ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Αλγόριθμοι Κατανεμημένων Συστημάτων

Συμπληρωματικές Διδακτικές Σημειώσεις για το μάθημα

ΚΑΤΑΝΕΜΗΜΕΝΑ ΣΥΣΤΗΜΑΤΑ

Διδάσκων: Κωνσταντίνος Αντωνής
Δρ. Μηχανικός Η/Υ & Πληροφορικής Πανεπιστημίου Πατρών

Λαμία, Δεκέμβριος 2015

Περιεχόμενα

1. Εισαγωγή

Στο εγχειρίδιο αυτό γίνεται μια παρουσίαση ενδεικτικών - αντιπροσωπευτικών αλγορίθμων για διάφορα θέματα που απασχολούν τα κατανεμημένα συστήματα. Σε πολλούς από αυτούς τους αλγορίθμους γίνεται και ανάλυση της πολυπλοκότητάς τους σε μηνύματα και χρόνο, παράγοντες που παίζουν πολύ σημαντικό ρόλο στην απόδοση ενός αλγορίθμου, ώστε να μπορεί να συγκριθεί με άλλους ανταγωνιστικούς.

Για το λόγο λοιπόν αυτό δίνουμε παρακάτω τους ορισμούς κάποιων εργαλείων εκτίμησης πολυπλοκότητας.

Θεωρούμε τη συνάρτηση T(n) η οποία εκφράζει μία πολυπλοκότητα μηνυμάτων ή χρόνου για έναν αλγόριθμο. Ορίζουμε:

1. $T(n) \in O(f(n))$, αν υπάρχουν σταθερές c και n_0 ώστε $T(n) \leq c*f(n)$, για κάθε $n \geq n_0$.

2. $T(n) \in \Omega(g(n))$, αν υπάρχουν σταθερές c και n_0 ώστε $T(n) \geq c*g(n)$, για κάθε $n \geq n_0$.

3. $T(n) \in \Theta(h(n))$, αν $T(n) \in O(h(n))$ και $T(n) \in \Omega(h(n))$.

2. Πρωτόκολλα Συγχρονιστών

Η ανάγκη για ταυτόχρονη εκμετάλλευση των ευνοϊκών χαρακτηριστικών των ασύγχρονων δικτύων και των σύγχρονων αλγορίθμων οδήγησε τους ερευνητές (1985) στο σχεδιασμό των συγχρονιστών. Τα πρωτόκολλα των συγχρονιστών, προσομοιώνοντας ένα ιδεατό σύγχρονο περιβάλλον σε ένα ασύγχρονο δίκτυο, εξασφαλίζουν επιτυχή αξιοποίηση των παραπάνω χαρακτηριστικών. Μάλιστα, οι αλγόριθμοι των συγχρονιστών είναι οι πρώτοι που καταφέρνουν κάτι τέτοιο στη γενική περίπτωση, καθώς μπορούν να λειτουργήσουν για κάθε σύγχρονο αλγόριθμο και κάθε δίκτυο.

2.1. Γενικά για συγχρονιστές

Για να γίνει φανερός ο ρόλος και η χρησιμότητα των συγχρονιστών είναι απαραίτητο να οριστούν τα χαρακτηριστικά των σύγχρονων και των ασύγχρονων δικτύων. Η προσφορά των συγχρονιστών πηγάζει ακριβώς από τα σχετικά πλεονεκτήματα και μειονεκτήματα της κάθε μιας από αυτές τις κατηγορίες δικτύων καθώς και των αντιστοίχων για αυτές κατανεμημένων αλγορίθμων.

2.1.1. Σύγχρονα και ασύγχρονα δίκτυα

Στο μοντέλο των σύγχρονων δικτύων υπάρχει ένα σφαιρικό ρολόι που τους παλμούς του ακούνε ταυτόχρονα όλοι οι επεξεργαστές. Στα σύγχρονα δίκτυα η επεξεργασία ακολουθεί το παρακάτω μοντέλο: μετά από ένα σφαιρικό σήμα εκκίνησης όλοι οι επεξεργαστές αρχίζουν την επεξεργασία ταυτόχρονα. Σε κάθε παλμό του σφαιρικού ρολογιού κάθε επεξεργαστής, ανάλογα με το πρόγραμμά του εκτελεί ένα υπολογιστικό βήμα και στέλνει μηνύματα σε κάποιους από τους γείτονές του. Η καθυστέρηση μετάδοσης στα κανάλια επικοινωνίας και το χρονικό διάστημα μεταξύ των παλμών είναι τέτοια ώστε όλα τα μηνύματα να φτάσουν στους προορισμούς τους έγκαιρα για να μπορούν εκεί να χρησιμοποιηθούν στο επόμενο υπολογιστικό βήμα. Είναι εγγυημένο δηλαδή στο σύγχρονο μοντέλο πως όλα τα τοπικά υπολογιστικά βήματα θα έχουν προλάβει να ολοκληρωθούν, όλα τα μηνύματα θα έχουν φτάσει στον προορισμό τους πριν το ρολόι χτυπήσει ξανά.

Αντίθετα, σύμφωνο με το μοντέλο του ασύγχρονου δικτύου, δεν υπάρχει σφαιρικό ρολόι, επομένως δεν υπάρχει και σφαιρικό σήμα εκκίνησης για τους επεξεργαστές – κόμβους του δικτύου. Επιπλέον, οι καθυστερήσεις διάδοσης των μηνυμάτων μεταξύ των επεξεργαστών είναι απρόβλεπτες. Κάθε μήνυμα φτάνει στον προορισμό του μετά από πεπερασμένο αλλά απροσδιόριστο χρόνο.

2.1.2. Τι πετυχαίνει ένας συγχρονιστής

Είναι φανερό από τους παραπάνω ορισμούς των δύο μοντέλων, πως σε ένα ασύγχρονο κατανεμημένο αλγόριθμο τα γεγονότα που συμβαίνουν σε ένα κόμβο του συστήματος δεν ενεργοποιούνται με την λήψη μηνυμάτων από τον κόμβο ή την ολοκλήρωση μιας διαδικασίας σε αυτόν. Τα γεγονότα λοιπόν σε κάθε κόμβο δεν συμβαίνουν σε κάποιο σε κάποιες προκαθορισμένες χρονικές στιγμές αλλά και οι ενέργειες όλων των επεξεργαστών του δικτύου δεν ακολουθούν κάποια προκαθορισμένη σειρά μεταξύ τους. Έτσι ο σχεδιασμός αλγορίθμων για ασύγχρονα κατανεμημένα συστήματα είναι πιο δύσκολος από τον των αντιστοίχων σύγχρονων αλγορίθμων. Ακόμα, οι ασύγχρονοι αλγόριθμοι υστερούν σημαντικά πολλές φορές σε

σχέση με τους αντίστοιχους σύγχρονους αλγορίθμους όσον αφορά την πολυπλοκότητά τους. Επιπλέον, η διόρθωση (debugging), ο έλεγχος (testing) και η ανάλυσή τους είναι πιο επίπονη σε σχέση με τους αντίστοιχους σύγχρονους αλγορίθμους. Από την άλλη πλευρά τα ασύγχρονα συστήματα είναι πολύ πιο διαδεδομένα και πιο εύκολα στην κατασκευή και τη συντήρηση από τα σύγχρονα. Είναι εμφανής έτσι η προσφορά ενός μηχανισμού που θα επέτρεπε σε σύγχρονους αλγορίθμους να μπορούν να τρέχουν σε ασύγχρονα συστήματα, αφού στη γενική περίπτωση ένα πρόγραμμα, σχεδιασμένο για σύγχρονα συστήματα δε μπορεί χωρίς καμιά επέμβαση – τροποποίηση να τρέξει σωστά σε ένα ασύγχρονο σύστημα.

Στο παρελθόν πολλοί ασύγχρονοι κατανεμημένοι αλγόριθμοι σχεδιάστηκαν τροποποιώντας αντίστοιχες σύγχρονες εκδοχές τους έτσι ώστε αυτές να μπορούν να τρέξουν πάνω σε ένα ιδεατό (virtual) σύγχρονο περιβάλλον. Αυτό το περιβάλλον το συνιστούσαν χαμηλού επιπέδου ad-hoc πρωτόκολλα ενσωματωμένα στο σχεδιασμό των ίδιων αυτών αλγορίθμων. Ο Awerbuch διέκρινε πρώτος αυτό το κοινό χαρακτηριστικό και πρότεινε ένα γενικό χαμηλού επιπέδου πρωτόκολλο, τον συγχρονιστή, που επιτρέπει τον ομοιόμορφο σχεδιασμό ασύγχρονων κατανεμημένων αλγορίθμων παρέχοντας τη δυνατότητα σε κάθε σύγχρονο αλγόριθμο να τρέχει πάνω σε οποιοδήποτε ασύγχρονο δίκτυο. Θα μπορούσαμε να χαρακτηρίσουμε τους συγχρονιστές σαν επίπεδα (layers) του λογικού του δικτύου του κατανεμημένου συστήματος. Ακόμα, οι συγχρονιστές αναφέρονται αλλού ως χαμηλού επιπέδου πρωτόκολλα και αλλού ως μεταγλωτιστές που έχουν ως είσοδο και έξοδο κατανεμημένους αλγόριθμους, σύγχρονους και ασύγχρονους αντίστοιχα. Ο Awerbuch περιγράφει τους συγχρονιστές σαν μια γενική τεχνική προσομοίωσης που θα επιτρέπει στο χρήστη να γράφει ένα αλγόριθμο για ασύγχρονο δίκτυο σαν αυτός να πρόκειται να τρέξει σε ένα σύγχρονο δίκτυο. Μια τεχνική δηλαδή που θα εξασφαλίζει πως το ασύγχρονο δίκτυο τελικά θα συμπεριφέρεται σαν σύγχρονο από τη σκοπιά της συγκεκριμένης εκτέλεσης του συγκεκριμένου σύγχρονου αλγορίθμου. Ο Awerbuch έχει επιδείξει μια ακόμη σημαντική χρήση των συγχρονιστών. Η εφαρμογή τους σε γνωστούς σύγχρονους αλγόριθμους έχει σαν αποτέλεσμα νέους ασύγχρονους αλγόριθμους που σε μερικές περιπτώσεις είναι αποδοτικότεροι σε χρόνο και μηνύματα από οποιονδήποτε εκ των προτέρων γνωστό «γνήσιο» ασύγχρονο αλγόριθμο.

2.1.3. Γενικά χαρακτηριστικά ενός συγχρονιστή

Παρά τους ποικίλους χαρακτηρισμούς με τους οποίους, όπως φάνηκε στην προηγούμενη παράγραφο, μπορούμε να αναφερθούμε σε ένα συγχρονιστή, υπάρχουν σε όσους συγχρονιστές έχουν ως τώρα παρουσιάσει κοινά χαρακτηριστικά και κοινές απαιτήσεις από τη λειτουργία τους.

Ένας συγχρονιστής είναι κατ' αρχήν ένα ασύγχρονο κατανεμημένο πρωτόκολλο που τρέχει στο ασύγχρονο σύστημα παράλληλα με ένα σύγχρονο αλγόριθμο π του οποίου η εκτέλεση είναι ο τελικός στόχος μας. Ο αλγόριθμος π είναι βέβαια ένα κατανεμημένο πρωτόκολλο που συνίσταται από προγράμματα σε κάθε κόμβο-επεξεργαστή του κατανεμημένου συστήματος. Το κάθε ένα από τα προγράμματα αυτά είναι αναλυμένο σε υπολογιστικά βήματα. Όταν ο αλγόριθμος π εκτελείται σε ένα σύγχρονο δίκτυο, κάθε βήμα i του προγράμματος ενός κόμβου εκτελείται κατά τον ίδιο παλμό του σφαιρικού ρολογιού με τα αντίστοιχα βήματα i των προγραμμάτων όλων των υπολοίπων κόμβων του συστήματος. Το πρόβλημα που παρουσιάζεται όταν θέλουμε να τρέξουμε τον αλγόριθμο π σε ένα ασύγχρονο δίκτυο

και ο ρόλος του synchronizer συνδέονται άμεσα με την απαραίτητη ανταλλαγή μηνυμάτων- δεδομένων μεταξύ των κόμβων κατά. τη διάρκεια της επεξεργασίας.

2.1.4. Η συνθήκη συγχρονισμού

Αν κάθε επεξεργαστής μπορούσε ανεξάρτητος από τους άλλους να ολοκληρώσει την εκτέλεση του κομματιού του προγράμματος του σύγχρονου αλγορίθμου που είναι εγκατεστημένο σ' αυτόν και να αναφέρει κάπου τα τελικά του αποτελέσματα τότε η ασυγχρονία του δικτύου δε θα ήταν πρόβλημα. Τότε όμως θα είχαμε και έναν τετριμμένο κατανεμημένο αλγόριθμο. Στην πραγματικότητα, κατά τη διάρκεια της επεξεργασίας κάθε κόμβος, αφού ολοκληρώσει κάποιο υπολογιστικό βήμα, ανταλλάσσει με κάποιους από τους γείτονές του μηνύματα που περιέχουν πιθανώς αποτελέσματα μερικών υπολογισμών. Στέλνει και λαμβάνει μηνύματα που περιέχουν απαραίτητες πληροφορίες για να συνεχιστεί ορθά η επεξεργασία και σ' αυτόν τον ίδιο κόμβο και στους γείτονές του. Ακριβώς όμως επειδή ο αλγόριθμος π που προσπαθούμε να εκτελέσουμε είναι ένας σύγχρονος αλγόριθμος, στο ιδεατό σύγχρονο σύστημα που προσπαθούμε να προσομοιώσουμε θα πρέπει να ικανοποιείται η ακόλουθη συνθήκη συγχρονισμού:

Κάθε κόμβος θα πρέπει στο πέρας του βήματος i του σύγχρονου αλγορίθμου να μην προχωρά στον επόμενο χτύπο του ιδεατού σφαιρικού ρολογιού, δηλαδή στο βήμα i+ 1 του σύγχρονου αλγορίθμου, προτού να έχει στη διάθεσή του από τους γείτονές του, για να χρησιμοποιήσει κατόπιν, τα αποτελέσματα του αντίστοιχου βήματος i των προγραμμάτων τους.

Αυτό ακριβώς εξασφαλίζει κάθε πρωτόκολλο ενός συγχρονιστή. Περιορίζει κατά κάποιον τρόπο την ασυγχρονία του δικτύου με σχετικά και όχι απόλυτα χρονικά περιθώρια τα οποία δεν υπαγορεύονται από κάποιο σφαιρικό ρολόι αλλά από την εξέλιξη της εκτέλεσης του σύγχρονου πρωτοκόλλου σε κάθε κόμβο του συστήματος. Ισοδύναμα με τη συνθήκη που διατυπώθηκε μπορούμε να πούμε πως ένας συγχρονιστής καταφέρνει να κρατά συγχρονισμένους τους κόμβους ενός ασύγχρονου δικτύου αν ποτέ η διαφορά μεταξύ του αριθμού του βήματος του σύγχρονου αλγορίθμου που εκτελεί ένας κόμβος και του αντίστοιχου αριθμού σε οποιονδήποτε από τους γείτονές του δε γίνεται μεγαλύτερη από 1.

Στους αλγορίθμους που παρουσιάζονται στη συνέχεια φαίνονται καθαρά οι διαφορετικές απόψεις για το πως θα πρέπει να υλοποιηθεί ο παραπάνω στόχος καθώς και τα υπέρ και κατά κάθε διαφορετικής προσέγγισης. Σε κάποιες τεχνικές προτείνεται για παράδειγμα η ομαδική προώθηση όλου του συστήματος κάθε φορά στον επόμενο ιδεατό παλμό του σύγχρονου αλγορίθμου (συγχρονιστής β), ενώ σε άλλες υποστηρίζεται η τοπική καθυστέρηση της επεξεργασίας σε ένα κόμβο μέχρις ότου αυτός λάβει τα απαραίτητα μηνύματα από τους γείτονές του (συγχρονιστής α).

2.1.5. Μέτρηση της απόδοσης ενός συγχρονιστή

Οι ορισμοί για την πολυπλοκότητα επικοινωνίας και χρόνου των συγχρονιστών έχουν μικρές παραλλαγές ανάλογα με το μοντέλο του διχτύου όπου εξετάζεται η απόδοση κάθε συγχρονιστή. Σε γενικές γραμμές όμως ως πολυπλοκότητα επικοινωνίας ενός συγχρονιστή θεωρούμε την επιβάρυνση σε μηνύματα που εισάγει η λειτουργία του συγχρονιστή ανά βήμα του σύγχρονου αλγορίθμου. Όμοια, ως πολυπλοκότητα χρόνου του συγχρονιστή θεωρούμε τις επιπλέον καθυστερήσεις που προσθέτει ανά παλμό η λειτουργία του συγχρονιστή στην πολυπλοκότητα χρόνου του σύγχρονου αλγορίθμου.

2.2. Οι αλγόριθμοι α, β, γ του Awerbuch

Ο Awerbuch ορίζοντας για πρώτη φορά την έννοια του συγχρονιστή παρουσιάζει τρεις συγχρονιστές, τους α, β και γ , με μεγαλύτερη έμφαση στον τρίτο από αυτούς που αποτελεί ένα συνδυασμό των δύο πρώτων απλούστερων αλγορίθμων. Ο συγχρονιστής α είναι αποδοτικός όσον αφορά το χρόνο, αλλά δαπανηρός σε επικοινωνία (μηνύματα) ενώ το αντίθετο ισχύει για τον β. Για το λόγο αυτό, ο συγχρονιστής γ σχεδιάστηκε έτσι ώστε να συνδυάζει τα πλεονεκτήματα απόδοσης των α και β και να είναι έτσι αποδοτικός και ως προς το χρόνο και ως προς τα ανταλλασσόμενα μηνύματα.

2.2.1. Το Μοντέλο του συστήματος

Το κατανεμημένο σύστημα βασίζεται σε ένα point-to-point (store-and-forward) δίκτυο επικοινωνίας. Έστω ότι V είναι το σύνολο των κόμβων - επεξεργαστών του δικτύου και E είναι το σύνολο των links, δηλαδή των διπλής κατεύθυνσης (bidirectional) καναλιών επικοινωνίας μεταξύ των κόμβων - επεξεργαστών .Ο Awerbuch κάνει ακόμα την υπόθεση πως οι κόμβοι του δικτύου είναι ονοματισμένοι (έχουν όλοι μια μοναδική ταυτότητα) και δεν έχουν κοινή μνήμη. Ακόμα τα μηνύματα έχουν σταθερό μήκος και μεταφέρουν ένα περιορισμένο ποσοστό πληροφορίας.

Η μορφή της επεξεργασίας στο σύστημα είναι η εξής: κάθε κόμβος δέχεται μηνύματα από τους γείτονές του, τα επεξεργάζεται, πραγματοποιεί τοπικούς υπολογισμούς και στη συνέχεια στέλνει ο ίδιος μηνύματα στους γείτονές του. Γίνεται η υπόθεση πως όλες αυτές οι ενέργειες πραγματοποιούνται σε αμελητέο χρόνο. Ακόμα θεωρείται πως το πολύ ένα μήνυμα είναι δυνατόν να σταλεί μέσω ενός συγκεκριμένου link κατά τη διάρκεια ενός συγκεκριμένου παλμού και πως η καθυστέρηση διάδοσης σε κάθε link είναι το πολύ ίση με μια χρονική μονάδα του σφαιρικού ρολογιού.

2.2.2. Η έννοια της ασφάλειας (safety)

Ο Awerbuch για να ορίσει τους τρεις συγχρονιστές του εισάγει κατ' αρχήν την έννοια της ασφάλειας. 'Ενας κόμβος του συστήματος είναι ασφαλής ως προς ένα συγκεκριμένο παλμό (βήμα του σύγχρονου αλγορίθμου) αν κάθε μήνυμα του σύγχρονου αλγορίθμου που εστάλη από αυτό τον κόμβο στο συγκεκριμένο παλμό έχει φτάσει στον προορισμό του. Ένας κόμβος μπορεί να γνωρίζει πότε είναι ασφαλής αν θεωρήσουμε ότι για κάθε μήνυμά του που φτάνει. σε έναν γείτονά του, του επιστρέφεται ένα acknowledgement. Ο κόμβος είναι τελικά ασφαλής όταν όλα τα μηνύματά του έχουν αναγνωρισθεί. Έτσι, σύμφωνα με τις υποθέσεις του μοντέλου του δικτύου, κάθε κόμβος μαθαίνει πως είναι. ασφαλής σε σταθερό χρόνο αφού μπει στον νέο παλμό.

Σε προηγούμενη ενότητα εξηγήσαμε ότι η βασική αρχή λειτουργίας ενός συγχρονιστή είναι να μην επιτρέπει σε ένα κόμβο του συστήματος να προχωρά στο επόμενο βήμα του σύγχρονου αλγορίθμου πριν λάβει από τους γείτονές του που έχουν ολοκληρώσει το ίδιο βήμα με αυτόν τα απαραίτητα για την επεξεργασία στο επόμενο βήμα μηνύματα-αποτελέσματα. Είναι εμφανές ότι αυτό ισοδυναμεί με το να μην προχωρά στον επόμενο παλμό ένας κόμβος αν όλοι οι γείτονές του δεν είναι ασφαλείς όσον αφορά τον παλμό που έχει μόλις ολοκληρωθεί. Αυτή είναι η συνθήκη που ικανοποιούν και οι τρεις αλγόριθμοι που προτείνει ο Awerbuch. Η διαφορά τους

έγκειται στον διαφορετικό τρόπο με τον οποίο στον κάθε αλγόριθμο πληροφορείται ο κάθε κόμβος ότι όλοι οι γείτονές του είναι ασφαλείς.

2.2.3. Ο Συγχρονιστής α

Σύμφωνα με το πρωτόκολλο του συγχρονιστή α, κάθε κόμβος όταν μαθαίνει, μέσω των acknowledgements, πως είναι ασφαλής, το αναφέρει σε όλους τους γείτονές του. Ένας κόμβος μόλις πληροφορηθεί με αυτόν τον τρόπο πως όλοι οι γείτονές του είναι ασφαλείς προχωρεί στην εκτέλεση του επόμενου παλμού.

Ο συγχρονιστής α είναι αποδοτικός ως προς το χρόνο, αλλά όχι ως προς τα μηνύματα. Συγκεκριμένα, έστω ότι V είναι ο πληθάριθμος του συνόλου των κορυφών (κόμβων) στο γράφο, και E ο πληθάριθμος του αντίστοιχου συνόλου των ακμών. Από κάθε ακμή περνάνε ακριβώς δύο μηνύματα (ένα προς κάθε κατεύθυνση) ανά παλμό. Άρα σε κάθε παλμό έχουμε 2E μηνύματα, δηλαδή O(E) μηνύματα ανά παλμό ρολογιού. Κάθε γράφος V κόμβων έχει το πολύ V(V-1)/2 ακμές (πλήρης γράφος), άρα έχουμε $O(V^2)$ μηνύματα.

Οι καθυστερήσεις των μηνυμάτων είναι 1 χρονική μονάδα. Αφού οι επικοινωνίες σε κάθε παλμό γίνονται μεταξύ γειτονικών κόμβων, κάθε κόμβος σε σταθερό χρόνο καταλαβαίνει ότι είναι ασφαλής. Άρα για κάθε παλμό έχουμε χρόνο O(1).

2.2.4. Ο Συγχρονιστής β

Σύμφωνα με αυτόν τον αλγόριθμο η προώθηση στον επόμενο παλμό ή η καθυστέρηση μέχρι να είναι αυτή η εξέλιξη δυνατή δε γίνεται τοπικά στον κάθε κόμβο και η πρόοδος του κάθε κόμβου στην επεξεργασία δεν εξαρτάται μόνο από το αν οι γείτονές του είναι ασφαλείς όπως στον προηγούμενο αλγόριθμο. Αντίθετα, το σύστημα ολόκληρο προχωρά συλλογικά από παλμό σε παλμό και κάθε κόμβος ενεργοποιεί τον επόμενο παλμό μόνο όταν όλοι οι κόμβοι του συστήματος έχουν ολοκληρώσει το προηγούμενο βήμα και είναι όλοι ασφαλείς.

Για το συγχρονιστή αυτόν απαιτείται μια φάση αρχικοποίησης όπου εκλέγεται ένας κόμβος αρχηγός στο δίκτυο και κατασκευάζεται ένα γεννητικό δέντρο με ρίζα τον κόμβο-αρχηγό. Ο αρχηγός είναι αυτός που δίνει την εντολή σε όλους τους κόμβους του δικτύου για να ενεργοποιήσουν τον επόμενο παλμό, διαδίδοντας ένα συγκεκριμένο σήμα μέσω του δέντρου. Μετά το τέλος της επεξεργασίας για έναν παλμό ακολουθείται μια διαδικασία γνωστή με το όνομα convergecast για να ειδοποιηθεί τελικά ο αρχηγός πως όλοι οι κόμβοι του συστήματος είναι ασφαλείς. Συγκεκριμένα, μόλις ένας κόμβος γνωρίζει πως ο ίδιος είναι ασφαλής και έχει πληροφορηθεί πως το ίδιο ισχύει και για όλα τα «παιδιά» του στο δέντρο, αναφέρει την πληροφορία αυτή στον «πατέρα» του στο δέντρο. Η ροή αυτής της πληροφορίας ξεκινάει από τα φύλλα του δέντρου και καταλήγει στη ρίζα, στον αρχηγό. Όταν ο αρχηγός πληροφορηθεί τελικά πως όλοι οι κόμβοι του συστήματος είναι ασφαλείς διαδίδει ξανά μέσω του δέντρου εντολή στους κόμβους του συστήματος να προχωρήσουν στον επόμενο παλμό.

Ο συγχρονιστής β είναι αποδοτικός σε μηνύματα, αλλά όχι σε χρόνο. Συγκεκριμένα, το γεννητικό δένδρο που κατασκευάζεται έχει V-1 ακμές. Σε κάθε παλμό έχουμε 2 μηνύματα ανά ακμή. Άρα κινούνται O(V) μηνύματα ανά παλμό στη χειρότερη περίπτωση. Ο χρόνος που απαιτείται εξαρτάται από το ύψος του δένδρου. Το ύψος του δένδρου είναι το πολύ V-1, άρα στη χειρότερη περίπτωση έχουμε O(V) χρονικές μονάδες.

2.2.5. Ο Συγχρονιστής γ

Όπως ήδη αναφέρθηκε, ο αλγόριθμος αυτός αποτελεί ένα συνδυασμό των δύο προηγουμένων. Με λίγα λόγια πρόκειται για εφαρμογή του αλγορίθμου α σε σύνολα κόμβων στο καθένα από τα οποία εφαρμόζεται ο αλγόριθμος β.

Στην απαιτούμενη φάση αρχικοποίησης για τον συγχρονιστή γ κατασκευάζεται ένα δάσος από γεννητικά δέντρα. Το σύνολο των κόμβων δηλαδή διαμερίζεται σε ξένα μεταξύ τους υποσύνολα (clusters). Σε κάθε ένα από αυτά εκλέγεται αρχηγός και κατασκευάζεται με ρίζα τον αρχηγό ένα γεννητικό, ως προς τους κόμβους του συγκεκριμένου cluster, δέντρο (intracluster tree). Ακόμα για κάθε δύο γειτονικά clusters επιλέγεται ένα link (preferred link). Τα links αυτά θα εξυπηρετούν την επικοινωνία μεταξύ των clusters. Ο αλγόριθμος γ διακρίνεται σε δύο φάσεις. Στην πρώτη εφαρμόζεται ο αλγόριθμος β σε κάθε cluster. Μόλις ο αρχηγός ενός cluster μάθει ότι όλοι οι κόμβοι του cluster είναι ασφαλείς, ότι δηλαδή ο cluster του είναι ασφαλής διαδίδει την πληροφορία αυτή στους κόμβους του cluster του, μέσω του intracluster δέντρου, και στους αρχηγούς των γειτονικών clusters, μέσω των preferred links.

Στη δεύτερη φάση του αλγορίθμου οι κόμβοι ενός cluster περιμένουν έως ότου να γίνουν ασφαλείς όλοι οι γειτονικοί clusters, σαν να εφαρμόζεται δηλαδή ο αλγόριθμος α στους clusters. Μόλις γίνει γνωστό σε έναν cluster πως όλοι οι γειτονικοί του είναι ασφαλείς τότε σε αυτόν ενεργοποιείται ο επόμενος παλμός σύμφωνα πάλι με τον αλγόριθμο β, με μήνυμα δηλαδή του αρχηγού του cluster μέσω του intracluster δέντρου.

Σύμφωνα με τον αλγόριθμο αυτό οι τύποι μηνυμάτων που ανταλλάσσονται είναι :

- **PULSE**: ενεργοποιεί ένα νέο παλμό στους κόμβους ενός cluster.
- **SAFE**: ένας κόμβος δηλώνει ότι είναι ασφαλής.
- **CLUSTER_SAFE**: δηλώνει ότι όλος ο cluster είναι ασφαλής (διαδίδεται σε όλο το cluster και στα γειτονικά clusters μέσω των preferred links).
- **READY**: δηλώνει ότι ένας κόμβος είναι έτοιμος για τον επόμενο παλμό (διαδίδεται από τα φύλλα προς τον αρχηγό σε ένα cluster όταν ο κόμβος έχει λάβει όλα τα απαραίτητα μηνύματα READY από τα παιδιά του και όλα τα απαραίτητα CLUSTER_SAFE από τα preferred links που καταλήγουν σε αυτόν.

Έστω V το σύνολο των κόμβων του δικτύου, E το σύνολο των links που είτε ανήκουν σε κάποιο δένδρο είτε είναι preferred links, και H το μέγιστο ύψος από όλα τα δένδρα. Από κάθε link δένδρου περνάνε σε κάθε παλμό οπωσδήποτε τα μηνύματα PULSE, SAFE, CLUSTER_SAFE και READY μία φορά το καθένα. Από κάθε preferred link περνούν τα μηνύματα CLUSTER_SAFE και READY από μία φορά το καθένα. Άρα έχουμε O(E) μηνύματα ανά παλμό ρολογιού. Σε κάθε cluster ο χρόνος ενός παλμού είναι O(H). Ο χρόνος διάδοσης των CLUSTER_SAFE μηνυμάτων είναι O(H). Άρα ο συνολικός χρόνος είναι O(H).

3. Πρωτόκολλα Εκλογής Αρχηγού

3.1. Αλγόριθμος εκλογής αρχηγού με τον αλγόριθμο δένδρου

Κατά το σχεδιασμό των κατανεμημένων αλγορίθμων για ποικίλες εφαρμογές μερικά πολύ γενικά προβλήματα για δίκτυα διεργασιών προκύπτουν συχνά σαν υποέργα. Αυτά τα στοιχειώδη έργα περιλαμβάνουν την καθολική μετάδοση πληροφορίας (broadcast information), την απόκτηση καθολικού συγχρονισμού μεταξύ διεργασιών, την εκκίνηση της εκτέλεσης κάποιου γεγονότος σε κάθε διεργασία, κλπ. Αυτές οι εργασίες εκτελούνται συνεχώς περνώντας μηνύματα σύμφωνα με κάποιο προϋπάρχον σχήμα τοπολογίας που εμπλέκει τη συμμετοχή όλων των διεργασιών. Οι αλγόριθμοι αυτοί είναι θεμελιώδεις λύσεις που χρησιμοποιούνται σε πιο πολύπλοκους αλγορίθμους όπως αλγόριθμοι εκλογής αρχηγού, ανίχνευσης τερματισμού, αμοιβαίου αποκλεισμού, που μπορούν να λειτουργήσουν με τη βοήθεια αυτών των αλγορίθμων ανταλλαγής μηνυμάτων.

Στον αλγόριθμο διάδοσης του δένδρου θεωρούμε ότι η τοπολογία είναι δένδρο ή οποιαδήποτε αυθαίρετη τοπολογία για την οποία όμως ένα γεννητικό δένδρο είναι διαθέσιμο. Υποθέτουμε ότι όλα τα φύλλα του δένδρου εκκινούν τον αλγόριθμο. Κάθε διεργασία στέλνει ακριβώς ένα μήνυμα στον αλγόριθμο. Εάν μία διεργασία έχει λάβει ένα μήνυμα από κάθε γείτονά της εκτός ενός, η διεργασία στέλνει ένα μήνυμα προς τον υπολειπόμενο γείτονα. Εάν μία διεργασία έχει λάβει μήνυμα από όλους τους γείτονές της, αποφασίζει.

Εάν η τοπολογία του δικτύου είναι δένδρο ή υπάρχει διαθέσιμο γεννητικό δένδρο, μπορεί να γίνει εκλογή αρχηγού με χρήση του αλγορίθμου διάδοσης κύματος του δένδρου (wave tree algorithm). Στον αλγόριθμο του δένδρου είναι απαραίτητο τουλάχιστον όλα τα φύλλα να είναι αρχικοποιητές του αλγορίθμου. Για να υπάρξει πρόοδος στον αλγόριθμο στην περίπτωση που μερικές μόνο από τις διεργασίες είναι αρχικοποιητές, προστίθεται μία φάση αφύπνισης (wakeup phase). Η διεργασία που θέλει να ξεκινήσει την εκλογή διοχετεύει στο δένδρο ένα μήνυμα wakeup με σκοπό αυτό να φτάσει σε όλες τις διεργασίες. Η λογική μεταβλητή ws χρησιμοποιείται για να κάνει μία διεργασία να στείλει ένα μήνυμα wakeup το πολύ μία φορά, και η μεταβλητή wr χρησιμοποιείται για τη μέτρηση των μηνυμάτων wakeup που έχουν ληφθεί από μία διεργασία. Όταν μία διεργασία έχει λάβει ένα μήνυμα wakeup από κάθε κανάλι επικοινωνίας (άρα από όλους τους γείτονές της), ξεκινά τον αλγόριθμο του διάδοσης του δένδρου για τον υπολογισμό της μικρότερης ταυτότητας ώστε κάθε διεργασία να αποφασίσει για την μικρότερη ταυτότητα στο δένδρο που θα παίξει το ρόλο του αρχηγού. Όταν μία διεργασία αποφασίσει γνωρίζει την ταυτότητα του αρχηγού. Εάν αυτή η ταυτότητα ισούται με την ταυτότητά της, γίνεται αρχηγός, αλλιώς χάνει την εκλογή.

Παρακάτω δίνεται ο αλγόριθμος εκλογής αρχηγού με τον αλγόριθμο του δένδρου σε μορφή ψευδοκώδικα. Οι μεταβλητές που χρησιμοποιούνται στον παρακάτω κώδικα αναφέρονται σε μία διεργασία p η οποία και τρέχει τον αλγόριθμο.

```
var   ws: boolean                    init false;
      wr: integer                    init 0;
      rec[q]: Boolean for each q ∈ Neigh  init false;
      v: P                           init p;
      state: (sleep, leader, lost)        init sleep;
```

```
begin if p is initiator then
      begin ws:=true;
            forall q ∈ Neigh do send <wakeup> to q
      end;
      while wr < #Neigh do
            begin receive <wakeup>;
                  wr:=wr+1;
                  if not ws then
                        begin ws:=true;
                              forall q ∈ Neigh do send <wakeup> to q
                        end
            end;
      /* Now start the tree algorithm */
      while #{q: ¬rec[q]} >1 do
            begin receive <tok,r> from q;
                  rec[q]:=true;
                  v:=min(v,r)
            end;
      send <tok, v> to q0 with ¬rec[q0];
      receive <tok,r> from q0;
      v:=min(v,r);
      if v=p then state:=leader else state:=lost;
      forall q ∈ Neigh, q≠q0 do send <tok,v> to q
end
```

Ο παραπάνω αλγόριθμος λύνει το πρόβλημα της εκλογής χρησιμοποιώντας O(N) μηνύματα και O(D) χρονικές μονάδες, όπου N είναι ο αριθμός των κόμβων και D το ύψος του δένδρου.

Απόδειξη. Από κάθε ακμή του δένδρου στέλνονται δύο <wakeup> και δύο <tok,r> μηνύματα. Αφού ο αριθμός των ακμών στο δένδρο είναι N-1, τότε η πολυπλοκότητα των μηνυμάτων είναι 4N-4, δηλαδή O(N). Μέσα σε D χρονικές μονάδες από τη στιγμή που ξεκινά τον αλγόριθμο η πρώτη διεργασία, κάθε διεργασία έχει στείλει <wakeup> μηνύματα, έτσι μέσα σε D+1 χρονικές μονάδες κάθε διεργασία έχει ξεκινήσει τον αλγόριθμο κύματος. Είναι εύκολο να διαπιστώσουμε ότι η πρώτη απόφαση λαμβάνει χώρα το πολύ D χρονικές μονάδες μετά την εκκίνηση του κύματος και η τελευταία το πολύ D χρονικές μονάδες μετά την πρώτη απόφαση. Άρα συνολικά έχουμε 3D+1 χρονικές μονάδες, δηλαδή O(D).

3.2. Ο αλγόριθμος του LeLann

Ο LeLann ήταν εκείνος που πρωτοπαρουσίασε αλγόριθμο για εκλογή αρχηγού σε κατανεμημένο σύστημα κυκλικά τοποθετημένων (σε τοπολογία δακτυλίου μονής κατεύθυνσης) επώνυμων επεξεργαστών. Αφορμή ήταν ο τρόπος αντικατάστασης ενός χαμένου token, κατά την διάρκεια λειτουργίας ενός δακτυλίου επεξεργαστών. Σύμφωνα με τον αλγόριθμό του, κάθε επεξεργαστής – αρχικοποιητής εκπέμπει μήνυμα με τον αριθμό του που πηγαίνει σε όλους τους άλλους. Έτσι, μετά από λίγο κάθε επεξεργαστής – αρχικοποιητής έχει μια πλήρη λίστα των αριθμών όλων των άλλων επεξεργαστών. Αυτό που του μένει είναι να βρει το μέγιστο (ή το ελάχιστο) της λίστας του και αν αυτός ο μέγιστος ταυτίζεται με τον αριθμό του, αυτός είναι και ο εκλεγμένος νικητής. Θεωρείται ότι τα κανάλια είναι FIFO και ότι κάθε αρχικοποιητής πρέπει να παράγει το δικό του token πριν λάβει κάποιο άλλο token από οποιονδήποτε άλλο επεξεργαστή – αρχικοποιητή. Αρκετά απλός σε υλοποίηση

και σκέψη αλγόριθμος που απαιτεί γραμμικό χρόνο για τις επικοινωνίες, αλλά υψηλό αριθμό διακινούμενων μηνυμάτων και διακινούμενων bits.

Παρακάτω φαίνεται σε μορφή ψευδοκώδικα ο αλγόριθμος:

```
var   List_p : set of P  init{p};
      state_p;
BEGIN
      If p is initiator then /* εάν ο επεξεργαστής είναι
αρχικοποιητής */
            Begin
                  state_p:=candidate; /* δηλώνει υποψήφιος και  */
                  send <tok,p> to Next_p; /* στέλνει την ταυτότητά του
            */
                  receive <tok,q>;
                  while q≠p do
                        begin /* τα μηνύματα των άλλων τα τοποθετεί
                  */
                              List_p:=List_p ∪ {q}; /* στη λίστα του */
                              send <tok,q> to Next_p; /* τα προωθεί */
                              receive <tok,q>
                        end;
                  if p=max(List_p) then state_p:=leader   /* συνθήκη
εκλογής */
                  else state_p:=lost
            end;
      else repeat receive <tok,q>;  /* αν δεν είναι αρχικοποιητής */
            send <tok,q> to Next_p;  /* απλώς προωθεί τα μηνύματα */
            if state_p=sleep then state_p:=lost
      until false
end
```

Ο αλγόριθμος του LeLann λύνει το πρόβλημα εκλογής αρχηγού σε δακτύλιο χρησιμοποιώντας $O(N^2)$ μηνύματα και σε $O(N)$ χρόνο στη χειρότερη περίπτωση.

Απόδειξη: Υπάρχουν το πολύ N διαφορετικά tokens που κυκλοφορούν στο δακτύλιο, όπου το καθένα κάνει N βήματα. Άρα η πολυπλοκότητα μηνυμάτων είναι $O(N^2)$. Σε N-1 χρονικές μονάδες το πολύ από τη στιγμή που ο πρώτος αρχικοποιητής έχει στείλει το token του, κάθε άλλος αρχικοποιητής έχει στείλει το δικό του, και κάθε αρχικοποιητής λαμβάνει το δικό του token N χρονικές μονάδες μετά. Άρα ο αλγόριθμος τερματίζει μετά από το πολύ 2N-1 χρονικές μονάδες.

3.3. Ο αλγόριθμος των Chang & Roberts

Ο αλγόριθμος των Chang & Roberts βελτιώνει τον αλγόριθμο του LeLann, απαιτώντας, κατά μέσο όρο, λιγότερα μηνύματα σε γραμμικό χρόνο. Εξακολουθεί όμως να διατηρεί τον αριθμό μηνυμάτων για τη χειρότερη περίπτωση του αλγορίθμου LeLann. Έχει τα προτερήματα της εύκολης υλοποίησης, της επεκτασιμότητας σε άλλες τοπολογίες και της πολύ καλής απόδοσης κάτω από ορισμένες συνθήκες χρονισμού του δικτύου.

Χαρακτηριστικά του αλγορίθμου είναι η άγνοια του συνολικού αριθμού των επεξεργαστών που παίρνουν μέρος στην εκλογή, η κίνηση μηνυμάτων προς τη μια κατεύθυνση μόνο και η δυνατότητα μη ταυτόχρονης αρχής του αλγορίθμου απ' όλους τους επεξεργαστές. Μάλιστα, αυτή η τελευταία περίπτωση, που είναι και η πιο πιθανή, δίνει καλύτερα αποτελέσματα από εκείνα του σύγχρονου ξεκινήματος.

Ο αλγόριθμος προσπαθεί, κατά τη διάρκεια εκτέλεσής του, να μαντέψει και επιλεκτικά να καταστρέψει τα μηνύματα των επεξεργαστών που δεν μπορούν να είναι νικητές. Ένας επεξεργαστής δεν μπορεί να είναι νικητής αν βρεθεί τουλάχιστον ένας άλλος με αριθμό μεγαλύτερο του δικού του. (Η παραλλαγή του αλγορίθμου που περιγράφουμε εκλέγει σαν αρχηγό τον επεξεργαστή με το μεγαλύτερο νούμερο).

Περιγραφικά, ο αλγόριθμος εργάζεται ως εξής:

Κάθε επεξεργαστής υποτίθεται ότι ξέρει το δικό του νούμερο. Όταν αντιληφθεί ότι γίνεται διαδικασία εκλογής - κι αυτό μπορεί να γίνει με δύο τρόπους: είτε να το καταλάβει από μόνος του, είτε να ειδοποιηθεί από το γείτονά του - παράγει ένα μήνυμα με το νούμερό του και το προωθεί αριστερά στο γείτονά του. Ένας επεξεργαστής που παίρνει κάποιο μήνυμα συγκρίνει τον αριθμό του με το δικό του νούμερο. Αν το δικό του νούμερο είναι μικρότερο, τότε υπάρχει πιθανότητα το μήνυμα που του ήρθε να είναι το μήνυμα του νικητή, το προωθεί λοιπόν προς τα αριστερά. Αν αντίθετα, το μήνυμα έχει μικρότερο αριθμό από το δικό του, τότε αποκλείεται να μεταφέρει τον αριθμό του νικητή και το μήνυμα αγνοείται. Αν τέλος ο αριθμός του μηνύματος και ο δικός του είναι ίδιοι, τότε καταλαβαίνει πως αυτός είναι ο επεξεργαστής με το μεγαλύτερο νούμερο στο σύστημα, αυτός εκλέγεται αρχηγός σ' αυτό το γύρο και η διαδικασία τερματίζεται.

Ο αλγόριθμος πραγματικά βρίσκει το μοναδικό μέγιστο αριθμό επεξεργαστή στο σύστημα. Κι αυτό, γιατί η μοναδική φορά κίνησης των μηνυμάτων και η τοπολογία του δακτυλίου αναγκάζουν ένα μήνυμα να περάσει απ' όλους τους άλλους επεξεργαστές πριν επιστρέψει στον εκπομπό του. Η φύση του αλγορίθμου επιτρέπει μόνο στο μήνυμα με το μεγαλύτερο αριθμό να συμπληρώσει έναν πλήρη κύκλο χωρίς να κοπεί από κάποιον από τους ενδιάμεσους επεξεργαστές. Έτσι, μόνο ο επεξεργαστής με το μεγαλύτερο νούμερο δέχεται πίσω το μήνυμά του και ανακηρύσσεται ο μοναδικός αρχηγός της διαδικασίας εκλογής.

Αναφέραμε στην εισαγωγή πως κάθε επεξεργαστής σε ένα κατανεμημένο σύστημα μπορεί να μοντελοποιηθεί σαν ένα αυτόματο που ανάλογα με την κατάστασή του και το μήνυμα που του έρχεται, μεταβαίνει σε καινούργια κατάσταση. Σαν στιγμιότυπο του όλου συστήματος μπορούμε να θεωρήσουμε το διάνυσμα των καταστάσεων των επεξεργαστών του.

Κανόνας 1: Ένας sleeping κόμβος:

1.1 Μπορεί να αφυπνιστεί αυτόματα, από μόνος του, να γίνει awake και να στείλει μήνυμα με τον αριθμό του προς τα αριστερά.

l.2 Αν αφυπνιστεί από το μήνυμα άλλου κόμβου, γίνεται awake και αν το νούμερό του είναι μικρότερο από αυτό του μηνύματος που τον ξύπνησε, προωθεί το μήνυμα, διαφορετικά αγνοεί το μήνυμα αφύπνισης και στέλνει το δικό του.

Κανόνας 2: Ένας awake κόμβος:

2.l Αν πάρει μήνυμα με αριθμό μικρότερο του δικού του, αγνοεί το μήνυμα.

2.2 Αν πάρει μήνυμα με αριθμό μεγαλύτερο του δικού του, το προωθεί.

2.3 Αν πάρει μήνυμα με αριθμό ίδιο με το δικό του, γίνεται elected και η διαδικασία της εκλογής τερματίζεται.

Από αλγοριθμική άποψη το πρωτόκολλο των Chang & Roberts έχει ως εξής:

Node i :

```
On receiving message j :
CASE status = sleeping
      BEGIN
            IF i > j THEN send i
            ELSE IF i < j THEN send j
            status := awake
      END
CASE status = awake
      BEGIN
            IF i > j THEN skip
            ELSE IF i < j THEN send j
            ELSE IF i = j THEN status := elected
      END
```

Κατά το τέλος του αλγορίθμου ο επεξεργαστής που τελικά εκλέχτηκε στέλνει κάποιο μήνυμα τερματισμού πληροφορώντας τους υπόλοιπους επεξεργαστές του δικτύου που πήραν μέρος στη διαδικασία της εκλογής για το τέλος της.

Ο αλγόριθμος των Chang & Roberts λύνει το πρόβλημα εκλογής αρχηγού σε δακτύλιο χρησιμοποιώντας $\Theta(N^2)$ μηνύματα στη χειρότερη περίπτωση.

Απόδειξη: Χρησιμοποιούνται το πολύ N tokens, όπου κάθε token κάνει το πολύ N βήματα. Άρα έχουμε $O(N^2)$ μηνύματα. Υποθέτουμε ακόμη ότι οι κόμβοι τοποθετούνται με αύξουσα σειρά πάνω στο δακτύλιο, οπότε όλα τα tokens «κόβονται» από τον κόμβο 0, πλην του δικού του. Άρα κάθε token i προωθείται κατά N-i βήματα. Άρα το σύνολο των μηνυμάτων είναι (1/2)N(N+1). Άρα έχουμε $\Omega(N^2)$ μηνύματα.

Ο αλγόριθμος των Chang & Roberts απαιτεί O(NlogN) μηνύματα στη μέση περίπτωση, θεωρώντας ότι όλοι οι κόμβοι είναι αρχικοποιητές.

Απόδειξη: Θα υπολογίσουμε το μέσο αριθμό μηνυμάτων από όλες τις διαφορετικές κυκλικές διατάξεις των N κόμβων. Έστω ότι είναι s ο κόμβος με τη μικρότερη ταυτότητα και p_i ο κόμβος που βρίσκεται i βήματα πριν τον s στο δακτύλιο. Υπάρχουν (N-1)! διαφορετικές διατάξεις στο δακτύλιο (εξαιρείται ο s). Θα υπολογίσουμε τον αριθμό των μηνυμάτων του p_i για όλες τις διατάξεις και θα αθροίσουμε για όλα τα i.

Το μήνυμα του s θα περάσει N φορές σε κάθε διάταξη και άρα N(N-1)! φορές συνολικά. Το μήνυμα του i θα περάσει από i κόμβους το πολύ πριν το «κόψει» ο s. Έστω $A_{i,k}$ ο αριθμός των κυκλικών διατάξεων όπου το μήνυμα του p_i περνάει ακριβώς k φορές. Ο συνολικός αριθμός των μηνυμάτων του p_i είναι

$$\sum_{k=1}^{i}(k \times A_{i,k})$$

Το μήνυμα του p_i περνάει ακριβώς i φορές στο (1/i)(N-1)! των διατάξεων. Παρόμοια, το ίδιο μήνυμα περνάει ακριβώς k φορές ($k \le i$) στο (1/k)(N-1)! των διατάξεων. Αυτό σημαίνει ότι το μήνυμα θα περάσει τουλάχιστον k φορές, αλλά όχι τουλάχιστον k+1 φορές. Οπότε ο αριθμός των διατάξεων που αυτό συμβαίνει είναι (για $k<i$):

$$A_{i,k} = \frac{1}{k}(N-1)! - \frac{1}{k+1}(N-1)! = \frac{1}{k(k+1)}(N-1)!$$

Ο συνολικός αριθμός λοιπόν είναι:

$$\sum_{k=1}^{i-1} k\left(\frac{1}{k(k+1)}(N-1)!\right) + i\frac{1}{i}(N-1)! = \left(\sum_{k=1}^{i}\frac{1}{k}\right)(N-1)!$$

Ο πρώτος όρος στο παραπάνω γινόμενο λέγεται i-οστός αρμονικός αριθμός και συμβολίζεται H_i. Ισχύει η ταυτότητα:

$$\sum_{i=1}^{m} H_i = (m+1)H_m - m$$

Αθροίζοντας λοιπόν όλα τα περάσματα των μηνυμάτων για να πάρουμε το συνολικό αριθμό και εφαρμόζοντας την παραπάνω ταυτότητα έχουμε ότι:

$$\sum_{i=1}^{N-1}[H_i(N-1)!] = (N \times H_{N-1} - (N-1)) \times (N-1)!$$

Αθροίζοντας τα N*(N-1)! περάσματα φτάνουμε σε ένα τελικό αριθμό περασμάτων $(N*H_{N-1}+1)(N-1)!=(N*H_N)(N-1)!$. Ισχύει ότι $(N*H_N) \cong 0.69N\log N$. Άρα αποδείχτηκε.

3.4. Ο αλγόριθμος των Peterson/Dolev-Klawe-Rodeh

Ο αλγόριθμος των Chang & Roberts επιτυγχάνει O(NlogN) (N ο αριθμός των κόμβων) πολυπλοκότητα μηνυμάτων στη μέση περίπτωση, αλλά στη χειρότερη περίπτωση όμως επιτυγχάνει $O(N^2)$. Ο αλγόριθμος των Peterson/Dolev-Klawe-Rodeh επιτυγχάνει πολυπλοκότητα μηνυμάτων O(NlogN) στη χειρότερη περίπτωση βελτιώνοντας τον αλγόριθμο των Chang & Roberts για την εκλογή αρχηγού σε δακτυλίους μονής κατεύθυνσης, όπου κάθε κόμβος έχει τη δική του μοναδική ταυτότητα.

Ο αλγόριθμος βασίζεται στην παρακάτω ιδέα. Αρχικά κάθε ταυτότητα είναι ενεργή, αλλά σε κάθε γύρο κάποιες ταυτότητες γίνονται παθητικές, όπως θα δείξουμε στη συνέχεια. Σε ένα γύρο μία ενεργή ταυτότητα συγκρίνει τον εαυτό της με τις ενεργές ταυτότητες των δύο γειτονικών της κόμβων, που βρίσκονται η μία κατά τη φορά και η άλλη αντίθετα από τη φορά των δεικτών του ρολογιού. Υποθέτοντας ότι εκλέγεται η μικρότερη ταυτότητα, αν μία ενεργή ταυτότητα αποτελεί τοπικό ελάχιστο (μετά την παραπάνω σύγκριση), τότε παραμένει ενεργή και προχωρά στον επόμενο γύρο, διαφορετικά γίνεται παθητική. Έτσι μετά από logN γύρους θα έχει απομείνει στο δακτύλιο μόνο μία ενεργή ταυτότητα και αυτή κερδίζει την εκλογή.

Η ιδέα αυτή μπορεί να εφαρμοστεί απ' ευθείας σε δακτυλίους που είναι διπλής κατεύθυνσης. Στους μονής κατεύθυνσης όμως δακτυλίους τα μηνύματα μπορούν να κινούνται μόνο κατά τη μία φορά (π.χ. αυτή των δεικτών του ρολογιού), πράγμα που κάνει δύσκολη την απόκτηση της ταυτότητας που βρίσκεται κατά την κατεύθυνση κίνησης των μηνυμάτων. Ας υποθέσουμε ότι έχουμε ένα δακτύλιο όπου βρίσκονται μεταξύ άλλων τρεις ενεργές σε σειρά ταυτότητες r,q,p. Τότε η q μπορεί να λάβει το μήνυμα της r, αλλά δεν μπορεί να λάβει το μήνυμα της p γιατί ο δακτύλιος είναι μονής κατεύθυνσης και δεν πρέπει να περιμένει ένα ολόκληρο κύκλο για να το παραλάβει. Για να γίνει όμως η σύγκριση η q στέλνει την ταυτότητά της στην p και η r δεν στέλνει απλώς την ταυτότητά της στην q, αλλά η q την προωθεί στην p, ώστε η p να κάνει τη σύγκριση. Οι διεργασίες που χάνουν την εκλογή και γίνονται παθητικές, απλώς προωθούν τα μηνύματα που λαμβάνουν. Εάν μία διεργασία λάβει μία ταυτότητα που είναι ίση με τη δική της τότε ανακηρύσσεται αρχηγός, ενώ αν λάβει μία ταυτότητα διαφορετική από τη δική της συγκρίνει την ταυτότητα αυτή με τη δική της και με το τρέχον ελάχιστο που έχει κρατήσει από τις προηγούμενες συγκρίσεις. Αν η παραληφθείσα ταυτότητα είναι η μικρότερη, τότε η διεργασία γίνεται παθητική και κρατά το νέο ελάχιστο.

Παρακάτω δίνεται ο αλγόριθμος με τη μορφή ψευδοκώδικα:

```
var   ci_p : P init p; /* Current identity of p */
      acn_p : P init undefined; /* id of anticlockwise active neighbor
*/
      win_p : P init undefined; /* id of winner */
      state_p : (active, passive, leader, lost) init active;

begin
      if p is initiator then state_p:= active else state_p:=passive;
      while win_p=undefined do {
            if state_p=active then {
                  send <one,ci_p>;
                  receive <one,q>;
                  acn_p:=q;
                  if acn_p=ci_p then  {/* acn_p is the minimum */
                        send <smal,acn_p>;
                        win_p:=acn_p;
                        receive <smal,q>;
                  }
                  else { /* acn_p is current id of neighbor */
                        send <two,acn_p>;
                        receive <two,q>;
                        if acn_p < ci_p and acn_p < q then ci_p:=acn_p;
                        else state_p:=passive;
                  }
            }
            else { /* state_p=passive */
                  receive <one,q>;
                  send <one,q>;
                  receive m;
                  send m; /* m is either <two,q> or <smal, q> */
                  if m is a <smal,q> message then win_p:=q
            }
      }
      if p=win_p then state_p:=leader else state_p:=lost
}
```

Ο αλγόριθμος λύνει το πρόβλημα της εκλογής σε μονής κατεύθυνσης δακτυλίους χρησιμοποιώντας O(NlogN) μηνύματα.

Απόδειξη: Λέμε ότι μία διεργασία βρίσκεται στον γύρο i όταν εκτελεί τη βασική επανάληψη του αλγορίθμου για i-οστή φορά. Οι γύροι δεν είναι σφαιρικά συγχρονισμένοι. Είναι πιθανό μία διεργασία να βρίσκεται αρκετούς γύρους μπροστά από μία άλλη διεργασία που βρίσκεται σε άλλο τμήμα του δακτυλίου. Αλλά, αφού κάθε διεργασία στέλνει και λαμβάνει ακριβώς δύο μηνύματα σε κάθε γύρο και τα κανάλια είναι FIFO, ένα μήνυμα λαμβάνεται συνεχώς στον ίδιο γύρο που στέλνεται. Στον πρώτο γύρο όλοι οι αρχικοποιητές είναι ενεργοί και κάθε ενεργή διεργασία διατηρεί μια διαφορετική «τρέχουσα ταυτότητα».

Για να συνεχίσουμε την απόδειξη πρέπει να δείξουμε ότι εάν ο γύρος i ξεκινά με k>1 ενεργές διεργασίες και κάθε διεργασία διατηρεί ένα διαφορετικό ci, τότε τουλάχιστον μία και το πολύ k/2 διεργασίες επιβιώνουν στο γύρο. Στο τέλος του γύρου πάλι όλες οι τρέχουσες ταυτότητες των ενεργών διεργασιών είναι διαφορετικές και περιλαμβάνουν τη μικρότερη ταυτότητα.

Το παραπάνω αποδεικνύεται ως εξής: με την ανταλλαγή των <one, q> μηνυμάτων, τα οποία διαδίδονται επίσης από τις παθητικές διεργασίες, κάθε ενεργή διεργασία αποκτά την τρέχουσα ταυτότητα του πρώτου ενεργού γείτονα προς την αντίθετη

φορά, η οποία σε όλες τις περιπτώσεις είναι διαφορετική από τη δική της. Ακολούθως, κάθε ενεργή διεργασία συνεχίζει το γύρο με την ανταλλαγή των <two,q> μηνυμάτων, με τα οποία κάθε ενεργή διεργασία αποκτά την τρέχουσα ταυτότητα του δεύτερου ενεργού γείτονα προς την αντίθετη φορά. Κάθε ενεργή διεργασία πλέον κατέχει μια διαφορετική τιμή για τη μεταβλητή acn, η οποία υπονοεί ότι οι επιζώντες του γύρου έχουν όλοι διαφορετική ταυτότητα στο τέλος του γύρου. Τουλάχιστον η ταυτότητα που ήταν η μικρότερη στην αρχή του γύρου επιβιώνει, έτσι υπάρχει τουλάχιστον μία διεργασία που επέζησε. Μία ταυτότητα επόμενη σε ένα τοπικό ελάχιστο δεν είναι τοπικό ελάχιστο, που σημαίνει ότι ο αριθμός των επιζώντων είναι το πολύ k/2.

Επίσης, αυτό σημαίνει ότι θα υπάρχει ένας γύρος με αριθμό $\leq \lfloor \log N \rfloor + 1$, που ξεκινά με ακριβώς μία ενεργή ταυτότητα, που είναι η μικρότερη ταυτότητα κάποιου αρχικοποιητή. Εάν ένας γύρος ξεκινήσει με ακριβώς μία ενεργή διεργασία p, με τρέχουσα ταυτότητα ci_p, ο αλγόριθμος ολοκληρώνεται μετά από αυτό το γύρο με $win_p=ci_p$ για κάθε q. Κι αυτό συμβαίνει γιατί το μήνυμα <one,ci_p> της p διεργασίας διαδίδεται από όλες τις διεργασίες και τελικά λαμβάνεται από την ίδια την p. Η διεργασία p αποκτά $acn_p=ci_p$ και στέλνει ένα μήνυμα <smal,acn_p> στο δακτύλιο, το οποίο προκαλεί κάθε διεργασία q να βγει από τη βασική επανάληψη με $win_p=acn_p$.

Ο αλγόριθμος τελειώνει σε κάθε διεργασία και όλες οι διεργασίες συμφωνούν στην ταυτότητα του αρχηγού. Υπάρχουν λοιπόν το πολύ $\lfloor \log N \rfloor + 1$ γύροι, σε κάθε ένα από τους οποίους ανταλλάσσονται ακριβώς 2N μηνύματα, το οποίο αποδεικνύει ότι η πολυπλοκότητα μηνυμάτων φράσσεται από το 2NlogN+O(N).

3.5. Ο αλγόριθμος των Itai & Rodeh για την εκλογή αρχηγού σε δακτύλιο ανώνυμων επεξεργαστών

Το πρόβλημα της εκλογής αρχηγού διαφοροποιείται στην περίπτωση που δεν υπάρχουν διακεκριμένες ταυτότητες στο δίκτυο. Τότε οι ταυτότητες των επεξεργαστών δεν μπορούν να χρησιμοποιηθούν και να δώσουν αξιόπιστα αποτελέσματα αφού είναι πολύ πιθανό να υπάρχουν επεξεργαστές με την ίδια μεγαλύτερη (ή μικρότερη) ταυτότητα στο δίκτυο. Τότε διακρίνουμε την ύπαρξη μιας συμμετρίας στο δίκτυο και το πρόβλημα που καλούμαστε να λύσουμε αναφέρεται στη διάσπαση της συμμετρίας του δικτύου. Θα περιγραφεί το πρωτόκολλο των Itai & Rodeh που λύνει το πρόβλημα σ' ένα ασύγχρονο δακτύλιο χρησιμοποιώντας O(NlogN) μηνύματα και O(N) χρονικές μονάδες. Ο αλγόριθμος κινείται στα πλαίσια εκείνου των Chang & Roberts με την κύρια διαφορά ότι οι επεξεργαστές δεν είναι αριθμημένοι με μοναδικό τρόπο. Κι εδώ ουσιαστικά αναζητείται ο επεξεργαστής με το μεγαλύτερο (ή το μικρότερο) αριθμό και τα μηνύματα μπορούν να κινηθούν προς μία και μόνο κατεύθυνση.

Αρχικά οι επεξεργαστές διαλέγουν για ταυτότητές τους κάποιους τυχαίους αριθμούς από το σύνολο {1,…,n} και στη συνέχεια γίνεται η εκλογή του μεγαλύτερου με ταυτόχρονο έλεγχο της μοναδικότητά του. Αν κάτι τέτοιο δεν είναι αλήθεια ο αλγόριθμος επαναλαμβάνεται. Για τον έλεγχο της μοναδικότητας είναι απαραίτητη η γνώση από όλους τους επεξεργαστές του μεγέθους του δακτυλίου, ενώ οι επαναλήψεις του αλγορίθμου σημειώνονται από τον αριθμό της φάσης που όπως θα δούμε μεταφέρεται και από τα μηνύματα του αλγορίθμου.

Εξ αιτίας του ασυγχρονισμού του δικτύου και της ομοιομορφίας των επεξεργαστών απέναντι στον αλγόριθμο, παρουσιάζονται δύο προβλήματα που καλούμαστε να λύσουμε για να μπορέσει να λειτουργήσει σωστά ο αλγόριθμος:

1. Κανένας επεξεργαστής δεν μπορεί να αποφασίσει από μόνος του να σταματήσει την προσπάθεια για την εκλογή του, αν δε σιγουρευτεί πρώτα πως υπάρχει τουλάχιστον ένας ακόμα που προσπαθεί να εκλεγεί. Σε αντίθετη περίπτωση μπορούμε να φτάσουμε σε μια κατάσταση όπου κανείς πια δεν ενδιαφέρεται για την εκλογή και ο αλγόριθμος πέφτει σε αδιέξοδο. Στην περίπτωσή μας το πρόβλημα αντιμετωπίζεται ως εξής: σε κάθε φάση του αλγορίθμου βγαίνουν από το παιχνίδι της εκλογής οι επεξεργαστές που καταλαβαίνουν πως είναι αδύνατον να εκλεγούν, επειδή ακριβώς μαθαίνουν πως υπάρχει ισχυρότερος υποψήφιος. Αν στη συγκεκριμένη φάση δεν υπάρξει μοναδικός νικητής συνεχίζουν μόνο οι επεξεργαστές που παρέμειναν ενεργοί στο τέλος της προηγούμενης φάσης. Δεν είναι δυνατόν να παραιτηθούν όλοι οι επεξεργαστές σε μια φάση – ο μεγαλύτερος ή οι μεγαλύτεροι, δεν έχουν λόγο να το κάνουν και πάντα θα υπάρχει ένας τουλάχιστον με μέγιστο νούμερο σε κάθε φάση. Έτσι δεν θα υπάρξει αδιέξοδο.

2. Ο επεξεργαστής δεν μπορεί βασισμένος μόνο στην ταυτότητα που κουβαλάει μαζί του ένα μήνυμα να διακρίνει με σιγουριά τα δικά του μηνύματα, αφού η αρίθμηση των επεξεργαστών δεν είναι εγγυημένα μοναδική. Το πρόβλημα αυτό λύνεται αν οι επεξεργαστές γνωρίζουν το συνολικό τους αριθμό στο δακτύλιο. Κάθε μήνυμα συνοδεύεται από ένα μετρητή που έχει την τιμή 1 στο ξεκίνημα του μηνύματος και αυξάνεται κάθε φορά που το μήνυμα προωθείται από κάποιον επεξεργαστή. Ένας επεξεργαστής που θα δεχτεί το μήνυμα με το μετρητή του ίσο με N γνωρίζει με βεβαιότητα πως είναι δικό του μήνυμα που συμπλήρωσε κύκλο και επέστρεψε σε αυτόν.

Μετά από αυτές τις δύο παρατηρήσεις μπορούμε να περιγράψουμε τον αλγόριθμό μας. Όταν ένας κόμβος καταλάβει πως γίνεται εκλογή στο δακτύλιο και αυτό μπορεί να γίνει είτε από δικό του συμπέρασμα είτε επειδή ειδοποιήθηκε από κάποιον άλλο, διαλέγει σαν ταυτότητά του έναν αριθμό από το σύνολο {1,…,N} και στέλνει προς μία ορισμένη κατεύθυνση, την κατεύθυνση κίνησης των μηνυμάτων, το παρακάτω μήνυμα: (ph_v, id_v, count, unique). Το ph_v είναι ο αριθμός της φάσης στην οποία βρίσκεται ο αλγόριθμος και στην αρχή έχει την τιμή 1, το id_v είναι η ταυτότητα του επεξεργαστή που διάλεξε, count είναι ο μετρητής στον οποίο αναφερθήκαμε παραπάνω και τέλος unique είναι μια λογική μεταβλητή που σηματοδοτεί την ύπαρξη ή όχι κάποιου μοναδικού νικητή. Σκοπός του μηνύματος είναι η επιστροφή του στον εκπομπό του και η ανακήρυξη του τελευταίου σε νικητή. Αυτό θα συμβεί μόνον όταν ικανοποιούνται δύο συνθήκες: ο id_v είναι ο μέγιστος αριθμός του δακτυλίου και είναι και ο μοναδικός. Κατά την πορεία του στο δακτύλιο το μήνυμα περνάει από άλλους επεξεργαστές που το προωθούν ή όχι. Αν συναντήσει κάποιο κόμβο u με (ph_u, id_u) > (ph_v, id_v) η πορεία του μηνύματος διακόπτεται, γιατί είτε έχει μικρότερη φάση, δηλαδή είναι άχρηστο για την τωρινή φάση του αλγορίθμου, είτε ο κόμβος που το έστειλε έχει μικρότερη ταυτότητα και άρα δεν μπορεί να εκλεγεί νικητής, είτε συμβαίνουν και τα δύο μαζί. Αν συμβαίνει το αντίθετο, τότε ο κόμβος u γίνεται μη ενεργός, δηλαδή (ph_u, id_u) < (ph_v, id_v), σταματά δηλαδή από εδώ και πέρα να συναγωνίζεται για την εκλογή, ενημερώνονται οι τιμές της φάσης και της ταυτότητά του και το μήνυμά μας προωθείται. Αν τώρα (ph_u, id_u) = (ph_v, id_v) τότε το μήνυμα ανήκει στη σωστή φάση και επιπλέον έχει ανακαλύψει έναν κόμβο που έχει τον ίδιο αριθμό με τον εκπομπό του. Ο τελευταίος έχει ήδη αποτύχει να κερδίσει την εκλογή, τουλάχιστον στη φάση που διανύει τώρα ο αλγόριθμος. Το μήνυμα προωθείται αλλά

η μεταβλητή unique αλλάζει σε false. Αν τελικά το μήνυμα επιστρέψει στον εκπομπό του τότε αυτός θα μάθει πως έχει μεν το μεγαλύτερο αριθμό στο δακτύλιο, αλλά υπάρχει τουλάχιστον ένας ακόμη ανταγωνιστής του με τον ίδιο αριθμό. Αυτό θα τον οδηγήσει στο ξεκίνημα μιας καινούριας φάσης του αλγορίθμου, διαλέγοντας καινούριο αριθμό, αυξάνοντας τον αριθμό της φάσης και στέλνοντας νέο μήνυμα.

Παρακάτω δίνεται ο αλγόριθμος σε μορφή ψευδοκώδικα.

```
var   state_p: (sleep, cand, leader, lost) init sleep;
      level_p:integer init 0;
      id_p: integer;
      stop_p:boolean init false;
if p is initiator then {
      level_p:=1; state_p:=cand; id_p:=rand({1,…,N});
      send <tok, level_p, id_p, 1, true> to Next_p
}
while not stop_p do {
      receive a message;
      if it is a token <tok, level, id, hops, un> then
            if hops=N and state_p=cand then
                  if un then {
                        state_p:=leader; send <ready> to Next_p;
                        receive <ready>; stop_p:=true;
                  }
                  else {
                        level_p:=level_p+1; id_p:=rand({1,…,N});
                        send <tok,level_p,id_p,1,true> to Next_p
                  }
            else if level>level_p or (level=level_p and id < id_p) then {
                  level_p:=level;    state_p:=lost;
                  send <tok, level, id, hops+1, false> to Next_p
            }
            else if level=level_p and id=id_p then
                  send <tok, level, id, hops+1, false> to Next_p
            else skip /* purge the token */
      else {
            send <ready> to Next_p; stop_p:=true;
      }
}
```

4. Πρωτόκολλα Δρομολόγησης

Ένας κόμβος σε ένα κατανεμημένο σύστημα, στη γενική περίπτωση δεν συνδέεται άμεσα με όλους τους άλλους κόμβους. Έτσι, ο κάθε κόμβος μπορεί να επικοινωνήσει άμεσα μόνο με ένα υποσύνολο κόμβων, τους γείτονές του.

Δρομολόγηση είναι ο όρος που χρησιμοποιούμε για να περιγράψουμε την διαδικασία με την οποία ένας κόμβος επιλέγει έναν ή περισσότερους γειτονικούς κόμβους για να προωθήσει ένα πακέτο πληροφορίας προς τον τελικό του στόχο. Ο σκοπός της κατασκευής ενός αλγορίθμου δρομολόγησης είναι να δημιουργηθεί μια διαδικασία λήψης απόφασης που να εξασφαλίζει την σωστή μεταφορά του κάθε πακέτου.

Προκειμένου να δρομολογηθεί η διαδικασία λήψης απόφασης, είναι αναγκαίο να υπάρχουν σε κάθε κόμβο πληροφορίες για την τοπολογία του δικτύου. Θεωρούμε ότι οι πληροφορίες αυτές είναι οργανωμένες σε πίνακες δρομολόγησης. Με την εισαγωγή της έννοιας αυτής το πρόβλημα της δρομολόγησης μπορεί να διαιρεθεί σε δύο μέρη:

1. Δημιουργία πινάκων: Οι τιμές των πινάκων πρέπει να υπολογισθούν κατά. Την διάρκεια της αρχικοποίησης του δικτύου και να ενημερώνονται με κάθε αλλαγή στην τοπολογία.

2. Προώθηση πακέτων: Όταν ένα πακέτο πρέπει να σταλεί μέσω του δικτύου, πρέπει να προωθηθεί χρησιμοποιώντας τους πίνακες δρομολόγησης.

Τα βασικά κριτήρια για την εκτίμηση της απόδοσης ενός αλγορίθμου δρομολόγησης:

1. Ορθότητα {Correctness): Ο αλγόριθμος πρέπει να μεταφέρει σωστά τα πακέτα στους τελικούς αποδέκτες του.

2. Πολυπλοκότητα (Complexity): Ο αλγόριθμος για τον υπολογισμό των τιμών των πινάκων δρομολόγησης πρέπει να χρειάζεται όσο λιγότερο χρόνο, χώρο και αριθμό μηνυμάτων.

3. Αποδοτικότητα (Efficiency): Ο αλγόριθμος πρέπει να χρησιμοποιεί τα μονοπάτια του δικτύου με τέτοιο τρόπο ώστε να δημιουργεί όσο το δυνατόν μικρότερες καθυστερήσεις λόγω συσσώρευσης μηνυμάτων σε μέρη του δικτύου. Ένας αλγόριθμος καλείται βέλτιστος αν χρησιμοποιεί τα "καλύτερα" μονοπάτια.

4. Βιωσιμότητα (Robustness): Σε περίπτωση αλλαγών στην τοπολογία, ο αλγόριθμος θα πρέπει να ενημερώνει σωστά τους πίνακες δρομολόγησης.

5. Προσαρμοστικότητα (Adaptiveness): Ο αλγόριθμος πρέπει να είναι σε θέση να ισορροπήσει τη ροή των πακέτων στο δίκτυο, αποφεύγοντας κόμβους που είναι φορτωμένοι και προτιμώντας κόμβους με μικρότερο φόρτο τη συγκεκριμένη χρονική στιγμή.

6. Δικαιοσύνη (Fairness): Ο αλγόριθμος θα πρέπει να εξυπηρετεί τον κάθε κόμβο στον ίδιο βαθμό.

Είναι φανερό ότι τα κριτήρια αυτά είναι αλληλοσυγκρουόμενα και έτσι είναι πολύ δύσκολο για έναν αλγόριθμο να τα ικανοποιήσει όλα σε μεγάλο βαθμό.

Ως συνήθως, το δίκτυο αναπαρίσταται με έναν γράφο, του οποίου οι κόμβοι αντιστοιχούν στις διαδικασίες (υπολογιστές) του δικτύου και οι ακμές στις συνδέσεις του δικτύου. Το αν ένας αλγόριθμος είναι βέλτιστος εξαρτάται από την ανεύρεση ή μη ενός βέλτιστου μονοπατιού στο γράφο. Υπάρχουν διάφορες εκδοχές για το τι

αποκαλούμε βέλτιστο, κάθε μια από τις οποίες δημιουργεί την δική της κλάση αλγορίθμων δρομολόγησης.

1. Ελάχιστες μεταβάσεις (Minimum hop): Το κόστος του μονοπατιού καθορίζεται ως ο αριθμός των βημάτων (μεταβάσεις από κόμβο σε κόμβο) του μονοπατιού.

2. Μικρότερο μονοπάτι (Shortest path): Σε κάθε ακμή του γράφου αντιστοιχίζεται ένα βάρος (μη αρνητικό), και το κόστος του μονοπατιού υπολογίζεται ως το άθροισμα των βαρών των ακμών του μονοπατιού.

3. Ελάχιστη καθυστέρηση (Minimum delay): Σε κάθε ακμή αντιστοιχούμε δυναμικά ένα βάρος, το οποίο εξαρτάται από την κυκλοφορία πακέτων στην ακμή αυτή. Στην περίπτωση αυτή πρέπει να επιλέξουμε μονοπάτι με την μικρότερη δυνατή καθυστέρηση λόγω κίνησης.

4.1. Ο αλγόριθμος Chandy – Misra

Ο αλγόριθμος που προτάθηκε από τους Chandy και Misra υπολογίζει όλα τα ελάχιστα μονοπάτια προς ένα προορισμό, χρησιμοποιώντας ένα κατανεμημένο υπολογισμό που αρχικοποιείται από ένα απλό κόμβο και στον οποίο συμμετέχουν και άλλοι κόμβοι μόνο αφού λάβουν κάποιο μήνυμα. Για τον αλγόριθμο αυτό θεωρούμε ότι κάθε κόμβος γνωρίζει ποιοι είναι οι γείτονές του και το κόστος της σύνδεσης με καθένα από αυτούς.

Για τον υπολογισμό, για όλους τους κόμβους, της απόστασης προς τον κόμβο u_0 (και ενός προτεινόμενου καναλιού), κάθε κόμβος u ξεκινά αρχικοποιώντας την απόσταση $D_u[u_0]$ ίση με άπειρο και περιμένει για την παραλαβή μηνυμάτων. Ο κόμβος u_0 στέλνει ένα μήνυμα <mydist, u_0,0>, όπου δηλώνει την απόσταση από τον εαυτό του ξεκινώντας τον αλγόριθμο. Όταν ένας κόμβος u λαμβάνει ένα μήνυμα <mydist, u_0,d> από τον γείτονα w, όπου η απόσταση $d+w_{uw} < D_u[u_0]$, ο u θέτει $D_u[u_0]$ ίσο με $d+w_{uw}$ και στέλνει ένα μήνυμα <mydist, u_0, $D_u[u_0]$> προς όλους τους γείτονές του.

Παρακάτω δίνεται ο ψευδοκώδικας του αλγορίθμου.

```
Var   Du[u0]: weight, init ∞;
      Nbu[u0]:node, init undefined;
For node u0 only:
Begin
      Du0[u0]=0;
      Forall w ∈ Neighu0 do send <mydist, u0, 0> to w
End
Processing  a <mydist, u0, d> message from neighbour w by u:
Begin
      Receive <mydist, u0, d> from w;
      If d+wuw < Du[u0] then
      Begin
            Du[u0]= d+wuw;
            Nbu[u0]=w;
            Forall x ∈ Neighu do send <mydist, u0, Du[u0]> to x
      End
End
```

Εάν τα κόστη όλων των καναλιών θεωρούνται ότι είναι ίσα, όλα τα συντομότερα μονοπάτια προς τον κόμβο u_0 υπολογίζονται ανταλλάσσοντας $O(N^2)$ μηνύματα ανά κανάλι και $O(N^2*|E|)$ μηνύματα συνολικά.

4.2. Ο αλγόριθμος Floyd-Warshal (μη κατανεμημένος)

Δίνεται ένας γράφος G=(V,E) με βάρη, και σημειώνουμε το βάρος της ακμής uv ως w_{uv}. Ο γράφος δεν περιέχει κύκλους με συνολικά αρνητικό βάρος. Το βάρος του μονοπατιού $u_0,\ldots,u_k$ είναι ίσο με το άθροισμα των βαρών όλων των συνδέσεων που μετέχουν στο μονοπάτι. Ως απόσταση μεταξύ u και v ορίζουμε το μικρότερο βάρος από κάθε μονοπάτι από το u στο v. Το πρόβλημα έγκειται στον υπολογισμό του $d(u,v)\ \forall\ u,v \in V$.

Για τον υπολογισμό των d(u,v) ο αλγόριθμος χρησιμοποιεί την έννοια των S-μονοπατιών.

Ορισμός: Έστω ότι το S είναι ένα υποσύνολο του V. Ένα μονοπάτι $<u_0,...,u_k>$ είναι S-μονοπάτι αν $\forall\ i : 0<i<k,\ u_i \in S$. Η S-απόσταση από το u στο v σημειώνεται με $d^s(u, v)$ και είναι το μικρότερο από τα βάρη όλων των S-μονοπατιών από το u στο v.

Ο αλγόριθμος ξεκινά με όλα τα ∅-μονοπάτια και προχωράει υπολογίζοντας S-μονοπάτια για όλο και μεγαλύτερα υποσύνολα του V, μέχρι να ληφθούν υπόψη όλα τα V-μονοπάτια, δηλαδή αυξητικά υπολογίζει S-μονοπάτια για μεγαλύτερα σύνολα S.

Ο αλγόριθμος σε ψευδογλώσσα δίνεται αμέσως παρτακάτω:

```
begin /* αρχικοποίησε το S σε ∅ και το D σε 0 απόσταση */
      S:= ∅;
      Forall u,v do
            if u=v then D[u,v] := 0
            else if uv ∈ E then D[u,v] := w_uv
            else D[u,v] := ∞ ;
      /* επέκτεινε το S */
      while S≠V do
      begin
            pick w from V\S;
            forall u ∈ V do
                  forall v ∈ V do
                        D[u,v]:= min (D[u,v], D[u,w] + D[w,v]);
            S := S ∪{w}
      end /* while */
end
```

Η βασική επανάληψη του αλγορίθμου εκτελείται N φορές και περιλαμβάνει N^2 λειτουργίες (οι οποίες μπορούν να εκτελεστούν παράλληλα ή σειριακά), που σημαίνει ότι η απόσταση ανάμεσα σε δύο οποιουσδήποτε κόμβους εκτελείται σε $\Theta(N^3)$ βήματα.

4.3. Ο κατανεμημένος αλγόριθμος του Toueg

Στην ενότητα αυτή θα ασχοληθούμε με ένα αλγόριθμο (Toueg) για τον ταυτόχρονο υπολογισμό όλων των πινάκων δρομολόγησης στο δίκτυο. Ο αλγόριθμος υπολογίζει το συντομότερο μονοπάτι για κάθε ζεύγος (u,v) κόμβων και αποθηκεύει στον u τον πρώτο κόμβο του μονοπατιού αυτού. Ο αλγόριθμος αυτός βασίζεται στον μη κατανεμημένο αλγόριθμο των Floyd – Warshall, ενώ και οι υποθέσεις του αλγόριθμου των Floyd-Warhal (ότι ο γράφος δεν έχει κύκλο με αρνητικό βάρος) είναι ρεαλιστικές για ένα κατανεμημένο σύστημα.

Στο εξής θα θεωρούμε ότι στο σύστημα μας ισχύουν τα εξής:

1. Κάθε κύκλος στο δίκτυο έχει ένα θετικό βάρος.

2. Κάθε κόμβος στο δίκτυο αρχικά γνωρίζει τα χαρακτηριστικά των άλλων κόμβων.

3. Κάθε κόμβος γνωρίζει ποιοι από τους κόμβους είναι γειτονές του (αποθηκεύονται σε πίνακα $Neigh_u[k]$) και τα βάρη των ακμών που εξέρχονται απ' αυτόν.

Στο πλαίσιο του μαθήματος θα εξετάσουμε μόνο την απλοποιημένη έκδοση του αλγόριθμου («απλός αλγόριθμος»).

Για να μεταβούμε σε κατανεμημένο αλγόριθμο οι μεταβλητές και οι πράξεις του αλγόριθμου των Floyd-Warshal πρέπει να μοιραστούν στους κόμβους του δικτύου. Η μεταβλητή (πίνακας) D[u,v] ανήκει στον κόμβο u και θα αναφέρεται στο εξής ως $D_u[v]$. Οι πράξεις πάνω στη μεταβλητή αυτή πρέπει να γίνουν στον κόμβο u, και αν χρειάζονται και τιμές αποθηκευμένες σε άλλους κόμβους για τις πράξεις στον N_u, αυτές πρέπει να σταλούν στον u. Οι πληροφορίες του κόμβου w στον αλγόριθμο Floyd-Warshal (pick w E V) στέλνονται σε όλους τους κόμβους ταυτόχρονα χρησιμοποιώντας broadcasting. Τέλος στον αλγόριθμο πρέπει να υπολογίσουμε, εκτός από το μήκος του μικρότερου μονοπατιού και την πρώτη ακμή του μονοπατιού αυτού ($Nb_u[v]$).

Ο "απλός" αλγόριθμος φαίνεται αμέσως παρακάτω σε μορφή ψευδοκώδικα:

```
var   S_u : set of nodes;
      D_u : array of weights;
      Nb_u : array of neighboring nodes;
begin
      S_u =∅
      forall v ∈ V do
            if v=u then
            begin
                  D_u[v] := 0;
                  Nb_u[v] := undef;
            end
            else if v ∈ Neigh_u then
            begin
                  D_u[v] := w_uv;
                  Nb_u[v] :=ν;
            end
            else
            begin
                  D_u[v] := ∞;
                  Nb_u[v] := undef;
            end
      while S_u ≠ V do
      begin
            pick w from V\S_u; /* όλοι οι κόμβοι πρέπει να επιλέξουν
      τον ίδιο κόμβο εδώ */
            if u=w then
                  broadcast the table D_w;
            else
                  receive table D_w;
            forall v ε V do
                  if D_u[w] + D_w[v] < D_u[v] then
                  begin
                        D_u[v] := D_u[w] + D_w[u];
                        Nb_u[v] := Nb_u[w];
                  end
            S_u := S_u ∪{w}
```

```
      end /*while*/
end
```

Ο αλγόριθμος των Floyd-Warshal μετατρέπεται εύκολα στον "απλό" αλγόριθμο. Κάθε κόμβος αρχικοποιεί τις μεταβλητές του και εκτελεί N φορές το βασικό βρόγχο. Ο αλγόριθμος δεν είναι ολοκληρωμένος, καθώς δεν έχουμε ακόμη ξεκαθαρίσει τις λειτουργίες broadcast και receive.

Ένα σημείο που χρίζει προσοχής είναι η επιλογή "pick w from V\S", καθώς όλοι οι κόμβοι πρέπει να επιλέξουν το ίδιο w εδώ. Υποθέσαμε όμως ότι ο κάθε κόμβος γνωρίζει όλους τους άλλους κόμβους, οπότε θεωρούμε ότι οι κόμβοι διατάσσονται με κάποια σειρά που είναι ίδια για όλους τους κόμβους.

Ο αλγόριθμος ανταλλάσσει O(N) μηνύματα ανά κανάλι και O(N*|E|) συνολικά.

4.4. Ο αλγόριθμος Merlin-Segall

Ο αλγόριθμος που προτάθηκε από τους Merlin και Segall υπολογίζει τους πίνακες δρομολόγησης για κάθε προορισμό εντελώς ξεχωριστά. Οι υπολογισμοί για διαφορετικούς προορισμούς δεν επηρεάζουν ο ένας τον άλλο. Για κάθε προορισμό v ο αλγόριθμος ξεκινά με ένα δένδρο T_v που «δείχνει» προς τον v και επαναληπτικά ενημερώνει το δένδρο αυτό ώστε να γίνει ένα βέλτιστο sink tree για τον κόμβο v.

Ένα γεννητικό δένδρο όπου όλοι οι κόμβοι του «δείχνουν» προς τον v είναι ένα sink tree προς τον v. Ενώ αν για κάθε v που ανήκει στο V υπάρχει ένα δένδρο $T_u=(V, E_v)$ τέτοιο ώστε $E \subseteq E_v$ και για κάθε κόμβο v που ανήκει στο V, το μονοπάτι από το u στο v μέσα από το T_v είναι ένα βέλτιστο μονοπάτι, τότε αυτό είναι ένα βέλτιστο sink tree προς τον v.

Για τον προορισμό v, κάθε κόμβος u διατηρεί μια εκτίμηση για την απόσταση από τον v ($D_u[v]$) και τον γείτονα προς τον οποίο προωθούνται πακέτα προς τον u ($Nb_u[v]$), ο οποίος είναι επίσης ο πατέρας του u στο sink tree T_v. Σε κάθε γύρο ενημέρωσης κάθε κόμβος u στέλνει την εκτιμώμενη απόστασή του $D_u[v]$ προς όλους τους γείτονές του, εκτός του $Nb_u[v]$ κόμβου σε ένα μήνυμα <mydist, v, $D_u[v]$>. Εάν ο κόμβος u λάβει από το γείτονά του w ένα μήνυμα <mydist, v, d> και αν $d+w_{uw}< D_u[v]$, ο u θα αλλάξει τον $Nb_u[v]$ σε w και το $D_u[v]$ σε $d+w_{uw}$. Ο γύρος ενημέρωσης ελέγχεται από τον v και απαιτεί την ανταλλαγή δύο μηνυμάτων W bits για κάθε κανάλι.

Μετά από i γύρους ενημέρωσης όλα τα συντομότερα μονοπάτια μέχρι i hops θα έχουν υπολογιστεί σωστά, έτσι ώστε μετά από N γύρους το πολύ όλα τα συντομότερα μονοπάτια προς τον v θα έχουν υπολογιστεί. Τα συντομότερα μονοπάτια προς κάθε προορισμό υπολογίζονται εκτελώντας τον αλγόριθμο ανεξάρτητα για κάθε προορισμό.

Ο αλγόριθμος αυτός υπολογίζει τα συντομότερα μονοπάτια ανταλλάσσοντας $O(N^2)$ μηνύματα ανά κανάλι, και $O(N^2 * |E|)$ μηνύματα συνολικά.

5. Πρωτόκολλα Ελέγχου Τερματισμού (Termination Detection Protocols)

Ένας υπολογισμός σε ένα κατανεμημένο σύστημα τερματίζει όταν ο αλγόριθμος φτάσει σε μια τελική κατάσταση, αυτό σημαίνει ότι φτάνει σε μια κατάσταση όπου δεν υπάρχουν άλλα βήματα στον αλγόριθμο που να μπορούν να εφαρμοστούν. Αλλά δεν συμβαίνει πάντα το γεγονός ότι σε μια τελική κατάσταση, κάθε διεργασία να βρίσκεται σε μια τελική κατάσταση, δηλαδή μια κατάσταση στην οποία δεν μπορεί για τη συγκεκριμένη διεργασία να εφαρμοστεί κάποιο επόμενο γεγονός. Θεωρείστε μια κατάσταση κατά την οποία κάθε διεργασία βρίσκεται σε μια κατάσταση που επιτρέπει την παραλαβή μηνυμάτων, αλλά όλα τα κανάλια είναι άδεια. Μια τέτοια κατάσταση είναι τελική, αλλά οι καταστάσεις στις οποίες βρίσκονται οι διεργασίες μπορούν να αποτελέσουν ενδιάμεσες καταστάσεις κατά την φάση εκτέλεσης του αλγορίθμου. Στην περίπτωση αυτή οι διεργασίες δεν έχουν καταλάβει ότι ο υπολογισμός τελείωσε και ο τερματισμός του υπολογισμού θεωρείται ότι είναι ασαφής. Ο τερματισμός υπολογισμού θεωρείται ότι είναι σαφής για μια διεργασία όταν αυτή βρίσκεται σε μια κατάσταση κατά την τελική κατάσταση του υπολογισμού που είναι τελική κατάσταση για την ίδια τη διεργασία. Ο ασαφής τερματισμός ενός υπολογισμού καλείται τερματισμός μηνυμάτων, γιατί από την στιγμή που ο υπολογισμός έφτασε σε μια τερματική κατάσταση δεν ανταλλάσσονται πλέον άλλα μηνύματα. Ο σαφής τερματισμός καλείται επίσης τερματισμός διεργασιών, γιατί οι διεργασίες έχουν πλέον φτάσει σε μια τελική κατάσταση.

Συνήθως είναι ευκολότερο να σχεδιάσει κανείς ένα αλγόριθμο που τερματίζεται ασαφώς, παρά το ανάποδο. Μάλιστα κατά τη διάρκεια της σχεδίασης του αλγορίθμου όλα τα στοιχεία που αφορούν τον κατάλληλο τερματισμό των διεργασιών μπορούν να αγνοηθούν, ο σχεδιασμός προσπαθεί απλά να ελέγξει και να περιορίσει τον συνολικό αριθμό των γεγονότων που μπορούν να συμβούν. Από την άλλη όμως η εφαρμογή ενός αλγορίθμου μπορεί να απαιτεί ότι οι διεργασίες πρέπει να τερματίζουν σαφώς. Μόνο μετά από σαφή τερματισμό τα αποτελέσματα ενός υπολογισμού μπορούν να θεωρηθούν σαν τελικά και οι μεταβλητές που χρησιμοποιήθηκαν μπορούν να αποδεσμευτούν.

Υπάρχουν γενικές μέθοδοι για τη μετατροπή αλγορίθμων τερματισμού μηνυμάτων σε αλγορίθμους τερματισμού διεργασιών. Οι μέθοδες αυτοί αποτελούνται από δύο επιπλέον αλγορίθμους που συνεργάζονται με τον αλγόριθμο τερματισμού μηνυμάτων (βασικός αλγόριθμος) και μεταξύ τους. Ο ένας από αυτούς τους αλγορίθμους παρατηρεί τον υπολογισμό και ανιχνεύσει ότι ο υπολογισμός έχει φτάσει σε μια τερματική κατάσταση. Τότε καλεί τον δεύτερο αλγόριθμο ο οποίος ένα μήνυμα τερματισμού σε όλες τις διεργασίες, κάνοντάς τες να μπουν σε μια τερματική κατάσταση.

Το πιο δύσκολο κομμάτι αυτής της μετατροπής είναι η κατασκευή του αλγορίθμου που ανιχνεύει τον τερματισμό. Ο αλγόριθμος διάδοσης του γεγονότος του τερματισμού είναι μάλλον κάτι απλό.

5.1. Οι κανόνες του βασικού αλγορίθμου

Το σύνολο των καταστάσεων για μια διεργασία διαιρείται σε δύο υποσύνολα, τις ενεργές και τις παθητικές καταστάσεις. Μια κατάσταση διεργασίας καλείται ενεργή όταν στην κατάσταση αυτή μπορεί να εφαρμοστεί ένα γεγονός αποστολής μηνύματος

ή ένα εσωτερικό γεγονός. Διαφορετικά η κατάσταση καλείται παθητική. Σε μια παθητική κατάσταση μόνο παραλαβή μηνυμάτων μπορεί να γίνει, ή μπορεί να μην εφαρμοστεί κανένα γεγονός, πράγμα που σημαίνει ότι η διεργασία έχει φτάσει σε μια τερματική κατάσταση. Μια διεργασία μπορεί να στείλει ένα μήνυμα αν βρίσκεται σε ενεργή κατάσταση, ενώ αν βρίσκεται σε παθητική και λάβει κάποιο μήνυμα τότε μόνο πηγαίνει σε ενεργή. Οι κανόνες λοιπόν που διέπουν τον βασικό αλγόριθμο μπορούν να συνοψιστούν στα εξής σημεία:

1. Μια διεργασία που βρίσκεται σε ενεργή κατάσταση μπορεί να πάει σε παθητική μόνο εφόσον συμβεί ένα εσωτερικό γεγονός.
2. Μια διεργασία γίνεται πάντα ενεργή όταν λάβει ένα μήνυμα.
3. Τα εσωτερικά γεγονότα κατά τα οποία μια διεργασία γίνεται παθητική είναι τα μόνα εσωτερικά γεγονότα που μπορούν να συμβούν.

Αρχικά θεωρούμε ότι δεν υπάρχουν μηνύματα που να βρίσκονται σε φάση αποστολής. Οι διεργασίες μπορεί αρχικά να είναι είτε ενεργές είτε παθητικές.

5.2. Ο αλγόριθμος του Dijkstra

Ο σκοπός του αλγορίθμου αυτού είναι να δώσει τη δυνατότητα σε έναν από τους κόμβους (στο συντονιστή) να ανακαλύψει αν έχει επιτευχθεί μια σταθερή κατάσταση, δηλαδή μια κατάσταση κατά την οποία όλοι οι κόμβοι είναι παθητικοί και κανένα μήνυμα δεν βρίσκεται σε φάση μετάδοσης. Σε αυτή την κατάσταση θεωρείται ότι ο κατανεμημένος βασικός αλγόριθμος έχει τελειώσει.

Ο αλγόριθμος του Dijkstra αναφέρεται σε δίκτυα με τοπολογία δακτυλίου. Μπορεί όμως να εφαρμοστεί και σε δίκτυα γενικής τοπολογίας, αρκεί να υπάρχει επικαλύπτων δακτύλιος για την τοπολογία αυτή.

Κάθε κόμβος διατηρεί ένα μετρητή c, ο οποίος αρχικά είναι ίσος με 0. Η αποστολή ενός μηνύματος αυξάνει τον c κατά 1. Η λήψη ενός μηνύματος μειώνει τον c κατά 1. Άρα το άθροισμα όλων των μετρητών c ισούται με τον αριθμό των μηνυμάτων που η παράδοσή τους εκκρεμεί ακόμη. Στο δακτύλιο κινείται μία σκυτάλη, η οποία όπως και κάθε κόμβος έχει ένα χρώμα. Αρχικά όλοι οι κόμβοι και η σκυτάλη έχουν χρώμα λευκό. Όταν ένας κόμβος λαμβάνει ένα μήνυμα του βασικού αλγορίθμου (όχι τη σκυτάλη δηλαδή), γίνεται μαύρος. Όταν ένας κόμβος προωθεί τη σκυτάλη γίνεται λευκός. Όταν ένας μαύρος κόμβος προωθεί τη σκυτάλη, τότε η σκυτάλη γίνεται μαύρη. Διαφορετικά η σκυτάλη διατηρεί το χρώμα της. Παρακάτω δίνεται ο αλγόριθμος σε μορφή ψευδοκώδικα:

```
Ο κόμβος 0 αρχικοποιεί τον αλγόριθμο στέλνοντας μία σκυτάλη με την
τιμή 0 στον κόμβο N-1
Όταν ο κόμβος i (i≠0) λάβει τη σκυτάλη
      Κρατάει τη σκυτάλη μέχρι ο κόμβος να γίνει παθητικός
      Στέλνει τη σκυτάλη στον κόμβο i-1 αυξάνοντας την τιμή της
σκυτάλης κατά c
      Αν ο κόμβος είναι μαύρος η σκυτάλη γίνεται μαύρη
      Ο κόμβος γίνεται λευκός
Όταν ένας κόμβος λάβει ένα μήνυμα του βασικού αλγορίθμου
      Γίνεται μαύρος
Όταν ο κόμβος 0 λάβει τη σκυτάλη
      Αν είναι παθητικός και λευκός
      Αν η σκυτάλη είναι λευκή
      Αν το άθροισμα της τιμής της σκυτάλης και του c είναι 0
            Έχουμε τερματισμό
```

```
Αλλιώς
      Ο κόμβος 0 ξεκινά νέο γύρο στον αλγόριθμο
```

5.3. Ο αλγόριθμος του Mattern

Ο Mattern πρότεινε έναν αλγόριθμο που ανιχνεύει τερματισμό πολύ γρήγορα, για την ακρίβεια, μία χρονική μονάδα μετά τη χρονική στιγμή που συνέβη. Ο αλγόριθμος ανιχνεύει τον τερματισμό ενός κεντρικοποιημένου υπολογισμού και θεωρεί ότι κάθε διεργασία μπορεί να στείλει ένα μήνυμα στον αρχικοποιητή του υπολογισμού απ' ευθείας.

Στον αλγόριθμο σε κάθε μήνυμα και κάθε διεργασία ανατίθεται μια τιμή πιστώσεως (credit value) η οποία είναι συνεχώς μεταξύ 0 και 1 και ο αλγόριθμος διατηρεί τις ακόλουθες θεωρήσεις σαν αμετάβλητες συνθήκες:

- Το άθροισμα όλων των πιστώσεων (μηνυμάτων και διεργασιών) είναι 1.
- Ένα μήνυμα του βασικού αλγορίθμου έχει θετική τιμή πίστωσης.
- Μία ενεργή διεργασία έχει θετική τιμή πίστωσης.

Μία διεργασία που κατέχει μια θετική τιμή πίστωσης αλλά δεν πληροί κάποια από τις παραπάνω τρεις συνθήκες (π.χ. μια παθητική διεργασία) στέλνει την τιμή της πίστωσής της στον αρχικοποιητή. Ο αρχικοποιητής λειτουργεί σαν τράπεζα, συλλέγοντας όλες τις πιστώσεις που στέλνονται σε αυτόν σε μία μεταβλητή που καλείται ret.

Όταν ο αρχικοποιητής κατέχει όλες τις πιστώσεις, η απαίτηση για ενεργές διεργασίες και μηνύματα βασικού αλγορίθμου να έχουν θετική τιμή πίστωσης, σημαίνει ότι δεν υπάρχουν τέτοιες διεργασίες και τέτοια μηνύματα στο δίκτυο. Άρα υπάρχει τερματισμός. Έτσι όταν ret=1 ο αρχικοποιητής ανακοινώνει τον τερματισμό.

Όταν μία ενεργή διεργασία στέλνει ένα μήνυμα, η πίστωσή της μοιράζεται ανάμεσα σε αυτή και το μήνυμα. Όταν μία διεργασία καθίσταται ενεργή παίρνει σαν τιμή πίστωσης αυτή που φέρει το μήνυμα που την ενεργοποιεί. Στην περίπτωση που μία διεργασία λάβει ένα μήνυμα ενώ είναι ήδη ενεργή, δεν χρειάζεται την πίστωση που αυτό φέρει, αλλά ούτε και την καταστρέφει. Η διεργασία τότε κάνει ένα από τα δύο παρακάτω, που και τα δύο οδηγούν σε σωστό αλγόριθμο:

- Η πίστωση που φέρει το μήνυμα στέλνεται στον αρχικοποιητή.
- Η πίστωση που φέρει το μήνυμα προστίθεται στην πίστωση της διεργασίας.

Παρακάτω δίνεται ο αλγόριθμος σε ψευδοκώδικα, σημειώνοντας ότι όταν μία ενεργή διεργασία λάβει μήνυμα, τότε η πίστωση που φέρει το μήνυμα προστίθεται στην πίστωση της διεργασίας.

```
var state_p: (active,passive) init if p=p_0 then active else passive;
cred_p: [0..1] init if p=p_0 then 1 else 0;
ret : [0..1] init 0;  /* for p_0 only */
S_p: /* state=active */
begin send<mes,cred_p/2>; cred_p:=cred_p/2; end
R_p:  /* A message <mes,c> has arrived at p */
begin
      receive <mes,c>;
      state_p:=active;
      cred_p:=cred_p+c;
end
I_p:  /* state_p=active */
```

```
begin
      state_p:=passive;
      send<ret,cred_p> to p_0;
      cred_p:=0;
end
A_p0: /* A <ret,c> message has arrived at p_0 */
begin
      receive <ret,c>;
      ret:=ret+c;
      if ret=1 then Announce_termination;
end
```

5.4. Ο αλγόριθμος των Dijkstra – Scholten

Η λύση που προτείνει ο αλγόριθμος αυτός βασίζεται στη διατήρηση ενός κατευθυνόμενου δένδρου, στους κόμβους του οποίου περιλαμβάνονται όλες οι ενεργές διεργασίες και όλα τα βασικά μηνύματα (μηνύματα του βασικού αλγορίθμου) που μεταδίδονται. Όταν το δένδρο αυτό καταστεί άδειο, τότε ο αλγόριθμος έχει τερματίσει. Ο αλγόριθμος απαιτεί ότι το δίκτυο δεν είναι κατευθυνόμενο, δηλαδή τα μηνύματα μπορούν να σταλούν μέσα από οποιοδήποτε κανάλι. Ο αλγόριθμος που θα εξετάσουμε ανιχνεύει τον τερματισμό σε ένα κεντρικοποιημένο βασικό αλγόριθμο. Ο αρχικοποιητής του βασικού υπολογισμού παίζει ένα σημαντικό ρόλο στην όλη διαδικασία.

Ο αλγόριθμος ανίχνευσης λοιπόν διατηρεί ένα κατευθυνόμενο δένδρο με τις ακόλουθες δύο ιδιότητες:

1. Είτε το δένδρο είναι άδειο, είτε είναι κατευθυνόμενο με ρίζα τον αρχικοποιητή του βασικού αλγορίθμου.
2. Το σύνολο των κόμβων του δένδρου περιλαμβάνει όλες τις ενεργές διεργασίες και όλα τα βασικά μηνύματα που βρίσκονται σε μετάδοση.

Όταν το δένδρο γίνει άδειο, τότε ο αρχικοποιητής καλεί την ρουτίνα που ανακοινώνει τον τερματισμό σε όλο το δίκτυο.

Όταν λοιπόν μια διεργασία στέλνει ένα μήνυμα ένα βασικό μήνυμα μπαίνει σαν κόμβος στο δένδρο, έχοντας σαν πατέρα την διεργασία. Όταν μια διεργασία λάβει ένα μήνυμα και περάσει σε ενεργή κατάσταση, τότε μπαίνει στο δένδρο και έχει σαν πατέρα τον κόμβο – αποστολέα του αντίστοιχου μηνύματος. Όταν ένα μήνυμα παραλαμβάνεται τότε πρέπει να διαγράφεται από το δένδρο. Κάθε διεργασία όμως διατηρεί ένα αριθμό που αφορά το σύνολο των παιδιών της στο δένδρο. Για το λόγο αυτό κάθε φορά που ένα μήνυμα παραλαμβάνεται ή μια διεργασία πρέπει να βγει από το δένδρο (γιατί έγινε παθητική), τότε η διεργασία αυτή στέλνει ένα σήμα στον πατέρα της, με σκοπό να τον ενημερώσει ότι πρέπει να μειώσει τον αριθμό των παιδιών του. Έτσι γίνεται σωστά η αφαίρεση κόμβων από το δένδρο.

Ο αλγόριθμος παρουσιάζεται παρακάτω σε μορφή ψευδοκώδικα.

```
var   state_p  : (active, passive) init if p=p_0 then active else
passive; /* p_0 είναι ο αρχικοποιητής του βασικού αλγορίθμου */
      sc_p : integer init 0;
      father_p : P init if p=p_0 then p else udef;
S_p  : {state_p=active} /* αποστολή μηνύματος με την ταυτότητα του
κόμβου */
      begin send <mes,p>; sc_p:=sc_p+1 end
R_p: { A message <mes,q> has arrived at p }
begin       receive <mes,q>;
```

```
            state_p:=active;
            if father_p=udef then father_p:=q
            else send <sig,q> to q
      end
I_p: {state=active}
      begin
            state_p:=passive;
            if sc_p=0 then
                  begin if father_p=p then Announce
                        else send <sig, father_p> to father_p;
                        father_p:=udef
                  end
      end
A_p: {A signal <sig, p> arrives at p}
      begin
            receive <sig,p>;
            sc_p:=sc_p-1;
            if sc_p=0 and state=passive then
                  begin if father_p=p then
                        Announce
                  else send <sig, father_p> to father_p;
                  father_p:=udef;
                  end
end
```

Ο αλγόριθμος των Dijkstra-Scholten χρησιμοποιεί Μ μηνύματα ελέγχου, όπου Μ το σύνολο των μηνυμάτων του βασικού αλγορίθμου.

Απόδειξη: Έστω S το άθροισμα όλων των sc_p μεταβλητών. Αρχικά το S είναι κενό, αυξάνεται όταν στέλνεται ένα μήνυμα βασικού αλγορίθμου και μειώνεται όταν ένα μήνυμα βασικού αλγορίθμου λαμβάνεται. Επίσης το S δεν είναι ποτέ αρνητικό. Άρα το σύνολο των μηνυμάτων ελέγχου δεν ξεπερνά τον αριθμό των μηνυμάτων του βασικού αλγορίθμου.

6. Ανοχή σε Σφάλματα (Fault Tolerance)

Στις προηγούμενες ενότητες θεωρήσαμε ότι οι διεργασίες είναι αξιόπιστες. Για διάφορους λόγους είναι ελκυστικό να μελετήσουμε τη συμπεριφορά του κατανεμημένου συστήματος με τη θεώρηση ότι οι διεργασίες μπορεί να καταρρεύσουν.

Διακρίνουμε λοιπόν δύο κατηγορίες αλγορίθμων και θα μελετηθούν αλγόριθμοι για καθεμιά από τις κατηγορίες αυτές:

- Αλγόριθμοι ευρωστίας (robust algorithms): κάθε βήμα μιας διεργασίας γίνεται με προσοχή για να επιβεβαιώσει ότι παρά τα όποια σφάλματα, οι σωστές διεργασίες εκτελούν σωστά βήματα.
- Αλγόριθμοι σταθεροποίησης (stabilization algorithms): οι σωστές διεργασίες μπορεί να επηρεαστούν από σφάλματα, αλλά ο αλγόριθμος εγγυάται την επανάκαμψη από οποιαδήποτε διαμόρφωση όταν η διεργασία επανέλθει σε σωστή συμπεριφορά.

Γενικά οι αλγόριθμοι σταθεροποίησης προσφέρουν προστασία από προσωρινά σφάλματα, π.χ. προσωρινή ανώμαλη συμπεριφορά συστατικών του συστήματος. Αυτά τα σφάλματα μπορεί να συμβούν σε μεγάλα τμήματα ενός κατανεμημένου συστήματος όταν οι φυσικές συνθήκες προσωρινά φτάσουν ακραίες τιμές, εισάγοντας εσφαλμένη συμπεριφορά μνημών και επεξεργαστών. Οι αλγόριθμοι ευρωστίας ασχολούνται με τα μόνιμα σφάλματα από ένα περιορισμένο σύνολο από συστατικά στοιχεία του συστήματος. Οι υπόλοιπες διεργασίες δεν επηρεάζονται από αυτά τα σφάλματα και παραμένουν σωστές.

6.1. Πρωτόκολλα Ευρωστίας (Robust Protocols)

Στους αλγόριθμους ευρωστίας διακρίνονται επίσης τρία διαφορετικά μοντέλα σφαλμάτων:

- Αρχικά-νεκρές διεργασίες (initially dead processes): μία διεργασία δεν εκτελεί κανένα βήμα του τοπικού της αλγορίθμου.
- Μοντέλο κατάρρευσης (crash model): μια διεργασία θεωρείται ότι καταρρέει εάν εκτελεί τον τοπικό της αλγόριθμο σωστά μέχρι ενός χρονικού σημείου, και δεν εκτελεί κανένα βήμα μετά.
- Βυζαντινή συμπεριφορά (byzantine behavior): μία διεργασία καλείται βυζαντινή εάν εκτελεί βήματα τα οποία είναι αυθαίρετα βήματα και όχι σύμφωνα με τον τοπικό της αλγόριθμο. Συγκεκριμένα, στέλνει μηνύματα με αυθαίρετο περιεχόμενο.

6.1.1. Ο αλγόριθμος των Fischer, Lynch & Paterson

Ο αλγόριθμος αυτός αναφέρεται στο μοντέλο αρχικά-νεκρών διεργασιών όπου κάθε σωστή διεργασία υπολογίζει το ίδιο σύνολο σωστών διεργασιών. Έστω ότι υπάρχουν τουλάχιστον L σωστές διεργασίες, όπου $L=\lceil (N+1)/2 \rceil$. Η ευκαμψία του αλγορίθμου είναι $\lfloor (N-1)/2 \rfloor$. Ο αλγόριθμος εκτελείται σε δύο φάσεις. Οι διεργασίες στέλνουν μηνύματα στον εαυτό τους, κάτι που είναι συνηθισμένο στους αλγορίθμους ευρωστίας γιατί διευκολύνει την ανάλυση. Ο ψευδοκώδικας του αλγορίθμου

παρουσιάζεται παρακάτω και όπου φαίνεται η λειτουργία «shout <mes>», σημαίνει το εξής: forall q ϵ P do send <mes> to q.

Οι διεργασίες κατασκευάζουν ένα κατευθυνόμενο γράφο G ανακοινώνοντας την ταυτότητά τους και περιμένουν για την λήψη L μηνυμάτων. Καθώς υπάρχουν τουλάχιστον L σωστές διεργασίες, κάθε σωστή διεργασία λαμβάνει καταλλήλως πολλά μηνύματα για να ολοκληρώσει αυτό το έργο. Οι επόμενοι της p σε ένα γράφο G είναι οι κόμβοι q από τους οποίους η p έχει λάβει ένα μήνυμα <name,q>.

Μία αρχικά-νεκρή διεργασία δεν στέλνει ούτε λαμβάνει μηνύματα σχηματίζοντας έτσι ένα απομονωμένο κόμβο στον G. Μια σωστή διεργασία έχει L επόμενους και έτσι δεν είναι απομονωμένη. Ένας «κόμπος» (knot) είναι ένα ισχυρά συνδεδεμένο συστατικό μέρος του γράφου, χωρίς εξερχόμενες ακμές, που περιέχει τουλάχιστον δύο κορυφές. Υπάρχει στον G ένας κόμπος που περιέχει σωστές διεργασίες και επειδή κάθε σωστή διεργασία έχει εξωτερικό βαθμό στο γράφο L, αυτός ο κόμπος έχει μέγεθος τουλάχιστον L. Ακολούθως, αφού 2L>N, υπάρχει ακριβώς ένας κόμπος, έστω K. Τελικά, αφού μια σωστή διεργασία p έχει L επόμενους, τουλάχιστον ένα επόμενος ανήκει στον K, υπονοώντας ότι όλες οι διεργασίες του K είναι απόγονοι της p.

Στη δεύτερη φάση οι διεργασίες κατασκευάζουν ένα παραγόμενο υπογράφο του G, που περιέχει τουλάχιστον τους απογόνους τους στο κατευθυνόμενο δένδρο, λαμβάνοντας το σύνολο των επόμενων κάθε διεργασίας που γνωρίζουν ότι είναι σωστή. Επειδή οι διεργασίες δεν καταρρέουν μετά την αποστολή ενός μηνύματος, δεν υπάρχει αδιέξοδο σε αυτή τη φάση. Μάλιστα, η p περιμένει να λάβει ένα μήνυμα από την q μόνο αν κατά την πρώτη φάση κάποιες διεργασίες έλαβαν ένα μήνυμα <name, q>, πράγμα που δείχνει ότι η q είναι σωστή.

Στο τέλος του αλγορίθμου κάθε σωστή διεργασία έχει λάβει το σύνολο των επόμενων καθεμιάς από τους απογόνους της, πράγμα που της επιτρέπει να υπολογίσει τον μοναδικό «κόμπο» στον G γράφο.

Παρακάτω δίνεται ο αλγόριθμος σε μορφή ψευδοκώδικα:

var $Succ_p$, $Alive_p$, $Rcvd_p$: sets of processes init Ø;

```
begin
    shout <name,p>;
    while #Succ_p<L do begin receive <name,q>; Succ_p= Succ_p ∪ {q} end;
    shout <pre,p, Succ_p>;
    Alive_p=Succ_p;
    while Alive_p Rcvd_p do
        begin
            receive <pre,p, Succ>;
            Alive_p= Alive_p ∪ Succ ∪ {q};
            Rcvd_p= Rcvd_p ∪ {q};
        end
    Compute a knot in G
end
```

6.2. Πρωτόκολλα Σταθεροποίησης (Stabilization Protocols)

Οι αλγόριθμοι που χαρακτηρίζονται από ευρωστία (robust algorithms) ακολουθούν μια πολιτική απαισιοδοξίας. Ελέγχουν όλη την εισερχόμενη πληροφορία και προχωρούν κάνοντας συνεχείς ελέγχους για να εγγυηθούν την νομιμότητα των βημάτων που κάνουν. Οι σταθεροποιούμενοι αλγόριθμοι (stabilizing algorithms) ακολουθούν μια πολιτική αισιοδοξίας η οποία μπορεί να προκαλέσει σε μια διεργασία να συμπεριφέρεται με αστάθεια αλλά εγγυάται την επιστροφή σε σωστή συμπεριφορά σε περιορισμένο αριθμό βημάτων μετά την εμφάνιση της αστάθειας.

Οι διεργασίες θεωρείται ότι δουλεύουν σωστά αλλά η συνολική διαμόρφωση του συστήματος μπορεί να διαταραχθεί από κάποια βλάβη. Ένας αλγόριθμος θεωρείται ότι είναι αυτο-σταθεροποιούμενος αν μπορεί να ξεκινήσει να συμπεριφέρεται σωστά ανεξάρτητα από τη διαμόρφωση του συστήματος πάνω στην οποία τον εφαρμόζουμε.

6.2.1. Ιδιότητες των σταθεροποιούμενων αλγορίθμων

Οι αλγόριθμοι αυτοί προσφέρουν 4 βασικά πλεονεκτήματα σε σχέση με τους κλασσικούς αλγορίθμους.

1) *Ανοχή σε λάθη (fault tolerance):* Ένας σταθεροποιούμενος αλγόριθμος προσφέρει αυτόματη προστασία απέναντι σε σφάλματα γιατί ο αλγόριθμος μπορεί να επανέλθει από οποιαδήποτε διαμόρφωση χωρίς να παίζει ρόλο η έκταση της καταστροφής πάνω στα δεδομένα.

2) *Αρχικοποίηση:* Η ανάγκη κατάλληλης αρχικοποίησης δεν υπάρχει αφού οι διεργασίες μπορούν να ξεκινήσουν από οποιαδήποτε κατάσταση και η συμπεριφορά του συστήματος είναι εγγυημένη.

3) *Δυναμική Τοπολογία:* Ένας σταθεροποιούμενος αλγόριθμος συγκλίνει σε λύση ακόμη και μετά την εμφάνιση μιας τοπολογικής αλλαγής.

Από την άλλη πλευρά υπάρχουν και τα ακόλουθα τρία μειονεκτήματα:

1) *Αρχικές ασυμβατότητες:* Πριν φτάσει σε κάποια νόμιμη διαμόρφωση ο αλγόριθμος μπορεί να δείξει ασταθή έξοδο.

2) Υ*ψηλή πολυπλοκότητα:* Οι αλγόριθμοι αυτοί είναι πολύ λιγότερο εύκολοι στην χρήση από τους κλασσικούς.

3) *Λεν ανακαλύπτεται η σταθεροποίηση:* Δεν είναι δυνατόν να βρεθεί πότε έχει φτάσει το σύστημα σε μια νόμιμη διαμόρφωση. Γι' αυτό και οι διεργασίες ποτέ δεν ασχολούνται με το αν η συμπεριφορά τους έχει γίνει αξιόπιστη.

6.2.2. Επικοινωνία στα σταθεροποιούμενα συστήματα.

Ας θεωρήσουμε ένα ασύγχρονο αλγόριθμο με πέρασμα μηνυμάτων στον οποίο υπάρχει μια διεργασία p όπου κάθε κατάσταση είναι κατάσταση αποστολής (η p μπορεί να στείλει ένα μήνυμα από οποιαδήποτε κατάσταση). Το σύστημα έχει μια εκτέλεση που αποτελείται μόνο από διαδικασίες αποστολής της p ενώ όλες οι άλλες διεργασίες παραμένουν "παγωμένες" στην αρχική τους κατάσταση και αυτή η συμπεριφορά δεν ικανοποιεί καμία προδιαγραφή.

Δεύτερον, θεωρείστε έναν αλγόριθμο όπου για κάθε διεργασία υπάρχουν καταστάσεις στις οποίες η διεργασία δεν στέλνει μηνύματα (αλλά μόνο λαμβάνει ή κάνει κάποιο

εσωτερικό βήμα). Κάθε διαμόρφωση στην οποία κάθε διεργασία είναι σε μια τέτοια κατάσταση και όλα τα κανάλια είναι άδεια είναι μια τερματική (terminal) διαμόρφωση, άρα πρέπει να ικανοποιεί την (όποια) προδιαγραφή. Ξανά μια οποιαδήποτε μη τετριμμένη προδιαγραφή ικανοποιείται από όλες τις διαμορφώσεις.

6.2.3. Παράδειγμα: Ο αλγόριθμος Token Ring του Dijkstra

Ο πρώτο σταθεροποιούμενος αλγόριθμος προτάθηκε από τον Dijkstra το 1974 και αφορούσε αμοιβαίο αποκλεισμό σε έναν δακτύλιο διεργασιών. Για το πρόβλημα αυτό, θεωρούμε ότι οι διεργασίες εκτελούν κώδικα σε κρίσιμες περιοχές αλλά μόνο μια διεργασία μπορεί να εκτελεί τον κώδικα αυτό κάθε χρονική στιγμή. Κάθε διεργασία εφοδιάζεται με μια τοπική συνθήκη (predicate) που όταν είναι αληθής δείχνει ότι η διεργασία έχει το δικαίωμα να εκτελεί τον κώδικα της κρίσιμης περιοχής. Ο ορισμός του προ6λήματος είναι:

1) Σε κάθε διαμόρφωση, μόνο μια διεργασία έχει το δικαίωμα εκτέλεσης του κώδικα στην κρίσιμη περιοχή

2) Κάθε διεργασία αποκτά το δικαίωμα συχνά.

Το πρόβλημα επιλύεται με το να γυρνά ένα token στο δακτύλιο και η διεργασία που το έχει, έχει και το δικαίωμα.

Περιγραφή της λύσης: Η κατάσταση της διαδικασίας i είναι ένας ακέραιος από το 0 ως το k-l όπου το k είναι ένας ακέραιος που ξεπερνά το N (μέγεθος δακτυλίου). Η διαδικασία i μπορεί να διαβάσει την κατάσταση της προηγούμενής της (εκτός φυσικά από τη δική της) και η διαδικασία 0 μπορεί να διαβάσει την κατάσταση της N-1 διεργασίας. Όλες οι διεργασίες εκτός της 0 είναι ισοδύναμες. Η διαδικασία i (διαφορετική από τη 0) έχει το δικαίωμα να εισέλθει στην κρίσιμη περιοχή αν η κατάστασή της είναι διαφορετική από αυτή της προηγούμενής της, ενώ η διαδικασία 0 το έχει όταν η κατάστασή της είναι ίδια με αυτή της N-1. Μια διαδικασία που έχει το δικαίωμα μπορεί επίσης να αλλάξει την κατάσταση της (το κάνει συνήθως αφού ολοκληρώσει τον κώδικα της κρίσιμης περιοχής). Μια αλλαγή κατάστασης σε μια διαδικασία προκαλεί πάντα και την απώλεια του δικαιώματός της. Η διαδικασία i (διαφορετική από τη 0) μπορεί να θέσει την κατάστασή της ίδια με αυτή της προηγούμενής της (όταν αυτό επιτρέπεται). Η διαδικασία 0 μπορεί να την κάνει διαφορετική από αυτή της N-1 (όταν είναι ίσο).

6.2.4. Προσανατολισμός δακτυλίου

Θεωρούμε έναν μη κατευθυνόμενο δακτύλιο από N διεργασίες όπου κάθε διεργασία έχει ονομάσει τη μία σύνδεσή της "επόμενη" και "προηγούμενη". (Η διεργασία αλλάζει τα ονόματα αν βρει ότι έχει ονομάσει και τις δύο συνδέσεις το ίδιο). Η ονομασία της διαδικασίας l για την σύνδεση pq καλείται l_{pq} .Το πρόβλημα απαιτεί από τον αλγόριθμο να κάνει το σύστημα να ικανοποιεί την' συνθήκη: για κάθε ακμή p τότε το l_{pq} είναι "επόμενη" αν και μόνο αν το l_{qp} είναι προηγούμενη".

Ο αλγόριθμος εκτός από καταχωρητές των συνδέσεων που έχουν τις τιμές pred (προηγούμενη) και succ (επόμενη) χρησιμοποιεί και μια μεταβλητή s με τιμές S,R και I. Σε ένα βήμα η διεργασία εξετάζει την κατάστασή της, την κατάσταση του γείτονα q και τους καταχωρητές των συνδέσεων και αλλάζει κατάσταση αν κάποια από τις s συνθήκες είναι true. Οι διεργασίες κυκλοφορούν κυκλικά στον δακτύλιο tokens, και κάθε διεργασία θέτει σαν επόμενό της την κατεύθυνση του τελευταίου token που έστειλε. Αν δύο token συναντηθούν το ένα εξαφανίζεται. Τελικά όλα τα

εναπομείναντα tokens ταξιδεύουν προς την ίδια κατεύθυνση το οποίο κάνει τον δακτύλιο να προσανατολιστεί (τα tokens παραμένουν και εξακολουθούν να περιστρέφονται).

```
var sp : (S,R,I) :
(1)   sp=I και sq=S και lqp=«επόμενη» τότε
      sp=R; if lpq=«επόμενη» then flip
(2)   sp=S και sq=R και lqp=«προηγούμενη» και lpq=«επόμενη» τότε sp=I
(3)   sp=R και lpq=«προηγούμενη» και !(sq=S και lqp=«επόμενη») τότε
sp=S
(4)   sp= sq=S και lqp =lpq =«επόμενη» τότε sp=R; flip
(5)   sp= sq=I και lqp =lpq =«επόμενη» τότε sp=S
```

procedure flip: αντιστροφή των τιμών succ και pred για την p.

Στην κατάσταση S, η διεργασία περιμένει να προωθήσει ένα token (στον τρέχοντα επόμενο) ενώ στην κατάσταση R περιμένει να παραλάβει ένα token. Η διεργασία κρατάει ένα token είτε αν είναι στην κατάσταση S είτε αν είναι στην κατάσταση R και η προηγούμενή της είναι στην κατάσταση S. Ένα token μετακινείται από την p στην q στο (2) και καταστρέφεται ή δημιουργείται στα (4) και (5) αντίστοιχα. Αυτά τα βήματα καλούνται "βήματα token". Τα βήματα (1) και (3) δεν έχουν κάποια επίδραση στα token και καλούνται "σιωπηλά βήματα". Μετά το βήμα (1) το επόμενο βήμα μιας διεργασίας είναι το (3) και μετά το (3) το βήμα (4) ή το (2) το οποίο δείχνει ότι ο αριθμός των σιωπηλών βημάτων φράσσεται αφού είναι γραμμικά ανάλογος αυτού των βημάτων του token.

Στην ενέργεια (1) αν η q θέλει να στείλει ένα token στην ανενεργή p, η p συμφωνεί να το δεχθεί και κάνει την q προηγούμενή της. Στο (2) η p παρατηρεί ότι η επόμενή της συμφωνεί να δεχθεί το token και γίνεται ανενεργή οπότε μετακινεί το token. Στο (3) η p παρατηρεί ότι η q (από την οποία δέχτηκε το token) έγινε ανενεργή και αρχίζει να μεταδίδει το token στην επόμενή της. Οι τελευταίες δύο ενέργειες αφορούν συμμετρικές περιπτώσεις, όπου (4) δύο tokens που ταξιδεύουν αντίθετα συναντιούνται και (5) δύο ανενεργές διεργασίες είναι προσανατολισμένες ανάποδα. Η πρώτη διεργασία που θα κάνει κάποιο βήμα σπάει την συμμετρία. Στο (4) η p δέχεται το token της q καταστρέφοντας το δικό της. Στο (5) η p δημιουργεί ένα token που θα επαναδιαμορφώσει τον προσανατολισμό της q. Μια διαμόρφωση είναι νόμιμη αν και μόνο αν είναι προσανατολισμένη.

7. Αμοιβαίος αποκλεισμός

7.1. Ένας συγκεντρωτικός αλγόριθμος

Σύμφωνα με τον αλγόριθμο αυτό μία διεργασία επιλέγεται ως συντονιστής. Ο συντονιστής διαθέτει μία ουρά FIFO όπου τοποθετούνται οι διεργασίες που έχουν αιτηθεί για είσοδο στην κρίσιμη περιοχή, αλλά αναγκαστικά περιμένουν μέχρι να ελευθερωθεί η κρίσιμη περιοχή. Όταν μία διεργασία θέλει να εισέλθει στην κρίσιμη περιοχή ρωτά τον συντονιστή. Αν καμία άλλη διεργασία δεν βρίσκεται στην κρίσιμη περιοχή, τότε ο συντονιστής δίνει άδεια εισόδου.

Διαφορετικά:

- Αφήνει τη διεργασία να περιμένει χωρίς απάντηση ή
- Στέλνει αρνητική απάντηση

Όταν η κρίσιμη περιοχή ελευθερώνεται τότε ο συντονιστής επιλέγει την πρώτη διεργασία στην ουρά.

7.2. Ο αλγόριθμος των Ricart & Agrawala

Ο αλγόριθμος αυτός είναι κατανεμημένος και βασίζεται στην ιδέα του συγκεντρωτικού αλγορίθμου που παρουσιάζεται παραπάνω. Απαιτεί την ολική διάταξη των γεγονότων στο σύστημα (π.χ. λογικά ρολόγια Lamport). Σύμφωνα λοιπόν με τον αλγόριθμο αυτό:

- Όταν μία διεργασία επιθυμεί την είσοδό της σε μία κρίσιμη περιοχή στέλνει ένα μήνυμα σε όλες τις άλλες
- Όταν μία διεργασία παραλήπτης λαμβάνει ένα τέτοιο μήνυμα:
    - Αν δεν βρίσκεται στην κρίσιμη περιοχή και δεν επιθυμεί να εισέλθει απαντά OK
    - Αν βρίσκεται στην κρίσιμη περιοχή
        - Δεν απαντά (με όριο χρόνου αναμονής στη μεριά του αποστολέα)
        - Στέλνει αρνητική απάντηση
    - Αν θέλει να εισέλθει στην κρίσιμη περιοχή συγκρίνει τις χρονοσφραγίδες και απαντά αναλόγως
- Όταν μία διεργασία αποστολέας μαζέψει όλες τις θετικές απαντήσεις εισέρχεται στην κρίσιμη περιοχή.

7.3. Ο αλγόριθμος Raymond

Σύμφωνα με τον αλγόριθμο της Raymond οι διεργασίες οργανώνονται σε ένα λογικό γεννητικό δένδρο (logical spanning tree), πράγμα που μειώνει το πλήθος των μηνυμάτων που ανταλλάσσονται σε O(logN).

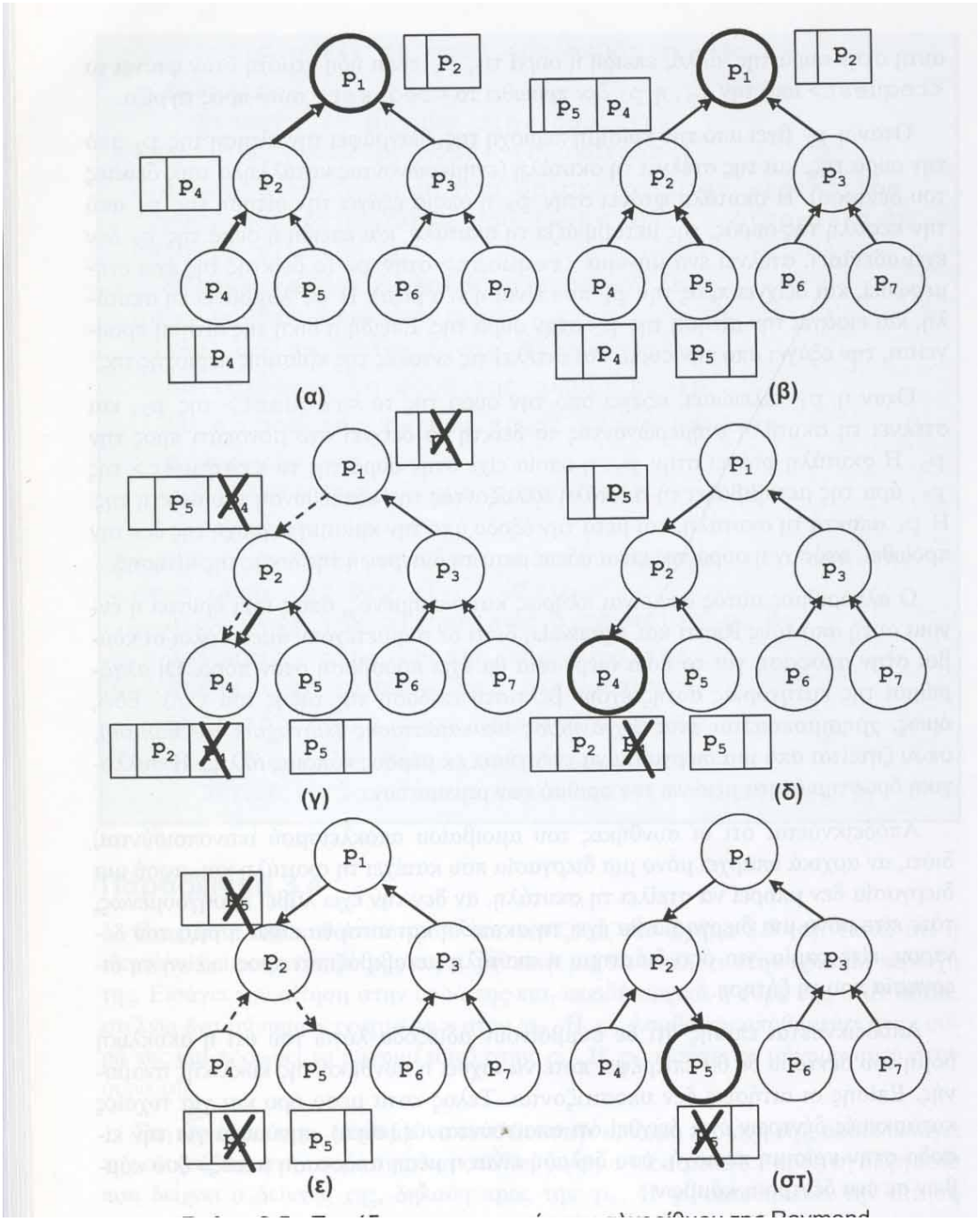
Οι ακμές του δένδρου αυτού είναι κατευθυνόμενες και δείχνουν πάντα στον πατέρα του κόμβου, από τον οποίο ξεκινάνε. Το δικαίωμα εισόδου το έχει και σε αυτή την περίπτωση μία μόνο διεργασία, ο κάτοχος (holder) της σκυτάλης. Η διεργασία αυτή βρίσκεται πάντα στη ρίζα του δένδρου. Η σκυτάλη διασχίζει τις ακμές του δένδρου στο μονοπάτι από τη ρίζα μέχρι τη διεργασία που το ζήτησε. Καθώς μεταβιβάζεται η σκυτάλη, η κατεύθυνση των ακμών αλλάζει κατά μήκος του μονοπατιού, προκειμένου στη νέα ρίζα να βρίσκεται η διεργασία που ζήτησε τη σκυτάλη. Κάθε διεργασία πρέπει να διατηρεί ένα δείκτη σε ένα μονοπάτι που οδηγεί στη

ρίζα, καθώς και μία ουρά όπου βρίσκονται οι αιτήσεις της ίδιας ή και άλλων διεργασιών που δεν έχουν λάβει ακόμα τη σκυτάλη.

Θεωρήστε την κατάσταση που απεικονίζεται στο παρακάτω σχήμα (α). Έστω ότι η σκυτάλη βρίσκεται στην p_1, και η p_4 επιθυμεί να εισέλθει στην κρίσιμη περιοχή της. Εισάγει την αίτηση στην ουρά της και, επειδή αρχικά η ουρά της ήταν άδεια, στέλνει ένα μήνυμα <request> στην p_2. Η p_2 λαμβάνει, αποθηκεύει στην ουρά της και προωθεί το μήνυμα αυτό στην p_1. Η p_1 εισάγει το μήνυμα αυτό στην ουρά της.

Στη συνέχεια η p_5 επιθυμεί να εισέλθει στην κρίσιμη περιοχή της, οπότε εισάγει την αίτησή της στην ουρά της, και στέλνει μήνυμα <request> προς τη διεργασία που δείχνει ο δείκτης της, δηλαδή προς την p_2. Η p_2 αποθηκεύει την αίτηση αυτή στην ουρά της, αλλά, επειδή η ουρά της p_2 είναι ήδη γεμάτη όταν φτάνει το <request> από την p_5, η p_2 δεν προωθεί το <request> αυτό προς τη ρίζα.

Όταν η p_1 βγει από την κρίσιμη περιοχή της, διαγράφει την αίτηση της p_2 από την ουρά της, και της στέλνει τη σκυτάλη (ενημερώνοντας κατάλληλα τους δείκτες του δένδρου). Η σκυτάλη φτάνει στην p_2 , η οποία είχε στην ουρά της το <request> της p_5, και μεταβιβάζει τη σκυτάλη και το <request> της p_5 στην p_4. Η p_4 διαγράφει την αίτησή της από την ουρά της, η οποία πλέον έχει μόνο την αίτηση που έλαβε από την p_2. Όταν η p_4 ολοκληρώσει την κρίσιμη περιοχή της βλέπει ότι υπάρχει αίτημα από την p_2 άρα της μεταβιβάζει τη σκυτάλη αλλάζοντας την κατεύθυνση του δείκτη της. Η p_5 αποκτά τη σκυτάλη, και μετά την έξοδο από την κρίσιμη περιοχή της δεν την προωθεί, εφόσον η ουρά της είναι άδεια μετά τη διαγραφή της δικής της αίτησης.


P1
P2
P3
P4
P5
P6
P7
(α)
(β)
(γ)
(δ)
(ε)
(στ)

8. Βιβλιογραφία

- Introduction to Distributed Algorithms, Gerard Tel, Cambridge University Press, 1994
- Distributed Algorithms, Nancy Lynch, Morgan Kaufman, 1996
- Κατανεμημένα Συστήματα με Java, Ι. Κ Κάβουρας, Ι. Ζ. Μίλης, Γ. Β. Ξυλωμένος, Α. Α. Ρουκουνάκη, Εκδόσεις Κλειδάριθμος, 2η έκδοση, 2005.