
# Εθνικο Μετσοβιο Πολυτεχνειο

Σχολη Ηλεκτρολογων Μηχανικων και Μηχανικων Υπολογιστων

Τομεας Τεχνολογιας Πληροφορικης και Υπολογιστων


# Επέκταση βιβλιοθήκης ασύγχρονης ανταλλαγής μηνυμάτων με χρήση δικτύου διασύνδεσης άμεσης πρόσβασης σε απομακρυσμένη μνήμη.

## Διπλωματικη Εργασια

του


**Κωνσταντίνου Σ. Αλεξόπουλου**

**Επιβλέπων:** Γεώργιος Γκούμας
Αναπληρωτής Καθηγητής Ε.Μ.Π.

Εργαστηριο Υπολογιστικων Συστηματων



Αθήνα, Οκτώβριος 2017

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών
Εργαστήριο Υπολογιστικών Συστημάτων

# Επέκταση βιβλιοθήκης ασύγχρονης ανταλλαγής μηνυμάτων με χρήση δικτύου διασύνδεσης άμεσης πρόσβασης σε απομακρυσμένη μνήμη.

## Διπλωματικη Εργασια

του

**Κωνσταντίνου Σ. Αλεξόπουλου**

**Επιβλέπων:** Γεώργιος Γκούμας
Αναπληρωτής Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 30η Οκτωβρίου 2017.

| *(Υπογραφή)* | *(Υπογραφή)* | *(Υπογραφή)* |
|---|---|---|
| ..... | ..... | ..... |
| Γεώργιος Γκούμας | Νεκτάριος Κοζύρης | Δημήτριος Σούντρης |
| Αναπληρωτής Καθηγητής Ε.Μ.Π. | Καθηγητής Ε.Μ.Π. | Επίκουρος Καθηγητής Ε.Μ.Π. |

Αθήνα, Οκτώβριος 2017

*(Υπογραφή)*

.....

**Κωνσταντίνος Σ. Αλεξόπουλος**

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

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών
Εργαστήριο Υπολογιστικών Συστημάτων

# Ευχαριστίες

Θα ήθελα να ευχαριστήσω τον Αναπληρωτή Καθηγητή Γεώργιο Γκούμα, για την εμπιστοσύνη που μου έδειξε, και την ευκαιρία που μου έδωσε να διεκπεραιώσω την διπλωματική μου εργασία ως μέρος του εργαστηρίου Υπολογιστικών Συστημάτων του ΕΜΠ. Θα ήθελα επίσης να ευχαριστήσω τον Υποψήφιο Διδάκτορα Στέφανο Γεράγγελο, για την κρίσιμη καθοδήγησή του κατά τις τελευταίες φάσεις της εργασίας μου.

Θέλω να ευχαριστήσω τους πρώην συναδέλφους μου στο CERN, τον Olof Bärring, τη Sima Baymani, τον Alexandru Grigore και τον Αράμ Σαντογίδη για την προθυμία τους να μου μεταφέρουν τη σοφία και την εμπειρία τους, βοηθώντας με να γίνω καλύτερος επαγγελματίας.

Ευχαριστώ την οικογένειά μου για την υποστήριξή της κατά τις σπουδές μου, τον πατέρα μου, Σπύρο, που μου έμαθε να είμαι προνοητικός, και τη μητέρα μου, Σάντυ, που μου δίδαξε την αξία της επιμονής της ελαστικότητας και της επιμονής, αλλά πρωτίστως ευχαριστώ τη γιαγιά μου, Βάνα, για την διαρκή και αμέριστη υποστήριξη της, στην οποία θα είμαι αιώνια ευγνώμων που με βοήθησε να γίνω ο άνθρωπος που είμαι σήμερα.

Επιπλέον, θα ήθελα να ευχαριστήσω τους συμφοιτητές μου, που μου έμαθαν την αξία της συνεργασίας και προσέφεραν χαρά στην ακαδημαϊκή μου πορεία.

Τέλος, θέλω να ευχαριστήσω την Ελένη, η οποία στάθηκε υπομονετικά δίπλα μου κατά τη δύσκολη διαδικασία της σύνταξης αυτής της διπλωματικής, προσφέροντας συνεχώς την εμψύχωσή της.

# Περίληψη

Καθώς η υπολογιστική ισχύς και η επίδοση εισόδου/εξόδου αυξάνεται με επιθετικούς ρυθμούς, ένας αριθμός RDMA-enabled δικτύων διασύνδεσης έχουν αρχίσει να κάνουν την εμφάνισή τους, υποσχόμενα χαμηλό latency και υψηλό throughput. Οι τεχνολογίες RDMA, υποστηρίζουν zero-copy λειτουργίες και την εκφόρτωση του επεξεργαστή, εκτελώντας απευθείας εγγραφές σε απομακρυσμένη μνήμη. Ωστόσο, η πλειοψηφία των κατανεμημένων, δικτυακά εντατικών εφαρμογών σήμερα είναι σχεδιασμένη γύρω από τη χρήση socket, διεπαφών οι οποίες είναι εγγενώς ασυμβίβαστες με την προσέγγιση της απομακρυσμένης μνήμης.

Ο σκοπός αυτής της διπλωματικής είναι να απευθυνθεί σε αυτή την διαφορά, ανάμεσα στα εδραιωμένα και αναδυόμενα πρότυπα επικοινωνίας, προσφέροντας μια διεπαφή για χρήση επικοινωνίας συνεκτικής μνήμης, στα πλαίσια των υπολογιστικών υποδομών υψηλών επιδόσεων. Αυτό επιτυγχάνεται επεκτείνοντας το ZeroMQ, μια βιβλιοθήκη ασύγχρονης ανταλλαγής μηνυμάτων, ώστε να κάνει χρήση του επιπέδου μεταφοράς RapidIO, μιας υψηλής επίδοσης, RDMA-enabled τεχνολογίας διασύνδεσης μεταγωγής πακέτων. Το ZeroMQ αποτελεί καλό υποψήφιο για τους σκοπούς αυτής της διπλωματικής, καθώς το επίπεδο μεταφοράς του είναι αφηρημένο και έχει ήδη επεκταθεί για έναν αριθμό διαφορετικών πρωτοκόλλων. Επιπροσθέτως, επιτρέπει την εύκολη χρήση του, ανεξαρτήτως του επιπέδου μεταφοράς, επιτρέποντας με αυτό τον τρόπο την άμεση εφαρμογή μιας επέκτασης.

Μέσα από αυτή τη δουλειά, μελετάται η διαδικασία της επέκτασης μιας κατανεμημένης εφαρμογής, η οποία βασίζεται θεμελιωδώς σε sockets, αξιολογώντας ταυτόχρονα τις προκλήσεις που συνοδεύουν την προαναφερθείσα διαφορά στα πρότυπα επικοινωνίας. Τέλος, παρατίθενται συμπεράσματα σχετικά με την διαφορά επιδόσεων, όπως επίσης και περιορισμοί στην διαδικα-

σία ανάπτυξης, ως αποτέλεσμά της χρήσης μιας νέας τεχνολογίας, στην περίπτωσή μας, ενός δικτύου διασύνδεσης άμεσης πρόσβασης σε απομακρυσμένη μνήμη.


## Λέξεις Κλειδιά

Δίκτυα Διασύνδεσης, Υπολογιστικές Υποδομές Υψηλών Επιδόσεων, Κατανεμημένα Συστήματα, Απευθείας Πρόσβαση σε Απομακρυσμένη Μνήμη, ZeroMQ, RapidIO

# Περιεχόμενα

# Κατάλογος σχημάτων

# Κατάλογος πινάκων

# Κεφάλαιο 1

# Εισαγωγή

Όπως προβλέπει ο νόμος του Moore, οι υπολογιστικές ικανότητες των σύγχρονων υπολογιστών συνεχίζουν να βελτιώνονται με εκθετικούς ρυθμούς Ωστόσο, μία πτυχή του νόμου του Amdahl δηλώνει πως η απόδοσή ενός συστήματος μπορεί να εκτιμηθεί μόνο αναλογικά με την ισορροπία ανάμεσα σε υπολογιστική ισχύ του επεξεργαστή, την ταχύτητα της μνήμης και την επίδοσης εισόδου/εξόδου. Τα δίκτυα διασύνδεσης χαρακτηρίζονται από σχεδιαστικούς περιορισμούς, δυσκολότερους από αυτούς που συναντώνται στο σχεδιασμό ημιαγωγών. Συνεπώς, εμφανίζεται ένα σημείο συμφόρησης. Ταυτόχρονα, ο αριθμός των στοιχείων που χρειάζεται να δικτυωθούν αυξάνεται ταχέως. Οι παρούσες αρχιτεκτονικές Ε/Ε δεν είναι κατάλληλα σχεδιασμένες για να υποστηρίξουν αυτό το φαινόμενο.

Σε μια προσπάθεια να αντιμετωπιστεί το πρόβλημα, οι σχεδιαστές συστημάτων έχουν αρχίσει τελευταία να εγκαταλείπουν το παραδοσιακό μοντέλο του κοινού διαύλου επικοινωνίας, και στρέφονται στα δίκτυα μεταγωγής πακέτων, τα οποία ευνοούν την ταχύτητα και την κλιμάκωση. [1]

Με τις τελευταίες εξελίξεις στους τομείς του Cloud και του Internet, τα data centers είναι στο προσκήνιο. Ένας αυξάνων αριθμός υπηρεσιών και εφαρμογών εκτελούνται πλέον εντός των data centers. Αυτό έχει οδηγήσει σε μία νέα ανάγκη για κλιμάκωση, έχοντας ως αποτέλεσμα τον καθορισμό νέων απαιτήσεων επίδοσης στο πεδίο του HPC. Επιπλέον, τα data centers σήμερα συχνά περιλαμβάνουν FPGAs, GPUs ή συστήματα αποθήκευσης, τα οποία συχνά χρησιμοποιούν αποκλειστικά και ιδιόκτητα πρωτόκολλα διασύνδεσης.

[3]

Έχουν προταθεί διάφορες αρχιτεκτονικές διασύνδεσης, οι οποίες επιδιώκουν την γεφύρωση των επικοινωνιών εντός και εκτός ενός κόμβου και την υποστήριξη ετερογένειας, με την έννοια της διασύνδεσης διαφορετικών τύπων συσκευών. Αυτές οι αρχιτεκτονικές μεταγωγής συνήθως υποστηρίζουν μηχανισμούς κοινής μνήμης ανάμεσα στους κόμβους, οι οποίοι απαιτούν την αντίστοιχη προσπάθεια ανάπτυξης. Ένα επιπλέον πρόβλημα εμφανίζεται και λόγω του κόστους αναβάθμισης, καθώς νέο υλικό είναι απαραίτητο να εγκατασταθεί σε όλη την έκταση του δικτύου.

## 1.1 Σκοπός

Ο σκοπός αυτής της εργασίας είναι η παροχή μιας διεπαφής επικοινωνίας για ένα κατανεμημένο περιβάλλον πολλαπλών κόμβων, το οποίο λειτουργεί με χρήση ενός RDMA-enabled επιπέδου μεταφοράς, ανταποκρινόμενο στις σύγχρονες τεχνολογίες διασύνδεσης. Με τον τρόπο αυτό, η υλοποίηση της διεπαφής απορροφά την προσπάθεια που απαιτείται για την επέκταση, και επιτρέπει την γρήγορη και εύκολη χρήση της τεχνολογίας σε ήδη υπάρχοντα συστήματα. Επιπρόσθετα, οι δυνατότητες ταχύτητας και κλιμάκωσης που συνοδεύουν την χρήση ενός RDMA-enabled δικτύου διασύνδεσης δύναται να χρησιμοποιηθούν σε ένα HPC περιβάλλον.

Οι σύγχρονες εφαρμογές δικτύωσης είναι σχεδιασμένες γύρω από sockets. Τα sockets παρέχουν μια βολική και αξιόπιστη διεπαφή για επικοινωνίες δικτύου. Ωστόσο, αυτό κοστίζει σε επίδοση, καθώς απαιτείται η προσπέλαση της στοίβας του δικτύου και η κατανάλωση κύκλων της CPU, τόσο για τον τοπικό, όσο και για τον απομακρυσμένο κόμβο. Αυτό έχει αρνητική επίδοση σε latency και δυνατότητες κλιμάκωσης.

Αυτές οι πτυχές παίζουν σημαντικό ρόλο στον τομέα του HPC, όπου τα επίπεδα της επίδοσης είναι κρίσιμα. Εδώ, μηχανισμοί χρήσης κοινής μνήμης οδηγούν σε χαμηλότερη καθυστέρηση για επικοινωνία ανάμεσα σε κόμβους, αφού δεν επιβαρύνουν τον επεξεργαστή. Τα RDMA-enabled δίκτυα διασύνδεσης συμπεριφέρονται επίσης καλύτερα σε συστήματα με πολλούς κόμβους, όπου λαμβάνουν χώρα πολλές συναλλαγές, και το χαμηλότερα υπεράνω κό-

στη ανά συναλλαγή έχουν σημαντική επίπτωση στην ταχύτητα. Ωστόσο, οι μηχανισμοί κοινής μνήμης απαιτούν χρόνο και προσπάθεια στο κομμάτι της ανάπτυξης και συντήρησης, καθώς απαιτείται ειδική εγκατάσταση και η διαχείριση δραστηριοτήτων προσπέλασης μνήμης. Συνεπώς, συνηθίζεται να χρησιμοποιούνται μόνο όταν οι απαιτήσεις για επίδοση είναι αυστηρές.

Ως αποτέλεσμα, παρατηρείται μια εγγενής ασυμφωνία ανάμεσα στους μηχανισμούς κοινής μνήμης και τα sockets. Στα πλαίσια αυτής της δουλειάς, θα διερευνηθεί μια λύση για γεφύρωση των δύο.

Για να απευθυνθούμε σε αυτά τα σημεία, επεκτείναμε το ZeroMQ [4], μία κατανεμημένη βιβλιοθήκη ανταλλαγής μηνυμάτων, προκειμένου να χρησιμοποιεί το RapidIO [5], ως επίπεδο μεταφοράς. Το ZeroMQ προσφέρει μια διεπαφή μηνυμάτων που καθιστά απλή την επιλογή του επιπέδου μεταφοράς από τον χρήστη. Επιπλέον, είναι βελτιστοποιημένο να ξεπερνάει τα συνήθη υπεράνω κόστη που συναντούνται σε άλλα παρόμοια συστήματα, διατηρώντας την επίδοση του συστήματος σε υψηλά επίπεδα. Τέλος, η ανάπτυξή του είναι στενά συνδεδεμένη με την χρήση sockets, γεγονός που μας επιτρέπει να μελετήσουμε ένα μηχανισμό που θα επιτρέψει τον συμβιβασμό των δύο προσεγγίσεων επικοινωνίας.

### 1.1.1 Συνεισφορές

Οι συνεισφορές στα πλαίσια αυτής της διπλωματικής εργασίας μπορούν να συνοψιστούν ως εξής:

- Η βιβλιοθήκη ZeroMQ επεκτάθηκε ώστε να χρησιμοποιεί το RapidIO ως επίπεδο μεταφοράς, το οποίο είναι ένα RDMA-enabled δίκτυο διασύνδεσης.
- Το πρωτόκολλο RapidIO βρήκε εφαρμογή σε ένα περιβάλλον HPC, εκτός του συνηθισμένου πεδίου του, τα ενσωματωμένα συστήματα.
- Έγινε μια πρακτική διερεύνηση της εφαρμογής των μηχανισμών της απευθείας προσπέλασης απομακρυσμένης μνήμης.

## 1.2 Δομή του Κειμένου

Η εργασία αυτή ακολουθεί την εξής δομή:

Στο Κεφάλαιο 2 ο αναγνώστης εισάγεται στο απαραίτητο θεωρητικό υπόβαθρο, πάνω στο οποίο είναι βασισμένο η εργασία. Περιγράφονται επίσης κύρια σημεία των τεχνολογιών που χρησιμοποιήθηκαν.

Το Κεφάλαιο 3 επικεντρώνεται στον σχεδιασμό της επέκτασης της βιβλιοθήκης και σε λεπτομέρειες της υλοποίησης.

Τα μετροπρογράμματα που χρησιμοποιήθηκαν για την εξαγωγή μετρήσεων και η αξιολόγηση των τελευταίων παρουσιάζεται στο Κεφάλαιο 4.

Παρεμφερείς εργασίας παρουσιάζονται στο Κεφάλαιο 5, πριν το Κεφάλαιο 6, όπου παρατίθενται συμπεράσματα και προτάσεις για μελλοντικές επεκτάσεις.

# Κεφάλαιο 2

# Θεωρητικό Υπόβαθρο

Το κεφάλαιο αυτό έχει σκοπό να εισάγει τον αναγνώστη στους όρους και τις έννοιες, οι οποίες είναι απαραίτητες για να ακολουθήσει την παρουσίαση αυτής της εργασίας. Το κεφάλαιο είναι χωρισμένο σε τρία μέρη: Το πρώτο εισάγει έννοιες σχετικές με τα δίκτυα διασύνδεσης γενικότερα. Το δεύτερο αφορά το RapidIO, το πρωτόκολλο που χρησιμοποιείται στα πλαίσια αυτής της δουλειάς. Το τρίτο αναφέρεται στο ZeroMQ, το πλαίσιο ανταλλαγής μηνυμάτων που επεκτάθηκε.

## 2.1 Δίκτυα Διασύνδεσης

Ο όρος διασύνδεση αναφέρεται στην δικτυακή σύνδεση δύο ή περισσοτέρων υπολογιστικών στοιχείων. Αυτά μπορεί να είναι οτιδήποτε, από ολοκληρωμένα κυκλώματα, πλακέτες και υπολογιστικά συστήματα, μέχρι δίκτυα ευρείας ζώνης. Τα δίκτυα διασύνδεσης ορίζουν τις δυνατότητες Ε/Ε κάθε στοιχείου, όπως ο επεξεργαστής και η μνήμη. Ωστόσο, η αποδοτικότητα του συστήματος ορίζεται από την ισορροπία ανάμεσα στην επίδοση του επεξεργαστή, την ταχύτητα της μνήμης αλλά και την ταχύτητα των λειτουργιών Ε/Ε, όπως ορίζεται από τον νόμο του Amdahl. Με άλλα λόγια, ακόμα και αν η επίδοση των εν μέρει στοιχείων είναι υψηλή, η επίδοση του συστήματος θα είναι χαμηλή μέχρις ότου το στοιχεία διασυνδεθούν μεταξύ τους αποδοτικά. Ταυτόχρονα, σύμφωνα με τον νόμο του Moore, η ταχύτητα και το μέγεθος των ημιαγωγών συνεχίζουν να βλέπουν πρόοδο, με γοργότερους ρυθμούς συ-

γκριτικά με τους ηλεκτρικούς και μηχανικούς περιορισμούς του σχεδιασμού δικτύων διασύνδεσης. Συνεπώς, προκύπτει μια θεμελιώδης ανισορροπία στην σχεδίαση συστημάτων, κατά την οποία τα συστήματα δεν μπορούν να εκμεταλλευτούν αποδοτικά την ισχύ των στοιχείων τους. Προκειμένου να ξεπεραστούν αυτά τα προβλήματα, έχει παρουσιαστεί ένας αριθμός δικτύων διασύνδεσης υψηλών επιδόσεων.

Το πιο διαδεδομένο δίκτυο διασύνδεσης επιπέδου συστήματος είναι το PCI (Peripheral Component Interconnect). Το PCI υποστηρίζει μια αρχιτεκτονική διαύλου επικοινωνίας, σύμφωνα με την οποία παίρνονται οι αποφάσεις που αφορούν τα ηλεκτρικά και τα μηχανικά μέρη των υπολογιστών τα τελευταία περίπου 30 χρόνια. Λόγω της επιδράσεως αυτής, το PCI έχει αποκτήσει ιδιαίτερη αδράνεια στη βιομηχανία, γεγονός που καθιστά την εφαρμογή νέων τεχνολογιών δύσκολη.

Εν τούτοις, ένα δίκτυο διασύνδεσης δεν στοχεύει μόνο το επίπεδο του συστήματος. Συνδέσεις "έξω από το κουτί" θα μπορούσαν επίσης να βελτιωθούν, ως μέρος ενός καθολικού, ομογενούς δικτύου. Ένας μεγάλος αριθμός σύγχρονων εφαρμογών εκτελείται σε συστοιχίες συστημάτων υψηλής επίδοσης, στις οποίες το φορτίο κατανέμεται στους διάφορες εξυπηρετητές του συστήματος, που συνήθως βρίσκονται σε κάποιο data center. Η επίδοση αυτού του συστήματος θα μπορούσε να βελτιωθεί σημαντικά, στην περίπτωση χρήσης ενός κοινού δικτύου διασύνδεσης, από το επίπεδο του PCB (Printed Circuit Board), μέχρι το επίπεδο δικτύωσης των υπολογιστικών κόμβων. Η διατήρηση ενός και μόνο πρωτοκόλλου κατά μήκος των κόμβων έχει θετικό αποτέλεσμα στην κλιμάκωση, υποστηρίζοντας χαμηλότερα υπεράνω έξοδα κατά την δρομολόγηση της κίνησης. Συμπερασματικά, ένα δίκτυο διασύνδεσης υψηλής επίδοσης θα μπορούσε να ωφελήσει όλα τα επίπεδα του υλικού, που συναντώνται σήμερα στο ραγδαίως εξελισσόμενο πεδίο του HPC.

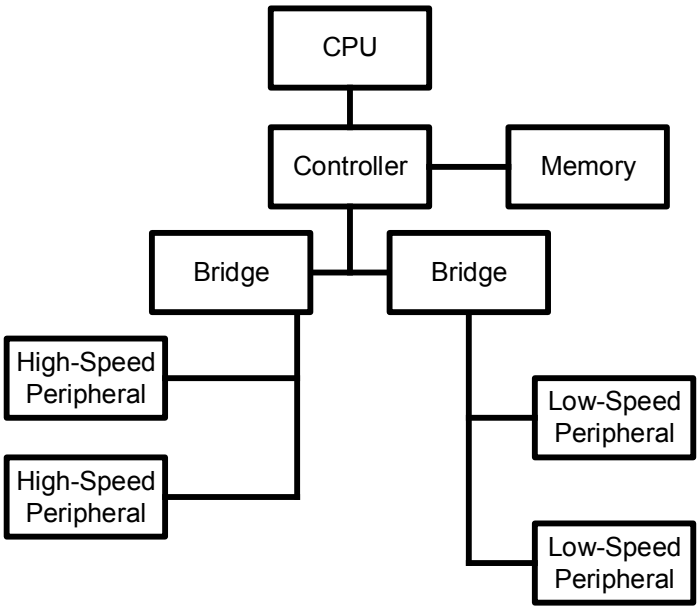
### 2.1.1 Αρχιτεκτονικές Διαύλων Επικοινωνίας και Δίκτυα Μεταγωγής

Η πιο κοινή αρχιτεκτονική δικτύων διασύνδεσης σήμερα είναι ο κοινός δίαυλος επικοινωνίας (shared bus). Αν και τα shared buses υπαγορεύουν αρκετούς περιορισμούς στην σύγχρονη σχεδίαση συστημάτων, παραμένουν δημο-

φιλή λόγω της ιστορίας τους. Οι μοντέρνες συστοιχίες υπολογιστών απαιτούν throughput αλλά και αξιοπιστία, που το PCI αδυνατεί να υποστηρίξει. Στο πρόβλημα αυτό απευθύνονται τα νέα δίκτυα διασύνδεσης, τα οποία βασίζονται σε μια αρχιτεκτονική μεταγωγής πακέτων, που επιτρέπει γρήγορες και κλιμακούμενες φυσικές τοπολογίες δικτύων.

Σε ένα σύστημα που χρησιμοποιεί μια αρχιτεκτονική διαύλων, όλα τα στοιχεία είναι συνδεδεμένα στον ίδιο δίαυλο. Επομένως, όλες οι συναλλαγές γίνονται σε κοινό μέσο επικοινωνίας. Ως άμεση συνέπεια, η αύξηση του αριθμού των συνδέσεων έχει ως αποτέλεσμα την μείωση του διαθέσιμου εύρους ζώνης ανά στοιχείο. Για να λυθεί αυτό, έγινε χρήση μιας ιεραρχίας διαύλων, ομαδοποιώντας τα γρήγορα στοιχεία με άλλα γρήγορα, και τα αργά με άλλα αργά, όπως φαίνεται στην εικόνα 2.1. Παρά ταύτα, μία πηγή συμφόρησης εξακολουθεί να υπάρχει, καθώς τα διαφορετικά buses θα πρέπει τελικώς να συνδεθούν σε ένα κεντρικό bus. Τεχνικές σχεδιασμούς σαν κι αυτήν επιβάλλουν ηλεκτρικούς και μηχανικούς περιορισμούς, οι οποίοι συμβάλλουν στην συμφόρηση. Κατ' αυτό τον τρόπο ο αριθμός των πιθανών συνδέσεων διατηρείται χαμηλός και αυξάνεται η πιθανότητα σφαλμάτων. Επί πλέον, η χρήση διαύλων απαγορεύει την επέκταση με εξωτερικά στοιχεία, απαιτώντας τη χρήση ενός άλλου δικτύου διασύνδεσης. [6, 7]

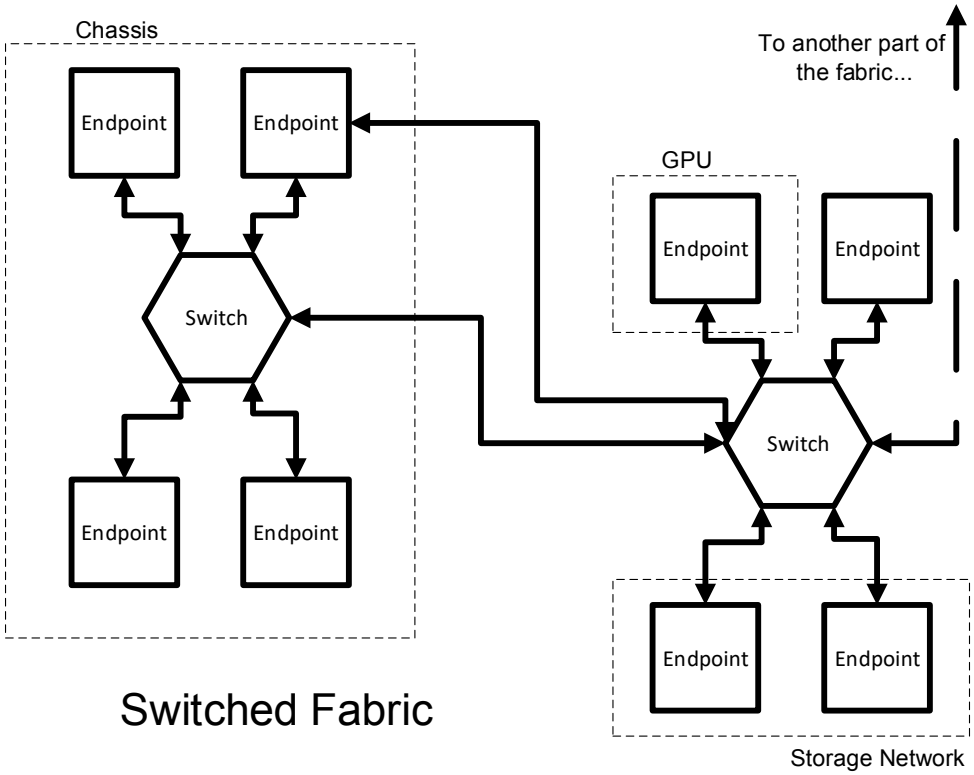
Οι αρχιτεκτονικές δικτύων μεταγωγής αποτελούν σχεδιασμού που υποστηρίζουν την επικοινωνία από σημείο-σε-σημείο. Κάθε στοιχείο έχει μια σύνδεση με ακριβώς ένα και κανένα άλλο στοιχείο στο άλλο άκρο. Αυτή η επιλογή, εγγυάται καλύτερη επίδοση σε σχέση με έναν δίαυλο, ενώ ταυτόχρονα διατηρεί την πολυπλοκότητα χαμηλή, μειώνοντας την πιθανότητα να προκύψουν σφάλματα. Ένα άλλο πλεονέκτημα των δικτύων μεταγωγής είναι η κλιμάκωση. Εδώ, όταν ένα στοιχείο δικτύου εισάγεται, δεδομένου πως οι μεταγωγείς είναι αρκετοί και αποδοτικοί, το εύρος ζώνης δεν περιορίζεται. Το μοντέλο αυτό υποστηρίζει επίσης πλεονάζουσες διαδρομές ανάμεσα σε στοιχεία, γεγονός που ενισχύει την αξιοπιστία του συστήματος.




Σχήμα 2.1: Typical shared bus architecture

### 2.1.2 Δίκτυα Διασύνδεσης σε Υπολογιστικές Υποδομές Υψηλών Επιδόσεων

Η υπολογιστική υποδομή υψηλών επιδόσεων (High Performance Computing, HPC), είναι ένας όρος που χρησιμοποιείται για να περιγράψει τον τομέα της υπολογιστικής επιστήμης που είναι υπεύθυνο για υπολογιστικά εντατικές εφαρμογές. Τέτοιες εφαρμογές προέρχονται συχνά από τα πεδία της φυσικής, των προβλέψεων, και μπορεί να αποτελούν προσομοιώσεις και μοντελοποιήσεις, μεταξύ άλλων. Στα πλαίσια του HPC, χρησιμοποιούνται υπερυπολογιστές, εν αντιθέσει με τους υπολογιστές γενικού σκοπού. Οι πρώτοι υπερυπολογιστές κατασκευάστηκαν την δεκαετία του 1960 και ουσιαστικά ήταν βελτιωμένες εκδόσεις των αντίστοιχων των καταναλωτών. Με τα χρόνια, εμφανίστηκε έντονος παραλληλισμός, με τους υπερυπολογιστές να γίνονται πολυπύρηνα συστήματα. Στη συνέχεια, τα συστήματα αυτά έγιναν μέρη συστοιχιών, αυξάνοντας τον αριθμό των κόμβων και πυρήνων με μεγαλύτερο ρυθμό. Σε εκείνο το σημείο, η επίδοση του δικτύου διασύνδεσης ανάμεσα στους διαφορετικούς κόμβους έγινε κρίσιμη. Στα πλαίσια αυτά, η διασύνδεση δύο κόμβων που βρίσκονται σε διαφορετική πλακέτα δεν χρειάζονταν μόνο




Σχήμα 2.2: Switched Fabric

εύρος ζώνης, αλλά και εξαιρετικά μικρή καθυστέρηση, καθώς οι επικοινωνία μεταξύ τους μπορούσε να αφορά εντολές. Φυσικά σε αυτό το επίπεδο συστήματος, η κλιμάκωση και η διαχείριση σφαλμάτων αποτελούν επίσης σημαντικές απαιτήσεις.

Ως απάντηση σε αυτές τις απαιτήσεις, έχουν κυκλοφορήσει στην αγορά τα τελευταία χρόνια τεχνολογίες διασύνδεσης, από τις οποίες η πιο επικρατούσα είναι το Infiniband. Ταυτόχρονα, άλλες τεχνολογίες που έβρισκαν ήδη εφαρμογή αλλού εξελίσσονται για να προσαρμοστούν στα νέα δεδομένα. Ένα τέτοιο παράδειγμα είναι οι εκδόσεις του Ethernet για ταχύτητες στα 10, 40 και 100Gbps. Ένα ακόμα πρότυπα διασύνδεσης που αφορά αυτό το πεδίο είναι το RapidIO. Ακολουθούν ορισμένες λεπτομέρειες για τα προαναφερθέντα.

### Infiniband

Το Infiniband (IB) είναι μια σειριακή αρχιτεκτονική διασύνδεσης Ε/Ε, βασισμένη σε μεταγωγή πακέτων. Υποστηρίζει υψηλό throughput και χαμηλό latency, με ονομαστικές ταχύτητες από τα 10Gb/s μέχρι τα 56Gb/s. Η κύρια εφαρμογή της είναι η διασύνδεση ανάμεσα σε υπολογιστές. Χάριν στην επίδοσή του συνηθίζεται να χρησιμοποιείται σε περιβάλλοντα HPC. Το Infiniband χρησιμοποιεί απευθείας προσπέλαση απομακρυσμένης μνήμης και μηνύματα για μεταφορές, σε σχέση με το TCP/IP για το Ethernet. Αυτό σημαίνει την πιθανή ανάγκη για ανάπτυξη λογισμικού σε πιθανή χρήση του στα πλαίσια υπάρχοντων λύσεων.

### 10Gb/40Gb/100Gb Ethernet

Τα 10Gb, 40Gb και 100Gb Ethernet, που πιο συχνά συναντώνται ως 10GbE, 40GbE και 100GbE, είναι οι επικαιροποιημένες εκδόσεις του πρωτοκόλλου Ethernet. Κάποιες από αυτές τις εκδόσεις υποστηρίζουν συμβατότητα προς τα πίσω. Με άλλα λόγια, καλώδια και μεταγωγείς που χρησιμοποιούνται για αυτές τις τεχνολογίες, υποστηρίζουν και παλαιότερες εκδόσεις. Κατ' αυτόν τον τρόπο δεν απαιτείται η ανανέωση όλου του συστήματος σε πιθανή αναβάθμιση. Καθώς στις περισσότερες σύγχρονες διατάξεις ήδη χρησιμοποιείται κάποια μορφή Ethernet, αυτό καθιστά την χρήση τους πιο ελκυστική. Φυσικά, ένας ακόμα σημαντικός παράγοντας είναι πως, για την συντριπτική πλειοψηφία, δεν χρειάζεται ανάπτυξη νέου λογισμικού για την υιοθέτηση αυτών των εκδόσεων.

Η επόμενη παράγραφος, παρουσιάζει το RapidIO, ένα δίκτυο διασύνδεσης επιπέδου συστήματος, το οποίο στοχεύει επίσης το πεδίο του Δικτύου Περιοχής Συστήματος (System Area Network (SAN)). Το SAN είναι ένας ακόμα όρος για να περιγράψει ένα δίκτυο που λαμβάνει μέρος στο πλαίσιο ενός data center, σχεδιασμένο για ταχύρυθμες διασυνδέσεις σε περιβάλλοντα συστοιχιών και πολυπύρηνων συστημάτων. Ωστόσο, ο όρος SAN συχνά αναφέρεται και στη διασύνδεση στοιχείων "μέσα στο κουτί".

## 2.2 RapidIO

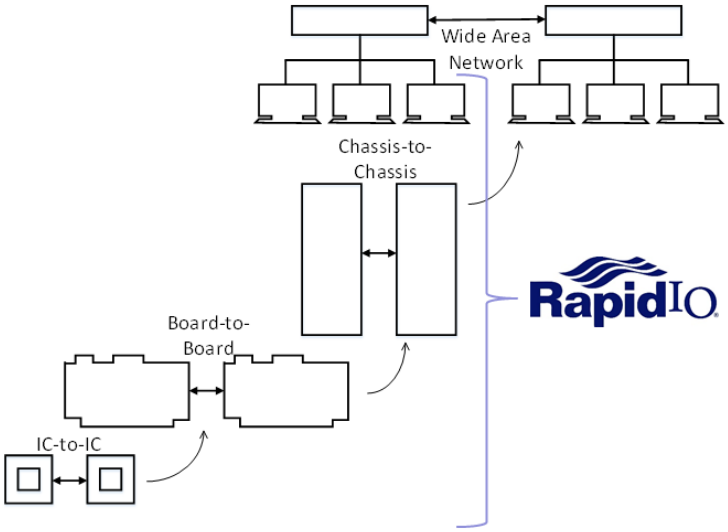
Το RapidIO είναι ένα δίκτυο διασύνδεσης επιπέδου συστήματος. Υποστηρίζει υψηλή επίδοση, χαμηλό αριθμό ακροδεκτών και μεταγωγή πακέτων.

Σε αντίθεση με σχεδιασμούς διαύλων κοινής μνήμης, όπως το PCI, το RapidIO υποστηρίζει μεταγωγή πακέτων, όπως το Infiniband. Τα δίκτυα διασύνδεσης μεταγωγής πακέτων είναι αντίστοιχα των αρχιτεκτονικών δικτύων μεταγωγής. Κάθε τελικό σημείο του συστήματος έχει ακριβώς μία σύνδεση με ένα άλλο στοιχείο ή μεταγωγέα. Οι λειτουργίες των δικτύων διασύνδεσης μεταγωγής συνήθως περιλαμβάνουν ένα μηχανισμό ανταλλαγής μηνυμάτων και αποστολής event, στα πλαίσια του δικτύου. Επιπρόσθετα, τεχνολογίες αυτού του πεδίου προσφέρουν υποστήριξη για Memory-Mapped I/O, ή MMIO, το οποίο θεωρείται και το πιο αποδοτικό μέσο επικοινωνίας.

Το RapidIO είναι ένα ανοιχτό πρότυπο και οι εφαρμογές του συμπεριλαμβάνουν τη διασύνδεση επεξεργαστών, μνήμης, συσκευών MMIO, συσκευών αποθήκευσης και υπολογιστικών συστημάτων. Αρχικά είχε σχεδιαστεί για χρήση ως εμπρόσθιος δίαυλος (FSB), αλλά στην πορεία πήρε την μορφή δικτύου διασύνδεσης επιπέδου συστήματος, και γνώρισε ένα ευρύ φάσμα εφαρμογών σε ενσωματωμένα συστήματα. Την ημερομηνία της συγγραφής, οι περισσότερες συνδέσεις που υποστηρίζουν RapidIO μπορούν να βρεθούν στον τομέα των τηλεπικοινωνιών, όπου η τεχνολογία συναντάται σε κάθε base station για 4G στον κόσμο. Ωστόσο, το πρωτόκολλο είναι ανεξάρτητο από την φυσική του υλοποίηση και συνεπώς η αρχιτεκτονική αποτελεί καλή υποψηφιότητα για την διασύνδεση συστημάτων που βρίσκονται σε διαφορετικά επίπεδα του υλικού. Ουσιαστικά, το RapidIO μπορεί να εφαρμοστεί σε οποιοδήποτε πεδίο, από την επικοινωνία μεταξύ chip στην επικοινωνία συστοιχιών υπολογιστικών συστημάτων σε data centers, όπως φαίνεται στην εικόνα 2.3. [8]

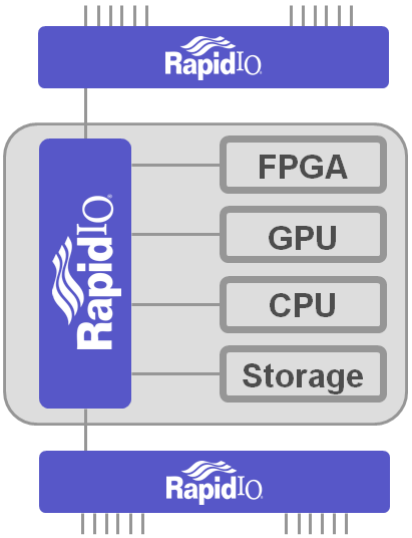
Έτσι, παρουσιάζεται η ευκαιρία της αδιάλειπτης ενοποίησης FPGA, GPU, CPU και αποθηκευτικών συστημάτων, ως μέρος του ίδιου δικτύου, καθιστώντας το RapidIO ιδανική λύση για ετερογενή συστήματα, όπως φαίνεται στην εικόνα 2.4. Το RapidIO αυτή τη στιγμή μπορεί να βρεθεί ως native implementation “on-chip” και σε προσαρμογείς που το γεφυρώνουν με PCIe.

Το τελευταίο specification [9] κυκλοφόρησε τον Ιούνιο του 2016, ορίζοντας τα-




Σχήμα 2.3: Interconnect application domains where RapidIO is applicable

χύτητες μέχρι 25Gbaud ανά γραμμή. Αυτό μεταφράζεται σε ταχύτητα δικτύου RapidIO των 100Gb/s.




Σχήμα 2.4: RapidIO exploitation in heterogeneous systems

### 2.2.1 Πρωτόκολλο

Το RapidIO είναι κυρίως υλοποιημένο στο υλικό. Η διαχείριση σφαλμάτων παίρνει κατά κύριο λόγο μέρος στο φυσικό στρώμα και προσφέρεται hardware termination. Έτσι, ανοίγει ο δρόμος για την επίτευξη πολύ χαμηλών καθυστερήσεων, και πιο σημαντικά, αφαιρεί φόρτο από τον επεξεργαστή. Μεταφέρο-

ντας την επιβάρυνση του πρωτοκόλλου στο υλικό, απελευθερώνονται κύκλοι του επεξεργαστή, μειώνοντας την κατανάλωση και επιτρέποντας την εφαρμογή του σε συστήματα πραγματικού χρόνου (realtime systems), αλλά και την κλιμάκωση του συστήματος. Το πρωτόκολλο υποστηρίζει δρομολόγηση βάσει προορισμού και υποστηρίζει την ανάπτυξη οποιασδήποτε φυσικής τοπολογίας. Σημαντικότερα, τα specifications του πρωτοκόλλου μπορούν να διαμεριστούν, σύμφωνα με τις ανάγκες της εκάστοτε φυσικής υλοποίησης. Κατ' αυτόν τον τρόπο, η πολυπλοκότητα του συστήματος και των στοιχείων μειώνεται, καθώς ταυτόχρονα μένει ανοιχτό το ενδεχόμενο μελλοντικών επεκτάσεων.

Η αρχιτεκτονική του RapidIO ορίζεται ως μια ιεραρχία τριών στρωμάτων, το λογικό στρώμα, το στρώμα μεταφοράς και το φυσικό στρώμα. [5]

**Λογικό Στρώμα**

Το Λογικό Στρώμα ορίζει το γενικότερο πρωτόκολλο και την μορφή των πακέτων. Με άλλα λόγια καθορίζει τον τρόπο κατά τον οποίο διαχειρίζεται η διασύνδεση από τα τελικά σημεία. Οι τύποι συναλλαγών επίσης ορίζονται σε αυτό το Στρώμα.

Ο πρώτος τύπος αφορά την προσπέλαση συσκευών βάσει μηνυμάτων. Ορίζεται μία θύρα στο υλικό ως mailbox, μέσω της οποίας μπορεί να λάβουν μέρος λειτουργίες αποστολής και υποδοχής. Τα μηνύματα που χρησιμοποιούν αυτό το μέσο μπορούν να αποτελούνται από έως και 16 πακέτα, 256 byte το καθένα. Αυτό υπολογίζεται σε ένα μέγιστο μέγεθος των 4KB. Μία άλλη θύρα για υποδοχή μηνυμάτων είναι το doorbell. Αυτά τα ελαφριά μηνύματα χρησιμοποιούνται συνήθως για την ειδοποίηση κάποιου συμβάντος, με τρόπο παρόμοιο αυτού των συνήθων σημάτων και διακοπών(interrupts). Μπορούν να μεταφέρουν πληροφορίας σε ένα πεδίο μήκους 16-bit, το οποίο ορίζεται από το λογισμικό. Αυτού του τύπου η συναλλαγή μπορεί να αναφερθεί και ως το προγραμματιστικό μοντέλο μεταφοράς μηνυμάτων του πρωτοκόλλου.

Υποστηρίζεται επίσης ένα προγραμματιστικό μοντέλο που κάνει χρήση καθολικά κοινής μνήμης. Αυτό επιτρέπει την χρήση λειτουργιών MMIO. Τέτοιου είδους συναλλαγές επιτρέπουν την πρόσβαση σε τοπικές περιοχές μνήμης, και την μετάφραση των λειτουργιών επί αυτών, στις αντίστοιχες λειτουργίες για απομακρυσμένες περιοχές μνήμης. Η έννοια αυτή συνήθως αναφέρεται ως

Άμεση Πρόσβαση σε Απομακρυσμένη Μνήμη (Remote Direct Memory Access (RDMA)). Το μοντέλο αυτό επιτρέπει στο RapidIO, ένα RDMA-enabled δίκτυο διασύνδεσης να βρει εφαρμογή σε πολυπύρηνα, κατανεμημένα συστήματα που παρουσιάζουν ανάγκες υψηλών υπολογιστικών επιδόσεων.

**Στρώμα Μεταφοράς**

Το Στρώμα Μεταφοράς παρέχει την απαραίτητη πληροφορία για την επιτυχή δρομολόγηση πακέτων RapidIO από ένα σημείο σε ένα άλλο. Τα πακέτα RapidIO δρομολογούνται με βάση ένα μοναδικό αναγνωριστικό που είναι συσχετισμένο με κάθε συσκευή. Η συσχέτιση αυτή γίνεται κατά μία διαδικασία που ονομάζεται απαρίθμηση (enumeration), και παίρνει μέρος κατά την εγκατάσταση του συστήματος. Το αναγνωριστικό μπορεί να πάρει τη μέγιστη τιμή των 65535, επιτρέποντας την συνύπαρξη 65536 συσκευών στο ίδιο δίκτυο.

Είναι σημαντικό να σημειωθεί πως το πρωτόκολλο δεν ενδιαφέρεται για την φυσική τοπολογία του δικτύου. Μπορεί να ρυθμιστεί να δουλεύει στο περιβάλλον οποιασδήποτε τοπολογίας δικτύου, από δέντρα και πλέγματα, μέχρι υπερκύβους και toroids. Το RapidIO κάνει χρήση δρομολόγησης πηγής (source routing). Κάθε πακέτο έχει μια διεύθυνση παραλήπτη, η οποία έχει καθοριστεί από τον αποστολέα. Η πληροφορία αυτή επεξεργάζεται εν συνεχεία από το δίκτυο, το οποίο θα δρομολογήσει την συναλλαγή. Με αυτόν τον τρόπο, πόροι καταλαμβάνονται μόνο για τον αποστολέα και τον παραλήπτη, επιτρέποντας την ομαλή λειτουργία παράλληλων συναλλαγών. Αυτή η διαδικασία έρχεται σε αντίθεση με τον μηχανισμό broadcast, που συναντάται στα παραδοσιακά συστήματα διαύλων, όπως το PCI, κατά τον οποίο το πακέτο θα απασχολήσει όλα τα συνδεδεμένα στοιχεία.

**Φυσικό Στρώμα**

Το Φυσικό Στρώμα προσφέρει την σειριακή υλοποίηση του φυσικού δικτύου διασύνδεσης. Αυτό περιλαμβάνει τον μηχανισμό μεταφοράς τον πακέτων, πληροφορίες για τον έλεγχο ροής (flow control), διαχείριση σφαλμάτων στο υλικό ως επίσης και τα ηλεκτρικά χαρακτηριστικά της συσκευής. Τα πακέτα του φυσικού στρώματος είναι μικρά, με μικρό overhead, και επιτρέπουν την σύντομη επεξεργασία τους, ελαφρύνοντας τον επεξεργαστή Όταν η αποκατάσταση σφαλμάτων συμβαίνει στο υλικό, η αναγνώριση και η αναμετάδοση

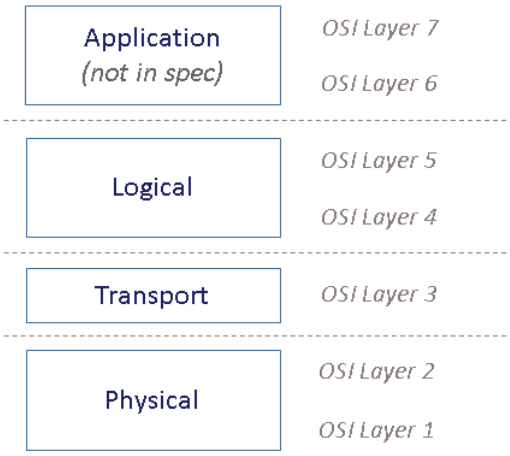
κατεστραμμένων ή χαμένων πακέτων ξεκινά άμεσα. Τα παραπάνω παρέχουν μια γρήγορη και αξιόπιστη παράδοση μεμονωμένων πακέτων.

Ο έλεγχος ροής ορίζεται επίσης στο φυσικό στρώμα και λαμβάνεται ιδιαίτερη μέριμνα ώστε να περιοριστεί το overhead και η πολυπλοκότητα. Διακρίνονται τρία είδη μηχανισμών ελέγχου ροής, έτσι ώστε συναλλαγές υψηλότερης προτεραιότητας να προηγούνται έναντι των χαμηλότερων προτεραιοτήτων. Repeat, throttle και credit. Ο μηχανισμός επανάληψης, ο πιο απλός, δηλώνει απλά ότι όταν ένας δέκτης δεν μπορεί να λάβει ένα πακέτο για οποιονδήποτε λόγο, μπορεί να απαντήσει με ένα σύμβολο ελέγχου "retry" στο τελικό σημείο της πηγής. Στη συνέχεια, η πηγή μπορεί να επαναμεταδώσει το πακέτο. Στο throttle χρησιμοποιείται το σύμβολο ελέγχου "idle", επιτρέποντας την εισαγωγή πακέτων που δρουν ως σήματα αναμονής. Τέλος, ο μηχανισμός credit, εκμεταλλεύεται ορισμένα σύμβολα ελέγχου για την επικοινωνία μεταξύ των τελικών σημείων. Με αυτόν τον τρόπο, μια συναλλαγή μπορεί να ξεκινήσει μόνο όταν ένας buffer είναι έτοιμος για εγγραφή.

Το RapidIO παρέχει ένα πλούσιο σύνολο λειτουργιών συντήρησης και διαχείρισης σφαλμάτων. Μια ειδική θύρα συντήρησης μπορεί να βρεθεί σε κάθε συσκευή η οποία προσφέρει πρόσβαση σε καταχωρητές που περιέχουν πληροφορίες σχετικά με τη συσκευή, συμπεριλαμβανομένων των δυνατοτήτων και των πληροφοριών μνήμης, των καταχωρητών ανίχνευσης σφαλμάτων και των καταχωρητών κατάστασης. Αυτοί οι καταχωρητές μπορούν επίσης να χρησιμοποιηθούν για την επαναφορά μιας συσκευής σε παθολογικά σενάρια. Η προδιαγραφή του φυσικού στρώματος αντιμετωπίζει επίσης την κάλυψη σφαλμάτων. Η ανίχνευση σφαλμάτων γίνεται μέσω της ανταλλαγής συμβόλων ελέγχου μεταξύ των τελικών σημείων. Σε περίπτωση που ο επανασυγχρονισμός δεν είναι επιτυχής, το υλικό μπορεί να δημιουργήσει ένα software trap, στέλνοντας μια διακοπή στο λογισμικό, διαβιβάζοντας την ευθύνη.

Το πρωτόκολλο ορίζεται με τέτοιο τρόπο ώστε να είναι ανεξάρτητο από τη προδιαγραφή του φυσικού στρώματος. Αυτό σημαίνει ότι το RapidIO μπορεί να χρησιμοποιηθεί σε σειριακές ή παράλληλες διεπαφές, μέσα χαλκού ή οπτικών ινών. Το RapidIO κρατάει χαμηλή την κατανάλωση ισχύος χρησιμοποιώντας LVDS(Low Voltage Differential Signaling).

Μια αντιστοιχία με τα στρώματα που παρουσιάστηκαν παραπάνω και το μοντέλο OSI (Open Systems Interconnection), φαίνεται στο σχήμα /reffig:osi.




Σχήμα 2.5: RapidIO protocol specification layers and their corresponding OSI layers.

### 2.2.2 RDMA

Το RapidIO είναι RDMA-enabled. Αυτός είναι ένας άλλος τρόπος για να δηλώσουμε ότι το πρωτόκολλο RapidIO υποστηρίζει Memory Mapped I/O, ή MMIO. Γενικά, τα δίκτυα διασύνδεσης με δυνατότητα RDMA χρησιμοποιούν τεχνικές memory coherence. Αυτό επιτρέπει κομμάτια μνήμης που βρίσκονται φυσικά σε διαφορετικά μέρη μέσα ή έξω από το μηχάνημα, να είναι προσβάσιμα από διαφορετικούς υπολογιστικούς κόμβους.

Το RDMA σημαίνει Remote Direct Memory Access, ή Απευθείας πρόσβαση σε Απομακρυσμένη Μνήμη. Τα RDMA-enabled δίκτυα διασύνδεσης μοιράζονται τρία χαρακτηριστικά που τα καθιστούν ελκυστικά για εφαρμογές σε εφαρμογές HPC.

**- Zero-Copy**

Ο όρος Zero-Copy αναφέρεται στη δυνατότητα μεταφοράς δεδομένων από έναν buffer στον κόμβο A σε έναν buffer του κόμβου B δίχως την προσπέλαση της στοίβας του δικτύου. Αυτό είναι προφανώς εφικτό με λειτουργίες MMIO, εφ' όσον το επιτρέπει η προγραμματική διεπαφή. Κανονικά, μια λειτουργία αποστολής/λήψης περιλαμβάνει μόνο τα εξής: Ο αποστολέας περνάει την διεύθυνση μνήμη των δεδομένων που θέλει να στείλει, και μία διεύθυνση στην κοινή μνήμη, στην προγραμματιστική διεπαφή, ενώ ο παραλήπτης απλά διαβάζει την κοινή μνήμη στο σχετικό σημείο. Αυτό εξαλείφει την ανάγκη για

ενδιάμεσες λειτουργίες πάνω σε μνήμη και την επιβάρυνση που τις συνοδεύει

**- Kernel bypass**

Ο πυρήνας δεν παίζει κάποιο ρόλο στην μεταφορά. Με άλλα λόγια, δεν χρειάζεται να γίνει κάποιο context change για να είναι επιτυχής η συναλλαγή με το απομακρυσμένο άκρο. Όλες οι απαραίτητες πληροφορίες ανταλλάσσονται κατά τη διάρκεια της φάσης εγκατάστασης μιας σύνδεσης. Αυτό περιλαμβάνει την ανταλλαγή διευθύνσεων μνήμης που θα χρησιμοποιηθούν μεταξύ των τελικών σημείων αλλά και της αντιστοίχησης τους προς την τοπική φυσική μνήμη και τον χώρο διευθύνσεων της εφαρμογής. Συνεπώς, μια κλήση συστήματος (system call) δεν χρειάζεται να πραγματοποιηθεί όταν μια συναλλαγή εκτελείται για μια ήδη-εγκατεστημένη σύνδεση.

**- CPU offloading**

Με την παράκαμψη της στοίβας δικτύου και του πυρήνα, οι κύκλοι του επεξεργαστή που καταναλώνονται ως αποτέλεσμα μιας συναλλαγής μειώνονται σημαντικά. Ταυτόχρονα, μια συναλλαγή δεν μεταφράζεται σε κατανάλωση κύκλων του απομακρυσμένου επεξεργαστή Αυτή η παρενέργεια ονομάζεται CPU offloading.

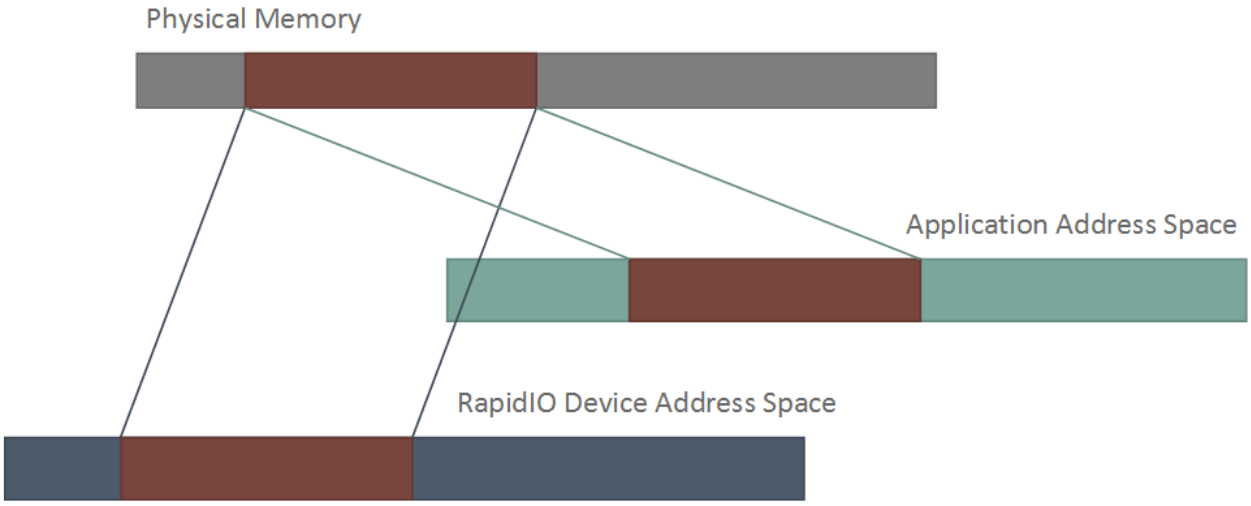
### 2.2.3 Προκλήσεις στην χρήση του RDMA

**Memory setup**

Το RDMA μπορεί να έχει πολλά πλεονεκτήματα, ωστόσο η αντίστοιχη προσπάθεια πρέπει να καταβληθεί από τον πλευρά του προγραμματιστή για να εκμεταλλευτεί τις δυνατότητές του.

Αρχικά, ορισμένες περιοχές μνήμης πρέπει να δεσμευτούν εκ των προτέρων, για να χρησιμοποιηθούν αποκλειστικά για λειτουργίες RDMA. Για ένα κανονικό σύστημα Linux αυτό θα συμβεί κατά την εκκίνηση, τροποποιώντας την γραμμή εντολών εκκίνησης του πυρήνα. Εκεί, πρέπει να προστεθεί μια παράμετρος που καθορίζει την αρχική διεύθυνση της μνήμης που θα δεσμευτεί, καθώς και το μέγεθός της. Μετά την δέσμευση της απαιτούμενης μνήμης, η εφαρμογή που σκοπεύει να την προσπελάσει πρέπει να την κάνει απεικονί-

σει στον χώρο διεύθυνσης διαδικασίας. Για παράδειγμα, όταν χρησιμοποιείται το RapidIO, ο χωρίς διευθύνσεων της διαδικασίας θα απεικονιστική στην φυσική μνήμη, η οποία με την σειρά της θα πρέπει να απεικονίζει τον χώρο διευθύνσεων που είναι ορατός στη συσκευή RapidIO. Αυτό το σχήμα μνήμης απεικονίζεται στο Σχήμα 2.6.




Σχήμα 2.6: Memory mapping when using RDMA

Μετά, η διεύθυνση της απεικονισμένης μνήμης πρέπει να μεταδοθεί στο απομακρυσμένο άκρο ή άκρα, προκειμένου να εκτελεστούν συναλλαγές με RDMA. Αυτό είναι κάτι που πρέπει να γίνει χειροκίνητα από την τρέχουσα εφαρμογή.

**Ενορχήστρωση**

Το RDMA περιγράφεται συχνά ως "επικοινωνία ενός δρόμου" (one way communication). Αυτό προκύπτει ως σύγκριση με την "αμφίδρομη επικοινωνία" (two way communication) ενός μοντέλου αποστολής/λήψης. Στο μοντέλο αυτό, ο αποστολέας θα στείλει δεδομένα σε ένα στόχο, με τον συγκεκριμένο στόχο να περιμένει να λάβει αυτά τα δεδομένα και με τη σειρά του να τα αποθηκεύσει σε συγκεκριμένη θέση. Για συναλλαγές με RDMA αυτό δεν συμβαίνει, καθώς το ένα άκρο θα κάνει μια άμεση πρόσβαση στη μνήμη, η οποία δεν απαιτεί τη συμμετοχή της CPU του απομακρυσμένου συστήματος. Η προφανής και άμεση παρενέργεια αυτού του γεγονότος είναι ότι το απομακρυσμένο άκρο πρέπει κάπως να ενημερωθεί για την πρόσβαση στη μνήμη. Με άλλα λόγια,

οι λειτουργίες RDMA απαιτούν ενορχήστρωση ως μέρος της συμβολής του χρήστη.

Υπο κανονικές περιστάσεις, η εγκατάσταση ενός περιβάλλοντος για RDMA, προκειμένου να εκτελεστεί μία συναλλαγή, θα απαιτούσε τα επόμενα βήματα υψηλού επιπέδου.

Στην πλευρά του παραλήπτη:

1. Απεικόνιση της δεσμευμένης φυσικής μνήμης στο χώρο διευθύνσεων της διεργασίας
2. Αρχικοποίηση ενός inbound window
3. Αποστολή της διεύθυνσης του inbound window στο απομακρυσμένο άκρο
4. Αναμονή για ειδοποίηση επιτυχούς εγγραφής
5. Ανάγνωση μνήμης

Στην πλευρά του αποστολέα:

1. Λήψη της απομακρυσμένης διεύθυνσης από το άλλο άκρο
2. Εκτέλεση εγγραφής στην απομακρυσμένη μνήμη
3. Αποστολή ειδοποίηση στο άλλο άκρο περί επιτυχούς εγγραφής

Για το RapidIO αυτή η ενορχήστρωση μπορεί να γίνει μέσω του προαναφερθείσαντος προγραμματιστικού μοντέλου μεταφοράς μηνυμάτων. Οι δυνατές επιλογές περιλαμβάνουν μηνύματα που χρησιμοποιούν μια θύρα γραμματοκιβωτίου (mailbox), η οποία όμως είναι χαμηλών επιδόσεων, και doorbells τα οποία είναι σήματα που λαμβάνουν χώρα στο επίπεδο του υλικού. Εναλλακτικά, θα μπορούσε να γίνει χρήση και του ίδιου του RDMA, περιμένοντας μέχρι να πραγματοποιηθεί κάποια προκαθορισμένη αλλαγή, όπως η αλλαγή στην τιμή ενός συγκεκριμένου bit στην μνήμη. Παρόμοιες τεχνικές χρησιμοποιούνται συνήθως για άλλα δίκτυα διασύνδεσης του ίδιου σχεδιασμού.

Μια άλλη σημαντική πτυχή είναι η διατήρηση των δεδομένων. Η μνήμη μπορεί να πανωγραφτεί(overwritten), εάν δεν υπάρχει ένας μηχανισμός κλειδώματος που να εμποδίζει την εγγραφεί. Αυτό το βάρος πέφτει στον προγραμματιστή και μπορεί να αποδειχθεί αρκετά δύσκολο όταν χειρίζεται ένα μεγάλο

αριθμό διαφορετικών κομματιών μνήμης, ειδικά όταν αυτά προέρχονται από πολλαπλούς κόμβους.

### 2.2.4 Link Speed

#### Θεωρητική Προσέγγιση

Οι κάρτες δικτύου που χρησιμοποιούνται για τους σκοπούς αυτής της δουλειάς, είναι κάρτες από RapidIO σε PCIe. Η ονομαστική ταχύτητα της γραμμής τους είναι 16Gbps. Και η πλευρά του RapidIO και η πλευρά του PCIe προσφέρουν

- 5 GBaud ταχύτητα σύνδεσης
- εύρος συνδέσμου(link width) x4
- κωδικοποίηση 8b\10b: λέξεις των 8bit απεικονίζονται σε σύμβολα των 10bit

Συνεπώς, η ταχύτητα του συνδέσμου σε bit υπολογίζεται ως εξής:

$$\textbf{Link Speed} = \frac{(linkwidth) * (linkspeed) * (\#bits)}{(\#symbols)} Gbps \implies$$

$$\textbf{Link Speed} = \frac{(4) * (5) * (8)}{(10)} = 16Gbps$$

Παρόλα αυτά, οι μεταφράσεις πρωτοκόλλου από PCIe σε RapidIO και αντίστροφα πρέπει επίσης να ληφθούν υπόψη. Αυτό σημαίνει ότι η ταχύτητα που υπολογίζεται στην προηγούμενη εξίσωση δεν παρατηρείται ποτέ στην πραγματικότητα. Η πραγματική ταχύτητα σύνδεσης μειώνεται σε περίπου 13,4Gbps. Ωστόσο, αυτό ισχύει μόνο σε για την μετάδοση. Λόγω καλύτερης ανταπόκρισης σε επίπεδο πρωτοκόλλου για την λήψη, η επιβάρυνση είναι χαμηλότερη και η ταχύτητα μπορεί να φτάσει τα 14,6 Gbps.

Οι παραπάνω τιμές αναφέρονται σε αποστολή και λήψη πάνω από RDMA, καθώς οι λειτουργίες αυτές έχουν λιγότερο overhead, και επιτρέπουν αυτά τα επίπεδα throughput.

Ακολουθεί η ανάλυση των δύο λειτουργιών, αποστολής και λήψης. Σε κάθε

ενέργεια αντιστοιχίζεται ένας αριθμός, ο οποίος θα χρησιμοποιείται ως αναφορά στην ενέργεια αυτή. Το κομμάτι που ακολουθεί προκύπτει από τις προδιαγραφές και του PCIe [10] και του RapidIO [11].

### DMA Transmit

Για το DMA Transmit, τα δεδομένα ξεκινούν από την CPU, προωθούνται στην Tsi721, η οποία στην συνέχεια τα στέλνει στο δίκτυο RapidIO. Ακολουθεί μια περιγραφή αυτής της διαδικασίας, στην οποία υπάρχουν λεπτομερείς υπολογισμοί για τον αριθμό των bytes που απαιτείται για κάθε συναλλαγή. Έτσι γίνεται ξεκάθαρο το ποσοστό της ταχύτητας του συνδέσμου που αντιστοιχεί σε overhead.

**– Η CPU θέλει να γράψει στο Tsi721 –**

*CPU -> Tsi721*

Η CPU κάνει μια εγγραφή σε καταχωρητή για να προκαλέσει τη μετάδοση (Tx) στην PCIe πλευρά της Tsi721. [1]

$S_c = (PCIeRegisterWrite + RegisterWritePaylaod) = 28Bytes$

*CPU <- Tsi721* [2]

Το PCIe απαντά με ένα PCIe Acknowledgment και ένα Credit Update. [2]

$S_p = (PCIeAcknowledgment + PCIeCreditUpdate) = (8 + 8) = 16Bytes$

**– Η Tsi721 ζητά έναν περιγραφητή για ανάγνωση από την CPU –**

*CPU <- Tsi721*

Το PCIe κάνει μια αίτηση για ανάγνωση στην CPU. [3]

$S_{p0} = (PCIeReadRequest) = 24Bytes$

*CPU -> Tsi721*

The CPU acknowledges the request. [4]

$S_{c0} = (PCIeAcknowledgment + PCIeCreditUpdate) = (8 + 8) = 16Bytes$

Και μετά στέλνει τον περιγραφητή. [5]

$S_{c1} = (PCIeReadCompletionOverhead + DescriptorSize) = (24 + 64) = 88Bytes$

$S_c = S_c0 + S_c1 = 104Bytes$

*CPU <- Tsi721*

Το PCIe κάνει acknowledge την τελευταία ενέργεια. [6]

$S_{p1} = (PCIeAcknowledgment + PCIeCreditUpdate) = (8 + 8) = 16Bytes$

$S_p = S_{p0} + S_{p1}40Bytes$

**– Ανάγνωση δεδομένων από την CPU –**

*CPU <- Tsi721*

Το PCIe κάνει αιτήματα ανάγνωσης. [7]

$S_{p0} = ((MTU/PCIeMaxReadSize) * ReadRequestSize) = (65536/4096) * 24 = 384Bytes$

*CPU -> Tsi721*

Η CPU κάνει acknowledge τα αιτήματα [8]

$S_{c0} = ((MTU/PCIeMaxReadSize)*(PCIeAcknowledgment+PCIeCreditUpdate)) = (655636/4096) * (8 + 8) = 256Bytes$

Και στέλνει τα δεδομένα [9]

$S_{c1} = MTU/PCIeEfficiency = 65536/84.21\% = 77824Bytes$

$S_c = S_{c0} + S_{c1} = 77824 + 256 = 78080Bytes$

*CPU <- Tsi721*

Το PCIe κάνει acknowledge τα δεδομένα. [10]

$S_{p1} = (MTU/PCIePayload) * (PCIeAcknowledgment + PCIeCreditUpdate) = (65536/128) * (8 + 8) = 8192Bytes$

$S_p = S_{p0} + S_{p1} = 8576Bytes$

**– Η Tsi721 στέλνει τα δεδομένα στο δίκτυο RapidIO –**

*Tsi721 -> RIO*

Η Tsi721 αποστέλλει τα δεδομένα. [11]

$S_{p0} = MTU/RIOEfficiency = 65536/94.12\% = 69630Bytes$

Και ένα σύμβολο ελέγχου (control symbol) για κάθε πακέτο που στάλθηκε. [12]

$S_{p1} = (MTU/RIOPayload) * RIOControlSymbolSize = (65536/256) * 4 = 1024Bytes$

$S_p = S_{p0} + S_{p1} = 70654Bytes$

*Tsi721 <- RIO*

Το RapidIO κάνει acknowledge κάθε πακέτο που λήφθηκε με ένα σύμβολο ελέγχου [13].

$S_r = (MTU/RIOPayload) * RIOControlSymbolSize = (65536/256) * 4 = 1024Bytes$

**– Tsi721 has to inform the CPU of the completion –**

*CPU <- Tsi721*

Το PCIe εκτελεί ένα “write completion” στην CPU [14]

$S_p = (PCIeTotal - PCIePayload + WriteCompletionSize) = 152 - 128 + 64 = 88Bytes$

*CPU -> Tsi721*

Η CPU κάνει acknowledge την τελευταία ενέργεια εγγραφής [15]

$S_c = PCIeAcknowledgment + PCIeCreditUpdate = 16Bytes$

**DMA Receive**

Για ένα DMA Receive, τα δεδομένα λαμβάνονται από την πλευρά του RapidIO, μετά πάνε στην Tsi721 και μετά προωθούνται στην CPU.

Για την λειτουργία αυτή η κατεύθυνση των δεδομένων είναι: RapidIO Network -> Tsi721 -> CPU.

**– Η Tsi721 λαμβάνει δεδομένα από το δίκτυο RapidIO –**

*Tsi721 <- RIO*

Ακριβώς το ίδιο με την αντίθετη κατεύθυνση στο βήμα της μεταφοράς δεδομένων για το DMA Tx.

Δεδομένα, [1]

$S_{r0} = MTU/RIOEfficiency = 65536/94.12 \ = 69630Bytes$

Και ένα σύμβολο ελέγχου για κάθε πακέτου που στάλθηκε, [2]

$S_{r1} = (MTU/RIOPayload) * RIOControlSymbolSize = (65536/256) * 4 = 1024Bytes$

$S_r = S_{r0} + S_{r1} = 70654Bytes$

*Tsi721 -> RIO*

Όπως και πριν, ακριβώς τα ίδια με την αντίθετη κατεύθυνση για το βήμα μεταφοράς δεδομένων στο DMA Tx.

Η Tsi721 κάνει acknowledge κάθε πακέτο που παραλήφθηκε, με ένα σύμβολο ελέγχου. [3]

$S_p = (MTU/RIOPayload) * RIOControlSymbolSize = (65536/256) * 4 = 1024Bytes$

**– Η Tsi721 στέλνει τα δεδομένα στην CPU –**

*CPU <- Tsi721*

Η Tsi721 στέλνει τα δεδομένα που περιέχονται στα πακέτα RapidIO, μέσω του συνδέσμου PCIe, στην CPU. [4]

$S = (MTU/PCIeEfficiency) = (65536/0.8421) = 77824Bytes$

*CPU -> Tsi721*

Και η CPU τα κάνει acknowledge. [5]

$S = (MTU/RIOPayload) * (PCIeAcknowledgment + PCIeCreditUpdate) = (65536/256) * (8 + 8) = 4096Bytes$

Όλα τα νούμερα που χρησιμοποιήθηκαν στους παραπάνω υπολογισμούς, μπορούν να βρεθούν στις προδιαγραφές των RapidIO και PCIe, αλλά και στον σχετικό πίνακα, βλ. Supporting Materials.

**Υπολογισμοί Ταχύτητας**

**DMA Tx**

Αθροίζοντας κάθε $S_c$ και $S_p$ στην πλευρά του **PCIe** της λειτουργίας DMA Tx έχουμε:

$sumS_c = 28 + 104 + 78080 + 16 = 78228 * 8 = 625824bits$

$sumS_p = 16 + 40 + 8576 + 88 = 8720 * 8 = 69760bits$

Τώρα, ο αριθμός των συναλλαγών ανά δευτερόλεπτο, στην πλευρά του PCIe θα είναι:

$numT_c = (PCIeSpeed * 1G)/sumS_c = 16000000000/625824 = 25566$

$numT_p = (PCIeSpeed * 1G)/sumS_p = 16000000000/69760 = 229358$

Η ταχύτητα για αυτό το κομμάτι της συναλλαγής θα καθοριστεί από τον χαμηλότερο πιθανό αριθμό συναλλαγών, καθώς αυτός θα είναι ο περιοριστικός παράγοντας. Συνεπώς, η ταχύτητα του PCIe θα περιοριστεί από την κίνηση που προέρχεται από την CPU.

$PCIeBusSpeed = (MTU * numT_c) * 8/1G = (65536 * 25566) * (8/1G) = 13.40Gbps$

Επαναλαμβάνοντας την διαδικασία για την πλευρά του **RapidIO**:

$sumS_p = 70654 * 8 = 565232bits$

$sumS_r = 1024 * 8 = 8192bits$

$numT_p = (RIOSpeed * 1G)/sumS_p = 16000000000/565232 = 28306$

$numT_r = (RIOSpeed * 1G)/sumS_r = 16000000000/565232 = 1953125$

Ακολουθώντας την ίδια λογική, η κίνηση που προέρχεται από την Tsi721 περιορίζει την ταχύτητα του δικτύου RapidIO.

$RapidIONetworkSpeed = (MTU * numT_p) * (8/1G) = (65536 * 28306) * (8/1G) = 14.84Gbps$

Είναι ξεκάθαρο, πως η πλευρά του PCIe είναι σημαντικά πιο αργή σε σχέση

με την πλευρά του RapidIO, καθορίζοντας την μέγιστη ταχύτητα για DMA Tx στα **13.40 Gbps**.

**DMA Rx**

Στην πλευρά του δικτύου**RapidIO**:

$S_r = 70654 * 8 = 565232bits$

$S_p = 1024 * 8 = 8192bits$

Που δίνουν:

$numT_r = (RIOSpeed * 1G)/S_r = 16000000000/565232 = 28306$

$numT_p = (RIOSpeed * 1G)/S_p = 16000000000/8192 = 1953125$

Ο χαμηλότερος αριθμός καθορίζει την μέγιστη ταχύτητα σε αυτό το μέρος του δικτύου:

$RapidIONetworkSpeed = (MTU * numT_p) * (8/1G) = (65536 * 1953125) * (8/1G) = 14.84Gbps$

Για την ταχύτητα του διαύλου **PCIe**:

$S_p = 77824 * 8 = 622592bits$

$S_c = 4096 * 8 = 32768bits$

Και προκύπτει:

$numT_p = (PCIeSpeed * 1G)/S_p = 16000000000/622592 = 25700$

$numT_c = (PCIeSpeed * 1G)/S_p = 16000000000/32768 = 488281$

Και η ταχύτητα για αυτό το μέρος της συναλλαγής:

$$PCIeBusSpeed = (MTU * numT_p) * (8/1G) = 13.47Gbps$$

Όπως και πριν, η πλευρά του PCIe ορίζει την ταχύτητα για το DMA Rx στα **13.47 Gbps**

Συμπερασματικά, ο δίαυλος PCIe πάντοτε υπερτερεί σε σχέση με την διασύνδεση του RapidIO ανεξάρτητα από τον τύπο συναλλαγής. Ωστόσο, οι προηγούμενοι υπολογισμοί έγιναν με την υπόθεση ότι το μέγεθος PCIe Max Completion για το σύστημά μας ήταν 128 Bytes. Ο επόμενος πίνακας περιέχει επίσης τα αντίστοιχα αποτελέσματα για ένα σύστημα που υποστηρίζει μέγεθος 256 Bytes PCIe Max Completion.

| | **128 Bytes** | **256 Bytes** |
|---|---|---|
| **DMA Tx Speed** | 13.40 Gbps | 14.55 Gbps |
| **DMA Rx Speed** | 13.47 Gbps | 14.63 Gbps |

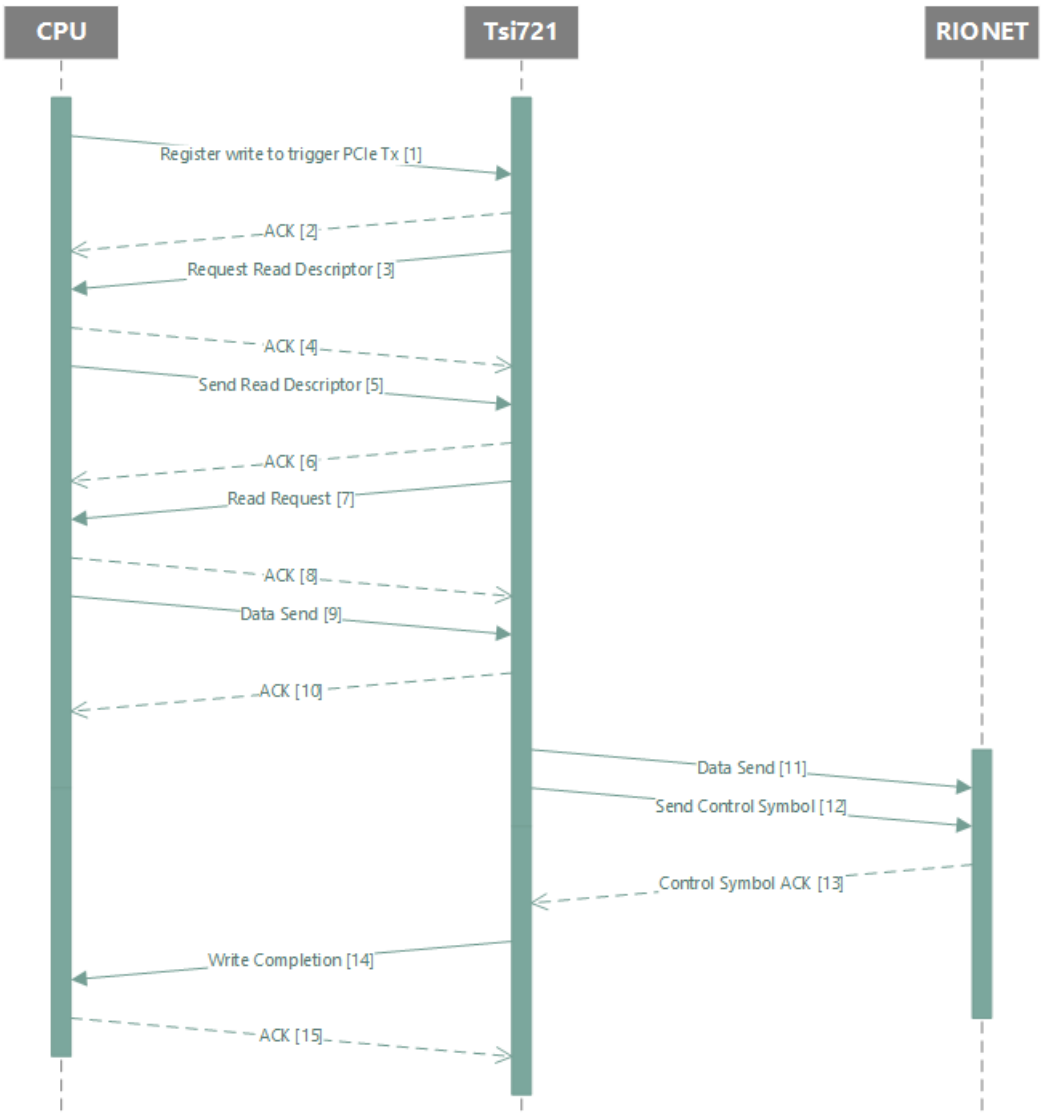
Πίνακας 2.1: PCIe Max Completion Size - Speed correlation

**Supporting Materials**

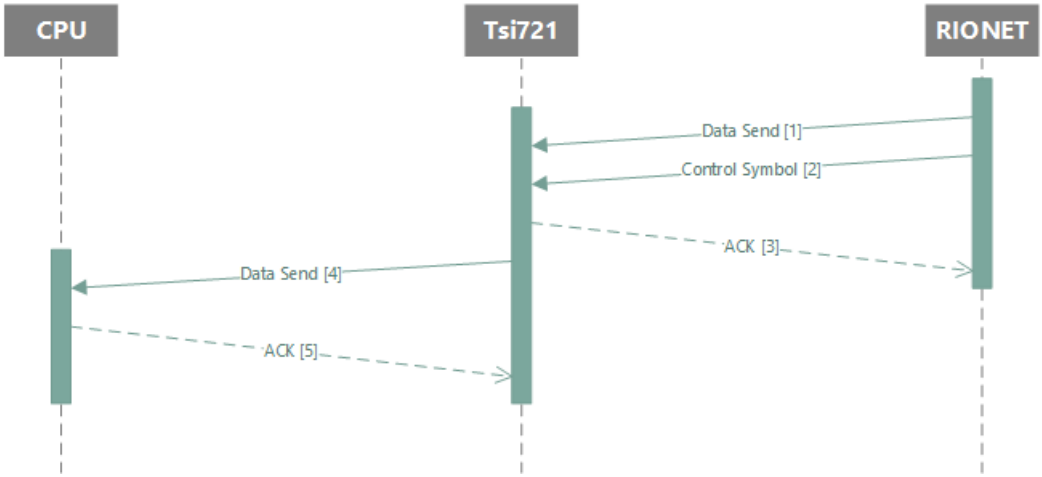
Στις εικόνες 2.7 και 2.8 απεικονίζονται τα διαγράμματα ακολουθίας για τα σενάρια των DMA Tx και DMA Rx. Οι actors σε αυτά τα διαγράμματα είναι:

- Η κύρια μνήμη και η CPU [CPU]
- Η κάρτα Tsi721 [Tsi721]
- Το δίκτυο του RapidIO, συμπεριλαμβανομένων άκρων και μεταγωγών [RIONET]

Αξίζει να σημειωθεί πως στην πλευρά της CPU στο Tsi721, όλες οι συναλλαγές γίνονται πάνω από PCIe, ενώ, φυσικά στην πλευρά του RapidIO, είναι όλες τύπου RapidIO.




Σχήμα 2.7: Sequence diagram for DMA Tx




Σχήμα 2.8: Sequence diagram for DMA Rx

| **SWRITE RapidIO Efficiency** | **Bytes** |
|---|---|
| Physical Header | 2 |
| Transport Header | 2 |
| Address | 6 |
| Data | 256 |
| CRC | 4 |
| PAD | 2 |
| Total | 272 |
| Efficiency | 94.12% |

Πίνακας 2.2: RapidIO Packet

| **PCIe Efficiency** | **Bytes** |
|---|---|
| Address Size | 64 |
| Header | 16 |
| MAX Completion Data | 128 |
| LCRC | 4 |
| ECRC | 4 |
| Total | 152 |
| Efficiency | 84.21% |

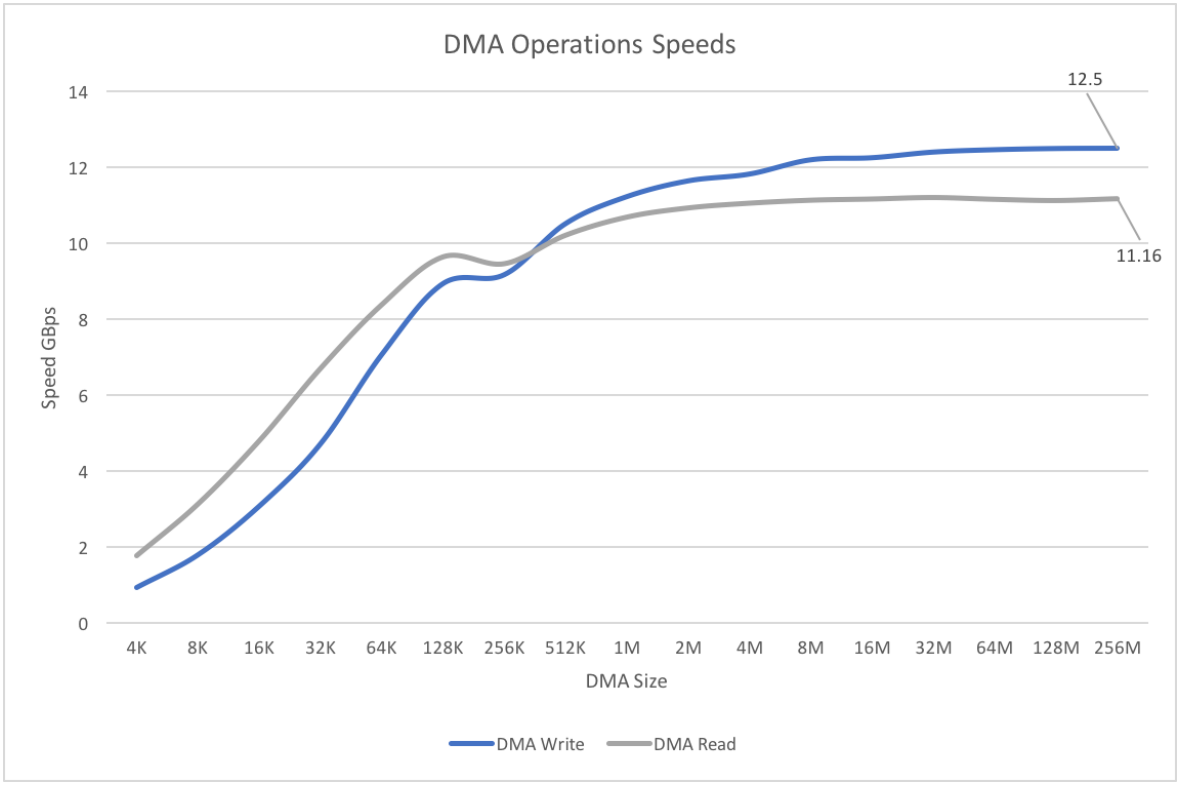
Πίνακας 2.3: PCIe Packet

| | **Bytes** |
|---|---|
| PCIe Read Request | 24 |
| PCIe Register Write Overhead | 20 |
| PCIe Register Write Payload | 8 |
| PCIe Acknowledge | 8 |
| PCIe Credit Update | 8 |
| PCIe Read Completion Overhead | 24 |
| RapidIO Control Symbol | 4 |
| RapidIO MAX Payload | 256 |
| MAX Completion Data | 128 |
| Descriptor Size | 64 |
| MTU | 65536 |

Πίνακας 2.4: Miscellaneous Values

### 2.2.5 Πειραματική Προσέγγιση

Μπορούμε να ελέγξουμε τα παραπάνω αποτελέσματα, μετρώντας το χρόνο γύρω από τις κλήσεις της βιβλιοθήκης, `dma_write` και `dma_read`. 512MB δεδομένων έχουν σταλεί επανειλημμένα, αυξάνοντας το μέγεθος των DMA buffer. Το μέγεθος των buffer διακυμάνθηκε από 4K έως 256MB, που ήταν το μέγεθος της δεσμευμένης μνήμης του συστήματος και συνεπώς το μέγιστο δυνατό μέγεθος. Τα αποτελέσματα απεικονίζονται στο σχήμα 2.9.




Σχήμα 2.9: Speed of DMA operations

Το παραπάνω σχήμα δείχνει ότι περίπου στα 8MB προσεγγίζεται ένα μέγιστο περί των 12,5 Gbps για την εγγραφή, ενώ η ανάγνωση περιορίζεται στα 11,16 Gbps. Αυτές οι ταχύτητες καθορίζουν το overhead που εισάγεται από τη βιβλιοθήκη και τον οδηγό, τα οποία παρουσιάζονται στον πίνακα 2.5

| | Write | Read |
|---|---|---|
| **Theoretical** | 13.40 Gbps | 13.47 Gbps |
| **Experimental** | 12.50 Gbps | 11.16 Gbps |
| **Overhead** | 1.10 Gbps | 2.31 Gbps |

Πίνακας 2.5: DMA Operations Speeds

### 2.2.6 Software Stack

Η στοίβα λογισμικού RapidIO που διατίθεται σήμερα αποτελείται από τους οδηγούς πυρήνα του Linux [12] και τις βιβλιοθήκες RRMAP [13], που αναπτύχθηκαν από την IDT. Το πακέτο λογισμικού βρίσκεται υπό ενεργό ανάπτυξη.

Τα προγράμματα χώρου χρηστών διασυνδέονται με τη διασύνδεση μέσω κλήσεων βιβλιοθήκης, οι οποίες με τη σειρά τους αλληλεπιδρούν με τον υποκείμενο οδηγό. Οι κλήσεις βιβλιοθήκης θα χειρίζονται όλες τις λειτουργίες διαχείρισης και συναλλαγών.

Τα μηνύματα στρώματος μεταφοράς καλούνται Channelized Messaging μέσα σε αυτήν τη στοίβα. Οι κλήσεις βιβλιοθήκης που είναι υπεύθυνες για τις ακόλουθες λειτουργίες είναι διαθέσιμες. Αρχικοποίηση μιας "υποδοχής" η οποία σχετίζεται με ένα συγκεκριμένο κανάλι γραμματοκιβωτίου. Ακρόαση για συνδέσεις στην υποδοχή. Αποδοχή συνδέσεων στην υποδοχή. Σύνδεση σε υποδοχή απομακρυσμένου τελικού σημείου σε συγκεκριμένο κανάλι. Αποστολή και λήψη στο πλαίσιο μιας συγκεκριμένης υποδοχής. Τα καναλιού μηνύματα θα αποτελούνται πάντοτε από προσωρινές μνήμες των 4096 byte. Από αυτά τα 2ο bytes απαιτούνται για το γενικό πρωτόκολλο, μειώνοντας το μέγεθος του ωφέλιμου φορτίου στα 4076 byte.

Όπως περιγράφηκε νωρίτερα, τα doorbells μεταφέρουν το αναγνωριστικό του αποστολέα, το αναγνωριστική του παραλήπτη και μία τιμή μήκους 16-bit που ορίζει ο χρήστης. Επιπλέον, μία συνάρτηση callback πρέπει να οριστεί για την περίπτωση που έρθει ένα doorbell event. Προτού ένα άκρο να μπορεί να δεχτεί και να εξυπηρετήσει εισερχόμενα doorbells, ένα εύρος τιμών πρέπει να δεσμευτεί. Αν ένα doorbell που καταφτάνει, δεν ανήκει σε αυτό το εύρος, αγνοείται. Όλες αυτές οι λειτουργίες εκτελούνται μέσω συγκεκριμένων συ-

ναρτήσεω που προσφέρει το λογισμικό.

Οι λειτουργίες RDMA μπορούν να πάρουν μέρος με αυθαίρετα μεγάλες περιοχές μνήμης. Ωστόσο, οι περιοχές αυτές εξαρτώνται από το μέγεθος της δεσμευμένης μνήμης για RDMA, γεγονός που συζητείται στο κεφάλαιο του πρωτοκόλλου. Όταν εκτελείται μια συναλλαγή RDMA, το ένα άκρο χρειάζεται να κάνει μια κλήση στη βιβλιοθήκη, ενώ το άλλο αρκεί να κάνει ένα memcpy. Αυτό εξαρτάται από τον τρόπο με τον οποίο η σύνδεση εγκαθιδρύθηκε. Η συμπεριφορά μπορεί να είναι και η αντίστροφη χωρίς καμία απολύτως ουσιαστική διαφορά.

## 2.3 ZeroMQ

Το ZeroMQ, ή 0MQ, είναι ένα σύστημα ασύγχρονης ανταλλαγής μηνυμάτων υψηλής επίδοσης. Λύσεις middleware που προσανατολίζονται γύρω από μηνύματα χρησιμοποιούνται σε μια πληθώρα εφαρμογών κατανεμημένων συστημάτων, από χρηματοοικονομικές υπηρεσίες μέχρι προσομοιώσεις φυσικής. Το ZeroMQ χρησιμοποιείται ως βιβλιοθήκη που επιτρέπει την ανταλλαγή μηνυμάτων ανάμεσα στα διάφορα στοιχεία του συστήματος. Τα μηνύματα αυτά μπορεί να είναι αυθαίρετα μεγάλα. Το ZeroMQ αγνοεί το περιεχόμενο που μεταφέρει και συνεπώς δεν ορίζει κάποιον περιορισμό στην χρήση του. Σε αντίθεση με άλλα συστήματα ανταλλαγής μηνυμάτων, το ZeroMQ, ως βιβλιοθήκη, δεν χρησιμοποιεί κάποιον μεσάζων(broker). Αυτό μειώνει την πολυπλοκότητα και καταργεί την ανάγκη για συντήρηση.

Αν και αρχικά το ZeroMQ απευθυνόταν σε αγοραπωλησίες μετοχών, άλλαξε κατεύθυνση, για να γίνει ένα τυπικό σύστημα υποστήριξης εφαρμογών σε κατανεμημένα περιβάλλοντα. Η βιβλιοθήκη έχει βρει πολλές εφαρμογές σε οργανισμούς όπως η Cisco, η NASA, η Samsung και το CERN. Το ZeroMQ είναι λογισμικό ανοιχτού κώδικα, από το οποίο έχουν γεννηθεί τα JeroMQ και Crossroads I/O, το οποίο αργότερα εξελίχθηκε στο Nanomg.

Η εσωτερική αρχιτεκτονική της βιβλιοθήκης του ZeroMQ είναι δομημένη με τέτοιο τρόπο ώστε να αποτελεί καλό υποψήφιο για την επέκτασή της με νέες τεχνολογίες, χωρίς να παρεμβάλλεται με τα άλλα μέρη του συστήματος. [14]

### 2.3.1 Εσωτερική Αρχιτεκτονική του ZeroMQ

#### Context

Το ZeroMQ χρησιμοποιεί την κλάση context, `ctx_t`, η οποία διατηρεί την καθολική κατάσταση της βιβλιοθήκης. Δημιουργείται ρητά από τον χρήστη και είναι η πρώτη ενέργεια για την αρχικοποίηση της υποδομής της βιβλιοθήκης. Το context διατηρεί πληροφορία σχετικά με τα sockets, τα νήματα Ε/Ε και τα τελικά σημεία.

#### Μοντέλο Συγχρονισμού

Το ZeroMQ είναι μια πολυνηματική εφαρμογή, στην οποία κάθε αντικείμενο ζει στο δικό του νήμα. Προκειμένου δύο αντικείμενα να επικοινωνήσουν μεταξύ τους, ανταλλάσσουν εντολές (και όχι μηνύματα). Κατ' αυτόν τον τρόπο εξαλείφεται η ανάγκη για ενορχήστρωση μέσω κλειδωμάτων. Συνεπώς, δεν συναντώνται mutex, σημαφόροι ή μεταβλητές συνθήκης. Προκειμένου να μπορεί ένα αντικείμενο να ανταλλάξει εντολές, πρέπει να είναι υποκλάση του `object_t`. Διαθέσιμες εντολές μπορούν να βρεθούν στο `command.hpp`. Οι εντολές δύνανται να σταλούν και με ορίσματα.

#### Νηματικό Μοντέλο

Υπάρχουν δύο τρόπου προσέγγισης των νημάτων του ZeroMQ. Ένας από την σκοπιά του λειτουργικού συστήματος και ένας από την σκοπιά της βιβλιοθήκης.

Το λειτουργικό σύστημα βλέπει δύο ειδών νήματα. Το πρώτο είδος είναι τα "application threads". Αυτά τα νήματα έχουν δημιουργηθεί εκτός του ZeroMQ και χρησιμοποιούνται για την προσπέλαση της προγραμματιστικής διεπαφής (API). Το δεύτερο είδος είναι τα "I/O threads". Τα νήματα αυτά δημιουργούνται εντός ενός ZeroMQ context και χρησιμοποιούνται για την διεκπεραίωση λειτουργιών αποστολής και λήψης στο παρασκήνιο. Για τα νήματα χρησιμοποιείται μια OS-agnostic κλάση φορητότητας, η `thread_t`.

Όσον αφορά το ZeroMQ, κάθε αντικείμενο που έχει `mailbox` θεωρείται νήμα.

Το mailbox είναι η δομή που υλοποιείται ως μέρος της κλάσης `mailbox_t` και χρησιμοποιείται ως ουρά για τις εντολές με προορισμό το αντικείμενο που ζει στο εν λόγω νήμα. Οι εντολές επεξεργάζονται σειριακά.

Ωστόσο, και τα I/O threads, όπως επίσης και τα sockets, έχουν ένα mailbox. Κάθε I/O thread μεταφράζεται σε ένα OS thread, και έχει ένα μοναδικό mailbox, στο οποίο δέχεται εισερχόμενες εντολές. Επιπροσθέτως, πολλαπλά sockets μπορούν να μοιράζονται ένα μόνο OS thread, και ενδεχομένως να μεταπηδούν μεταξύ διαφορετικών νημάτων.

### I/O Threads

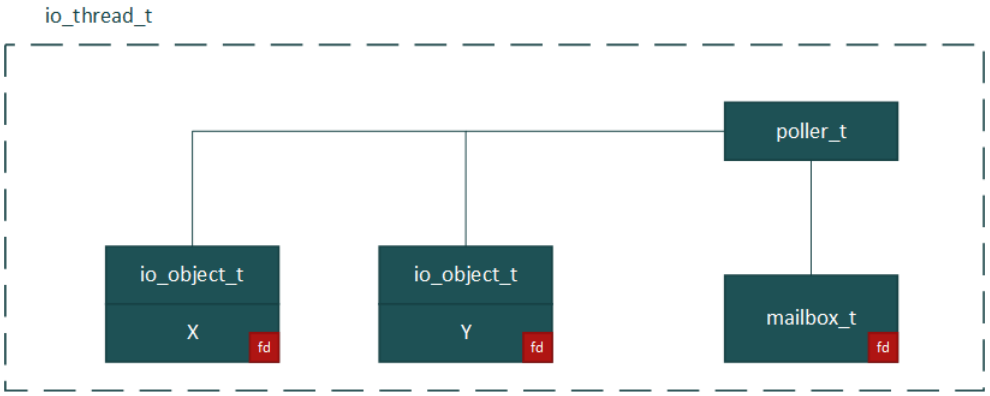
Τα I/O threads τρέχουν στο παρασκήνιο, και είναι υπεύθυνα να διαχειρίζονται την κίνηση του δικτύου με ασύγχρονο τρόπο. Η κλάση `iothread_t` είναι υποκλάση της `thread_t`, που αναφέρθηκε νωρίτερα. Είναι επίσης υποκλάση της `object_t`, ώστε να επιτρέπει την ανταλλαγή εντολών με άλλα αντικείμενα για να οργανώνονται οι διάφορες λειτουργίες.

Επίσης, κάθε I/O thread έχει ένα αντικείμενο poller. Ο poller (`poller_t`) προσφέρει ένα επίπεδο αφαίρεσης για τυχόν διαφορετικές μεθόδους polling που θα χρησιμοποιηθούν, όπως το poll ή το select.

Επί πλέον, κάθε αντικείμενο που ζει σε ένα I/O thread, προέρχεται από την βοηθητική κλάση `io_object_t`. Ένα `io_object_t` επιτρέπει την εγγραφή ενός περιγραφητή αρχείου (file descriptor (fd)) με τον poller του νήματος, μέσω της συνάρτησης `add_fd()`. Έτσι, κάθε φορά που ένα fd event παίρνει μέρος, ο poller θα ενεργοποιήσει μία συνάρτηση callback. Συνεπώς, κάθε `io_object` υλοποιεί τις συναρτήσεις, `in_event()` και `out_event()`, για την διαχείριση fd events. Όταν ο περιγραφητής αρχείου δεν είναι πια απαραίτητος μπορεί να διαγραφεί, μέσω της συνάρτησης `rm_fd()`. Χρονομετρητές μπορούν επίσης να χρησιμοποιηθούν με αντίστοιχο τρόπο.

Μία άλλη σημαντική παρατήρηση, είναι πως το `io_thread` θα κάνει εγγραφή ενός ακόμα περιγραφητή αρχείου στον poller. Αυτό το fd είναι για την εξυπηρέτηση του mailbox του ίδιου του νήματος. Όταν μια εντολή καταφτάνει, ο poller θα καλέσει την `in_event()` του `io_thread_t`, η οποία θα προωθήσει την εντολή στο προορίζων αντικείμενο που φιλοξενεί.

Στην εικόνα 2.10 παρουσιάζονται συνοπτικά τα παραπάνω.




Σχήμα 2.10: I/O Thread Overview

Στην παραπάνω εικόνα, το αντικείμενο χ π.χ., μπορεί να έχει κάνει την εγγραφή ενός περιγραφητή αρχείου με τον poller. Ας υποθέσουμε πως αυτός ο περιγραφητής αρχείου αναφέρεται στο σύστημα αρχείων του λειτουργικού συστήματος. Όταν κάποιος εκτελέσει μια εγγραφή στο αρχείο αυτό, ένα POLLIN event θα πάρει μέρος, και θα "πιαστεί" από τον poller. Σε επόμενο βήμα, ο poller θα καλέσει την `in_event` συνάρτηση του αντικειμένου X, η οποία θα διαχειριστεί το event αναλόγως.

### Δέντρα Αντικειμένων

Τα αντικείμενα που δημιουργούνται εντός της βιβλιοθήκης του ZeroMQ είναι οργανωμένα σε μια ιεραρχία δέντρου, με την ρίζα του να είναι πάντα ένα socket. Όπως συζητήθηκε νωρίτερα, κάθε αντικείμενο μπορεί να ζει σε διαφορετικό I/O thread, με την εξαίρεση του socket, το οποίο ζει σε OS thread. Ο σκοπός αυτής της σχεδιαστικής απόφασης είναι η εφαρμογή ενός ντετερμινιστικού μηχανισμού τερματισμού λειτουργίας. Γενικά μιλώντας, όταν ένα αντικείμενο τερματίζει, θα στείλει σχετικές εντολές τερματισμού σε όλα τα παιδιά του, προτού τερματίσει το ίδιο.

Προκειμένου να υπάρξει μέριμνα για ακραίες περιπτώσεις, όπως η απόφαση για ταυτόχρονο τερματισμό τόσο του γονέα όσο και του παιδιού, όταν ένα αντικείμενο θέλει να τερματίσει, θα ζητήσει από τον γονέα του να το κάνει για αυτό.

Ο μηχανισμός δέντρων αντικειμένων είναι υλοποιημένος στην `own_t`, υπο-

κλάση της `object_t`, ώστε να επιτρέπει την ανταλλαγή μηνυμάτων ανάμεσα στα στοιχεία του δέντρου. Στην κλάση `own_t` υλοποιείται επίσης η συνάρτηση `launch_child`, η οποία συνδέει το αντικείμενο-παιδί με το παρόν I/O thread, διαδικασία η οποία επίσης ξεκινάει την εκτέλεσή του.

#### Messages & Pipes

Το ZeroMQ πληροί πολύπλοκες απαιτήσεις για τα μηνύματά του, τα οποία δρομολογούνται μέσω αποδοτικών υλοποιήσεων pipes, συμβάλλοντας στην επίδοσή του. Ωστόσο, αυτές οι δομές δεδομένων δεν έχουν ιδιαίτερο ενδιαφέρον στα πλαίσια αυτής της εργασίας και συνεπώς δεν διερευνώνται περαιτέρω

### 2.3.2 Σύνοψη & Κρίσιμα Μέρη

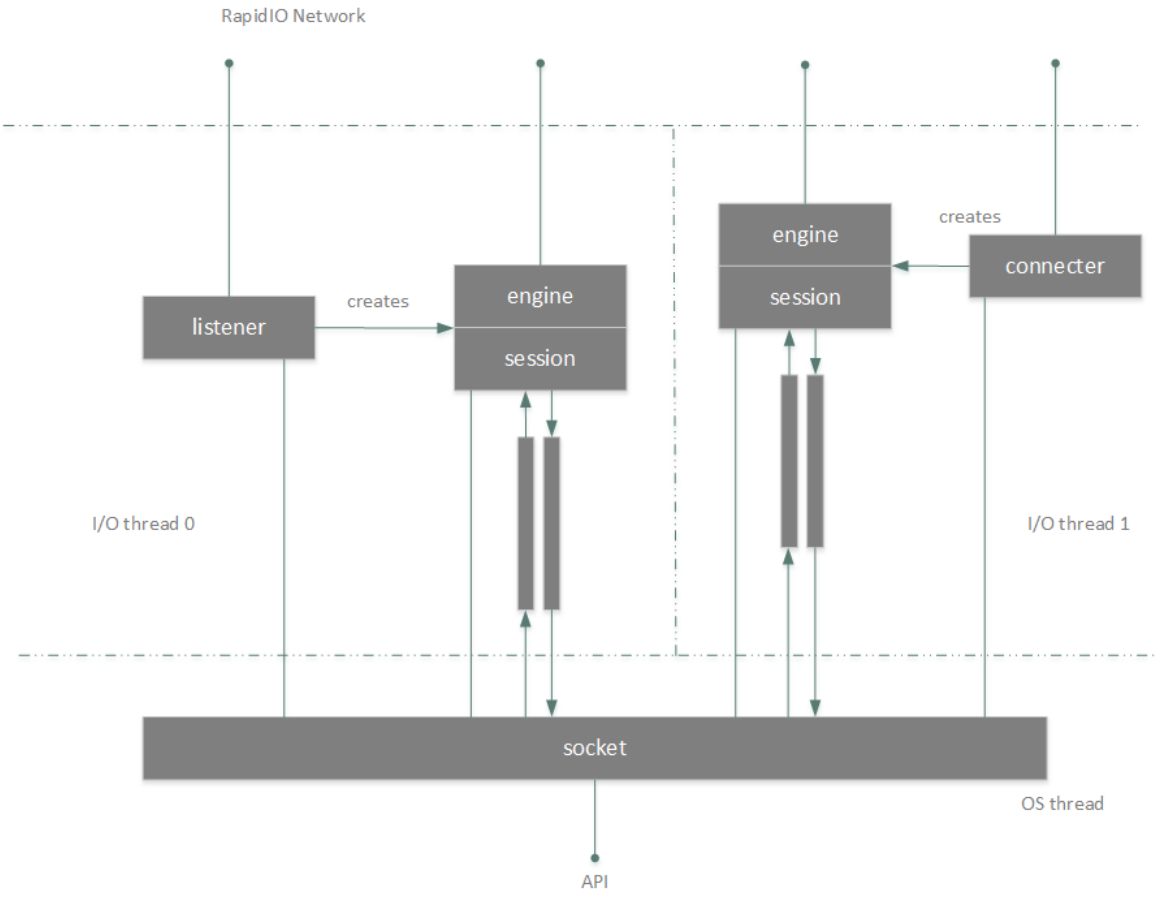
Το context είναι η πρώτη επαφή του χρήστη με το ZeroMQ. Χρειάζεται να δημιουργηθεί προκειμένου να δημιουργήσει και να συσχετίσει με αυτό το socket, ή sockets που πρόκειται να χρησιμοποιηθούν.

Το αντικείμενο του socket ζει μέσα σε ένα OS thread. Ανάλογα με τον ρόλο του socket, αυτό θα δημιουργήσει ένα παιδί το οποίο θα είναι είτε listener, είτε connecter. Ένας listener είναι αποτέλεσμα μιας κλήσης `bind()`, ενώ ένας connecter είναι αποτέλεσμα μιας κλήσης `connect()`.

Ο listener περιμένει αιτήσεις σύνδεσης στη συσχετισμένη διεπαφή/διεύθυνση. Στην περίπτωση που μια αίτηση είναι επιτυχής, ο listener θα δημιουργήσει ένα αντικείμενο session, το οποίο με την σειρά του θα δημιουργήσει ένα αντικείμενο engine. Η ροή πληροφορίας ανάμεσα σε socket και session είναι δυνατή μέσω pipes. Το engine, είναι το αντικείμενο που διαχειρίζεται την επικοινωνία μέσω του δικτύου. Session και engine ζουν μέσα σε ένα I/O thread, και συνεπώς είναι υποκλάσεις του `io_object_t`.

Με ανάλογο τρόπο, ο connecter αιτείται σύνδεσης σε μία διεπαφή/διεύθυνση, η οποία, αν είναι επιτυχημένη, θα έχει ως αποτέλεσμα την δημιουργία ενός ζεύγους session/engine.

Τα παραπάνω συνοψίζονται στην εικόνα 2.11.




Σχήμα 2.11: Internal Architecture

# Κεφάλαιο 3

# Σχεδιασμός και Υλοποίηση της επέκτασης του ZeroMQ με το RapidIO

## 3.1 Εισαγωγή

Η επέκταση του ZeroMQ πάνω από RapidIO, εισάγει ένα νέο επίπεδο μεταφοράς στη βιβλιοθήκη μηνυμάτων. Ο σκοπός είναι να παρουσιαστεί ένα Proof of Concept, μία αφαιρετική διεπαφή γύρω από ένα δίκτυο διασύνδεσης με δυνατότητα RDMA, η οποία θα επιτρέπει τη χρήση μιας διεπαφής τύπου socket.

Αυτό επιτρέπει στους χρήστες ZeroMQ να επωφεληθούν από τα πλεονεκτήματα που έχει να προσφέρει το RapidIO με έναν εύκολο τρόπο. Οι υπάρχοντες χρήστες του ZeroMQ πρέπει μόνο να αλλάξουν τη διεύθυνση στα προγράμματά τους, προκειμένου να χρησιμοποιήσουν ένα διαφορετική transport. Ο χρήστης δεν χρειάζεται να είναι εξοικειωμένος με τις διάφορες έννοιες γύρω από το RDMA ούτε με το πρωτόκολλο RapidIO. Αυτό εξαλείφει την ανάγκη για προσπάθειες υλοποίησης, επιτρέποντας τη χρήση οποιωνδήποτε υπάρχοντων δομών, στα πλαίσια υπολογιστικών εγκαταστάσεων και εφαρμογών, καθώς και την ομαλή εναλλαγή μεταξύ των πιθανών στρωμάτων μεταφοράς.

## 3.2 Επέκταση της Αρχιτεκτονικής

Προκειμένου να επεκτείνουμε το RapidIO ώστε να χρησιμοποιεί το RapidIO transport, πρέπει να υλοποιηθούν ορισμένες κλάσεις αποκλειστικά για το RapidIO, επιπρόσθετα με κάποιες μικρές αλλαγές σε βασικές κλάσεις του πυρήνα της βιβλιοθήκης.

Παρακάτω, παρατίθεται μια λίστα με τις κλάσεις που χρειάζονται προσθήκες, καθώς και μια περιγραφή των απαραίτητων ενεργειών.

#### Διευθυνσιοδότηση

Η κλάση του ZeroMQ για διευθυνσιοδότηση, χρειάζεται να ξέρει πότε μία διεύθυνση ενδέχεται να αντιστοιχεί σε RapidIO, και συνεπώς να αναλυθεί με διαφορετικό τρόπο. Αυτό επιτρέπει στην κατάλληλη κλάση να εκτελέσει την ανάλυση, σε αυτή την περίπτωση, την κλάση του RapidIO.

#### Socket

Η κλάση των ZeroMQ socket ονομάζεται `socket_base`. Ορισμένες συναρτήσεις υλοποιούνται από τις κλάσεις των διαφόρων τύπων socket. Ωστόσο, άλλες συναρτήσεις όπως η `bind()` και η `connect()` μοιράζονται την ίδια βασική υλοποίηση για διαφορετικούς τύπους ZeroMQ socket αλλά και transport.

Για το bind, η βασική κλάση των socket, θα ελέγξει ποιος είναι ο τύπος του transport, και θα δημιουργήσει το σχετικό listener object, περνώντας τη διεύθυνση ως όρισμα. Από το σημείο αυτό και μετά, ο listener, που είναι πλέον transport-specific, είναι υπεύθυνος για την υπόλοιπη διαδικασία.

Για την δημιουργία μιας σύνδεσης, η κλάση socket θα αναλύσει σωστά τη διεύθυνση, πριν δημιουργήσει ένα αντικείμενο session. Αυτό το αντικείμενο θα είναι υπεύθυνο για τη νέα σύνδεση. Θα δημιουργήσει επίσης pipes ανάμεσα στο socket και το νέο session, σύμφωνα με τους περιορισμούς του εκάστοτε transport.

Για τις παραπάνω λειτουργίες χρειάστηκε να επεκταθεί η κλάση `socket_base` για να προστεθεί υποστήριξη για το RapidIO.

### Session

Το κομμάτι του Session που είναι transport-specific αφορά την εγκαθίδρυση της σύνδεσης. Η προσπάθεια ανοίγματος μια σύνδεσης γίνεται εμμέσως, όταν το αντικείμενο του session δημιουργείται από το socket. Το session είναι μετά υπεύθυνο για την δημιουργία του κατάλληλου connecter, το οποίο θα αναλάβει τα υπόλοιπα.

Αν η διεύθυνση που έχει περαστεί στο session από το socket ταιριάζεται με τη μορφή του RapidIO, τότε θα δημιουργήσει ένα `rio_connecter` αντικείμενο

Το κείμενο που ακολουθεί περιγράφει τις κλάσεις του RapidIO που υλοποιήθηκαν, προκειμένου να επιτραπεί στο ZeroMQ να χρησιμοποιήσει το RapidIO transport. Για την εκπλήρωση των απαιτήσεων που θέτει η αρχιτεκτονική του ZeroMQ, πρέπει να δημιουργηθούν οι παρακάτω κλάσεις:

- Μια κλάση διευθυνσιοδότησης
- Ένας listener
- Ένας connecter
- Μία μηχανή (engine)

Εκτός αυτών, δημιουργήθηκε και μία πέμπτη κλάση. Η λειτουργία του RapidIO mailbox περιγράφεται παρακάτω.

### RapidIO Address

Το ZeroMQ χρησιμοποιεί μια κλάση διευθυνσιοδότησης για την ανάλυση της διεύθυνσης, όπως αυτή δίνεται από την προγραμματιστική διεπαφή, σε μορφή συμβολοσειράς. Ας υποθέσουμε πως ο χρήστης θα χρησιμοποιήσει μία διεύθυνση του τύπου “tcp://192.168.1.2:4545”. Η συμβολοσειρά αυτή περιέχει τρία απαραίτητα αναγνωριστικά. Το πρώτο, είναι το αναγνωριστικό του πρωτοκόλλου, εν προκειμένω tcp. Αυτό δίνει την οδηγία στο ZeroMQ να χρησιμοποιήσει τις κατάλληλες κλάσεις για τις μετέπειτα ενέργειες και λειτουργίας. Το δεύτερο και τρίτο αναγνωριστικό είναι η διεύθυνση IP και η θύρα αντίστοιχα, που φυσικά μεταφράζονται σε ένα συγκεκριμένο άκρο στο σύστημα. Αυτό καλύπτεται από την κλάση `tcp_address` στο ZeroMQ.

Κατ' αναλογία, υλοποιήθηκε η κλάση `rio_address`. Για το RapidIO, δεν υπάρχει αυστηρό πρότυπο διευθυνσιοδότησης, όπως υπάρχει για το TCP/IP. Ένα άκρο ορίζεται ως ο συνδυασμός των `destination ID` και `channel`. Προκειμένου να διατηρηθεί μια ομοιότητα με το TCP/IP, μία διεύθυνση RapidIO στα πλαίσια του ZeroMQ, αναμένεται να έχει την ακόλουθη μορφή: rio://[destination id]:[port]. Την συμβολοσειρά αυτή θα επεξεργαστεί η κλάση `rio_address`, η οποία θα αναλύσει την διεύθυνση στα αντίστοιχα αναγνωριστικά. Από εκεί και πέρα, άλλα μέρη του ZeroMQ θα χρησιμοποιούν αυτή την κλάση για να πάρουν τη σχετική πληροφορία.

### RapidIO Mailbox

Το ZeroMQ έχει κλάσεις `mailbox`, όπως περιγράφεται στο κεφάλαιο **??**. Η κλάση `rioh_mailbox` δεν σχετίζεται με αυτές. Ο σκοπός της είναι να προσομοιώσει file descriptor events, τα οποία συνήθως εμφανίζονται ως αποτέλεσμα σε ενέργειες πάνω σε sockets. Προκειμένου να υλοποιηθεί η προσομοίωση αυτή, το `rioh_mailbox` δημιουργεί έναν τεχνητό περιγραφητή αρχείου, ο οποίος συσχετίζεται με το εκάστοτε αντικείμενο. Τέτοια αντικείμενα μπορεί να είναι ο listener, ο connecter ή το engine, με άλλα λόγια οτιδήποτε έχει να κάνει με ανοιχτή σύνδεση. Ένα event στον περιγραφητή αρχείου προκύπτει ως το αποτέλεσμα της λήψης ενός doorbell, το οποίο μεταφέρει μια τιμή που είναι αποκλειστικά συσχετισμένη με τον συγκεκριμένο περιγραφητή. Το προσομοιωμένο event στον περιγραφητή αρχείου θα πιαστεί μετά από οποιονδήποτε το περίμενε. Προκειμένου να είναι αυτό το σχέδιο αποτελεσματικό, το `rioh_mailbox` πρέπει να έχει τον τεχνητό περιγραφητή αρχεία αλλά και το destination ID του συσχετισμένου άκρου. Για την διαχείριση των doorbells, χρησιμοποιεί ένα νήμα το οποίο τρέχει στο προσκήνιο, προκειμένου να πιάσει και να επεξεργαστεί εισερχόμενα doorbells. Η κλάση προσφέρει επίσης συναρτήσει για να στείλει doorbells σε απομακρυσμένα `rioh_mailboxes`.

Περισσότερες λεπτομέρειες και μια εικόνα που βοηθάει στην κατανόηση των παραπάνω μπορούν να βρεθούν στο 3.3.

### RapidIO Listener

Η κλάση του listener στο ZeroMQ είναι εκείνη που είναι υπεύθυνη για την αναμονή εισερχόμενων αιτημάτων σύνδεσης, την έγκρισή τους και τη δημιουργία ενός ζεύγους session/engine για κάθε νέα σύνδεση. Ο listener θα δημιουργήσει αρχικά RapidIO socket και μετά θα κάνει bind αυτού στην σχετική διεύθυνση. Το πρώτο γεγονός που θα πραγματοποιηθεί θα είναι η άφιξη ενός doorbell, το οποίο θα χειριστεί το `rioh_mailbox` του listener, προκαλώντας ένα file descriptor event στο fd που είναι συσχετισμένο με τον listener, που θα καλέσει με την σειρά του ένα `in_event`. Εκεί, θα γίνουν οι απαραίτητες προετοιμασίες για τη νέα σύνδεση πριν από την δημιουργία του ζεύγους session/engine.

Σε αυτό το σημείο, είναι απαραίτητο να τονίσουμε τα ακόλουθα σημεία. Το RapidIO socket που δημιουργείται εδώ υποστηρίζει μόνο Channelized Messages. Το υποσύστημα του RapidIO χρησιμοποιεί sockets σαν αυτό για να ανταλλάξει τις απαραίτητες πληροφορίες σύνδεσης και εμπορίας, όπως είναι οι διευθύνσεις για χρήση RDMA. Μετά από αυτό το είδος ανταλλαγής πληροφοριών ένα bound socket εξυπηρετεί μόνο τις μελλοντικές αιτήσεις σύνδεσης, επαναλαμβάνοντας την ίδια διαδικασία, έως ότου ο χρήστης ξεκινήσει την διαδικασία τερματισμού.

### RapidIO Connecter

Ο Connecter είναι υπεύθυνος να συνδεθεί στο απομακρυσμένο άκρο. Ως πρώτο βήμα, δημιουργεί το RapidIO socket, το οποίο θα χρησιμοποιηθεί για την νέα σύνδεση. Μετά, στέλνει ένα doorbell στον listener, προκειμένου να μπει στο `in_event` του, και να βρεθεί σε μια κατάσταση να δεχθεί το αίτημα σύνδεσης. Μετά από αυτό γίνεται η σχετική κλήση στην βιβλιοθήκη για να γίνει η σύνδεση. Κατόπιν, ο connecter θα επιστρέψει.

Όπως και πριν, τα sockets που αναφέρονται εδώ είναι για Channelized Messaging, που χρησιμοποιούνται αποκλειστικά για την εγκαθίδρυσης της σύνδεσης.

### RapidIO Engine

Το engine, είναι το μέρος του συστήματος που αποτελεί την διεπαφή με το δίκτυο RapidIO για τις διάφορες ανάγκες μεταφοράς δεδομένων. Διαχειρίζεται την αποστολή και λήψη δεδομένων, υποθέτοντας και τους δύο ρόλους, ανεξάρτητα με το αν η δημιουργία του ήταν αποτέλεσμα μιας ενέργειας σύνδεσης ή μιας ενέργειας αποδοχής.

Κατά την αποστολή, το engine θα πραγματοποιήσει ένα RDMA write, απευθείας στην μνήμη του απομακρυσμένου κόμβου. Αφού επιστρέψει η κλήση, το engine θα στείλει ένα doorbell στο απομακρυσμένο engine, προκειμένου να ενεργοποιήσει ένα file descriptor event που θα πιάσει ο απομακρυσμένος κόμβος Η διαδικασία αυτή λαμβάνει μέρος εντός της συνάρτησης `out_event` .

Προκειμένου να γίνει παραλαβή, το engine χρησιμοποιεί ένα νήμα ως μέρος του `rioh_mailbox`, το οποίο αναμένει συνεχώς την παραλαβή ενός doorbell, προκειμένου να προκαλέσει ένα file descriptor event. Αυτό το event θα έχει ως αποτέλεσμα την κλήση της συνάρτησης `in_event`, όπου το engine θα διαβάσει τα νέα δεδομένα.

Οι παραπάνω λειτουργίες αποστολής/λήψης χρειάζεται να λάβουν μέρος με την βοήθεια αυστηρού ελέγχου, ώστε να αποφευχθεί η πιθανότητα για memory corruption.

### Βοηθητικές Συναρτήσεις

Ορισμένες λειτουργίες και δομές δεδομένων που είναι κοινές για ορισμένες ή όλες τις παραπάνω κατηγορίες συγκεντρώνονται σε ένα μόνο αρχείο επικεφαλίδων. Το αρχείο rio.hpp περιλαμβάνει λειτουργίες που υποστηρίζουν τις απαραίτητες ενέργειες για την επιτυχή εγκαθίδρυση μιας νέας σύνδεσης RapidIO τόσο για τις πλευρές του listener, όσο και για του connecter. Αυτές οι ενέργειες περιλαμβάνουν τα εξής: Η αρχικοποίηση του Channelized Messaging socket, η δέσμευση RDMA-enabled μνήμης και η ανταλλαγή των διευθύνσεων inbound και outbound διευθύνσεων. Το αρχείο παρέχει επίσης λειτουργίες για την δέσμευση και την κατάλληλη κατανομή του εύρους των doorbell. Σε αυτό το αρχείο ορίζεται επίσης μια δομή που περιέχει τις απα-

ραίτητες πληροφορίες που σχετίζονται με μια σύνδεση RapidIO. Αυτή είναι η δομή που περιλαμβάνει κρίσιμες πληροφορίες σχετικά με τη νέα σύνδεση, η οποία μεταφέρεται από τον listener/connecter στο engine. Οι κατάλληλες λειτουργίες τερματισμού περιλαμβάνονται επίσης σε αυτό το αρχείου. Τέλος, εδώ ορίζονται διάφορες καθολικές τιμές για το RapidIO, όπως η διεύθυνση της δεσμευμένης μνήμης RDMA, το μέγεθός της, το μέγεθος ενός κελιού RDMA και το μήκος του κυκλικού buffer.

## 3.3 Λεπτομέρειες Υλοποίησης

### Προσομοίωση File Descriptor Events

Η μεγαλύτερη πρόκληση στη επέκταση του ZeroMQ για χρήση του RapidIO ήταν η εγγενής ανάγκη του πρώτου για την χρήση περιγραφητών αρχείου. Το ZeroMQ συνδέεται στενά με file descriptor events και τα χρησιμοποιεί για να προκαλέσει σχεδόν κάθε είδους ενέργεια. Αυτή η ανάγκη πηγάζει από τη χρήση των sockets για την αναπαράσταση οποιουδήποτε άκρου σε συνδέσεις TCP/IP. Δεδομένου ότι τα sockets αντιμετωπίζονται ουσιαστικά ως ανοικτά αρχεία, ο περιγραφητής του αρχείου που τις περιγράφει είναι εμφανής κατά μήκος του κώδικα του ZeroMQ.

Κατ' αρχήν, τα δίκτυα διασύνδεσης δεν προσφέρουν, υποστηρίζουν και ούτε χρειάζεται μια διεπαφή σαν τα sockets και κατ' επέκταση δεν κάνουν χρήση περιγραφητών αρχείου. Αυτό αποτελεί και ένα από τα κυρίως προβλήματα στην χρήση τους σήμερα. Οι υπάρχουσες εφαρμογές δεν ενδείκνυνται για την εισαγωγή ενός νέου στρώματος μεταφοράς. Το ίδιο ισχύει και για το RapidIO. Καθώς μια διεπαφή τύπου socket δεν προσφέρεται, προκύπτει η ανάγκη να προσομοιώσουμε την χρήση περιγραφητών αρχείων σε περίπτωση που θέλουμε να χρησιμοποιήσουμε το πρωτόκολλο σε ένα προτυποποιημένο περιβάλλον.

Στο πλαίσια της παρούσας επέκτασης, αυτή η επιπλοκή αντιμετωπίζεται με τον ακόλουθο τρόπο. Ο πυρήνας του linux προσφέρει έναν τρόπο δημιουργίας ενός "τεχνητού" περιγραφητή αρχείου που μπορεί να χρησιμοποιηθεί ως μηχανισμός αναμονής/ειδοποίησης μέσω της κλήσης eventfd (2). Το `eventfd()`

επιστρέφει έναν περιγραφητή αρχείου επί του οποίου μπορούν να εκτελεστούν οι πράξεις εγγραφής (write(2)) και ανάγνωσης (read(2)), που προκαλούν την αύξηση και μείωση αντίστοιχα έναν μετρητή του αντικειμένου eventfd ο οποίος διατηρείται από τον πυρήνα. Μέσω αυτού του μετρητή ο αριθμός των file descriptor events μπορεί να παρακολουθηθεί δημιουργώντας αποτελεσματικά έναν εικονικό περιγραφητή αρχείων, πάνω στον οποίο μπορούμε να έχουμε πλήρη έλεγχο. Αυτό μας επιτρέπει να τον εκμεταλλευτούμε και να τον χρησιμοποιούμε εναλλακτικά ως "socket interface" για τις ανάγκες μας.

Συνεπώς, προκειμένου να προσομοιωθεί ένα file descriptor event, μπορούμε απλά να στείλουμε ένα doorbell στο εκάστοτε άκρο, το οποίο θα τρέχει ήδη ένα νήμα για να διαχειρίζεται εισερχόμενα doorbells. Εκεί, το νήμα θα πιάσει το εισερχόμενο doorbell, θα διαβάσει την τιμή που μεταφέρει και θα εκτελεστεί ένα write(2) στον συσχετισμένο περιγραφητή αρχείου. Αυτή η ενέργεια θα ενεργοποιήσει ένα file descriptor event, το οποίο εν συνεχεία θα πιαστεί από τον υπεύθυνο poller (poll (2)). Στη συνέχεια ο poller θα κάνει την κλήση στην `out_event`, σε συμφωνία με την γενικότερη λογική του codebase του ZeroMQ.

### Διαχείριση των Doorbells

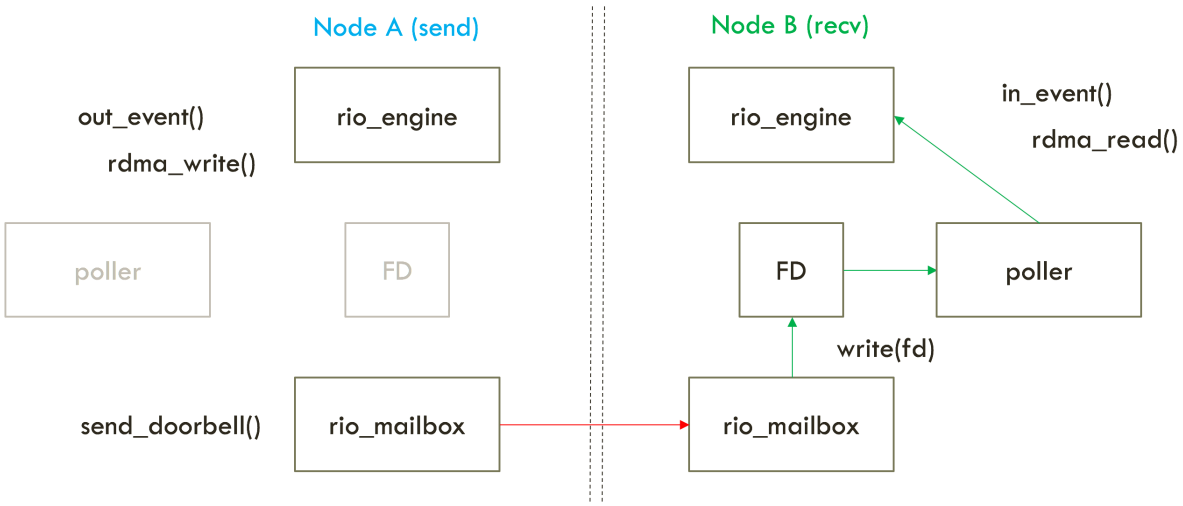
Προκειμένου να είναι επιτυχές το μοντέλο που περιγράφηκε παραπάνω, πρέπει οι λειτουργίες σχετικές με τα doorbells να υλοποιηθούν ως μέρος της κλάσης του RapidIO mailbox. Καθένας εκ των connecter, listener και engine, πρέπει να αρχικοποιήσουν ένα δικό του RapidIO Mailbox κατά την δημιουργία τους. Το νέο mailbox, θα δημιουργήσει έναν περιγραφητή αρχείου με μια κλήση eventfd(2), ο οποίος θα συσχετιστεί με την κλάση που δημιούργησε το mailbox. Επίσης, το `rioh_mailbox` θα ξεκινήσει ένα νήμα, το οποίο θα τρέχει στο παρασκήνιο, υπεύθυνο για την διαχείριση των εισερχόμενων doorbell, κάνοντας τις αντίστοιχες ενέργειες. Για κάθε doorbell θα ελεγχθεί η προέλευση και ο προορισμός της. Αν κάποιο από αυτά τα δύο πεδία δεν είναι τα αναμενόμενα, θα αγνοηθεί.

Όταν το `rioh_mailbox` του listener δεχθεί ένα doorbell, θα εκτελέσει ένα write(2) στον περιγραφητή αρχείου, γεγονός που θα προκαλέσει με την σειρά του την

κλήση της `in_event()` του `rio_listener`. Στα πλαίσια αυτής της συνάρτησης, θα λάβει μέρος μια κλήση `accept()` για να δεχθεί το αίτημα νέας σύνδεσης.

Το `rio_connecter` δεν αναμένεται ποτέ να δεχθεί ένα doorbell. Το `rioh_mailbox` του connecter εξυπηρετεί μόνο την αποστολή ενός doorbell, όταν αιτείται νέας σύνδεσης. Δεν δημιουργείται νήμα που ακούει για εισερχόμενα doorbells και δεν δεσμεύει το σχετικό εύρος.

Η χρήση των doorbells στα πλαίσια του engine είναι δεμένη με την λογική της διαχείρισης της κοινής μνήμης και περιγράφεται σε επόμενη παράγραφο.




Σχήμα 3.1: Mailbox functions overview

### Κατανομή Εύρους των Doorbell

Τα doorbell events μεταφέρουν την ακόλουθη πληροφορία: Το destination ID του κόμβου προέλευσης, το destination ID του κόμβου προορισμού και ένα πεδίο `uint64_t`. Καθώς διαφορετικά μέρη του ίδιου άκρου περιμένουν doorbells, είναι σημαντικό να υπάρχει ένα μηχανισμός με βάση των οποίων να γίνεται η διαφοροποίησή τους. Στις επόμενες παραγράφους περιγράφεται ο μηχανισμός σύμφωνα με τον οποίο αντιστοιχίζεται σε κάθε άκρο ένα διαφορετικό εύρος τιμών.

Ο listener ενδιαφέρεται μόνο για ένα doorbell που έχει το σωστό destination ID (το δικό του) και μεταφέρει την τιμή `RIO_ACCEPT`. Η `RIO_ACCEPT` ορίζεται ως "μαγικός αριθμός" στο αρχείο `rio.hpp`. Ένα doorbell με την ίδια τιμή δεν θα

έπρεπε να εμφανιστεί ως μέρος κανενός άλλου σεναρίου, εκτός της προσπάθειας εγκαθίδρυσης νέας σύνδεσης. Ο connecter ακολουθεί τον ίδιο κανόνα, αλλά περιμένει την τιμή `RIO_CONNECT`, η οποία επίσης ορίζεται στο `rio.hpp`. Στην παρούσα υλοποίηση, οι δύο αυτοί αριθμοί αποτελούν την αρχή του επιτρεπτού εύρους, δηλαδή τις τιμές 0 και 1.

Για το engine, τα πράγματα είναι πιο περίπλοκα. Μία εκτέλεση του ZeroMQ πρέπει να μπορεί να διατηρεί ανοιχτές παράλληλες συνδέσεις ταυτόχρονα. Κάθε σύνδεση πρέπει να επιτρέπει την μεταφορά δεδομένων ακόμα και αν άλλες συνδέσεις είναι ενεργές. Αυτό το κομμάτι είναι αναπόσπαστο μέρος της λογικής διαχείρισης μνήμης. Η δεσμευμένη μνήμη για RDMA αρχικά διαιρείται σε τόσα μέρη όσα και και το πλήθος των πιθανών συνδέσεων. Με άλλα λόγια, αν το σύστημα αποτελείται από 4 κόμβους, ένας κόμβος, ο Α θα πρέπει να έχει χωρίσει την "εισερχόμενη" μνήμη του σε τρία, ίσα μέρη, για τους κόμβους Β, Γ, και Δ. Τα τρία αυτά μέρη θα είναι μέρος της δομής του κυκλικού buffer, η χρήση του οποίο επίσης εξηγείται στην επόμενη παράγραφο.

Προκειμένου να υποστηριχθεί ο μηχανισμός που περιγράφεται παραπάνω, το εύρος των doorbell απεικονίζεται σε θέσεις μνήμης κατά τον ακόλουθο τρόπο. Για τον κόμβο Α, η πρώτη σύνδεση που εγκαθιδρύεται θα αναφέρεται πάντα στις τρεις πρώτες θέσεις μνήμης. Το engine που θα δημιουργηθεί για να διαχειρίζεται αυτή τη σύνδεση, θα αγνοήσει οποιαδήποτε doorbells, των οποίων οι τιμές δεν είναι συσχετισμένες με τις θέσεις σε αυτή την περιοχή της μνήμης. Η ίδιο λογικής ισχύει και για μελλοντικές συνδέσεις και μετέπειτα αντικείμενα engine.

Οι τιμές των doorbell, υπολογίζονται ως εξής. Οι αρχικές τιμές 0 και 1, οι οποίες είναι δεσμευμένες για τον listener και τον connecter, παρακάμπτονται. Το εύρος των doorbells, για την πρώτη σύνδεση θα ξεκινήσει στο 2. Θα τελειώσει σε μια τιμή, ίση με τον συνολικό αριθμό θέσεων του circular buffer, διαιρούμενη με τον αναμενόμενο αριθμό συνδέσεων. Αν εφαρμόζουμε την ίδια λογική, κάθε επόμενο εύρος doorbell μπορεί να υπολογιστεί ως εξής.

$$range\_start = prev\_range\_end$$
$$range\_end = prev\_range\_end + cbuf\_length/num\_conns$$
$$prev\_range\_end = range\_end$$

Όπου το $cbuf\_length$ είναι το μήκος του circular buffer και το $num\_conns$ είναι ο μέγιστος δυνατός αριθμός συνδέσεων. Για το πρώτο εύρος, η αρχική τιμή $prev\_range\_end$ είναι 2, δηλαδή `RIO_CONNECT`+1. Τελικά το δεσμευμένο εύρος θα είναι [$range\_start$, $range\_end$).

Προκειμένου να γίνουν τα παραπάνω πιο κατανοητά, ας υποθέσουμε πως έχουμε τις παρακάτω τιμές:

| num_conns | 8 |
|---|---|
| cbuf_length | 16 |

Σε αυτή την περίπτωση η κατανομή του εύρους των doorbells θα ήταν η ακόλουθη:

| RIO_ACCEPT | [0] |
|---|---|
| RIO_CONNECT | [1] |
| CONNECTION_0 | [2-18) |
| CONNECTION_1 | [18-34) |
| CONNECTION_2 | [34-49) |
| CONNECTION_3 | [49-65) |
| CONNECTION_4 | [65-81) |
| CONNECTION_5 | [81-97) |
| CONNECTION_6 | [97-113) |
| CONNECTION_7 | [113-129) |

### Διαχείριση Μηνυμάτων και Μνήμης

Όταν ένας χρήσης του ZeroMQ θέλει να κάνει μια συναλλαγή, τα σχετικά δεδομένα θα περαστούν ως έχουν σε μια κλήση `zmq_send()` του API. Ο χρήστης θα στέλνει μια παρτίδα δεδομένων, περιμένοντας ακριβώς την ίδια παρτίδα δεδομένων να είναι διαθέσιμη στο απομακρυσμένο άκρο. Για αυτό το μέρος του κειμένου, η παρτίδα αυτή θα αναφέρεται ως "το μήνυμα".

Όταν το RapidIO πρόκειται να κάνει μια μεταφορά δεδομένων, το ενδιαφέρουν μόνο τα bytes που είναι να σταλούν, και πως να τα μεταφέρει στο σωστό σημείο της απομακρυσμένης μνήμης. Είναι κοινό για το μέγεθος του μηνύματος να είναι μεγαλύτερο από την διαθέσιμη RDMA μνήμη. Έτσι, προκύπτει η ανάγκη να σπάσουμε το αρχικό μήνυμα σε μικρότερα μέρη, ώστε

να φέρουμε την συναλλαγή εις πέρας. Είναι επίσης πιθανό να συνυπάρχουν πολλές ενεργές συνδέσεις κάποια στιγμή στον χρόνο, κρίνοντας απαραίτητο να υπάρχει αποκλειστική μνήμη για κάθε σύνδεση. Συνεπώς, είναι αναγκαίο να υλοποιηθεί ένας μηχανισμός που επιτρέπει τέτοια σενάρια.

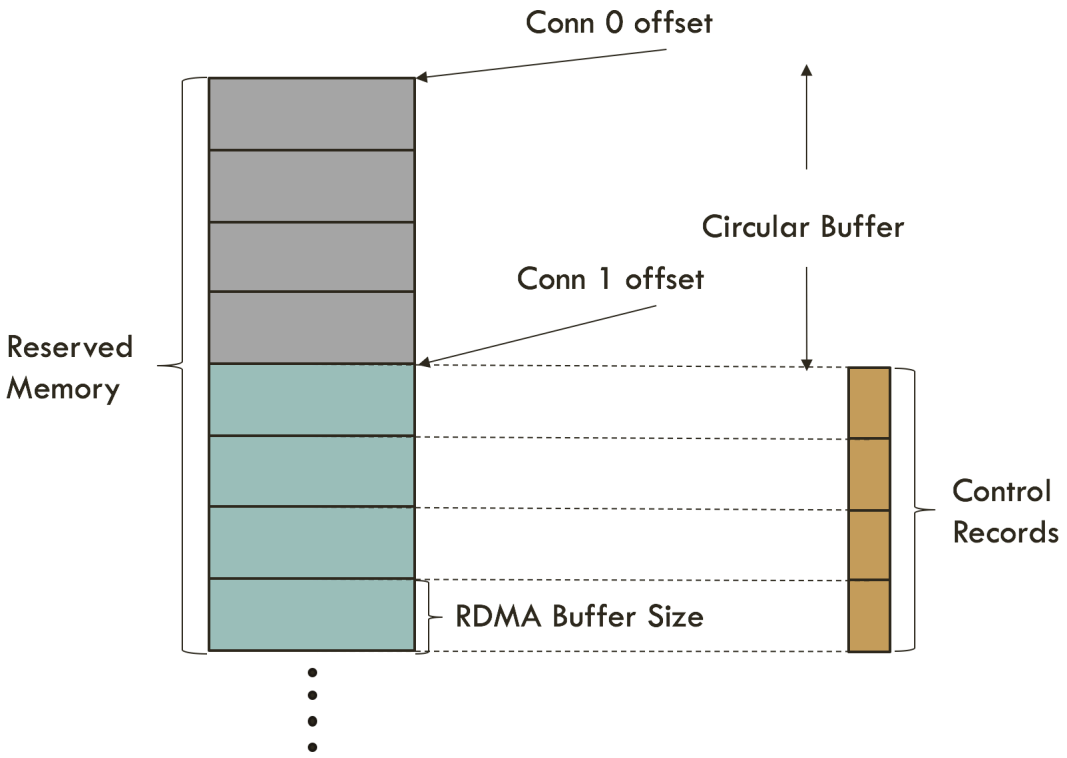
Ξεκινάμε με δύο κόμβους RapidIO, οι οποίοι έχουν το ίδιο μέγεθος μνήμης RDMA. Όταν εγκαθιδρύεται η σύνδεση, τα δύο άκρα ανταλλάσσουν inbound memory addresses, ώστε κάθε άκρο να ξέρει που να γράψει. Η μνήμη RDMA είναι χωρισμένη σε έναν αυθαίρετο αριθμό από RDMA buffers. Κάθε λειτουργία `send/recv` εκτελείται για τόσα bytes, όσο το μέγεθος ενός RDMA buffer. Κάθε engine, διατηρεί δύο πίνακες από flags. Ο πρώτος, αφορά το αν η παρούσα θέση στην εισερχόμενη μνήμη διατηρεί δεδομένα τα οποία έχουν διαβαστεί ή όχι. Ο δεύτερος, αφορά το αν η παρούσα θέση στην εξερχόμενη μνήμη είναι διαθέσιμη για γράψιμο. Το engine έχει επίσης δύο δείκτες, έναν για κάθε πίνακα. Ας εξετάσουμε τρία ενδεχόμενα.

Πρώτον, ας υποθέσουμε ότι το μήνυμα που πρέπει να μεταδοθεί είναι μικρότερο ή ίσο με το μέγεθος του RDMA buffer. Σε αυτήν την περίπτωση, ο αποστολέας θα ελέγξει εάν η τρέχουσα θέση του εξερχόμενου δείκτη είναι διαθέσιμη για εγγραφή. Αν είναι, ο αποστολέας εκτελεί μια εγγραφή RDMA, ενημερώνει τον εξερχόμενο πίνακα για να επισημάνει τη θέση ως μη διαθέσιμη για εγγραφή και στέλνει ένα doorbell στον δέκτη για να τον ενημερώσει για την ύπαρξη νέων δεδομένων. Μετά από αυτήν την ενέργεια, ο δέκτης θα «ξυπνήσει» με το doorbell, το οποίο θα μεταφέρει ως payload τον εξερχόμενο δείκτη του αποστολέα (δηλ. Τον εισερχόμενο δείκτη του δέκτη) και θα κάνει update το αντίστοιχο flag στην πίνακα εισερχόμενης μνήμης. Ο δέκτης θα ελέγξει αν η θέση μνήμης έχει πράγματι είναι γραμμένη, και αν είναι, θα διαβάσει τα δεδομένα, προτού προωθήσει το μήνυμα στο RapidIO session, το οποίο αργότερα θα επιστρέψει μέσω της `zmq_recv()` του API. Σημειώνεται ότι σε αυτό το βήμα δεν γίνεται αντιγραφή μνήμης. Το στοιχείο συστήματος που περιμένει τα δεδομένα απλώς μεταφέρει τον δείκτη στην σχετική θέση μνήμης. Αμέσως μετά την ”ανάγνωση” των δεδομένων, ο δέκτης θα στείλει ένα doorbell πίσω στον αποστολέα, έτσι ώστε ο αποστολέας να ενημερώσει τον πίνακά του καθιστώντας διαθέσιμη τη θέση μνήμης μόλις ανάγνωσης για να γράψει ξανά. Ωστόσο, αυτό δεν συμβαίνει συγχρονισμένα. Κατ’ αυτόν τον τρόπο, ο αποστολές μπορεί να βάλει τα επόμενα δεδομένα κατά κάποιο

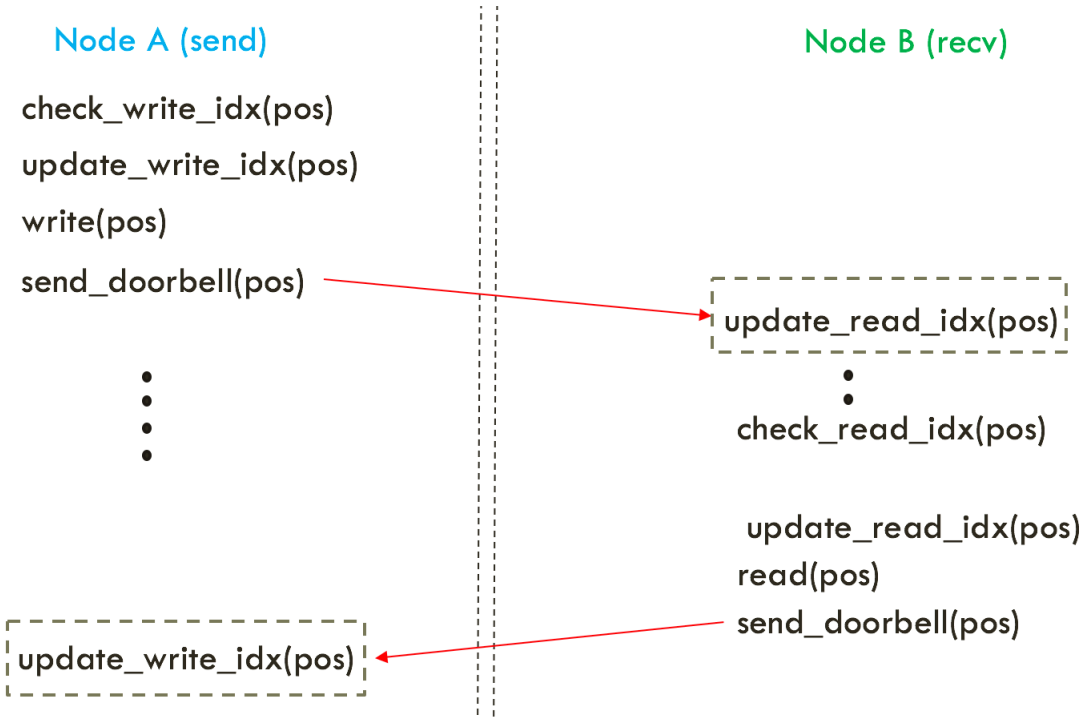
τρόπο σε μια ουρά, προτού επεξεργαστούν τα προηγούμενα.

Τώρα, ας υποθέσουμε ότι το μήνυμα που πρέπει να μεταδοθεί είναι μεγαλύτερο από το μέγεθος του RDMA buffer, αλλά όχι μεγαλύτερο από τη συνολική δεσμευμένη μνήμη RDMA. Σε αυτή την περίπτωση, το μήνυμα θα χωριστεί σε κομμάτια. Για κάθε κομμάτι θα επαναληφθεί η διαδικασία που περιγράφεται στο προηγούμενο παράδειγμα. Σε αυτό το σενάριο, ο ασύγχρονος σχεδιασμός του συστήματος θα αξιοποιηθεί και οι αποστολές των επόμενων κομματιών δεδομένων δεν θα περιορίζονται από την ταχύτητα ανάγνωσης. Σε κάποια χρονική στιγμή, πριν ολοκληρωθεί η συναλλαγή, όλα τα δεδομένα θα είναι παρόντα στον κόμβο λήψης και ο αποστολέας θα έχει επιστρέψει. Αυτό ευνοεί την απόδοση σε σύγκριση με ένα πλήρως συγχρονισμένο σχέδιο.

Τέλος, έχουμε την περίπτωση όπου το μήνυμα που πρέπει να μεταδοθεί είναι μεγαλύτερο από την δεσμευμένη μνήμη RDMA. Σε αυτή την περίπτωση, η ακολουθία των ενεργειών παραμένει η ίδια, αλλά σε κάποιο σημείο ο αποστολέας θα πρέπει να μπλοκάρει, σε περίπτωση που είναι πολύ πιο γρήγορος σε σύγκριση με τον δέκτη. Και τα δύο μέρη θα πρέπει να εκτελέσουν έναν πλήρη κύκλο πάνω στον circular buffer, ενδεχομένως και περισσότερο, ανάλογα με το μέγεθος του μηνύματος.




Σχήμα 3.2: Memory Scheme Overview




Σχήμα 3.3: Sequence for single RDMA Buffer Transfer

# Κεφάλαιο 4

# Αξιολόγηση

Το κεφάλαιο αυτό έχει σκοπό την αξιολόγηση της δουλειάς που παρουσιάστηκε στα προηγούμενα κεφάλαια αυτής της εργασίας. Αρχικά, εξηγούνται οι επιλογές των μετροπρογραμμάτων και οι διάφορες αναλύσεις, πριν παρουσιαστούν οι αντίστοιχες μέθοδοι. Στη συνέχεια παρουσιάζονται οι μετρήσεις και προσφέρεται μια ερμηνεία των αποτελεσμάτων.

## 4.1 Hardware & Software Setup

Το hardware setup που χρησιμοποιήθηκε για τους σκοπούς του benchmarking αποτελείται από τέσσερα 2U Quad units, κάθε ένα από τα οποία περιλαμβάνει 4 κόμβους, αθροίζοντας στους 16. Κάθε κόμβος έχει έναν Intel Xeon L5640, με το ρολόι του στα 2.27GHz και 48GB μνήμης. Σε κάθε κόμβο είναι κουμπωμένη μια IDT Tsi721 PCIe to RapidIO bridge card, με ονομαστική ταχύτητα στα 16GB/s. Οι κάρτες δικτύου (Network Interfaces Cards (NICs)) είναι συνδεδεμένες σε ένα 38-port Top of Rack (ToR) RapidIO Generation 2 switch. Κάθε port του switch είναι ρυθμισμένο στα 20Gb/s. Οι συνδέσεις γίνονται με QSFP+ καλώδια, ή Quad Small Form-factor Pluggable cables. Υποστηρίζουν τέσσερα κανάλια των 10Gb/s το καθένα, γεγονός το οποίο τα καθιστά παραπάνω από επαρκή για τις ανάγκες μας.

Το λογισμικό του RapidIO αποτελείται από την βιβλιοθήκη RRMAP, και τους οδηγούς πυρήνα του Linux.

## 4.2 Benchmarks & Μετρήσεις

Για το benchmarking, χρησιμοποιήθηκαν τα ZeroMQ performance tests. Τα τεστ περιλαμβάνουν τον client, ο οποίος θα είναι ο πρώτος που θα εκτελέσει ένα send, και τον server, που θα είναι ο πρώτος που θα δεχτεί ένα μήνυμα.

Πρώτα εκτελείται ο server. Θα αρχικοποιήσει το context του ZeroMQ και ένα socket τύπου `ZMQ_SERVER. Μετά, το socket θα γίνει bound στη RapidIO διεύθυνση που αποτελείται από το destination ID του server και από το κανάλι στο οποίο θα δέχεται Channelized Messages. Τότε, αρχικοποιείται ένα ZeroMQ message, προτού γίνει η κλήση για receive, η οποία θα μπλοκάρει μέχρις ότου γίνει η παραλαβή του μηνύματος, προκύψει κάποιο σφάλμα ή φτάσουμε σε timeout. Μετά, το μήνυμα θα σταλεί πίσω στον client. Η διαδικασία αυτή θα επαναληφθεί για όσες φορές οριστεί στα ορίσματα του προγράμματος. Τέλος, ο αποστολές θα κάνει clean up, κλείνοντας την δομή του message, το socket και το context. Δεν παίρνονται μετρήσεις στην πλευρά του server.

Ο client εκτελείται αφού το socket του server έχει γίνει bound. Ο client θα αρχικοποιήσει το context του ZeroMQ και θα δημιουργήσει ένα ZeroMQ socket τύπου ZMQ_CLIENT. Στην συνέχεια θα επιχειρήσει να συνδεθεί στην διεύθυνση που αντιστοιχεί στον server. Αν όλα πάνε καλά, η δομή του μηνύματος θα δημιουργηθεί, προτού ξεκινήσει έναν timer, και στείλει το μήνυμα στον server. Μετά την κλήση του send, ο αποστολέας θα περιμένει να παραλάβει ξανά το μήνυμα. Το ζευγάρι send/receive θα τρέξει όσες φορές, όσες έχει οριστεί στα ορίσματα του προγράμματος. Αφού τελειώσουν όλες οι επαναλήψεις, ο timer θα σταματήσει. Αυτή η διαδικασία μας δίνει στον client τον χρόνο που χρειάστηκε να στείλει και να παραλάβει ένα μήνυμα συγκεκριμένου μεγέθους για ορισμένο αριθμό φορών.

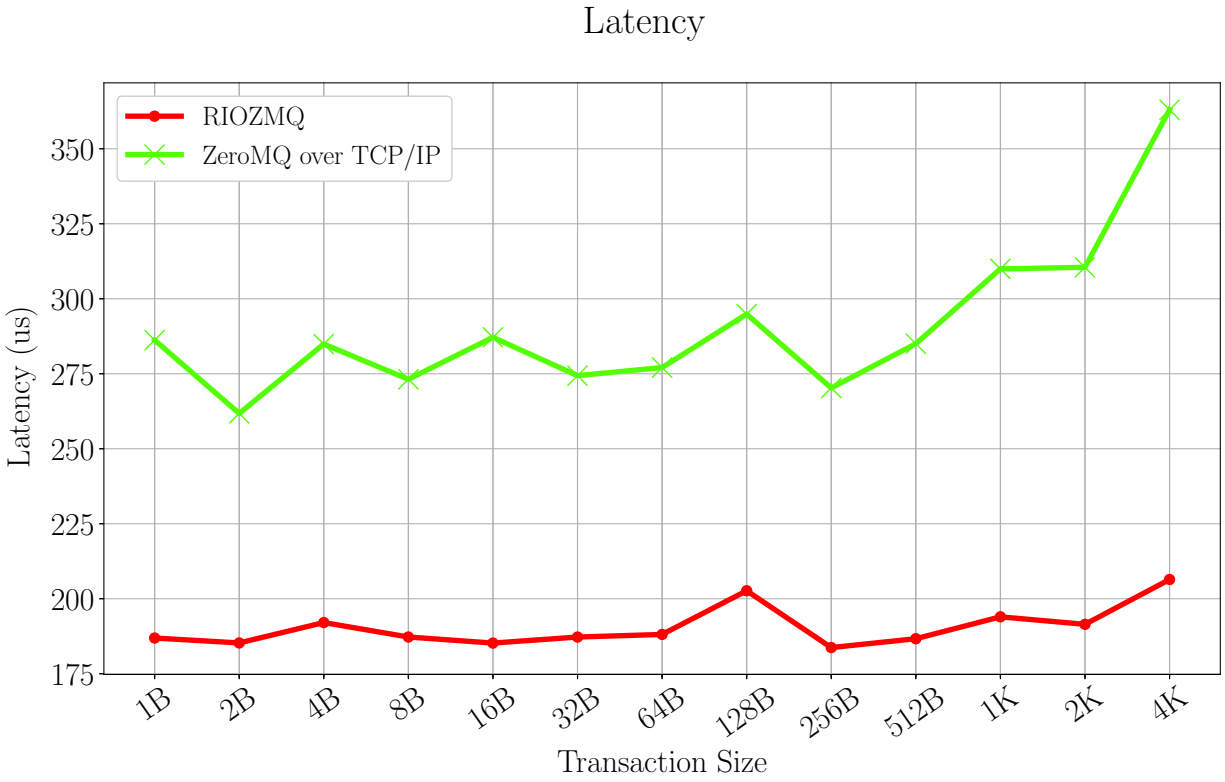
Αν ο χρόνος που πέρασε διαιρεθεί από τον αριθμό των επαναλήψεων και μετά διαιρεθεί με το δύο, έχουμε το latency του συστήματος, ή με άλλα λόγια, τον χρόνο που παίρνει στο σύστημα να στείλει ένα μήνυμα συγκεκριμένου μεγέθους από το ένα σημείο στο άλλο. Φυσικά, αφού γνωρίζουμε το latency, μπορούμε να υπολογίσουμε το throughput, διαιρώντας το μέγεθος του message με το latency.

Στις επόμενες ενότητες, η επίδοση αξιολογείται για διαφορετικές παραμέτρους του συστήματος.

### 4.2.1 Latency

Εδώ, αξιολογείται το Latency του συστήματος, σε σχέση με το message size.

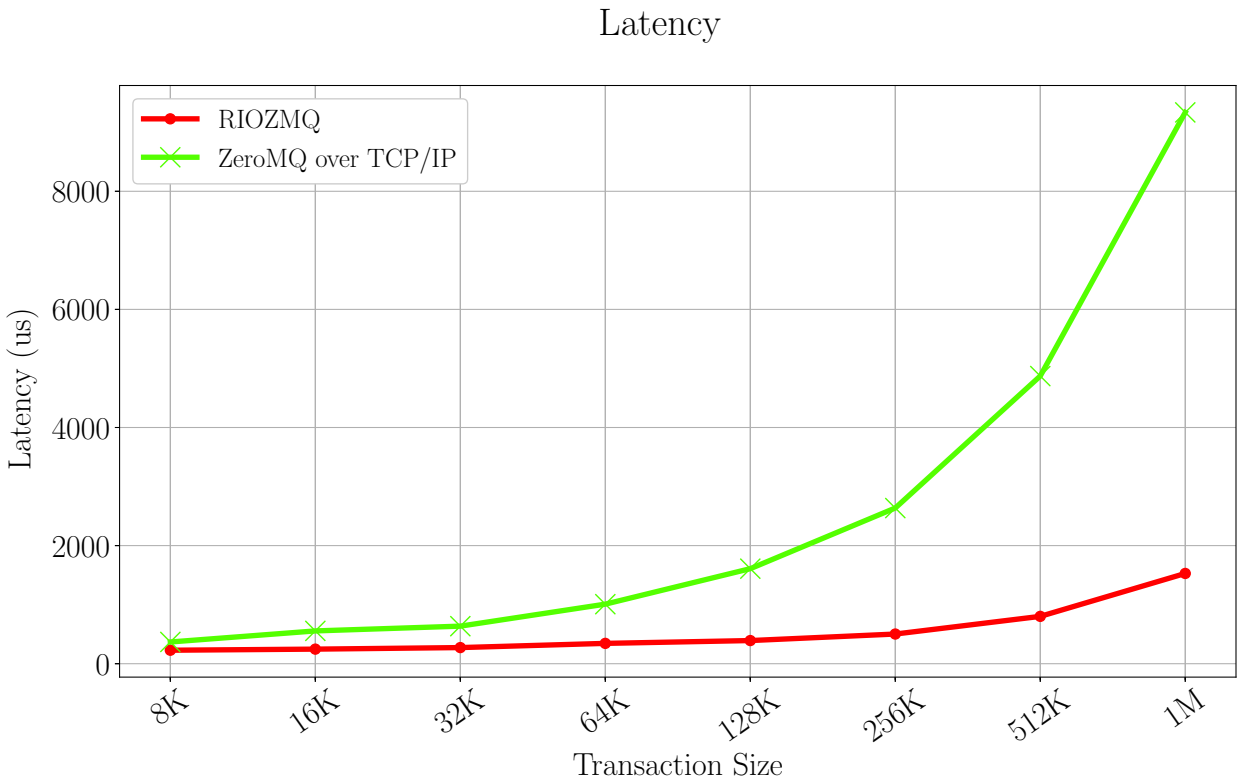
Τα αποτελέσματα παρουσιάζονται στις εικόνες που ακολουθούν. Οι γραφικές παραστάσεις περιλαμβάνουν επίσης μία γραμμή που αντιπροσωπεύει τις τιμές που πάρθηκαν για το ίδιο τεστ πάνω από TCP με 1GbE.




Σχήμα 4.1: Latency over RapidIO and TCP/IP [Small]

Το μετροπρόγραμμα έτρεξε 10000 φορές για κάθε μέγεθος συναλλαγής. Το RDMA buffer size για το RapidIO είχε τεθεί στα 4KB, το μικρότερο δυνατό, χρησιμοποιώντας μόνο μια θέση του circular buffer.

Όπως μπορούμε να δούμε, η επίδοση του συστήματος πάνω από RapidIO είναι καλύτερη σε σχέση με το TCP. Αυτό αναμενόταν, καθώς το πρωτόκολλο RapidIO μπορεί να πετύχει πολύ καλύτερο latency σε σύγκριση με το TCP.




Σχήμα 4.2: Latency over RapidIO and TCP/IP [Large]

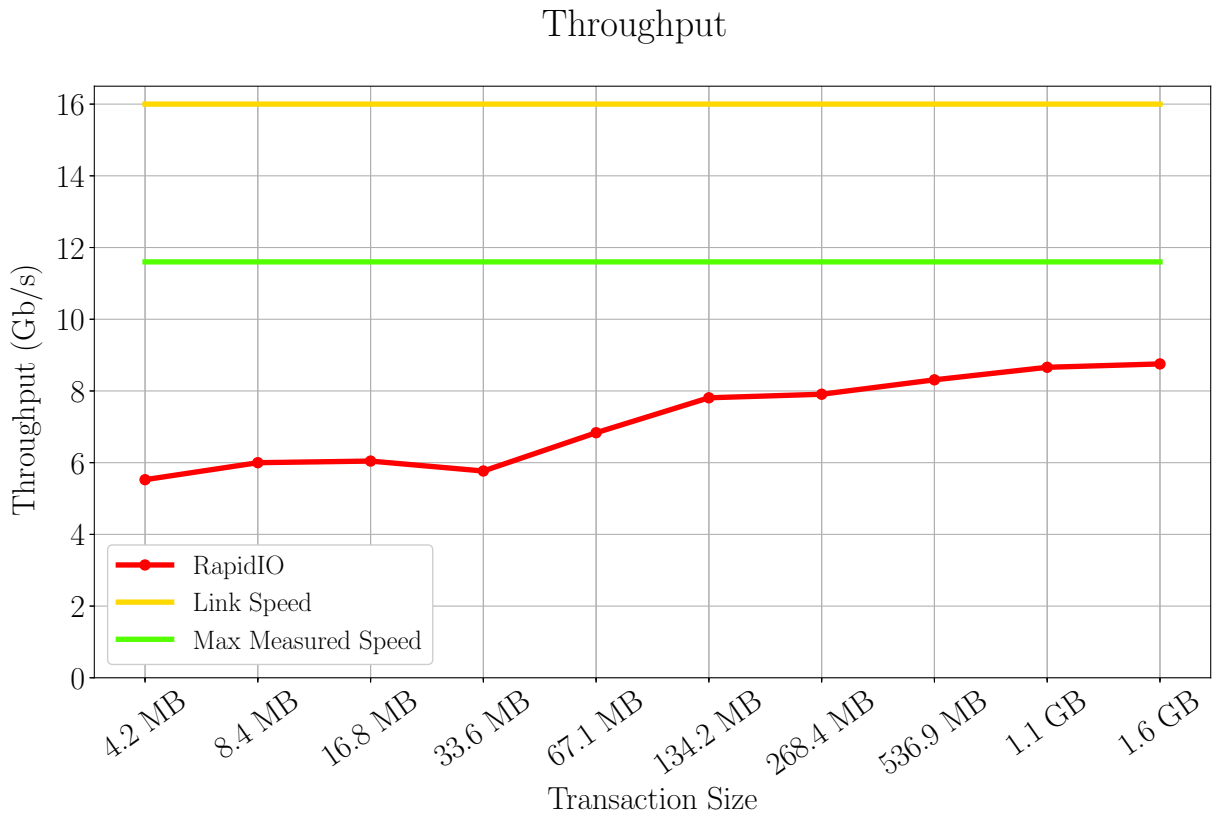
Αυτό οφείλεται κυρίως στη διαφορά του overhead μεταξύ των δύο μεθόδων. Το TCP εισάγει υψηλή πολυπλοκότητα, μέσω της ανάγκης του να διασχίσει τη στοίβα δικτύου, ακόμη και για την αποστολή ενός byte, που απαιτεί context changes και καταναλώνει κύκλους CPU. Το RapidIO από την άλλη πλευρά απλώς αντιγράφει το byte στην απομακρυσμένη μνήμη, παρακάμπτοντας τον επεξεργαστή του απομακρυσμένου υπολογιστή.

Επιπλέον, ο ρυθμός με τον οποίο αυξάνεται το latency σε σχέση με τα μεταφερόμενα byte είναι σημαντικά χαμηλότερος για την χρήση του RapidIO, πράγμα που μπορεί να σημαίνει και καλή συμπεριφορά σε κλίμακα.

### 4.2.2 Throughput

Στην επόμενο γράφημα, παρουσιάζεται το throughput του συστήματος σε σχέση με το message size. Η εικόνα αυτή αναφέρεται σε one-to-one σενάριο

Για κάθε συναλλαγή, εισάγεται ένα overhead ως μέρος της ενορχήστρωσης του




Σχήμα 4.3: ZeroMQ Throughput over RapidIO

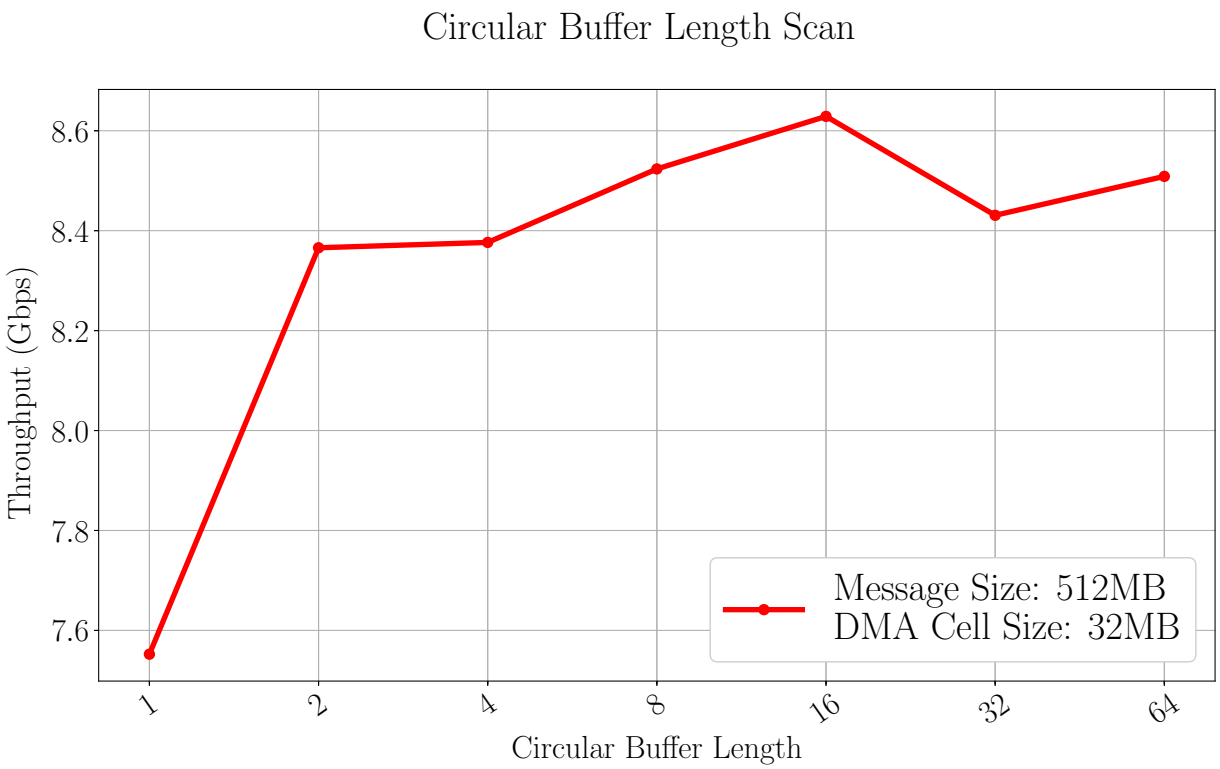
RDMA. Αυτό σημαίνει ότι όταν μια συναλλαγή είναι μικρή, οι επιδόσεις επηρεάζονται σημαντικά από γενικές εντολές. Ωστόσο, καθώς το μέγεθος αυξάνει το αντίκτυπο των διαφόρων overhead σταματάνε να είναι τόσο σημαντικά και το link saturation βελτιώνεται. Αυτό είναι επίσης αυτό που παρατηρείται στην παραπάνω γραφική παράσταση. Βλέπουμε ότι για αρκετά μεγάλες συναλλαγές (πάνω από 512MB), η απόδοση φτάνει σε ένα μέγιστο, περίπου στα 8,5Gbps. Αυτό είναι λίγο περισσότερο από το μισό της ονομαστικής ταχύτητας της σύνδεσης. Ωστόσο, όπως παρουσιάζεται στο κεφάλαιο 2, αφού υπολογιστεί το κόστος στην μετάφραση των πρωτοκόλλων του RapidIO και του PCIe, και αφού υπολογιστεί και το overhead της βιβλιοθήκης και του οδηγού, η μέγιστη μετρήσιμη ταχύτητα είναι περίπου στα 11.16Gbps. Αν αυτή η ταχύτητα θεωρηθεί ως η μέγιστη, το ZeroMQ over RapidIO, επιτυγχάνει περίπου 77% saturation.

μετά τη λογιστική μετάφραση από τις μεταφράσεις RapidIO και PCIe και τη ζύγιση των γενικών εξόδων του προγράμματος οδήγησης του πυρήνα και των βιβλιοθηκών, η αναμενόμενη πρακτική ταχύτητα θα πρέπει να είναι περίπου

11,1 Gbps. Αν αυτή η ταχύτητα θεωρηθεί ως μέγιστη, το ZeroMQ πάνω από το RapidIO μπορεί να επιτύχει περίπου 77

### 4.2.3 Circular Buffer Length

Το γράφημα 4.4 παρουσιάζει τη σάρωση της παραμέτρου που αφορά το μήκος του circular buffer, η οποία είναι μέρος της υλοποίησης της λογικής του RDMA για το RapidIO.




Σχήμα 4.4: RapidIO Circular Buffer Length Scan

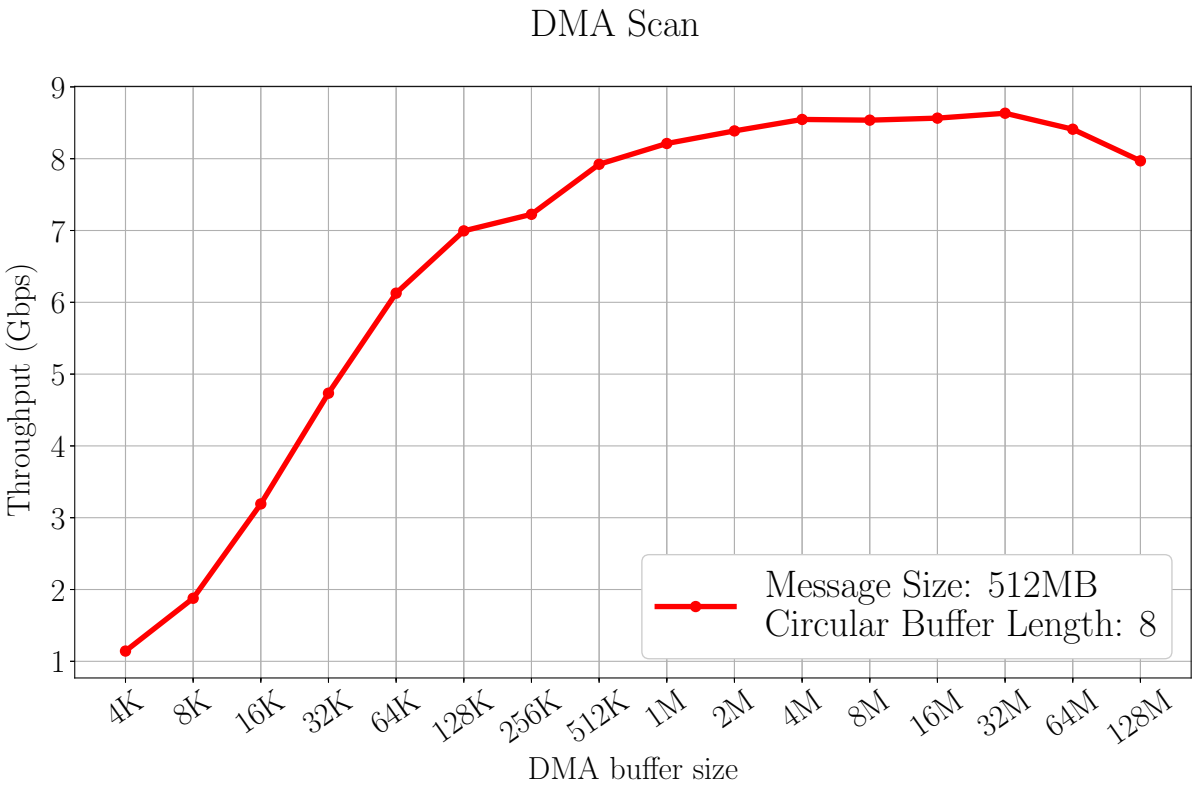
Για κάθε μήκος του circular buffer, το benchmark έτρεξε 20 φορές. Το μήνυμα παρέμεινε σταθερό στα 512MB, και το RDMA Buffer size ήταν 32MB σε κάθε περίπτωση.

Στην γραφική, μπορούμε να παρατηρήσουμε μία σημαντική αύξηση στην επίδοση, όταν αυξήσουμε το μέγεθός του από μία σε δύο θέσεις. Η αύξηση αυτή εξηγείται ως εξής. Ας υποθέσουμε πως ο circular buffer αποτελείται από μια θέση. Εκτελείται ένα dma write. Το επόμενο write θα πρέπει να περιμένει μέχρι να ολοκληρωθεί η αντίστοιχη λειτουργία ανάγνωσης, όποτε και θα ει-

δοποιηθεί ο αποστολέας. Αυτό εισάγει καθυστέρηση. Αυτή η καθυστέρηση διορθώνεται αμέσως προσθέτοντας μια άλλη θέση στο κυκλικό buffer. Φυσικά σε αυτή την περίπτωση, η τρίτη επακόλουθη εγγραφή θα περιμένει και ούτω καθεξής. Ωστόσο, βλέπουμε ότι για ένα μήκος μεγαλύτερο από δύο, η αύξηση της απόδοσης, εάν υπάρχει, είναι σημαντικά χαμηλότερη. Η καλύτερη τιμή φαίνεται να είναι μεταξύ 8 και 16 θέσεων. Μετά από αυτό, φαίνεται ότι το overhead των υπολογισμών που προστίθεται, επιβαρύνει την επίδοση.

### 4.2.4 RDMA Buffer Size

Η γραφική παράσταση 4.5 δείχνει μια σάρωση του μεγέθους του RDMA Buffer, που αντιστοιχεί σε μια θέση του circular buffer.




Σχήμα 4.5: RapidIO DMA Cell Size Scan

Για κάθε RDMA buffer size, το benchmark έτρεξε 20 φορές. Το μέγεθος του μηνύματος εδώ επίσης έμεινε σταθερό στα 512MB, χρησιμοποιώντας ένα κυκλικό buffer 8 θέσεων.

Όπως αναμενόταν, για μεγαλύτερο RDMA Buffer size έχουμε καλύτερη από-

δοση. Για μικρότερα μεγέθη , τα μηνύματα πρέπει να κοπούν σε μεγαλύτερο αριθμό κομματιών, εισάγοντας εισάγοντας μεγαλύτερο overhead, και συνεπώς ρίχνοντας την επίδοση. Το καλύτερο αποτέλεσμα προέρχεται από RDMA buffers μεγέθους 4, 8, 16 και 32 MB, με το τελευταίο να είναι το καλύτερο, αλλά μόνο ελαφρώς

## 4.3 Breakdown Analysis

Σε αυτή την ενότητα γίνεται μια προσπάθεια κατανόησης της κατανομής της απόδοσης στα διάφορα στοιχεία του συστήματος.

Για να γίνει αυτό, λήφθηκαν μετρήσεις χρόνου σε διάφορες τοποθεσίες σε όλη την κρίσιμη διαδρομή των συναλλαγών αποστολής και λήψης.

### 4.3.1 Send

Τα αποτελέσματα της εκτέλεσης του send φαίνονται στις γραφικές παραστάσεις 4.8 και **??**.

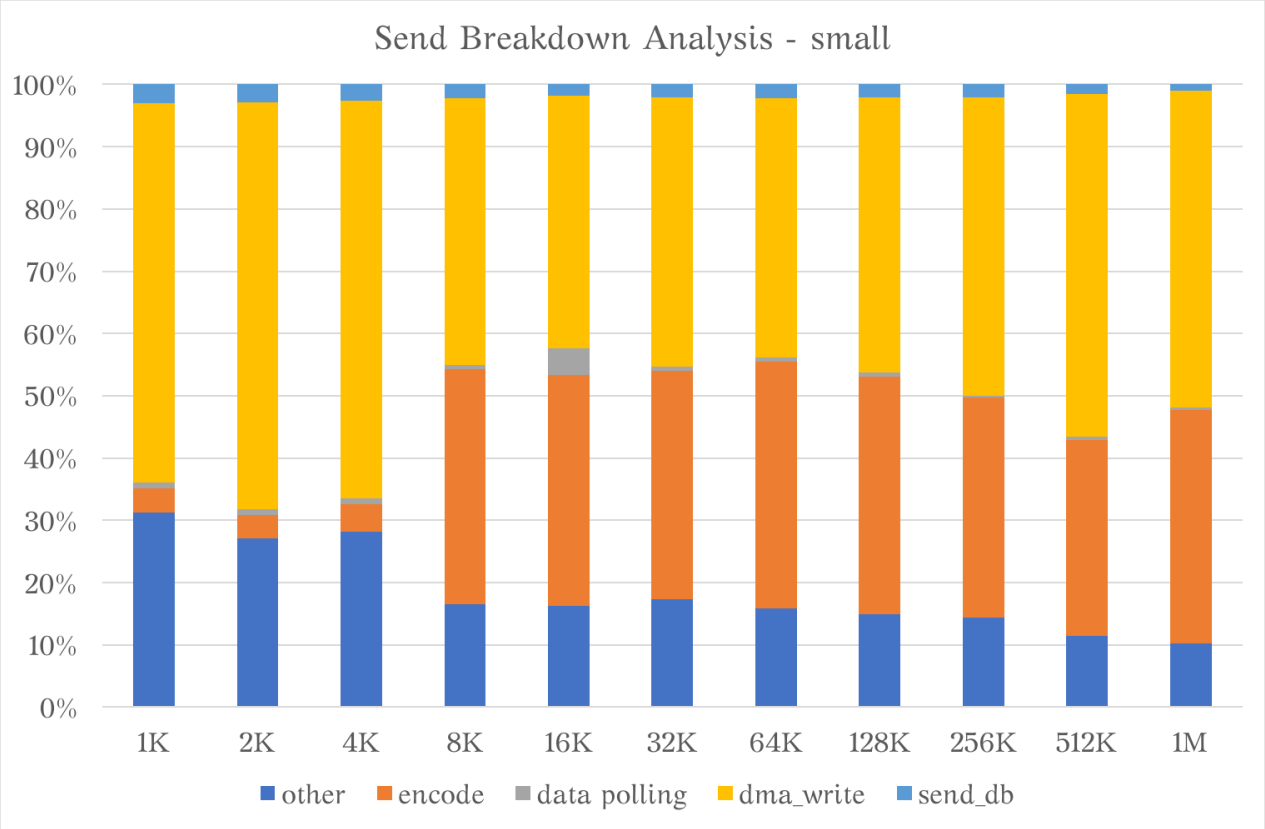
Όπως φαίνεται στα γραφήματα, η περισσότερος χρόνος καταναλώνεται από την κλήση της βιβλιοθήκης `dma_write()`. Εκτός αυτού, η συνάρτηση του `encode` παίρνει επίσης αρκετό χρόνο. Αυτό συμβαίνει επειδή η κλάση encoder του ZeroMQ, κάνει ένα `memcpy()`. Όπως βλέπουμε, πέρα από τα `dma_write()` και `encode()`, οι επιπτώσεις των υπολοίπων είναι ασήμαντες.

Συνεπώς, δεν αντιμετωπίσαμε απροσδόκητες καθυστερήσεις.

### 4.3.2 Receive

Τα γραφήματα των εικόνων **??** και **??** αναπαριστούν τα αποτελέσματα για την πλευρά του receive.

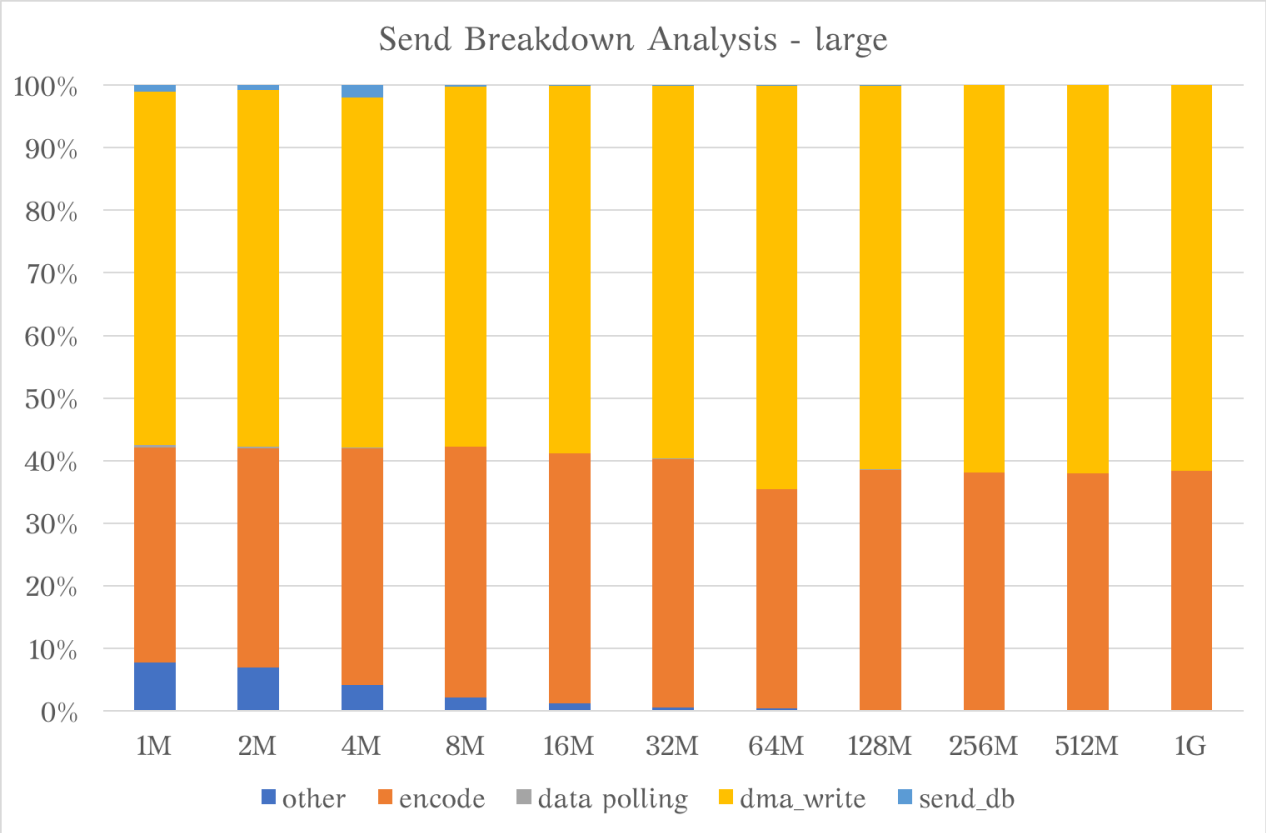
Εδώ, η καταχώρηση `in_event` είναι αυτή που παίρνει τον περισσότερο χρόνο. Αναφέρεται στον χρόνο που περνάει ανάμεσα σε δύο διαδοχικές κλήσεις της `in_event`. Ο χρόνος αυτός μπορεί να περιλαμβάνει τον χρόνο αναμονής για ένα doorbell, την διαχείρισή του, και το overhead από τις λειτουργίες επί του



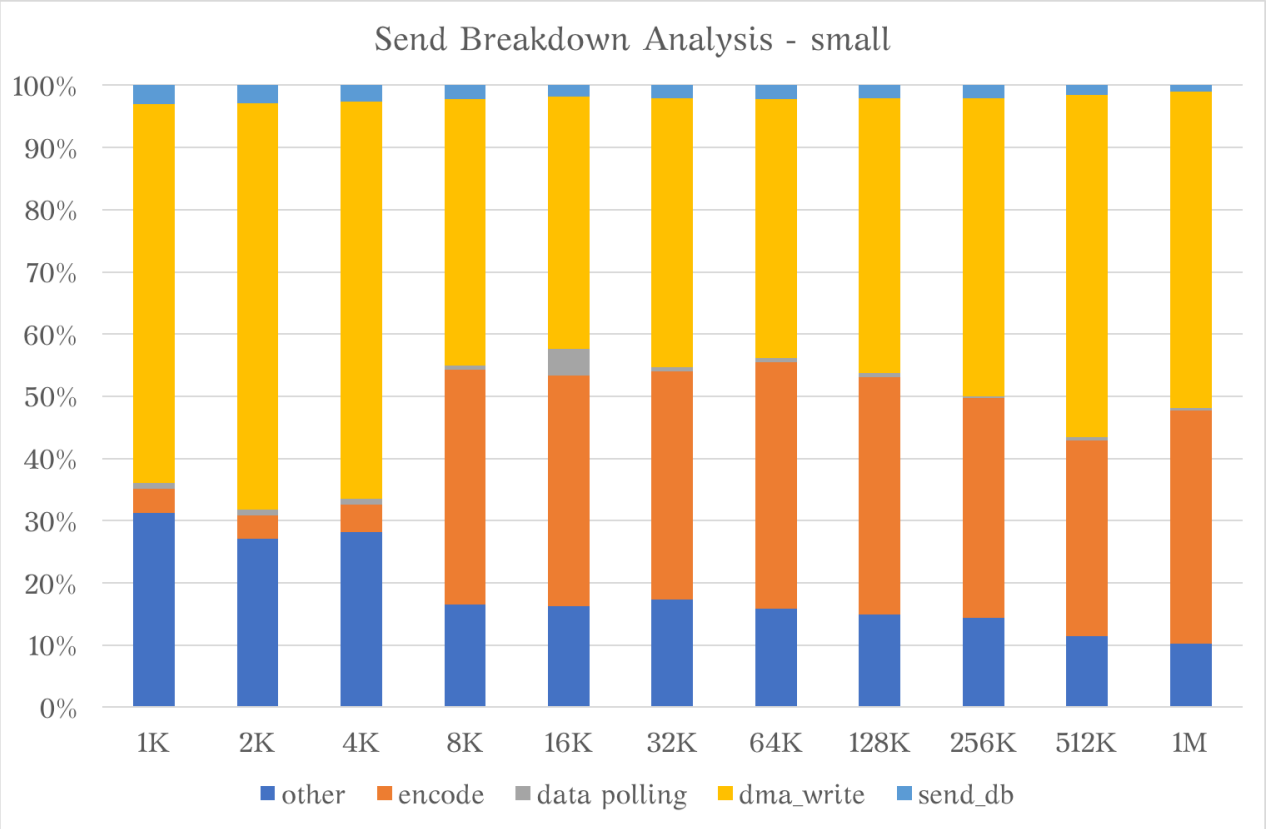

Σχήμα 4.6: Breakdown analysis for the ZeroMQ sender [Small]

file descriptor. Η συνάρτηση `decode()` παίρνει επίσης αρκετό χρόνος καθώς τα μηνύματα μεγαλώνουν, διότι περιλαμβάνει ένα memcpy, το οποίο απαιτεί πόρους.

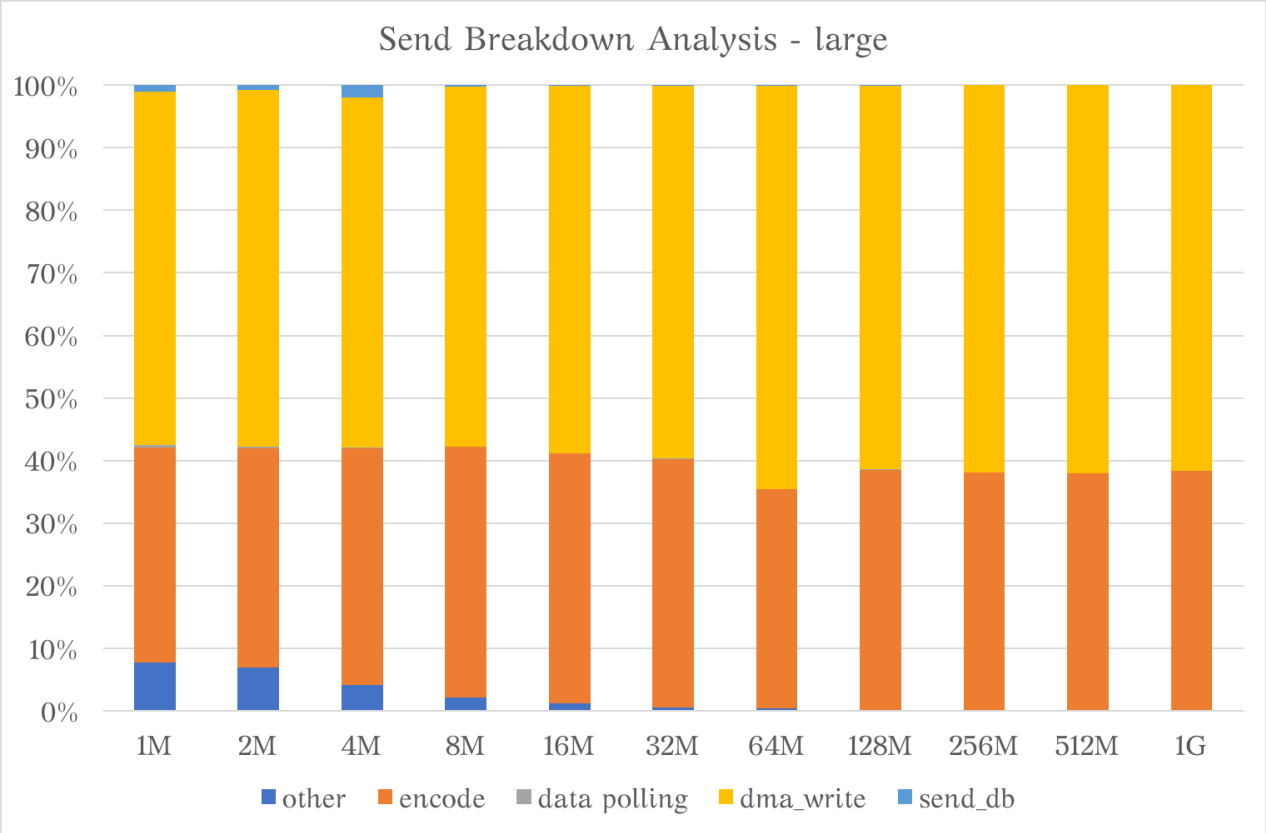
Γενικώς, από την ανάλυση αυτή προέκυψαν δύο σημεία. Πρώτον, οι απαραίτητες λειτουργίες επί του περιγραφητή αρχείου εισάγουν μια σημαντική καθυστέρηση. Δεύτερον, τα memcpy που λαμβάνουν μέρος στα πλαίσια των encoder και decoder του ZeroMQ είναι τα κύρια σημεία συμφόρησης της υλοποίησης.




Σχήμα 4.7: Breakdown analysis for the ZeroMQ sender [Large]




Σχήμα 4.8: Breakdown analysis for the ZeroMQ sender [Small]




Σχήμα 4.9: Breakdown analysis for the ZeroMQ sender [Large]

# Κεφάλαιο 5

# Σχετικές Εργασίες

## 5.1 Εισαγωγή

Αυτό το κεφάλαιο αναφέρεται σε εργασίες σχετικά με τις διάφορες τεχνολογίες που χρησιμοποιήθηκαν. Το πρώτο μέρος αναφέρεται σε εφαρμογές του πρωτοκόλλου RapidIO. Το δεύτερο μιλάει για την χρήση του ZeroMQ σε διάφορα πλαίσια, και τέλος γίνεται λόγος για προηγούμενες διπλωματικές παρόμοιου περιεχομένου.

## 5.2 RapidIO

Εργασίες σχετικά με το RapidIO είναι κατά γενική ομολογία δύσκολο να βρεθούν. Καθώς το παραδοσιακό πεδίο εφαρμογής του ήταν τα ενσωματωμένα συστήματα, υποθέτουμε πως ένα μεγάλο μέρος της έρευνας γύρω από αυτό είναι ιδιόκτητο Έχουν γίνει μελέτες σχετικά με την πιθανή χρήση του σε realtime εφαρμογής, όπως παρουσιάζεται στο [15], το 2004. Το 2010, είχε γίνει μία δημοσίευση σχετικά με την εφαρμογή του ως δίκτυο διασύνδεσης ενός υψηλής απόδοσης, ετερογενούς, ενσωματωμένου συστήματος, όπως περιγράφεται στο [16].

Το 2005, παρουσιάστηκε η υλοποίηση ενός οδηγού δικτύου για το Linux, στο [17].

Το 2016 κυκλοφόρησαν προσαρμογείς δικτύου από RapidIO σε PCIe, που

άνοιξαν τον δρόμο για πρακτικά οποιαδήποτε εφαρμογή [18]. Η δουλειά που παρουσιάστηκε στο [?], περιλαμβάνει την εφαρμογή του στρώματος μεταφοράς RapidIO για το ROOT, ένα πλαίσιο επεξεργασίας δεδομένων με κατεύθυνση την ανάλυση δεδομένων φυσικής και προσομοιώσεων, [20] και για το DAQPIPE, ένα μετροπρόγραμμα για την αξιολόγηση διαφορετικών δικτυακών αρχιτεκτονικών ως προετοιμασία για την επικείμενη αναβάθμιση του πειράματος LHCb [?] στο CERN, [?]. Οι συνεισφορές αυτές έκαναν το πρώτο βήμα για την εισαγωγή του RapidIO στον χώρο του HPC.

## 5.3 ZeroMQ

Το ZeroMQ είναι μια δημοφιλής λύση ανάμεσα στα συστήματα ανταλλαγής μηνυμάτων, που χρησιμοποιείται σε ένα ευρύ πεδίο εφαρμογών. Για παράδειγμα, το ZeroMQ, ήταν ο κύριος υποψήφιος για την χρήση του ως λύση middleware στο CERN, για την λειτουργία των επιταχυντών [23].

Πιο κοντά σε αυτήν την δουλειά, έχουν υλοποιηθεί στο παρελθόν επεκτάσεις για δίκτυα διασύνδεσης υψηλών επιδόσεων για την βιβλιοθήκη, όπως το VMCI [24], το οποίο παρέχει ένα κανάλι επικοινωνίας ανάμεσα σε εικονικές μηχανές και host. Μια άλλη υλοποίηση υποστηρίζει το SCIF [25], το οποίο προσφέρει ένα κανάλι επικοινωνία ανάμεσα σε host επεξεργαστές και Xeon Phi coprossecors. Το SCIF είναι επίσης RDMA-enabled.

## 5.4 RDMA-enabled δίκτυα διασύνδεσης

Στα πλαίσια προηγούμενων διπλωματικών εργασιών που έχουν διεκπεραιωθεί στο Εργαστήριο Υπολογιστικών Συστημάτων του ΕΜΠ, έχει γίνει ουσιώδη δουλειά σχετικά με RDMA-enabled δίκτυα διασύνδεσης. Μία ομάδα εργασιών αναφέρεται στην χρήση μεταγωγών δικτύου 10GbE όπου στο [26], σχεδιάζεται και υλοποιείται το πρωτόκολλο SLURPoE, για χρήση πάνω από υποδομή που υποστηρίζει RDMA. Το ίδιο πρωτόκολλο ερευνήθηκε περαιτέρω στα [27, 28], για ανάγκες εικονικοποίησης.

Σε μια άλλη ομάδα έγινε χρήση του δικτύου διασύνδεσης myrinet, μια τεχνολογία μεταγωγής πακέτων που επίσης υποστηρίζει την χρήση RDMA, για

διάφορες εφαρμογές, όπως Δίκτυα Αποθήκευσης [29, 30] και περιβάλλοντα πολυπύρηνης διασύνδεσης [31].

Όπως γίνεται αισθητό, τα δίκτυα διασύνδεσης που υποστηρίζουν απευθείας πρόσβαση απομακρυσμένης μνήμης έχουν υποσχόμενα χαρακτηριστικά, όπως χαμηλό latency, μεγάλο εύρος ζώνης και υποστήριξη κλιμάκωσης, καθιστώντας τα ενδιαφέροντα σε διάφορα πεδία εφαρμογής.

# Κεφάλαιο 6

# Επίλογος και Μελλοντικές Επεκτάσεις

Αυτό το κεφάλαιο αποτελεί ένα επίλογο για τη δουλειά που έχει γίνει στα πλαίσια της παρούσας διπλωματικής. Παρουσιάζεται μια σύνοψη των σημαντικότερων σημείων, πριν συζητηθούν πιθανές μελλοντικές επεκτάσεις.

## 6.1 Σύνοψη

Ως αποτέλεσμα αυτής της διπλωματικής εργασίας, έχει υλοποιήθηκε μια επέκταση της βιβλιοθήκης ανταλλαγής μηνυμάτων ZeroMQ για την υποστήριξη του στρώματος μεταφοράς του RapidIO. Η επέκταση αυτή επιτρέπει τη χρήση του ZeroMQ ως διεπαφή επικοινωνίας σε ένα κατανεμημένο σύστημα πολλαπλών κόμβων. Εφόσον το ZeroMQ παρέχει μια διεπαφή που είναι αφηρημένο σε μεγάλο βαθμό σε σχέση με την εφαρμογή του, διευκολύνεται η χρήση του RapidIO σε περιβάλλοντα HPC. Το coding effort που συνοδεύει την υλοποίηση ενός νέου στρώματος μεταφοράς έχει αρθεί, καθώς το ZeroMQ επιτρέπει την εύκολη επιλογή του πρωτοκόλλου που θα χρησιμοποιήσει.

Το ZeroMQ είναι σχεδιασμένο και υλοποιημένο για χρήση με sockets. Η διαδικασία υλοποίησης της επέκτασής του επέτρεψε τη διερεύνηση της ασυμβατότητας μεταξύ των δύο προγραμματιστικών προτύπων, του socket programming δηλαδή και του MMIO. Προτάθηκε ένα απλός μηχανισμός για την ενοποίηση των δύο προτύπων.

Η χρήση του RapidIO έχει διερευνηθεί εκτός του περιβάλλοντος που συνήθως έβρισκε εφαρμογή, του χώρου δηλαδή των ενσωματωμένων συστημάτων. Οι μετρήσεις της επίδοσής του έγιναν στο πλαίσιο ενός συστήματος με 16 κόμβους, επιτρέποντας την αξιολόγηση του πρωτοκόλλου.

## 6.2 Μελλοντικές Επεκτάσεις

Ως μέρος αυτής της διπλωματικής δημιουργήθηκε ένα Proof of Concept. Ωστόσο, η επίδοση του συστήματος μπορεί να βελτιωθεί με ένα αριθμό πιθανών τροποποιήσεων και επεκτάσεων. Επιπλέον, θα μπορούσαν να διερευνηθούν διαφορετικές πτυχές αξιολόγησης. Η ενότητα αυτή προτείνει ενδεχόμενες κατευθύνσεις για μελλοντικές επεκτάσεις.

Στο επίπεδο της υλοποίησης:

- Ο circular buffer που υλοποιήθηκε στο RapidIO engine του ZeroMQ έχει περιθώρια για βελτιστοποίηση. Ο αριθμός των RDMA buffers που χρησιμοποιούνται ανά συναλλαγή μπορεί να υπολογίζεται δυναμικά, ανάλογα με το μέγεθος του μηνύματος και τους περιορισμούς της δεσμευμένης μνήμης.
- Ο circular buffer θα μπορούσε να αντικατασταθεί από κάποια άλλη δομή δεδομένων, όπως μια ουρά. Με την κατάλληλη εφαρμογή αλγορίθμου, η επίδοση θα μπορούσε να βελτιωθεί.
- Θα μπορούσε να διερευνηθεί η αποκλειστική χρήση Zero-Copy λειτουργιών κατά μήκος του critical path.
- Τα κομμάτια του ZeroMQ που προέκυψαν ως σημεία συμφόρησης στο Breakdown Analysis, είναι υποψήφια βελτιστοποίησης. Αυτό ισχύει όχι μόνο για την περίπτωση του RapidIO, αλλά για το ZeroMQ γενικότερα.

Στο επίπεδο της αξιολόγησης:

- Προκειμένου να μετρήσουμε την επίδοση των doorbells, θα μπορούσαμε να πάρουμε logs από το NIC, για το κομμάτι της αξιολόγησης στο οποίο αναφέρεται το breakdown στην πλευρά του receive.
- Το σύστημα θα μπορούσε να αξιολογηθεί με παραπάνω από 16 κόμβους.

Αν μελετήσουμε περαιτέρω την συμπεριφορά του σε κλίμακα, πιθανόν να εμφανιστούν και άλλα σημεία προς βελτίωση.

- Θα μπορούσαμε να τρέξουμε ένα μετροπρόγραμμα που ανταποκρίνεται στην πραγματικότητα, σαν μια προσομοίωση φυσική σε ένα κατανεμημένο σύστημα. Αυτό θα πίεζε την υλοποίηση τόσο στο latency, όσο και στο throughput, αξιολογώντας την πιθανή του εφαρμογή σε τέτοια σενάρια

# Βιβλιογραφία


[1] Mellanox Technologies, *Introduction to InfiniBand™ - White Paper* http://www.mellanox.com/pdf/whitepapers/IB_Intro_WP_190.pdf

[2] G. F. PFister, *An introduction to the Infiniband architecture*, High Performance Mass Storage and Parallel I/O:Technologies and Applications. New York, Wiley-IEEE Press, 2002, pp 616-632

[3] Intel Corporation, *Intel QuickPath Architecture* https://www.intel.com/pressroom/archive/reference/whitepaper_QuickPath.pdf

[4] *ZeroMQ* https://github.com/zeromq/libzmq

[5] Motorola Semiconductor Product Sensor *RapidIO™: An Embedded System Component Network Architecture* http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.93.4945&rep=rep1&type=pdf

[6] S. Fuller, RapidIO Trade Association, *RapidIO: The Embedded System Interconnect* John Wiley & Sons, Ltd, 2005

[7] RapidIO.org, *RapidIO™ The Interconnect Architecture for High Performance Embedded Systems - White Paper* http://www.rapidio.org/files/techwhitepaper_rev3.pdf

[8] G. Shippen, *System Interconnect Fabrics:Ethernet versus RapidIO® Technology* http://cache.freescale.com/files/32bia/doc/app_note/AN3088.pdf

[9] RapidIO.org, *RapidIO™ Interconnect Specification Version 4.0* http://www.rapidio.org/wp-content/uploads/2016/06/RapidIO-Specification-4.0.pdf

[10] PCI-SIG, *PCI Express® Base Specification Revision 2.1*

[11] RapidIO.org, *RapidIO™ Interconnect Specification Version 3.1* http://www.rapidio.org/wp-content/uploads/2014/10/RapidIO-3.1-Specification.pdf

[12] RapidIO.org *RapidIO Linux Kernel Driver*, 2016 `https://github.com/RapidIO/kernel-rapidio`

[13] RapidIO.org, *RapidIO Remote Memory Access Platform software*, 2016 `https://github.com/RapidIO/RapidIO_RRMAP`

[14] *Internal Architecture of libzmq - White Paper* http://zeromq.org/whitepapers:architecture

[15] D. Bueno, A. Leko, C. Conger,I. Troxel and A. D. George, *Simulative analysis of the RapidIO embedded interconnect architecture for real-time, network-intensive applications* , 2004, Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks

[16] W. Changrui, C. Fan and C. Huizhi, *A high-performance heterogeneous embedded signal processing system based on serial RapidIO interconnection*, 2010, 3rd IEEE International Conference on Computer Science and Information Technology **2** 611-614

[17] M. Porter, *RapidIO for Linux*, 2005 http://landley.net/kdocs/ols/2005/ols2005v2-pages-43-56.pdf

[18] Integrated Device Technology *Tsi721™ Datasheet*, 2016 https://www.idt.com/document/dst/tsi721-datasheet

[19] S. Baymani, K. Alexopoulos and S. Valat, *Exploring RapidIO technology within a DAQ system event building network*, 2017, IEEE Transactions on Nuclear Science **9** 2598 - 2605

[20] Rene Brun and Fons Rademakers, *ROOT - An Object Oriented Data Analysis Framework*, Proceedings AIHENP'96 Workshop, Lausanne, Sep. 1996, Nucl. Inst. & Meth. in Phys. Res. A 389 (1997) 81-86. See also http://root.cern.ch/.

[21] LHCb Collaboration, *The LHCb Detector at the LHC* http://inspirehep.net/record/796248/

[22] D. Cámpora, S. Valat, B. Vőneki, S. Baymani, *LHCB-DAQPIPE Wiki* `https://gitlab.cern.ch/svalat/lhcb-daqpipe-v2/wikis/home`

[23] A. Dworak, F. Ehm, W. Sliwinski, M. Sobczak, *MIDDLEWARE TRENDS AND MARKET LEADERS 2011* http://zeromq.wdfiles.com/local–files/intro%3Aread-the-manual/Middleware%20Trends%20and%20Market%20Leaders%202011.pdf

[24] I. Kulakov, *VMCI extension for ZeroMQ* `https://github.com/Kentzo/libzmq/tree/vmci`

[25] A. Santogidis, *SCIF extension for ZeroMQ* `https://github.com/theWayofthecode/libzmq/tree/scif`

[26] Ν. Νικολέρης, *Σχεδίαση και Υλοποίηση μηχανισμού απευθείας απομακρυσμένης πρόσβασης στη μνήμη με χρήση προγραμματιζόμενου προσαρμογέα δικτύου 10GbE*, Διπλωματική Εργασία, Εθνικό Μετσόβιο Πολυτεχνειό, 2009

[27] Ε. Κοζύρη, *Ένταξη Σημασιολογίας Δικτύων Διασύνδεσης Υψηλής Επίδοσης σε Εικονικές Μηχανές*, Διπλωματική Εργασία, Εθνικό Μετσόβιο Πολυτεχνειό, 2010

[28] Σ. Ψωμαδάκης, *Δίκτυα Διασύνδεσης Υψηλής Επίδοσης σε Εικονικά Περιββάλοντα*, Διπλωματική Εργασία, Εθνικό Μετσόβιο Πολυτεχνείο, 2011

[29] Α. Νάνος, *Σχεδίαση Και Υλοποίηση Μηχανισμού Μεταφοράς Δεδομένων Από Συσκευές Αποθήκευσης Σε Δίκτυα Myrinet, Χωρίς Τη Μεσολάβηση Της Ιεραρχίας Μνήμης*, Διπλωματική Εργασία, Εθνικό Μετσόβιο Πολυτεχνείο 2006

[30] Κ. Βενετσανόπουλος, *Σχεδίαση και Υλοποίηση Οδηγού Συσκευής για τη Χρήση Προσαρμογέα Myrinet ως Αποθηκευτικού Μέσου Υψηλής Επίδοσης στο Λειτουργικό Σύστημα Linux*, Διπλωματική Εργασίας, Εθνικό Μετσόβιο Πολυτεχνείο, 2010

[31] Β. Λιασκοβίτης, *Χρονοδρομολόγηση Φωλιασμένων Βρόχων σε Συστοιχία Πολυεπεξεργαστών Συνδεδεμένων με Myrinet*, Διπλωματική Εργασία, Εθνικό Μετσόβιο Πολυτεχνείο, 2004