αριθμό των πακέτων Data που ελήφθησαν από την τελευταία ενεργοποίηση και καταχωρεί το στιγμιαίο goodput σε μία δομή.

Η πλατφόρμα δε διαθέτει κάποια λογική εντοπισμού απώλειας και επαναποστολής πακέτων. Αυτό είναι όμως είναι κάτι που θα είχε ενδιαφέρον να εξεταστεί στο μέλλον – δηλαδή πως συγκρίνεται ο συνδυασμός DTN με απλή εφαρμογή και TCP/UDP με εφαρμογή που αντισταθμίζει τις απώλειες. Ο κώδικας της πλατφόρμας βρίσκεται στο παράρτημα 2.



Εικόνα 1 – αριστερά: η πλατφόρμα πειραμάτων, δεξιά: η λίστα των next hops για το prefix /dtn/info

Οι παράμετροι των πειραμάτων είναι:

- Μέγεθος πακέτων Data (σε byte). Έγιναν δοκιμές με πακέτα 1000 και 3500 byte.
- Τύπος επικοινωνίας (back-to-back ή conversation).
- Ρυθμός αποστολής Interest (μόνο σε λειτουργία back-to-back).
- Πρωτόκολλο (DTN, TCP, UDP).
- Προβλήματα συνδεσιμότητας.
- Επεξεργαστική Ισχύς συσκευών.
- Μέσο αποθήκευσης συσκευών.

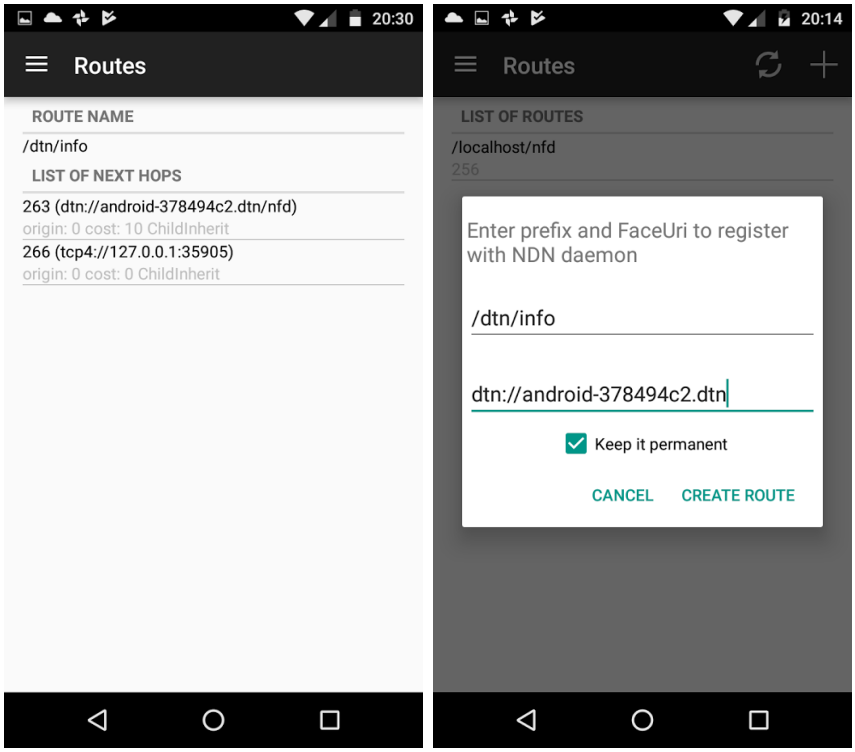
Οι μετρικές που εξήχθησαν είναι:

- Packet Delivery Ratio (πόσα πακέτα Data έφτασαν στον client σε σχέση με τα πακέτα Interest που στάλθηκαν).
- Στιγμιαίο και μέσο Goodput (**ως προς τον client - μόνο τα Data – μία μέτρηση ανά 3 δευτερόλεπτα**).
- RTT (Round Trip Time) ανά πακέτο (**δηλαδή ο χρόνος από τη στιγμή που αναχωρεί το Interest μέχρι την άφιξη των Data στην πλατφόρμα και όχι στον IBR-DTN**).
- Μέσο και διάμεσο RTT
- Συνολικός χρόνος
- Overhead εξαιτίας του JNI (μετρήθηκε στον NFD)

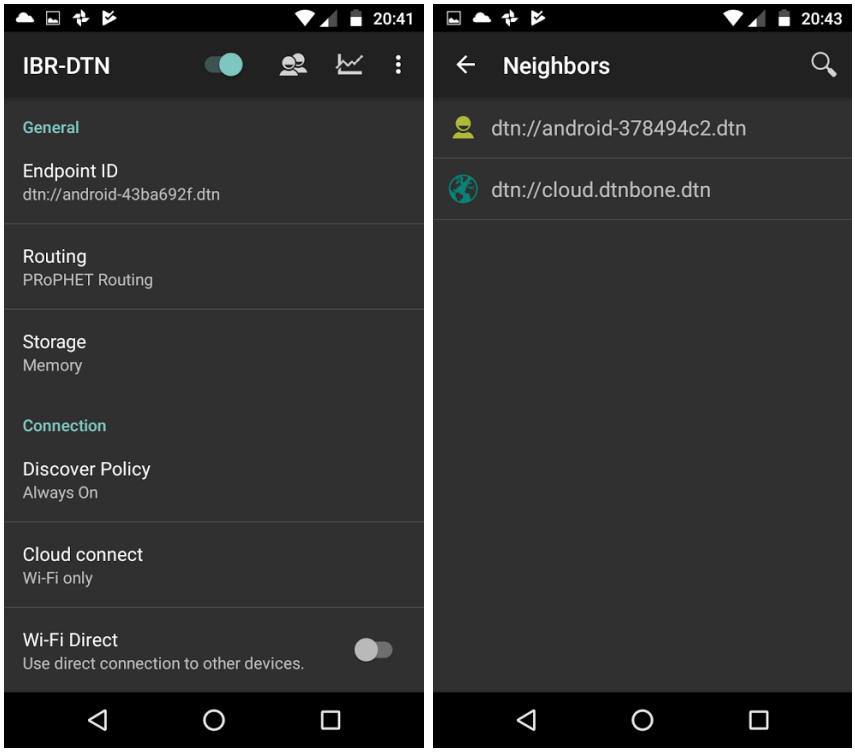
Τα προβλήματα στη συνδεσιμότητα προσομοιώθηκαν με τη χρήση ενός αγώγιμου κουτιού (Faraday Cage), με τοιχώματα επαρκούς πάχους, το οποίο περιέκλειε πλήρως τη συσκευή και δεν επέτρεπε τη διέλευση ηλεκτρομαγνητικών κυμάτων Wi-Fi (2.4 GHz) με αποτέλεσμα την απώλεια της σύνδεσης. Η επιλογή αυτή έγινε διότι η μέθοδος εισόδου και εξόδου από την εμβέλεια του Wi-Fi δίνει λιγότερο αξιόπιστα και μη συγκρίσιμα αποτελέσματα. Συνολικά έγιναν δοκιμές χωρίς προβλήματα συνδεσιμότητας, με απώλεια σύνδεσης μία φορά κατά τη διάρκεια του πειράματος και με πολλαπλές απώλειες. Για όλες τις δοκιμές, οι συσκευές βρίσκονταν σε οπτική επαφή με τον ασύρματο δρομολογητή σε απόσταση ενός μέτρου. Η απώλεια σύνδεσης συμβαίνει στην ίδια χρονική απόσταση από την αρχή του πειράματος σε κάθε δοκιμή ώστε τα αποτελέσματα να είναι συγκρίσιμα. Συνολικά πραγματοποιήθηκαν 60 δοκιμές με την κάθε μία να διαρκεί 4 με 8 λεπτά. Εξ’ αιτίας της αργής διαδικασίας διεξαγωγής των δοκιμών, οι πολλές επαναλήψεις ανά πείραμα ήταν αδύνατες. Εντούτοις, όταν προέκυπτε κάποιο αποτέλεσμα το οποίο οδηγούσε σε ποιοτικά συμπεράσματα, η δοκιμή επαναλαμβανόταν για επαλήθευση.

Η διαδικασία διεξαγωγής κάθε δοκιμής είναι η εξής:

1. Επιλογή Παραμέτρων (π.χ. Μέγεθος πακέτων 1000 bytes, back-to-back, 100ms περίοδος αποστολής, πρωτόκολλο DTN) στον client και στον server.
2. Καταχώρηση της διαδρομής (route) προς τον άλλο κόμβο στον NFD (εικόνα 2α). Κάθε route έχει δύο χαρακτηριστικά: prefix και face URI. Όταν ο NFD λαμβάνει κάποιο πακέτο, εξετάζει το όνομά του**. Εάν υπάρχει καταχώρηση στο Forwarding Information Base (FIB) με prefix ίδιο με το Name ή με αρχικό τμήμα του (longest prefix match) τότε το πακέτο προωθείται προς το Face που ορίζεται από το face URI.** Για παράδειγμα, στην εικόνα 2β φαίνονται τα routes που δημιουργούνται κατά την εκτέλεση του πειράματος. Το 263 δημιουργείται χειροκίνητα και με βάση αυτό τα πακέτα που ξεκινούν από /dtn/info προωθούνται μέσω του DTN face (το οποίο καθρίζεται από το scheme, dtn://) στη διεύθυνση dtn://android-378494c2.dtn. Τα TCP και UDP έχουν ως face κάτι της μορφής tcp://192.268.1.8 όμως ο IBR-DTN daemon δεν χρησιμοποιεί διευθύνσεις IP αλλά Endpoint IDs (εικόνα 3α, 3β). Το 266 δημιουργείται αυτόματα από την πλατφόρμα δοκιμών κατά την καταχώρηση του Name και επιτρέπει την προώθηση των πακέτων από τον NFD προς την πλατφόρμα.
3. Εκκίνηση του πειράματος.



Εικόνα 2 – αριστερά: η λίστα των πιθανών next hops για το prefix /dtn/info κατά τη διεξαγωγή των πειραμάτων, δεξιά: εισαγωγή ενός νέου next hop



Εικόνα 3 – αριστερά: το κεντρικό activity του IBR-DTN, δεξιά: η λίστα γειτόνων

Τέλος, στην εικόνα 2β φαίνεται η δυνατότητα ορισμού του face ως permanent (τα permanent faces δεν καταστρέφονται σε περίπτωση σφάλματος όπως για παράδειγμα όταν χάνεται η σύνδεση Wi-Fi). Η επιλογή αυτή είναι αδιάφορη στην περίπτωση χρήσης του DTN face αφού ο IBR-DTN daemon και όχι το λειτουργικό σύστημα παραλαμβάνει τα πακέτα. Αντίθετα ένα UDP ή TCP face που δεν είναι permanent καταστρέφεται. **Δυστυχώς η έκδοση του NFD που τροποποιήθηκε δεν παρουσιάζει σταθερή συμπεριφορά**. Μάλιστα, το debugging στο UDP face (δεν έχει υλοποιηθεί ο μηχανισμός για permanent persistency στο TCP face ([15] σελ. 12) έδειξε ότι ακόμη και αν επιλεχθεί το πεδίο “Keep it permanent”, το face που δημιουργείται είναι είτε persistent είτε on-demand ενώ δεν βρέθηκε κάποιος εύκολος τρόπος επίλυσης του προβλήματος (η αντικατάσταση του εισερχόμενου πεδίου persistency με τον τύπο permanent δεν οδήγησε σε φυσιολογική συμπεριφορά). Το γεγονός αυτό σημαίνει ότι τις περισσότερες φορές – αλλά όχι πάντα – το face και το route που «δείχνουν» προς τον άλλο κόμβο διαγράφονται μετά την πρώτη απώλεια σύνδεσης.

5.2 Αποτελέσματα Πειραμάτων

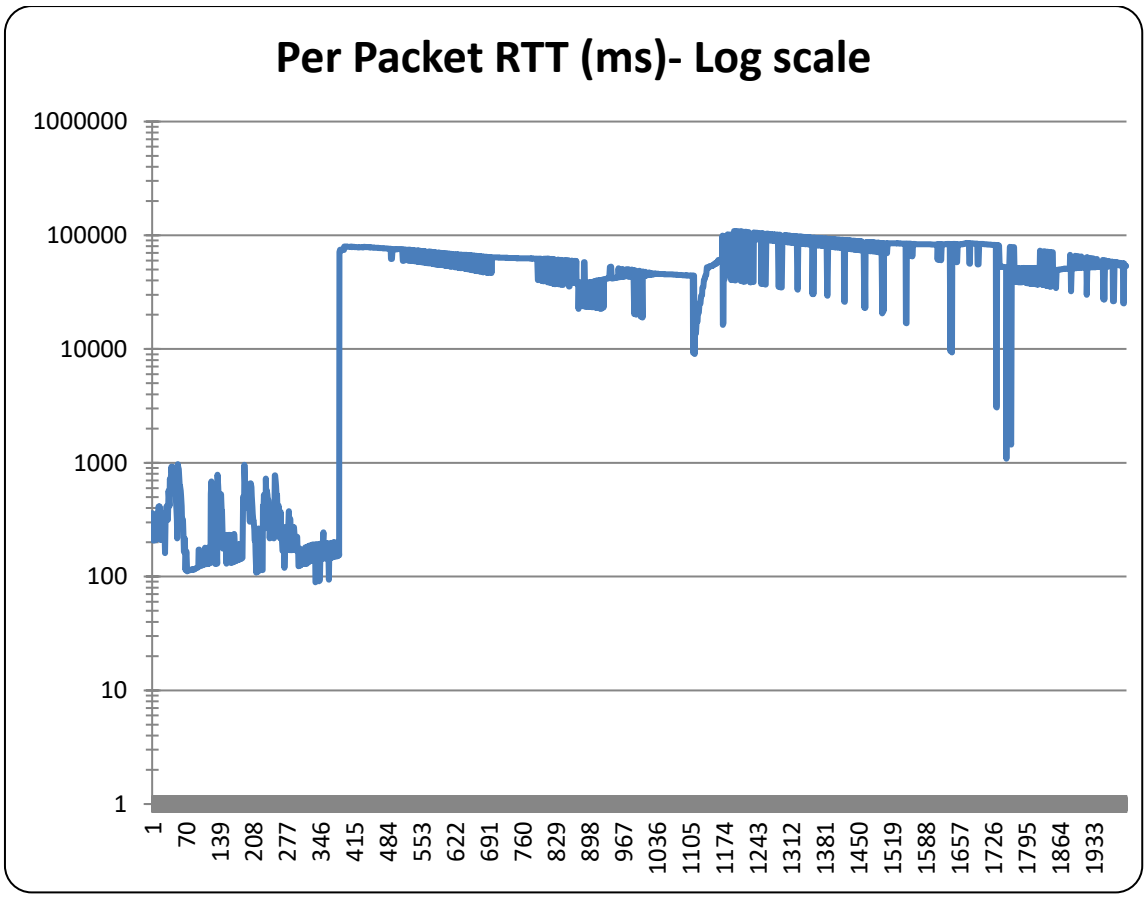
Σε αυτό το υποκεφάλαιο θα παρουσιαστούν τα πειραματικά αποτελέσματα ως εξής: Αρχικά θα γίνει παράθεση των μετρήσεων που ελήφθησαν κατά τη χρήση του

DTN face. Στη συνέχεια θα συγκριθούν τα πρωτόκολλα DTN, TCP και UDP. Τέλος θα σχολιαστεί η απόδοση της υλοποίησης.

5.2.1 Συμπεριφορά του DTN face

Στόχος του DTN face είναι αύξηση της αξιοπιστίας. Έτσι η αναμενόμενη συμπεριφορά σε περίπτωση προβληματικής σύνδεσης ή απώλειας αυτής είναι η ανταλλαγή (tradeoff) της μη απώλειας πακέτων με την αύξηση της καθυστέρησης. Σε όλα τα πειράματα που ακολουθούν, **οι μετρικές έχουν εξαχθεί από τον client** η περίοδος της back to back αποστολής είναι 100ms έκτος εάν δηλωθεί διαφορετικά. Επίσης υπάρχουν δύο τύποι απώλειας σύνδεσης. Ο πρώτος είναι απώλεια Wi-Fi μετά από 40 δευτερόλεπτα διάρκειας 30 δευτερολέπτων και ο δεύτερος περιλαμβάνει τρείς απώλειες σύνδεσης διάρκειας 30 δευτερολέπτων έκαστη οι οποίες συμβαίνουν στους χρόνους 00:20, 01:50 και 04:00. Στο εξής θα αναφέρονται ως τύπος 1 και τύπος 2 αντίστοιχα.

Μία διακοπή



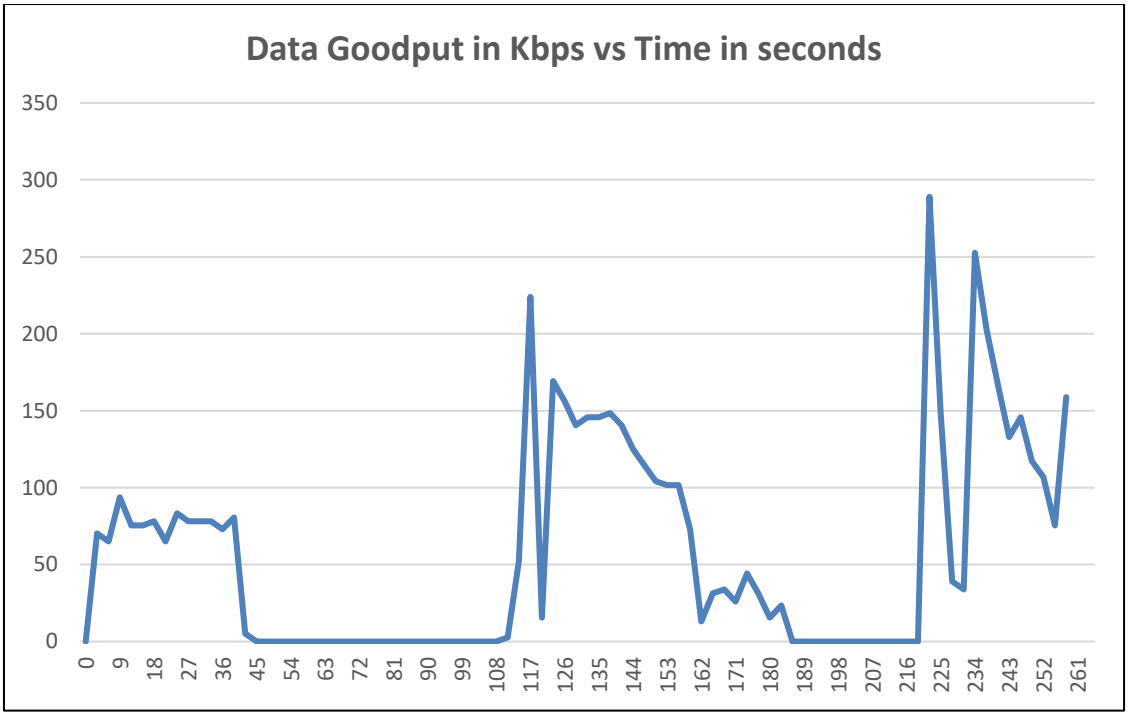
Διάγραμμα 1 – DTN, μία διακοπή, πακέτα 1000B, back-to-back, SD card

Στο διάγραμμα 1 φαίνεται ακριβώς αυτό. Οι παράμετροι είναι: πακέτα 1000 byte, back-to-back, απώλειες τύπου 1, μέσο αποθήκευσης κάρτα sd. Περίπου στο πακέτο 385 χάνεται η σύνδεση. Όσο ισχύει αυτό, ο IBR-DTN συνεχίζει να λαμβάνει Interests από τον client τα οποία τα αποθηκεύει δεδομένου ότι αδυνατεί να τα στείλει (λογική *store-and-forward*). Από τη χρονική στιγμή που η σύνδεση στο δίκτυο καθίσταται δυνατή, μεσολαβούν δύο περίοδοι. Η πρώτη είναι ο χρόνος που χρειάζεται η συσκευή ώστε να επανασυνδεθεί και η δεύτερη ο χρόνος που έχει ανάγκη ο IBR-DTN ώστε να αποκαταστήσει τη σύνδεση με τον άλλο κόμβο (η καθυστέρηση αυτή φαίνεται καλύτερα στο διάγραμμα 2).

Μετά την επανασύνδεση η επικοινωνία συνεχίζεται κανονικά με το πρώτο πακέτο Data να φτάνει στον client με πρόσθετη καθυστέρηση 30s + χρόνος αποκατάστασης Wi-Fi + χρόνος αποκατάστασης IBR-DTN ~= 70 sec. Οι δύο τελευταίες καθυστερήσεις δεν είναι σταθερές μεταξύ των πειραμάτων και έχουν μεγάλη διακύμανση.

Παρότι ο IBR-DTN daemon του server λαμβάνει Interests back-to-back με σταθερό ρυθμό για αρκετή ώρα μετά την επανασύνδεση, δεν κατορθώνει να επαναφέρει την καθυστέρηση στο επίπεδο πριν τη διακοπή. Η συμπεριφορά αυτή βελτιώνεται όταν το μέσο αποθήκευσης των πακέτων είναι ταχύτερο **(τα πειράματα που ακολουθούν εκθέτουν το worst case scenario για το DTN face)**. Η πλευρά της απόδοσης θα εξεταστεί αναλυτικά στο υποκεφάλαιο 5.2.3.

Τέλος, η εμφάνιση spikes ελαττωμένου RTT εξηγείται από το γεγονός ότι παρότι ο IBR-DTN λαμβάνει τα Data και τα Interest σε αύξουσα σειρά (/dtn/info/1…n), από τη στιγμή που τα αποθηκεύει και στο εξής ενδέχεται να τα προωθήσει εκτός σειράς (το οποίο δεν αποτελεί πρόβλημα αφού κάθε πακέτο είναι ονοματισμένο). Έτσι ενδέχεται να εξυπηρετούνται ταυτόχρονα φρέσκα και εκκρεμή πακέτα.

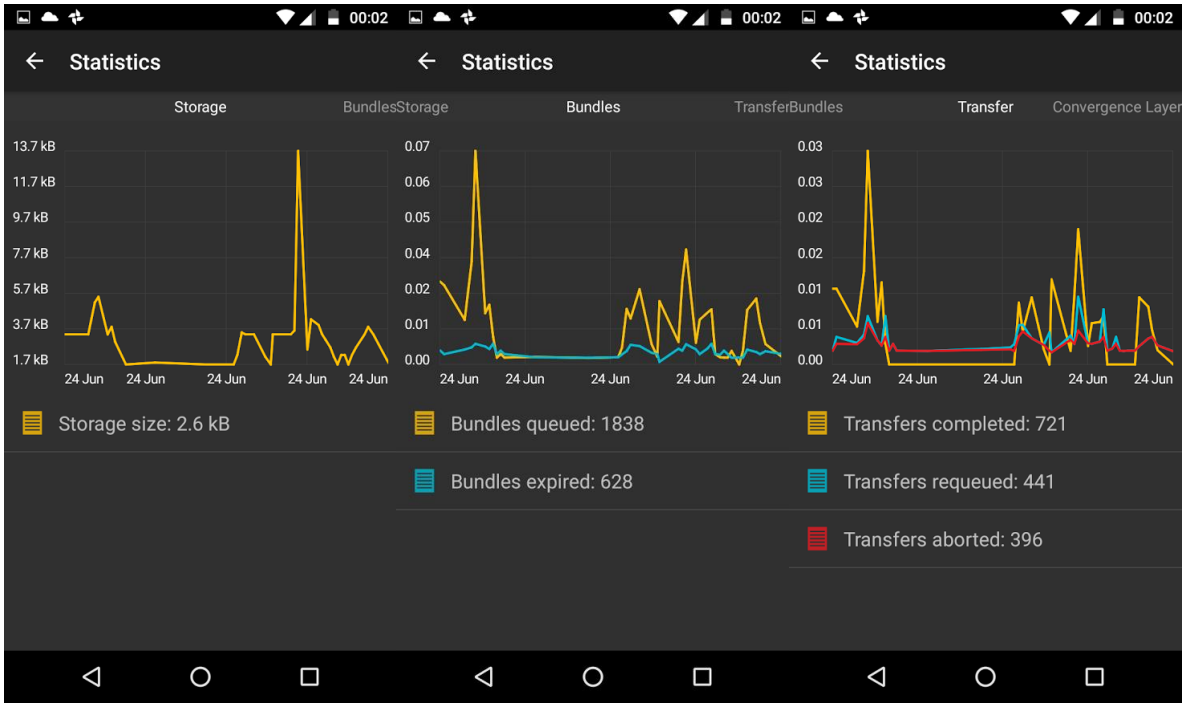


Διάγραμμα 2 – DTN, μία διακοπή, πακέτα 1000B, back-to-back, SD card

Το μέγιστο θεωρητικό Data goodput δεδομένης της αποστολής ενός Interest από τον client κάθε 100ms είναι 1000[byte] * 8 / 1024 = 78.125 Kbps.

Πριν τη διακοπή (~30 sec) το goodput βρίσκεται πράγματι σε αυτά τα επίπεδα. Μετά τη διακοπή το όριο αυτό ξεπερνιέται επειδή ο IBR-DTN daemon του client έχει συγκεντρώσει πολλά Interests και τα στέλνει μαζικά.

Στο διάστημα 190 – 220 sec το goodput μηδενίζεται εκ νέου. Αυτή το φορά όμως δεν έχει διακοπεί εξωτερικά το Wi-Fi, μάλιστα δεν υπάρχει κανένα πρόβλημα σύνδεσης. Να υπενθυμιστεί σε αυτό το σημείο ότι το goodput μετριέται στον client και αφορά στα πακέτα Data. Από τα στατιστικά που δίνει η εφαρμογή του IBR-DTN (εικόνα 4 – τυχαία στιγμή, άσχετη με το πείραμα) παρατηρήθηκε ότι ο IBR-DTN όταν βρίσκεται υπό φόρτο τείνει να ξεχωρίζει τη λήψη από την αποστολή. Δηλαδή στη συγκεκριμένη περίπτωση, η συμπεριφορά του daemon του server ή του client στο διάστημα 185-216 ήταν η λήψη Interests χωρίς την προώθηση τους στο DTN face ή η μη προώθηση προς το δίκτυο πακέτων Data τα οποία έχει ήδη λάβει από το DTN face ή αντίστροφα στην περίπτωση του client. Η συμπεριφορά αυτή εξαρτάται και από τα χαρακτηριστικά των συσκευών όπως θα δούμε στο υποκεφάλαιο 5.2.3.



Εικόνα 4 – τα στατιστικά που παρέχει ο IBR-DTN

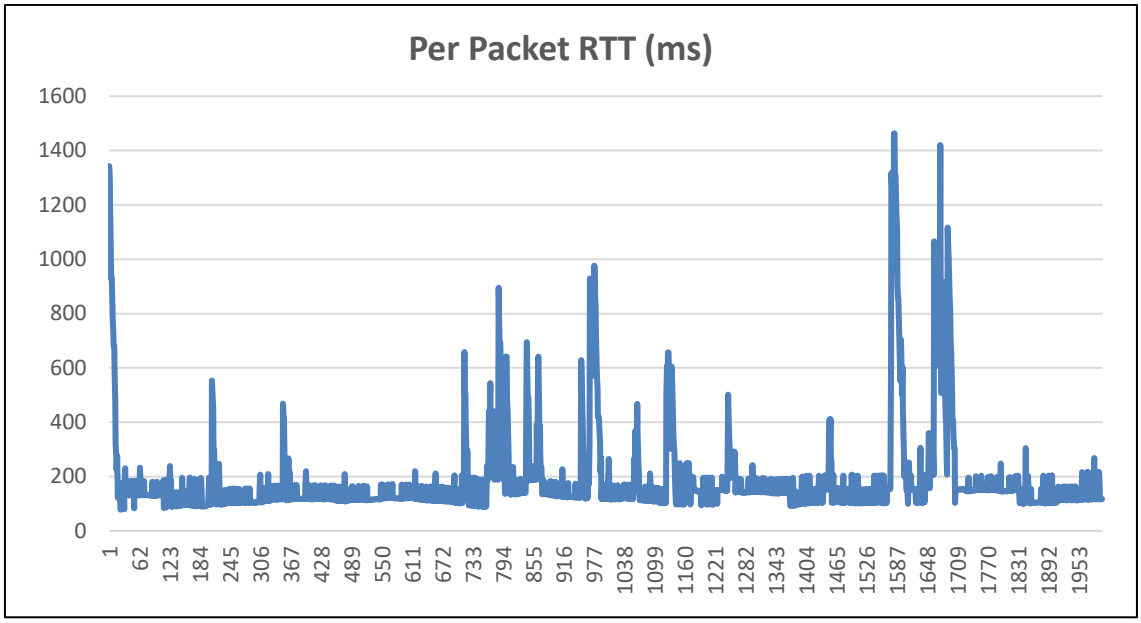
Η χρήση του DTN face γίνεται με το δεδομένο ότι η απόδοση θα είναι μειωμένη. Σύμφωνα με τις προδιαγραφές του UMOBILE (κεφάλαιο 3), το DTN Face προορίζεται για εφαρμογές με χαλαρές απαιτήσεις σχετικά με την ποιότητα των υπηρεσιών. Το κρίσιμο στοιχείο είναι η αξιοπιστία. Πράγματι, το packet delivery ratio (PDR) είναι 1 (πίνακας 1). **Όλες οι μετρήσεις εκτός από την Interests Received έγιναν στον client.** Το μέσο goodput δεν βρίσκεται πολύ χαμηλότερα του θεωρητικού ορίου. Τέλος, συμπεριλήφθηκε η διάμεση τιμή του per packet RTT για να μειωθεί η επίδραση των outliers (τα οποία εδώ είναι περιορισμένα).

Transaction Time (s)	Goodput (Kbps)	Interests Sent	Interests Received	Data Received	PDR	Average RTT ms	Median RTT ms
260	60.1	2000	2000	2000	1	52201	54667

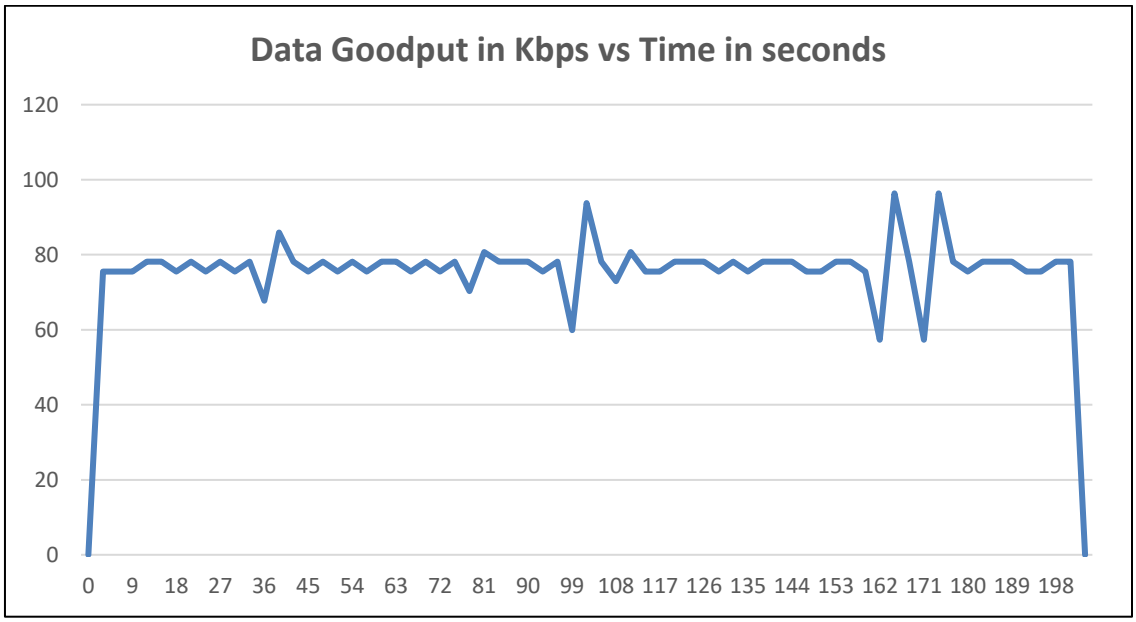
Πίνακας 1 – DTN, μία διακοπή, πακέτα 1000B, back-to-back, SD card

Στη συνέχεια ακολουθούν τα αποτελέσματα του πειράματος όταν αυτό διεξάγεται με τις ίδιες παραμέτρους χωρίς όμως προβλήματα σύνδεσης ώστε να υπάρχει ένα σημείο αναφοράς.

Κανονική λειτουργία



Διάγραμμα 3 – DTN, κανονική λειτουργία, πακέτα 1000B, back-to-back, SD card



Διάγραμμα 4 – DTN, κανονική λειτουργία, πακέτα 1000B, back-to-back, SD card

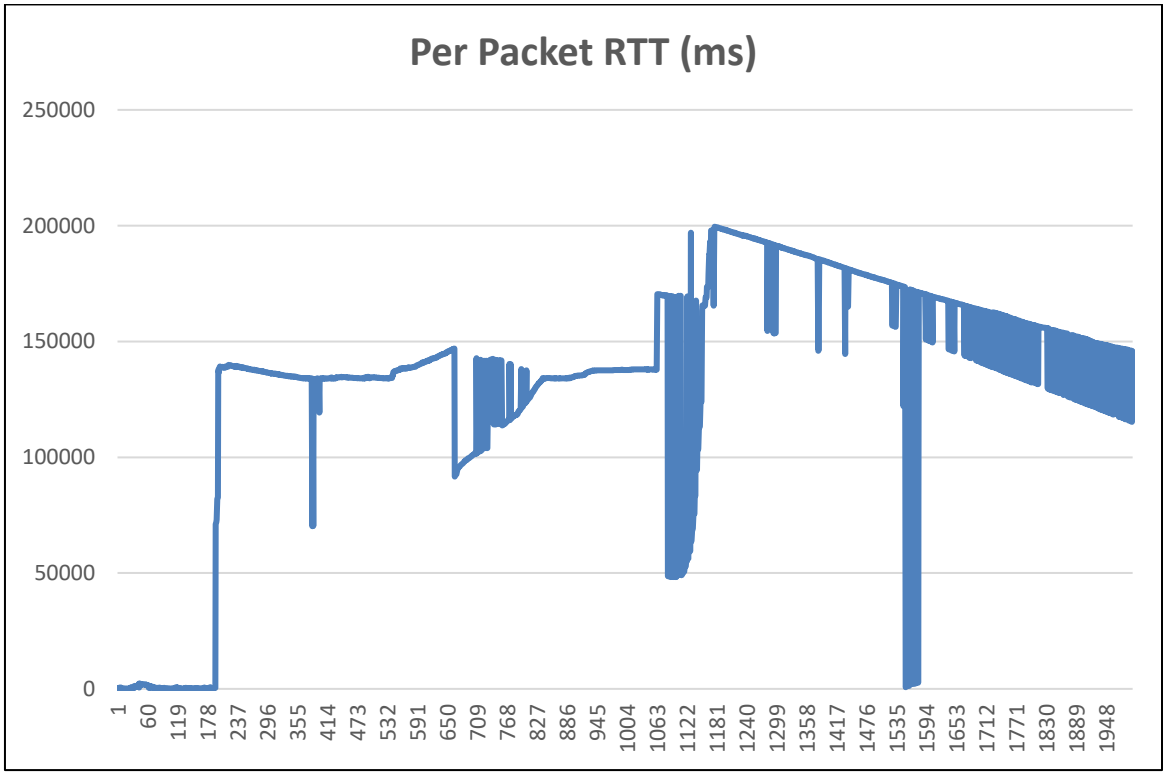
Transaction Time (s)	Goodput (Kbps)	Interests Sent	Interests Received	Data Received	PDR	Average RTT ms	Median RTT ms
202	77.35	2000	2000	2000	1	188	143

Πίνακας 2 – DTN, κανονική λειτουργία, πακέτα 1000B, back-to-back, SD card

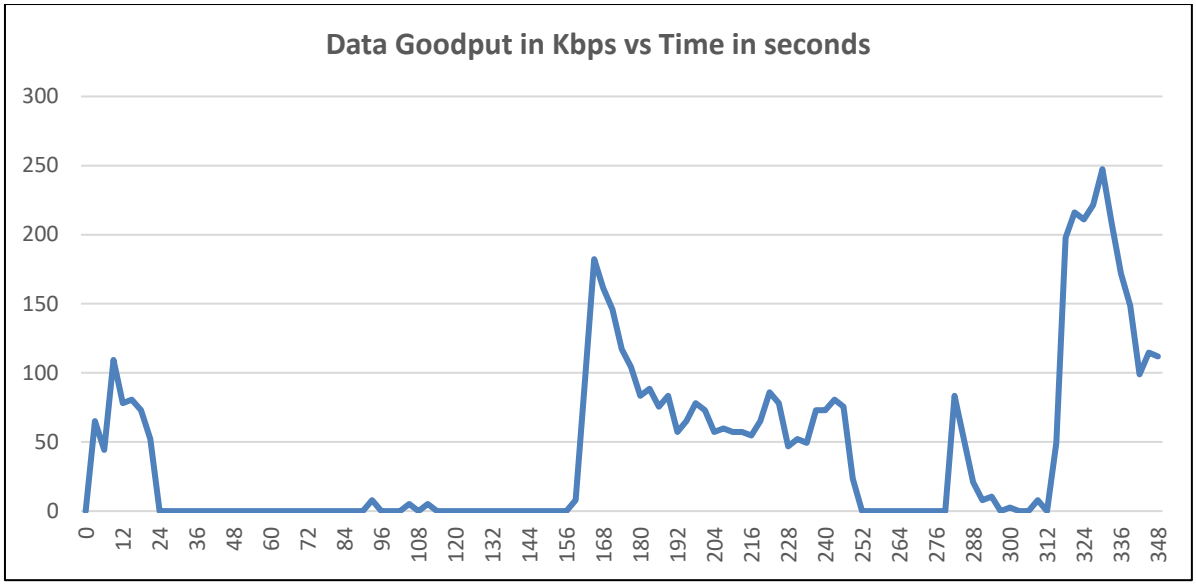
Είναι ενδιαφέρον να παρατηρήσουμε τη συμπεριφορά που εμφανίστηκε στο προηγούμενο πείραμα (της διάκρισης αποστολής και λήψης) να εμφανίζεται σε πολύ μικρότερο βαθμό. Τα σημεία του διαγράμματος 3 στα οποία εμφανίζεται απότομη αύξηση του RTT συμπίπτουν με τα σημεία στο διάγραμμα 4 στα οποία το goodput αρχικά μειώνεται και μετά αυξάνεται. Πρόκειται για το ίδιο φαινόμενο συγκέντρωσης πακέτων στη μνήμη του IBR-DTN.

Τρεις διακοπές

Η συμπεριφορά της υλοποίησης είναι παρόμοια στην περίπτωση πολλαπλών διακοπών με αυτή της μίας.



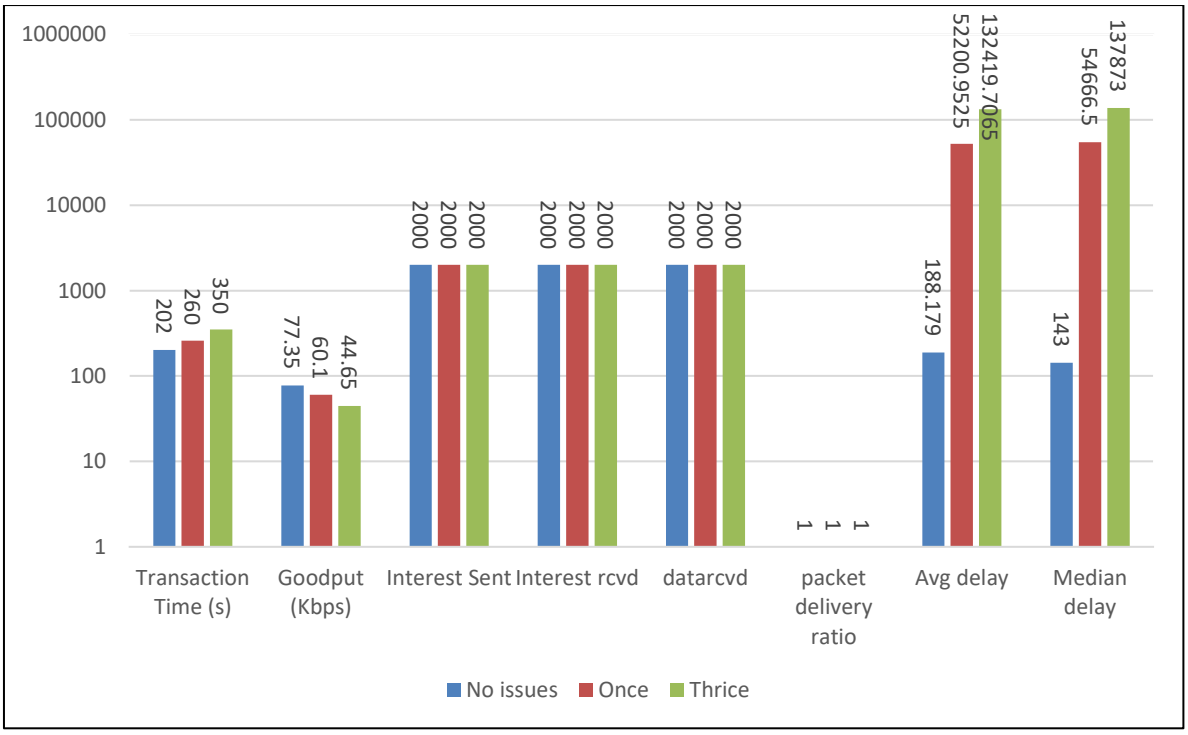
Διάγραμμα 5 – DTN, τρεις διακοπές, πακέτα 1000B, back-to-back, SD card



Διάγραμμα 6 – DTN, τρεις διακοπές, πακέτα 1000B, back-to-back, SD card

Transaction Time (s)	Goodput (Kbps)	Interests Sent	Interests Received	Data Received	PDR	Average RTT ms	Median RTT ms
350	44.65	2000	2000	2000	1	132420	137873

Πίνακας 3 – DTN, τρεις διακοπές, πακέτα 1000B, back-to-back, SD card



Διάγραμμα 7 – συγκεντρωτικά αποτελέσματα χρήσης του DTN face σε τρία σενάρια συνδεσιμότητας

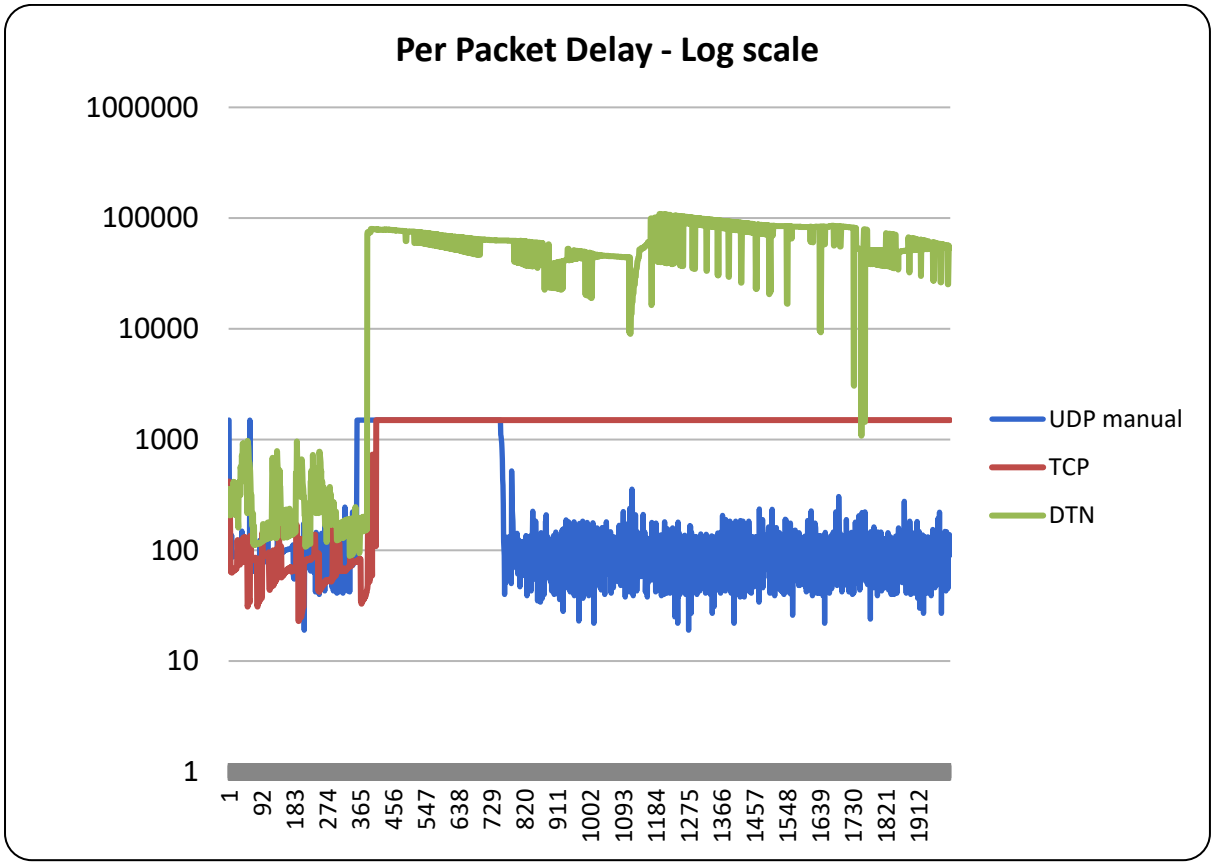
Το DTN face πετυχαίνει και εδώ το στόχο της μη απώλειας πακέτων. Ενδιαφέρον παρουσιάζει το γεγονός ότι η ανάκτηση της επικοινωνίας μετά την πρώτη διακοπή (~20 sec) (διάγραμμα 6) καθυστερεί τόσο ώστε να έρθει η επόμενη διακοπή (~110 sec). Τέλος, μία ομάδα πακέτων μετά το πακέτο 1500 εμφανίζει μειωμένη καθυστέρηση. Αυτό συμβαίνει διότι ο IBR-DTN του server επεξεργάζεται εισερχόμενα Interests εκτός σειράς ενώ υπάρχουν πολλά τα οποία εκκρεμούν. Παρατηρούμε επίσης, καθώς η συνδεσιμότητα γίνεται χειρότερη, την ήπια μείωση του goodput, την αύξηση του συνολικού χρόνου και τη σημαντική αύξηση του χρόνου RTT η οποία οφείλεται σε κάποιο bottleneck και θα εξερευνηθεί στο υποκεφάλαιο 5.2.3.

Περισσότερες παράμετροι θα τροποποιηθούν στη συνέχεια όπου γίνεται σύγκριση μεταξύ της χρήσης των πρωτοκόλλων DTN, TCP και UDP

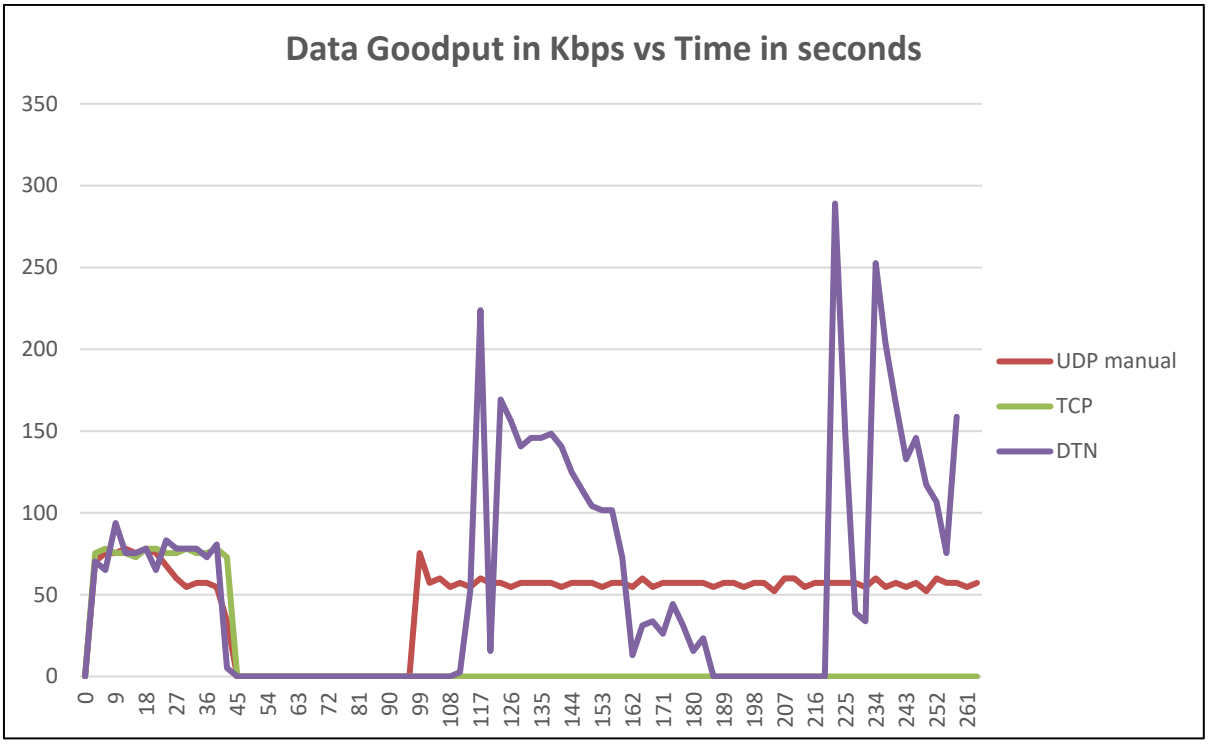
5.2.2 Σύγκριση με UDP και TCP

Για την πρώτη σύγκριση θα χρησιμοποιηθούν οι ίδιες παράμετροι με προηγουμένως, δηλαδή: πακέτα μεγέθους 1000B, back-to-back, απώλειες τύπου 1.

Μία διακοπή



Διάγραμμα 8 – μία διακοπή, πακέτα 1000B, back-to-back, SD card



Διάγραμμα 9 – μία διακοπή, πακέτα 1000B, back-to-back, SD card

	Transaction Time (s)	Goodput (Kbps)	Interest Sent	Interest rcvd	Data Received	Delivery Ratio	Avg delay	Median delay
UDP manual	269	46.38	2000	1600	1597	0.7985	380.70285	95
TCP	208	15.36	2000	409	409	0.2045	1210.5585	1500
DTN	260	60.1	2000	2000	2000	1	52200.953	54666.5

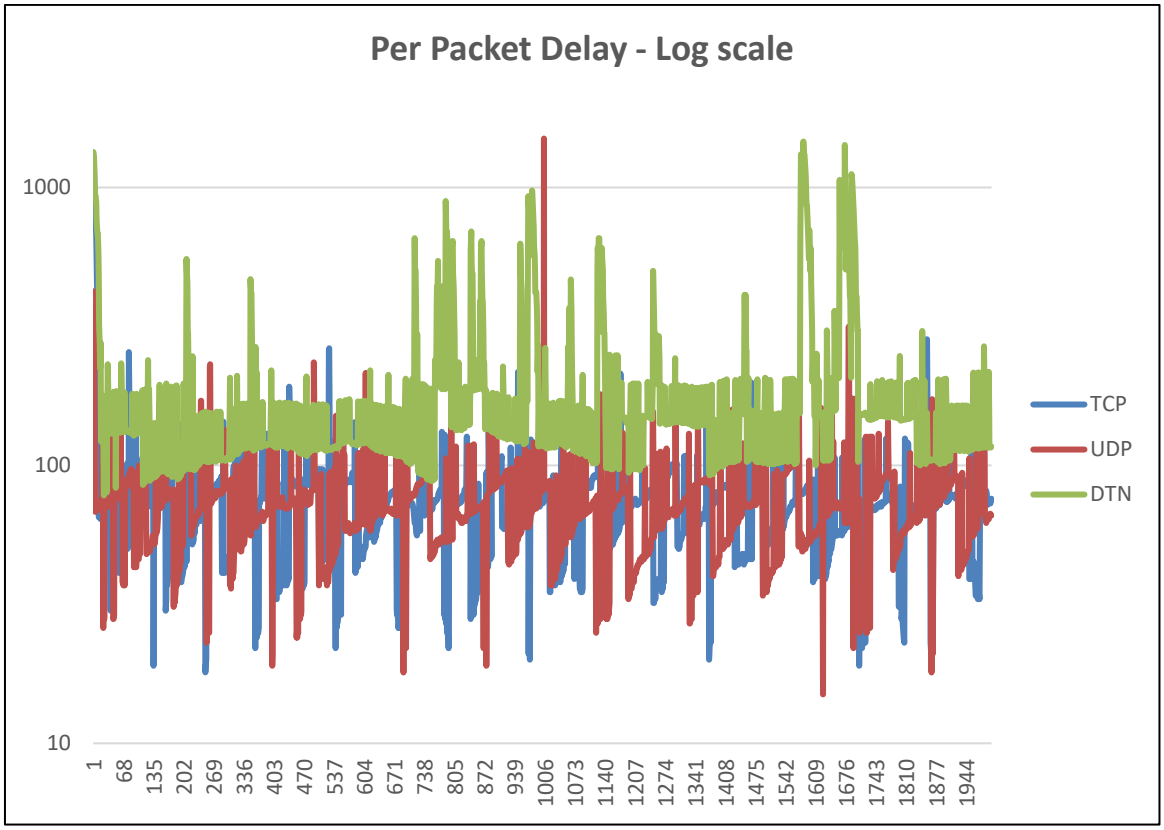
Διάγραμμα 10 – μία διακοπή, πακέτα 1000B, back-to-back, SD card

Στο συγκεκριμένο πείραμα, μετά την απώλεια σύνδεσης, και το TCP και το UDP έχασαν την καταχώρηση του αντίστοιχου route στον NFD. Όμως, κατά τη διάρκεια της ανάκτησης της σύνδεσης Wi-Fi, το route του UDP προστέθηκε χειροκίνητα (εγκαίρως) ώστε να δειχθεί η καλύτερη δυνατή περίπτωση. Γενικά τα UDP και TCP έχουν παρόμοια συμπεριφορά η οποία είναι προβλέψιμη (με δεδομένο ότι η εφαρμογή δεν αντισταθμίζει τις απώλειες με επανέκδοση των Interests). Εάν το route χαθεί, η μετάδοση δεν επανεκκινεί ποτέ. Εάν το route δεν χαθεί, η μετάδοση συνεχίζεται μετά την αποκατάσταση του Wi-Fi και τα πακέτα που θα λαμβάνονταν στο μεταξύ έχουν χαθεί λόγω timeout (αφού τα interest δεν φεύγουν ποτέ). Το timeout έχει οριστεί και για τα δύο πρωτόκολλα στα 1500ms, χρόνος περίπου 20 φορές μεγαλύτερος από το μέσο RTT σε κανονικές συνθήκες.

Στο διάγραμμα 8, οι ευθείες γραμμές των TCP και UDP με τιμή 1500 αντιστοιχούν σε timeout. Το TCP χάνει όλα τα πακέτα μετά το 385 ενώ το UDP χάνει αυτά που αντιστοιχούν στο κενό συνδεσιμότητας και μερικά στην αρχή. Το DTN δεν έχει απώλειες και γι’ αυτό το λόγο πετυχαίνει και το καλύτερο goodput. Το UDP χάνει το 20% των πακέτων, ποσοστό ανάλογο του χρόνου διακοπής συν το χρόνο επανασύνδεσης Wi-Fi. Τέλος στο διάγραμμα 9 φαίνεται ο επιπρόσθετος χρόνος τον οποίο χρειάζεται ο IBR-DTN για να ανακτήσει τη σύνδεση με τον άλλο κόμβο.

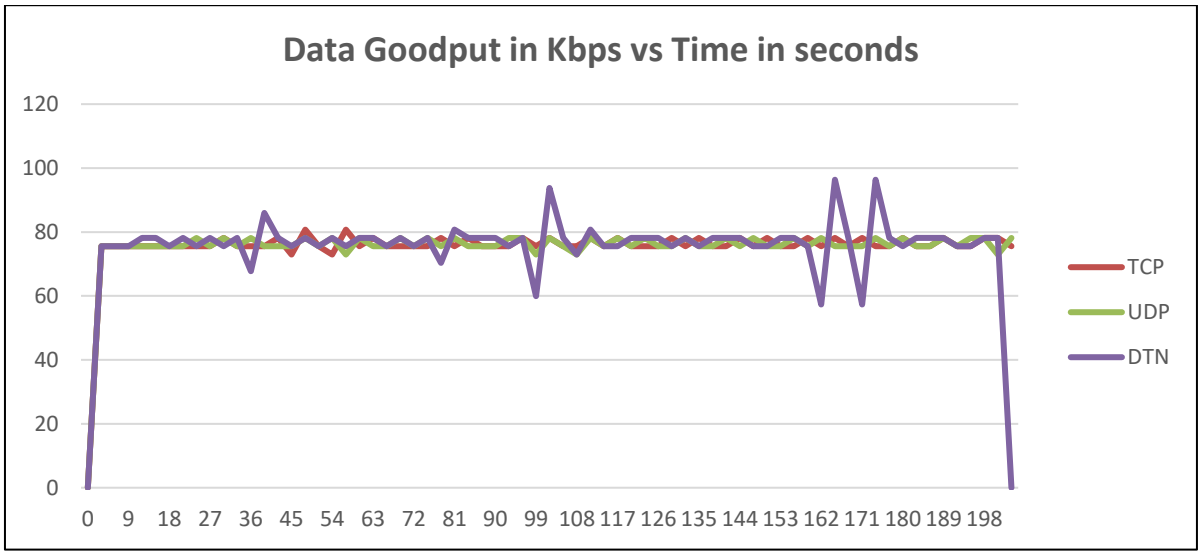
Η συμπεριφορά σε περίπτωση πολλαπλών διακοπών ή με χρήση μεγαλύτερων πακέτων είναι παρόμοια οπότε τα αποτελέσματα αυτά παραλείπονται. Στη συνέχεια παρατίθεται η σύγκριση των πρωτοκόλλων σε κανονικές συνθήκες λειτουργίας με τις υπόλοιπες παραμέτρους ίδιες.

Κανονικές συνθήκες

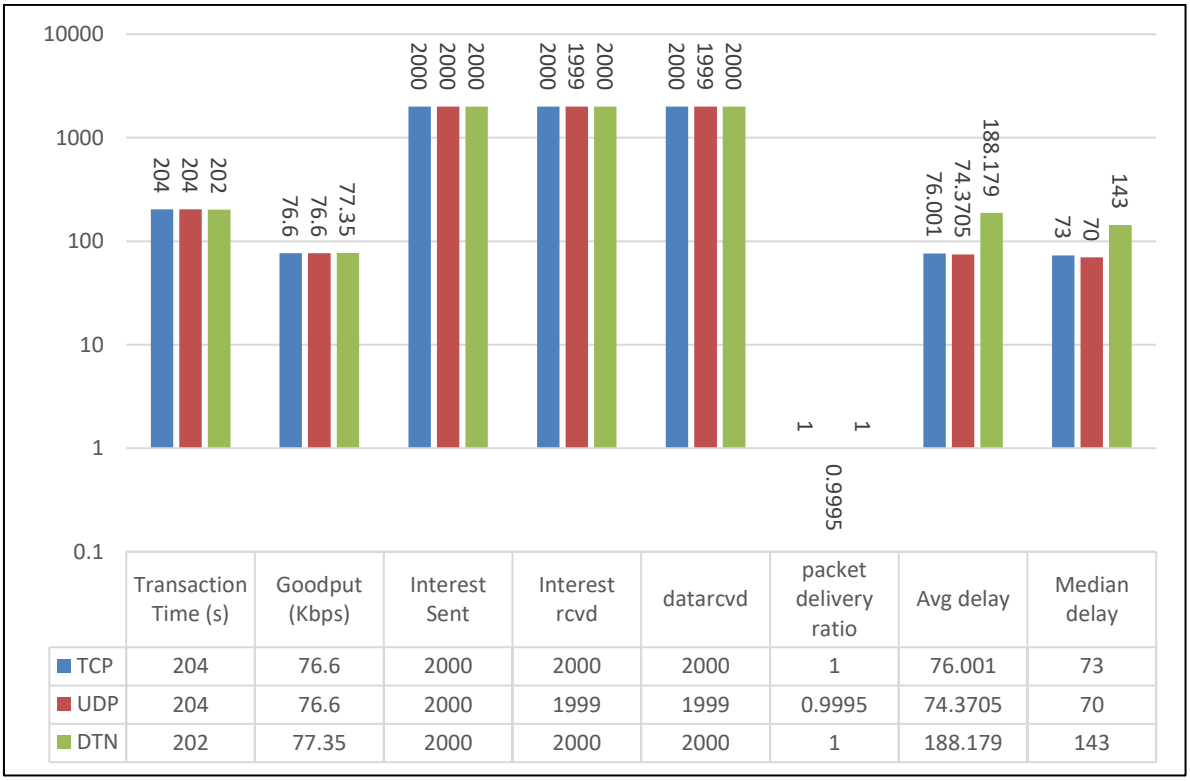


Διάγραμμα 11 – κανονική λειτουργία, πακέτα 1000B, back-to-back, SD card

Από τα διαγράμματα 11-13 παρατηρούμε ότι, υπό κανονικές συνθήκες, το DTN face πετυχαίνει αντίστοιχο goodput με τα TCP, UDP έχοντας όμως διπλάσιο RTT. Ακόμη, το UDP χάνει πακέτα για δεύτερη φορά.



Διάγραμμα 12 – κανονική λειτουργία, πακέτα 1000B, back-to-back, SD card

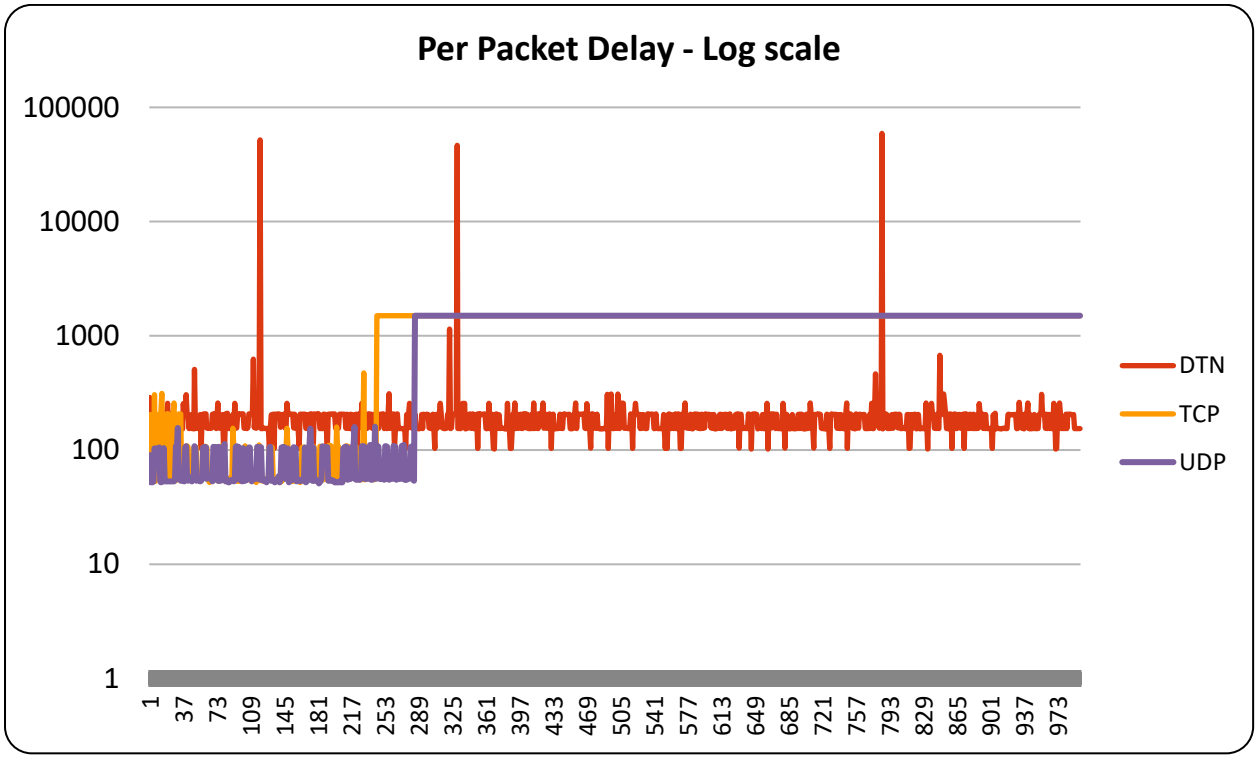


	Transaction Time (s)	Goodput (Kbps)	Interest Sent	Interest rcvd	datarcvd	packet delivery ratio	Avg delay	Median delay
TCP	204	76.6	2000	2000	2000	1	76.001	73
UDP	204	76.6	2000	1999	1999	0.9995	74.3705	70
DTN	202	77.35	2000	2000	2000	1	188.179	143

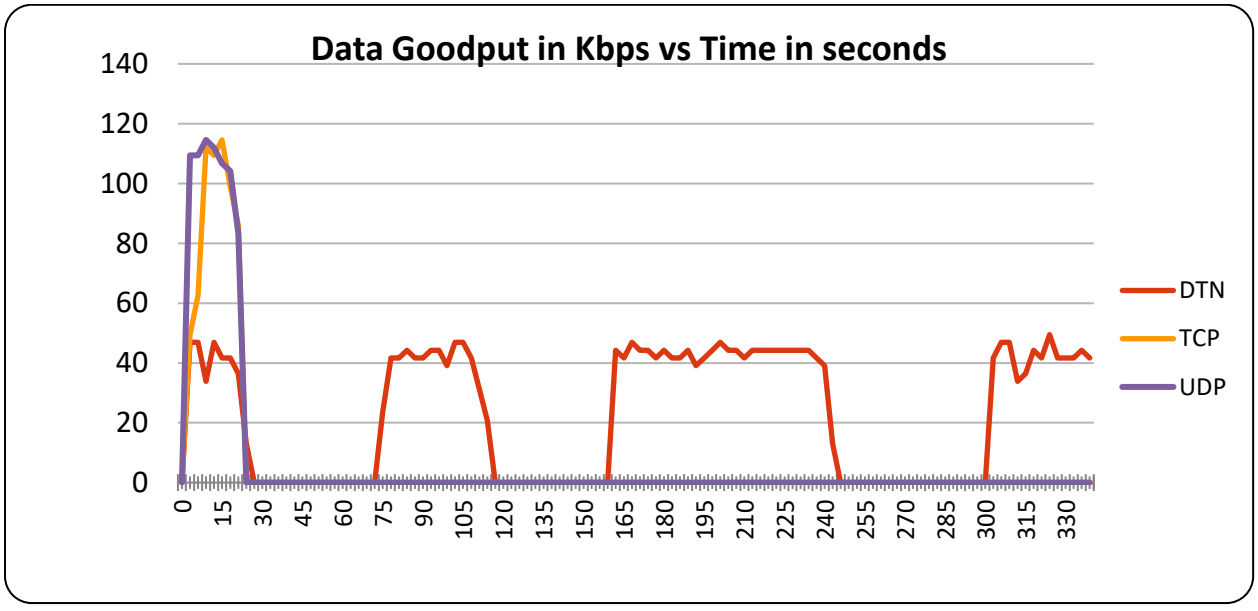
Διάγραμμα 13 – κανονική λειτουργία, πακέτα 1000B, back-to-back, SD card

Στη συνέχεια θα εξετάσουμε τον εναλλακτικό τρόπο αποστολής, σύμφωνα με τον οποίο, η αποστολή ενός νέου Interest γίνεται μόνο όταν ληφθούν τα Data που αντιστοιχούν στο προηγούμενο. Οι παράμετροι είναι: πακέτα 1000 byte, τρόπος αποστολής conversation, απώλειες τύπου 2.

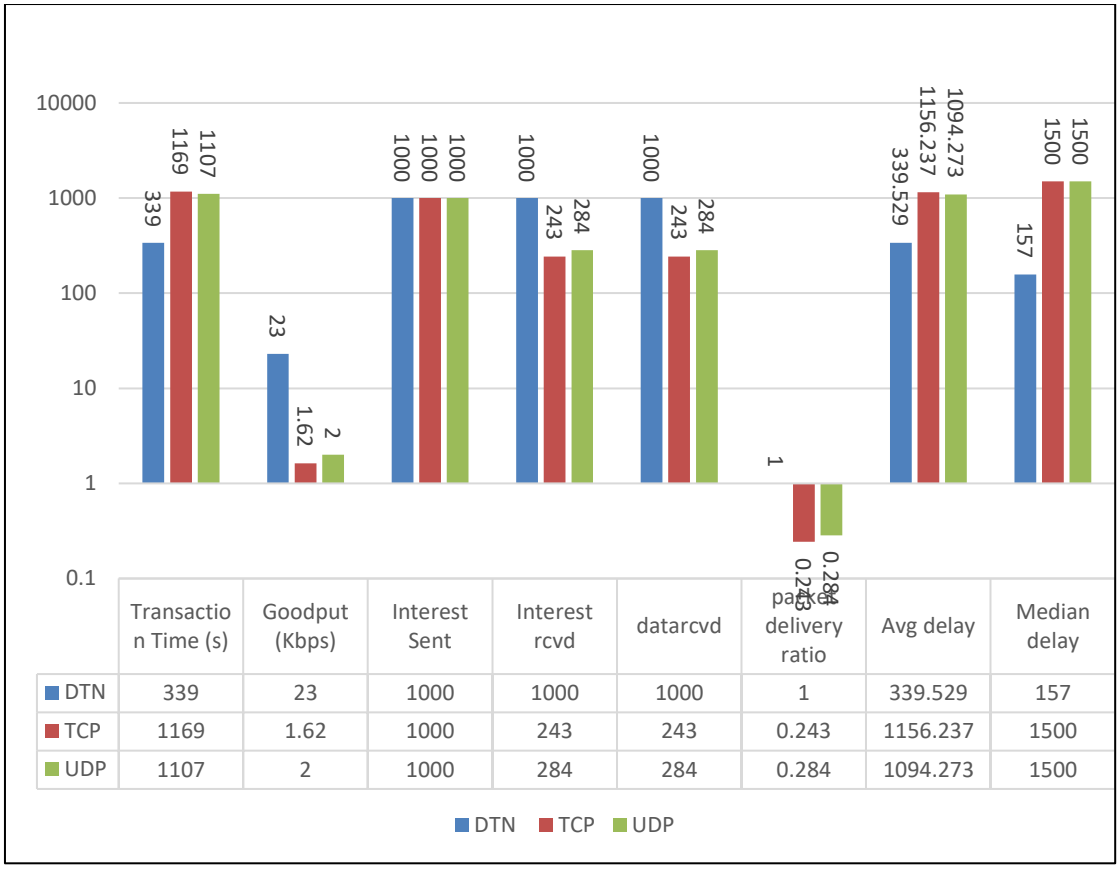
Conversation mode



Διάγραμμα 14 - τρεις διακοπές, πακέτα 1000B, conversation, SD card



Διάγραμμα 15 – τρεις διακοπές, πακέτα 1000B, conversation, SD card



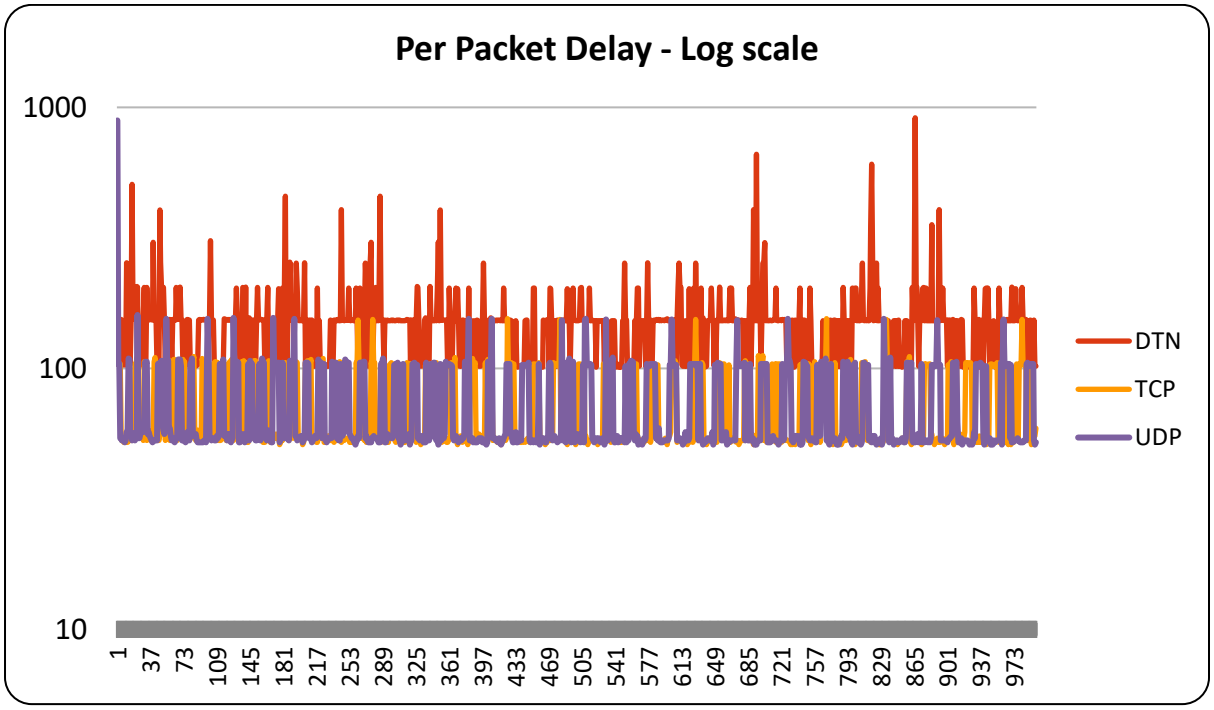
Διάγραμμα 16 – τρεις διακοπές, πακέτα 1000B, conversation, SD card

Εδώ τα πρωτόκολλα TCP και UDP αποτυγχάνουν λόγω της μη μονιμότητας των routes. Ακόμη παρατηρούμε ότι πλέον το spike καθυστέρησης κατά την απώλεια σύνδεσης αφορά ένα πακέτο αφού όσο δεν λαμβάνονται Data η πλατφόρμα δοκιμών δεν στέλνει καινούρια Interest. Αυτός είναι και ο λόγος του μεγάλου Transaction Time των UDP και TCP αφού τα timeout δεν αλληλεπικαλύπτονται αλλά προκύπτουν το ένα μετά το άλλο.

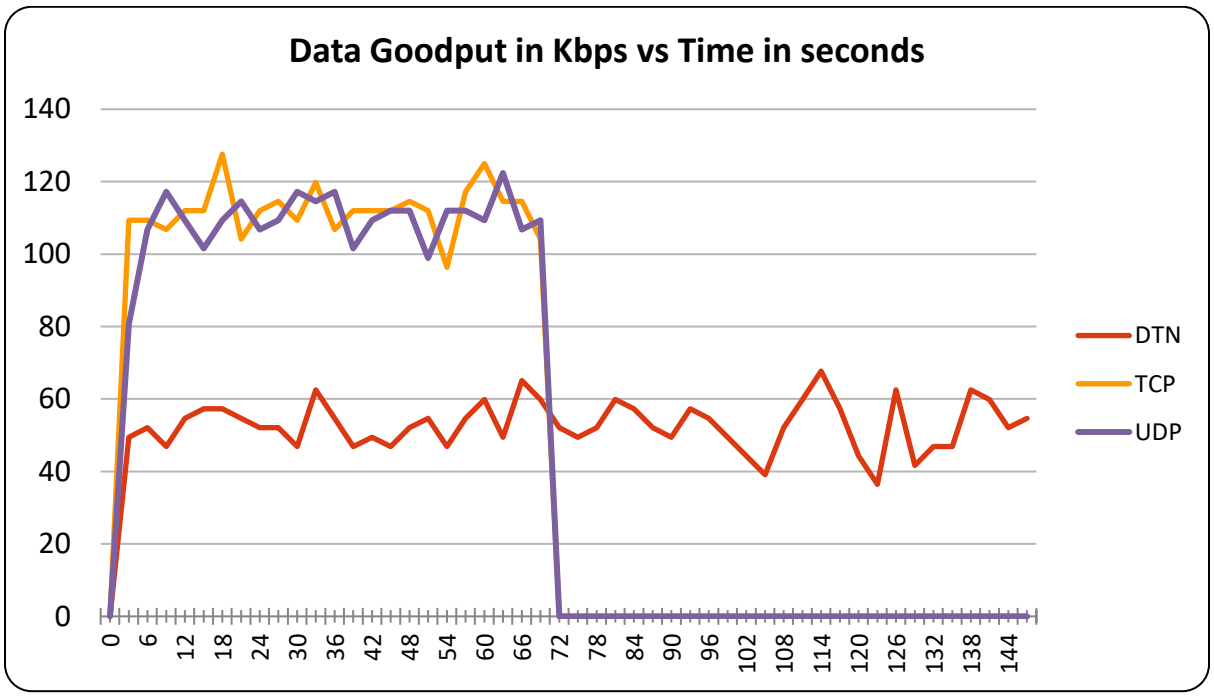
Το DTN face μειονεκτεί σημαντικά ως προς το goodput σε σχέση με την περίπτωση που τα πακέτα είναι back-to-back. Αυτό φαίνεται στο πρώτο χρονικό διάστημα (0-20 sec) αλλά θα φανεί καλύτερα υπό συνθήκες φυσιολογικής λειτουργίας. Η αιτία για το περίπου μισό goodput που πετυχαίνει το DTN face είναι φυσικά το σχεδόν διπλάσιο RTT το οποίο δεν είχε σημασία όταν τα Interest εκδίδονταν ανεξαρτήτως απόκρισης. Παρ' όλα αυτά, επιτυγχάνεται για ακόμη μια φορά 100% packet delivery ratio.

Ακολουθούν τα αποτελέσματα των δοκιμών με τις ίδιες παραμέτρους αλλά σε συνθήκες κανονικής λειτουργίας.

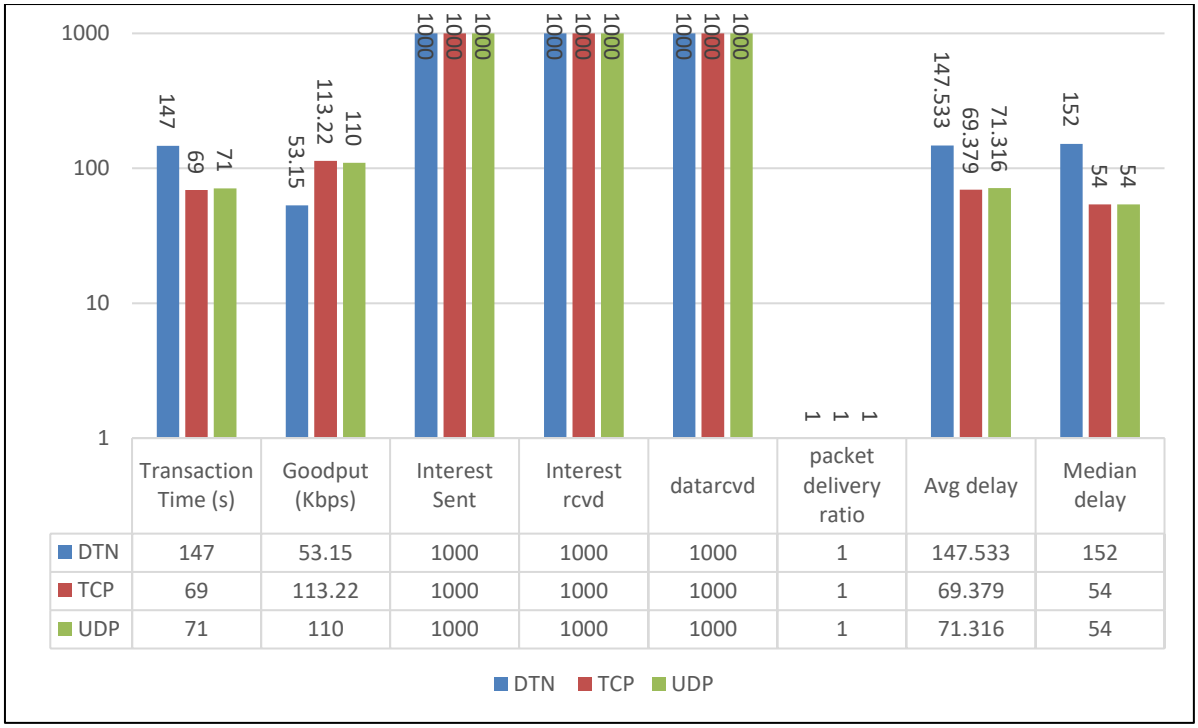
Κανονική λειτουργία, conversation



Διάγραμμα 17 – κανονική λειτουργία, πακέτα 1000B, conversation, SD card



Διάγραμμα 18 – κανονική λειτουργία, πακέτα 1000B, conversation, SD card



Διάγραμμα 19 – κανονική λειτουργία, πακέτα 1000B, conversation, SD card

Εδώ βλέπουμε καθαρά τη γραμμική σχέση RTT και goodput στο conversation mode.

Συμπερασματικά, το DTN face επιτυγχάνει το σκοπό του και παρέχει πλήρη αξιοπιστία σε εφαρμογές με χαλαρές προδιαγραφές ποιότητας υπηρεσιών. Η αποστολή των Interests back-to-back ωφελεί το DTN face αφού εξαλείφεται η επίδραση του μεγαλύτερου χρόνου RTT στο goodput.

5.2.3 Χαρακτηριστικά απόδοσης

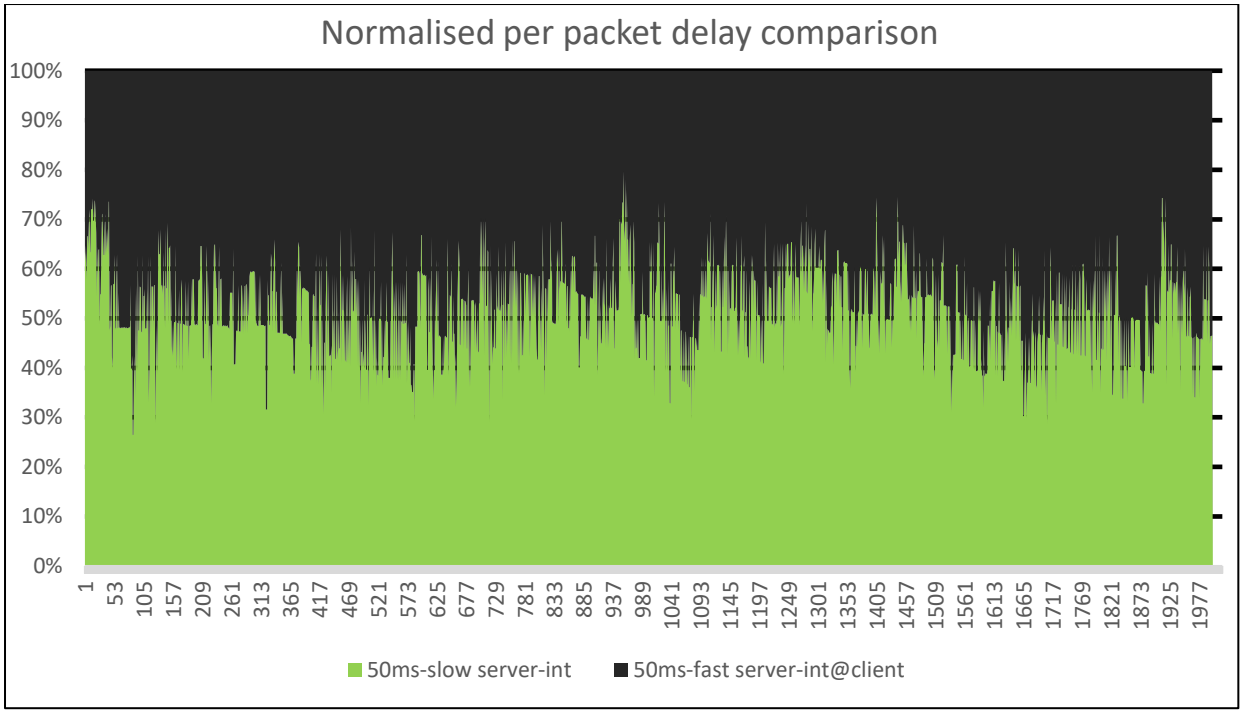
Σε αυτό το υποκεφάλαιο θα εξεταστεί η απόδοση του IBR-DTN ανάλογα με το bit-rate, την επεξεργαστική ισχύ του server και του client, την ταχύτητα του χώρου αποθήκευση, καθώς και το μέγεθος των πακέτων. Ακόμη θα παρουσιαστούν οι μετρήσεις για την επιπρόσθετη καθυστέρηση που εισάγει η διεπαφή NDN – DTN.

5.2.3α Χαρακτηριστικά απόδοσης του IBR-DTN

Για τη διεξαγωγή των πειραμάτων χρησιμοποιήθηκαν δύο διαφορετικές συσκευές Android. Η μία φέρει τον επεξεργαστή Snapdragon 410 (4 cores) ενώ η άλλη το μοντέλο Snapdragon 808 (6 cores) της Qualcomm. Οι δύο επεξεργαστές είναι μοντέλα του 2014 με λιθογραφία στα 28nm και 20nm αντίστοιχα. Το μοντέλο 808 έχει περίπου διπλάσια επεξεργαστική ισχύ σε σχέση με το 410 σε single-threaded και multi-threaded σενάρια. Στα προηγούμενα υποκεφάλαια, όλες οι δοκιμές διεξήχθησαν με τη συσκευή η οποία φέρει τον επεξεργαστή 410 να λειτουργεί ως server.

Ο IBR-DTN daemon και στον client και στον server λειτουργεί με τη λογική *store-and-forward* με την αποθήκευση να συμβαίνει στην εσωτερική μνήμη flash της συσκευής ή σε κάποια πρόσθετη sd κάρτα. Συνεπώς, ενδέχεται και η ταχύτητα αποθήκευσης να επηρεάζει την απόδοση. **Σε όλα τα προηγούμενα πειράματα χρησιμοποιήθηκε η εξωτερική κάρτα sd της «αργής» συσκευής αφού αυτός είναι ο προεπιλεγμένος χώρος αποθήκευσης του IBR-DTN.** Η «γρήγορη» συσκευή δεν διαθέτει κάρτα sd οπότε η προεπιλογή είναι η εσωτερική μνήμη flash.

Αρχικά θα χρησιμοποιήσουμε τις εξής παραμέτρους: πακέτα μεγέθους 1000 byte, back-to-back, χωρίς πρόβλημα σύνδεσης. Ακόμη, με σκοπό τον εντοπισμό του bottleneck που αυξάνει σημαντικά το RTT στα διαγράμματα 1, 5 και 8 και δεν επιτρέπει στον IBR-DTN να επαναφέρει την καθυστέρηση στα προ-διακοπής επίπεδα, **αυξήθηκε ο ρυθμός αποστολής Interest από 10 σε 20 το δευτερόλεπτο** (50 ms sending period). **Στο εξής θα χρησιμοποιηθούν stacked διαγράμματα όπου οι καθυστερήσεις είναι κανονικοποιημένες και αθροίζουν 100% ώστε να είναι ευκολότερη η σύγκριση.**

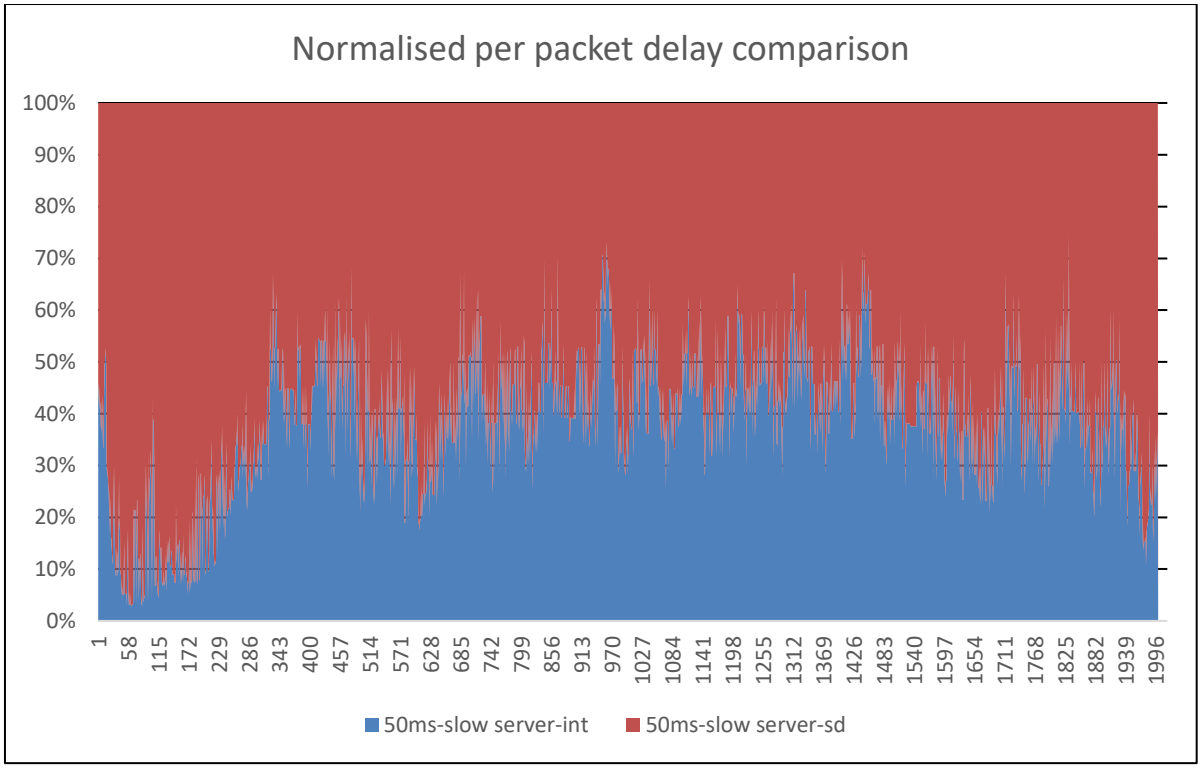


Διάγραμμα 20 – κανονική λειτουργία, σύγκριση προδιαγραφών συσκευής

Το πράσινο εμβαδό στο διάγραμμα 20 αντιστοιχεί στην περίπτωση που χρέη server εκτελεί η αργή συσκευή ενώ το μαύρο αντιστοιχεί στη γρήγορη. Και οι δύο συσκευές χρησιμοποιούν την εσωτερική τους μνήμη (int). Παρ' όλα αυτά, η ταχύτητα των εσωτερικών μνημών μπορεί να διαφέρει σημαντικά μεταξύ συσκευών. **Το βέβαιο είναι ότι και οι δύο εσωτερικές μνήμες είναι ταχύτερες και ,ίσως σημαντικότερα, έχουν μικρότερο latency (αφού πρόκειται για εγγραφή και ανάγνωση μικρών πακέτων) σε σχέση με την sd κάρτα η οποία είναι τύπου sdc4 (έως 4 MB/s).**

Συνεπώς, ο ελαφρώς μειωμένος χρόνος RTT (167 vs 150 ms) που αντιστοιχεί στη δεύτερη περίπτωση δεν μπορεί να αποδοθεί σε κάποιο συγκεκριμένο χαρακτηριστικό και ενδέχεται να αποτελεί τυχαίο σφάλμα. Ούτως ή άλλως, μέσω του server και του client διέρχεται η ίδια ποσότητα δεδομένων με τη διαφορά να είναι ότι ο client είναι αυτός που, στην αρχή του πειράματος, στέλνει μαζικά Interests (τα οποία έχουν μέγεθος λίγα byte) και έχει την ευκαιρία να υπερφορτώσει τον server από νωρίς.

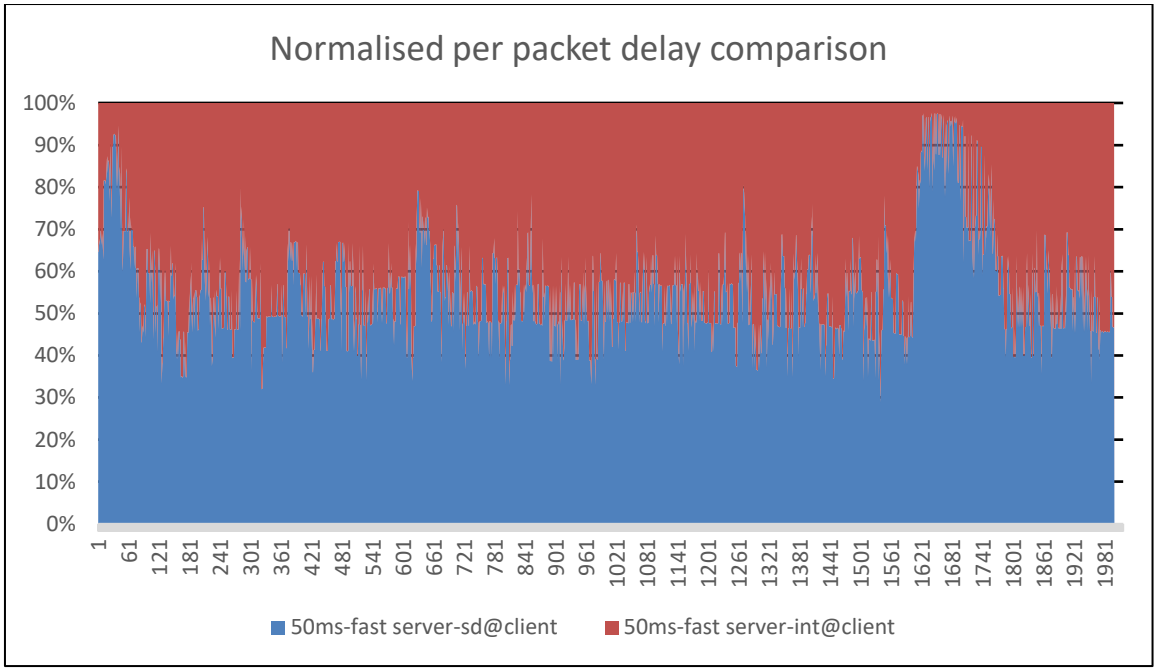
Στη συνέχεια θα εξεταστή η επίδραση της χρήσης της εξωτερικής κάρτας sd.



Διάγραμμα 21 – κανονική λειτουργία, σύγκριση μεταξύ της χρήσης κάρτας sd και εσωτερικής μνήμης από τον server

Στο διάγραμμα 21 φαίνεται η επίδραση της χρήσης της κάρτας sd ως μέσο αποθήκευσης του IBR-DTN του server έναντι της εσωτερικής μνήμης (int) όταν αυτός είναι η αργή συσκευή. Ο μέσος χρόνος RTT αυξάνεται κατά 130% από 167ms σε 383ms. Επίσης, στην αρχή του πειράματος ο server με το αργό μέσο αποθήκευσης φαίνεται να μην μπορεί να ανταποκριθεί με τρόπο παρόμοιο με τα διαγράμματα 1, 5 και 8.

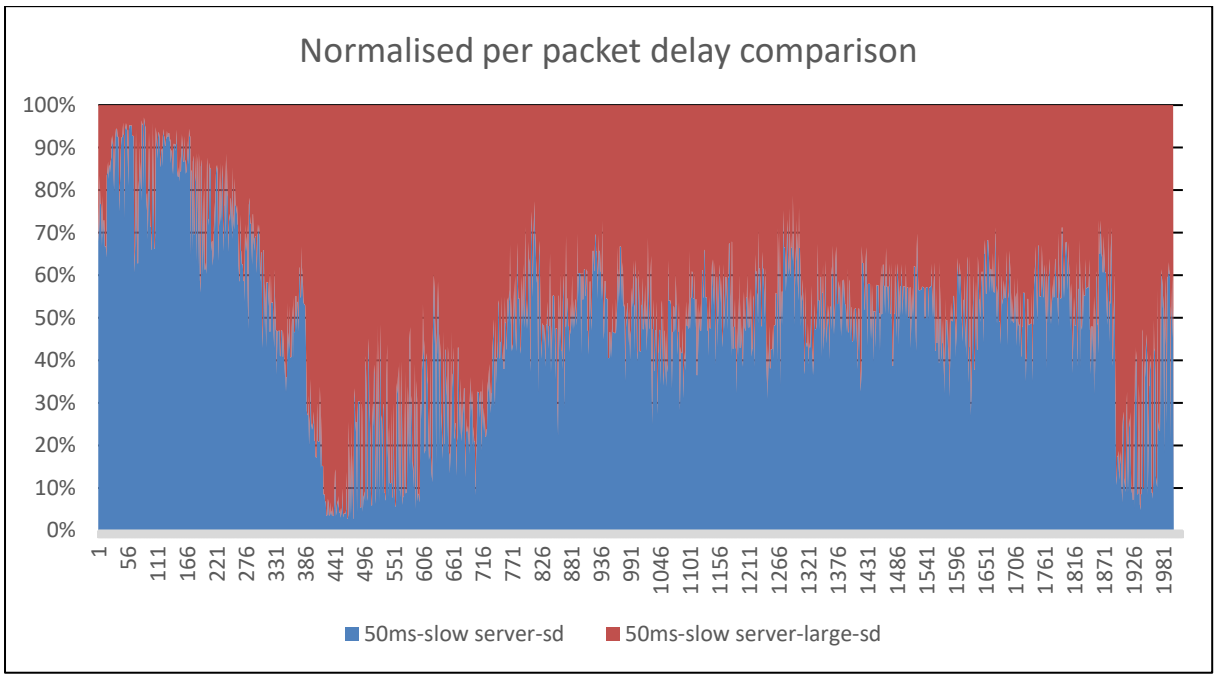
Παρόμοια είναι η συμπεριφορά όταν server είναι η γρήγορη συσκευή και ο client εναλλάσσει την εσωτερική και την εξωτερική μνήμη (διάγραμμα 22)



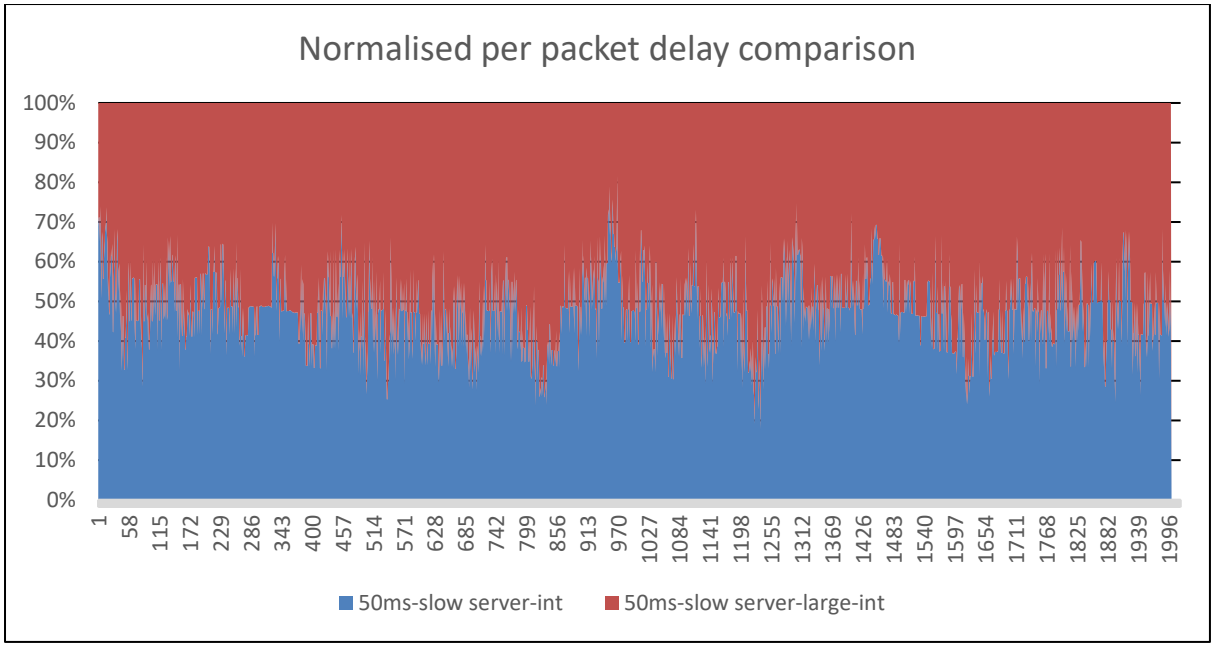
Διάγραμμα 22 – κανονικές συνθήκες, σύγκριση μεταξύ της χρήσης κάρτας sd και εσωτερικής μνήμης από τον client

Τώρα η μέση καθυστέρηση για το άνω εμβαδό είναι 150ms ενώ για το κάτω είναι 310ms.

Στα διαγράμματα 23 και 24 φαίνεται η επίδραση της χρήσης πακέτων 3500 byte αντί 1000 byte (ξανά internal vs sd card).

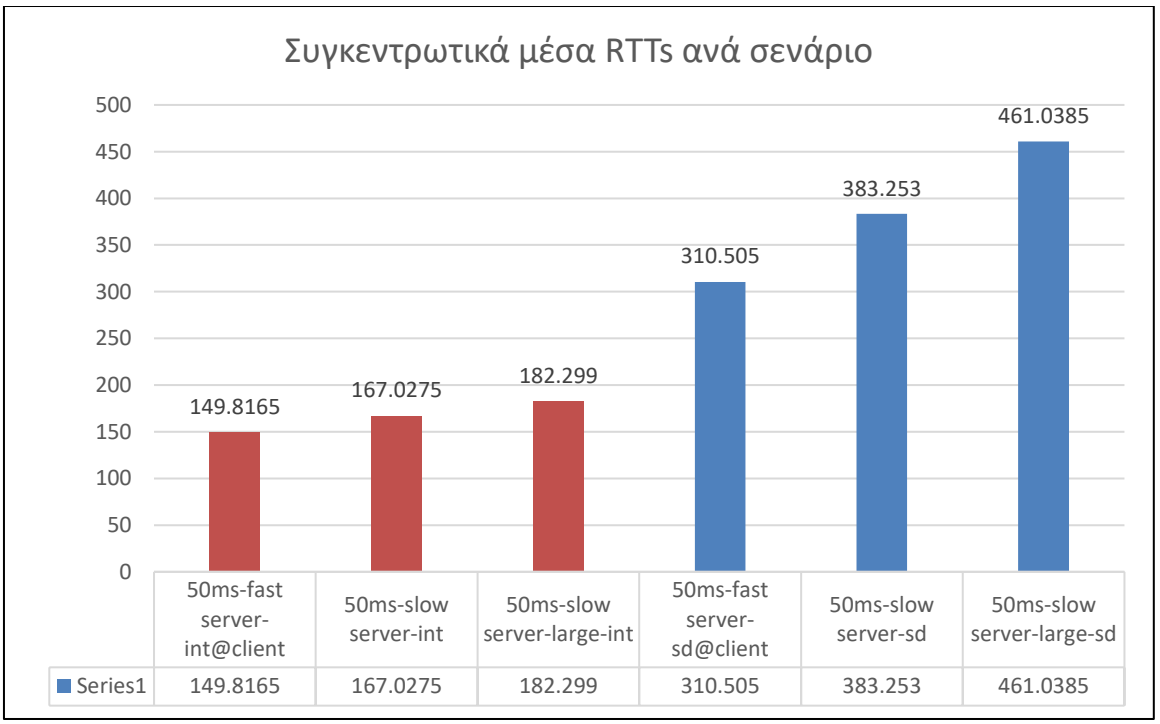


Διάγραμμα 23 – κανονικές συνθήκες, επίδραση της αύξησης του μεγέθους των πακέτων, αποθήκευση στην sd



Διάγραμμα 24 – κανονικές συνθήκες, επίδραση της αύξησης του μεγέθους των πακέτων, αποθήκευση στην εσωτερική μνήμη

Στο διάγραμμα 23 χρησιμοποιείται η κάρτα sd στον server με τον μέσο χρόνο RTT να αυξάνεται από 383ms σε 461ms λόγω της αύξησης του μεγέθους των πακέτων. Αντίστοιχα, όταν χρησιμοποιείται η εσωτερική μνήμη το μέσο RTT αυξάνεται από 167ms σε 182ms (διάγραμμα 24). Στο διάγραμμα 25 παρουσιάζονται συγκεντρωτικά οι χρόνοι RTT κάθε σεναρίου.



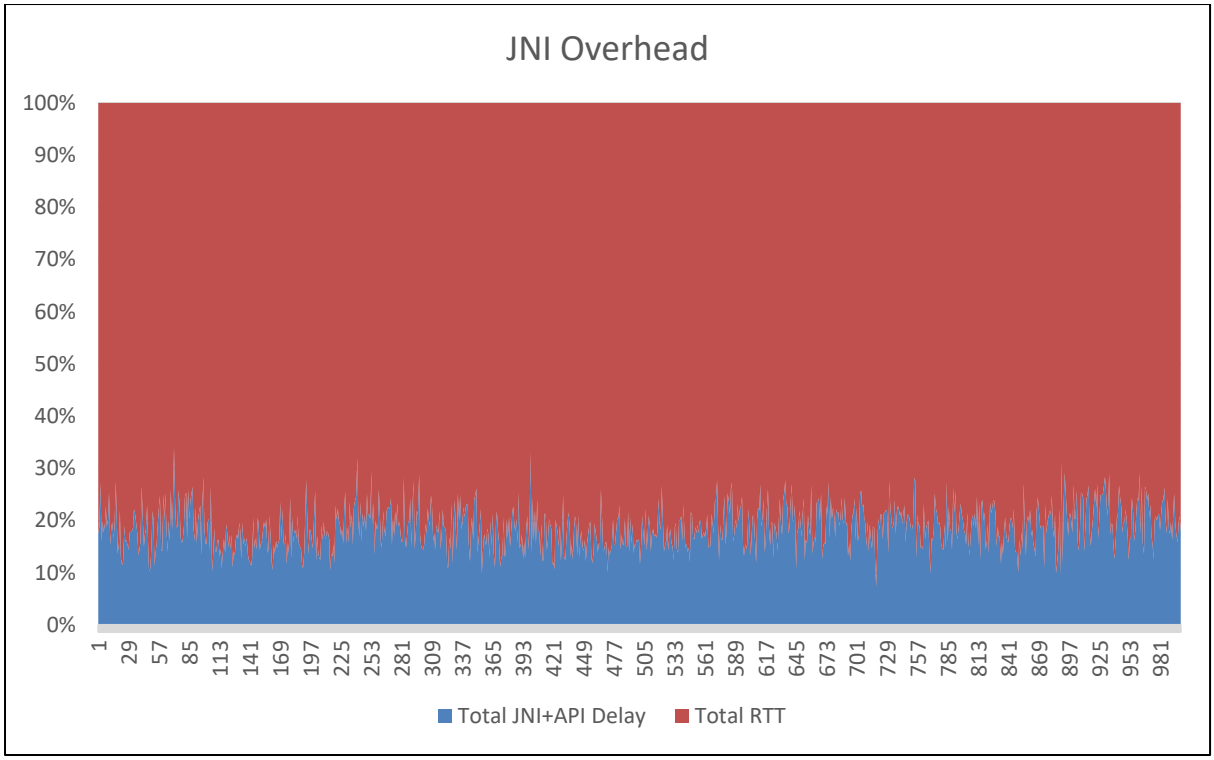
Διάγραμμα 25 – συγκεντρωτικά αποτελέσματα

Συμπερασματικά, ο περιοριστικός παράγοντας για την απόδοση του IBR-DTN **ενδεχομένως** να είναι οι δυνατότητες του μέσου αποθήκευσης (διάγραμμα 25, κόκκινο). **Όμως τα αποτελέσματα αυτά είναι ενδεικτικά**, καθώς η συμπεριφορά του IBD-DTN είναι απρόβλεπτη και φαίνεται να εξαρτάται και από την προηγούμενή του κατάσταση **όταν το bit-rate ξεπεράσει ένα κατώφλι** (**στα πειράματα που παρουσιάστηκαν στα προηγούμενα κεφάλαια χρησιμοποιήθηκε το μισό bit-rate, γεγονός που αυξάνει σε πολύ μεγάλο βαθμό την επαναληψιμότητα των δοκιμών**). Για την ενδελεχή έρευνα της απόδοσής του, είναι απαραίτητη η συγκέντρωση πολύ μεγάλου δείγματος αποτελεσμάτων. Η επίδραση των λοιπών προδιαγραφών της συσκευής καθώς και του μεγέθους των πακέτων πάνω στο χρόνο RTT είναι πιθανώς μικρή και δεν μπορεί να προσδιοριστεί με σιγουριά. Σε μία επέκταση της παρούσας εργασίας, μία από τις προτεραιότητες θα ήταν η μεγαλύτερη αυτοματοποίηση των πειραματικών διαδικασιών ώστε να είναι εφικτή η συλλογή ικανού δείγματος για στατιστική ανάλυση.

5.2.3β Χαρακτηριστικά απόδοσης της διασύνδεσης NDN – DTN

Το γραμμένο σε C++ DTN face, χρησιμοποιεί το Java Native Interface για την επικοινωνία του με το IBR-DTN API το οποίο είναι γραμμένο σε Java. Το γεγονός αυτό εισάγει σημαντικό overhead σε κάθε πακέτο και η τροποποίηση του DTN face για την ελάττωση αυτού είναι μία από τις βασικές προτεραιότητες σε μία πιθανή επέκταση της παρούσας εργασίας. Εντούτοις, οι προδιαγραφές του UMOBILE για το DTN face ορίζουν ότι θα χρησιμοποιηθεί από εφαρμογές με χαλαρές απαιτήσεις ποιότητας υπηρεσιών και σε περιπτώσεις προβληματικής δικτύωσης στις οποίες οι μεγάλες καθυστερήσεις είναι δεδομένες.

Στα πειράματα που παρουσιάστηκαν προηγουμένως οι συσκευές, όταν δεν βρίσκονταν εντός του αγώγιμου κουτιού, ήταν τοποθετημένες πολύ κοντά στον ασύρματο δρομολογητή και σε οπτική επαφή. Το γεγονός αυτό ελαχιστοποιεί την πιθανότητα απώλειας πακέτων από τα συμβατικά πρωτόκολλα επιπέδου μεταφοράς και μειώνει τον χρόνο RTT. Αυτές δεν είναι οι συνθήκες στις οποίες θα χρησιμοποιηθεί το DTN face αλλά επιλέχθηκαν για τη βελτίωση της αξιοπιστίας των πειραμάτων και την ανάγκη να είναι συγκρίσιμα τα αποτελέσματα. Στη συνέχεια θα παρουσιαστεί ένα πιο ρεαλιστικό σενάριο. Αρχικά όμως θα αναφερθούν τα αποτελέσματα σχετικά με το overhead που εισάγεται από το JNI και την χρήση του IBR-DTN API (στο εξής JNI overhead).



Διάγραμμα 26 – επιπρόσθετη καθυστέρηση λόγω της επικοινωνίας με το IBR-DTN API μέσω JNI.

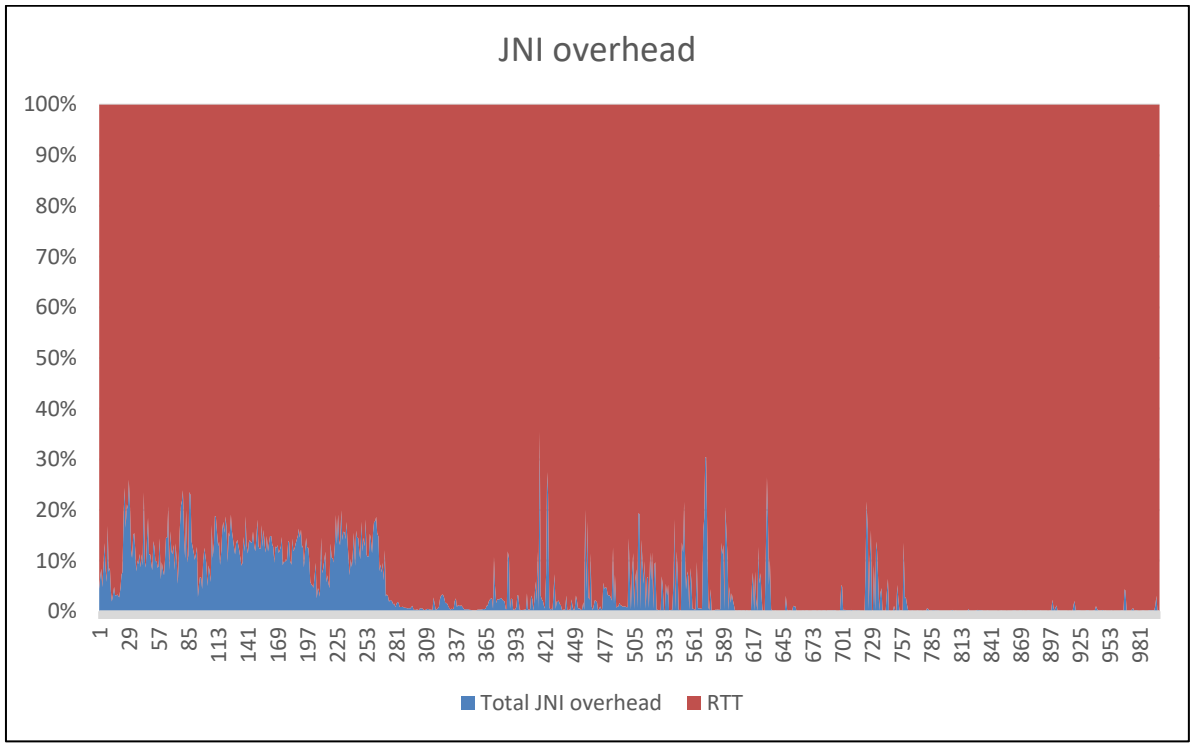
Το JNI overhead μετριέται ξεχωριστά στον NFD και του client και του server για κάθε πακέτο. Συνολικά λαμβάνονται τέσσερις μετρήσεις (ώστε να γίνει σύγκριση με το RTT) – δύο από τον κάθε ένα – οι οποίες συγκεντρώνονται και αθροίζονται. Στον client ο χρόνος κατά την αποστολή ενός Interest ξεκινάει να μετρά πριν ακριβώς το πακέτο φύγει από το DtnTransport (C++ - σχήμα 15 σελ. 25) και σταματάει όταν καλείται η εντολή send του API (Java) με επόμενο βήμα τον IBR-DTN daemon. Ο χρόνος κατά τη λήψη ξεκινά όταν ο DTNBroadCastReceiver (Java) παραλάβει το πακέτο από τον IBR-DTN και σταματά όταν αυτό φτάσει στο DtnChannel (C++). Στον server τα σημεία μέτρησης είναι τα ίδια αλλά αντιστρέφεται ο τύπος των πακέτων.

Στο διάγραμμα 26 φαίνεται το JNI overhead σε σχέση με το συνολικό RTT. Οι παράμετροι της δοκιμής είναι: πακέτα 1000 byte, back-to-back, κανονικές συνθήκες, 100ms περίοδος αποστολής. Το JNI overhead είναι ένα σημαντικό 22% και παρά τις προσπάθειες περεταίρω βελτιστοποίησης του κώδικα (π.χ.[10] και [11]) η βελτίωση ήταν της τάξης του 5% – 10%

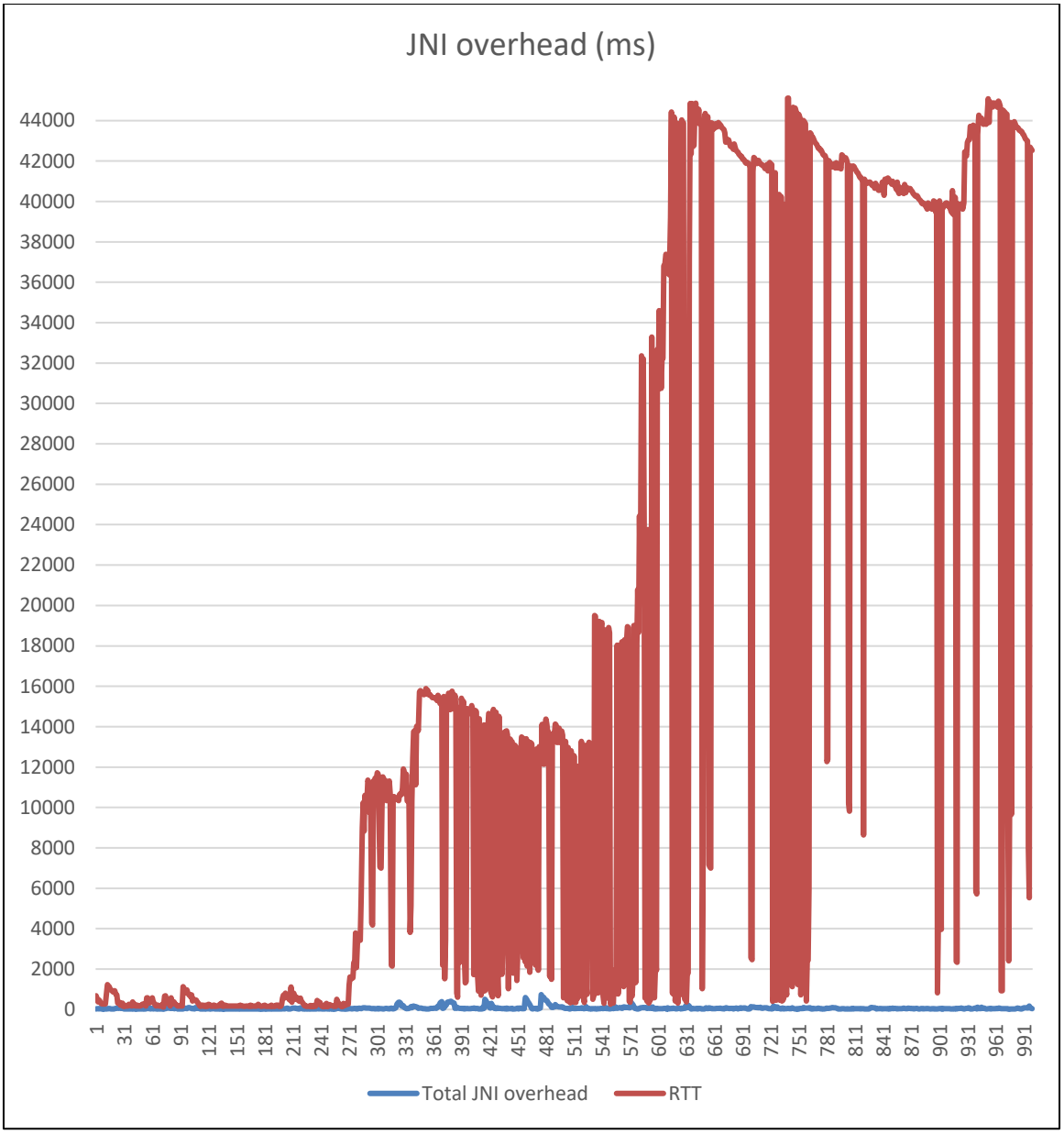
[10] https://www.ibm.com/developerworks/library/j-jni/index.html

[11] https://developer.android.com/training/articles/perf-jni.html

Στα διαγράμματα 27 και 28 παρατίθενται τα αποτελέσματα του ίδιου πειράματος σε λιγότερο ιδανικές συνθήκες.



Διάγραμμα 27 – επιπρόσθετη καθυστέρηση λόγω της επικοινωνίας με το IBR-DTN API μέσω JNI.



Διάγραμμα 28 – επιπρόσθετη καθυστέρηση λόγω της επικοινωνίας με το IBR-DTN API μέσω JNI σε απόλυτες τιμές.

Στο πείραμα αυτό η συσκευή server βρίσκεται στην ίδια θέση όμως ο client βρίσκεται σε απόσταση 12 μέτρων με εμπόδια ενδιάμεσα με συνέπεια η ένδειξη ισχύος σήματος Wi-Fi να είναι 3/4 ή σπάνια 2/4. Ακόμη και με αυτό, το όχι πολύ κακό, επίπεδο σύνδεσης, η ίδια δοκιμή με UDP οδήγησε στην απώλεια τεσσάρων στα χίλια πακέτα ενώ το μέσο RTT του DTN μέχρι το πακέτο 270 (που η καθυστέρηση είναι ακόμη ελεγχόμενη) είναι αυξημένο με τιμή 320ms. Μετά το πακέτο 271 ο συνδυασμός της μέτριας σύνδεσης και της χρήσης sd κάρτας για αποθήκευση από τον server εκτοξεύει τον χρόνο RTT. Συνολικά, σε αυτό το πείραμα, το JNI overhead είναι 0,37% του συνολικού RTT ενώ για τα πακέτα 0 έως 270 είναι 11%. Το

παραπάνω πείραμα σε καμία περίπτωση δεν εξαντλεί τα πιθανά σενάρια χρήσης. Είναι όμως ενδεικτικό των καθυστερήσεων που μπορούν να παρουσιασθούν σε περιπτώσεις χρήσης του DTN face στις οποίες η δικτυακή υποδομή είναι είτε ανεπαρκής είτε παρουσιάζει σφάλμα με συνέπεια η σύνδεση σε αυτή να είναι διακοπτόμενη.

Κεφάλαιο 6

Συμπεράσματα και Προοπτικές

Ο στόχος της εργασίας αυτής ήταν η διασύνδεση των πρωτοκόλλων NDN και DTN σε φορητές συσκευές Android, ώστε να αξιοποιηθούν τα πλεονεκτήματα της πληροφοριοκεντρικής δικτύωσης σε συνθήκες περιορισμένης, προβληματικής ή υπερφορτωμένης δικτυακής υποδομής και να επιτευχθεί αξιόπιστη επικοινωνία μεταξύ φορητών συσκευών. Οι πειραματικές μετρήσεις έδειξαν ότι το νέο protocol stack πετυχαίνει πλήρως αξιόπιστη μεταφορά δεδομένων σε συνθήκες διαλείπουσας ασύρματης σύνδεσης Wi-Fi με αντάλλαγμα τον αυξημένο χρόνο RTT (όπως συνηθίζεται στα delay tolerant πρωτόκολλα). Έτσι, καλυφθήκαν οι προδιαγραφές που θέτει η αρχιτεκτονική UMOBILE για την παροχή ενός νέου επιπέδου ποιότητας υπηρεσιών, το *less-than-best-effort,* το οποίο μπορεί να αξιοποιηθεί από εφαρμογές που έχουν χαλαρές απαιτήσεις σε latency αλλά στοχεύουν στην αξιοπιστία.

Ακόμη, το NDN-DTN protocol stack επιτρέπει στις NDN εφαρμογές να παραλείψουν την ανάπτυξη μηχανισμών εντοπισμού και αντιμετώπισης απωλειών. Η χρήση του προσφέρει δηλαδή έμφυτη (inherent) αξιοπιστία.

Πιο συγκεκριμένα, το NDN-DTN protocol stack πέτυχε 100% packet delivery ratio, σε αντίθεση με τα TCP και UDP τα οποία δεν είναι σχεδιασμένα γι’ αυτό το σκοπό, σε όλα τα πειράματα διατηρώντας το goodput στο ίδιο επίπεδο με αυτά όταν ο client έστελνε Interests back-to-back. Σε λειτουργία conversation, όπου για να σταλεί ένα Interest πρέπει πρώτα να ληφθεί το πακέτο Data που αντιστοιχεί στο προηγούμενο, το stack υπολείπεται πετυχαίνοντας, στο συγκεκριμένο σενάριο, το μισό goodput λόγω του περίπου διπλάσιου μέσου χρόνου RTT.

Όσον αφορά την απόδοση της υλοποίησης, μεταξύ των παραμέτρων του μεγέθους των πακέτων Data, των γενικών επιδόσεων συσκευής (CPU + δυνατότητες εσωτερικής μνήμης), και τη χρήση ή μη της εξωτερικής κάρτας sd για την αποθήκευση των πακέτων από τον IBR-DTN (λογική *store-and-forward*) η τελευταία φαίνεται να είναι η σημαντικότερη και αποτελεί τον περιοριστικό παράγοντα όταν το bit rate ξεπεράσει ένα κατώφλι. Ο μέσος χρόνος RTT μπορεί να περιοριστεί εάν ελαττωθεί το overhead που εισάγεται από τη χρήση του Java Native Interface. Μία βασική προτεραιότητα σε μελλοντική επέκταση της εργασίας θα είναι η παράκαμψη του JNI και η χρήση κάποιας μεθόδου για inter-process communication (π.χ. UNIX sockets). Ακόμη, η υλοποίηση θα προσαρμοστεί στη νέα έκδοση 0.2.3 του NFD για Android η οποία βελτιώνει τη διαχείριση των permanent routes[12], κάτι που αποτέλεσε

[12] https://github.com/named-data-mobile/NFD-android/blob/master/RELEASE_NOTES.rst

πρόβλημα κατά την πειραματική διαδικασία, σε σχέση με την έκδοση 0.2.1 που ήταν διαθέσιμη στην αρχή της ανάπτυξης.

Επίσης, υπάρχουν πολλές παράμετροι και σενάρια που έχει ενδιαφέρον να εξεταστούν. Ένα κύριο είναι η χρήση ενός πραγματικού δικτύου με περισσότερους κόμβους, ώστε να αναδυθούν περισσότερο τα χαρακτηριστικά του NDN (π.χ. caching). Ακόμη, έχει νόημα ή διεξαγωγή πειραμάτων χρησιμοποιώντας το πρωτόκολλο Wi-Fi Direct. Μέχρι και την έκδοση 0.2.2 το Wi-Fi direct δεν περιλαμβανόταν στον NFD όμως η λειτουργικότητα αυτή υποστηρίζεται στην παρούσα υλοποίηση μέσω του IBR-DTN και έχει χρησιμοποιηθεί με επιτυχία.

Πολλά συμπεράσματα μπορούν επίσης να εξαχθούν από τις ακόλουθες δύο συγκρίσεις:

1) τη χρήση μεγάλου μεγέθους πακέτων (λίγα Megabytes) σε συνδυασμό με DTN ενάντια στη χρήση συνηθισμένου μεγέθους πακέτων με UDP/TCP. Το σενάριο αυτό είναι δυνητικά χρήσιμο για την υπηρεσία service migration μέσω drone του UMOBILE. Ένα μοναδικό Interest προκαλεί απάντηση με το σύνολο των επιθυμητών δεδομένων και ο IBR-DTN αναλαμβάνει τα fragmentation και reassembly αντί τα δεδομένα να αποτελούνται από τμήματα και να χρειάζονται πολλαπλά round trips.
2) τη σύγκριση μεταξύ μίας εφαρμογής που φροντίζει για την επανέκδοση των Interests για τα οποία δεν λαμβάνει Data και μίας που βασίζεται αποκλειστικά στον INR-DTN.

Τέλος, μία άλλη ενδιαφέρουσα περίπτωση, ειδικά για ευκαιριακά σενάρια (σελ. 19 σχήμα 10), είναι η χρήση DTN tunneling για τη σύνδεση δύο NDN-enabled ακραίων κόμβων με τους ενδιάμεσους να προωθούν τα πακέτα μόνο μέσω IBR-DTN.

Βιβλιογραφία

1. L. Zhang, A. Afanasyev, J. Burke, V. Jacobson, k. claffy, P. Crowley C. Papadopoulos, L. Wang, B. Zhang, "Named Data Networking"
2. George Xylomenos, Christopher N. Ververidis, Vasilios A. Siris, Nikos Fotiou, Christos Tsilopoulos, Xenofon Vasilakos, Konstantinos V. Katsaros, and George C. Polyzos, A Survey of Information-Centric Networking Research. Communications Surveys Tutorials, IEEE, 16(2), Second 2014.
3. M. Handley, "Why the Internet only just works," *BT Technology Journal*, vol. 24, no. 3, pp. 119–129, July 2006.
4. J. Rexford and C. Dovrolis, "Future Internet architecture: clean-slate versus evolutionary research," *Communications of the ACM*, vol. 53, no. 9, pp. 36–40, September 2010.
5. P. Mendes, W. Moreira, S. Diamantopoulos, C. A. Sarros, D. Vardalis, I. Komnios, V. Tsaoussidis, I. Psaras, S. Rene, A. Lertsinsrubtavee, C. Molina-Jimenez, A. Sathiaseelan, L. Amaral Lopes, I. Sedano, S. Sachez Perez, S. Rute, UMOBILE architecture report (1) 2016.
6. U. Drolia, N. Mickulicz, R. Gandhi, P. Narasimhan, Krowd: A Key-Value Store for Crowded Venues.
7. I. Psaras, S. Reñé, K. V. Katsaros, V. Sourlas, G. Pavlou, N. Bezirgiannidis, S. Diamantopoulos, I. Komnios, and V. Tsaoussidis. 2016. Keyword-based mobile application sharing. In *Proceedings of the Workshop on Mobility in the Evolving Internet Architecture* (MobiArch '16). ACM, New York, NY, USA, 1-6. DOI: http://dx.doi.org/10.1145/2980137.2980141
8. Cisco. (2011, June) Visual networking index: Forrecast andmethodology, 2010-2015. White Paper. [Online]. Available: http://www.cisco.com/go/vni
9. OxCERT's blog (2017). News and views from the Network Security Team at the University of Oxford http://blogs.it.ox.ac.uk/oxcert/2014/05/22/on-reflection-emerging-denial-of-service-attacks-and-you/
10. C. Yi, A. Afanasyev, I. Moiseenko, L. Wang, B. Zhang, L. Zhang, A case for stateful forwarding plane. *Computer Communications 36 (2013) 779-791*.
11. Ioannis Psaras, Lorenzo Saino, Mayutan Arumaithurai, KK Ramakrishnan, and George Pavlou. Name-based replication priorities in disaster cases. In Computer

Communications Workshops (INFOCOM WKSHPS), 2014 IEEE Conference on, pages 434-439. IEEE, 2014.

12. I. Psaras, L. Saino, and G. Pavlou. "Revisiting resource pooling: The case for in-network resource sharing." Proceedings of the 13th ACM Workshop on Hot Topics in Networks. ACM, 2014.
13. C. M. Malliou, N Bezirgiannidis, V. Tsaoussidis, Nulk Data Transders through an Airline Delay-Tolerant Network.
14. “NFD - Named Data Networking Forwarding Daemon," http://named-data.net/doc/NFD/current/, 2014.
15. A. Afanasyev, J. Shi, B. Zhang, L. Zhang, I. Moiseenko, Y. Yu, W. Shang, Y. Li, S. Mastorakis, Y. Huang, J. P. Abraham, E. Newberry, S. DiBenedetto, C. Fan, C. Papadopoulos,D. Pesavento, G. Grassi, G. Pau, H. Zhang, T. Song, H. Yuan, H. B. Abraham, P. Crowley, S. O.Amin, V. Lehman, M. Chowdhury, and L. Wang. NDN, Technical Report NDN-0021.
16. “NFD management protocol," http://redmine.named-data.net/projects/nfd/wiki/Management, 2014.
17. D. Vardalis, NDN-DTN integration documentation. Included in UMOBILE architecture report (1) 2016 (2016).
18. NDN Packet Format Specification 0.2-2 documentation (2017). http://named-data.net/doc/NDN-TLV/current/
19. K. Scott, Bundle Protocol Specification (2017). https://tools.ietf.org/html/rfc5050

Παράρτημα 1

DtnService.java

```
public class DtnService extends DTNIntentService {

    static {
        System.loadLibrary("nfd-wrapper");
    }

    private String IBRDTN_Affix = "nfd";
    public static final String ACTION_START = "INITIALIZE";
    public static final String ACTION_SEND_MESSAGE = "SEND_MESSAGE";

    private static final String ACTION_MARK_DELIVERED = "de.tubs.ibr.dtn.example.DELIVERED";
    private static final String EXTRA_BUNDLEID = "de.tubs.ibr.dtn.example.BUNDLEID";


    public static int reveiveIntentCount = 0;
    public static int sendIntentCount = 0;
    public static int markIntentCount = 0;

    private DTNClient.Session mSession = null;
    private final IBinder mBinder = new LocalBinder();


    public native String stringFromJNI();
    public native void initializeNativeInterface();
    public native void queueBundleJNI(String source, byte[] payload);

    public final static String TAG = "DEBFIN_DtnService";

    public DtnService() {
        super(TAG);

    }

    public class LocalBinder extends Binder {
        public DtnService getService() {
            //Log.i(TAG, "LocalBinder_getService");
            return DtnService.this;
        }
    }

    @Override
    public IBinder onBind(Intent intent) {
        //Log.i(TAG, "onBind");
        return mBinder;
    }
    @Override
    public int onStartCommand(Intent intent, int flags, int startId){
        super.onStartCommand(intent, flags, startId);
        //Log.i(TAG,"DtnService_onStartCommand (does nothing)");
        return START_STICKY;
    }
```

```
@Override
public void onDestroy(){
    super.onDestroy();
    //Log.i(TAG,"DtnService_onDestroy (does nthing)");
}

@Override
public void onCreate(){
    super.onCreate();
    //Log.i(TAG,"DtnService_onCreate");
    serviceStartDtn();

    //register this Service at IBR-DTN
    Registration reg = new Registration(IBRDTN_Affix);
    try {
        initialize(reg);
    } catch (ServiceNotAvailableException e) {
        Log.i(TAG, "Service not available");
    }


}

public void sendMessage(String Destination, byte[] Payload){

        Intent i = new Intent(DtnService.this, DtnService.class);
        i.setAction(ACTION_SEND_MESSAGE);
        i.putExtra("Destination", Destination);
        i.putExtra("Data", Payload);
        startService(i);
}
  /**
 * Filter the intent-action and do corresponding work
 * @param intent received intent
 */
@Override
protected void onHandleIntent(Intent intent) {
    String action = intent.getAction();

    if (de.tubs.ibr.dtn.Intent.RECEIVE.equals(action)) {
        if (mSession == null) {
            Log.i(TAG, "mSession when receiving is null for some reason");

        }else {
                        }
        try {
            while (mSession.queryNext());
        } catch (SessionDestroyedException e) {
            Log.e(TAG, "session destroyed", e);
        } catch (NullPointerException n) {
            Log.e(TAG, "Null Pointer Exception while receiving");
        }
    }
    else if (ACTION_START.equals(action)) {
        Log.i(TAG,ACTION_START + "... onCreate needs to have been called");
```

```
    }
    //if a message has to be sent
    else if (ACTION_SEND_MESSAGE.equals(action)) {
        //Log.i(TAG,"SEND_onHandleIntent #" + String.valueOf(sendIntentCount));


        SingletonEndpoint destination = new
        SingletonEndpoint(intent.getStringExtra("Destination"));

        try {
            //create destination endpoint

            //send the given payload to destination, set bundle lifetime to 1 hour
            if (mSession == null) {
                Log.i(TAG, "mSession is null");
            }else {
                                    }
            mSession.send(destination, 3600, intent.getByteArrayExtra("Data"));
        } catch (SessionDestroyedException e) {
            Log.e(TAG, "session destroyed", e);
        } catch (NullPointerException n) {
            Log.e(TAG, "Null Pointer Exception while sending");
        }
    }
    //if a received bundle should be marked as delivered
    else if (ACTION_MARK_DELIVERED.equals(action) ) {
        try {
            BundleID id = intent.getParcelableExtra(EXTRA_BUNDLEID);
            if (id != null) mSession.delivered(id);
        } catch (SessionDestroyedException e) {
            Log.e(TAG, "session destroyed", e);
        }
    }
}
/**
 * called if service is connected to IBR-DTN
 * saves the session and sets DataHandler processing bundles
 *
 * @param session current Session
 */
@Override
protected void onSessionConnected(Session session) {
    //Log.i(TAG,"onSessionConnected");
    mSession = session;
    mSession.setDataHandler(mDataHandler);
}
```

```
    @Override
    protected void onSessionDisconnected() {
        //Log.i(TAG,"onSessionDisconnected");
        mSession = null;
    }

    private DataHandler mDataHandler = new DtnExtendedDataHandler() {
        @Override
        protected void onMessage(BundleID id, byte[] data) {

            // send to nfd
            String source = id.getSource().toString();
            String payload = data.toString();
            String altPayload = new String(data);

            byte[] by = data;

            String temp = source.substring(6);
            String[] splited = temp.split("/");

            queueBundleJNI("dtn://" + splited[0],by);

            // mark the bundle as delivered
            Intent i = new Intent(DtnService.this, DtnService.class);
            i.setAction(ACTION_MARK_DELIVERED);
            i.putExtra(EXTRA_BUNDLEID, id);
            startService(i);
        }
    };

    /*
     * Gives NFD the necessary JavaVM pointer for later calls
     */
    private void // was synchronised
    serviceStartDtn() {
        if (!m_isDtnStarted) {
            m_isDtnStarted = true;
            //Log.i(TAG,"serviceStartDtn");
            initializeNativeInterface();
        }
    }



    private boolean m_isDtnStarted = false;
}
```

DtnBroadcastReceiver.java

```
package net.named_data.nfd.utils;

import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.util.Log;

import net.named_data.nfd.service.DtnService;
/**
 * BroadcastReceiver for Intents from IBR-DTN
 */

public class DtnBroadcastReceiver extends BroadcastReceiver {

    private static final String TAG = "DEBFIN";


    @Override
    public void onReceive(Context context, Intent intent) {

        String action = intent.getAction();

        if (action.equals(de.tubs.ibr.dtn.Intent.RECEIVE))
        {
            Intent i = new Intent(context, DtnService.class);
            i.setAction(de.tubs.ibr.dtn.Intent.RECEIVE);
            context.startService(i);
        }
    }

}
```

nfd-wrapper.cpp (τμήμα)

```
#include "nfd-wrapper.hpp"
#include <string>
#include "daemon/nfd.hpp"
#include "rib/service.hpp"
#include "core/global-io.hpp"
#include "core/config-file.hpp"
#include "core/logger.hpp"
#include "core/privilege-helper.hpp"
#include <stdlib.h>
#include <boost/property_tree/info_parser.hpp>
#include <boost/thread.hpp>
#include <mutex>
#include <android/log.h>
// ------------- DTN
#include "NFD/daemon/face/DTNtest.hpp"
#include "NFD/daemon/face/dtn-transport.hpp"
#include "NFD/daemon/face/dtn-channel.hpp"
// ------------- DTN

•••
std::string initialConfig =
      "general\n"
      "{\n"
      "}\n"
      "\n"
      "log\n"
      "{\n"
      "  default_level ALL\n"
      "  NameTree INFO\n"
      "  BestRouteStrategy2 INFO\n"
      "  InternalFace INFO\n"
      "  Forwarder INFO\n"
      "  ContentStore INFO\n"
      "  DeadNonceList INFO\n"
      "}\n"
      "tables\n"
      "{\n"
      "  cs_max_packets 100\n"
      "\n"
      "  strategy_choice\n"
      "  {\n"
      "    /               /localhost/nfd/strategy/best-route\n"
      "    /localhost      /localhost/nfd/strategy/multicast\n"
      "    /localhost/nfd  /localhost/nfd/strategy/best-route\n"
      "    /ndn/broadcast  /localhost/nfd/strategy/multicast\n"
      "    /ndn/multicast  /localhost/nfd/strategy/multicast\n"
      "  }\n"
      "}\n"
      "\n"
      "face_system\n"
      "{\n"

•••
```

```
"  tcp\n"
      "  {\n"
      "    listen yes\n"
      "    port 6363\n"
      "    enable_v4 yes\n"
      "    enable_v6 yes\n"
      "  }\n"
      "\n"
      "  udp\n"
      "  {\n"
      "    port 6363\n"
      "    enable_v4 yes\n"
      "    enable_v6 yes\n"
      "    idle_timeout 600\n"
      "    keep_alive_interval 25\n"
      "    mcast no\n"
      "  }\n"
      "  dtn\n"
      "  {\n"
      "    host localhost\n"
      "    port 4550\n"
      "    endpointPrefix dtn-node1\n"
      "    endpointAffix /nfd\n"
      "  }\n"
      "  websocket\n"
      "  {\n"
      "    listen yes\n"
      "    port 9696\n"
      "    enable_v4 yes\n"
      "    enable_v6 yes\n"
      "  }\n"
      "}\n"
      "\n"
      "authorizations\n"
      "{\n"
      "  authorize\n"
      "  {\n"
      "    certfile any\n"
      "    privileges\n"
      "    {\n"
      "      faces\n"
      "      fib\n"
      "      strategy-choice\n"
      "    }\n"
      "  }\n"
      "}\n"
      "\n"
      "rib\n"
      "{\n"
      "  localhost_security\n"
      "  {\n"
      "    trust-anchor\n"
      "    {\n"
      "      type any\n"
      "    }\n"
      "  }\n"
      "\n"
      "  auto_prefix_propagate\n"
      "  {\n"
      "    cost 15\n"
      "    timeout 10000\n"
      "    refresh_interval 300\n"
      "    base_retry_wait 50\n"
      "    max_retry_wait 3600\n"
      "  }\n"
      "}\n"
      "\n";
```

...

DTN

```
static JavaVM *jvm;
JNIEnv *GlobalEnv;

jclass DtnServiceClass;
jobject DtnServiceObject;
jmethodID mid;
nfd::DtnChannel *pDtnChannel;

void
printPayload2(uint8_t*,int);

jint JNI_OnLoad(JavaVM* vm, void* reserved)
{
    JNIEnv* env;
    if (vm->GetEnv(reinterpret_cast<void**>(&env), JNI_VERSION_1_6) != JNI_OK) {

        return -1;
    }

    jclass cls = env->FindClass("net/named_data/nfd/service/DtnService");

    DtnServiceClass = (jclass)env->NewGlobalRef(cls);
    env->DeleteLocalRef(cls);

    int gotVM = env->GetJavaVM(&jvm);

    mid = env->GetMethodID(DtnServiceClass, "sendMessage", "(Ljava/lang/String;[B)V");

    return JNI_VERSION_1_6;
}

////////////////////////////////////////////////////////////////////////////////////////
std::string jstring2string(JNIEnv *env, jstring jStr) {

    if (!jStr)
        return "";

    std::vector<char> charsCode;
    const jchar *chars = env->GetStringChars(jStr, NULL);
    jsize len = env->GetStringLength(jStr);
    jsize i;

    for( i=0;i<len; i++){
        int code = (int)chars[i];
        charsCode.push_back( code );
    }
    env->ReleaseStringChars(jStr, chars);

    return  std::string(charsCode.begin(), charsCode.end());
}
```

```
// Queue Received Bundle
JNIEXPORT void JNICALL
Java_net_named_1data_nfd_service_DtnService_queueBundleJNI(
        JNIEnv *env,
        jobject obj,
        jstring rE,
        jbyteArray pl){

        int len = (int)(env->GetArrayLength(pl));

        jbyte* temp = env->GetByteArrayElements(pl,NULL);

        uint8_t* payload = (uint8_t*) temp;

        std::string remoteEndpoint = jstring2string(env,rE);

        pDtnChannel->queueBundle(remoteEndpoint, payload, len);

        env->ReleaseByteArrayElements(pl, temp, 0);
}

void
registerChannelToWrapper(nfd::DtnChannel *pChannel);

// Send packet to IBRDTN.
void
sendToIBRDTN(std::string destination, std::string data){
        JNIEnv *env;
                if (jvm != NULL){

                    int attachOK = jvm->AttachCurrentThread(&env, NULL);
                    // TODO 1 was stupid
                    int envStat = jvm->GetEnv((void**)&env, JNI_VERSION_1_6);

                }
        const char *cDestination = destination.c_str();

        jstring jDestination = env->NewStringUTF(cDestination);

        const uint8_t* ui8Data = reinterpret_cast<const uint8_t*>(&data[0]);

        jbyte* tempjData = (jbyte*)ui8Data;


        jbyteArray jData = env->NewByteArray(data.length());

        env->SetByteArrayRegion(jData,0,data.length(),tempjData);
        // TODO add null exception HANDLING
        env->CallVoidMethod(DtnServiceObject,mid,jDestination,jData);

        jvm->DetachCurrentThread();
}
```

```
// Get the DTN channel object so that it can be passed by queueBundle
void
registerChannelToWrapper(nfd::DtnChannel *pChannel){
    pDtnChannel = pChannel;
}

// Get the Java VM for later calls (sendToIBRDTN)
JNIEXPORT void JNICALL
Java_net_named_1data_nfd_service_DtnService_initializeNativeInterface(
        JNIEnv *env,
        jobject obj){

    int gotVM = env->GetJavaVM(&jvm);

    DtnServiceObject = env->NewGlobalRef(obj);

}



...
```

dtn-transport.cpp

```
#include "dtn-transport.hpp"
#include "../../../nfd-wrapper.hpp"
#include <time.h>
#include <android/log.h>
#define LOG_TAG "DEBFIN"
#define LOGD(...) __android_log_print(ANDROID_LOG_DEBUG, LOG_TAG, __VA_ARGS__)
namespace nfd {
namespace face {
NFD_LOG_INIT("DtnTransport");

DtnTransport::DtnTransport(std::string localEndpoint, std::string remoteEndpoint, std::string
ibrdtndHost, uint16_t ibrdtndPort):
              m_ibrdtndHost(ibrdtndHost), m_ibrdtndPort(ibrdtndPort)
{
  this->setLocalUri(FaceUri("dtn-node1",""));
  this->setRemoteUri(FaceUri(remoteEndpoint.substr(6),""));
  this->setScope(ndn::nfd::FACE_SCOPE_NON_LOCAL);
  this->setPersistency(ndn::nfd::FACE_PERSISTENCY_PERSISTENT);
  this->setLinkType(ndn::nfd::LINK_TYPE_POINT_TO_POINT);
  this->setMtu(MTU_UNLIMITED);

  NFD_LOG_FACE_INFO("Creating DTN transport");
}
// receive bundle ----------called by dtn-channel::processBundle-----------------------------
void
DtnTransport::receiveBundle(uint8_t* payload, int len)
{
        //NFD_LOG_FACE_INFO("Received: " << b.getPayloadLength() << " payload bytes");
        bool isOk = false;

        //TLV block
        Block element;

        std::tie(isOk, element) = Block::fromBuffer(payload, len);

        if (!isOk) {
            NFD_LOG_FACE_WARN("Failed to parse incoming packet");
            // This packet won't extend the face lifetime
            return;
        }
            // create Packet object with Block element as input
        Transport::Packet tp(std::move(element));
        tp.remoteEndpoint = 0;

        delete[] payload;
        this->receive(std::move(tp));
}
// ---------------------------------------------------------------------------------------

void
DtnTransport::beforeChangePersistency(ndn::nfd::FacePersistency newPersistency)
{
}
```

```
void
DtnTransport::doClose()
{
}

void
DtnTransport::doSend(Transport::Packet&& packet)
{

        std::string localURI = getLocalUri().toString();
        std::string localHost = getLocalUri().getHost();
        std::string ibrdtnHost = m_ibrdtndHost;



        std::string str(packet.packet.begin(),packet.packet.end());

        uint8_t* databuf = reinterpret_cast<uint8_t*>(&str[0]);
        int len = str.length();

        std::string destinationAddress = getRemoteUri().getScheme() + "://" +
getRemoteUri().getHost() + getRemoteUri().getPath();


        sendToIBRDTN(destinationAddress,str);

}

} // namespace face
} // namespace nfd
```

dtn-channel.cpp

```
#include "dtn-channel.hpp"
#include "dtn-transport.hpp"
#include <string>
#include <sys/stat.h> // for chmod()
#include "generic-link-service.hpp"
#include "core/scheduler.hpp"
// TODO
#include "../../../nfd-wrapper.cpp"
#include <android/log.h>
#define LOG_TAG2 "DEBFIN"
#define LOGD2(...) __android_log_print(ANDROID_LOG_DEBUG, LOG_TAG2, __VA_ARGS__)

namespace nfd {

NFD_LOG_INIT("DtnChannel");


//class DtnChannel-------------------------------------------------------------------------------
DtnChannel::DtnChannel(const std::string &endpointPrefix, const std::string &endpointAffix,
const std::string &ibrdtnHost, uint16_t ibrdtndPort)
  : m_endpointPrefix(endpointPrefix), m_endpointAffix(endpointAffix),
m_ibrdtnHost(ibrdtnHost), m_ibrdtndPort(ibrdtndPort) // "","nfd","localHost",4550"
{
  m_is_open = false;

  setUri(FaceUri(m_endpointPrefix, m_endpointAffix));
}

DtnChannel::~DtnChannel()
{
  if (isListening()) {
    m_is_open = false;
  }
}
//-------------------------------------------------------------------------------Class DtnChannel

void
DtnChannel::listen(const FaceCreatedCallback& onFaceCreated,
                         const FaceCreationFailedCallback& onReceiveFailed)
                         //int backlog/* = acceptor::max_connections*/)
{
  if (isListening()) {
    NFD_LOG_WARN("[" << m_endpointAffix << "] Already listening");
    return;
  }
  m_is_open = true;

  m_onFaceCreated = onFaceCreated;
  m_onReceiveFailed = onReceiveFailed;

  std::string app = m_endpointAffix.substr(1); // Remove leading '/'

  ioService = &getGlobalIoService();
  registerChannelToWrapper(this);
}
//-----------------------------------------------------------------------------------------Listen
```

```
void
DtnChannel::queueBundle(std::string remoteEndpoint, uint8_t* payload, int len)
{
        auto f1 = boost::bind(&DtnChannel::processBundle, this, _1, _2, _3);
        uint8_t *payloadHeap = new uint8_t[len];
        std::copy(payload, payload + len, payloadHeap);
        ioService->post(       [=] {f1(remoteEndpoint, payloadHeap, len);}         );
}
// sends bundle to the face for processing
//processBundle-------------------------------------------------------------------------
void
DtnChannel::processBundle(std::string remoteEndpoint, uint8_t* payload, int len)
{
      NFD_LOG_INFO("DTN bundle received from " << remoteEndpoint);
      NFD_LOG_DEBUG("[" << m_endpointAffix << "] New peer " << remoteEndpoint);

      bool isCreated = false;
      shared_ptr<Face> face = nullptr;

      std::tie(isCreated, face) = createFace(remoteEndpoint,
      ndn::nfd::FACE_PERSISTENCY_ON_DEMAND);
      if (face == nullptr)
      {
            NFD_LOG_WARN("[" << m_endpointAffix << "] Failed to create face for peer " <<
remoteEndpoint);
            if (m_onReceiveFailed)
                  m_onReceiveFailed(504,remoteEndpoint); // ADDED 504 (random error code) as
the callback has changed
            return;
      }
      m_onFaceCreated(face);
      // dispatch the bundle to the face for processing
      static_cast<face::DtnTransport*>(face->getTransport())->receiveBundle(payload,len);
}

//connect-------------------------------------------------------------------------------
// called by dtn-factory::createFace
void
DtnChannel::connect(const std::string &remoteEndpoint,
                    ndn::nfd::FacePersistency persistency,
                    const FaceCreatedCallback& onFaceCreated,
                    const FaceCreationFailedCallback& onConnectFailed)
{
  shared_ptr<Face> face;
  face = createFace(remoteEndpoint, persistency).second;
  if (face == nullptr)
  {
    NFD_LOG_WARN("[" << m_endpointAffix << "] Connect failed");
    if (onConnectFailed)
      //onConnectFailed(remoteEndpoint); //compile fails
    return;
  }
  // Need to invoke the callback regardless of whether or not we had already
  // created the face so that control responses and such can be sent
  onFaceCreated(face);
}
//-------------------------------------------------------------------------------connect
```

```
//createFace-----------------------------------------------------------------------------
std::pair<bool, shared_ptr<Face>>
DtnChannel::createFace(const std::string& remoteEndpoint, ndn::nfd::FacePersistency
persistency)
{
  auto it = m_channelFaces.find(remoteEndpoint);
  if (it != m_channelFaces.end()) {
    // we already have a face for this endpoint, just reuse it
      auto face = it->second;
      face->setPersistency(persistency);
    return {false, face};
  }
  // else, create a new face
  // Create Link Service
  auto linkService = make_unique<face::GenericLinkService>();

  // Create Transport
  std::string localEndpoint = m_endpointPrefix + m_endpointAffix;
  auto transport = make_unique<face::DtnTransport>(localEndpoint, remoteEndpoint,
m_ibrdtnHost, m_ibrdtndPort );

  // Create Face
  auto face = make_shared<Face>(std::move(linkService), std::move(transport));
  face->setPersistency(persistency);
  m_channelFaces[remoteEndpoint] = face;
  connectFaceClosedSignal(*face,
    [this, remoteEndpoint] {
      NFD_LOG_TRACE("Erasing " << remoteEndpoint << " from channel face map");
      m_channelFaces.erase(remoteEndpoint);
    });
  return {true, face};
}
//accept---------------------------------------------------------------------------------
void
DtnChannel::accept(const FaceCreatedCallback& onFaceCreated,
                         const FaceCreationFailedCallback& onAcceptFailed)
{
}

} // namespace nfd
```

dtn-factory.cpp

```
#include "dtn-factory.hpp"
namespace nfd {
shared_ptr<DtnChannel>
DtnFactory::createChannel(const std::string &endpointPrefix, const std::string &endpointAffix,
const std::string ibrdtndHost, uint16_t ibrdtndPort)
{
  // checks for existing channel
  auto channel = findChannel(endpointAffix);
  if (channel)
    return channel;
  // creates channel
  channel = make_shared<DtnChannel>(endpointPrefix, endpointAffix, ibrdtndHost, ibrdtndPort);
  // adds channel to the map
  m_channels[endpointAffix] = channel;
  return channel;
}
void
DtnFactory::createFace(const FaceUri& uri,
                       ndn::nfd::FacePersistency persistency,
                       bool wantLocalFieldsEnabled,
                       const FaceCreatedCallback& onCreated,
                       const FaceCreationFailedCallback& onConnectFailed)
{
  std::string dtnEndpoint = uri.getScheme() + "://" + uri.getHost() + uri.getPath();
  std::string dtnAffix = uri.getPath();
  dtnAffix = "/nfd";
  // look through the map of channels, if you find the one with the right affix connect to it.
  for (const auto& i : m_channels) {
      if (i.first == dtnAffix ) {
        i.second->connect(dtnEndpoint, persistency, onCreated, onConnectFailed);
        return;
      }
  }
}

std::vector<shared_ptr<const Channel>>
DtnFactory::getChannels() const
{
  std::vector<shared_ptr<const Channel>> channels;
  channels.reserve(m_channels.size());
  for (const auto& i : m_channels)
    channels.push_back(i.second);
  return channels;
}
// returns the existing channel or nullptr
shared_ptr<DtnChannel>
DtnFactory::findChannel(const std::string &endpointAffix) const
{
  auto i = m_channels.find(endpointAffix);
  if (i != m_channels.end())
    return i->second;
  else
    return nullptr;
}

} // namespace nfd
```

face-manager.cpp (τμήμα) [17]

```
...
void
FaceManager::processSectionDtn(const ConfigSection& configSection, bool isDryRun)
{

  std::string ibrdtndHost = "localhost";
  uint16_t ibrdtndPort = 4550;
  std::string endpointPrefix = "";
  std::string endpointAffix = "nfd";

  for (const auto& i : configSection) {
        if (i.first == "host") {
                    ibrdtndHost = i.second.get_value<std::string>();
              NFD_LOG_TRACE("IBRDTND host set to " << ibrdtndHost);
        }
        else if (i.first == "port") {
                    ibrdtndPort = ConfigFile::parseNumber<uint16_t>(i, "dtn");
              NFD_LOG_TRACE("IBRDTND port set to " << ibrdtndPort);
        }
        else if (i.first == "endpointPrefix") {
                    endpointPrefix = i.second.get_value<std::string>();
              NFD_LOG_TRACE("IBRDTND endpoint prefix set to " << endpointPrefix);
        }
        else if (i.first == "endpointAffix") {
                    endpointAffix = i.second.get_value<std::string>();
              NFD_LOG_TRACE("IBRDTND endpoint affix set to " << endpointAffix);
        }
  }

  if (!isDryRun) {
    if (m_factories.count("dtn") > 0) {
      return;
    }

    NFD_LOG_INFO("Setting up DTN");
    shared_ptr<DtnFactory> factory = make_shared<DtnFactory>();
    m_factories.insert(std::make_pair("dtn", factory));
    // create channel
    shared_ptr<DtnChannel> dtnChannel = factory->createChannel(endpointPrefix, endpointAffix,
ibrdtndHost, ibrdtndPort);

    dtnChannel->listen(bind(&FaceTable::add, &m_faceTable, _1), nullptr);
    NFD_LOG_INFO("DTN setup finished");
  }
}
```

Παράρτημα 2

MainActivity.java

```
package com.duth.konspras.thesisexperiments;
import android.content.Intent;
import android.os.Handler;
import android.os.Message;
import android.os.Messenger;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.AdapterView;
import android.widget.Spinner;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.TextView;
import net.named_data.jndn.Face;
import java.util.Arrays;
import com.duth.konspras.thesisexperiments.InterestSenderThread;
import java.util.LinkedList;

public class MainActivity extends AppCompatActivity{

    private String[] currentNeighbours;
    private final static String TAG = "Logo_Main";
    private static Face m_face = null;
    private static volatile int interestOffset = 0;
    private static boolean stopSendingInterests = false;
    public static volatile LinkedList GoodputList;
    public static TextView InterRecView;
    // Testing
    public static volatile int numOfDataPacksORTimeoutsReceived = 0;
    public static final int interestReceiverThreadPeriod = 50;
    public static int interestLifetime_ms;
    public static boolean stopGoodputCalc = false;
    // Parameters
    public static final int numOfInterestsToSend = 2000;
    public static final int sendInterestPeriod_ms = 100;
    public static String currentProt = "udp";
    public static final int DTNinterestLifetime_ms = 20000000;
    public static final int UDPinterestLifetime_ms = 1500;
    public static final int TCPinterestLifetime_ms = 1500;
    public static final double freshnessPeriod_ms = 1000;
    public static final int packetSize_bytes = 3500;
    public static final boolean isBackToBack = false;
    // Results
    public static long[] interestSendTime;
    public static long[] dataReceiveTime;
    public static long[] diffs;
    public static long startTime;
    public static long stopTime;
    public static long endTime;
    public static double goodput;
    public volatile static long dataPacksRec = 0;
```

```
    public volatile static long interestPacksRec = 0;
    public static double totalData_bytes;

    public static void setFace(Face face){
        m_face = face;
    }
    public static Face getFace(){
        return m_face;
    }
    public static void incrOffset(){
        interestOffset++;
    }
    public static int getOffset(){
        return interestOffset;
    }
    public static boolean getStopSendingInterests(){
        return stopSendingInterests;
    }
    public static void updServerGui(){
        InterRecView.setText(String.valueOf(interestPacksRec));
    }
    public static void experimentStopped(){
        Log.i(TAG,"==================== Experiment Stopped ====================");
        Log.i(TAG, "Data or Timeout Num: " + numOfDataPacksORTimeoutsReceived);
        stopTime = System.currentTimeMillis();
    }
    public static void experimentEnded(){
        endTime = System.currentTimeMillis();
        experimentStopped();

        stopGoodputCalc = true;
        Log.i(TAG,"==================== Experiment Ended ====================");
        stopSendingInterests = true;
        Log.i(TAG,"----------");
        calcGoodput();
        Log.i(TAG,"----------");
        totalData_bytes = dataPacksRec*packetSize_bytes;
        Log.i(TAG, "Total Data (bytes): " + String.valueOf(totalData_bytes));
        Log.i(TAG,"----------");
        calcMomentGood();
        Log.i(TAG,"----------");
        calcDelay();
        Log.i(TAG,"----------");
    }

    public static void calcDelay(){
        Log.i(TAG,"calcDelay ARRAY SIZE IS: (int) - (data) " + interestSendTime.length + "-" +
dataReceiveTime.length);
        diffs = new long[numOfInterestsToSend];
        for (int i=0;i<numOfInterestsToSend;i++){
            diffs[i] = dataReceiveTime[i] - interestSendTime[i];
            // trying to waste time so that logging works (misses lines otherwise)
            Log.i(TAG,"," + i + "," + diffs[i]);
            for (int j = 0;j <100000; j++){
                double k = 1;
                k++;
                k--;
            }

        }
    }
```

```
    public static void calcMomentGood(){
        Log.i(TAG,"calcMomentGood");
        int size = GoodputList.size();
        for (int i = 0; i<size;i++){
            Log.i(TAG,"," + i + "," + GoodputList.get(i));
        }
    }

    public static void calcGoodput(){
        // Transaction Time is:
        Long transTime = (endTime - startTime)/1000;
        double doubleTransTime = transTime.doubleValue();
        Log.i(TAG,"Transaction Time is: " + String.valueOf(doubleTransTime));
        // Bytes rcvd are packetSize_bytes
        // bits per packet rcvd are:
        double bits = packetSize_bytes * 8;
        // Megabits per packet rcvd:
        double KbitsPP = bits / 1024;
        //Log.i(TAG,"MbitsPP is: " + String.valueOf(MbitsPP));
        // Total Mbits is
        double Mbits = dataPacksRec*KbitsPP;
        // Mbits/sec is:
        double Kbps = Mbits/doubleTransTime;
        goodput = Kbps;
        Log.i(TAG, "Goodput is: " + String.valueOf(goodput) + " Kbps");
    }



    public void printResults(){
        Log.i(TAG,"==================== Print Results ====================");

        Log.i(TAG,"----------");
        calcGoodput();
        Log.i(TAG,"----------");
        Log.i(TAG, "Interests sent: " + numOfInterestsToSend);
        Log.i(TAG,"----------");
        Log.i(TAG, "Data Packs Received: " + String.valueOf(dataPacksRec));
        Log.i(TAG,"----------");
        totalData_bytes = dataPacksRec*packetSize_bytes;
        Log.i(TAG, "Total Data (bytes): " + String.valueOf(totalData_bytes));
        Log.i(TAG,"----------");
        calcMomentGood();
        Log.i(TAG,"----------");
        calcDelay();
        Log.i(TAG,"----------");
    }
@Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        if (currentProt.equals("dtn")) {
            interestLifetime_ms = DTNinterestLifetime_ms;
        } else if(currentProt.equals("tcp")){
            interestLifetime_ms = TCPinterestLifetime_ms;
        } else if(currentProt.equals("udp")){
            interestLifetime_ms = UDPinterestLifetime_ms;
        }
```

```
        setFace(new Face ("localhost"));


        final Button goBut = (Button) findViewById(R.id.button);
        goBut.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v){

                interestOffset = 0;
                InterestSenderThread interestSenderThread = new InterestSenderThread();
                startTime = System.currentTimeMillis();
                interestSenderThread.start();

                GoodputVsTimeThread goodputVsTimeThread = new GoodputVsTimeThread();
                goodputVsTimeThread.start();
            }
        });

        final Button stopBut = (Button) findViewById(R.id.button2);
        stopBut.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v){
                Log.i(TAG,"Stop Experiment");
                stopSendingInterests = true;
            }
        });

        final Button printServerBut = (Button) findViewById(R.id.button3);
        printServerBut.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v){
                Log.i(TAG,"PRINT RESULTS");
                printResults();

            }
        });

        final Button printClientBut = (Button) findViewById(R.id.button4);
        printClientBut.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v){
                Log.i(TAG,"PRINT RESULTS");
                updServerGui();

            }
        });

        InterRecView = (TextView) findViewById(R.id.textViewInter);



        // Start Interest Receiver
        Log.i(TAG, "start interestReceiver");
        InterestReceiverThread interestReceiver = new InterestReceiverThread();
        interestReceiver.start();
    }


}
```

InterestReceiverThread.java

```
package com.duth.konspras.thesisexperiments;

import android.app.NativeActivity;
import android.os.AsyncTask;
import android.util.Log;
import net.named_data.jndn.Data;
import net.named_data.jndn.Face;
import net.named_data.jndn.Interest;
import net.named_data.jndn.InterestFilter;
import net.named_data.jndn.Name;
import net.named_data.jndn.OnData;
import net.named_data.jndn.OnInterestCallback;
import net.named_data.jndn.OnRegisterFailed;
import net.named_data.jndn.OnTimeout;
import net.named_data.jndn.security.KeyChain;
import net.named_data.jndn.security.identity.IdentityManager;
import net.named_data.jndn.security.identity.MemoryIdentityStorage;
import net.named_data.jndn.security.identity.MemoryPrivateKeyStorage;
import net.named_data.jndn.security.KeyChain;
import java.lang.Thread;

public class InterestReceiverThread extends Thread {

    private String TAG = "Logo_InterestReceiverTh";
    private boolean m_shouldStop = false;

    @Override
    public void run() {
        Name name = new Name("/" + MainActivity.currentProt +"/info/");
        Face face = MainActivity.getFace();
        try {
            Log.i(TAG,"try to setCommandSigningInfo");
            MemoryIdentityStorage identityStorage = new MemoryIdentityStorage();
            MemoryPrivateKeyStorage privateKeyStorage = new MemoryPrivateKeyStorage();
            IdentityManager identityManager = new IdentityManager(identityStorage,
privateKeyStorage);
            KeyChain keyChain = new KeyChain(identityManager);
            Log.i(TAG,"-try to setCommandSigningInfo");
            try {
                keyChain.getDefaultCertificateName();
            } catch (net.named_data.jndn.security.SecurityException e) {
                keyChain.createIdentity(new Name("/ThesisExp"));
                keyChain.getIdentityManager().setDefaultIdentity(new Name("/ThesisExp"));
            }
            Log.i(TAG,"--try to setCommandSigningInfo");
            face.setCommandSigningInfo(keyChain, keyChain.getDefaultCertificateName());
            Log.i(TAG,"try to register prefix");
```

```
            face.registerPrefix(name,
                    new OnInterestCallback() {
                        @Override
                        public void onInterest(Name prefix, Interest interest, Face face, long
var, InterestFilter interestFilter) {
                            Log.i(TAG, "onInterest");
                            //m_shouldStop = true;
                            DataSenderThread dataSender = new DataSenderThread(prefix,
interest);
                            dataSender.start();
                            //InterestReceiver thread = new InterestReceiver();
                            //thread.start();
                            MainActivity.interestPacksRec ++;

                        }

                    },
                    new OnRegisterFailed() {
                        @Override
                        public void onRegisterFailed(Name prefix) {
                            Log.i(TAG, "onRegisterFailed");
                            //m_shouldStop = true;
                        }

                    }
            );

            while (!m_shouldStop) {
                //Log.i(TAG, "sleep");
                face.processEvents();
                Thread.sleep(MainActivity.interestReceiverThreadPeriod);
            }
        } catch (Exception e) {
            Log.e(TAG, "exception sending interest: " + e.getMessage());
        }

    }
}
```

DataSenderThread.java

```
package com.duth.konspras.thesisexperiments;
import android.app.NativeActivity;
import android.os.AsyncTask;
import android.util.Log;
import net.named_data.jndn.Data;
import net.named_data.jndn.Face;
import net.named_data.jndn.Interest;
import net.named_data.jndn.InterestFilter;
import net.named_data.jndn.MetaInfo;
import net.named_data.jndn.Name;
import net.named_data.jndn.OnData;
import net.named_data.jndn.OnInterestCallback;
import net.named_data.jndn.OnRegisterFailed;
import net.named_data.jndn.OnTimeout;
import net.named_data.jndn.security.KeyChain;
import net.named_data.jndn.security.identity.IdentityManager;
import net.named_data.jndn.security.identity.MemoryIdentityStorage;
import net.named_data.jndn.security.identity.MemoryPrivateKeyStorage;
import net.named_data.jndn.util.Blob;
import net.named_data.jndn.util.SignedBlob;
import java.util.Arrays;
import java.util.Random;
import java.io.IOException;
import java.lang.Thread;

public class DataSenderThread extends Thread {
    private Name prefix;
    private Interest interest;
    private String TAG = "Logo_DataSenderThread";

    public DataSenderThread(Name prefix, Interest interest){
        this.prefix = prefix;
        this.interest = interest;
    }

    @Override
    public void run(){
        byte[] tempDatabyte = infoGenerator();
        String temp = interest.getName().toString();// + "/"+ dataString;
        Log.i(TAG, "Sending Data for interest name : " + temp);
        Name dataName = new Name(temp);
        Blob dataBlob = new Blob(tempDatabyte,true);
        MetaInfo metaInfo = new MetaInfo();
        metaInfo.setFreshnessPeriod(MainActivity.freshnessPeriod_ms);
        Data data = new Data();
        data.setName(dataName);
        data.setContent(dataBlob);
        data.setMetaInfo(metaInfo);
        try {
            MainActivity.getFace().putData(data);
        } catch (IOException e){
            Log.i(TAG, "IOException");
        }
    }
```

```
private byte[] infoGenerator(){

        //long temp1 = System.currentTimeMillis();
        Random random = new Random();
        String numbers = null;
        byte[] bytes = new byte[MainActivity.packetSize_bytes];

        random.nextBytes(bytes);

        //numbers = Arrays.toString(bytes);
        long temp2 = System.currentTimeMillis();
        //Log.i(TAG,"RANDOM NUM GEN TOOK: " + String.valueOf(temp2 - temp1) + "ms");
        return bytes;
    }
}
```

InterestSenderThread.java

```
package com.duth.konspras.thesisexperiments;
import android.app.IntentService;
import android.content.Intent;
import android.os.Bundle;
import android.os.Handler;
import android.os.Message;
import android.util.Log;
import net.named_data.jndn.Data;
import net.named_data.jndn.Face;
import net.named_data.jndn.Interest;
import net.named_data.jndn.InterestFilter;
import net.named_data.jndn.Name;
import net.named_data.jndn.OnData;
import net.named_data.jndn.OnTimeout;
import net.named_data.jndn.util.Blob;
import java.lang.Thread;
import java.nio.ByteBuffer;

public class InterestSenderThread extends Thread {
    private final String TAG = "Logo_InterestSenderThrd";
    private boolean m_shouldStop = false;
    private volatile boolean stop = false;
    private long [] interestSendTime;
    private long [] dataReceiveTime;

    @Override
    public void run(){
        int numOfInterestsToSend = MainActivity.numOfInterestsToSend;
        int sendInterestPeriod_ms = MainActivity.sendInterestPeriod_ms;
        interestSendTime = new long[MainActivity.numOfInterestsToSend];
        dataReceiveTime = new long[MainActivity.numOfInterestsToSend];
        if (!MainActivity.isBackToBack) {
            while (MainActivity.getOffset() < numOfInterestsToSend &
!MainActivity.getStopSendingInterests()) {
                try {
                    Log.i(TAG, "Thread try to send interest " +
String.valueOf(MainActivity.getOffset()));
                    sendInterest();
                    MainActivity.incrOffset();
                    sleep(sendInterestPeriod_ms);
                } catch (Exception e) {
                    Log.i(TAG, "Exception while sending interest");
                }
            }
        } else {
            sendInterest();
            MainActivity.incrOffset();
        }
        MainActivity.experimentStopped();
    }
    public void sendInterest(){
        try {
            Face face = MainActivity.getFace();
            int interestNum = MainActivity.getOffset();
            Name name = new Name("/" + MainActivity.currentProt +"/info/" +
String.valueOf(interestNum));
```

```
Interest interest = new Interest(name);
            interest.setInterestLifetimeMilliseconds(MainActivity.interestLifetime_ms);
            interest.setMustBeFresh(true);
            interestSendTime[interestNum] = System.currentTimeMillis();
            face.expressInterest(interest,
                    new OnData() {
                        @Override
                        public void onData(Interest interest, Data data) {
                            MainActivity.numOfDataPacksORTimeoutsReceived ++;
                            MainActivity.dataPacksRec++;

                            String dataName = data.getName().toString();
                            Log.i(TAG, "Received data name is: " + dataName);

                            String[] parts = dataName.split("/");
                            int currentDataIndex = Integer.parseInt(parts[parts.length - 1]);
                            dataReceiveTime[currentDataIndex] = System.currentTimeMillis();

                            Blob dataBlob = data.getContent();

                            ByteBuffer bb = dataBlob.buf();
                            byte[] bytes = new byte[bb.remaining()];
                            bb.get(bytes);

                            if (MainActivity.numOfDataPacksORTimeoutsReceived ==
MainActivity.numOfInterestsToSend){
                                MainActivity.dataReceiveTime = dataReceiveTime;
                                MainActivity.interestSendTime = interestSendTime;
                                MainActivity.experimentEnded();
                            } else {
                                if (MainActivity.isBackToBack){
                                    sendInterest();
                                    MainActivity.incrOffset();
                                }
                            }
                        }
                    },
                    new OnTimeout() {
                        @Override
                        public void onTimeout(Interest interest) {
                            Log.i(TAG,"onTimeout");
                            MainActivity.numOfDataPacksORTimeoutsReceived ++;
                            if (MainActivity.numOfDataPacksORTimeoutsReceived ==
MainActivity.numOfInterestsToSend){

                                MainActivity.dataReceiveTime = dataReceiveTime;
                                MainActivity.interestSendTime = interestSendTime;
                                MainActivity.experimentEnded();
                            } else {
                                if (MainActivity.isBackToBack){
                                    sendInterest();
                                    MainActivity.incrOffset();
                                }
                            }
                        }
                    });
        } catch (Exception e) {
            Log.e(TAG, "exception sending interest: " + e.getMessage());
        }
    }
}
```

GoodputVsTimeThread.java

```
package com.duth.konspras.thesisexperiments;

import android.app.IntentService;
import android.content.Intent;
import android.os.Bundle;
import android.os.Handler;
import android.os.Message;
import android.util.Log;
import java.util.LinkedList;

import net.named_data.jndn.Data;
import net.named_data.jndn.Face;
import net.named_data.jndn.Interest;
import net.named_data.jndn.InterestFilter;
import net.named_data.jndn.Name;
import net.named_data.jndn.OnData;
import net.named_data.jndn.OnTimeout;
import net.named_data.jndn.util.Blob;
import java.lang.Thread;
import java.nio.ByteBuffer;

public class GoodputVsTimeThread extends Thread {

    private final String TAG = "Logo_GoodputVsTimeThd";
    private long prevPackNum = 0;


    @Override
    public void run(){

        MainActivity.GoodputList = new LinkedList();

        while(!MainActivity.stopGoodputCalc){

            try {
                long diff = MainActivity.dataPacksRec - prevPackNum;
                prevPackNum = MainActivity.dataPacksRec;
                double Kbits = (double)MainActivity.packetSize_bytes * 8 /1024;
                double totalKbits = Kbits * diff;

                double totalKbitsPerSec = totalKbits/3;

                MainActivity.GoodputList.add(totalKbitsPerSec);


                sleep(3000);
            } catch (Exception e){
                Log.i(TAG, "Couldn't sleep");
            }
        }

    }
}
```