ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΝΙΚΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΕΡΓΑΣΤΗΡΙΟ ΜΙΚΡΟΕΠΕΞΕΡΤΑΣΤΩΝ ΚΑΙ ΥΛΙΚΟΥ

Διπλωματική εργασία με θέμα:

Αξιολόγηση Λογισμικού Ανίχνευσης Δικτυακών Εισβολών με Υποβοήθηση από Υλικό

Δημόπουλος Βασίλης
Α. Μ. 1999030015

Εξεταστική Επιτροπή:
Αναπλ. Καθ. Πνευματικάτος Διονύσιος (επιβλέπων)
Καθ. Δόλλας Απόστολος
Επίκ. Καθ. Σαμολαδάς Βασίλης

Περιεχόμενα

1. Εισαγωγη 1

2. Περιγραφή του Snort 5

2.1 Οι κανόνες του Snort 5

2.1.1 Οι επικεφαλίδες των κανόνων 8
2.1.2 Οι ρυθμίσεις (options) των κανόνων 11
2.1.3 Βοηθητικές ρυθμίσεις 11
2.1.4 Ρυθμίσεις περιεχομένου 12
2.1.5 Ρυθμίσεις μη-περιεχομένου... 13
2.1.6 Ρυθμίσεις ολοκληρωμένης αναζήτησης 16

2.2 Ο Βελτιστοποιητής Κανόνων 17

2.2.1 Η τυπική επεξεργασία κανόνων στο Snort 17
2.2.2 Η δημιουργία των ομάδων από κανόνες 18
2.2.3 Η επιλογή μίας ομάδας 19
2.2.4 Ο Βελτιστοποιητής κανόνων στην πράξη 20
2.2.5 Μοναδικοί κανόνες και Γενικοί κανόνες 22
2.2.6 Μοναδικές συγκρούσεις 23
2.2.7 Καθορισμένες και μη καθορισμένες συγκρούσεις 24

2.3 Η μηχανή πολλαπλής αναζήτησης 25

2.3.1 Παραμετρική επιθεώρηση κανόνων 26
2.3.2 Επιθεώρηση κανόνων με πολλαπλή αναζήτηση 27
2.3.3 Αναζήτηση στο Snort και κατηγορίες αναζητήσεων 27
2.3.4 Αναζήτηση περιεχομένου 29
2.3.5 Οι αλγόριθμοι πολλαπλής αναζήτησης 30
2.3.6 Αναζήτηση χωρίς περιεχόμενο 31
2.3.7 Επεξεργασία γεγονότων 32
2.3.8 Τεχνικές προστασίας από επιθέσης στέρησης εξυπηρέτησης (DoS).32

2.4 Προεπεξεργαστές 33
2.5 Οι τρόποι λειτουργίας του Snort 34
2.6 Σύνοψη 35

3. Η θεωρία των αλγορίθμων αναζήτησης 36

3.1 Ο αλγόριθμος Aho-Corasick 37

3.1.1 Μηχανή αναγνώρισης συμβολοσειρών 37
3.1.2 Κατασκευή των συναρτήσεων μετάβασης (goto), αποτυχίας (failure) και εξόδου (output) 41
3.1.3 Ιδιότητες των αλγορίθμων *Simple_AC*, *AC_goto* και *AC_fail* 47
3.1.4 Χρονική πολυπλοκότητα των αλγορίθμων *Simple_AC*, *AC_goto* και *AC_fail* 50

3.1.5 Απαλοιφή των μεταβάσεων αποτυχίας (failure transitions) 53

3.2 Ο αλγόριθμος Boyer-Moore 56

3.2.1 Μια ανεπίσημη περιγραφή 57
3.2.2 Ένα παράδειγμα [2] 60
3.2.3 Ο αλγόριθμος 62
3.2.4 Θεωρητική ανάλυση 65
3.2.5 Ιστορική εξέλιξη του αλγορίθμου – η παραλλαγή του Horspool 73

3.3 Ο αλγόριθμος Wu-Manber 75

3.3.1 Γενική περιγραφή του αλγορίθμου 76
3.3.2 Το στάδιο προεπεξεργασίας 78
3.3.3 Το στάδιο της αναζήτησης 82
3.3.4 Μια πρόχειρη εκτίμηση του χρόνου εκτέλεσης 83
3.3.5 Πειραματικά αποτελέσματα 84

3.4 Σύνοψη και εμπειρική σύγκριση 86

4. Υλοποίηση των αλγορίθμων αναζήτησης συμβολοσειρών 88

4.1 Υλοποίηση του Aho-Corasick 88

4.1.1 Διαφορετικές υλοποιήσεις 89
4.1.2 Εναλλακτική υλοποίηση – Aho-Corasick μαζί με Boyer-Moore 97
4.1.3 Υποστήριξη αναζήτησης χωρίς διαφοροποίηση μικρών-
-κεφαλαίων 101

4.2 Υλοποίηση του Boyer-Moore 106

4.2.1 Υποστήριξη αναζητήσεων χωρίς διαφοροποίηση 108

4.3 Υλοποίηση του αλγορίθμου Wu-Manber 109

4.3.1 Υποστήριξη αναζητήσεων χωρίς διαφοροποίηση 111

4.4 Σύνοψη 113

5. Περιγραφή του συνολικού συστήματος TUCNIDS και των φίλτρων υλικού.....114

5.1 Η αρχικοποίηση του TUCNIDS 116

5.1.1 Το διάβασμα των κανόνων 116
5.1.2 Η βελτιστοποίηση των κανόνων 118
5.1.3 Αναδιάταξη των ομάδων από κανόνες 122
5.1.4 Στατικός προσδιορισμός πιθανών συγκρούσεων 123
5.1.5 Επιλογή αλγορίθμου αναζήτησης 124

5.2 Ο έλεγχος της διερχόμενης κίνησης δικτύου 126

5.2.1 Κατηγοριοποίηση επικεφαλίδας πακέτου 127
5.2.2 Εξέταση δεδομένων πακέτου 128

5.3 Συνδυασμός του TUCNIDS με φίλτρα σε υλικό (hardware) 130

5.3.1 Φίλτρο κατηγοριοποίησης επικεφαλίδας πακέτου (header classification) 131
5.3.2 Φίλτρο προσεγγιστικής αναγνώρισης συμβολοσειρών (approximate string matching) 134

5.4 Θεωρητική ανάλυση της απόδοσης των φίλτρων υλικού 137

5.4.1 Μείωση του αριθμού των πακέτων που προωθούνται στο λογισμικό 137
5.4.2 Μείωση του συνολικού χρόνου επεξεργασίας των πακέτων 139

5.5 Ανακεφαλαίωση 142

6. Πειραματικές μετρήσεις και αποτελέσματα 143

6.1 Οι συνθήκες των πειραμάτων 144

6.1.1 Κατανομή του μήκους των πακέτων 147

6.2 Σύγκριση της ακρίβειας του TUCNIDS και του Snort 148
6.3 Σύγκριση της ταχύτητας του TUCNIDS και του Snort 150

6.3.1 Μεταβολή του χρόνου εκτέλεση ως προς την πιθανότητα *h* 152
6.3.2 Μεταβολή του χρόνου εκτέλεση ως προς την πιθανότητα *p* 154
6.3.3 Μετρήσεις του χρόνου επεξεργασίας πακέτων για το TUCNIDS 155
6.3.4 Μέγιστοι ρυθμοί επεξεργασίας 158

6.4 Χρήση του φίλτρου κατηγοριοποίησης επικεφαλίδων 160

6.4.1 Επίδραση του φίλτρου κατηγοριοποίησης στον χρόνο εκτέλεσης 161
6.4.2 Μεταβολή του χρόνου εκτέλεσης ως προς την πιθανότητα *h* 163
6.4.3 Επίδραση του φίλτρου κατηγοριοποίησης στον χρόνο επεξεργασίας πακέτων 164

6.5 Χρήση του φίλτρου προσεγγιστικής αναγνώρισης συμβολοσειρών σε συνδυασμό με το φίλτρο κατηγοριοποίησης 167

6.5.1 Επίδραδη των δύο φίλτρων στον χρόνο εκτέλεσης 168
6.5.2 Μεταβολή του χρόνου εκτέλεσης ως προς την πιθανότητα *p* 171
6.5.3 Επίδραση των δύο φίλτρων στον χρόνο επεξεργασίας πακέτων 173

6.6 Ποσοστιαία συνεισφορά κάθε φίλτρου 175
6.7 Συμπεράσματα 177

Επίλογος 178

Βιβλιογραφικές αναφορές 181
Διαδικτυακές αναφορές 183

Κεφάλαιο 1.

Εισαγωγή.

Τα τελευταία χρόνια παρατηρείται εκρηκτική ανάπτυξη του Διαδικτύου καθώς γίνεται ολοένα ευκολότερη η πρόσβαση σε αυτό. Παράλληλα διαπιστώνεται συνεχής βελτίωση της ποιότητα των παρεχομένων από αυτό υπηρεσιών ως συνέπεια της αύξησης των ταχυτήτων πρόσβασης. Ολοένα και περισσότεροι ιδιώτες αλλά και εταιρίες χρησιμοποιούν το Διαδίκτυο για ενημέρωση, ψυχαγωγία, αγορές, επιχειρηματικές δραστηριότητες και άλλους σκοπούς.

Καθώς το Διαδίκτυο γίνεται συνεχώς πιο προσφιλές και αυξάνεται ο αριθμός των ανθρώπων που το χρησιμοποιούν, γίνεται εντονότερη η ανάγκη για συστήματα προστασίας, συστήματα που θα αποτρέπουν την πρόσβαση σε μη εξουσιοδωτημένα άτομα ή την εκτέλεση ασυνήθιστων και, πιθανώς, επικίνδυνων λειτουργιών.

Ένα πολύ διαδεδομένο σύστημα προστασίας είναι τα λεγόμενα firewall, τα οποία μπορούν να υλοποιηθούν σε λογισμικό ή σε υλικό. Η λειτουργία των συστήματων τύπου firewall είναι να εξετάζουν τις επικεφαλίδες (header) των διερχομένων πακέτων και να επιτρέπουν ή να απαγορεύουν την πρόσβαση σε συγκεκριμένους χρήστες, ενώ μπορούν να ελέγχουν και το είδος της επικοινωνίας που γίνεται. Με ένα firewall μπορεί, για παράδειγμα, ένας οργανισμός να προστατεύει το εσωτερικό του δίκτυο (intranet) απαγορεύοντας την πρόσβαση σε υπολογιστές που δεν ανήκουν στο δίκτυο αυτό.

Μια λογική επέκταση των συστημάτων firewall, η οποία έχει αρχίσει να αναπτύσεται εδώ και λίγα χρόνια, είναι τα συστήματα ανίχνευσης εισβολών δικτύου (Network Intrusion Detection Systems – NIDS). Η διαφορά ενός συστήματος ανίχνευσης εισβολών δικτύου από ένα firewall είναι ότι το NIDS, εκτός από το να ελέγχει τις επικεφαλίδες των διερχομένων πακέτων, μπορεί να εξετάζει και τα περιεχόμενα (payload) των πακέτων για την ύπαρξη συγκεκριμένων ύποπτων συμβολοσειρών. Καθώς είναι σαφώς πιο περίπλοκα από τα firewall, τα συστήματα ανίχνευσης εισβολών δικτύου υλοποιούνται, μέχρι τώρα, μόνο σε λογισμικό. Ένα από τα πλέον διαδεδομένα

συστήματα ανίχνευσης εισβολών δικτύου είναι το Snort, το οποίο διατίθεται δωρεάν μαζί με τον πηγαίο κώδικά του. Παρά την δωρεάν του διάθεση, αποτελεί ένα από τα καλύτερα συστήματα ανίχνευσης εισβολών και χρησιμοποιείται ως σημείο αναφοράς σε αυτή τη διπλωματική.

Τα συστήματα ανίχνευσης εισβολών δικτύου παρέχουν μεγάλη ευελιξία στη χρήση τους και πολλές δυνατότητες ελέγχου της διερχόμενης κίνησης. Η λειτουργία, όμως, της αναζήτησης συμβολοσειρών μέσα στα δεδομένα ενός πακέτου έχει το μειονέκτημα ότι είναι αρκετά χρονοβόρα και απαιτητική σε πόρους συστήματος. Σε περίπτωση που η διερχόμενη κίνηση είναι μεγαλύτερη από αυτή που μπορεί να επεξεργαστεί το σύστημα ανίχνευσης επέρχεται κορεσμός του συστήματος, γεγονός που συνεπάγεται καθυστερήσεις των πακέτων και μεγάλύτερο ποσοστό απώλειας πακέτων λόγο έλλειψης χώρου, ενώ επίσης υπάρχει το ενδεχόμενο ορισμένα επικίνδυνα πακέτα καταφέρουν να διαφύγουν από τον έλεγχο.

Το λογισμικό παρέχει ευελιξία ως προς την υλοποίηση και τις λειτουργίες του συστήματος ανίχνευσης. Καθώς, όμως, οι ταχύτητες μεταφοράς δεδομένων από το δίκτυο αυξάνονται με πολύ μεγαλύτερο ρυθμό από ότι αυξάνεται η ταχύτητα του λογισμικού ανίχνευσης, είναι λογικό να περιμένουμε πως στο εγγύς μέλλον το πρόβλημα του κορεσμού θα είναι ακόμα πιο έντονο.

Μια λύση στο πρόβλημα του κορεσμού πιθανό να προσφέρει η χρήση υλικού, καθώς μπορεί να παρέχει ρυθμούς επεξεργασίας πολύ μεγαλύτερους από εκείνους του λογισμικού. Δυστυχώς, όμως, δεν είναι εφικτή η υλοποίηση ενός ολοκληρομένου συστήματος ανίχνευσης αποκλειστικά σε υλικό λόγω της πολυπλοκότητας που έχει ένα τέτοιο σύστημα αλλά κυρίως λόγω της απαιτούμενης ευελιξίας: σε ένα σύστημα ανίχνευσης εισβολών τα δεδομένα (για παράδειγμα οι διευθύνσεις υπό επιτήρηση ή οι συμβολοσειρές προς αναζήτηση) μπορούν να αλλάζουν πολύ εύκολα και συχνά.

Τα παραπάνω γεγονότα αποτέλεσαν και το κίνητρο αυτής της διπλωματικής, στην οποία εξετάζονται διάφοροι τρόποι επιτάχυνσης και διευκόλυνσης της λειτουργίας των συστημάτων ανίχνευσης εισβολών δικτύου με τη χρήση κατάλληλων διατάξεων υλικού.

Η εργασία αυτή ολοκληρώθηκε σε δύο στάδια. Το πρώτο στάδιο της εργασίας αφορά την υλοποίηση ενός σύστηματος ανίχνευσης εισβολών, το οποίο ονομάσαμε TUCNIDS (Technical University of Crete Network Intrusion Detection System). Το

σύστημα αυτό παρέχει μόνο την βασική λειτουργία ενός συστήματος ανίχνευσης εισβολών, στην οποία κάθε χρησιμοποιούμενος κανόνας αντιστοιχεί σε μία συμβολοσειρά προς αναζήτηση. Ο λόγος που υποστηρίζουμε μόνο τη βασική λειτουργία ενός συστήματος ανίχνευσης στο TUCNIDS είναι γιατί θέλαμε να εξετάσουμε εάν μπορεί να επιταχυνθεί η λειτουργία της αναζήτησης συμβολοσειρών (content matching) μέσω καλύτερων υλοποιήσεων των αλγορίθμων αναζήτησης. Σαν σημείο αναφοράς αλλά και σαν μέτρο σύγκρισης για το TUCNIDS χρησιμοποιήθηκε το σύστημα Snort, καθώς θεωρείται ένα από τα καλύτερα συστήματα ανίχνευσης που υπάρχουν.

Στο δεύτερο και τελευταίο στάδιο της διπλωματικής εξετάζουμε κατά πόσο θα μπορούσε να συνδυαστεί η λειτουργία του TUCNIDS με φίλτρα υλοποιημένα σε υλικό. Ύστερα από μελέτη του προβλήματος καταλήξαμε σε δύο περιπτώσεις φίλτρων υλικού με τις οποίες μπορεί να υποβοηθηθεί το λογισμικό. Η πρώτη περίπτωση που εξετάζουμε είναι ένα φίλτρο κατηγοριοποίησης επικεφαλίδων, το οποίο θα εκτελεί πολύ γρήγορα την διαδικασία της κατηγοριοποίησης των πακέτων με βάση τις επικεφαλίδες τους, διαδικασία που απαιτεί σημαντικό χρόνο όταν γίνεται στο λογισμικό, ενώ ταυτόχρονα θα μειωνεί τον αριθμό των πακέτων που επεξεργάζονται από το λογισμικό. Το δεύτερο φίλτρο υλικού που προτείνουμε είναι ένα φίλτρο προσεγγιστικής αναγνώρισης συμβολοσειρών, η χρήση του οποίου μπορεί να μειώσει τον αριθμό των πακέτων που επεξεργάζονται από το λογισμικό.

Η λειτουργία των φίλτρων υλικού προσομοιώθηκε σε θεωρητικό επίπεδο (δηλαδή δεν έγινε πραγματική υλοποίηση των φίλτρων) με στόχο να υπολογιστούν τα οφέλη για το σύστημα ανίχνευσης εισβολών στην περίπτωση ιδανικά γρήγορων φίλτρων υλικού. Από τη θεωρητική ανάλυση των φίλτρων αλλά και την προσωμοίωση της λειτουργίας τους καταλήγουμε στο συμπέρασμα ότι τα φίλτρα υλικού μπορούν να μειώσουν σημαντικά τον χρόνο που απαιτείται από το λογισμικό για την επεξεργασία ενός αριθμού πακέτων.

Η διάρθρωση της υπόλοιπης εργασίας έχει ώς εξής. Αρχικά, στο δεύτερο κεφάλαιο παρουσιάζεται ο τρόπος λειτουργίας του Snort καθώς και τα σημαντικότερα χαρακτηριστικά του. Καθώς το Snort είναι από τα πλέον επιτυχημένα συστήματα ανίχνευσης εισβολών, μπορεί να μας δώσει σημαντικές πληροφορίες για την κατασκευή ενός καλού συστήματος ανίχνευσης.

Το τρίτο κεφάλαιο είναι αφιερωμένο στη θεωρία των αλγορίθμων αναζήτησης συμβολοσειρών που κρίθηκαν ως πιο κατάλληλοι για χρήση στο σύστημα ανίχνευσης εισβολών δικτύου. Οι αλγόριθμοι αναζήτησης που περιγράφονται στο συγκεκριμένο κεφάλαιο είναι ο Aho-Corasick, ο Boyer-Moore και ο Wu-Manber.

Στο τέταρτο κεφάλαιο περιγράφεται ο τρόπος με τον οποίο υλοποιήθηκε κάθε ένας από τους αλγορίθμους αναζήτησης που τελικά χρησιμοποιήθηκαν στο TUCNIDS και αναλύονται οι σχεδιαστικές επιλογές που έγιναν.

Το πέμπτο κεφάλαιο ασχολείται με τον τρόπο λειτουργίας του TUCNIDS συνολικά. Επίσης παρουσιάζεται η θεωρητική λειτουργία των δύο φίλτρων υλικού που θα χρησιμοποιηθούν για την υποβοήθηση του συστήματος ανίχνευσης εισβολών, καθώς και μια θεωρητική ανάλυση του βαθμού στον οποίο τα φίλτρα αυτά μπορούν να μειώσουν τον φόρτο σε ένα σύστημα ανίχνευσης εισβολών υλοποιημένου σε λογισμικό.

Τέλος, στο έκτο κεφάλαιο παρουσιάζονται τα αποτελέσματα από τη σύγκριση του TUCNIDS με το Snort ως προς τον αριθμό των επικίνδυνων πακέτων που εντοπίζουν αλλά και ως προς την ταχύτητά τους. Επιπλέον, παρουσιάζονται τα αποτελέσματα της χρήσης των φίλτρων υλικού σε συνδυασμό με το TUCNIDS.

Κεφάλαιο 2.

Περιγραφή του Snort.

Στον τομέα των συστημάτων ανίχνευσης εισβολών δικτύου, το Snort αποτελεί ένα από τα πλέον διαδεδομένα, αποτελεσματικά και αξιόπιστα συστήματα. Πρόκειται για μία εφαρμογή ανοιχτού κώδικα που έχει αναπτυχθεί ώστε να λειτουργεί σε μία μεγάλη ποικιλία λειτουργικών συστημάτων (Windows, Solaris, συστήματα τύπου Unix και άλλα) η οποία διατίθεται δωρεάν μέσω της ιστοσελίδας www.snort.org. Το γεγονός της ελεύθερης διάθεσης του πηγαίου κώδικα αποτέλεσε κίνητρο και πρόκληση για αρκετούς ανθρώπους, με αποτέλεσμα τη σταδιακή δημιουργία μίας μεγάλης κοινότητας από άτομα που ασχολούνται, τακτικά ή περιστασιακά, με το Snort και αναζητούν τρόπους ώστε να γίνει ακόμα πιο αποτελεσματικό και γρήγορο. Με το πέρασμα του χρόνου και μέσα από πολλές εξελίξεις στη μορφή του, το Snort έχει φτάσει πλέον στην έκδοση 2.1.3, η οποία μπορεί να υποστηρίζει ταχύτητες σύνδεσης μέχρι και της τάξης των λίγων Gigabit.

Στη συνέχεια του κεφαλαίου παρουσιάζεται οι ο τρόπος λειτουργίας του Snort καθώς και ορισμένα από τα κυριότερα χαρακτηριστικά και υποσυστήματά του.

2.1 Οι κανόνες του Snort.

Οι κανόνες του Snort είναι αυτοί που υποδηλώνουν όχι μόνο τι θέλουμε να ψάξει το σύστημα στα εισερχόμενα πακέτα, αλλά επίσης και τι ενέργεια θέλουμε να εκτελέσει σε περίπτωση που βρεθεί πακέτο που να ικανοποιεί όλες τις συνθήκες του κανόνα.

Σχεδόν όλοι οι κανόνες έχουν γραφτεί με στόχο την ανίχνευση και απαγόρευση διέλευσης σε πακέτα που περιέχουν «υπογραφές εισβολής» (intrusion signatures), δηλαδή ορισμένα χαρακτηριστικά στοιχεία ενός συγκεκριμένου τύπου επίθεσης, τα οποία μπορεί να είναι καθορισμένες παράμετροι στην επικεφαλίδα του πακέτου (packet

header) ή/και στα δεδομένα του πακέτου (payload). Τέτοιου είδους πακέτα συνήθως αποτελούν μέρος προσπάθειας που γίνεται με στόχο να πληγεί η ακεραιότητα του υπολογιστή του χρήστη και δεν θέλουμε να επιτραπεί η διέλευσή τους. Δίνεται, όμως, και η δυνατότητα συγγραφής κανόνων οι οποίοι έχουν ως στόχο να *επιτρέπουν* τη διέλευση πακέτων που πληρούν συγκεκριμένες προϋποθέσεις.

Οι κανόνες του Snort βρίσκονται κατηγοριοποιημένοι σε διάφορα αρχεία με κατάληξη ".rules", με το κάθε αρχείο να περιέχει, συνήθως, κανόνες που συμπεριλαμβάνονται σε ένα συγκεκριμένο τύπο επίθεσης απέναντι στο σύστημα. Για παράδειγμα, το αρχείο "ftp.rules" περιέχει κανόνες που είναι γραμμένοι με στόχο πακέτα που σχετίζονται με εφαρμογές ftp.

Σε κάθε έκδοση του Snort που είναι διαθέσιμη στο δίκτυο συμπεριλαμβάνονται και συγκεκριμένα αρχεία κανόνων, ενώ ανά περιόδους γίνονται διαθέσιμα νέα αρχεία κανόνων ώστε να είναι εύκολος ο εκσυγχρονισμός του Snort απέναντι σε νέες μορφές επίθεσης. Επιπλέον, ο χρήστης μπορεί να επιλέγει ποια αρχεία κανόνων θέλει να χρησιμοποιούνται στο σύστημα του κάνοντας απλές τροποποιήσεις σε ένα αρχείο ρύθμισης (configuration) του Snort. Το σημαντικότερο, όμως, είναι πως ο χρήστης έχει τη δυνατότητα να γράφει νέους κανόνες, επιτρέποντας έτσι την προσαρμογή του Snort στις απαιτήσεις του κάθε χρήστη και στο είδος κίνησης που αναμένεται να υπάρχει στο συγκεκριμένο δίκτυο.

Το Snort χρησιμοποιεί μια απλή γλώσσα περιγραφής κανόνων η οποία όμως προσφέρει πληθώρα επιλογών που την κάνουν αρκετά ισχυρή και ευέλικτη. Υπάρχουν ορισμένες απλές αρχές που χρειάζεται να θυμάται κάποιος που ενδιαφέρεται να γράψει δικούς του κανόνες για το Snort.

Οι περισσότεροι κανόνες γράφονται σε μια μόνο γραμμή. Σε εκδώσεις του Snort πριν από την 1.8 κάτι τέτοιο ήταν απαραίτητο, όμως στις νεότερες εκδόσεις είναι εφικτό να έχουμε κανόνες που καταλαμβάνουν περισσότερο από μία γραμμή, αρκεί στο τέλος κάθε ενδιάμεσης γραμμής να βρίσκεται ο χαρακτήρας '\'.

Όλοι οι κανόνες αποτελούνται από δύο λογικές ενότητες, την επικεφαλίδα του κανόνα (rule header) και τις ρυθμίσεις του κανόνα (rule option). Στην επικεφαλίδα του κανόνα περιέχονται οι πληροφορίες για την ενέργεια που θα εκτελέσει το Snort σε περίπτωση που επαληθευτεί ο κανόνας, το πρωτόκολλο, οι διευθύνσεις IP του αποστολέα

και του παραλήπτη μαζί με τις κατάλληλες μάσκες δικτύου (netmasks) καθώς και οι θύρες (ports) αποστολέα και παραλήπτη. Το τμήμα των ρυθμίσεων του κανόνα περιλαμβάνει τα μηνύματα προειδοποίησης (alerts) που θα παράγει το Snort σε περίπτωση επαλήθευσης καθώς και πληροφορίες για το ποια τμήματα του εισερχόμενου πακέτου πρέπει να εξεταστούν ώστε να αποφασίσουμε αν το πακέτο είναι ύποπτο, οπότε το Snort θα δράσει κατάλληλα, ή όχι.

Ένας τυπικός κανόνας του Snort φαίνεται παρακάτω.

```
alert tcp any any -> 192.168.1.0/24 111 (content:"|00 01 86 a5|"; msg:"mountd
access";)
```

Οτιδήποτε πριν την πρώτη παρένθεση αποτελεί την επικεφαλίδα του κανόνα, ενώ το τμήμα μέσα στις παρενθέσεις αποτελεί τις ρυθμίσεις του κανόνα. Στο κομμάτι των ρυθμίσεων του κανόνα, οι λέξεις που βρίσκονται αμέσως πριν από ‘:’ είναι οι εντολές ή λέξεις κλειδιά των ρυθμίσεων. Αξίζει να σημειωθεί οτι το τμήμα των ρυθμίσεων δεν είναι αναγκαίο για τη δήλωση ενός κανόνα, όμως χρησιμοποιείται για να δίνεται ένας πιο ακριβής προσδιορισμός των πακέτων για τα οποία θέλουμε να δημιουργηθεί κάποια προειδοποίηση ή των πακέτων που θέλουμε να καταγράψουμε ή να απορρίψουμε. Όλα τα προσδιοριστικά στοιχεία ενός κανόνα πρέπει να ισχύουν για ένα πακέτο ώστε να εκτελεστεί η ενέργεια του κανόνα. Αν εξετάσουμε όλα μαζί τα προσδιοριστικά στοιχεία ενός κανόνα μπορούμε να θεωρήσουμε ότι δημιουργούν μια μεγάλη πρόταση λογικού ΚΑΙ. Αντίστοιχα, οι διάφοροι κανόνες μέσα σε μία βιβλιοθήκη κανόνων του Snort μπορούν να θεωρηθούν σαν μια μεγάλη πρόταση λογικού Ή.

Στη συνέχεια υπάρχει μια λεπτομερής περιγραφή των στοιχείων της επικεφαλίδας ενός κανόνα καθώς και των διαφόρων ρυθμίσεων που μπορούμε να χρησιμοποιήσουμε στο αντίστοιχο τμήμα.

2.1.1 Οι επικεφαλίδες των κανόνων.

Η επικεφαλίδα κάθε κανόνα περιέχει τις πληροφορίες που καθορίζουν από που έρχεται το ύποπτο πακέτο, που πηγαίνει και τι είδους είναι, καθώς επίσης και τι πρέπει να κάνουμε στην περίπτωση που μας έρθει ένα πακέτο με τα συγκεκριμένα χαρακτηριστικά του κανόνα.

Το πρώτο στοιχείο σε κάθε κανόνα είναι η ενέργεια που θα εκτελεστεί (action), δηλαδή ο τρόπος που θα αντιδράσει το Snort σε περίπτωση που βρεθεί ένα πακέτο το οποίο ικανοποιεί όλα τα στοιχεία του κανόνα. Υπάρχουν οι εξής πέντε προκαθορισμένες ενέργειες στο Snort:

1. alert – δημιουργία μηνύματος προειδοποίησης και καταγραφή του πακέτου.
2. log – καταγραφή του πακέτου.
3. pass – επιτρέπεται η διέλευση του πακέτου.
4. activate – δημιουργία προειδοποίησης και ενεργοποίηση ενός dynamic κανόνα.
5. dynamic – παραμένει ανενεργός μέχρι να επαληθευτεί ο αντίστοιχος activate κανόνας και μετά ενεργεί ως κανόνας τύπου log.

Ο χρήστης μπορεί να καθορίσει και νέους τύπους ενέργειας κανόνα και να τους συσχετίσει με ένα ή περισσότερα προγράμματα εξόδου. Για παράδειγμα, η ακόλουθη δήλωση καθορίζει ένα νέο τύπο κανόνων οι οποίοι καταγράφουν πακέτα στο syslog και σε βάση δεδομένων MySQL:

```
ruletype redalert
{
        type alert
        output alert_syslog: LOG_AUTH LOG_ALERT
        output database: log, mysql, user=snort dbname=snort host=localhost
}
```

Το επόμενο πεδίο στην επικεφαλίδα ενός κανόνα είναι το πρωτόκολλο. Μέχρι στιγμής το Snort υποστηρίζει τέσσερα πρωτόκολλα στην ανάλυση εισερχομένων

πακέτων, τα πρωτόκολλα tcp, udp, ip και icmp. Στο μέλλον πιθανόν να υπάρξει υποστήριξη και για άλλα πρωτόκολλα, όπως ARP, IGRP, OSPF και άλλα.

Αμέσως μετά το πρωτόκολλο βρίσκεται το τμήμα που περιέχει τις πληροφορίες για τις διευθύνσεις IP και τις θύρες του κανόνα. Θα περιγράψουμε πρώτα τη μορφή των διευθύνσεων IP.

Η λέξη-κλειδί '*any*' μπορεί να χρησιμοποιηθεί όταν θέλουμε τον κανόνα να ισχύει για οποιαδήποτε διεύθυνση IP. Κάθε διεύθυνση αποτελείται από μία συνεχόμενη διεύθυνση IP (δηλαδή τέσσερις ακέραιους αριθμούς από 0 μέχρι 255 με '.' ανάμεσα τους) και ένα μπλοκ CIDR. Το μπλοκ CIDR το οποίο υποδηλώνει τη μάσκα δικτύου που εφαρμόζεται στην διεύθυνση κάθε πακέτου πρωτού γίνει η σύγκριση με τη διεύθυνση του κανόνα. Για παράδειγμα, ο συνδυασμός 192.168.1.0/24 υποδηλώνει την ομάδα διευθύνσεων από 192.168.1.0 μέχρι 192.168.1.255. Η χρήση του CIDR μας επιτρέπει να δηλώνουμε με ένα σύντομο τρόπο μεγάλες ομάδες διευθύνσεων. Εάν κάποιο εισερχόμενο πακέτο έχει τη διεύθυνσή του μέσα στο συγκεκριμένο πεδίο τιμών τότε επαληθεύεται αυτό το κομμάτι του κανόνα.

Στις διευθύνσεις μπορεί να εφαρμοστεί ένας τελεστής αντιστροφής, ο οποίος συμβολίζεται με '!' και λέει στο Snort ότι ταιριάζουν όλες οι διευθύνσεις εκτός από την αναγραφόμενη. Για παράδειγμα, στην ακόλουθη επικεφαλίδα κανόνα δε μας ενδιαφέρει η διεύθυνση αποστολέα, καθώς είναι ρυθμισμένη σε *any*, και θέλουμε η διεύθυνση του παραλήπτη να μην ανήκει στην ομάδα 192.168.1.0/24.

```
alert tcp any any -> !192.168.1.0/24 111 (…)
```

Το Snort υποστηρίζει επίσης τη δήλωση ομάδας από συγκεκριμένες διευθύνσεις. Μια ομάδα διευθύνσεων εσωκλείεται σε αγκύλες και κάθε διεύθυνση διαχωρίζεται από τις γειτονικές τις με ','. Δεν επιτρέπεται η ύπαρξη κενού (space) στο εσωτερικό μιας ομάδας διευθύνσεων. Για παράδειγμα, η έκφραση ' ![192.168.1.0/24,10.1.1.0/24] ' μας λέει ότι ενδιαφερόμαστε για τις διευθύνσεις που δεν ανήκουν σε καμία από τις δύο ομάδες διευθύνσεων.

Εκτός από τις διευθύνσεις πρέπει να δηλωθούν και οι θύρες (ports) για τις οποίες ισχύει ο κανόνας. Και σε αυτή την περίπτωση μπορεί να χρησιμοποιηθεί η λέξη *any* αν θέλουμε να δηλώσουμε ότι ο κανόνας ισχύει για όλες τις θύρες. Μπορούμε επίσης να δώσουμε συγκεκριμένη τιμή σε μία θύρα, για παράδειγμα την τιμή 80 για http.

Επιπλέον, υπάρχει η δυνατότητα να δηλώσουμε μία ομάδα συνεχόμενων θυρών χρησιμοποιώντας ':' ανάμεσα στα δύο όρια της ομάδας. Για παράδειγμα, η έκφραση '1:1024' ισχύει για όλες τις θύρες από 1 μέχρι 1024. Όταν δηλώνουμε μία ομάδα μπορούμε να παραλείψουμε το ένα άκρο του ορισμού δηλώνοντας, με τον τρόπο αυτό, ότι ενδιαφερόμαστε για όσες διευθύνσεις είναι μικρότερες ή μεγαλύτερες από μια άλλη. Δηλαδή, μια έκφραση της μορφής '500: ' ισχύει για θύρες μεγαλύτερες οι ίσες από την 500.

Και στην περίπτωση των θυρών υπάρχει ο τελεστής αντιστροφής '!' που μπορεί να εφαρμοστεί πριν από κάθε έκφραση και αντιστρέφει την ισχύ της.

Το τελευταίο συστατικό της επικεφαλίδας ενός κανόνα είναι ο τελεστής κατεύθυνσης. Ο τελεστής αυτός βρίσκεται ανάμεσα στα δύο ζεύγη διεύθυνσης IP-θύρας και συμβολίζει την κατεύθυνση της κίνησης που μας ενδιαφέρει. Συνήθως ο τελεστής κατεύθυνσης γράφεται σαν '->', οπότε το ζεύγος που βρίσκεται αριστερά του αντιστοιχεί στη διεύθυνση IP και τη θύρα του αποστολέα, ενώ το ζεύγος δεξιά του τελεστή αντιστοιχεί στον παραλήπτη. Υπάρχει επίσης και ο αμφίδρομος τελεστής '<>' με τον οποίο δηλώνουμε ότι δε μας ενδιαφέρει ο προσανατολισμός της κίνησης. Ένα παράδειγμα χρήσης του αμφίδρομου τελεστή για την καταγραφή και των δύο πλευρών μίας επικοινωνίας telnet είναι το ακόλουθο.

```
log tcp !192.168.1.0/24 any <> 192.168.1.0/24 23
```

Πρέπει να σημειωθεί ότι δεν υπάρχει τελεστής <- ώστε να υπάρχει συνέπεια στον τρόπο που γράφονται οι κανόνες, δηλαδή το πρώτο ζεύγος να αντιστοιχεί πάντα στον αποστολέα και το δεύτερο στον παραλήπτη.

2.1.2 Οι ρυθμίσεις (options) των κανόνων.

Οι ρυθμίσεις των κανόνων αποτελούν την καρδιά του μηχανισμού ανίχνευσης του Snort καθώς συνδυάζουν ευκολία χρήσης, αποτελεσματικότητα και ευελιξία. Κάθε ρύθμιση σε έναν κανόνα διαχωρίζεται από την επόμενη με το χαρακτήρα ';'. Οι λέξεις-κλειδιά των ρυθμίσεων διαχωρίζονται από τα ορίσματά τους μέσω ':'. Οι ρυθμίσεις των κανόνων υπάγονται στις ακόλουθες τέσσερις κύριες κατηγορίες:

Βοηθητικές: Παρέχουν επιπλέον πληροφορίες για τον κανόνα χωρίς, όμως, να σχετίζονται με την αναζήτηση.

Περιεχομένου: Όλες οι επιλογές του τύπου αυτού προσδιορίζουν δεδομένα που θα αναζητηθούν στο περιεχόμενο (payload) του πακέτου και μπορούν να συσχετίζονται μεταξύ τους.

Μη-περιεχομένου: Εξετάζουν πληροφορίες που δεν βρίσκονται στο περιεχόμενο (payload) του πακέτου.

Ολοκληρωμένης αναζήτησης: Προσδιορίζουν την εκτέλεση ορισμένων γεγονότων που θα συμβούν όταν επαληθευτεί ο κανόνας.

Στη συνέχεια ακολουθεί μια σύντομη περιγραφή των ρυθμίσεων που υπάγονται σε κάθε κατηγορία.

2.1.3 Βοηθητικές ρυθμίσεις.

- **msg:** η ρύθμιση 'msg' καθορίζει το μήνυμα που θα εμφανιστεί μαζί με το περιεχόμενο του πακέτου ή σε μία προειδοποίηση. Περιλαμβάνει απλές συμβολοσειρές κειμένου, στις οποίες χρησιμοποιείται το σύμβολο '\' σαν χαρακτήρας διαφυγής όταν θέλουμε να δηλώσουμε ένα σύμβολο που μπορεί να μπερδέψει τον parser του Snort (όπως τον χαρακτήρα ';').
- **reference:** επιτρέπει στον κανόνα να αναφερθεί σε ένα εξωτερικό σύστημα αναγνώρισης επιθέσεων. Σαν ορίσματα υποστηρίζονται ορισμένα συστήματα καθώς και μοναδικά URL. Χρησιμοποιείται από ένα κατάλληλο plugin εξόδου το οποίο

παρέχει σύνδεση στην τοποθεσία όπου βρίσκονται επιπλέον πληροφορίες για την προειδοποίηση που παράγει ο κανόνας.

- **sid:** χρησιμοποιείται ως αναγνωριστικό ανάμεσα στους διάφορους κανόνες. Κάθε κανόνας πρέπει να έχει και διαφορετικό "sid". Χρησιμοποιείται σε συνδυασμό με τη ρύθμιση "rev".
- **rev:** αντιστοιχεί στον αριθμό αναθεώρησης (revision) του κανόνα. Σε συνδυασμό με τη ρύθμιση "sid" επιτρέπει την ανανέωση και αντικατάσταση των κανόνων με νεότερες πληροφορίες.
- **classtype:** χρησιμοποιείται για την κατηγοριοποίηση των κανόνων σε διαφορετικές κλάσεις. Κάθε κλάση αντιστοιχεί σε ένα διαφορετικό είδος επίθεσης εναντίον του συστήματος, ενώ επίσης έχει και μια καθορισμένη προτεραιότητα. Ο χρήστης έχει τη δυνατότητα να καθορίσει δικές του κλάσεις.
- **priority:** αναθέτει την επιθυμητή προτεραιότητα στον συγκεκριμένο κανόνα. Μπορεί να χρησιμοποιηθεί και σε περίπτωση που θέλουμε να αλλάξουμε την προκαθορισμένη προτεραιότητα της ρύθμισης "classtype". Όσο μικρότερη είναι η τιμή που δίνουμε στη ρύθμιση αυτή, τόσο πιο σημαντικός είναι ο κανόνας (δηλαδή η τιμή 1 αντιστοιχεί στη μεγαλύτερη προτεραιότητα).

2.1.4 Ρυθμίσεις περιεχομένου.

- **content:** αποτελεί ένα από τα σημαντικότερα χαρακτηριστικά του Snort, καθώς επιτρέπει στον χρήστη να ορίζει κανόνες οι οποίοι αναζητούν συγκεκριμένες συμβολοσειρές στο εσωτερικό των πακέτων. Για τους κανόνες που περιέχουν την επιλογή "content" εκτελείται πρώτα ένας έλεγχος του πακέτου με τον κατάλληλο αλγόριθμο και, σε περίπτωση που βρεθεί το ζητούμενο περιεχόμενο, εξετάζονται οι υπόλοιπες παράμετροι. Ο έλεγχος κάνει διαχωρισμό μεταξύ κεφαλαίων και μικρών γραμμάτων (δηλαδή είναι case sensitive). Το όρισμα αυτής της ρύθμισης μπορεί να δοθεί με τη μορφή απλής συμβολοσειράς, με δεκαεξαδική μορφή ή με συνδυασμό των δύο. Το κομμάτι που αντιστοιχεί σε δεκαεξαδική μορφή ορίζεται με '|' στην αρχή και το τέλος, ενώ στο εσωτερικό κάθε χαρακτήρας γράφεται με δύο

δεκαεξαδικά ψηφία και διαχωρίζεται από τον επόμενο με κενό. Ένα παράδειγμα content με συνδυασμό συμβολοσειρών και δεκαεξαδικών είναι το {content:"|5c 00|P|00|I|00|P|00|E|00 5c|";}. Στον ίδιο κανόνα μπορούν να υπάρχουν περισσότερες από μία εντολές “content”. Επίσης, μπορούμε να έχουμε τον τελεστή αντιστροφής πριν από το όρισμα του content, γεγονός που σημαίνει ότι ο κανόνας ισχύει όταν *δεν* υπάρχει το συγκεκριμένο περιεχόμενο μέσα στο πακέτο.

Η εντολή “content” μπορεί να συνδυαστεί με μια ή περισσότερες εντολές τροποποίησης, οι οποίες επηρεάζουν τον τρόπο με τον οποίο θα γίνει η αναζήτηση του περιεχομένου. Αυτές είναι οι:

1. **depth:** καθορίζει μέχρι πόσο βάθος μέσα στο πακέτο θα εξετάσουμε για την ύπαρξη του ζητούμενου περιεχομένου. Για παράδειγμα, αν η τιμή του depth ήταν 5 τότε θα αναζητούσαμε το περιεχόμενο μέσα στα πρώτα 5 bytes των δεδομένων του πακέτου (δηλαδή τα 5 bytes αμέσως μετά τις επικεφαλίδες).
2. **offset:** με την εντολή αυτή ο χρήστης δηλώνει πόσα bytes μετά από την αρχή των δεδομένων θέλει να ξεκινήσει η αναζήτηση για το περιεχόμενο. Δηλαδή ένα offset ίσο με 5 σημαίνει πως θα ξεκινήσει η αναζήτηση 5 bytes μετά την αρχή των δεδομένων.
3. **distance:** για να χρησιμοποιηθεί η εντολή αυτή χρειάζεται προηγουμένως να έχουμε ορίσει δύο content προς αναζήτηση. Η εντολή αυτή υποδηλώνει μέχρι πόσα bytes από το τέλος του πρώτου content θα ψάξουμε για το δεύτερο content. Μπορεί να θεωρηθεί κάτι αντίστοιχο με το depth ως προς τη θέση που τελειώνει το πρώτο content.
4. **within:** και αυτή η εντολή απαιτεί την ύπαρξη δύο content. Με την εντολή αυτή δηλώνουμε μέχρι πόσα bytes επιτρέπουμε να παρεμβάλλονται ανάμεσα σε δύο content.
5. **nocase:** η εντολή αυτή υποδηλώνει ότι η αναζήτηση του περιεχομένου θα γίνει χωρίς διαφοροποίηση ανάμεσα σε κεφαλαία και μικρά γράμματα (δηλαδή χωρίς case sensitivity).
6. **rawbytes:** η εντολή αυτή επιτρέπει στον κανόνα να εξετάζει τα αρχικά περιεχόμενα του πακέτου, αγνοώντας οποιαδήποτε αποκωδικοποίηση μπορεί να έγινε από τους προεπεξεργαστές.

- **uricontent:** η παράμετρος “uricontent” στη γλώσσα του Snort υποδηλώνει αναζήτηση στο *κανονικοποιημένο* πεδίο αίτησης URI. Αυτό σημαίνει πως, σε περίπτωση που ένας κανόνας περιλαμβάνει στοιχεία που κανονικοποιούνται, όπως %2f ή αλλαγές καταλόγου, ο κανόνας αυτός δε θα ενεργοποιηθεί ποτέ. Αυτό γίνεται επειδή ο κανόνας αναζητά για πράγματα που αφαιρούνται κατά την κανονικοποίηση. Για παράδειγμα, το URI:

  /scripts/..%c0%af../winnt/system32/cmd.exe?/c+ver

  κανονικοποιείται σε:

  /winnt/system32/cmd.exe?/c+ver
- **isdataat:** με την εντολή αυτή προσθέτουμε τον περιορισμό πως θέλουμε να υπάρχουν δεδομένα στη συγκεκριμένη θέση, δηλαδή η θέση αυτή να μη βρίσκεται εκτός ορίων του πακέτου. Μπορεί να χρησιμοποιηθεί ταυτόχρονα με ένα “content” και με την παράμετρο “relative”, στην οποία περίπτωση θέλουμε να υπάρχει περιθώριο N bytes από το τέλος του “content”.
- **pcre:** επιτρέπει τη συγγραφή κανόνων σε μορφή κανονικών εκφράσεων συμβατών με perl.
- **byte_test:** συγκρίνει ένα πεδίο από bytes με μία συγκεκριμένη ακολουθία. Η σύγκριση μπορεί να γίνει με αρκετούς τελεστές (για παράδειγμα <,>,=,!,&).
- **byte_jump:** η εντολή αυτή μας επιτρέπει να εξάγουμε ορισμένα bytes από τα δεδομένα, να τα μετατρέπουμε στον αντίστοιχο αριθμό και να εκτελούμε ένα άλμα κατά τόσα bytes για περαιτέρω εξέταση περιεχομένου ή σύγκριση bytes. Με τον τρόπο αυτό μπορούν να δημιουργηθούν μέθοδοι ανίχνευσης που λαμβάνουν υπόψη τις αριθμητικές τιμές που περιέχονται στα δεδομένα.

2.1.5 Ρυθμίσεις μη-περιεχομένου.

- **fragoffset:** επιτρέπει τη σύγκριση του πεδίου fragment offset της επικεφαλίδας IP με μια καθορισμένη τιμή. Για παράδειγμα, αν η τιμή fragoffset είναι ίση με 0 τότε ο κανόνας ενεργοποιείται στα πακέτα που είναι αδιάσπαστα ή είναι το πρώτο κομμάτι ενός μεγαλύτερου πακέτου.

- **ttl:** σύγκριση της τιμής TTL (Time To Live) της επικεφαλίδας IP με μία συγκεκριμένη τιμή ή πεδίο τιμών.
- **tos:** έλεγχος του πεδίου TOS (Type Of Service) του IP με μία τιμή.
- **id:** έλεγχος του πεδίου id της IP επικεφαλίδας για μια συγκεκριμένη τιμή. Ορισμένα εργαλεία θέτουν συγκεκριμένες τιμές σε αυτό το πεδίο για διάφορους σκοπούς.
- **ipopts:** ελέγχει αν υπάρχουν στο πακέτο ορισμένες επιλογές (options) του πρωτοκόλλου IP. Ο αριθμός και το είδος των options για τις οποίες γίνεται έλεγχος καθορίζεται από το χρήστη.
- **Fragbits:** εξετάζει αν τα bits του κατακερματισμού και το δεσμευμένο bit είναι ενεργοποιημένα στην επικεφαλίδα IP.
- **dsize:** εξετάζει αν τα δεδομένα του πακέτο έχουν μέγεθος που κυμαίνεται μέσα σε καθορισμένα όρια. Μπορεί να χρησιμοποιηθεί για τον έλεγχο ασυνήθιστα μεγάλων πακέτων και για την ανίχνευση υπερχείλισης buffer.
- **flags:** ελέγχει τις τιμές που έχουν συγκεκριμένα flags της επικεφαλίδας TCP. Τα flags που εξετάζονται και οι επιθυμητές τιμές καθορίζονται από τον χρήστη.
- **flow:** χρησιμοποιείται παράλληλα με τον προεπεξεργαστή επανασύνθεσης ροών TCP (TCP stream reassembly). Επιτρέπει την εφαρμογή του κανόνα μόνο σε συγκεκριμένη κατεύθυνση της ροής, για παράδειγμα στη ροή προς τον server.
- **seq:** εξετάζει αν ο αριθμός ακολουθίας (sequence) στην επικεφαλίδα TCP έχει μια συγκεκριμένη τιμή.
- **ack:** ελέγχει μήπως ο αριθμός επιβεβαίωσης (acknowledgement) στην επικεφαλίδα TCP έχει μια συγκεκριμένη τιμή.
- **window:** εξετάζει το μέγεθος του παραθύρου TCP.
- **itype:** ελέγχει για ένα συγκεκριμένο τύπο (type) ICMP.
- **icode:** συγκρίνει τον κωδικό (code) ICMP με μία τιμή ή εξετάζει αν βρίσκεται μέσα σε συγκεκριμένα όρια.
- **icmp_id:** ελέγχει για συγκεκριμένη τιμή του ICMP id. Αποτελεί χρήσιμο εργαλείο καθώς ορισμένα προγράμματα κρυφών καναλιών χρησιμοποιούν συγκεκριμένες τιμές των πεδίων ICMP.
- **icmp_seq:** με την εντολή αυτή ελέγχουμε την τιμή ακολουθίας του ICMP.

- **rpc:** η εντολή αυτή χρησιμοποιείται για την ανίχνευση των αριθμών της εφαρμογής (application), της έκδοσης (version) και της διαδικασίας (procedure) σε αιτήσεις SUNRPC CALL. Πρέπει να σημειωθεί πως η εντολή RPC είναι πιο γρήγορη από ότι αν ψάχναμε για τις συγκεκριμένες τιμές χρησιμοποιώντας εντολές content.
- **ip_proto:** αυτή η εντολή επιτρέπει τη σύγκριση του αριθμού του IP με ορισμένες τιμές.
- **same_ip:** εξετάζει αν η διεύθυνση αποστολέα και παραλήπτη είναι ίδιες.

2.1.6 Ρυθμίσεις ολοκληρωμένης αναζήτησης.

- **logto:** η χρήση της εντολής σε έναν κανόνα καθορίζει το αρχείο μέσα στο οποίο θα αποθηκεύονται όσα πακέτα επαληθευτούν.
- **session:** χρησιμοποιείται για να εξάγει δεδομένα από επικοινωνίες (sessions) TCP. Υπάρχουν περιπτώσεις όπου είναι επιθυμητό να δούμε τι στάλθηκε από τους χρήστες κατά τη διάρκεια μίας επικοινωνίας telnet ή ftp για παράδειγμα. Η χρήση της εντολής "session" μπορεί να επιβραδύνει σημαντικά το Snort, επομένως είναι καλύτερα να αποφεύγεται η χρήση της σε περιπτώσεις έντονης κίνησης.
- **resp:** με την εντολή αυτή επιχειρείται το κλείσιμο μίας επικοινωνίας (session) σε περίπτωση που επαληθευτεί ο κανόνας. Στο Snort, η τακτική αυτή καλείται Ευέλικτη Απάντηση (Flexible Response). Η λειτουργία του μηχανισμού ευέλικτης απάντησης δεν είναι προκαθορισμένα ενεργή, χρειάζεται να ενεργοποιηθεί από τον χρήστη. Πάντως χρειάζεται προσοχή στη χρήση της εντολής αυτής καθώς είναι πιθανό να δημιουργηθεί άπειρος βρόχος.
- **react:** και αυτή η εντολή στηρίζεται στο μηχανισμό ευέλικτης απάντησης και παρέχει επιπλέον τρόπους διαχείρισης των ροών που ενεργοποιούν τον αντίστοιχο κανόνα. Παρέχει τη δυνατότητα κλεισίματος της ροής που ενεργοποίησε τον κανόνα καθώς και την δυνατότητα εμφάνισης κατάλληλου προειδοποιητικού μηνύματος στο χρήστη. Όπως με τη "resp", έτσι και με τη "react" υπάρχει ο κίνδυνος δημιουργίας άπειρου βρόχου.

- **tag:** επιτρέπει την καταγραφή περισσότερων πακέτων εκτός από το ένα που επαληθεύει τον κανόνα. Μόλις ο κανόνας που περιέχει την εντολή "tag" ενεργοποιηθεί, η επιπλέον κίνηση του δικτύου που περιλαμβάνει τον ίδιο αποστολέα ή/και τον παραλήπτη καταγράφεται, ώστε να είναι δυνατή η μετέπειτα ανάλυση της κίνησης που παρουσιάστηκε αμέσως μετά την επίθεση.

2.2 Ο Βελτιστοποιητής Κανόνων.

Ο Βελτιστοποιητής Κανόνων (Rule Optimizer) αποτελεί σημαντικό κομμάτι του Snort από την έκδοση 2.0 και μετά. Σκοπός της λειτουργίας του Βελτιστοποιητή είναι να επιταχύνει τη λειτουργία του Snort και αυτό το επιτυγχάνει διαχωρίζοντας τους κανόνες σε ξεχωριστές ομάδες με κατάλληλο τρόπο ώστε κάθε εισερχόμενο πακέτο να συγκρίνεται μονάχα με μία ομάδα από κανόνες και η σύγκριση αυτή να μπορεί να γίνεται με τρόπο παράλληλο.

2.2.1 Η τυπική επεξεργασία κανόνων στο Snort.

Το Snort χρησιμοποιεί τις παραμέτρους των πρωτοκόλλων για να διαχωρίζει τους κανόνες σε ξεχωριστές ομάδες. Στις παλιότερες εκδόσεις του Snort, χρησιμοποιούταν μια καθαρά παραμετρική αναζήτηση για να εξετάζεται κατά πόσο ένα πακέτο ικανοποιεί τις παραμέτρους μίας ομάδας κανόνων.

Οι κανόνες κατατάσσονταν σε διαφορετικές ομάδες με βάση τέσσερα γενικά χαρακτηριστικά, τέσσερις βασικές παραμέτρους:

1. Η διεύθυνση IP του αποστολέα.
2. Η διεύθυνση IP του παραλήπτη.
3. Οι πιθανές τιμές για τη θύρα του αποστολέα.
4. Οι πιθανές τιμές για τη θύρα του παραλήπτη.

Όταν εξεταζόταν ένα πακέτο, γινόταν έλεγχος του κατά πόσο ταιριάζει με αυτές τις τέσσερις παραμέτρους κάθε ομάδας κανόνων. Εάν το πακέτο δεν ταίριαζε και στις παραμέτρους μίας ομάδας τότε ο έλεγχος προχωρούσε στην επόμενη ομάδα κανόνων. Σε περίπτωση που το πακέτο πράγματι ικανοποιούσε όλες τις παραμέτρους τότε όλοι οι κανόνες μέσα στην ομάδα εξετάζονταν διαδοχικά και οι υπόλοιπες παράμετροι κάθε κανόνα εξετάζονταν κατά σειρά. Μόλις ολοκληρωνόταν η αναζήτηση όλων των κανόνων σε μία ομάδα, οι τέσσερις βασικές παράμετροι του πακέτου εξετάζονταν με αυτούς της επόμενης ομάδας κανόνων και η διαδικασία επαναλαμβανόταν.

Όπως γίνεται εμφανές, η διαδικασία αυτή είναι αποδοτική μόνο σε περίπτωση που έχουμε ένα μικρό αριθμό κανόνων. Εάν, για παράδειγμα, υπάρχουν περισσότεροι από 10 κανόνες σε κάθε ομάδα τότε η μέθοδος αυτή είναι αρκετά χρονοβόρα, καθώς οι κανόνες εξετάζονται διαδοχικά και όλες οι παράμετροι κάθε κανόνα πρέπει να εξεταστούν ανεξάρτητα. Ένας άλλος λόγος για τον οποίο η παλαιότερη μέθοδος επεξεργασίας των κανόνων δεν ήταν αποδοτική είναι ότι κάθε πακέτο μπορούσε να ικανοποιεί τις παραμέτρους περισσότερων από μίας ομάδας κανόνων, γεγονός που σήμαινε πως έπρεπε να εξεταστούν όλοι οι κανόνες αυτών των ομάδων ξεχωριστά.

Προκειμένου να βελτιωθεί η ταχύτητα επεξεργασίας των κανόνων, ήταν απαραίτητη η υλοποίηση μεθόδων ομαδικής επεξεργασίας. Το πρόβλημα ήταν οτι η παραδοσιακή επεξεργασία κανόνων στο Snort δεν προσφερόταν για εφαρμογή ομαδικής επεξεργασίας επειδή σε κάθε εισερχόμενο πακέτο αντιστοιχούσαν περισσότερες από μία ομάδες κανόνων. Για να είναι δυνατή η αποδοτική εφαρμογή μεθόδων ομαδικής επεξεργασίας πρέπει για κάθε εισερχόμενο πακέτο να χρειάζεται να εξετάσουμε τους κανόνες που περιέχονται σε μία μόνο ομάδα. Η δημιουργία και επιλογή των ομάδων από κανόνες είναι η αποκλειστική λειτουργία του Βελτιστοποιητή Κανόνων.

2.2.2 Η δημιουργία των ομάδων από κανόνες.

Ο Βελτιστοποιητής Κανόνων δημιουργεί ομάδες από κανόνες με μορφή κατάλληλη ώστε να είναι εφικτή η αποδοτική χρήση μεθόδων ομαδικής επεξεργασίας.

Για να επιτευχθεί ο σκοπός αυτός, χρειάζεται ο Βελτιστοποιητής να πληρεί δύο προϋποθέσεις:

1. Να δημιουργεί όσο το δυνατόν πιο μικρές και αποδοτικές ομάδες κανόνων.
2. Να δημιουργεί διακριτές ομάδες με τέτοιο τρόπο ώστε για κάθε εισερχόμενο πακέτο να απαιτείται η εξέταση μίας και μόνο ομάδας κανόνων.

Ο Βελτιστοποιητής δημιουργεί τις ομάδες αυτές κατά την εκκίνηση του Snort χρησιμοποιώντας τις πιο μοναδικές παραμέτρους των κανόνων. Οι παράμετροι των κανόνων που επιλέγονται είναι διαφορετικές για κάθε πρωτόκολλο μεταφοράς, επειδή κάθε πρωτόκολλο έχει διαφορετικές παραμέτρους που το κάνουν μοναδικό. Για παράδειγμα, ένας κανόνας TCP μπορεί να ξεχωρίζει από άλλους κανόνες TCP με βάση τις θύρες αποστολέα και παραλήπτη, ενώ ένας κανόνας ICMP μπορεί να είναι μοναδικός λόγω του τύπου ICMP του συγκεκριμένου κανόνα. Ο Βελτιστοποιητής εξετάζει όλες τις μοναδικές παραμέτρους και τις χρησιμοποιεί για τη δημιουργία υποομάδων από κανόνες. Έτσι, προκύπτουν μικρότερες ομάδες κανόνων και, το σημαντικότερο, κάθε εισερχόμενο πακέτο μπορεί να κατηγοριοποιηθεί σε μια υποομάδα.

2.2.3 Η επιλογή μίας ομάδας.

Κατά τη διάρκεια της λειτουργίας του Snort, ο Βελτιστοποιητής κανόνων επιλέγει μία ομάδα κανόνων για κάθε πακέτο που επεξεργάζεται. Σε ορισμένες περιπτώσεις ασυνήθιστων πακέτων υπάρχει η πιθανότητα επιλογής δύο ομάδων. Το φαινόμενο αυτό καλείται «μοναδική σύγκρουση» (unique conflict) και εξετάζεται λεπτομερώς αργότερα.

Πρέπει να σημειωθεί ότι ο Βελτιστοποιητής είναι το πρώτο στάδιο της μηχανής πολλαπλής αναζήτησης κανόνων, καθώς η επιλογή της ομάδας κανόνων εξαρτάται από την ισότητα ορισμένων παραμέτρων του πακέτου με τις παραμέτρους των κανόνων. Υπό αυτή την έννοια, ο Βελτιστοποιητής επιλέγει μόνο τους κανόνες οι οποίοι θα μπορούσαν να επαληθεύονται στο εισερχόμενο πακέτο και, κατά συνέπεια, χρειάζεται να εξεταστούν λεπτομερώς. Στη συνέχεια η μηχανή πολλαπλής αναζήτησης εκτελεί λεπτομερή έλεγχο

των παραμέτρων περιεχομένου (content) που περιλαμβάνουν οι κανόνες χρησιμοποιώντας μεθόδους ομαδικής επεξεργασίας.

2.2.4 Ο Βελτιστοποιητής κανόνων στην πράξη.

Ο Βελτιστοποιητής διαχωρίζει τους κανόνες σε διακριτές και όσο το δυνατόν μικρότερες ομάδες στις οποίες το Snort μπορεί να έχει γρήγορη πρόσβαση. Αυτό επιτυγχάνεται με τη χρήση των ακόλουθων μοναδικών παραμέτρων, οι οποίες εξαρτώνται από το πρωτόκολλο μεταφοράς.

- **TCP/UDP**: θύρα αποστολέα, θύρα παραλήπτη.
- **ICMP**: τύπος ICMP.
- **IP**: αναγνωριστικό πρωτοκόλλου μεταφοράς.

Ας δούμε πιο συγκεκριμένα τη συμπεριφορά σε κάθε περίπτωση.

TCP/UDP

Οι πλέον χαρακτηριστικές παράμετροι των πρωτοκόλλων TCP και UDP είναι οι θύρες αποστολέα και παραλήπτη. Οι θύρες διαχωρίζονται σε δύο γενικές κατηγορίες: τις δεσμευμένες και τις μη-δεσμευμένες. Αυτό σημαίνει πως οι περισσότερες επικοινωνίες ανάμεσα σε hosts θα χρησιμοποιούν μία δεσμευμένη θύρα (συνήθως του server) ενώ η άλλη θύρα θα είναι μη δεσμευμένη (μεγαλύτερη από 1024 και συνήθως του client). Η ιδιότητα αυτή επιτρέπει στον Βελτιστοποιητή να χρησιμοποιεί τη δεσμευμένη θύρα σαν μοναδική παράμετρο. Με τη χρήση της δεσμευμένης θύρας, το Snort επιλέγει τις ομάδες κανόνων σύμφωνα με το αν η δεσμευμένη θύρα βρίσκεται από τη μεριά του αποστολέα ή του παραλήπτη του πακέτου. Εάν η δεσμευμένη θύρα βρίσκεται στη μεριά του αποστολέα τότε, συνήθως, η συγκεκριμένη κίνηση δεδομένων γίνεται από τον server προς τον client. Σε περίπτωση που η δεσμευμένη θύρα είναι από τη μεριά του παραλήπτη, το πιθανότερο είναι πως έχουμε κίνηση δεδομένων από τον client.

Παρατηρώντας τα πρωτόκολλα TCP/UDP στην πράξη γίνεται ευκολότερη η κατανόηση αυτού του γεγονότος. Τα πρωτόκολλα TCP και UDP παρέχουν αμφίδρομη

επικοινωνία μεταξύ client και server. Το γεγονός αυτό συνεπάγεται ότι το Snort παρατηρεί πακέτα από τον client να πηγαίνουν στον server και πακέτα από τον server να κατευθύνονται στον client. Έτσι, αντί να εφαρμόζονται οι κανόνες που απευθύνονται και σε επικοινωνία από client αλλά και σε επικοινωνία από server, ο Βελτιστοποιητής διαχωρίζει τους κανόνες με βάση τη θέση της δεσμευμένης θύρας ως θύρας αποστολέα ή θύρας παραλήπτη. Με τον τρόπο αυτό οι κανόνες αίτηση-client και απάντηση-server κατατάσσονται σε διακριτές ομάδες. Έτσι αποφεύγεται η περίσσεια του να εξετάζονται τα πακέτα από τον client με τους κανόνες που αντιστοιχούν σε απαντήσεις-server και αντίστροφα.

Για παράδειγμα, η αίτηση ενός HTTP client πρέπει να εξεταστεί με το σύνολο των κανόνων για αιτήσεις client και όχι με την ομάδα κανόνων για απαντήσεις server. Επομένως, όταν ένα πακέτο έχει σταλεί από τον HTTP client (δηλαδή η θύρα του παραλήπτη είναι η 80), τόσο η θύρα του αποστολέα όσο και εκείνη του παραλήπτη εξετάζονται για την ύπαρξη δεσμευμένης θύρας. Σχεδόν σε όλες τις περιπτώσεις που έχουμε κίνηση από HTTP client, η θύρα του αποστολέα δεν είναι δεσμευμένη θύρα (δηλαδή έχει τιμή μεγαλύτερη από 1024), όμως θα είναι δεσμευμένη η θύρα του παραλήπτη επειδή το πακέτο προορίζεται σε έναν server HTTP, ο οποίος συνήθως βρίσκεται στη θύρα 80 ή σε μία άλλη καθορισμένη θύρα για HTTP servers. Άρα, στην περίπτωση του πακέτου από έναν HTTP client θα επιλεγεί η ομάδα κανόνων που έχει την παράμετρο της θύρας παραλήπτη ίση με 80 και της θύρας αποστολέα ως οποιαδήποτε και το πακέτο θα εξεταστεί για την ύπαρξη των κανόνων που συμπεριλαμβάνονται στην ομάδα αυτή.

ICMP

Για τη βελτιστοποίηση των κανόνων ICMP χρησιμοποιείται μόνο το είδος του ICMP. Οι κανόνες που δεν ορίζουν κάποια συγκεκριμένη τιμή για το είδος θεωρούνται γενικοί κανόνες ICMP και προστίθενται σε κάθε ξεχωριστή ομάδα από ICMP κανόνες, συμπεριλαμβανομένης και της γενικής ομάδας. Για παράδειγμα, ένα εισερχόμενο πακέτο ICMP με κωδικό είδους 8 (echo request) θα επιθεωρηθεί με την ομάδα κανόνων που έχει τιμή για το είδος ίση με 8. Σε περίπτωση που δεν αντιστοιχεί κάποια ομάδα κανόνων στον κωδικό είδους του πακέτου, η εξέταση γίνεται με τη γενική ομάδα κανόνων ICMP.

IP

Ο Βελτιστοποιητής Κανόνων υποστηρίζει το πρωτόκολλο IP χρησιμοποιώντας το πεδίο του πρωτοκόλλου μεταφοράς, που βρίσκεται στην επικεφαλίδα IP ενός πακέτου, σαν παράμετρο διαχωρισμού των κανόνων. Όλοι οι κανόνες IP που περιλαμβάνουν TCP, UDP ή ICMP συμπεριλαμβάνονται στην ομάδα κανόνων του αντίστοιχου πρωτοκόλλου. Όμως άλλα πρωτόκολλα που δεν υποστηρίζονται κατηγοριοποιούνται στην ομάδα IP. Στην πράξη, ο Βελτιστοποιητής χρησιμοποιεί τις πληροφορίες του πρωτοκόλλου μεταφοράς για τη δημιουργία και την επιλογή των ομάδων από κανόνες. Προκειμένου ο Βελτιστοποιητής να επιλέγει μόνο μία ομάδα κανόνων, είναι αναγκαίος ο συνδυασμός των κανόνων που βρίσκονται αποκλειστικά σε μια ομάδα κανόνων με τους γενικούς κανόνες που αντιστοιχούν στο συγκεκριμένο πρωτόκολλο μεταφοράς.

2.2.5 Μοναδικοί κανόνες και Γενικοί κανόνες.

Οι μοναδικοί κανόνες καθορίζονται από την ύπαρξη μοναδικών παραμέτρων. Αυτές οι μοναδικές παράμετροι μπορεί να είναι συγκεκριμένες πληροφορίες στην επικεφαλίδα του πρωτοκόλλου μεταφοράς ή οι διευθύνσεις IP στις οποίες απευθύνεται ο κανόνας. Οι γενικοί κανόνες δεν έχουν καμία μοναδική παράμετρο ώστε να διαχωρίζονται από άλλους κανόνες. Για παράδειγμα, ένας κανόνας μπορεί να γραφτεί ο οποίος θα λέει «εξέτασε όλα τα πακέτα αν έχουν αυτό το περιεχόμενο». Στην περίπτωση αυτή, οι παράμετροι της επικεφαλίδας του κανόνα δεν είναι μοναδικές καθώς δεν υπάρχει τίποτα εκτός από το περιεχόμενο το οποίο να διαχωρίζει αυτόν από οποιονδήποτε άλλο κανόνα.

Τόσο οι μοναδικοί όσο και οι γενικοί κανόνες πρέπει να έχουν διαφορετικές ομάδες κανόνων. Χρησιμοποιώντας τις μοναδικές ομάδες κανόνων μαζί με τη γενική ομάδα καλύπτονται όλες οι πιθανότητες επιλογής. Η δημιουργία της μοναδικής ομάδας καλύπτει την περίπτωση που ένα πακέτο ταιριάζει με κάποιον μοναδικό κανόνα. Αντίστοιχα, η δημιουργία της γενικής ομάδας καλύπτει την περίπτωση που το πακέτο δεν ταιριάζει με καμία μοναδική ομάδα, καθώς το πακέτο θα πρέπει να εξεταστεί ως προς το πλήθος των γενικών κανόνων για το συγκεκριμένο πρωτόκολλο.

Προκειμένου να λειτουργεί αυτός ο διαχωρισμός ανάμεσα σε μοναδικούς και γενικούς κανόνες, οι γενικοί κανόνες πρέπει να εισάγονται και μέσα σε κάθε μοναδική ομάδα. Διαφορετικά θα έπρεπε να εξετάζονται δύο ομάδες κανόνων σε κάθε πακέτο, η μοναδική ομάδα και η γενική ομάδα, καθώς στην περίπτωση που ένα πακέτο ταιριάζει σε έναν ειδικό κανόνα είναι πιθανό να ταιριάζει και σε ένα γενικό κανόνα.

Με τον τρόπο αυτό ο Βελτιστοποιητής Κανόνων επιλέγει είτε μία μοναδική είτε τη γενική ομάδα κανόνων, ανάλογα με το πακέτο που εξετάζεται. Τι γίνεται στην περίπτωση που ένα πακέτο ταιριάζει με περισσότερες από μια μοναδικές ομάδες? Τότε λέμε ότι έχουμε μία μοναδική σύγκρουση.

2.2.6 Μοναδικές συγκρούσεις.

Οι μοναδικές συγκρούσεις παρουσιάζονται όταν υπάρχουν δύο ή περισσότερες παράμετροι με τις οποίες γίνεται η επιλογή της ομάδας κανόνων. Σε περίπτωση που και οι δύο παράμετροι έχουν δεσμευμένη τιμή τότε συμβαίνει μία μοναδική σύγκρουση. Το φαινόμενο αυτό εμφανίζεται μόνο για τα πρωτόκολλα TCP και UDP.

Υπάρχουν δύο είδη μοναδικής σύγκρουσης: οι καθορισμένες συγκρούσεις και οι μη καθορισμένες. Οι καθορισμένες συγκρούσεις αντιμετωπίζονται με το συνδυασμό τον δύο συγκρουόμενων ομάδων κανόνων σε μία καινούργια και την εξέταση του πακέτου με την νέα ομάδα. Οι μη καθορισμένες συγκρούσεις μπορούν να αντιμετωπιστούν με διάφορους τρόπους, ενώ κριτήριο για την επιλογή αποτελεί η επιθυμητή απόδοση από πλευράς του χρήστη.

Πρέπει να σημειωθεί πως δεν είναι εφικτό να μετατραπούν όλες οι μοναδικές συγκρούσεις σε καθορισμένες καθώς αυξάνει υπερβολικά ο χώρος κατάστασης του συστήματος. Για παράδειγμα, αν υπάρχουν 1000 μοναδικοί κανόνες για τη θύρα αποστολέα και ακόμα 1000 μοναδικοί κανόνες για τη θύρα παραλήπτη θα χρειαζόμασταν 1,000,000 επιπλέον ομάδες κανόνων για κάθε πιθανή σύγκρουση.

Ενώ οι καθορισμένες μοναδικές συγκρούσεις δεν είναι σπάνιες, οι μη καθορισμένες εμφανίζονται αρκετά σπάνια και όποτε εμφανίζονται είναι συνήθως άξιες προσοχής.

2.2.7 Καθορισμένες και μη καθορισμένες συγκρούσεις.

Ένα παράδειγμα καθορισμένης σύγκρουσης γίνεται εμφανές σε αιτήσεις DNS μεταξύ δύο servers. Κατά τη διάρκεια αυτής της επικοινωνίας και οι δύο θύρες θα έχουν την τιμή 53, οπότε προκύπτει το πρόβλημα του ποια ομάδα κανόνων πρέπει να χρησιμοποιηθεί. Η απάντηση είναι και οι δύο ομάδες, επομένως συνδυάζονται σε μία τρίτη ομάδα, με την οποία και εξετάζεται σε ένα πέρασμα το πακέτο.

Οι μη καθορισμένες συγκρούσεις είναι αρκετά σπάνιες και συνήθως σχετίζονται με κάποιο ασυνήθιστο ή εχθρικό γεγονός. Παρά την σπανιότητα αυτού του είδους συγκρούσεων χρειάζεται ένας μηχανισμός επιλογής, καθώς σε κάθε μη καθορισμένη σύγκρουση αντιστοιχούν δύο ομάδες κανόνων με τις οποίες πρέπει να γίνει εξέταση του πακέτου.

Υπάρχουν τρεις εναλλακτικές σχετικά με τον τρόπο που θα αντιμετωπιστεί μια μη καθορισμένη σύγκρουση: η διπλή επιθεώρηση, η επιθεώρηση πρώτης απάντησης και η στοχαστική επιθεώρηση.

1. *Διπλή επιθεώρηση*: ο πιο βασικός τρόπος αντιμετώπισης της σύγκρουσης. Από τη στιγμή που αντιστοιχούν δύο ομάδες κανόνων στο εισερχόμενο πακέτο, τότε η εξέταση γίνεται και με τις δύο. Με τον τρόπο αυτό είναι σίγουρο ότι θα βρεθούν όλα τα ύποπτα περιεχόμενα του πακέτου, όμως πρακτικά διπλασιάζεται και ο χρόνος επεξεργασίας για το πακέτο. Η αύξηση του χρόνου επεξεργασίας καθιστά αυτή τη μέθοδο ευάλωτη σε «επιθέσεις στέρησης εξυπηρέτησης» (Denial Of Service attacks – DOS) μέσω κατασκευασμένων πακέτων που θα δημιουργούν μη καθορισμένες συγκρούσεις.
2. *Επιθεώρηση πρώτης απάντησης*: Είναι σχεδόν ίδια με την διπλή επιθεώρηση. Η μόνη διαφορά είναι ότι στην περίπτωση που η εξέταση του πακέτου με την πρώτη ομάδα μας δώσει θετική απάντηση (δηλαδή πως επαληθεύτηκαν κάποιοι κανόνες) τότε δεν εξετάζουμε το πακέτο και με τη δεύτερη ομάδα. Η μέθοδος αυτή προσφέρει καλύτερη απόδοση από τη μέθοδο διπλής εξέτασης αν η εξέταση σταματήσει με την πρώτη ομάδα, προστατεύοντας σε κάποιο βαθμό από επιθέσεις DOS. Το μειονέκτημα είναι ότι δεν παρέχει την εγγυημένη προστασία

από επιθέσεις DOS που παρέχει η στοχαστική επιθεώρηση, ενώ επίσης δεν εγγυάται πως θα βρεθούν όλα τα ύποπτα περιεχόμενα του πακέτου.

3. *Στοχαστική επιθεώρηση*: Στη μέθοδο αυτή επιλέγεται τυχαία μια από τις δύο υποψήφιες ομάδες. Με τη μέθοδο αυτό προστατευόμαστε από επιθέσεις DOS που εκμεταλλεύονται μη καθορισμένες συγκρούσεις. Υπάρχει, όμως, η πιθανότητα να επιτραπεί η διέλευση σε πακέτα με επικίνδυνο περιεχόμενο σε περίπτωση που επιλεγεί η λάθος ομάδα κανόνων.

2.3 Η μηχανή πολλαπλής αναζήτησης.

Στην καρδιά του Snort 2.0 βρίσκεται η Μηχανή Επιθεώρησης Πολλαπλών Κανόνων (High Performance Multi-Rule Inspection Engine), η οποία επωμίζεται το φορτίο της αναζήτησης ύποπτου περιεχόμενου που μπορεί να βρίσκεται μέσα στα εισερχόμενα πακέτα.

Η διαδικασία της επιθεώρησης της κίνησης δικτύου για τυχόν επαληθεύσεις των κανόνων εκτελείται σε τρία βήματα:

1. Βελτιστοποίηση των κανόνων για την παραγωγή αποδοτικών ομάδων.
2. Χρήση αλγορίθμων πολλαπλής αναζήτησης για γρήγορη αναζήτηση των ζητούμενων περιεχομένων.
3. Παραμετρικές τεχνικές αναζήτησης που επιτρέπουν πολύπλοκες επιθεωρήσεις παραμέτρων.

Στην πράξη τα εισερχόμενα πακέτα εξετάζονται πρώτα από τον Βελτιστοποιητή κανόνων ώστε να επιλεγεί η κατάλληλη ομάδα κανόνων με την οποία θα εξεταστεί κάθε πακέτο. Στη συνέχεια η μηχανή πολλαπλής αναζήτησης ανιχνεύει το πακέτο για περιεχόμενα που ταιριάζουν με τους κανόνες που επιλέχθηκαν, κατασκευάζει μία λίστα από τους κανόνες που επαληθεύτηκαν και επιλέγει τον καλύτερο κανόνα για καταγραφή χρησιμοποιώντας ένα απλό σύνολο προτεραιοτήτων.

Η βελτιστοποίηση κανόνων και η πολλαπλή αναζήτηση παρέχουν την υψηλή απόδοση που απαιτείται στα σύγχρονα δίκτυα υψηλών ταχυτήτων. Η χρήση της παραδοσιακής μεθόδου του Snort για παραμετρική αναζήτηση επιτρέπει την επέκταση της γλώσσας κανόνων χωρίς να επηρεάζεται η υψηλή απόδοση της μηχανής επιθεώρησης. Ο κατάλληλος συνδυασμός των μεθόδων αυτών επιτρέπει την εφαρμογή της κάθε μεθόδου στον τομέα που αποδίδει καλύτερα, ενώ επιτρέπει στη γλώσσα του Snort να παραμείνει ευέλικτη για μελλοντικές εξελίξεις.

Η μηχανή πολλαπλής αναζήτησης του Snort μπορεί να επιθεωρεί δίκτυα με ταχύτητες Gigabit, ανιχνεύοντας και καταγράφοντας γεγονότα από πολύ μεγάλες ομάδες κανόνων χωρίς να υπάρχουν απώλειες πακέτων.

2.3.1 Παραμετρική επιθεώρηση κανόνων.

Σε μία παραμετρική αναζήτηση γίνεται ακολουθιακή εξέταση κάθε παραμέτρου. Σε περίπτωση που μία παράμετρος δεν επαληθευτεί η αναζήτηση αποτυγχάνει. Σε παλιότερες εκδώσεις του Snort η επαλήθευση των κανόνων γινόταν με την εκτέλεση παραμετρικής αναζήτησης, όπου οι παράμετροι ήταν οι ρυθμίσεις (options) του συγκεκριμένου κανόνα. Η πολυπλοκότητα και η απόδοση ενός τέτοιου τύπου επιθεώρησης είναι γενικώς συνάρτηση του αριθμού των κανόνων, του αριθμού των παραμέτρων σε κάθε κανόνα και του συνδυασμένου χώρου κατάστασης όλων των παραμέτρων.

Σε μία τυπική παραμετρική αναζήτηση, η σειρά με την οποία προσπαθούμε να επαληθεύσουμε τις διαφορετικές παραμέτρους εξαρτάται από τη δυνατότητα της κάθε παραμέτρου να τερματίσει γρήγορα μία άσκοπη αναζήτηση ή να περιορίσει τον χώρο κατάστασης. Η πραγματική σειρά εξέτασης των παραμέτρων εξαρτάται από την βαθύτερη κατανόηση του προβλήματος. Συχνά υπάρχουν μία ή περισσότερες σειρές εξέτασης των παραμέτρων που παρέχουν μία προφανή σχεδόν βέλτιστη αναζήτηση.

Οι κανόνες του Snort ομαδοποιούνται χρησιμοποιώντας ορισμένες κύριες παραμέτρους, όπως είναι οι διευθύνσεις IP και οι θύρες του πρωτοκόλλου. Στις παλαιότερες εκδόσεις, σε περίπτωση που ένα πακέτο ικανοποιούσε τις κύριες

παραμέτρους μίας ομάδας τότε εκτελούταν διαδοχικές παραμετρικές αναζητήσεις για κάθε κανόνα μέσα στην ομάδα. Η παραμετρική αναζήτηση έχει το προσόν ότι επιτρέπει την ύπαρξη μίας ευέλικτης γλώσσας για τη συγγραφή κανόνων. Δυστυχώς, όμως, υπάρχει το πρόβλημα ότι το κόστος της παραμετρικής αναζήτησης είναι ευθέως ανάλογο με τον αριθμό των κανόνων, πρόβλημα σημαντικό όταν χρησιμοποιείται μεγάλος αριθμός κανόνων.

2.3.2 Επιθεώρηση κανόνων με πολλαπλή αναζήτηση.

Μία μέθοδος πολλαπλής αναζήτησης (set based search) διαφέρει από μια παραμετρική αναζήτηση κατά τον εξής τρόπο: μια παραμετρική αναζήτηση εξετάζει ένα κανόνα τη φορά και εξετάζει τις παραμέτρους του συγκεκριμένου κανόνα, μια μέθοδος πολλαπλής αναζήτησης επεξεργάζεται μία ή περισσότερες παραμέτρους τη φορά και την εξετάζει για όλους τους κανόνες παράλληλα. Έτσι προκύπτει πως μια αποτελεσματική μέθοδος πολλαπλής αναζήτησης μπορεί να είναι πολύ πιο αποδοτική από την παραμετρική αναζήτηση.

Το Snort χρησιμοποιεί τις επιλογές content και uricontent των κανόνων για να εκτελέσει μια πολλαπλή αναζήτηση μίας παραμέτρου χρησιμοποιώντας αλγόριθμους πολλαπλής αναζήτησης (multi-pattern search). Οι παράμετροι αυτοί είναι πολύ αποτελεσματικές όταν ενδιαφερόμαστε να απορρίψουμε πακέτα και ροές από περαιτέρω εξέταση. Επιπλέον, επωφελούνται σημαντικά από μεθόδους παράλληλη εξέταση υψηλής απόδοσης.

2.3.3 Αναζήτηση στο Snort και κατηγορίες αναζητήσεων.

Καθώς οι μέθοδοι πολλαπλής αναζήτησης προσφέρουν μεγάλη απόδοση, οι τρέχουσες εκδόσεις του Snort χρησιμοποιούν αρχικά πολλαπλή αναζήτηση μίας παραμέτρου, της επιλογής content ή της uricontent του κανόνα. Σε περίπτωση που η πολλαπλή αναζήτηση μας επιστρέψει κάποιον κανόνα για τον οποίο επαληθεύτηκε η

παράμετρος, χρειάζεται να εξεταστούν και οι υπόλοιπες παράμετροι του κανόνα. Εφόσον τώρα εξετάζεται μόνο ένας κανόνας, χρησιμοποιείται μια τυπική παραμετρική αναζήτηση, όπως γινόταν στις παλαιότερες εκδόσεις του Snort.

Υπάρχουν τέσσερις κατηγορίες κανόνων που ίσως να χρειάζεται εξεταστούν σε ένα πακέτο. Κάθε μία από αυτές τις τέσσερις κατηγορίες υποδιαιρείται σε δύο διαφορετικές περιπτώσεις αναζήτησης: αναζήτηση περιεχομένου (content search) και αναζήτηση χωρίς περιεχόμενο (no-content search). Οι τέσσερις κατηγορίες κανόνων είναι οι παρακάτω:

1. *Κανόνες πεδίων πρωτοκόλλου*: Η αναζήτηση σε πεδία πρωτοκόλλου επιτρέπει σε έναν κανόνα να προσδιορίζει ένα συγκεκριμένο πεδίο μέσα σε ένα πρωτόκολλο στο οποίο ψάχνουμε για ύποπτο περιεχόμενο. Για παράδειγμα, το Snort χρησιμοποιεί την επιλογή 'uricontent' για την εκτέλεση αναζήτησης μέσα στα πεδία URI μίας αίτησης HTTP.
2. *Κανόνες γενικού περιεχομένου*: Οι κανόνες γενικού περιεχομένου επιτρέπουν σε ένα κανόνα να εκτελεί μια γενική αναζήτηση μέσα σε ολόκληρα τα δεδομένα του πακέτου για μία συγκεκριμένη συμβολοσειρά ή ακολουθία από bytes. Για παράδειγμα, η δυνατότητα αυτή μπορεί να χρησιμοποιηθεί για την αναζήτηση οποιασδήποτε ASCII ή δυαδικής ακολουθίας η οποία μπορεί να θεωρηθεί ύποπτη, όπως 'root' ή 'passwd'.
3. *Κανόνες ανώμαλων πακέτων*: Η συγκεκριμένη κατηγορία αφορά τους κανόνες που καθορίζουν χαρακτηριστικά του πακέτου ή της επικεφαλίδας του που αποτελούν λόγο ανησυχίας σε περίπτωση που είναι ασυνήθιστα, ανώμαλα. Οι κανόνες ανώμαλων πακέτων δεν χρειάζονται αναζήτηση συγκεκριμένου περιεχομένου, αντίθετα επικεντρώνονται σε άλλα χαρακτηριστικά του πακέτου. Παρόλο που όλες οι κατηγορίες κανόνων μπορούν να περιέχουν και στοιχεία ανίχνευσης κάποιας ανωμαλίας, η συγκεκριμένη κατηγορία περιλαμβάνει μόνο κανόνες που δεν χρειάζονται καθόλου αναζήτηση περιεχομένου. Σαν παράδειγμα μπορούμε να αναφέρουμε έναν κανόνα που θα ενδιαφέρεται μόνο για πακέτα που έρχονται από μία συγκεκριμένη διεύθυνση και θύρα και ίσως έχει και άλλους περιορισμούς για τα πεδία της επικεφαλίδας.

4. *Κανόνες IP*: Οι κανόνες IP εφαρμόζονται στο επίπεδο του πρωτοκόλλου IP και ίσως χρειάζεται να εφαρμόζονται σε κάθε πακέτο IP. Σε περίπτωση που το πρωτόκολλο ενός πακέτου είναι TCP, UDP ή ICMP, οι κανόνες αυτοί εξετάζονται ταυτόχρονα με την αντίστοιχη ομάδα για το συγκεκριμένο πρωτόκολλο επιπέδου εφαρμογών. Οι κανόνες IP μπορούν να έχουν αναζήτηση περιεχομένου ή όχι.

2.3.4 Αναζήτηση περιεχομένου.

Η αναζήτηση περιεχομένου αφορά στη χρήση της Μηχανής Επιθεώρησης Πολλαπλών Κανόνων για την ανεύρεση όλων των στιγμιότυπων κάθε ζητούμενης συμβολοσειράς μέσα σε ένα πακέτο. Η πολλαπλή αναζήτηση επιτυγχάνεται με τη συλλογή ενός αριθμού από ζητούμενες συμβολοσειρές και την εκτέλεση μίας ταυτόχρονης αναζήτησης. Αυτός ο τύπος αναζήτησης μπορεί να επιθεωρήσει έναν όγκο δεδομένων και να βρει όλα τα στιγμιότυπα για κάθε μία από της ζητούμενες συμβολοσειρές. Το Snort δε σταματάει πλέον στην $1^{η}$ συμβολοσειρά που ταιριάζει. Αντίθετα, κάθε στιγμιότυπο που βρίσκεται εισάγεται σε μια λίστα και αργότερα, όταν έχει ολοκληρωθεί ο έλεγχος του πακέτου για τα περιεχόμενα, επεξεργάζεται κατά την επιλογή γεγονότος.

Οι ζητούμενες συμβολοσειρές αντιστοιχούν σε συμβολοσειρές που ορίζονται σε έναν κανόνα με τη χρήση των επιλογών ‘content’ και ‘uricontent’. Η δυνατότητα αυτή μπορεί να προκαλέσει μέχρι και δύο ξεχωριστές πολλαπλές αναζητήσεις, μία για τους κανόνες πεδίων πρωτοκόλλου (HTTP και Telnet) και μια για τους κανόνες γενικού περιεχομένου. Δηλαδή, αυτό συμβαίνει σε περίπτωση που και τα δύο είδη κανόνων εφαρμόζονται στο πακέτο ή την ροή υπό εξέταση.

Η πολλαπλή αναζήτηση εκτελείται σε δύο στάδια. Το πρώτο στάδιο είναι η πολλαπλή αναζήτηση των περιεχομένων. Το στάδιο αυτό βρίσκει όλα τα στιγμιότυπα των περιεχομένων που έχουν οριστεί από κανόνες και αναγνωρίζει σε ποιον κανόνα αντιστοιχεί το κάθε περιεχόμενο, όμως δεν εξετάζει τις υπόλοιπες παραμέτρους που μπορεί να είναι ορισμένες στον κάθε κανόνα. Σε περίπτωση που η επιθεώρηση του

πακέτου επαληθεύσει την ύπαρξη ζητούμενων συμβολοσειρών, εκτελείται το δεύτερο στάδιο της αναζήτησης, δηλαδή μια τυπική παραμετρική αναζήτηση για κάθε κανόνα που επαληθεύτηκε στο πρώτο στάδιο. Το δεύτερο στάδιο είτε επικυρώνει την ύπαρξη ενός κανόνα είτε αποφασίζει πως, παρόλο που βρέθηκε περιεχόμενο που ορίζεται σε κάποιον κανόνα, οι υπόλοιπες παράμετροι του κανόνα δεν επαληθεύτηκαν.

Κατά τη διάρκεια της πρώτης φάσης, η αναζήτηση όλων των περιεχομένων που ορίζονται στους κανόνες της επιλεγμένης ομάδας κανόνων γίνεται εν παραλλήλω, με τη χρήση ενός αλγορίθμου πολλαπλής αναζήτησης. Η μηχανή πολλαπλής αναζήτησης του Snort περιλαμβάνει δύο διαφορετικούς αλγορίθμους πολλαπλής αναζήτησης. Ο πρώτος είναι ένας τροποποιημένος αλγόριθμος τύπου Wu-Manber, ενώ ο δεύτερος είναι ένας αλγόριθμος τύπου Aho-Corasick. Υπάρχει επίσης και ένας απλοϊκός αλγόριθμος Boyer-Moore πολλαπλής αναζήτησης για την επεξεργασία ομάδων με λιγότερους από 10 κανόνες.

2.3.5 Οι αλγόριθμοι πολλαπλής αναζήτησης.

Ο αλγόριθμος Wu-Manber είναι η προκαθορισμένη επιλογή για την εκτέλεση των αναζητήσεων και έχει τις μικρότερες απαιτήσεις χώρου. Χρησιμοποιεί μεθόδους hashing για το διαχωρισμό των κανόνων σε υποομάδες, ενώ διαθέτει έναν πίνακα μετατόπισης «κακού χαρακτήρα» ώστε να επιταχύνεται η αναζήτηση με την υπερπήδηση ορισμένων bytes. Πρόκειται για αρκετά ευέλικτο αλγόριθμο, ο οποίος έχει χρησιμοποιηθεί κατά καιρούς σε διάφορα εργαλεία του UNIX. Λόγω της ύπαρξης του πίνακα μετατόπισης, οι ομάδες κανόνων που έχουν μεγάλη τιμή για το ελάχιστο μήκος (δηλαδή για το μήκος της μικρότερης συμβολοσειράς μέσα στην ομάδα) ωφελούνται σημαντικά στο θέμα απόδοσης. Για κάθε ομάδα κανόνων απαιτούνται περίπου 128K επιπλέον μνήμης για τη διαχείριση της ομάδας.

Παρέχεται, επίσης, η δυνατότητα χρήσης του αλγορίθμου Aho-Corasick σαν εναλλακτικός τρόπος αναζήτησης, ο οποίος χρησιμοποιεί περίπου δύο με τρεις φορές περισσότερη μνήμη από τον Wu-Manber. Πρόκειται για μία τυπική υλοποίηση του αλγορίθμου Aho-Corasick στην οποία δεν χρησιμοποιείται πίνακας μετατοπίσεων τύπου

Boyer-Moore για την υπερπήδηση χαρακτήρων. Ο αλγόριθμος Aho-Corasick είναι λιγότερο ευαίσθητος από τον Wu-Manber ως προς το ελάχιστο μήκος των συμβολοσειρών, παρόλα αυτά η απόδοση βελτιώνεται για μεγάλες τιμές του ελαχίστου μήκους. Οι απαιτήσεις σε μνήμη του Aho-Corasick εξαρτώνται άμεσα από τον αριθμό τον bytes που καταλαμβάνουν όλες οι συμβολοσειρές προς αναζήτηση, ενώ ο αλγόριθμος αυτός έχει χρησιμοποιηθεί και προσαρμοστεί για αναζητήσεις σε βάσεις δεδομένων ή σε αναζητήσεις πολύ μεγάλων συνόλων από συμβολοσειρές.

Γενικά, ο Wu-Manber αποτελεί την προτιμητέα επιλογή για το Snort. Ο αλγόριθμος Aho-Corasick μπορεί να είναι λίγο γρηγορότερος από τον Wu-Manber, απαιτεί όμως περισσότερη μνήμη. Όταν ενδιαφερόμαστε για την απόδοση των αλγορίθμων πολλαπλής αναζήτησης πρέπει να λαμβάνουμε υπόψη ορισμένες παραμέτρους.

1. Τα εγγενή χαρακτηριστικά απόδοσης αλγορίθμου – η απόδοση εξαρτάται από τους κανόνες ή από τα δεδομένα.
2. Την απόδοση της αρχιτεκτονικής της CPU – κόστος πρόσβασης στη μνήμη, κόστος αστοχίας της cache, σχετικά κόστη εντολών μετατόπισης, πρόσθεσης, σύγκρισης και άλματος (jump).
3. Τον μεταγλωττιστή και τις χρησιμοποιούμενες βελτιστοποιήσεις.
4. Τις απαιτήσεις της εφαρμογής σε μνήμη και τις δυνατότητες του συστήματος.

Ο απλοϊκός αλγόριθμος Boyer-Moore πολλαπλής αναζήτησης χρησιμοποιείται για μικρές ομάδες κανόνων. Οι δοκιμές έχουν δείξει πως για 10 ή λιγότερους κανόνες ο αλγόριθμος αυτός είναι το ίδιο καλός ή και καλύτερος από τους υπόλοιπους.

2.3.6 Αναζήτηση χωρίς περιεχόμενο.

Η αναζήτηση χωρίς περιεχόμενο εφαρμόζεται για τους κανόνες που δεν περιέχουν την επιλογή ‘content’ ή ‘uricontent’. Οι κανόνες αυτοί εξετάζονται με την τυπική παραμετρική αναζήτηση του Snort. Οι κανόνες χωρίς περιεχόμενο ελέγχουν πεδία της επικεφαλίδας του πακέτου για περίεργες τιμές. Παρόλο που χρησιμοποιείται

απευθείας η παραμετρική αναζήτηση, η ταχύτητα αυτού του τύπου της αναζήτησης είναι πολύ καλή.

2.3.7 Επεξεργασία γεγονότων.

Κατά τη διάρκεια της εξέτασης ενός πακέτου κάθε κανόνας που επαληθεύεται προστίθεται σε μία λίστα. Μόλις ολοκληρωθεί η εξέταση του πακέτου τα γεγονότα μέσα στη λίστα επεξεργάζονται και επιλέγεται ένα από αυτά για καταγραφή.

Κάθε κανόνας που επαληθεύεται θεωρείται ως ένα γεγονός. Καθώς μπορεί με ένα πακέτο να σχετίζονται πολλά γεγονότα, κάθε γεγονός που εμφανίζεται εισάγεται σε μία λίστα γεγονότων μέχρι να ολοκληρωθεί η επεξεργασία του πακέτου. Κάθε γεγονός στη λίστα μπορεί να αντιστοιχεί σε οποιονδήποτε κανόνα του Snort. Μετά από την ολοκλήρωση της επεξεργασίας του πακέτου επιλέγεται ένα από τα γεγονότα για τελική καταγραφή και η λίστα αδειάζει.

Το Snort καταγράφει ένα γεγονός για κάθε πακέτο, γεγονός που καθιστά αναγκαία την ύπαρξη μίας πολιτικής επιλογής γεγονότων. Η επιλογή γεγονότων γίνεται με βάση τις εξής προτεραιότητες:

- Η σειρά με την οποία επεξεργάζονται τα γεγονότα: καταγραφή (log), προειδοποίηση (alert) ή έγκριση (pass).
- Οι κανόνες περιεχομένου υπερισχύουν των κανόνων ανώμαλων πακέτων.
- Κανόνες πεδίων πρωτοκόλλου με περιεχόμενο, όπως γεγονότα HTTP, προηγούνται των κανόνων γενικού περιεχομένου.
- Επαλήθευση μεγαλύτερου σε μήκος περιεχομένου υπερισχύει των μικρότερων περιεχομένων.

2.3.8 Τεχνικές προστασίας από επιθέσεις στέρησης εξυπηρέτησης (DoS).

Ο όρος «επίθεση στέρησης εξυπηρέτησης» (Denial Of Service attacks - DoS) χρησιμοποιείται για να περιγράψει την περίπτωση όπου το σύστημα ανίχνευσης

εισβολών δέχεται μεγάλο αριθμό από πακέτα φτιαγμένα με κατάλληλο τρόπο ώστε να απαιτούν μεγάλο χρόνο επεξεργασίας. Σε περίπτωση που ο αριθμός των πακέτων και ο απαιτούμενος χρόνος για το καθένα είναι αρκετά μεγάλος δεσμεύονται οι πόροι του συστήματος σε τέτοιο βαθμό ώστε να μη μπορεί να δεχτεί καινούργια πακέτα και να τα απορρίπτει.

Οι αλγόριθμοι πολλαπλής αναζήτησης ψάχνουν για όλα τα στιγμιότυπα όλων των ζητούμενων συμβολοσειρών σε κάθε πακέτο που εξετάζεται. Το γεγονός αυτό προσφέρει ιδανική ευκαιρία για επιθέσεις στέρησης εξυπηρέτησης. Σε περίπτωση που η μηχανή αναζήτησης ενεργοποιείται με κάθε εμφάνιση ζητούμενης συμβολοσειράς, τότε οποιαδήποτε επαναλαμβανόμενη συμβολοσειρά που αντιστοιχεί σε μια ύποπτη ακολουθία μπορεί να χρησιμοποιηθεί για να προκαλέσει στέρηση εξυπηρέτησης σε έναν απλοϊκό αλγόριθμο πολλαπλής αναζήτησης.

Για το λόγο αυτό έχει ενσωματωθεί στη μηχανή πολλαπλής αναζήτησης μια στρατηγική προστασίας από επιθέσεις στέρησης εξυπηρέτησης. Η στρατηγική αυτή αποτελείται από τα ακόλουθα σημεία:

1. Κάθε κανόνας που εξετάζεται στο πακέτο σημειώνεται κατάλληλα.
2. Οι σημειωμένοι κανόνες δεν επανεξετάζονται.
   A. Είτε ο κανόνας επαληθεύτηκε, οπότε βρίσκεται μέσα στη λίστα γεγονότων.
   B. Είτε απέτυχε να επαληθευτεί και δεν υπάρχει λόγος να επανεξεταστεί.

Τα πακέτα ελέγχονται διαδοχικά και εξετάζονται για εμφανίσεις περιεχομένων τύπου ‘content’ ή ‘uricontent’. Κάθε φορά που βρίσκουμε ένα ζητούμενο περιεχόμενο αυτό σημειώνεται και επικυρώνεται, επομένως μετά ουσιαστικά δε χρησιμοποιείται στην αναζήτηση του τρέχοντος πακέτου.

2.4 Προεπεξεργαστές.

Το Snort υποστηρίζει τη λειτουργία διαφόρων προεπεξεργαστών με τους οποίους επεκτείνεται η λειτουργικότητα του Snort, ενώ ο χρήστης μπορεί να υλοποιήσει τους

δικούς του προεπεξεργαστές και να τους συμπεριλάβει με ευκολία στο Snort. Ο κώδικας των προεπεξεργαστών εκτελείται μετά από την αποκωδικοποίηση του πακέτου και πριν από την εφαρμογή της μηχανής αναζήτησης.

Υπάρχουν διάφοροι προεπεξεργαστές για το Snort, οι οποίοι μπορούν να εκτελούν λειτουργίες διαμόρφωσης των εισερχόμενων πακέτων (για παράδειγμα ο προεπεξεργαστής 'Frag2' συναρμολογεί κατακερματισμένα πακέτα IP), να μετράνε την απόδοση του συστήματος ('perfmonitor'), να προστατεύουν από ορισμένα είδη επιθέσεων (για παράδειγμα ο 'stream4' παρακολουθεί την κατάσταση κάθε ροής TCP και απορρίπτει πακέτα από ροές που δεν έχουν δημιουργηθεί μέσω 3-way handshake) και άλλα.

Επειδή οι προεπεξεργαστές αποτελούν προέκταση της βασικής λειτουργικότητας και όχι απαραίτητο στοιχείο, δε θα αναφερθούμε με περισσότερες λεπτομέρειες σε αυτούς.

2.5 Οι τρόποι λειτουργίας του Snort.

Το Snort διαθέτει τους ακόλουθους τρεις διαφορετικούς τρόπους λειτουργίας:

1) *Παρατήρησης* (sniffer mode): όταν επιλέγεται αυτός ο τρόπος λειτουργίας, το Snort απλά διαβάζει τα εισερχόμενα πακέτα και εμφανίζει στην οθόνη τις επικεφαλίδες των πακέτων ή/και τα δεδομένα τους.
2) *Καταγραφής* (packet logger mode): με τη λειτουργία καταγραφής, το Snort απλά καταγράφει όλα τα εισερχόμενα πακέτα. Η καταγραφή των πακέτων μπορεί να γίνει με δύο τρόπους. Ο πρώτος τρόπος είναι λεπτομερής καταγραφή, οπότε τα πακέτα αποθηκεύονται σε διαφορετικούς φακέλους ανάλογα με τη διεύθυνση IP και η καταγραφή των πακέτων γίνεται σε μορφή εύκολα αναγνώσιμη από τον χρήστη. Ο δεύτερος τρόπος καταγραφής των πακέτων είναι σε ένα ενιαίο αρχείο σε μορφή tcpdump, δηλαδή με λίγες πληροφορίες για το πότε και που ήρθε το πακέτο αλλά χωρίς καμία αποκωδικοποίηση. Προφανώς ο δεύτερος τρόπος καταγραφής είναι ο γρηγορότερος.

3) *Ανίχνευσης εισβολών δικτύου* (Network Intrusion Detection): η πιο ενδιαφέρουσα μέθοδος λειτουργίας του Snort. Αρχικά διαβάζεται το κατάλληλο αρχείο ρυθμίσεων, στο οποίο περιέχονται οι πληροφορίες για τα αρχεία κανόνων που θα χρησιμοποιηθούν, τους προεπεξεργαστές και άλλα. Αφού ολοκληρωθεί η αρχικοποίηση και γίνει ο διαχωρισμός των κανόνων στις κατάλληλες ομάδες, το σύστημα είναι έτοιμο να δεχτεί τα πακέτα. Κάθε πακέτο που έρχεται αρχικά αποκωδικοποιείται και περνάει από τους προεπεξεργαστές (εάν υπάρχουν). Εφόσον το πακέτο δεν απορριφθεί τότε εξετάζεται για πιθανή επαλήθευση κανόνων. Αν το πακέτο είναι καθαρό και δεν ενεργοποιηθεί ούτε ένας κανόνας τότε το Snort δεν εκτελεί καμία ενέργεια. Σε περίπτωση που επαληθευτούν κανόνες τότε καταγράφεται (συνήθως) το πακέτο ενώ αποθηκεύονται και πληροφορίες σχετικά με τους κανόνες που ενεργοποιήθηκαν. Η μορφή με την οποία καταγράφονται τα πακέτα αλλά και η λεπτομέρεια των πληροφοριών για τους κανόνες που ενεργοποιήθηκαν επιλέγονται ανάλογα με τις απαιτήσεις του χρήστη.

2.6 Σύνοψη.

Στο κεφάλαιο αυτό παρουσιάστηκαν τα σημαντικότερα στοιχεία από την εσωτερική δομή και τη λειτουργία του Snort. Καθώς το Snort αποτελεί ένα από τα πλέον διαδεδομένα συστήματα ανίχνευσης εισβολών, η ανάλυση της δομής του μπορεί να μας δώσει σημαντικές πληροφορίες για το πως κατασκευάζεται ένα σωστό και αποδοτικό σύστημα ανίχνευσης εισβολών δικτύου. Πιο συγκεκριμένα, από τα χαρακτηριστικά που παρουσιάστηκαν στο κεφάλαιο αυτό, η προσοχή μας κατά τη διάρκεια της ανάπτυξης του TUCNIDS εστιάστηκε στη δομή και τις λειτουργίες των κανόνων του Snort, στον Βελτιστοποιητή Κανόνων, στον τρόπο διαχείρησης των ομάδων από κανόνες καθώς και στην αντιμετώπιση των συγκρούσεων ανάμεσα σε διαφορετικές ομάδες.

Κεφάλαιο 3.

Η θεωρία των αλγορίθμων αναζήτησης συμβολοσειρών.

Η σημαντικότερη λειτουργία κάθε Συστήματος Ανίχνευσης Εισβολών Δικτύου, η οποία αποτελεί και το πλέον απαιτητικό κομμάτι της λειτουργίας του συστήματος ως προς το χρόνο και την επεξεργαστική ισχύ που χρειάζεται, είναι η ανίχνευση των πακέτων για το αν περιέχουν ύποπτες συμβολοσειρές και ποιες μπορεί να είναι αυτές.

Το πρόβλημα της αναζήτησης συγκεκριμένων συμβολοσειρών μέσα σε κάποιο μεγαλύτερο κείμενο έχει απασχολήσει για αρκετά χρόνια όχι μόνο την προγραμματιστική κοινότητα αλλά και άλλους επιστημονικούς κλάδους (π.χ. βιολογία). Το αποτέλεσμα αυτού του εκτεταμένου προβληματισμού ήταν να προκύψουν αρκετοί «καλοί» αλγόριθμοι, χωρίς όμως να υπάρχει κάποιος που να είναι βέλτιστος για όλες τις περιπτώσεις. Έτσι, για οποιαδήποτε εφαρμογή στηρίζεται σε αναζήτηση συμβολοσειρών προκύπτει το ενδιαφέρον πρόβλημα της εύρεσης του αλγορίθμου ο οποίος ταιριάζει καλύτερα στις απαιτήσεις και τα δεδομένα του συγκεκριμένου προβλήματος. Τα κριτήρια με τα οποία γίνεται η επιλογή του αλγορίθμου είναι συνήθως η γενική απόδοση (ταχύτητα) του αλγορίθμου και το μέγεθος του προβλήματος (για παράδειγμα ο καλύτερος αλγόριθμος για την αναζήτηση 2 συμβολοσειρών δεν είναι απαραίτητα ο κατάλληλος σε περίπτωση έχουμε αναζήτηση 2000 συμβολοσειρών), ενώ σε κάποιες περιπτώσεις η επιλογή μπορεί να γίνει με βάση την πολυπλοκότητα υλοποίησης του αλγορίθμου ή τις απαιτήσεις του σε χώρο-μνήμη.

Ύστερα από θεωρητική μελέτη διαφόρων αλγορίθμων αναζήτησης συμβολοσειρών καταλήξαμε στους εξής τρεις για υλοποίηση και χρήση στο σύστημα TUCNIDS : τον αλγόριθμο *Aho-Corasick*, τον αλγόριθμο *Boyer-Moore* και τον *Wu-Manber*. Στις ακόλουθες παραγράφους του κεφαλαίου αυτού παρουσιάζεται σε θεωρητικό επίπεδο ο καθένας από αυτούς τους τρεις αλγόριθμος.

3.1 Ο αλγόριθμος Aho-Corasick.

Ο συγκεκριμένος αλγόριθμος προτάθηκε το 1975 από τους Alfred V. Aho και Margaret J. Corasick [1] και παραμένει μέχρι τις μέρες μας μία από τις πλέον ισχυρές μεθόδους αναζήτησης σε περίπτωση που θέλουμε να αναζητήσουμε έναν μεγάλο αριθμό λέξεων. Προέκυψε από το συνδυασμό των ιδεών του αλγορίθμου Knuth-Morris-Pratt με τη θεωρία πεπερασμένων αυτομάτων.

Συνοπτικά, η ιδέα του αλγορίθμου είναι ότι όλες οι συμβολοσειρές που θέλουμε να αναζητήσουμε συνδυάζονται σε ένα πεπερασμένο αυτόματο, στο οποίο δίνουμε στη συνέχεια σαν είσοδο το κείμενο που ψάχνουμε για τις συμβολοσειρές. Με ένα μόνο πέρασμα του κειμένου, δηλαδή έλεγχο όλων των χαρακτήρων του κειμένου από την αρχή μέχρι το τέλος, το αυτόματο βρίσκει όλα τα στιγμιότυπα των συμβολοσειρών που ψάχνουμε.

Τα σημαντικά προτερήματα του Aho-Corasick είναι ότι το κόστος για τον έλεγχο ενός κειμένου είναι ανεξάρτητο τόσο του αριθμού των συμβολοσειρών που ψάχνουμε όσο και του ελάχιστου μήκους, δηλαδή του μήκους της μικρότερης από τις συμβολοσειρές που αναζητούμε. Αντίστοιχα, τα μειονεκτήματα αυτής της μεθόδου είναι οι μεγάλες απαιτήσεις σε χώρο και το γραμμικό κόστος του αλγορίθμου ως προς το μήκος του κειμένου ακόμα και στην καλύτερη περίπτωση (best-case cost).

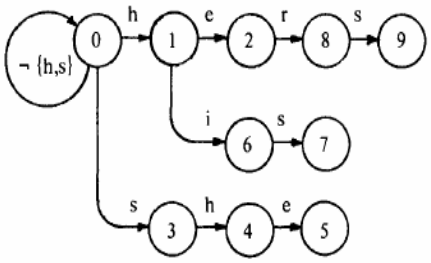
3.1.1 Μηχανή αναγνώρισης συμβολοσειρών.

Η υλοποίηση και λειτουργία του αλγορίθμου γίνεται ως εξής. Αρχικά έχουμε το σύνολο $K=\{y_1, y_2, \ldots, y_n\}$ των *λέξεων-κλειδιών* που αναζητούμε, όπου κάθε *λέξη* είναι μια ακολουθία χαρακτήρων πεπερασμένου μήκους, ενώ επίσης έχουμε την αυθαίρετη συμβολοσειρά x που ονομάζουμε *κείμενο*. Το ζητούμενο είναι να εντοπιστούν και να αναγνωριστούν όλες οι υπολέξεις (substrings) του x που αποτελούν στοιχεία του K. Επιπλέον, υπάρχει το ενδεχόμενο αλληλοεπικάλυψης των υπολέξεων μέσα στο x.

Μια μηχανή αναγνώρισης συμβολοσειρών για το K είναι ένα πρόγραμμα που δέχεται το x σαν είσοδο και επιστρέφει τις θέσεις μέσα στο x όπου εντοπίστηκαν λέξεις

του *K* ως υπολέξεις του κειμένου. Η μηχανή αναγνώρισης αποτελείται από ένα σύνολο *καταστάσεων*, με κάθε κατάσταση να αντιστοιχεί σε ένα μοναδικό αριθμό – ταυτότητα της κατάστασης. Η μηχανή επεξεργάζεται το κείμενο *x* διαβάζοντας διαδοχικά όλα τα σύμβολα του *x*, εκτελώντας τις κατάλληλες μεταβάσεις καταστάσεων και περιστασιακά τυπώνοντας μήνυμα στην έξοδο, όποτε εντοπίζεται μία από τις λέξεις. Η συμπεριφορά της μηχανής αναγνώρισης καθορίζεται από τρεις συναρτήσεις: Τη συνάρτηση μετάβασης (goto) *g*, τη συνάρτηση αποτυχίας (failure) *f* και τη συνάρτηση εξόδου *output*.

Στην εικόνα *3α* βλέπουμε τις συναρτήσεις που προκύπτουν για τη μηχανή αναγνώρισης του συνόλου λέξεων {he, she, his, hers}.



(a) Goto function.

i	1	2	3	4	5	6	7	8	9
$f(i)$	0	0	0	1	2	0	3	0	3

(b) Failure function.

i	*output* (i)
2	{he}
5	{she, he}
7	{his}
9	{hers}

(c) Output function.

Εικόνα 3α) οι συναρτήσεις μετάβασης, αποτυχίας και εξόδου για το παράδειγμα

Μία εκ των καταστάσεων (συνήθως η 0) καθορίζεται ως η *κατάσταση εκκίνησης*. Στην εικόνα *3α* οι καταστάσεις είναι οι 0,1,...,9. Η συνάρτηση μετάβασης μετατρέπει ένα ζεύγος, το οποίο αποτελείται από μία κατάσταση και ένα σύμβολο εισόδου, σε μία νέα κατάσταση ή σε μήνυμα αποτυχίας (fail). Ο κατευθυνόμενος γράφος της εικόνας *3α* απεικονίζει τη συνάρτηση μετάβασης. Για παράδειγμα, η ακμή με το γράμμα 'h' από την κατάσταση 0 στην 1 ισοδυναμεί με g(0,h) =1. Η απουσία ακμής για ορισμένα γράμματα σημαίνει *αποτυχία*. Έτσι, g(1,σ) = *αποτυχία* για όλα τα σύμβολα εισόδου σ που δεν είναι 'e' ή 'i'. Κάθε μηχανή αναγνώρισης έχει την ιδιότητα g(0,σ) ≠ *αποτυχία* για όλα τα σύμβολα εισόδου σ, ιδιότητα που αργότερα θα δείξουμε πως εγγυάται ότι σε κάθε κύκλο η μηχανή επεξεργάζεται ένα σύμβολο εισόδου.

Η συνάρτηση αποτυχίας *f* δημιουργεί μία αντιστοιχία κάθε κατάστασης σε μία άλλη κατάσταση. Χρησιμοποιούμε την *f* κάθε φορά που η συνάρτηση μετάβασης μας επιστρέφει *αποτυχία*. Ορισμένες καταστάσεις καθορίζονται ως καταστάσεις εξόδου,

γεγονός που υποδηλώνει ότι βρέθηκε κάποιο σύνολο λέξεων-κλειδιών. Η συνάρτηση εξόδου ταυτίζει με κάθε κατάσταση ένα σύνολο (πιθανώς κενό σύνολο).

Ένας κύκλος επεξεργασίας της μηχανής καθορίζεται ως εξής: Έστω *s* η τρέχουσα κατάσταση της μηχανής και *a* το σύμβολο του κειμένου *x* που θα επεξεργαστούμε τώρα. Υπάρχουν δύο περιπτώσεις. Στην πρώτη περίπτωση, εάν g(s,α) = s', η μηχανή κάνει μια μετάβαση στην κατάσταση s' και σαν σύμβολο εισόδου θεωρούμε το επόμενο γράμμα του κειμένου. Επιπλέον, αν το *output(s')* δεν είναι κενό σύνολο, η μηχανή εμφανίζει στην έξοδο το σύνολο *output(s')* καθώς και τη θέση στο κείμενο του τρέχοντος συμβόλου εισόδου. Στη δεύτερη περίπτωση, εάν g(s,α)=*αποτυχία* η μηχανή εξετάζει τη συνάρτηση αποτυχίας *f* και εκτελεί μια *μετάβαση αποτυχίας*. Εάν f(s)=s' η μηχανή επαναλαμβάνει τον κύκλο εκτέλεσης με *s'* σαν την τρέχουσα κατάσταση και *α* σαν το σύμβολο εισόδου.

Αρχικά, η τρέχουσα κατάσταση της μηχανής είναι η κατάσταση εκκίνησης και το πρώτο σύμβολο του κειμένου είναι το τρέχων σύμβολο εισόδου. Εν συνεχεία η μηχανή επεξεργάζεται το κείμενο εκτελώντας έναν κύκλο επεξεργασίας για κάθε σύμβολο του κειμένου.

Για παράδειγμα, θεωρούμε τη μηχανή *M* που περιγράφεται από τις συναρτήσεις της εικόνας *3α*, στην οποία εφαρμόζουμε σαν κείμενο εισόδου το "ushers". Η συμπεριφορά της θα είναι η ακόλουθη:

Είσοδος:	u	s	h	e	r	s	
Κατάσταση:	0	0	3	4	5	8	9
Μετάβαση αποτυχίας:				2			

Ας επικεντρωθούμε στον κύκλο επεξεργασίας όπου η *M* βρίσκεται στην κατάσταση 4 και το σύμβολο εισόδου είναι το *e*. Αφού g(4,e)=5 η μηχανή μεταβαίνει στην κατάσταση 5, πηγαίνει στο επόμενο σύμβολο εισόδου και εμφανίζει στην έξοδο το σύνολο output(5), υποδηλώνοντας ότι εντόπισε τις λέξεις "he" και "she" στο τέλος τις θέσης 4 του κειμένου.

Στην κατάσταση 5 με σύμβολο εισόδου r η μηχανή εκτελεί δύο μεταβάσεις στον κύκλο της. Εφόσον g(5,r)=*αποτυχία* η *M* μεταβαίνει στην κατάσταση 2 λόγω του f(5)=2. Στη συνέχεια, επειδή g(2,r)=8, η *M* εισέρχεται στην κατάσταση 8 και προχωράει στο επόμενο σύμβολο εισόδου. Επειδή output(8)=κενό δεν δημιουργείται κάποια έξοδος.

Ο ακόλουθος αλγόριθμος περιγράφει συνοπτικά, με μορφή ψευδοκώδικα, τη λειτουργία της μηχανής αναγνώρισης.

Αλγόριθμος Simple_AC: Μηχανή αναγνώρισης συμβολοσειρών.

Είσοδος: Ένα κείμενο εισόδου $x=\alpha_1\alpha_2...\alpha_n$, όπου κάθε α_i είναι ένα σύμβολο εισόδου και μία μηχανή αναγνώρισης *M* με συνάρτηση μετάβασης *g*, συνάρτηση αποτυχίας *f* και συνάρτηση εξόδου *output* όπως περιγράφηκαν προηγουμένως.

Έξοδος: Τοποθεσίες μέσα στο *x* όπου εντοπίστηκαν λέξεις-κλειδιά.

```
begin
        state ← 0
        for i ← 0 until n do
        begin
                while g(state,αi) = fail do state ← f(state)
                state ← g(state,αi)
                if output(state) ≠ empty then
                begin
                        print i
                        print output(state)
                end
        end
end
```

(Κάθε επανάληψη του **for**-loop αντιστοιχεί σε ένα κύκλο επεξεργασίας της μηχανής.)

Ο αλγόριθμος *Simple_AC* είναι μοντελοποιημένος παρόμοια με τον αλγόριθμο Knuth-Morris-Pratt [5] για την αναζήτηση μίας λέξης-κλειδί σε ένα κείμενο. Στην παράγραφο 3.1.5 περιγράφεται μια εκδοχή του αλγορίθμου που χρησιμοποιεί ντετερμινιστικό αυτόματο, αποφεύγοντας τις μεταβάσεις σε καταστάσεις αποτυχίας (fail-state transitions).

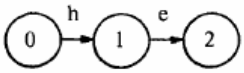
3.1.2 Κατασκευή των συναρτήσεων μετάβασης (goto), αποτυχίας (failure) και εξόδου (output).

Οι τρεις συναρτήσεις g, f και output θεωρούνται *έγκυρες* για ένα σύνολο λέξεων-κλειδιών αν με τη χρήση αυτών των συναρτήσεων ο αλγόριθμος 3.1 υποδεικνύει πως η λέξη-κλειδί *y* τελειώνει στη θέση *i* του κειμένου *x* αν και μόνο αν *x=uyv* και το μήκος του *uy* είναι *i*.

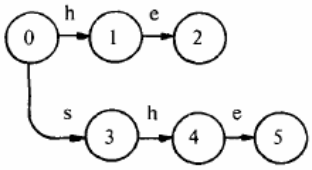
Στο σημείο αυτό θα περιγράψουμε πως κατασκευάζονται έγκυρες συναρτήσεις μετάβασης, αποτυχίας και εξόδου. Υπάρχουν δύο στάδια για κατασκευή των συναρτήσεων. Στο πρώτο στάδιο καθορίζονται οι καταστάσεις και η συνάρτηση μετάβασης, ενώ στο δεύτερο στάδιο υπολογίζεται η συνάρτηση αποτυχίας. Η κατασκευή της συνάρτησης εξόδου ξεκινάει στο πρώτο στάδιο και ολοκληρώνεται στο δεύτερο.

Για την κατασκευή της συνάρτησης μετάβασης κατασκευάζεται πρώτα ο *γράφος μεταβάσεων*. Η κατασκευή του γράφου ξεκινάει από ένα μοναδικό κόμβο που αντιπροσωπεύει την αρχική κατάσταση 0. Στη συνέχεια προστίθεται κάθε λέξη-κλειδί y στο γράφο με τη δημιουργία μιας κατευθυνόμενης διαδρομής, η οποία ξεκινάει από την αρχική κατάσταση και περιγράφει διαδοχικά όλα τα σύμβολα της λέξης. Η λέξη y προστίθεται στη συνάρτηση εξόδου της κατάστασης στην οποία ολοκληρώνεται η διαδρομή. Η προσθήκη κάθε καινούργιας λέξης γίνεται με τον τρόπο που θα γινόταν σε ένα δέντρο, με την αρχική μας κατάσταση να αντιστοιχεί στη ρίζα του δέντρου. Όσο οι χαρακτήρες του προθέματος της λέξης ταιριάζουν με τους αντίστοιχους άλλων λέξεων δε δημιουργούνται καινούργιες ακμές και κόμβοι, ενώ σε περίπτωση που βρούμε διαφορά προσθέτουμε ένα καινούργιο υποδέντρο.

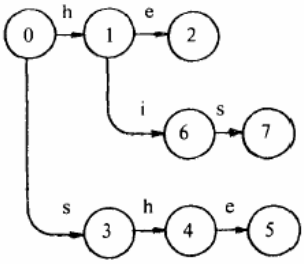
Ας εξετάσουμε το παράδειγμα που αναφέρθηκε παραπάνω με τις τέσσερις λέξεις-κλειδιά, τις {he,she,his,hers}. Με την προσθήκη της πρώτης λέξης στο γράφο προκύπτει η εξής μορφή:



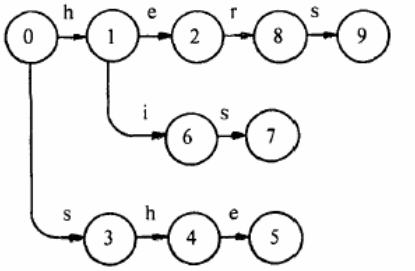
Η διαδρομή από την κατάσταση 0 στην κατάσταση 2 ισοδυναμεί με τη λέξη “he”, έτσι συσχετίζουμε την κατάσταση 2 με την έξοδο (output) “he”. Στη συνέχεια προστίθεται η λέξη “she” και ο γράφος αποκτά την ακόλουθη μορφή:



Στον γράφο που προέκυψε η κατάσταση 5 είναι συσχετισμένη με την έξοδο “she”. Μετά την εισαγωγή της λέξης “his” προκύπτει ο ακόλουθος γράφος:



Κατά την εισαγωγή της λέξης “his” παρατηρούμε ότι ήδη υπάρχει μία ακμή που ξεκινάει από την κατάσταση 0 στην 1 για το σύμβολο ‘h’, επομένως δε χρειάζεται να προσθέσουμε καινούργια ακμή που να ξεκινάει από την κατάσταση 0 για σύμβολο εισόδου το ‘h’. Η κατάσταση 7 είναι συσχετισμένη με την έξοδο “his”. Με την προσθήκη και τις τελευταίας λέξης, της “hers”, ο γράφος που προκύπτει είναι ο ακόλουθος:



Σε αυτή την περίπτωση αξιοποιήσαμε την ήδη υπάρχουσα μετάβαση για το σύμβολο ‘h’ από την κατάσταση 0 στην 1 και την μετάβαση για το σύμβολο ‘e’ από την κατάσταση 1 στην 2. Η κατάσταση 9 συσχετίζεται με την έξοδο “hers”.

Μέχρι το σημείο αυτό ο γράφος έχει τη μορφή κατευθυνόμενου δέντρου. Για να ολοκληρωθεί η κατασκευή της συνάρτησης μετάβασης προσθέτουμε ένα βρόχο από την κατάσταση 0 πίσω στον εαυτό της για όλα τα υπόλοιπα σύμβολα εισόδου, δηλαδή για κάθε σύμβολο εκτός από το 'h' και το 's'. Με τον τρόπο αυτό καταλήγουμε στο γράφο που υπάρχει στην εικόνα *3α*, ο οποίος αντιπροσωπεύει τη συνάρτηση μετάβασης.

Η συνάρτηση αποτυχίας (failure) κατασκευάζεται με τη βοήθεια της συνάρτησης μετάβασης. Ορίζουμε ως *βάθος* μίας κατάστασης *s* του γράφου μεταβάσεων ως το μήκος της συντομότερης διαδρομής από την αρχική κατάσταση στην κατάσταση *s*. Συνεπώς, για το γράφο της εικόνας *3α* προκύπτει ότι η αρχική κατάσταση 0 έχει βάθος 0, οι καταστάσεις 1 και 3 βρίσκονται σε βάθος 1, οι καταστάσεις 2, 4 και 6 έχουν βάθος 2 κ.ο.κ.

Θα υπολογίσουμε τη συνάρτηση αποτυχίας πρώτα για όλες τις καταστάσεις που έχουν βάθος, στη συνέχεια για όλες τις καταστάσεις με βάθος 2 και συνεχίζουμε εξετάζοντας τις καταστάσεις που βρίσκονται σε ολοένα και αυξανόμενο βάθος μέχρι να υπολογίσουμε τη συνάρτηση αποτυχίας για όλες τις καταστάσεις (πλην της αρχικής κατάστασης 0, για την οποία δεν ορίζεται κατάσταση αποτυχίας). Ο αλγόριθμος για τον υπολογισμό της συνάρτησης αποτυχίας *f* για μία κατάσταση είναι απλός σαν ιδέα. Αρχικά θέτουμε $f(s)=0$ για όλες τις καταστάσεις s που βρίσκονται σε βάθος 1. Ας υποθέσουμε τώρα ότι η *f* έχει υπολογιστεί για όλες τις καταστάσεις που βρίσκονται σε βάθος μικρότερο του *d*. Η συνάρτηση *f* για τις καταστάσεις βάθους *d* προκύπτει από τη συνάρτηση *f* για καταστάσεις μικρότερου βάθους. Οι καταστάσεις βάθους *d* μπορούν να βρεθούν από τις υπαρκτές μεταβάσεις που ξεκινούν από καταστάσεις βάθους *d*-1.

Συγκεκριμένα, για να υπολογιστεί η συνάρτηση αποτυχίας για τις καταστάσεις με βάθος *d* εξετάζεται κάθε κατάσταση *r* βάθους *d*-1 και εκτελούνται οι ακόλουθες ενέργειες.

1. Αν $g(r,\alpha)$ = *αποτυχία* για όλα τα α, δε γίνεται τίποτα.
2. Διαφορετικά, για κάθε σύμβολο α τέτοιο ώστε $g(r,\alpha) = s$ εκτελούνται τα ακόλουθα:
   (α) Θέτουμε $state = f(r)$.
   (β) Εκτελείται η εντολή $state = f(state)$ μηδέν ή περισσότερες φορές μέχρι να βρεθεί τιμή του *state* ώστε $g(state,\alpha) \neq$ *αποτυχία* (επισημαίνεται πως, από τη

στιγμή που $g(0,\alpha) \neq$ αποτυχία για όλα τα α, θα βρίσκεται πάντα μια κατάλληλη τιμή για το *state*).

(γ) Θέτουμε $f(s) = g(state, a)$.

Για παράδειγμα, στον υπολογισμό της συνάρτησης αποτυχίας από το γράφο της εικόνας *3α* η πρώτη κίνηση θα ήταν να θέσουμε $f(1) = f(3) = 0$ καθώς οι καταστάσεις 1 και 3 βρίσκονται σε βάθος 1. Στη συνέχεια μπορεί να υπολογιστεί η συνάρτηση αποτυχίας για τις καταστάσεις 2, 6 και 4 καθώς αυτές βρίσκονται σε βάθος 2. Για να βρεθεί η $f(2)$ θέτουμε $state = f(1) = 0$ και επειδή $g(0, e) = 0$ προκύπτει ότι $f(2) = 0$. Για τον υπολογισμό της $f(6)$ θέτουμε $state = f(1) = 0$, επειδή όμως $g(0, i) = 0$ έχουμε $f(6) = 0$. Στον υπολογισμό της $f(4)$ αρχικά έχουμε $state = f(3) = 0$ και επειδή $g(0, h)=1$ προκύπτει ότι $f(4) = 1$. Ακολουθώντας την ίδια διαδικασία βρίσκουμε τη συνάρτηση αποτυχίας για όλες τις καταστάσεις (οι τιμές για το παράδειγμα φαίνονται στον πίνακα *failure* της εικόνας *3α*.

Κατά τη διάρκεια του υπολογισμού της συνάρτησης αποτυχίας γίνεται και ενημέρωση της συνάρτησης εξόδου σε όσες καταστάσεις χρειάζεται. Όταν καθορίσουμε για μια κατάσταση s ότι $f(s) = s'$ τότε προσθέτουμε όλα τα στοιχεία του συνόλου $output(s')$ στο σύνολο $output(s)$.

Για παράδειγμα, από τον γράφο της εικόνας *3α* προκύπτει ότι $f(5) = 2$. Έτσι, προσθέτουμε τα στοιχεία του συνόλου $output(2)$, που είναι η λέξη {he}, με τα στοιχεία του συνόλου $output(5)$, δηλαδή {she}, και προκύπτει το νέο σύνολο $output(5)$ το οποίο περιέχει τα στοιχεία {he,she}. Τα τελικά σύνολα *output* για όλες τις καταστάσεις φαίνονται στον πίνακα *output* της εικόνας *3α*.

Η κατασκευή των συναρτήσεων μετάβασης, αποτυχίας και εξόδου από το σύνολο λέξεων-κλειδιών K περιγράφεται συνοπτικά από τους παρακάτω αλγόριθμο.

Αλγόριθμος AC_goto: Κατασκευή της συνάρτησης μετάβασης

Είσοδος: Σύνολο των λέξεων κλειδιών $K = \{y_1, y_2, \ldots, y_n\}$.

Έξοδος: Συνάρτηση μετάβασης g και μερικώς υπολογισμένη συνάρτηση εξόδου *output*.

Μέθοδος: Θεωρούμε το σύνολο $output(s)$ κενό τη στιγμή που δημιουργείται η κατάσταση s, και την $g(s, a)$ = *αποτυχία* σε περίπτωση που το α δεν είναι ορισμένο ή αν η $g(s, a)$ δεν έχει ακόμα οριστεί. Η διαδικασία $enter(y)$ εισάγει στο γράφο τη διαδρομή που αντιστοιχεί στην λέξη y.

```
begin
        newstate ← 0
        for i ← 1 until k do enter(y_i)
        for all a such that g(0, a) = fail do g(0, a) ← 0
end

procedure enter(a_1, a_2, …, a_m)
begin
        state ← 0; j ← 1
        while g(state, a_j) ≠ fail do
        begin
                state ← g(state, a_j)
                j ← j+1
        end
        for p ← j until m do
        begin
                newstate ← newstate + 1
                g(state, a_p) ← newstate
                state ← newstate
        end
        output(state) ← {a_1, a_2,…, a_m}
end
```

Αλγόριθμος AC_fail: Κατασκευή της συνάρτησης αποτυχίας.

Είσοδος: Συνάρτηση μετάβασης *g* και συνάρτηση εξόδου *output* από τον αλγόριθμο *AC_goto*.

Έξοδος: Συνάρτηση αποτυχίας *f* και συνάρτηση εξόδου *g*.

```
begin
        queue g(s, a) ← empty
        for each a such that g(0, a) = s ≠ 0 do
        begin
                queue ← queue U {s}
                f(s) ← 0
        end
        while queue ≠ empty do
        begin
```

```
            let r be the next state in queue
            queue ← queue – {r}
            for each a such that g(r, a) = s ≠ fail do
            begin
                queue ← queue U {s}
                state ← f(r)
                while g(state, a) = fail do state = f(state)
                f(s) ← g(state, a)
                output(s) = output(s) U output( f(s) )
            end
        end
end
```

Ο πρώτος βρόχος **for** υπολογίζει τις καταστάσεις βάθους 1 και της εισάγει σε μία λίστα first-in first-out η οποία στο πρόγραμμα ορίζεται σαν τη μεταβλητή *queue*. Ο κεντρικός βρόχος **while** υπολογίζει το σύνολο των καταστάσεων βάθους *d* από το σύνολο των καταστάσεων βάθους *d*-1.

Η συνάρτηση αποτυχίας που παράγει ο αλγόριθμος *AC_fail* δεν είναι βέλτιστη υπό την ακόλουθη έννοια. Θεωρούμε τη μηχανή αναγνώρισης *M* της εικόνας *3α*. Παρατηρούμε ότι $g(4, e) = 5$. Αν η *M* βρίσκεται στην κατάσταση 4 και το τρέχον σύμβολο εισόδου a_i είναι διαφορετικό από το *e* τότε η *M* μεταβαίνει στην κατάσταση $f(4) = 1$. Εφόσον η *M* έχει ήδη διαπιστώσει πως $a_i \neq e$, η *M* δε χρειάζεται να εξετάσει την τιμή της συνάρτησης μετάβασης από την κατάσταση 1 για σύμβολο εισόδου *e*. Επιπλέον, αν η λέξη "his" δεν υπήρχε τότε η *M* θα μπορούσε να μεταβεί από την κατάσταση 4 απευθείας στην κατάσταση 0, αποφεύγοντας μια περιττή ενδιάμεση μετάβαση στην κατάσταση 1.

Οι περιττές μεταβάσεις μπορούν να αποφευχθούν αν χρησιμοποιήσουμε μια διαφορετική έκδοση του αλγορίθμου *Simple_AC*, η οποία χρησιμοποιεί ντετερμινιστικό αυτόματο και περιγράφεται στην παράγραφο 3.1.5.

3.1.3 Ιδιότητες των αλγορίθμων *Simple_AC*, *AC_goto* και *AC_fail*.

Σε αυτή την παράγραφο θα δειχτεί ότι οι συναρτήσεις μετάβασης, αποτυχίας και εξόδου που κατασκευάζονται με την βοήθεια των αλγορίθμων *AC_goto* και *AC_fail* για ένα σύνολο λέξεων *K* ισχύουν πραγματικά για το *K*.

Κάθε συμβολοσειρά μπορεί να συμβολιστεί με *uv*, όπου το *u* συμβολίζει το πρόθεμα της συμβολοσειράς ενώ το *v* είναι η κατάληξη. Αν το *u* δεν είναι κενή συμβολοσειρά τότε λέμε ότι είναι *υπαρκτό* πρόθεμα. Παρομοίως, αν το *v* δεν είναι κενή συμβολοσειρά τότε είναι *υπαρκτή* κατάληξη.

Θεωρούμε πως μία συμβολοσειρά *u* ισοδυναμεί με την κατάσταση *s* μίας μηχανής αναγνώρισης αν η συντομότερη διαδρομή στο γράφο μεταβάσεων από την αρχική κατάσταση στην *s* ισοδυναμεί με τη *u*. Η αρχική κατάσταση ισοδυναμεί με την κενή συμβολοσειρά.

Η πρόταση που θα διατυπώσουμε τώρα χαρακτηρίζει τη συνάρτηση αποτυχίας που κατασκευάζεται με τη βοήθεια του αλγορίθμου *AC_fail*.

ΠΡΟΤΑΣΗ 1. *Υποθέτουμε ότι στον γράφο μεταβάσεων η κατάσταση s ισοδυναμεί με τη συμβολοσειρά u και η κατάσταση t ισοδυναμεί με τη συμβολοσειρά v. Στην περίπτωση αυτή, f(s) = t αν και μόνο αν η v είναι το μεγαλύτερο υπαρκτό πρόθεμα της u και είναι επίσης πρόθεμα κάποιας λέξης-κλειδί.*

ΑΠΟΔΕΙΞΗ. Η απόδειξη προκύπτει με αναγωγή ως προς το μήκος της *u* (προκύπτει επίσης ισοδύναμα ως προς το βάθος της *s*). Από τον αλγόριθμο *AC_fail* έχουμε $f(s) = 0$ για όλες τις καταστάσεις *s* με βάθος 1. Αφού κάθε κατάσταση βάθους 1 ισοδυναμεί με μια συμβολοσειρά μήκους 1 η πρόταση ισχύει για όλες τις συμβολοσειρές μήκους 1.

Για να εφαρμόσουμε την αναγωγή, δεχόμαστε ότι η πρόταση 1 ισχύει για όλες τις συμβολοσειρές μήκους *j*, όπου $j > 1$. Υποθέτουμε μια συμβολοσειρά $u = a_1a_2...a_j$ και επίσης τη συμβολοσειρά *v* που είναι το μακρύτερο υπαρκτό πρόθεμα της *u*, ενώ είναι και πρόθεμα κάποιας λέξης-κλειδί. Υποθέτουμε ότι η *u* ισοδυναμεί με την κατάσταση *s* και η

συμβολοσειρά $a_1a_2...a_{j-1}$ ισοδυναμεί με την κατάσταση r. Έστω $r_1, r_2, \ldots, r_n$ μια ακολουθία καταστάσεων τέτοια ώστε

1. $r_1 = f(r)$,
2. $r_{i+1} = f(r_i)$ όταν $1 \leq i < n$,
3. $g(r_i, a_j) =$ *αποτυχία* όταν $1 \leq i < n$ και
4. $g(r_n, a_j) = t \neq$ *αποτυχία*.

(Σε περίπτωση που $g(r_1, a_j) \neq$ *αποτυχία* τότε $r_1 = r_j$). Η ακολουθία $r_1, r_2, \ldots, r_n$ ισοδυναμεί με την ακολουθία των τιμών που παίρνει η μεταβλητή *state* στον εσωτερικό βρόχο **while** του αλγορίθμου *AC_fail*. Η εντολή αμέσως μετά από αυτόν τον βρόχο **while** έχει σαν αποτέλεσμα $f(s) = t$. Ισχυριζόμαστε ότι η t αντιστοιχεί στο μεγαλύτερο υπαρκτό πρόθεμα της u, το οποίο επίσης είναι πρόθεμα κάποιας λέξης-κλειδί.

Για να το αποδείξουμε αυτό, ας υποθέσουμε τη συμβολοσειρά v_i η οποία αντιστοιχεί στην κατάσταση r_i, όπου $1 \leq i \leq n$. Από την υπόθεση που έχουμε κάνει και στην οποία έχουμε στηριχτεί ώστε να αποδείξουμε την πρόταση μέσω αναγωγής, η v_1 είναι το μεγαλύτερο υπαρκτό πρόθεμα της ακολουθίας $a_1a_2...a_{j-1}$, ενώ επίσης αποτελεί πρόθεμα κάποιας λέξης-κλειδί. Η v_2 είναι το μεγαλύτερο υπαρκτό πρόθεμα της v_1 και είναι πρόθεμα κάποιας λέξης-κλειδί. Η v_3 είναι το μεγαλύτερο υπαρκτό πρόθεμα της v_2 και είναι πρόθεμα κάποιας λέξης-κλειδί και συνεχίζουμε με την ίδια λογική.

Επομένως, η v_n είναι το μεγαλύτερο υπαρκτό πρόθεμα της $a_1a_2...a_{j-1}$ έτσι ώστε η συμβολοσειρά v_na_j να αποτελεί πρόθεμα μίας λέξης-κλειδί. Έτσι προκύπτει πως η συμβολοσειρά v_na_j είναι το μακρύτερο υπαρκτό πρόθεμα της u που είναι και πρόθεμα κάποιας λέξης-κλειδί. Από τη στιγμή που ο αλγόριθμος *AC_fail* θέτει $f(s) = g(r_n, a_j) = t$ η απόδειξη ολοκληρώθηκε.

Η ακόλουθη πρόταση χαρακτηρίζει τη συνάρτηση εξόδου που κατασκευάζεται με τη βοήθεια των αλγορίθμων *AC_goto* και *AC_fail*.

ΠΡΟΤΑΣΗ 2. *Το σύνολο output(s) εμπεριέχει τη λέξη y αν και μόνο αν η y είναι λέξη-κλειδί η οποία αποτελεί κατάληξη της συμβολοσειράς που ισοδυναμεί με την κατάσταση s.*

ΑΠΟΔΕΙΞΗ. Από τον αλγόριθμο *AC_goto* προκύπτει ότι κάθε φορά που εισάγεται στο γράφο μεταβάσεων μία κατάσταση s, η οποία ισοδυναμεί με την λέξη-κλειδί y, δημιουργείται το σύνολο *output*(s) το οποίο περιέχει την y. Δεδομένης αυτής της αρχικοποίησης θα αποδείξουμε, μέσω αναγωγής ως προς το βάθος της κατάστασης s, πως για το σύνολο *output*(s) ισχύει η έκφραση *output*(s) = { y | η y είναι λέξη-κλειδί η οποία αποτελεί κατάληξη της συμβολοσειράς που ισοδυναμεί με την κατάσταση s}.

Η έκφραση αυτή σίγουρα ισχύει για την αρχική κατάσταση, που βρίσκεται σε βάθος 0. Κάνοντας την παραδοχή ότι η έκφραση ισχύει για όλες τις καταστάσεις βάθους μικρότερου από d, θεωρούμε μία κατάσταση s βάθους d. Έστω u η συμβολοσειρά που αντιστοιχεί στην κατάσταση s.

Θεωρούμε μία συμβολοσειρά y που εμπεριέχεται στο σύνολο *output*(s). Εάν η y είχε εισαχθεί στο σύνολο *output*(s) από τον αλγόριθμο *AC_goto*, τότε $y=u$ και η y είναι λέξη-κλειδί. Σε περίπτωση που η y είχε εισαχθεί στο σύνολο *output*(s) από τον αλγόριθμο *AC_fail* προκύπτει ότι η y ανήκει στο σύνολο *output*($f(s)$). Από την παραδοχή που κάναμε για να εφαρμόσουμε την αναγωγή, η y είναι λέξη-κλειδί η οποία είναι κατάληξη της συμβολοσειράς που ισοδυναμεί με την κατάσταση $f(s)$. Μέσω της πρότασης 1, κάθε τέτοια λέξη-κλειδί πρέπει να είναι πρόθεμα της u.

Αντίστροφα, θεωρούμε την y ως μία λέξη-κλειδί που είναι και κατάληξη της u. Αφού η y είναι λέξη-κλειδί, υπάρχει μία κατάσταση t που αντιστοιχεί στην συμβολοσειρά y. Από τον αλγόριθμο *AC_goto* είναι σίγουρο πως το σύνολο *output*(t) περιέχει την y. Επομένως, σε περίπτωση που $y = u$ προκύπτει πως $s = t$ και το *output*(s) σίγουρα περιέχει την y. Εάν η y είναι υπαρκτό πρόθεμα της u, τότε μέσω της παραδοχής που κάναμε για την αναγωγή και της πρότασης 1 γνωρίζουμε ότι το σύνολο *output*($f(s)$) περιέχει την y. Λόγω της δομής του αλγορίθμου *AC_fail*, ο οποίος εξετάζει τις καταστάσεις κατά σειρά αυξανόμενου βάθους, η τελευταία εντολή του αλγορίθμου *AC_fail* προσθέτει όλα τα στοιχεία του συνόλου *output*($f(s)$), επομένως και την y, στο σύνολο *output*(s).

Η ακόλουθη πρόταση χαρακτηρίζει τη συμπεριφορά του αλγορίθμου *Simple_AC* για ένα κείμενο εισόδου $x = a_1a_2...a_n$.

ΠΡΟΤΑΣΗ 3. *Μετά από τον j-οστό κύκλο επεξεργασίας, ο αλγόριθμος 1 θα βρίσκεται στην κατάσταση s αν και μόνο αν η s αντιστοιχεί στη μεγαλύτερου μήκους κατάληξη της* $a_1a_2...a_j$ *που είναι πρόθεμα κάποιας λέξης-κλειδί.*

ΑΠΟΔΕΙΞΗ. Παρόμοια με την απόδειξη της πρότασης 1.

ΘΕΩΡΗΜΑ 1. *Οι αλγόριθμοι AC_goto και AC_fail δημιουργούν σωστές συναρτήσεις μετάβασης, αποτυχίας και εξόδου.*

ΑΠΟΔΕΙΞΗ. Προκύπτει από τις προτάσεις 2 και 3.

3.1.4 Χρονική πολυπλοκότητα των αλγορίθμων *Simple_AC*, *AC_goto* και *AC_fail*.

Στην παράγραφο αυτή θα εξεταστεί η χρονική πολυπλοκότητα των αλγορίθμων *Simple_AC*, *AC_goto* και *AC_fail*. Θα δειχτεί πως, όταν χρησιμοποιούνται οι συναρτήσεις μετάβασης, αποτυχίας και εξόδου που κατασκευάζονται με τη βοήθεια των αλγορίθμων *AC_goto* και *AC_fail*, ο αριθμός των μεταβάσεων κατάστασης που εκτελεί ο αλγόριθμος *Simple_AC* κατά την επεξεργασία ενός κειμένου είναι ανεξάρτητος από τον αριθμό των λέξεων-κλειδιών. Επιπλέον θα δειχτεί πως οι αλγόριθμοι *AC_goto* και *AC_fail* μπορούν να υλοποιηθούν με κατάλληλο τρόπο ώστε η εκτέλεσή τους να απαιτεί χρόνο γραμμικά ανάλογο με το άθροισμα των μηκών όλων των λέξεων-κλειδιών που εμπεριέχονται στο σύνολο *K*.

ΘΕΩΡΗΜΑ 2. *Χρησιμοποιώντας της συναρτήσεις μετάβασης, αποτυχίας και εξόδου που δημιουργούνται από τους αλγορίθμους AC_goto και AC_fail , ο αλγόριθμος Simple_AC εκτελεί λιγότερες από 2n μεταβάσεις κατάστασης κατά την επεξεργασία ενός κειμένου εισόδου μήκους n χαρακτήρων.*

ΑΠΟΔΕΙΞΗ. Σε κάθε κύκλο επεξεργασίας ο αλγόριθμος *Simple_AC* εκτελεί μηδέν ή περισσότερες μεταβάσεις αποτυχίας (failure transitions) ακολουθούμενες από ακριβώς μία κανονική μετάβαση (goto transition). Από μία κατάσταση *s* βάθους *d* ο αλγόριθμος *Simple_AC* δεν μπορεί ποτέ να εκτελέσει περισσότερες από *d* μεταβάσεις αποτυχίας σε ένα κύκλο επεξεργασίας. Κατά την επεξεργασία ενός κείμενου εισόδου

μήκους n ο αλγόριθμος *Simple_AC* εκτελεί ακριβώς n κανονικές μεταβάσεις. Επομένως, ο συνολικός αριθμός μεταβάσεων κατάστασης είναι μικρότερος από $2n$.

Η πραγματική χρονική πολυπλοκότητα του αλγορίθμου *Simple_AC* εξαρτάται από το πόσο μας κοστίζει όταν θέλουμε:

1. Να προσδιορίσουμε την $g(s, a)$ για κάθε κατάσταση s και κάθε σύμβολο εισόδου a,
2. να προσδιορίσουμε την $f(s)$ για κάθε κατάσταση s,
3. να διαπιστώσουμε αν το σύνολο $output(s)$ είναι κενό ή όχι, και
4. να βγάλουμε το $output(s)$ στην κατάλληλη έξοδο.

Η συνάρτηση μετάβασης θα μπορούσε να αποθηκευτεί σε έναν πίνακα δυο διαστάσεων, ο οποίος θα επέτρεπε να καθορίσουμε την τιμή $g(s, a)$ σε σταθερό χρόνο για όλα τα s και a. Σε περίπτωση όμως που το μέγεθος του αλφάβητου (δηλαδή οι διακριτές τιμές που μπορεί να πάρει το σύμβολο εισόδου a) και το σύνολο των λέξεων-κλειδιών είναι μεγάλο, είναι πιο οικονομικό από θέμα χώρου να διατηρούνται μόνο οι υπαρκτές μεταβάσεις σε μία ταξινομημένη, ως προς το σύμβολο εισόδου, γραμμική λίστα για κάθε κατάσταση s, την οποία ύστερα θα ψάχνουμε χρησιμοποιώντας δυαδική αναζήτηση (binary search). Μια τέτοια αναπαράσταση θα έκανε το κόστος προσδιορισμού της $g(s, a)$ λογαριθμικά ανάλογο με τον αριθμό των υπαρκτών μεταβάσεων από την κατάσταση s. Ένας λογικός συμβιβασμός είναι να διατηρούνται οι πιο συχνά χρησιμοποιούμενες καταστάσεις, όπως η αρχική κατάσταση 0, σε πίνακες άμεσης πρόσβασης από τους οποίους μπορούμε να βρούμε την επόμενη κατάσταση χρησιμοποιώντας ως δείκτη για τον πίνακα το σύμβολο εισόδου. Με τον τρόπο αυτό μπορούμε να βρίσκουμε, για τις καταστάσεις που χρησιμοποιούνται πιο συχνά, την τιμή της $g(s, a)$ σε σταθερό χρόνο για όλα τα a. Για τις καταστάσεις που δε χρησιμοποιούνται τόσο συχνά ή έχουν λίγες υπαρκτές μεταβάσεις συμφέρει η δεύτερη μέθοδος, με τη συνάρτηση μετάβασης να διατηρείται σε γραμμική ταξινομημένη λίστα.

Μία άλλη αντιμετώπιση, ισοδύναμη με τη μέθοδο της ταξινομημένης γραμμικής λίστας, είναι να διατηρούνται οι υπαρκτές μεταβάσεις σε ένα δυαδικό δέντρο για κάθε κατάσταση.

Η συνάρτηση αποτυχίας μπορεί να αποθηκεύεται σε ένα μονοδιάστατο πίνακα έτσι ώστε η $f(s)$ να βρίσκεται σε σταθερό χρόνο για κάθε s.

Έτσι, το κομμάτι του αλγορίθμου *Simple_AC* που δεν εμφανίζει κάποια έξοδο μπορεί να υλοποιηθεί ώστε να επεξεργάζεται ένα κείμενο μήκους n σε cn βήματα, όπου c μία σταθερά ανεξάρτητη από τον αριθμό των λέξεων-κλειδιών.

Ας εξετάσουμε τώρα τον απαιτούμενο χρόνο για την εμφάνιση μίας εξόδου. Μπορούμε να χρησιμοποιήσουμε έναν μονοδιάστατο πίνακα ώστε να βρίσκουμε σε σταθερό χρόνο αν το σύνολο *output*(s) είναι κενό για οποιαδήποτε κατάσταση s. Το κόστος της εμφάνισης της εξόδου σε κάθε κύκλο επεξεργασίας είναι ανάλογο με το άθροισμα των μηκών των λέξεων στο σύνολο *output*(s), όπου s η κατάσταση στην οποία βρίσκεται ο αλγόριθμος *Simple_AC* στο τέλος ενός κύκλου επεξεργασίας. Στις περισσότερες εφαρμογές, το σύνολο *output*(s) θα περιέχει, συνήθως, το πολύ μία λέξη, επομένως ο απαιτούμενος χρόνος για την εμφάνιση της εξόδου σε κάθε θέση της εισόδου είναι σταθερός.

Βεβαίως, είναι πιθανό ένας μεγάλος αριθμός από λέξεις-κλειδιά να αντιστοιχεί σε κάθε θέση της εισόδου. Σε αυτή την περίπτωση ο αλγόριθμος 3.1.1 θα σπαταλήσει σημαντικό χρόνο για την εμφάνιση της εξόδου. Στη χειρότερη περίπτωση θα χρειάζεται να εκτυπώνουμε σχεδόν τις λέξεις του K σχεδόν σε κάθε θέση της εισόδου. Σαν παράδειγμα ας θεωρήσουμε την τετριμμένη περίπτωση στην οποία $K = \{\alpha, \alpha^2, \alpha^3, ..., \alpha^K\}$ και το κείμενο εισόδου είναι το α^n. Παρόλα αυτά, οποιοσδήποτε άλλος αλγόριθμος αναζήτησης συμβολοσειρών θα ήταν αναγκασμένος να εμφανίσει τον ίδιο αριθμό εξόδων σε κάθε θέση του κειμένου εισόδου, επομένως είναι λογικό να χρησιμοποιούμε σα μέτρο σύγκρισης ανάμεσα στους αλγορίθμους το χρόνο που ξοδεύει ο κάθε ένας για να βρει τις θέσεις στις οποίες αντιστοιχούν λέξεις-κλειδιά.

Τέλος, ας εξετάσουμε το κόστος υπολογισμού των συναρτήσεων μετάβασης, αποτυχίας και εξόδου με τη βοήθεια των αλγορίθμων *AC_goto* και *AC_fail*.

ΘΕΩΡΗΜΑ 3. *Ο αλγόριθμος AC_goto χρειάζεται χρόνο γραμμικά ανάλογο με το άθροισμα των μηκών των λέξεων-κλειδιών.*

ΑΠΟΔΕΙΞΗ. Προφανής.

ΘΕΩΡΗΜΑ 4. Ο αλγόριθμος *AC_fail* μπορεί να υλοποιηθεί ώστε να εκτελείται σε χρόνο ανάλογο με το άθροισμα των μηκών των λέξεων-κλειδιών.

ΑΠΟΔΕΙΞΗ. Χρησιμοποιώντας λογική παρόμοια με εκείνη του θεωρήματος 2, μπορεί να δειχτεί ότι ο συνολικός αριθμός εκτελέσεων της εντολής $state \leftarrow f(state)$ κατά τη διάρκεια της εκτέλεσης του αλγορίθμου *AC_fail* φράσσεται από το άθροισμα των μηκών των λέξεων-κλειδιών. Χρησιμοποιώντας συνδεδεμένες λίστες για να απεικονίσουμε το σύνολο εξόδου για κάθε κατάσταση, μπορούμε να εκτελούμε την εντολή $output(s) \leftarrow output(s) \cup output(f(s))$ σε σταθερό χρόνο. Ας σημειωθεί ότι τα σύνολα $output(s)$ και $output(f(s))$ είναι διακριτά πριν την εκτέλεση της εντολής. Επομένως, ο κυρίαρχος παράγοντας στον χρόνο εκτέλεσης του αλγορίθμου *AC_fail* είναι το άθροισμα των μηκών των λέξεων-κλειδιών.

3.1.5 Απαλοιφή των μεταβάσεων αποτυχίας (failure transitions).

Σε αυτό το σημείο θα δειχτεί ο τρόπος με τον οποίο μπορούν να απαλειφθούν οι μεταβάσεις αποτυχίας από τον αλγόριθμο *Simple_AC* χρησιμοποιώντας τη συνάρτηση επόμενου βήματος ενός ντετερμινιστικού αυτομάτου στη θέση των συναρτήσεων μετάβασης και αποτυχίας.

Ένα ντετερμινιστικό αυτόματο αποτελείται από ένα πεπερασμένο σύνολο καταστάσεων S και μία συνάρτηση επόμενου βήματος δ με κατάλληλη μορφή ώστε για κάθε κατάσταση s και σύμβολο εισόδου α, η τιμή $\delta(s, a)$ να είναι μία κατάσταση του S. Δηλαδή, συνοπτικά, ένα ντετερμινιστικό αυτόματο εκτελεί ακριβώς μία μετάβαση κατάστασης για κάθε σύμβολο εισόδου. Χρησιμοποιώντας τη συνάρτηση επόμενου βήματος δ ενός κατάλληλου ντετερμινιστικού αυτομάτου στη θέση της συνάρτησης μετάβασης στον αλγόριθμο *Simple_AC* μπορούμε να αποφύγουμε όλες τις μεταβάσεις αποτυχίας. Αυτό επιτυγχάνεται με την απλή αντικατάσταση των δύο πρώτον εντολών στο βρόχο **for** του αλγορίθμου *Simple_AC* με την εντολή $state \leftarrow \delta(state, a_i)$. Με τη χρήση της συνάρτησης δ, ο αλγόριθμος *Simple_AC* εκτελεί ακριβώς μία μετάβαση κατάστασης για κάθε σύμβολο εισόδου.

Η κατάλληλη συνάρτηση επόμενου βήματος δ μπορεί να υπολογιστεί από τις συναρτήσεις μετάβασης και αποτυχίας που βρίσκονται με τη βοήθεια των αλγορίθμων *AC_goto* και *AC_fail* εφαρμόζοντας τον αλγόριθμο *Final_AC*. Ο αλγόριθμος *Final_AC* απλά υπολογίζει εκ των προτέρων το αποτέλεσμα κάθε πιθανής ακολουθίας μεταβάσεων αποτυχίας. Ο χρόνος που απαιτεί ο αλγόριθμος *Final_AC* είναι γραμμικά ανάλογος με το μέγεθος του συνόλου των λέξεων-κλειδιών. Στην πράξη, ο αλγόριθμος *Final_AC* μπορεί να υπολογίζεται ταυτόχρονα με τον αλγόριθμο *AC_fail*.

Αλγόριθμος Final_AC: Κατασκευή ντετερμινιστικού πεπερασμένου αυτόματου.

Είσοδος: Συνάρτηση μετάβασης *g* από τον αλγόριθμο *AC_goto* και συνάρτηση αποτυχίας *f* από τον αλγόριθμο *AC_fail*.

Έξοδος: Συνάρτηση επόμενου βήματος δ.

```
begin
    queue ← empty
    for each symbol a do
    begin
        δ(0, a) ← g(0, a)
        if g (0, a) ≠ 0 then queue = queue U { g (0, a) }
    end
    while queue ≠ empty do
    begin
        let r be the next state in queue
        queue ← queue - {r}
        for each symbol a do
            if g(r, a) = s ≠ fail do
            begin
                queue ← queue U {s}
                δ(r, a) ← s
            end
            else δ (r, a) ← δ ( f(r), a)
    end
end
```

Η συνάρτηση επόμενου βήματος που υπολογίζεται με τον αλγόριθμο *Final_AC* από τις συναρτήσεις μετάβασης και αποτυχίας για το παράδειγμα της εικόνας *3α* φαίνεται στην εικόνα *3β*.

	input symbol	next state
state 0:	h	1
	s	3
	.	0
state 1:	e	2
	i	6
	h	1
	s	3
	.	0
state 9: state 7: state 3:	h	4
	s	3
	.	0
state 5: state 2:	r	8
	h	1
	s	3
	.	0
state 6:	s	7
	h	1
	.	0
state 4:	e	5
	i	6
	h	1
	s	3
	.	0
state 8:	s	9
	h	1
	.	0

Εικόνα 3β) η συνάρτηση επόμενου βήματος

Η συνάρτηση επόμενου βήματος που περιγράφεται στην εικόνα *3β* ερμηνεύεται με τον ακόλουθο τρόπο. Για παράδειγμα, από την κατάσταση 0 έχουμε μετάβαση για σύμβολο εισόδου 'h' στην κατάσταση 1, μετάβαση για σύμβολο εισόδου 's' στην κατάσταση 3 και μετάβαση για οποιοδήποτε άλλο σύμβολο εισόδου στην κατάσταση 0. Σε κάθε κατάσταση, η κουκκίδα '**.**' συμβολίζει όλα τα σύμβολα εισόδου εκτός από αυτά που έχουν ήδη δηλωθεί για τη συγκεκριμένη κατάσταση. Αυτή η μέθοδος απεικόνισης της συνάρτησης επόμενου βήματος είναι οικονομικότερη από ότι να αποθηκεύαμε τη συνάρτηση δ σε ένα πίνακα δύο διαστάσεων. Παρόλα αυτά, ο απαιτούμενος χώρος στη μνήμη για την αποθήκευση της δ είναι κάπως μεγαλύτερος από εκείνον που απαιτείται για την αντίστοιχη συνάρτηση g από την οποία κατασκευάστηκε η δ καθώς πολλές από τις καταστάσεις στην δ περιέχουν η κάθε μία μεταβάσεις από αρκετές καταστάσεις της συνάρτησης g.

Η χρήση του ντετερμινιστικού πεπερασμένου αυτόματου στον αλγόριθμο *Simple_AC* μπορεί να μειώσει τον αριθμό μεταβάσεων κατάστασης μέχρι και 50%. Αυτό το ποσοστό εξοικονόμησης όμως είναι πρακτικά ανέφικτο καθώς στις τυπικές εφαρμογές ο αλγόριθμος *Simple_AC* θα βρίσκεται τον περισσότερο χρόνο στην κατάσταση 0, από την οποία δεν υπάρχουν μεταβάσεις αποτυχίας. Ο υπολογισμός με ακρίβεια του ποσοστού μείωσης των μεταβάσεων κατάστασης είναι δύσκολος, καθώς δεν υπάρχει

ένας συγκεκριμένος ορισμός για το «τυπικό» σύνολο λέξεων-κλειδιών και για το «τυπικό» κείμενο.

3.2 Ο αλγόριθμος Boyer-Moore.

Ο αλγόριθμος αυτός δημοσιεύτηκε το 1977 από τους Robert S. Boyer και J Strother Moore [2]. Μέχρι και σήμερα βρίσκει συχνά χρήση, είτε στην αρχική του μορφή είτε σε διάφορες παραλλαγές που έχουν προταθεί κατά καιρούς (όπως η παραλλαγή του Horspool, των Apostolico και Giancarlo και άλλοι).

Αποτελεί την πιο αποτελεσματική λύση σε περίπτωση που αναζητούμε μία μόνο συμβολοσειρά σε ένα μεγαλύτερο κείμενο. Συνοπτικά, η ιδέα του αλγορίθμου είναι ότι ευθυγραμμίζουμε τη συμβολοσειρά που αναζητάμε με την αρχή του κειμένου και εξετάζουμε πρώτα τον χαρακτήρα στην *τελευταία* θέση της συμβολοσειράς. Αν είναι ίδιοι οι χαρακτήρες συγκρίνουμε τους προηγούμενους και συνεχίζουμε με παρόμοιο τρόπο μέχρι να βρούμε κάποια διαφορά. Όταν βρεθεί κάποια διαφορά μετατοπίζουμε κατάλληλο αριθμό θέσεων (από 1 θέση μέχρι όσες θέσεις είναι το μήκος της ζητούμενης συμβολοσειράς) τη συμβολοσειρά μας προς τα δεξιά και επαναλαμβάνουμε τη διαδικασία.

Το σημαντικότερο προτέρημα του αλγορίθμου Boyer-Moore είναι η ταχύτητά του, καθώς σε μία τυπική περίπτωση χρειάζεται χρόνο της τάξης $\mathbf{O}(n / m)$, όπου n είναι το μήκος του κειμένου εισόδου και m το μήκος της ζητούμενης συμβολοσειράς. Προφανώς, όσο μεγαλύτερη σε μήκος είναι η ζητούμενη συμβολοσειρά τόσο γενικά ταχύτερος είναι ο αλγόριθμος. Ένας άλλος παράγοντας που επηρεάζει θετικά τον αλγόριθμο είναι το μέγεθος του αλφάβητου. Άλλα πλεονεκτήματα είναι η σχετική ευκολία υλοποίησης και οι μικρές απαιτήσεις σε χώρο.

Το μεγάλο, όμως, μειονέκτημα του Boyer-Moore είναι πως είναι ιδανικός μόνο όταν αναζητάμε μία συμβολοσειρά: σε περίπτωση που θέλουμε να ψάξουμε για x διαφορετικές συμβολοσειρές πρέπει να εκτελέσουμε τον αλγόριθμο x φορές. Επίσης, ο

Boyer-Moore στη χειρότερη περίπτωση χρειάζεται 3*n* συγκρίσεις για να επεξεργαστεί ένα κείμενο μήκους *n* χαρακτήρων.

3.2.1 Μία ανεπίσημη περιγραφή.

Ας υποθέσουμε αρχικά πως έχουμε μια συμβολοσειρά με το όνομα *pat* μήκους *patlen* χαρακτήρων, η οποία αποτελεί τη συμβολοσειρά που σκοπεύουμε να αναζητήσουμε μέσα σε κάποια μεγαλύτερη συμβολοσειρά με το όνομα *string*.

Η βασική ιδέα στην οποία στηρίζεται ο αλγόριθμος είναι ότι μπορούμε να αποκτήσουμε πιο σημαντική πληροφορία όταν η ταυτοποίηση της ζητούμενης συμβολοσειράς ξεκινάει από το τέλος προς την αρχή παρά από την αρχή προς το τέλος. Ας φανταστούμε αρχικά πως η *pat* τοποθετείται ακριβώς από πάνω από την αρχή της *string*, δηλαδή στο αριστερό μέρος, με τέτοιο τρόπο ώστε οι πρώτοι χαρακτήρες των δύο συμβολοσειρών να είναι ευθυγραμμισμένοι μεταξύ τους. Εστιάζουμε την προσοχή μας στην πληροφορία που μπορούμε να πάρουμε αν εξετάσουμε τον χαρακτήρα *char* που βρίσκεται στη θέση *patlen* από την αρχή. Αυτός ο χαρακτήρας βρίσκεται σε αντίστοιχη θέση με τον τελευταίο χαρακτήρα της *pat*.

Παρατήρηση 1. Σε περίπτωση που γνωρίζουμε πως ο *char* δεν περιέχεται μέσα στην *pat* τότε δε χρειάζεται να εξετάσουμε την περίπτωση να υπάρχει κάποιο στιγμιότυπο της *pat* που να ξεκινάει στη θέση 1, 2, ..., *patlen* της συμβολοσειράς *string*: για ένα τέτοιο στιγμιότυπο θα ήταν απαραίτητο ο *char* να εμφανίζεται σε κάποιο σημείο της συμβολοσειράς *pat*.

Παρατήρηση 2. Γενικότερα, αν η τελευταία (δεξιότερη) εμφάνιση του χαρακτήρα *char* μέσα στην *pat* γίνεται στη θέση $delta_1$, τότε γνωρίζουμε ότι μπορούμε να ολισθήσουμε την *pat* κατά $delta_1$ θέσεις προς τα δεξιά χωρίς να κάνουμε άλλες συγκρίσεις. Ο λόγος για την ολίσθηση αυτή είναι πως αν μετακινούσαμε την pat κατά λιγότερες θέσεις από $delta_1$ τότε η εμφάνιση του χαρακτήρα *char* μέσα στη *string* θα συνέπιπτε με κάποιον χαρακτήρα της *pat* με τον οποίο δεν υπάρχει περίπτωση να ταυτίζονται. Μία τέτοια ταύτιση θα απαιτούσε να βρίσκεται ένα στιγμιότυπο του *char*

μέσα στην *pat* σε δεξιότερη θέση από αυτήν που έχει βρεθεί ότι είναι η πιο δεξιά θέση εμφάνισής του.

Επομένως σε περίπτωση που ο *char* δεν ταιριάζει με τον τελευταίο χαρακτήρα της *pat* μπορούμε να προσπεράσουμε τους επόμενους $delta_1$ χαρακτήρες της συμβολοσειράς *string* χωρίς να τους εξετάσουμε. Η $delta_1$ είναι συνάρτηση με όρισμα τον χαρακτήρα *char* που εξετάζουμε από την *string*. Αν ο *char* δεν εμφανίζεται μέσα στην *pat* τότε η τιμή της $delta_1$ είναι ίση με patlen. Σε διαφορετική περίπτωση, αν δηλαδή ο *char* εμφανίζεται μέσα στη συμβολοσειρά *pat*, η τιμή της $delta_1$ είναι ίση με τη διαφορά του μήκους *patlen* μείον τη θέση της πιο δεξιάς εμφάνισης του *char* μέσα στην *pat*.

Ας υποθέσουμε ότι ο *char* ταιριάζει με τον τελευταίο χαρακτήρα της *pat*. Σε αυτή την περίπτωση πρέπει να διαπιστώσουμε αν ο προηγούμενος χαρακτήρας της συμβολοσειράς *string* ταιριάζει με τον προτελευταίο χαρακτήρα της *pat*. Εάν ισχύει και αυτό συνεχίζουμε να πηγαίνουμε προς τα πίσω και να συγκρίνουμε τα ζεύγη των χαρακτήρων μέχρι να έχουμε ταιριάξει όλους τους χαρακτήρες της *pat* με αντίστοιχους του κειμένου (οπότε και έχουμε βρει με επιτυχία ένα στιγμιότυπο της *pat*) ή μέχρι να βρούμε, μετά από επιτυχείς συγκρίσεις των τελευταίων *m* χαρακτήρων, έναν άλλο χαρακτήρα *char* της συμβολοσειράς *string* που δεν ταιριάζει με τον αντίστοιχο χαρακτήρα της *pat*.

Στη δεύτερη περίπτωση θέλουμε να μετατοπίσουμε την *pat* προς τα δεξιά και να εξετάσουμε την επόμενη πιθανή θέση ταύτισης. Φυσικά θέλουμε η μετατόπιση αυτή να είναι όσο το δυνατόν μεγαλύτερη.

Παρατήρηση 3(α). Μπορούμε να εφαρμόσουμε τη λογική που χρησιμοποιήθηκε παραπάνω – στηριζόμενοι στον χαρακτήρα *char* που δεν ταίριαξε και την $delta_1$ – για να ολισθήσουμε την *pat* κατά *k* θέσεις προς τα δεξιά ώστε να ευθυγραμμίσουμε τις δύο γνωστές εμφανίσεις του *char*. Στη συνέχεια θα εξετάσουμε τον χαρακτήρα της *string* που είναι ευθυγραμμισμένη με την τελευταία λέξη της συμβολοσειράς *pat*. Συνεπώς θα μετατοπίσουμε την προσοχή μας προς τα δεξιά κατά $k + m$ θέσεις. Η απόσταση *k* που μπορούμε να μετακινήσουμε την *pat* εξαρτάται από τη θέση που εμφανίζεται ο χαρακτήρας *char* μέσα στην *pat*. Εάν η δεξιότερη εμφάνιση του *char* μέσα στην *pat* είναι πιο δεξιά από τον χαρακτήρα στον οποίο απέτυχε η σύγκριση (δηλαδή μέσα στο κομμάτι

της *pat* το οποίο έχουμε ήδη εξετάσει επιτυχώς) θα έπρεπε να μετακινήσουμε την *pat* προς τα *αριστερά* ώστε να ευθυγραμμίσουμε τις δύο γνωστές εμφανίσεις του *char*. Φυσικά κάτι τέτοιο δεν είναι επιθυμητό. Στην περίπτωση αυτή θεωρούμε τη συνάρτηση $delta_1$ άχρηστη και μετακινούμαστε κατά μία θέση δεξιά, δηλαδή με $k = 1$ (το οποίο είναι πάντα ορθό). Αυτή η μετακίνηση στρέφει την προσοχή μας κατά $m + 1$ θέσεις δεξιά στη *string*. Σε περίπτωση που η δεξιότερη εμφάνιση του χαρακτήρα *char* στη συμβολοσειρά *pat* είναι πιο αριστερά από τη θέση στην οποία απέτυχε η σύγκριση των χαρακτήρων, μπορούμε να μετακινήσουμε την *pat* προς τα δεξιά κατά $k = delta_1(char) - m$ έτσι ώστε να ευθυγραμμίσουμε τις δύο εμφανίσεις του *char*. Η ενέργεια αυτή μετατοπίζει το ενδιαφέρον μας μέσα στη string κατά $delta_1(char) - m + m = delta_1(char)$ θέσεις προς τα δεξιά. Παρόλα αυτά, είναι πιθανό να μπορούμε να επιτύχουμε καλύτερο αποτέλεσμα

Παρατήρηση 3(β). Γνωρίζουμε πως οι επόμενοι m χαρακτήρες της *string* ταιριάζουν με τους *m* τελευταίους χαρακτήρες της συμβολοσειράς *pat*. Έστω ότι αυτή η υποσυμβολοσειρά της *pat* συμβολίζεται με *subpat*. Γνωρίζουμε επιπλέον ότι η συγκεκριμένη εμφάνιση της *subpat* μέσα στην *string* προηγείται από έναν χαρακτήρα, τον *char*, ο οποίος διαφέρει από τον χαρακτήρα που προηγείται της καταληκτικής εμφάνισης της *subpat*. Σε γενικές γραμμές, το σκεπτικό που χρησιμοποιήθηκε παραπάνω μπορεί να επεκταθεί ώστε να μετατοπίσουμε την *pat* προς τα δεξιά κατά ένα συγκεκριμένο αριθμό θέσεων με σκοπό την ευθυγράμμιση του στιγμιότυπου της συμβολοσειράς *subpat* που εντοπίσαμε στη string με τη δεξιότερη εμφάνιση της *subpat* μέσα στην *pat*, η οποία όμως δεν ακολουθεί χαρακτήρα ίδιο με εκείνον που προηγείται της καταληκτικής εμφάνισης της *subpat* μέσα στην *pat*. Μία τέτοια επανεμφάνιση της subpat μέσα στην pat ονομάζεται «λογική επανεμφάνιση» (plausible reoccurrence). Ο λόγος που χρησιμοποιήθηκε η φράση «σε γενικές γραμμές» προηγουμένως είναι ότι πρέπει να επιτρέπουμε τη δεξιότερη λογική επανεμφάνιση της συμβολοσειράς *subpat* να «πέφτει έξω» από την αριστερή άκρη της *pat*. Το σημείο αυτό διασαφηνίζεται αργότερα.

Επομένως, σύμφωνα με την Παρατήρηση 3(β), εάν έχουμε ταιριάξει τους τελευταίους *m* χαρακτήρες της *pat* προτού αποτύχει μία σύγκριση, μπορούμε να μετατοπίσουμε την *pat* κατά *k* θέσεις προς τα δεξιά, όπου το *k* εξαρτάται από τη θέση μέσα στη *pat* της δεξιότερης λογικής επανεμφάνισης της κατάληξης μήκους *m*. Ύστερα από την μετατόπιση κατά *k*, θέλουμε να επιθεωρήσουμε τον χαρακτήρα της

συμβολοσειράς *string* που είναι ευθυγραμμισμένος με τον τελευταίο χαρακτήρα της *pat*. Συνεπώς, στην πραγματικότητα μετατοπίζουμε προς τα δεξιά την προσοχή μας μέσα στη string κατά k + m θέσεις. Αυτή την απόσταση την ονομάζουμε $delta_2$ κα καθορίζεται σα συνάρτηση της θέσης *j* μέσα στη συμβολοσειρά *pat* στην οποία θέση απέτυχε η σύγκριση. Το *k* είναι απλά η απόσταση ανάμεσα στην καταληκτική εμφάνιση της συμβολοσειράς *subpat* και στη δεξιότερη λογική επανεμφάνιση και είναι πάντα μεγαλύτερη ή ίση με 1. Το *m* ισούται απλά με $patlen - j$.

Σε περίπτωση που έχουμε συγκρίνει επιτυχώς τους τελευταίους m χαρακτήρες της συμβολοσειράς pat πριν αποτύχει μία σύγκριση, είναι ξεκάθαρο πως επιθυμούμε να μετατοπίσουμε την προσοχή μας προς τα δεξιά μέσα στη συμβολοσειρά string κατά $m + 1$ ή κατά $delta_1(char)$ ή κατά $delta_2(j)$, ανάλογα με το ποια από τις τρεις τιμές μας δίνει τη μεγαλύτερη μετατόπιση. Από τον ορισμό της $delta_2$ ως $k + m$, με το *k* να είναι πάντα μεγαλύτερο ή ίσο του 1, είναι προφανές ότι η $delta_2$ είναι τουλάχιστον ίση με την τιμή $1 + m$. Επομένως μπορούμε απλά να επιλέξουμε την απόσταση που θα μετατοπίσουμε την προσοχή μας μέσα στη συμβολοσειρά *string* από το μέγιστο των δύο συναρτήσεων *delta*. Ο κανόνας αυτός ισχύει ακόμα και στην περίπτωση που $m = 0$, δηλαδή όταν αποτύχει η σύγκριση του τελευταίου χαρακτήρα της *pat* με τον αντίστοιχο χαρακτήρα της *string*, καθώς στην περίπτωση αυτή $j = patlen$ και $delta_2(j) \geq 1$.

3.2.2 Ένα παράδειγμα.

Στο παράδειγμα που ακολουθεί [2] θα χρησιμοποιούμε ένα '↑' κάτω από μία θέση της συμβολοσειράς *string* ώστε να υποδεικνύουμε τον χαρακτήρα *char*. Όταν αυτός ο "δείκτης" μετατοπίζεται προς τα δεξιά θεωρούμε ότι παρασύρει μαζί του τη δεξιά άκρη της *pat*. Όποτε ο δείκτης μετακινείται προς τα αριστερά η *pat* παραμένει σταθερή ως προς την *string*.

```
pat:        AT-THAT
string: ... WHICH-FINALLY-HALTS.--AT-THAT-POINT . . .
                      ↑
```

Από τη στιγμή που ο χαρακτήρας ‘F’ δεν εμφανίζεται πουθενά μέσα στη συμβολοσειρά *pat*, μπορούμε να αξιοποιήσουμε την παρατήρηση 1 και να μετακινήσουμε τον δείκτη 7 θέσεις προς τα δεξιά. Αφού ο δείκτης μετακινήθηκε προς τα δεξιά, μετακινούμε και την *pat*.

pat: AT-THAT
string: ... WHICH-FINALLY-HALTS.--AT-THAT-POINT . . .
↑

Με τη βοήθεια της παρατήρησης 2 μπορούμε να μετακινήσουμε το δείκτη δεξιά κατά 4 θέσεις ώστε να ευθυγραμμιστούν οι δύο παύλες.

pat: AT-THAT
string: ... WHICH-FINALLY-HALTS.--AT-THAT-POINT . . .
↑

Αυτή τη φορά ο *char* ταυτίζεται με τον τελευτάιο χαρακτήρα της συμβολοσειράς *pat*. Επομένως κάνουμε ένα βήμα προς τα αριστερά.

pat: AT-THAT
string: ... WHICH-FINALLY-HALTS.--AT-THAT-POINT . . .
↑

Χρησιμοποιώντας την παρατήρηση 3(α) μπορούμε να ολισθήσουμε το δείκτη κατά 7 θέσεις προς τα δεξιά επειδή η L δεν εμφανίζεται σε κανένα σημείο της *pat*. Να σημειωθεί ότι αυτή η μετατόπιση του δείκτη μετακινεί την *pat* κατά 6 θέσεις. Στο σημείο αυτό αξίζει να αναφέρουμε ότι η συνάρτηση $delta_2$ σε αυτή την περίπτωση θα μας επέτρεπε να μετακινήσουμε το δείκτη δεξιά κατά 4, ώστε να ευθυγραμμιστεί η συμβολοσειρά “T” που εξετάσαμε επιτυχώς με το “T” στην αρχή της λέξης “THAT” στη συμβολοσειρά *pat*. Όμως, όπως αναφέρθηκε προηγουμένως, κάθε φορά επιλέγουμε τη μεγαλύτερη μετατόπιση.

pat: AT-THAT
string: ... WHICH-FINALLY-HALTS.--AT-THAT-POINT . . .

↑

Και πάλι ο *char* ταιριάζει με τον τελευταίο χαρακτήρα της *pat*. Κάνοντας ένα βήμα αριστερά βλέπουμε ότι ο προτελευταίος χαρακτήρας της *pat* ταιριάζει με τον αντίστοιχο της *string*. Μετά από ακόμη ένα βήμα αριστερά έχουμε:

pat: AT-THAT

string: ... WHICH-FINALLY-HALTS.--AT-THAT-POINT . . .

↑

Σε αυτή τη σύγκριση υπάρχει διαφορά ανάμεσα στους δύο χαρακτήρες, επομένως καταφεύγουμε στην Παρατήρηση 3(β). Η $delta_2$ μας συμφέρει σε αυτή την περίπτωση καθώς μας επιτρέπει να μετακινήσουμε το δείκτη δεξιά κατά 7 θέσεις, ώστε να ευθυγραμμίσουμε την ακολουθία "AT" που βρέθηκε μέσα στη συμβολοσειρά *string* με τη λέξη "AT" στην αρχή της *pat*. Για την περίπτωση αυτή η $delta_1$ έδινε αποτέλεσμα 4.

pat: AT-THAT

string: ... WHICH-FINALLY-HALTS.--AT-THAT-POINT . . .

↑

Αυτή τη φορά διαπιστώνουμε πως κάθε χαρακτήρας της συμβολοσειράς pat ταιριάζει με τον αντίστοιχο χαρακτήρα στην *string*, άρα ανακαλύψαμε τη ζητούμενη συμβολοσειρά. Αξίζει να αναφερθεί ότι έγιναν μόνο 14 συγκρίσεις με περιεχόμενα της συμβολοσειράς *string*. Οι επτά από αυτές ήταν απαραίτητες ώστε να επιβεβαιώσουμε ότι ανακαλύφθηκε η ζητούμενη λέξη. Οι άλλες επτά μας επέτρεψαν να προσπεράσουμε τους πρώτους 22 χαρακτήρες της συμβολοσειράς *string*.

3.2.3 Ο αλγόριθμος.

Στο σημείο αυτό θα παρουσιαστεί ο αλγόριθμος με λεπτομερή μορφή. Παρακάτω χρησιμοποιείται ο συμβολισμός *pat*(*j*) όταν αναφερόμαστε στον *j*-οστό χαρακτήρα της συμβολοσειράς *pat* . Η μέτρηση ξεκινάει από τα αριστερά, δηλαδή *pat*(1) είναι ο πρώτος χαρακτήρας και *pat*(*patlen*) ο τελευταίος.

Υποθέτουμε πως οι τιμές για τις δύο συναρτήσεις $delta_1$ και $delta_2$ βρίσκονται αποθηκευμένες σε δύο πίνακες. Ο πίνακας για την $delta_1$ περιέχει τόσα στοιχεία όσα και τα διαφορετικά σύμβολα του αλφάβητου που χρησιμοποιείται. Η τιμή της συνάρτησης για έναν συγκεκριμένο χαρακτήρα *char* θα συμβολίζεται με $delta_1(char)$. Ο δεύτερος πίνακας περιέχει αριθμό στοιχείων ίσο με το μήκος της pat. Το *j*-οστό στοιχείο του πίνακα συμβολίζεται με $delta_2(j)$. Τα στοιχεία και των δύο πινάκων περιέχουν μη-αρνητικές ακέραιες τιμές.

Οι πίνακες αρχικοποιούνται ύστερα από προεπεξεργασία της συμβολοσειράς *pat*. Τα ακριβή δεδομένα για κάθε στοιχείο θα προσδιοριστούν αργότερα, όταν γίνει ξεκάθαρος ο τρόπος με τον οποίο θα χρησιμοποιούνται.

Ο αλγόριθμος αναζήτησης μπορεί να περιγραφεί με τον ακόλουθο ψευδοκώδικα:

Αλγόριθμος ΒΜ: Αναζήτηση συμβολοσειράς.

Είσοδος: Συμβολοσειρά προς αναζήτηση *pat*, κείμενο *string* μέσα στο οποίο γίνεται η αναζήτηση και δύο πίνακες $delta_1$ και $delta_2$.

Έξοδος: Θέση μέσα στο *string* από όπου βρέθηκε ότι ξεκινάει στιγμιότυπο της *pat*.

```
begin
        stringlen ← length of string.
        i ← patlen.
top:    if i > stringlen then return false.
        j ← patlen.
loop:   if j = 0 then return j + 1.
        if string(i) = pat(j) then
        begin
                j ← j - 1 .
                i ← i - 1 .
                goto loop.
        end
        i ← i + max(delta1 ( string(i) ), delta2 (j )).
        goto top.
end.
```

Αν ο παραπάνω αλγόριθμος μας επιστρέψει αρνητική απάντηση (false), τότε η συμβολοσειρά *pat* δεν εμφανίζεται μέσα στην *string*. Σε περίπτωση που ο αλγόριθμος

μας επιστρέψει έναν αριθμό, ο αριθμός αυτός αντιπροσωπεύει τη θέση μέσα στη *string* από την οποία ξεκινάει το στιγμιότυπο της *pat*.

Ο πίνακας $delta_1$ διαθέτει ένα στοιχείο για κάθε χαρακτήρα του αλφάβητου. Ο πίνακας $delta_1$ ορίζεται ως:

$delta_1(char)$ = Εάν ο char δεν εμφανίζεται μέσα στην pat, το αποτέλεσμα είναι ίσο με το μήκος *patlen*, διαφορετικά είναι ίσο με *patlen* - *j*, όπου *j* ο μεγαλύτερος ακέραιος για τον οποίο ισχύει *pat*(*j*)=*char*.

Ο πίνακας $delta_2$ έχει μία τιμή για κάθε έναν από τους ακεραίους από 1 μέχρι *patlen*. Σε γενικές γραμμές, η τιμή $delta_2(j)$ μας δείχνει (α) πόσες θέσεις μπορούμε να ολισθήσουμε τη συμβολοσειρά *pat* προς τα δεξιά έτσι ώστε να την ευθυγραμμίσουμε με την ανακαλυφθείσα εμφάνιση (μέσα στη συμβολοσειρά *string*) των τελευταίων *patlen* – *j* χαρακτήρων της *pat* με την δεξιότερη λογική επανεμφάνιση συν (β) την επιπλέον απόσταση που πρέπει να ολισθήσουμε τον "δείκτη" προς τα δεξιά ώστε να επαναληφθεί η διαδικασία στο δεξιό άκρο της *pat*. Για να καθορίσουμε συγκεκριμένα την $delta_2$ πρέπει να καθορίσουμε τη δεξιότερη λογική επανεμφάνιση της υπολέξης κατάληξης της *pat*. Προς αυτό το στόχο κάνουμε τις εξής παραδοχές: έστω ότι το $ είναι ένας χαρακτήρας που δεν εμφανίζεται μέσα στην *pat* και ας δεχτούμε πως αν το i είναι μικρότερο του 1 τότε *pat*(*i*) = $. Επιπλέον, θεωρούμε πως δύο ακολουθίες χαρακτήρων $[c_1c_2\ldots c_n]$ και $[d_1d_2\ldots d_n]$ είναι όμοιες αν για όλα τα i από 1 μέχρι n ισχύει είτε $c_i = d_i$, είτε c_i = $ είτε d_i = $.

Τέλος, ορίζουμε πως η θέση *rpr*(*j*) της δεξιότερης λογικής επανεμφάνισης της υπολέξης κατάληξης η οποία ξεκινάει στη θέση *j* + 1, για όλα τα *j* από 1 μέχρι *patlen*, είναι ίση με τη μεγαλύτερη τιμή *k*, που όμως είναι μικρότερη ή ίση με *patlen*, τέτοια ώστε οι ακολουθίες [*pat*(*j* + 1)*pat*(*j* + 2)…*pat*(*patlen*)] και [*pat*(*k*)*pat*(*k* + 1)…*pat*(*patlen* - *j* - 1] να είναι όμοιες ενώ παράλληλα ισχύει είτε ότι $k \leq 1$ είτε ότι $pat(k - 1) \leq pat(j)$. Δηλαδή, η θέση της δεξιότερης λογικής επανεμφάνισης της συμβολοσειράς *subpat*, η οποία ξεκινά στη θέση *j* + 1 της *pat*, είναι η δεξιότερη θέση στην οποία εμφανίζεται η *subpat* μέσα στην pat χωρίς όμως να υπάρχει ακριβώς πριν την αρχή της ο χαρακτήρας *pat*(*j*), ο οποίος βρίσκεται αμέσως πριν την αρχή της καταληκτικής *subpat*. Παράλληλα,

επιτρέπεται ο προηγούμενος χαρακτήρας ή ένα μέρος της επανεμφάνισης να βρίσκεται εκτός του αριστερού ορίου της *pat*. Να σημειωθεί ότι η rpr(j) μπορεί να είναι αρνητική λόγω αυτού του γεγονότος.

Επομένως, η απόσταση κατά την οποία πρέπει να μετατοπίσουμε την *pat* ώστε να ευθυγραμμίσουμε την ανακαλυφθείσα συμβολοσειρά που ξεκινάει στην θέση $j + 1$ με την δεξιότερη λογική επανεμφάνισή της είναι $j + 1 - rpr(j)$. Η απόσταση μέχρι το τέλος της pat είναι απλά $patlen - j$. Η τιμή $delta_2(j)$ είναι απλά το άθροισμα των δύο. Άρα, η συνάρτηση delta_2 ορίζεται από τον ακόλουθο τύπο:

$$delta_2(\text{j}) = patlen + 1 - rpr(j)$$

Για να ξεκαθαρίσουμε τον ορισμό αυτό ας εξετάσουμε τα δύο παρακάτω παραδείγματα.

j:	1	2	3	4	5	6	7	8	9
pat:	A	B	C	X	X	X	A	B	C
$delta_2(j)$:	14	13	12	11	10	9	11	10	1

j:	1	2	3	4	5	6	7	8	9
pat:	A	B	Y	X	C	D	E	Y	X
$delta_2(j)$:	17	16	15	14	13	12	7	10	1

3.2.4 Θεωρητική ανάλυση.

Η προεπεξεργασία για τη συνάρτηση $delta_1$ χρειάζεται ένα πίνακα μεγέθους ίσου με τον αριθμό των στοιχείων που υπάρχουν στη χρησιμοποιούμενο αλφάβητο. Η υλοποίησή μας πρώτα αρχικοποιεί όλα τα στοιχεία του πίνακα σε τιμή ίση με *patlen* και στη συνέχεια ενημερώνει τις κατάλληλες θέσεις του $delta_1$ εκτελώντας ένα σειριακό πέρασμα κατά μήκος της συμβολοσειρά *pat*. Επομένως το κόστος της προεπεξεργασίας για την $delta_1$ είναι γραμμικό ως προς το μέγεθος του αλφάβητου συν το μήκος της *pat*.

Για την υλοποίηση του αλγορίθμου σε περιπτώσεις με πολύ μεγάλα αλφάβητα, θα μπορούσε κανείς να υλοποιήσει τον πίνακα $delta_1$ σαν ένα hash array. Στη χειρότερη περίπτωση, η πρόσβαση στον $delta_1$ κατά τη διάρκεια της αναζήτησης θα στοίχιζε αριθμό εντολών της τάξης του μήκους *patlen*, δημιουργώντας σημαντική καθυστέρηση. Επομένως, ο αλγόριθμος με την τρέχουσα μορφή σίγουρα δεν εκτελείται σε γραμμικό ως προς $i + patlen$ χρόνο για αλφάβητα άπειρων στοιχείων.

Θεωρητική ανάλυση του αλγορίθμου έχει δείξει ότι εξακολουθεί να τρέχει σε γραμμικό χρόνο ακόμα και αν παραλειφθεί η $delta_1$ και το αποτέλεσμα αυτό ισχύει όταν το αλφάβητο είναι άπειρο. Όμως μια τέτοια παράληψη έχει σημαντικό αρνητικό αντίκτυπο στην απόδοση του αλγορίθμου. Ο χρόνος που απαιτείται για τη δημιουργία του πίνακα $delta_2$ είναι γραμμικός ως προς το μήκος *patlen*.

Ο αλγόριθμος έχει πολύ καλή απόδοση στη γενική περίπτωση. Παρόλα αυτά, η συμπεριφορά του αλγορίθμου στη χειρότερη περίπτωση αξίζει να μελετηθεί. Έχει αποδειχθεί ότι η εκτέλεση του αλγορίθμου (μετά τη φάση της προεπεξεργασίας) απαιτεί χρόνο γραμμικό ως προς $i + patlen$, με την προϋπόθεση ότι υπάρχει διαθέσιμος χώρος που είναι γραμμική συνάρτηση του μεγέθους του αλφάβητου και του *patlen*. Συγκεκριμένα, στην περίπτωση που θέλουμε να ανακαλύψουμε ότι η συμβολοσειρά *pat* δεν εμφανίζεται στους πρώτους n χαρακτήρες τότε χρειάζεται να εξετάσουμε το πολύ $3*n$ χαρακτήρες. Η γραμμικότητα του αλγορίθμου είναι εγγυημένη από τη συνάρτηση $delta_2$.

Στο σημείο αυτό θα αναλυθεί η συμπεριφορά του αλγορίθμου στη μέση περίπτωση με τη χρήση ενός στοχαστικού μοντέλου για την απόδοσή του. Όπως θα φανεί αργότερα στην υποενότητα αυτή, η θεωρητική ανάλυση υποδεικνύει πως ο αλγόριθμος συνήθως είναι «υπογραμμικός» (sublinear) τόσο σχετικά με τον αριθμό των αναφορών που γίνονται στη συμβολοσειρά εισόδου *string* όσο και σχετικά με τον αριθμό των εντολών που εκτελούνται (για τη συγκεκριμένη υλοποίηση).

Η ανάλυση που ακολουθεί στηρίζεται στην εξής απλοποιητική υπόθεση: Κάθε χαρακτήρας των συμβολοσειρών *pat* και *string* είναι μία ανεξάρτητη τυχαία μεταβλητή. Η πιθανότητα ένας χαρακτήρας της *pat* ή της string να είναι ίδιος με οποιοδήποτε χαρακτήρα του αλφάβητου είναι p.

Ας υποθέσουμε ότι μόλις έχουμε ολισθήσει την pat ορισμένες θέσεις προς τα δεξιά και ότι η σύγκριση του τελευταίου χαρακτήρα της pat με τον αντίστοιχο της string μας έδωσε αρνητικό αποτέλεσμα, δηλαδή οι χαρακτήρες είναι διαφορετικοί. Επιθυμούμε να βρούμε την αναμενόμενη τιμή της αναλογίας ανάμεσα στο κόστος για να ανακαλύψουμε μια διαφορά ανάμεσα σε δύο χαρακτήρες και στην αποστάση κατά την οποία θα μετακινήσουμε την *pat* μόλις ανακαλύψουμε τη διαφορά. Αν ορίσουμε το κόστος να είναι ίσο με τον αριθμό των αναφορών που γίνονται στη συμβολοσειρά *string* μέχρι να ανακαλύψουμε τους διαφορετικούς χαρακτήρες, μπορούμε να υπολογίσουμε την αναμενόμενη τιμή για το μέσο αριθμό αναφορών στη *string* ανά χαρακτήρα που περνάμε. Σε περίπτωση που ορίσουμε το κόστος σαν το συνολικό αριθμό εντολών που εκτελούνται όταν ανακαλύπτεται ένα ζεύγος διαφορετικών χαρακτήρων, μπορούμε να υπολογίσουμε την αναμενόμενη τιμή για τον αριθμό των εντολών που εκτελούνται ανά χαρακτήρα που περνάμε.

Στη συνέχεια θα χρησιμοποιείται η φράση «μόνο οι m τελευταίοι χαρακτήρες της *pat* ταιριάζουν» όταν θέλουμε να εκφράσουμε ότι «οι τελευταίοι m χαρακτήρες της pat ταιριάζουν με τους αντίστοιχους *m* χαρακτήρες της *string* αλλά ο (*m* – 1)-οστός χαρακτήρας από το τέλος της *pat* δεν ταιριάζει με τον αντίστοιχο χαρακτήρα της *string*».

Η αναμενόμενη τιμή για τον λόγο του κόστους ως προς τον αριθμό των χαρακτήρων που περνάμε δίνεται από την έκφρασή:

$$\left(\sum_{m=0}^{patlen-1} cost(m) * prob(m)\right) \Big/ \left(\sum_{m=0}^{patlen-1} prob(m) * \left(\sum_{k=1}^{patlen} skip(m,k) * k\right)\right)$$

όπου *cost*(*m*) είναι το κόστος που σχετίζεται με το να βρούμε ότι μόνο οι τελευταίοι *m* χαρακτήρες της *pat* ταιριάζουν, *prob*(*m*) είναι η πιθανότητα να ταιριάζουν μόνο οι τελευταίοι m χαρακτήρες της pat και *skip*(*m*, *k*) είναι η πιθανότητα να μετακινηθούμε προς τα δεξιά κατά *k* θέσεις στην περίπτωση που ταιριάζουν μόνο οι τελευταίοι *m* χαρακτήρες της *pat*.

Σύμφωνα με τις υποθέσεις που έχουμε κάνει, η πιθανότητα να ταιριάζουν μόνο οι τελευταίοι m χαρακτήρες της *pat* είναι:

$$prob(m) = p^m(1-p)/(1-p^{patlen})$$

Ο παρανομαστής οφείλεται στην υπόθεση ότι υπάρχει διαφορά ανάμεσα στους χαρακτήρες.

Η πιθανότητα να μετακινήσουμε την *pat* κατά *k* θέσεις μπορεί να καθοριστεί αναλύοντας τον τρόπο με τον οποίο αυξάνεται το *i*. Όμως, πρέπει να σημειωθεί ότι παρόλο που αυξάνουμε το *i* κατά τη μεγαλύτερη τιμή που παίρνουμε από τις δύο συναρτήσεις *delta*, η πραγματική μετατόπιση της *pat* γίνεται κατά $max - m$ θέσεις καθώς η αύξηση του *i* περιλαμβάνει επίσης και τον αριθμό *m* μέχρι να φτάσουμε στο τέλος της συμβολοσειράς *pat*. Επομένως, όταν αναλύουμε τη συνεισφορά των δύο συναρτήσεων *delta* μας αφορά ο αριθμός των θέσεων που μας επιτρέπουν να μετατοπίσουμε την *pat* και όχι ο αριθμός με τον οποίο αυξάνουμε το *i*. Τέλος, να θυμίσουμε ότι στην περίπτωση που ο διαφορετικός χαρακτήρας *char* της συμβολοσειράς *string* εμφανίζεται μέσα στους m χαρακτήρες της *pat* που ήδη έχουν ταιριάξει, τότε η $delta_1$ μας είναι άχρηστη και πάντα μετατοπίζουμε την *pat* κατά $delta_2$ θέσεις. Η πιθανότητα ώστε να μην μας προσφέρει χρήσιμη πληροφορία η $delta_1$ είναι μόνο $(1 - (1 - p)^m)$. Την πιθανότητα αυτή θα την αποκαλούμε $probdelta_1worthless(m)$.

Οι προϋποθέσεις υπό τις οποίες η $delta_1$ θα μας επιτρέψει φυσιολογικά να μετακινηθούμε προς τα δεξιά κατά *k* μπορούν να κατηγοριοποιηθούν σε τέσσερις περιπτώσεις ως εξής: (α) η $delta_1$ μας επιτρέπει να μετακινηθούμε προς τα δεξιά κατά μία θέση αν ο *char* είναι ο $(m + 2)$-οστός χαρακτήρας από το τέλος της *pat* (ή σε περίπτωση που δεν υπάρχουν άλλοι χαρακτήρες στην *pat*) και ο char δεν εμφανίζεται στα δεξιά εκείνης της θέσης (γεγονός που ισχύει με πιθανότητα $(1 - p)^m$ * (εάν $m + 1 = patlen$ τότε 1, αλλιώς *p*). (β) Η $delta_1$ μας επιτρέπει να ολισθήσουμε δεξιά κατά *k*, όπου $1 < k < patlen - m$, με την προϋπόθεση ότι η δεξιότερη εμφάνιση του *char* στην pat γίνεται $m + k$ χαρακτήρες από το δεξί άκρο της *pat* (το οποίο ισχύει με πιθανότητα $p * (1- p)^{k+m-1}$). (γ) Σε περίπτωση που $patlen–m > 1$, η $delta_1$ επιτρέπει να ολισθήσουμε την *pat* κατά $patlen - m$ θέσεις προς τα δεξιά σε περίπτωση που ο char δεν εμφανίζεται καθόλου μέσα στην *pat* (το οποίο ισχύει με πιθανότητα $(1- p)^{patlen-1}$, δεδομένου ότι γνωρίζουμε πως ο *char* δεν είναι ο $(m+1)$-οστός χαρακτήρας από το τέλος της *pat*). (δ) Τέλος, η $delta_1$ δεν μας επιτρέπει ποτέ να ολισθήσουμε κατά περισσότερες από $patlen - m$ θέσεις (αφού η μέγιστη τιμή της $delta_1$ είναι *patlen*).

Επομένως μπορούμε να ορίσουμε την πιθανότητα $probdelta_1(m, k)$, δηλαδή την πιθανότητα να μετατοπιστούμε δεξιά κατά k θέσεις όταν μόνο οι τελευταίοι m χαρακτήρες της pat ταιριάζουν, ως εξής:

$probdelta_1(m, k)$ = εάν ($k = 1$) τότε $(1 - p)^m$ * (εάν $m + 1 = patlen$ τότε 1 αλλιώς p);
αλλιώς αν $i < k < patlen - m$ τότε p* $(1 - p)^{k+m-1}$;
αλλιώς αν $k = patlen - m$ τότε $(1 - p)^{patlen-1}$;
αλλιώς (δηλαδή αν $k > patlen - m$) 0.

Στο σημείο αυτό θα κάνουμε παρόμοια ανάλυση για τη συνάρτηση $delta_2$. Η $delta_2$ μας επιτρέπει να μετατοπιστούμε δεξιά κατά k σε περίπτωση που (α) η πράξη αυτή ευθυγραμμίζει την ανακαλυφθείσα ακολουθία στη συμβολοσειρά string των τελευταίων m χαρακτήρων της *pat* με μία λογική επανεμφάνιση των ίδιων m χαρακτήρων σε ένα άλλο σημείο της *pat*, και (β) καμία μικρότερη μετατόπιση δε δημιουργεί τέτοια ευθυγράμμιση. Η πιθανότητα probpr(m, k) ώστε η συμβολοσειρά κατάληξης της pat να έχει λογική επανεμφάνιση k θέσεις αριστερά από τον πρώτο της χαρακτήρα είναι:

$probpr(m,$ k) = εάν $m + k < patlen$ τότε $(1 - p) * p^m$
αλλιώς $p^{patlen-k}$

Μπορούμε τώρα να ορίσουμε την πιθανότητα $probdelta_2(m, k)$, η οποία αντιπροσωπεύει την πιθανότητα με την οποία η συνάρτηση $delta_2$ θα μας επιτρέψει να μετακινηθούμε κατά k θέσεις προς τα δεξιά δεδομένου ότι ταιριάξανε οι m τελευταίοι χαρακτήρες της pat, με τον εξής αναδρομικό τύπο:

$$probdelta_2(m, k) = probpr(m, k)\left(1 - \sum_{n=1}^{k-1} probdelta_2(m, n)\right).$$

Πάντα επιλέγουμε τη μεγαλύτερη τιμή ανάμεσα στις δύο συναρτήσεις *delta* για να μετακινηθούμε προς τα δεξιά (ελέγχοντας και την πιθανότητα η $delta_1$ να μη μας δώσει χρήσιμη πληροφορία). Σε περίπτωση που οι τιμές των δύο συναρτήσεων delta ήταν ανεξάρτητες, τότε η πιθανότητα να μετακινηθούμε δεξιά κατά k θέσεις θα ήταν

απλά το άθροισμα όλων των γινομένων δύο πιθανοτήτων, της πιθανότητας η μία συνάρτηση *delta* να μας επιτρέψει μετακίνηση κατά *k* θέσεις και της πιθανότητας η άλλη *delta* να μας επιτρέψει μετακίνηση μικρότερη ή ίση με *k* θέσεις.

Όμως οι πιθανότητες για τις δύο συναρτήσεις δεν είναι τελείως ανεξάρτητες. Συγκεκριμένα, ας εξετάσουμε την περίπτωση όπου η $delta_1$ δε μας δίνει χρήσιμη πληροφορία. Τότε, ο *char* που μόλις διαβάσαμε από τη συμβολοσειρά *string* εμφανίζεται σε κάποια από τις τελευταίες *m* θέσεις της *pat* και δεν ταιριάζει με τον (*m*+1)-οστό χαρακτήρα. Όμως, σε περίπτωση που η $delta_2$ μας δώσει αποτέλεσμα 1 προκύπτει ότι η ολίσθηση αυτών των *m* χαρακτήρων προς τα αριστερά κατά 1 δημιουργεί μία ταύτιση. Το γεγονός αυτό δημιουργεί την εντύπωση ότι οι τελευταίοι *m* χαρακτήρες της *pat* είναι ίδιοι με τον (*m*+1)-οστό χαρακτήρα από το τέλος. Το πρόβλημα είναι ότι εμείς γνωρίζουμε πως ο (*m*+1)-οστός χαρακτήρας είναι διαφορετικός από τον *char*. Έτσι προκύπτει ότι ο *char* δεν γίνεται να εμφανίζεται στους τελευταίους *m* χαρακτήρες της *pat*, καταρρίπτοντας με τον τρόπο αυτό την υπόθεσή μας ότι η $delta_1$ μας ήταν άχρηστη. Επομένως, σε περίπτωση που η $delta_1$ δε μας προσφέρει ωφέλιμη πληροφορία, η πιθανότητα ώστε η $delta_2$ να υποδείξει μια μετατόπιση κατά 1 είναι 0 και, κατά συνέπεια, η πιθανότητα ώστε η $delta_2$ να υποδείξει μία από τις μεγαλύτερες μετατοπίσεις αυξάνεται ανάλογα..

Η αλληλεπίδραση αυτή ανάμεσα στις δύο *delta* γίνεται επίσης αισθητή, αν και σε μικρότερο βαθμό, για τις επόμενες *m* δυνατές τιμές της $delta_2$ τις οποίες, όμως, δεν εξετάζουμε. Η επιλογή αυτή συνεπάγεται ότι τα αποτελέσματα της ανάλυσής μας θα είναι λίγο χειρότερα από τα αναμενόμενα καθώς επιτρέπουμε μικρότερες μετακινήσεις μέσω της $delta_2$ σε περιπτώσεις που στην πραγματικότητα θα είχαμε μεγαλύτερες τιμές.

Η πιθανότητα να μας δώσει η $delta_2$ μετακίνηση k θέσεων όταν έχουν ταιριάξει μόνο οι τελευταίοι m χαρακτήρες της pat και δεδομένου ότι η $delta_1$ δεν μας έδωσε χρήσιμη πληροφορία, είναι:

$$
\begin{aligned}
probdelta_2'(m,k) = \quad & \text{if } k = 1 \\
& \quad \text{then } 0 \\
& \quad \text{else} \\
& \quad probpr(m,k)\left(1 - \sum_{n=2}^{k-1} probdelta_2'(m,n)\right).
\end{aligned}
$$

Τέλος μπορούμε να ορίσουμε την πιθανότητα *skip*(*m*, *k*), δηλαδή την πιθανότητα μετακινηθούμε δεξιά κατά *k* όταν ταιριάζουν μόνο οι m τελευταίοι χαρακτήρες της *pat* ως εξής:

$$
\begin{aligned}
skip(m, k) = \ & \text{if } k = 1 \\
& \text{then } probdelta_1(m, 1) * probdelta_2(m, 1) \\
& \text{else } probdelta_1 worthless(m) * probdelta_2'(m, k) \\
& \quad + \sum_{n=1}^{k-1} probdelta_1(m, k) * probdelta_2(m, n) \\
& \quad + \sum_{n=1}^{k-1} probdelta_1(m, n) * probdelta_2(m, k) \\
& \quad + probdelta_1(m, k) * probdelta_2(m, k).
\end{aligned}
$$

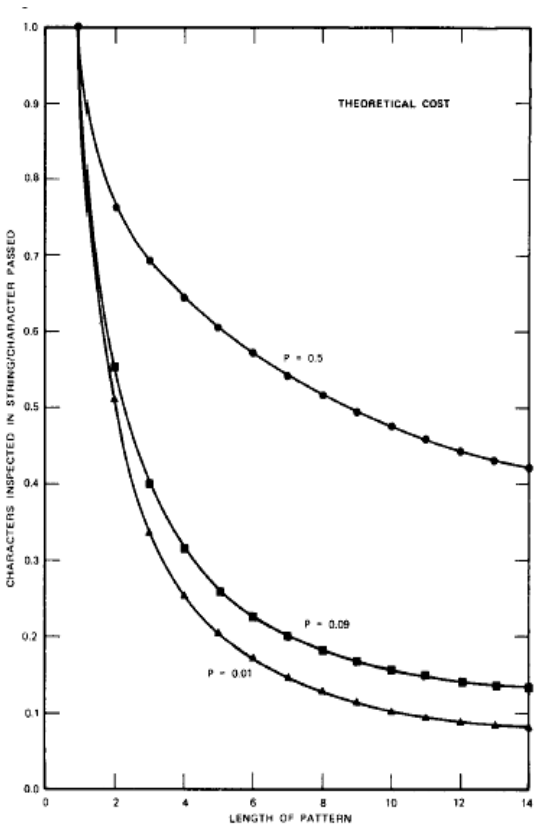
Στο σημείο αυτό θα εξετάσουμε τις δύο εναλλακτικές συναρτήσεις κόστους. Αν ενδιαφερόμαστε για τον αριθμό των αναφορών που έγιναν στη συμβολοσειρά string ανά χαρακτήρα που περνάμε, η συνάρτηση *cost*(*m*) θα είναι απλά ίση με *m*+1, τον αναγκαίο αριθμό αναφορών ώστε να επιβεβαιωθεί ότι μόνο οι m τελευταίοι της συμβολοσειράς *pat* ταιριάζουν.

Προκειμένου να αναλύσουμε τον αριθμό των εντολών που εκτελούνται ανά χαρακτήρα που περνάμε, η συνάρτηση cost(m) πρέπει να είναι ίση με τον αριθμό των εντολών που εκτελούνται στην περίπτωση που ταιριάζουν μόνο οι m τελευταίοι χαρακτήρες της *pat*. Για τη συγκεκριμένη υλοποίηση που έγινε προκύπτει ότι:

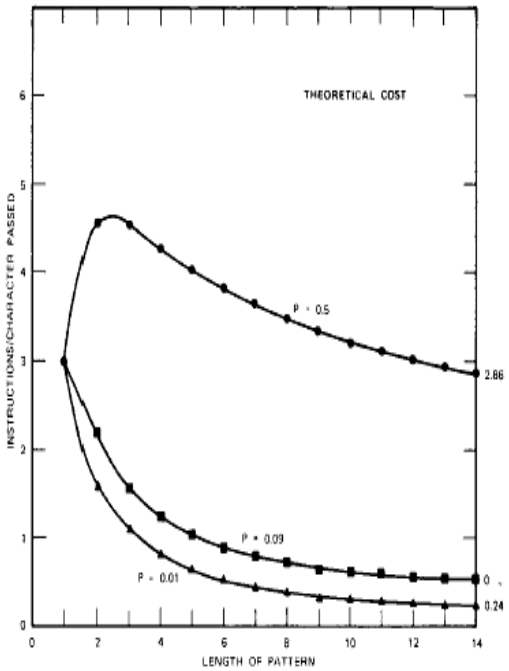
cost(*m*) = **if** *m* = 0 **then** 3 **else** 12 + 6 *m*.

Υπολογίσαμε την αναμενόμενη τιμή του κόστους ανά χαρακτήρα που περνάμε χρησιμοποιώντας τους παραπάνω τύπους (και για τους δύο ορισμούς του κόστους). Ο υπολογισμός έγινε για μήκος συμβολοσειράς προς αναζήτηση από 1 μέχρι 14 και για τιμές του p κατάλληλες για τρία διαφορετικά μεγέθη αλφάβητου: για μία τυχαία δυαδική συμβολοσειρά το *p* είναι 0.5, για μία αυθαίρετη συμβολοσειρά της Αγγλικής γλώσσας είναι περίπου 0.09, ενώ για ένα αλφάβητο με 100 διαφορετικά σύμβολα είναι ίσο με 0,01. Η τιμή του *p* για το Αγγλικό αλφάβητο καθορίστηκε ύστερα από στατιστική ανάλυση. Στις εικόνες *3γ* και *3δ* [2] φαίνονται οι γραφικές παραστάσεις για τις δύο συναρτήσεις κόστους ως συνάρτηση του μήκους της συμβολοσειράς. Σε κάθε μία από τις γραφικές η πάνω καμπύλη αντιστοιχεί στην τιμή p=0.5, δηλαδή στο δυαδικό αλφάβητο,

η μεσαία καμπύλη είναι για το αγγλικό αλφάβητο με p=0.09 και η χαμηλότερη καμπύλη αναπαριστά το αλφάβητο των 100 χαρακτήρων, για το οποίο p=0.01.



Εικόνα 3γ) Ο αριθμός των αναφορών στη συμβολοσειρά string ανά χαρακτήρα που περάσαμε [2]



Εικόνα 3δ) Αριθμός των εντολών που εκτελέστηκαν ανά χαρακτήρα που περάσαμε [2]

Στην *εικόνα 3γ* έχουμε τη γραφική παράσταση της πρώτης συνάρτησης κόστους, δηλαδή του λόγου του αριθμού των χαρακτήρων που εξετάστηκαν προς τον αριθμό των χαρακτήρων που περάσαμε, σαν συνάρτηση του μήκους της συμβολοσειράς. Το πιο ενδιαφέρον συμπέρασμα που προκύπτει από τη συγκεκριμένη γραφική είναι ο αλγόριθμος *είναι αναμενόμενο* ότι θα εκτελέσει λιγότερες από *i* + *patlen* αναφορές στη συμβολοσειρά εισόδου string πριν βρεθεί η ζητούμενη συμβολοσειρά στη θέση *i*. Για παράδειγμα, στην περίπτωση του Αγγλικού αλφάβητου με συμβολοσειρές μήκους 5 ο αλγόριθμος αναμένεται να έχει λιγότερες από (*i* + 5)/4 αναφορές στη συμβολοσειρά εισόδου.

Παρατηρούμε, επίσης, ότι ο ρυθμός με τον οποίο μειώνεται ο αριθμός των αναφορών στη συμβολοσειρά εισόδου ελαττώνεται καθώς αυξάνει το μήκος της συμβολοσειράς. Η συμπεριφορά αυτή οφείλεται στο γεγονός ότι με την αύξηση του μήκους της συμβολοσειράς αυξάνονται οι πιθανότητες ο χαρακτήρας που μόλις διαβάσαμε να υπάρχει σε κάποια θέση της συμβολοσειράς, επομένως μειώνεται και η απόσταση κατά την οποία μπορούμε να μετακινηθούμε προς τα δεξιά.

Στην εικόνα *3δ* φαίνεται η γραφική παράσταση της δεύτερης συνάρτησης κόστους, δηλαδή του λόγου του αριθμού των εντολών που εκτελέστηκαν προς το πλήθος των χαρακτήρων που περάσαμε, ως προς το μήκος της συμβολοσειράς. Διαπιστώνουμε ότι η συγκεκριμένη υλοποίηση του αλγορίθμου *αναμένεται*, όταν έχουμε επαρκώς μεγάλο αλφάβητο, πως θα εκτελεί λιγότερες από $i + patlen$ εντολές προτού ανακαλυφθεί η ζητούμενη συμβολοσειρά στη θέση i. Συνεπώς, η πολυπλοκότητα της υλοποίησης του αλγορίθμου είναι συνήθως χαμηλότερη από γραμμική ακόμα και ως προς τον αριθμό των εκτελούμενων εντολών.

Υπάρχει μια δυσκολία στην εκτίμηση της χρησιμότητας της συνάρτησης $delta_2$. Για παράδειγμα, εάν το αλφάβητο είναι μεγάλο και η ζητούμενη συμβολοσειρά είναι μικρή τότε ο υπολογισμός και η χρήση της $delta_2$ δε μας συμφέρει ιδιαίτερα καθώς το πιθανότερο είναι ότι ο χαρακτήρας που διαβάζουμε κάθε φορά από τη string δε θα εμφανίζεται μέσα στη ζητούμενη συμβολοσειρά και ο αλγόριθμος θα εκτελεί κυρίως τον βρόχο *fast*, αδιαφορώντας για την $delta_2$. Αντίστροφα, σε περίπτωση που έχουμε μεγάλη συμβολοσειρά και μικρό αλφάβητο η $delta_2$ γίνεται πολύ σημαντική, καθώς η εκτέλεση του αλγορίθμου συχνά θα εξέρχεται από τον βρόχο *fast*, η $delta_1$ θα μας δίνει γενικώς μικρές τιμές (καθώς είναι πολύ πιθανό για όλα τα σύμβολα να εμφανίζονται μέσα στην *pat*) και η μόνη πιθανότητα για μεγάλη μετατόπιση είναι να υπάρχει κάπου επανεμφάνιση της συμβολοσειράς κατάληξης. Παρόλο που είναι δύσκολη η ακριβής εκτίμηση της συνεισφοράς της $delta_2$, αξίζει να σημειωθεί ότι η γραμμικότητα του αλγορίθμου στην χειρότερη περίπτωση οφείλεται αποκλειστικά στην ύπαρξη της $delta_2$.

3.2.5 Ιστορική εξέλιξη του αλγορίθμου – η παραλλαγή Horspool.

Οι συναρτήσεις $delta_1$ και $delta_2$ που ορίζονται στη διατύπωση του αλγορίθμου έμειναν γνωστές ως η *ευρετική κακού χαρακτήρα* (bad character heuristic) και η *ευρετική καλής κατάληξης* (good suffix heuristic) αντίστοιχα.

Ο αλγόριθμος Boyer-Moore μέσα σε μικρό χρονικό διάστημα από τη δημοσίευσή του βρέθηκε πως είναι πολύ γρηγορότερος από οποιονδήποτε άλλο αλγόριθμο όταν ενδιαφερόμαστε για την αναζήτηση μίας συμβολοσειράς. Το γεγονός αυτό οδήγησε στην ευρύτατη εφαρμογή του αλγορίθμου (ο οποίος αποτελεί σημείο αναφοράς ακόμη και στις μέρες μας, περισσότερο από 25 χρόνια μετά τη δημοσίευση), ενώ επίσης προκάλεσε το ενδιαφέρον αρκετών ερευνητών σχετικά με το αν μπορεί να βελτιωθεί απόδοσή του. Με το πέρασμα του χρόνου προτάθηκαν διάφορες παραλλαγές και τροποποιήσεις του αλγορίθμου, όπως είναι ο αλγόριθμος Apostolico-Giancarlo [7] ή ο αλγόριθμος Turbo Boyer-Moore [8].

Σχεδόν όλες οι τροποποιήσεις του αλγορίθμου Boyer-Moore παρουσιάζουν έναν πιο έξυπνο αλλά και περίπλοκο αλγόριθμο. Η μόνη εξαίρεση είναι η παραλλαγή του Horspool [6], η οποία στην ουσία είναι μια απλοποίηση του αλγορίθμου Boyer-Moore. Καθώς επιλέξαμε, για τους λόγους που αναλύονται στην παράγραφο 4.2, να υλοποιησουμε την παραλλαγή του Horspool αντί για τον παραδοσιακό αλγόριθμο Boyer-Moore ή κάποιον άλλο παρόμοιο αλγόριθμο, θα παραθέσουμε στο σημείο αυτό μία σύντομη περιγραφή του αλγορίθμου του Horspool.

Η πρόταση του R. Horspool το 1980 ήταν να χρησιμοποιείται μόνο η ευρετική κακού χαρακτήρα στον χαρακτήρα που συγκρίνεται με τον τελευταίο χαρακτήρα της ζητούμενης συμβολοσειράς. Η πρόταση αυτή προέκυψε από την εξής παρατήρηση: η ευρετική κακού χαρακτήρα που χρησιμοποιεί ο αλγόριθμος Boyer-Moore δεν είναι αποδοτική σε περίπτωση που έχουμε μικρό αλφάβητο. Οταν όμως το αλφάβητο είναι μεγάλο συγκριτικά με το μήκος της συμβολοσειράς που ψάχνουμε, όπως συμβάινει συνήθως όταν έχουμε τον πίνακα των χαρακτήρων ASCII ή όταν ψάχνουμε μέσα σε γραπτό κείμενο, η συνάρτηση αυτή γίνεται πολύ χρήσιμη. Επομένως, χρησιμοποιώντας μόνο την ευρετική κακου χαρακτήρα μπορούμε να έχουμε έναν αρκετά αποδοτικό και απλό αλγόριθμο.

Ο αλγόριθμος Horspool χρειάζεται χρόνο προεπεξεργασίας της τάξης $\boldsymbol{O}(m+\sigma)$, ενώ οι απαιτήσεις του σε χώρο είναι $\boldsymbol{O}(\sigma)$, όπου σ το πλήθος των στοιχείων στο χρησιμοποιούμενο αλφάβητο και m το μήκος της ζητούμενης συμβολοσειράς. Για τη χειρότερη περίπτωση η παραλλαγή του Horspool έχει χαμηλότερη απόδοση από τον αλγόριθμο Boyer-Moore, όμως μπορεί να αποδειχτεί πως για τη μέση περίπτωση ο αριθμός των συγκρίσεων που εκτελεί ο Horspool για κάθε χαρακτήρα του κειμένου εισόδου κυμαίνεται ανάμεσα σε $1/m$ και $2/(m+1)$.

Πρακτικά, ο αλγόριθμος Horspool μπορεί να εκτελεί σε ορισμένες περιπτώσεις λίγο περισσότερες συγκρίσεις από τον απλό αλγόριθμο Boyer-Moore. Το γεγονός, όμως, ότι ο αλγόριθμος Horspool είναι πιο απλός σημαίνει ότι για κάθε χαρακτήρα που εξετάζεται απαιτείται μικρότερος αριθμός εντολών. Το αποτέλεσμα είναι ο αλγόριθμος Horspool να είναι συχνά πιο γρήγορος από τον Boyer-Moore.

3.3 Ο αλγόριθμος Wu-Manber.

Πρόκειται για τον πιο πρόσφατο από τους αλγορίθμους που εξετάστηκαν, καθώς δημοσιεύτηκε το 1994 από τους Sun Wu και Udi Manber [3]. Είναι σχεδιασμένος ώστε να μπορεί να εκτελεί παράλληλη αναζήτηση ενός μεγάλου συνόλου από συμβολοσειρές, του οποίο το μέγεθος μπορεί να κυμαίνεται από μερικές δεκάδες συμβολοσειρές μέχρι δεκάδες χιλιάδες.

Ο αλγόριθμος Wu-Manber έχει δανειστεί κάποιες ιδέες από τον αλγόριθμο Boyer-Moore, καθώς και στον Wu-Manber χρησιμοποιείται η λογική του να εξετάζονται λίγοι χαρακτήρες που βρίσκονται σε θέσεις αντίστοιχες με το τέλος των ζητούμενων συμβολοσειρών και ανάλογα με τις τιμές αυτών των χαρακτήρων να προσπερνάμε γρήγορα κάποιους χαρακτήρες ή να τους εξετάζουμε πιο προσεκτικά.

Πρόκειται για έναν αρκετά αποδοτικό αλγόριθμο σε περίπτωση που ενδιαφερόμαστε για την αναζήτηση μεγάλου αριθμού συμβολοσειρών. Ένα από τα σημαντικά προτερήματα του αλγορίθμου είναι ο μεγάλος βαθμός στον οποίο μπορεί να τροποποιηθεί, αλλάζοντας απλά συγκεκριμένες σταθερές, ώστε να ταιριάζει καλύτερα

στα δεδομένα του προβλήματός μας. Επίσης, οι απαιτήσεις του αλγορίθμου σε χώρο είναι σχετικά ανεξάρτητες από τον αριθμό των συμβολοσειρών ενώ μπορούν να καθοριστούν σε μεγάλο βαθμό από το χρήστη ανάλογα με τις απαιτήσεις και της δυνατότητες του υπολογιστή.

Το μόνο σημαντικό μειονέκτημα του αλγορίθμου είναι ότι εξαρτάται σε μεγάλο βαθμό από το μήκος της μικρότερης συμβολοσειράς προς αναζήτηση, με αποτέλεσμα ο αλγόριθμος να χάνει σημαντικά σε απόδοση σε περίπτωση που υπάρχει έστω και μία πολύ μικρή συμβολοσειρά (για παράδειγμα συμβολοσειρά μήκους μικρότερου από 3).

3.3.1 Γενική περιγραφή του αλγορίθμου.

Ο αλγόριθμος αυτός έχει σχεδιαστεί για την επίλυση του προβλήματος της αναζήτησης πολλών συμβολοσειρών μέσα σε κάποιο κείμενο. Έστω $P = \{p_1, p_2,\ldots, p_k\}$ το σύνολο των ζητούμενων συμβολοσειρών με κάθε συμβολοσειρά να αποτελείται από μία ακολουθία χαρακτήρων από ένα σταθερό αλφάβητο Σ. Έστω $T = t_1, t_2\ldots, t_N$ ένα μεγάλο κείμενο το οποίο επίσης αποτελείται από χαρακτήρες του Σ. Το ζητούμενο είναι να βρούμε όλες τις εμφανίσεις μέσα στο T όλων των συμβολοσειρών που ανήκουν στο σύνολο P.

Η κεντρική ιδέα του αλγορίθμου Boyer-Moore, η οποία κατά κάποιο τρόπο χρησιμοποιείται και στον αλγόριθμο Wu-Manber, έχει ως εξής: έστω ότι έχουμε μια συμβολοσειρά προς αναζήτηση μήκους m της οποίας στιγμιότυπα αναζητούμε μέσα σε ένα κείμενο εισόδου. Ξεκινάμε συγκρίνοντας τον *τελευταίο* χαρακτήρα της ζητούμενης συμβολοσειράς με t_m, δηλαδή τον m-οστό χαρακτήρα από την αρχή του κειμένου εισόδου. Σε περίπτωση που οι δύο χαρακτήρες δεν ταιριάζουν, γεγονός πολύ πιθανότερο για περισσότερα κείμενα εισόδου από ότι να ταιριάζουν οι δύο χαρακτήρες, τότε εξετάζουμε που υπάρχει η δεξιότερη εμφάνιση του χαρακτήρα *tm* μέσα στη ζητούμενη συμβολοσειρά και μετακινούμαστε κατάλληλο αριθμό θέσεων προς τα δεξιά.

Για παράδειγμα, στην περίπτωση που ο χαρακτήρας t_m δεν εμφανίζεται καθόλου μέσα στη ζητούμενη συμβολοσειρά τότε μπορούμε με ασφάλεια να μεταφέρουμε το σημείο σύγκρισης κατά m θέσεις προς τα δεξιά μέσα στο κείμενο εισόδου και να

εξετάσουμε στο επόμενο βήμα τον χαρακτήρα t_{2m}. Εάν ο tm εμφανίζεται στην τέταρτη θέση από το τέλος μέσα στη συμβολοσειρά που ψάχνουμε τότε θα μετακινούμασταν κατά 4 θέσεις προς τα δεξιά και ούτω καθεξής. Σε κείμενα φυσικής γλώσσας το σύνηθες είναι να έχουμε τις περισσότερες φορές μετακινήσεις μεγέθους m, γεγονός που οδηγεί σε έναν ταχύτατο αλγόριθμο στην πράξη.

Την ιδέα αυτή θα θέλαμε να εφαρμόσουμε και στην περίπτωση που έχουμε πολλές συμβολοσειρές προς αναζήτηση. Καθώς, όμως, αυξάνεται ο αριθμός των συμβολοσειρών που ψάχνουμε (και εμείς θέλουμε να υποστηρίζουμε την αναζήτηση χιλιάδων συμβολοσειρών παράλληλα) αυξάνονται και οι πιθανότητες ότι τα περισσότερα, αν όχι όλα, τα σύμβολα του χρησιμοποιούμενου αλφαβήτου θα ταιριάζουν με τον τελευταίο χαρακτήρα κάποιας συμβολοσειράς, άρα θα υπάρχουν πολύ λίγες μεγάλες μετακινήσεις. Θα δείξουμε πως μπορεί να αντιμετωπιστεί το πρόβλημα αυτό διατηρώντας παράλληλα την ουσία (και την ταχύτητα) του αλγορίθμου Boyer-Moore.

Στην πρώτη φάση του αλγορίθμου Wu-Manber γίνεται η προεπεξεργασία του συνόλου των συμβολοσειρών προς αναζήτηση. Για εφαρμογές όπου ένα συγκεκριμένο σύνολο από συμβολοσειρές χρησιμοποιείται για πολλές αναζητήσεις είναι ωφέλιμο να αποθηκεύονται τα αποτελέσματα της προεπεξεργασίας σε ένα αρχείο στο δίσκο (ή ακόμα και στη μνήμη). Παρόλα αυτά, η φάση της προεπεξεργασίας είναι αρκετά αποδοτική ώστε να μπορεί να εκτελείται επί τόπου για της περισσότερες περιπτώσεις.

Κατά τη διάρκεια της προεπεξεργασίας κατασκευάζονται 3 πίνακες, ο πίνακας SHIFT, ο πίνακας HASH και ο πίνακας PREFIX. Ο πίνακας SHIFT είναι παρόμοιος, αν και με κάποιες διαφορές, με τον αντίστοιχο πίνακα «κακού χαρακτήρα» που χρησιμοποιεί ένας αλγόριθμος τύπου Boyer-Moore. Ο πίνακας SHIFT χρησιμοποιείται για να αποφασίσουμε πόσους χαρακτήρες του κειμένου εισόδου μπορούμε να προσπεράσουμε, συνεπώς να αγνοήσουμε, με ασφάλεια κάθε φορά. Οι πίνακες HASH και PREFIX χρησιμοποιούνται σε περίπτωση που ο πίνακας SHIFT μας δώσει μηδενική τιμή και χρησιμοποιούνται για την επιλογή της υποψήφιας συμβολοσειράς που μπορεί να ταιριάζει με το κείμενο και την επαλήθευση της πιθανής εμφάνισης αυτής της συμβολοσειράς μέσα στο κείμενο. Οι ακριβείς λεπτομέρειες για την κατασκευή των πινάκων δίνονται παρακάτω.

3.3.2 Το στάδιο προεπεξεργασίας.

Το πρώτο πράγμα που κάνουμε είναι να βρούμε το μήκος της μικρότερης από της ζητούμενες συμβολοσειρές, έστω m. Στη συνέχεια, για κάθε μία από της συμβολοσειρές θα εξετάζουμε μόνο τους πρώτους m χαρακτήρες. Δηλαδή εφαρμόζουμε έναν άτυπο περιορισμό ότι όλες οι ζητούμενες συμβολοσειρές έχουν το ίδιο μήκος. Στην πορεία γίνεται εμφανές ότι η παράμετρος m είναι σημαντική για την απόδοση του αλγορίθμου. Σε περίπτωση που οι συμβολοσειρές είναι πολύ μικρές, για παράδειγμα μήκους 2, δεν μπορούμε ποτέ να μετακινηθούμε περισσότερες από δύο θέσεις κάθε φορά, γεγονός που μειώνει δραστικά την απόδοση.

Αντί να εξετάζουμε έναν χαρακτήρα του κειμένου κάθε φορά, τους εξετάζουμε σε μπλοκ μεγέθους B χαρακτήρων. Έστω ότι το M ισοδυναμεί με το συνολικό μέγεθος όλων των συμβολοσειρών, δηλαδή $M = k*m$, ενώ με την τιμή c συμβολίζουμε το μέγεθος του αλφάβητου. Όπως θα φανεί στην παράγραφο 3.3.4, μία καλή τιμή της παραμέτρου B είναι $\log_c 2M$, αν και στην πράξη χρησιμοποιείται η τιμή $B = 2$ ή $B = 3$. Ο πίνακας SHIFT εξυπηρετεί τον ίδιο ρόλο με την ευρετική κακού χαρακτήρα ενός αλγορίθμου Boyer-Moore, με την μόνη διαφορά ότι καθορίζει την τιμή κατά την οποία θα μετακινηθούμε εξετάζοντας τους τελευταίους B χαρακτήρες, αντί να εξετάζει μόνο τον τελευταίο. Για παράδειγμα, σε περίπτωση που μία ακολουθία B χαρακτήρων του κειμένου δεν εμφανίζεται σε καμία από τις συμβολοσειρές, τότε μπορούμε να μετακινηθούμε κατά $m–B+1$ θέσεις προς τα δεξιά.

Ας υποθέσουμε τώρα ότι ο πίνακας SHIFT κρατάει μία τιμή για κάθε δυνατή συμβολοσειρά μήκους B. Αυτό συνεπάγεται ότι το μέγεθος του πίνακα θα είναι $|c|^B$ (στην πραγματικότητα χρησιμοποιείται μικρότερος πίνακας, στον οποίο αρκετές συμβολοσειρές αντιστοιχούν στην ίδια τιμή για εξοικονόμηση χώρου). Κάθε συμβολοσειρά μεγέθους B αντιστοιχεί, με τη χρήση μιας συνάρτησης hash που θα παρουσιάσουμε αργότερα, σε έναν ακέραιο αριθμό ο οποίος χρησιμοποιείται σαν δείκτης στον πίνακα SHIFT. Οι τιμές στα διάφορα στοιχεία του πίνακα SHIFT ισοδυναμούν με το πόσους χαρακτήρες μπορούμε να υπερπηδήσουμε, κινούμενοι προς τα δεξιά, ενώ εξετάζουμε το κείμενο.

Έστω $X=x_1\ldots x_B$ η συμβολοσειρά μήκους B χαρακτήρων που διαβάστηκε από το κείμενο. Υποθέτουμε πως η συμβολοσειρά X αντιστοιχεί στην i-οστή του πίνακα SHIFT. Προκύπτουν δύο περιπτώσεις:

1. Η ακολουθία X δεν εμφανίζεται σαν υποσυμβολοσειρά σε κανένα μέλος του συνόλου P. Σε αυτή την περίπτωση μπορούμε με σιγουριά να κινηθούμε κατά $m-B+1$ χαρακτήρες προς τα μπρος στο κείμενο όπου γίνεται η αναζήτηση. Αν μετακινούμασταν κατά λιγότερες θέσεις τότε οι B τελευταίοι χαρακτήρες οι οποίοι μόλις εξετάστηκαν θα αντιστοιχούσαν σε κάποιους χαρακτήρες στο εσωτερικό ορισμένων συμβολοσειρών, γεγονός που γνωρίζουμε ότι δεν ισχύει. Στη θέση SHIFT[i] αποθηκεύουμε την τιμή $m-B+1$.
2. Η ακολουθία X εμφανίζεται σε ορισμένες συμβολοσειρές. Εάν ισχύει αυτό τότε ψάχνουμε για την δεξιότερη θέση όπου εμφανίζεται η X μέσα σε κάποια από της ζητούμενες συμβολοσειρές. Ας υποθέσουμε ότι η X βρέθηκε να τελειώνει στη θέση q της συμβολοσειράς p_j και δε βρέθηκε σε καμία άλλη συμβολοσειρά σε θέση μεγαλύτερη της q. Θα αποθηκεύσουμε στη θέση SHIFT[i] την τιμή m-q.

Για να υπολογίσουμε τις τιμές του πίνακα SHIFT αντιμετωπίζουμε κάθε συμβολοσειρά $p_i=a_1a_2\ldots a_m$ ξεχωριστά. Για κάθε υποσυμβολοσειρά $a_{j-B+1}\ldots a_j$ μήκους B της p_i βρίσκουμε τη θέση h του πίνακα SHIFT στην οποία αντιστοιχεί και θέτουμε την τιμή SHIFT[h] ίση με το ελάχιστο ανάμεσα στην τρέχουσα τιμή SHIFT[h] (αρχικά κάθε στοιχείο του SHIFT έχει τιμή ίση με m-B+1) και την τιμή m-j (ο αριθμός των θέσεων που πρέπει να μετακινηθούμε ώστε να ευθυγραμμιστούμε με τη συγκεκριμένη υποσυμβολοσειρά).

Οι τιμές που περιέχει ο πίνακας SHIFT είναι οι μεγαλύτερες δυνατές ώστε να κάνουμε ασφαλείς μετακινήσεις. Αν αντικαταστήσουμε οποιοδήποτε στοιχείο του πίνακα SHIFT με *μικρότερη* τιμή τότε θα γίνονται μικρότερες μετατοπίσεις και ο αλγόριθμος θα πάρει περισσότερο χρόνο, όμως θα εξακολουθεί να είναι ασφαλής και θα αναγνωριστούν όλα τα στιγμιότυπα των ζητούμενων συμβολοσειρών. Συνεπώς μπορούμε να χρησιμοποιήσουμε ένα συμπιεσμένο πίνακα SHIFT, σε κάθε στοιχείο του οποίου αντιστοιχούν περισσότερες από μία συμβολοσειρές και η τιμή του συγκεκριμένου στοιχείου είναι το ελάχιστο όλων. Στην εφαρμογή *agrep* που δημιουργήθηκε από τους

συγγραφείς χρησιμοποιούνται και οι δύο περιπτώσεις: όταν ο αριθμός των συμβολοσειρών είναι σχετικά μικρός επιλέγεται B=2 και χρησιμοποιείται πλήρης πίνακας SHIFT, διαφορετικά επιλέγεται B=3 και χρησιμοποιείται συμπιεσμένος πίνακας SHIFT. Σε κάθε περίπτωση ο αλγόριθμος δεν παρουσιάζει καμία διαφορά όταν ο πίνακας SHIFT είναι συμπιεσμένος.

Όσο βρίσκουμε τιμές του πίνακα SHIFT μπορούμε να μετακινούμαστε προς τα δεξιά και α συνεχίζουμε την ανίχνευση. Κάτι τέτοιο συμβαίνει τις περισσότερες φορές. Σε ένα τυπικό παράδειγμα που εκτελέστηκε, η τιμή της μετατόπισης ήταν μηδέν για το 5% των ελέγχων όταν είχαμε 100 συμβολοσειρές, 27% για 1000 συμβολοσειρές και 53% για 5000 συμβολοσειρές).

Σε περίπτωση, όμως, που σε κάποιο έλεγχο βρούμε μηδενική τιμή μετατόπισης, είναι πιθανό η τρέχουσα υποσυμβολοσειρά του κειμένου να είναι το στιγμιότυπο κάποιας ζητούμενης συμβολοσειράς. Το ερώτημα που προκύπτει είναι ποια συμβολοσειρά μπορεί να ταιριάζει. Με σκοπό να αποφευχθεί ο έλεγχος αυτής της υποσυμβολοσειράς του κειμένου με κάθε μία από τις συμβολοσειρές, χρησιμοποιείται μία μέθοδος hashing ώστε να ελαχιστοποιηθεί ο αριθμός των συμβολοσειρών που θα εξεταστούν.

Προηγουμένως είχαμε υπολογίσει την αντιστοιχία κάθε πιθανής συμβολοσειράς μήκους B με έναν ακέραιο αριθμό που χρησιμοποιείται σαν δείκτης στον πίνακα SHIFT. Ακριβώς ο ίδιος αριθμός χρησιμοποιείται ως δείκτης σε ένα δεύτερο πίνακα, τον HASH. Το i-οστό στοιχείο του πίνακα HASH περιέχει ένα δείκτη σε μία λίστα από συμβολοσειρές, των οποίων οι B τελευταίοι χαρακτήρες όταν γίνουν hashed δίνουν την τιμή i.

Ο πίνακας HASH συνήθως είναι αρκετά αραιός, καθώς το πολύ να είναι γεμάτος αριθμός στοιχείων ίσος με το σύνολο των συμβολοσειρών (σε περίπτωση που κάθε συμβολοσειρά δίνει αποτέλεσμα hash διαφορετικό από όλες τις άλλες) ενώ ο πίνακας SHIFT έχει τιμές για όλες τις πιθανές υποσυμβολοσειρές μήκους B. Το γεγονός αυτό συνεπάγεται ότι σημαντικό ποσοστό της μνήμης μένει αχρησιμοποίητο, όμως μας δίνεται η δυνατότητα επαναχρησιμοποίησης της τιμής που μας επέστρεψε η συνάρτηση hash, γεγονός που μεταφράζεται σε σημαντική μείωση του χρόνου. Μία εναλλακτική για εξοικονόμηση χώρου είναι να κάνουμε το μέγεθος του πίνακα HASH ίσο με κάποια

δύναμη του 2 η οποία να αποτελεί κλάσμα του μεγέθους του πίνακα SHIFT ώστε να χρησιμοποιούμε μόνο τα τελευταία bits της συνάρτησης hash.

Έστω ότι *h* είναι η τιμή της συνάρτησης hash για τους *B* χαρακτήρες του κειμένου και ας υποθέσουμε ότι SHIFT[*h*]=0. Το στοιχείο HASH[*h*] είναι ένας δείκτης *p* σε δύο διαφορετικούς πίνακες ταυτόχρονα: Διατηρούμε μία λίστα με δείκτες στις συμβολοσειρές, την PAT_POINT, η οποία είναι ταξινομημένη ως προς τις τιμές hash των *B* τελευταίων χαρακτήρων κάθε συμβολοσειράς. Ο p δείχνει στην αρχή της λίστας των συμβολοσειρών οι οποίες έχουν τιμή hash ίση με *h*. Για να βρούμε το τέλος της λίστας συνεχίζουμε να αυξάνουμε το δείκτη μέχρι να γίνει ίσος με την τιμή HASH[*h*+1] (επειδή ολόκληρη η λίστα είναι ταξινομημένη ανάλογα με τις τιμές hash των συμβολοσειρών). Έτσι, για παράδειγμα, αν SHIFT[*i*] $\leq$ 0 τότε θα έχουμε ότι HASH[*h*]=HASH[*h*+1] επειδή καμία συμβολοσειρά δεν έχει κατάληξη που να μας δίνει τιμή hash ίση με *h*. Επιπλέον υπάρχει ο πίνακας PREFIX, του οποίου η σκοπιμότητα θα εξηγηθεί συντόμως.

Τα κείμενα φυσικής γλώσσας δεν είναι τελείως τυχαία. Για παράδειγμα, οι καταλήξεις 'ion' και 'ing' είναι πολύ συχνές σε κείμενα Αγγλικής γλώσσας. Αυτές οι καταλήξεις όχι μόνο θα εμφανίζονται πολύ συχνά στο κείμενο, αλλά είναι πιθανό να εμφανίζονται και σε αρκετές από τις συμβολοσειρές που αναζητάμε. Το αποτέλεσμα θα είναι να υπάρχουν συγκρούσεις στον πίνακα HASH καθώς οι συμβολοσειρές με την ίδια κατάληξη θα αντιστοιχούν στην ίδια θέση του πίνακα. Όποτε συναντάμε τέτοιες καταλήξεις στο κείμενο η τιμή του πίνακα SHIFT θα είναι 0 (με την προϋπόθεση ότι είναι καταλήξεις και σε ορισμένες συμβολοσειρές), και θα χρειαστεί να εξετάσουμε ξεχωριστά όλες τις συμβολοσειρές με την συγκεκριμένη κατάληξη για δούμε αν ταιριάζουν με το κείμενο. Με στόχο την επιτάχυνση αυτής της διαδικασίας εισάγεται ο πίνακας PREFIX.

Εκτός από το να αντιστοιχούμε σε κάθε κατάληξη μήκους *B* μια τιμή, ακολουθούμε ανάλογη διαδικασία για τους *B'* αρχικούς χαρακτήρες κάθε συμβολοσειράς και εισάγουμε τις τιμές στον πίνακα PREFIX (βρέθηκε ότι μία καλή τιμή είναι *B'* = 2). Κάθε φορά που βρίσκουμε μία τιμή SHIFT ίση με μηδέν και πηγαίνουμε στον πίνακα HASH για να εξετάσουμε μήπως ταιριάζει κάποια συμβολοσειρά, εξετάζουμε και τις τιμές του πίνακα PREFIX. Σε κάθε θέση του πίνακα HASH δεν περιέχεται, για την

αντίστοιχη κατάληξη, μόνο ο δείκτης στη λίστα με τις συμβολοσειρές που μπορεί να ταιριάζουν, αλλά περιέχονται και οι τιμές hash του προθέματος κάθε μίας από αυτές. Το επόμενο βήμα είναι να υπολογίσουμε την τιμή hash των χαρακτήρων του κειμένου που αντιστοιχούν στις θέσεις του προθέματος (δηλαδή βρίσκονται $m - B'$ θέσεις προς τα αριστερά) και να τη χρησιμοποιήσουμε ώστε να απορρίψουμε εύκολα τις συμβολοσειρές που, ενώ μεν έχουν ίδια κατάληξη, διαφέρουν στο πρόθεμα.

Αυτή η μέθοδος αποτελεί έναν αποτελεσματικό τρόπο φιλτραρίσματος καθώς είναι πολύ πιο απίθανο να έχουμε δύο συμβολοσειρές που έχουν κοινό πρόθεμα *και* κοινή κατάληξη. Είναι επίσης και ένας καλός τρόπος εξισορρόπησης υπό την έννοια ότι ο επιπλέον φόρτος που χρειάζεται για τον υπολογισμό της τιμής hash των προθεμάτων είναι σημαντικός μόνο όταν η τιμές του SHIFT είναι συχνά 0, γεγονός που σημαίνει πως υπάρχουν πολλές συμβολοσειρές και μεγαλύτερη πιθανότητα σύγκρουσης.

Η φάση της προεπεξεργασίας μπορεί να φαίνεται αρκετά περίπλοκη, στην πραγματικότητα όμως ολοκληρώνεται πολύ γρήγορα. Στην υλοποίησή μας το μέγεθος των τριών πινάκων τέθηκε ίσο με 2^{15}=32,768 στοιχεία. Η εκτέλεση του αλγορίθμου με είσοδο ένα σχεδόν κενό αρχείο κειμένου, το οποίο αποτελεί μέτρηση του χρόνου που χρειάζεται για την προεπεξεργασία, χρειάστηκε μόλις 0.16 δευτερόλεπτα για 10,000 συμβολοσειρές.

3.3.3 Το στάδιο της αναζήτησης.

Στο σημείο αυτό παρουσιάζεται αναλυτικότερα η φάση της αναζήτησης μέσα στο κείμενο εισόδου. Ο κεντρικός βρόχος του αλγορίθμου αποτελείται από τα ακόλουθα 4 βήματα.

i) Υπολογισμός της τιμής hash h με βάση τους τρέχοντες B χαρακτήρες του κειμένου (δηλαδή τους $t_{j-B+1}...t_j$, όπου j η τρέχουσα θέση μέσα στο κείμενο).
ii) Έλεγχος της τιμής SHIFT[h]: αν είναι μεγαλύτερη από μηδέν κινούμαστε προς τα δεξιά και εκτελούμε πάλι το βήμα i), διαφορετικά πηγαίνουμε στο βήμα iii).
iii) Υπολογισμός της τιμής hash των χαρακτήρων του προθέματος στο κείμενο (δηλαδή των χαρακτήρων $t_{j-m+1}\ldots t_{j-m+B'+1}$). Η τιμή αυτή καλείται text_prefix.

iv) Για κάθε p τέτοιο ώστε HASH[h] $\leq p <$ HASH[h+1] ελέγχεται αν PREFIX[p] = test_prefix. Σε περίπτωση που είναι ίσα τότε εξετάζεται η αντίστοιχη συμβολοσειρά (η οποία βρίσκεται στη θέση PAT_POINT[p]) απευθείας με το κείμενο.

3.3.4 Μια πρόχειρη εκτίμηση του χρόνου εκτέλεσης.

Ακολουθεί μια εκτίμηση του χρόνου εκτέλεσης του αλγορίθμου βασισμένη στην υπόθεση ότι τόσο οι συμβολοσειρές προς αναζήτηση όσο και το κείμενο εισόδου είναι τυχαίες συμβολοσειρές με ομοιόμορφη κατανομή. Στην πραγματικότητα, βέβαια, ούτε το κείμενο είναι τυχαίο ούτε οι συμβολοσειρές, όμως η υπόθεση αυτή μας επιτρέπει να αποκτήσουμε μια γενική ιδέα για την απόδοση του αλγορίθμου. Στη συνέχεια θα δειχτεί ότι ο αναμενόμενος χρόνος εκτέλεσης εξαρτάται σε μικρότερο από γραμμικό βαθμό από το μέγεθος το κειμένου (όμως όχι κατά πολύ μικρότερο).

Έστω ότι N είναι το μήκος του κειμένου εισόδου, P ο αριθμός των συμβολοσειρών, m το μήκος κάθε συμβολοσειράς, $M=mP$ το συνολικό μέγεθος όλων των συμβολοσειρών και υποθέτουμε ότι $N \geq M$. Επίσης, έστω c το μέγεθος του αλφάβητου. Το μέγεθος του μπλοκ των χαρακτήρων που διαβάζουμε με σκοπό να βρούμε μία τιμή του πίνακα SHIFT καθορίζεται ως $B= \log_c 2M$. Ο πίνακας SHIFT περιέχει τιμές για όλες τις πιθανές συμβολοσειρές μήκους B, συνεπώς υπάρχουν $c^B = c^{\log_c 2M} \leq 2Mc$ στοιχεία μέσα στον πίνακα. Ο πίνακας SHIFT μπορεί να κατασκευαστεί σε χρόνο $\boldsymbol{O}(M)$ επειδή κάθε υποσυμβολοσειρά μήκους B οποιασδήποτε συμβολοσειράς εξετάζεται μία φορά και χρειάζεται, κατά μέσο όρο, σταθερό χρόνο για την εξέτασή της.

Ξεχωρίζουμε δύο περιπτώσεις όσον αφορά το χρόνο αναζήτησης. Η πρώτη περίπτωση εμφανίζεται όταν η τιμή του πίνακα SHIFT είναι διαφορετική του μηδενός. Σε αυτή την περίπτωση δε χρειάζεται επιπλέον επεξεργασία στο συγκεκριμένο σημείο του κειμένου και εκτελούμε απλά μία μετατόπιση. Η δεύτερη περίπτωση είναι πιο περίπλοκη και αφορά τις στιγμές που βρίσκουμε τιμή SHIFT ίση με 0. Εάν ισχύει κάτι τέτοιο είμαστε αναγκασμένοι να διαβάσουμε περισσότερους χαρακτήρες από το κείμενο και να συμβουλευτούμε και τον πίνακα HASH.

ΠΡΟΤΑΣΗ 1. Η πιθανότητα ώστε μία τυχαία συμβολοσειρά μεγέθους B να μας δώσει τιμή μετατόπισης i, όπου $0 \leq i \leq m\text{-}B+1$, είναι $\leq 1/2m$.

ΑΠΟΔΕΙΞΗ. Για τον πίνακα SHIFT το πολύ P= M/m συμβολοσειρές μας δίνουν τιμή i για κάθε i από 0 μέχρι $m\text{-}B+1$. Όμως, το πλήθος όλων των πιθανών συμβολοσειρών μήκους B είναι τουλάχιστο $2M$. Επομένως η πιθανότητα ώστε μία τυχαία συμβολοσειρά να μας δώσει τιμή μετατόπισης i είναι $\leq 1/2m$. Η πιθανότητα αυτή ισχύει επίσης και για την περίπτωση που η τιμή του SHIFT είναι 0.

Η πρόταση 1 μας οδηγεί στο συμπέρασμα ότι η αναμενόμενη τιμή της μετατόπισης είναι $m/2$. Από τη στιγμή που απαιτείται χρόνος $\boldsymbol{O}(B)$ κάθε φορά που θέλουμε να υπολογίσουμε ένα αποτέλεσμα της συνάρτησης hash, ο συνολικός χρόνος που θα απαιτείται στις περιπτώσεις που έχουμε μη μηδενικές μετατοπίσεις είναι $\boldsymbol{O}(BN/m)$.

Το επιπλέον φιλτράρισμα που γίνεται με βάση τα προθέματα των συμβολοσειρών κάνει πολύ μικρή την πιθανότητα να έχουμε λάθος αναγνώριση μίας συμβολοσειράς ως πιθανής για ύπαρξη μέσα στο κείμενο. Πιο συγκεκριμένα, ας υποθέσουμε ότι $B' = B$. Η ώστε πιθανότητα μία συγκεκριμένη συμβολοσειρά να έχει κοινή κατάληξη *και* κοινό πρόθεμα με μία άλλη συμβολοσειρά είναι $< 1/M$, δηλαδή ουσιαστικά αμελητέα. Επομένως, ο χρόνος που χρειάζεται στην περίπτωση που έχουμε τιμή SHIFT ίση με 0 είναι και αυτός $\boldsymbol{O}(B)$, εκτός από την περίπτωση που έχουμε πραγματικά εμφάνιση κάποιας συμβολοσειράς μέσα στο κείμενο, οπότε χρειάζεται να συγκρίνουμε ολόκληρη τη συμβολοσειρά με το κείμενο και απαιτείται χρόνος $\boldsymbol{O}(m)$. Εφόσον η τιμή 0 για τη μετατόπιση εμφανίζεται σε ποσοστό $<1/2m$ των ελέγχων που κάνουμε (όπως προκύπτει από την πρόταση 1), ο απαιτούμενος χρόνος και σε αυτή την περίπτωση είναι επίσης $\boldsymbol{O}(BN/m)$.

3.3.5 Πειραματικά αποτελέσματα.

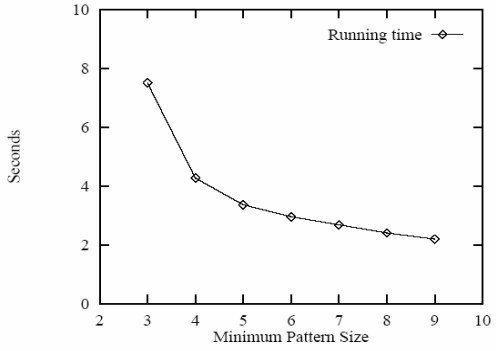
Οι συγγραφείς [3] εκτέλεσαν αρκετά πειράματα δοκιμάζοντας διαφορετικές ρυθμίσεις και δεδομένα με σκοπό να εξετάσουν την απόδοση του αλγορίθμου απέναντι

σε άλλους υπάρχοντες αλγορίθμους, αλλά και να μελετήσουν την επίδραση που μπορεί να έχει η αλλαγή μίας παραμέτρου του προβλήματος στην απόδοση του συστήματος.

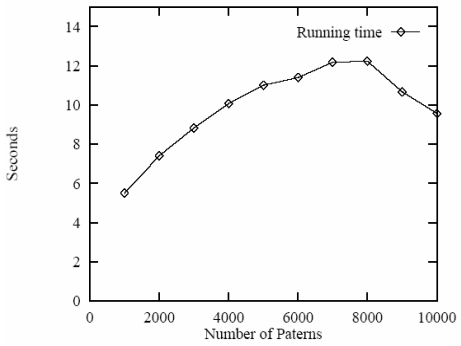
Ως προς τη σύγκριση του Wu-Manber με άλλους αλγόριθμους, βρέθηκε ότι σχεδόν σε όλες τις περιπτώσεις, με εξαίρεση εκείνες όπου οι ζητούμενες συμβολοσειρές ήταν πάρα πολύ μικρές ή πολύ λίγες στο πλήθος, ο αλγόριθμος Wu-Manber ήταν σαφώς γρηγορότερος, ενώ η διαφορά στην ταχύτητα γινόταν ακόμη πιο αισθητή καθώς αυξανόταν ο αριθμός των συμβολοσειρών.

Σημαντικό ενδιαφέρον έχει η εξέταση της ταχύτητας του αλγορίθμου ως συνάρτηση του ελάχιστου μήκους *m* των συμβολοσειρών προς αναζήτηση. Τα αποτελέσματα φαίνονται στην εικόνα *3ε*. Στην δεύτερη εικόνα, την *3στ*, παρατηρούμε την καμπύλη του χρόνου εκτέλεσης ως συνάρτηση του αριθμού των αναζητούμενων συμβολοσειρών.

Στην περίπτωση της εικόνας *3ε*, για κάθε διαφορετική τιμή ελάχιστου μήκους χρησιμοποιήθηκαν 100 διαφορετικές συμβολοσειρές, ενώ το μέσο μήκος σε κάθε περίπτωση ήταν περίπου μία μονάδα μεγαλύτερο από το ελάχιστο μήκος. Παρατηρούμε σταθερή μείωση του χρόνου εκτέλεσης ως προς το ελάχιστο μήκος, γεγονός αναμενόμενο αν σκεφτούμε πως όσο αυξάνει το m τόσο μεγαλύτερες μετατοπίσεις μπορούμε να εκτελούμε. Η καμπύλη που προέκυψε ταιριάζει σε μεγάλο βαθμό με την καμπύλη 1/(*m*-*c*) όπου *c* μία μικρή σταθερά τέτοια ώστε η ποσότητα *m*-*c* να είναι ίση με τον μέσο αριθμό θέσεων που κινούμαστε σε κάθε μετατόπιση.



Εικόνα 3ε) Ο χρόνος εκτέλεσης ως συνάρτηση του ελάχιστου μήκους των συμβολοσειρών [3].



Εικόνα 3στ) Ο χρόνος εκτέλεσης ως συνάρτηση του πλήθους των συμβολοσειρών [3].

Στην εικόνα *3στ* παρατηρούμε ένα παράδοξο φαινόμενο. Αρχικά βλέπουμε να αυξάνεται ο χρόνος καθώς αυξάνεται το πλήθος των συμβολοσειρών, γεγονός απόλυτα λογικό και αναμενόμενο καθώς αύξηση του πλήθους συνεπάγεται μείωση της μέσης τιμής της μετατόπισης που μπορούμε να κάνουμε κάθε φορά. Από ένα σημείο και μετά, όμως, παρατηρούμε *μείωση* του χρόνου εκτέλεσης παράλληλα με την αύξηση του πλήθους των συμβολοσειρών. Το παράδοξο αυτό δεν οφείλεται στη θεωρητική δομή του αλγορίθμου αλλά στην υλοποίηση που χρησιμοποίησαν οι συγγραφείς για τη δοκιμή του: το πρόγραμμα που κατασκεύασαν οι συγγραφείς ελέγχει ένα κείμενο εισόδου μία γραμμή ανά φορά. Σε περίπτωση που βρεθεί κάποια από τις συμβολοσειρές μέσα σε μία γραμμή, η γραμμή αυτή τυπώνεται στην έξοδο και ο αλγόριθμος προχωράει αμέσως στην επεξεργασία της επόμενης γραμμής. Έτσι, καθώς το πλήθος των συμβολοσειρών γίνεται πολύ μεγάλο παρουσιάζεται το φαινόμενο οι περισσότερες γραμμές του κειμένου να περιέχουν κάποια συμβολοσειρά κοντά στην αρχή τους με αποτέλεσμα η αναζήτηση στις γραμμές αυτές να τελειώνει πολύ γρήγορα.

Πάντως, ακόμη και με το αρχικά παράδοξο φαινόμενο της εικόνας *3στ*, παρατηρούμε και στην πράξη δύο πλευρές της συμπεριφοράς του αλγορίθμου τις οποίες θα μπορούσαμε διαισθητικά να υποψιαστούμε. Πρώτον, ότι η ταχύτητα του αλγορίθμου αυξάνεται καθώς μεγαλώνει το ελάχιστο μήκος συμβολοσειράς και, δεύτερον, ότι η ταχύτητά του μειώνεται καθώς αυξάνεται ο αριθμός των συμβολοσειρών. Παρόλα αυτά, ο αλγόριθμος Wu-Manber αποτελεί την, μέχρι τώρα, καλύτερη λύση σε περίπτωση που θέλουμε να αναζητήσουμε ένα μεγάλο πλήθος συμβολοσειρών μέτριου ή μεγάλου μήκους.

3.4 Σύνοψη και εμπειρική σύγκριση.

Στο κεφάλαιο αυτό παρουσιάστηκαν οι τρεις αλγόριθμοι αναζήτησης συμβολοσειρών που θεωρήσαμε ως κατάλληλους για χρήση σε ένα σύστημα ανίχνευσης εισβολών δικτύου. Καθώς οι τρεις αυτοί αλγόριθμοι διαφέρουν σημαντικά στον τρόπο

λειτουργίας τους, εξετάζοντας τη θεωρία τους μπορούμε εύκολα να διακρίνουμε συγκεκριμένες περιπτώσεις για τις οποίες ο ένας από τους αλγόριθμους θα υπερτερεί των άλλων.

Ο αλγόριθμος Aho-Corasick θα είναι σίγουρα ο καλύτερος σε περίπτωση που το ελάχιστο μήκος σε ένα σύνολο συμβολοσειρών είναι πολύ μικρό, δηλαδή πολύ κοντά στη μονάδα. Οι άλλοι δύο αλγόριθμοι εκτελούν μετακινήσεις ώστε να αποφύγουν περιττούς ελέγχους χαρακτήρων, όμως το μέγεθος της μετακίνησης σε κάθε περίπτωση δεν μπορεί να υπερβαίνει το ελάχιστο μήκος συμβολοσειράς. Έτσι, σε περιπτώσεις που το ελάχιστο μήκος συμβολοσειράς είναι πολύ μικρό, οι άλλοι αλγόριθμοι εκτελούν περισσότερες από n συγκρίσεις, όπου n το μήκος του ελεγχόμενου κειμένου που είναι και το όριο για τον αλγόριθμο Aho-Corasick.

Για τον Boyer-Moore, η μόνη περίπτωση να είναι γρηγορότερος από τους υπόλοιπους είναι όταν θέλουμε να αναζητήσουμε μόνο μία συμβολοσειρά.

Τέλος, ο Wu-Manber θα είναι ο πλέον κατάλληλος στις περιπτώσεις που το ελάχιστο μήκος των συμβολοσειρών είναι σχετικά μεγάλο. Σίγουρα δεν θα είναι κατάλληλος για ελάχιστο μήκος 3 ή μικρότερο. Ας θεωρήσουμε την περίπτωση με ελάχιστο μήκος συμβολοσειρών ίσο με 3 και με τον αριθμό B των χαρακτήρων κατάληξης που ελέγχουμε είναι ίσο με 2: αφού το B είναι 2, η μέγιστη μετακίνηση θα είναι κατά 3+1-2 = 2 θέσεις, επομένως θα γίνονται τουλάχιστο n έλεγχοι σε ένα κείμενο μήκους n, κόστος μεγαλύτερο από εκείνο του Aho-Corasick. Προφανώς, για μικρότερη τιμή ελάχιστου μήκους η ταχύτητα του Wu-Manber γίνεται ακόμα χειρότερη.

Οι παρατηρήσεις μας αυτές μπορούν να μας βοηθήσουν στην περίπτωση που θα θέλουμε να επιλέξουμε τον καλύτερο αλγόριθμο για την αναζήτηση ενός συγκεκριμένου συνόλου συμβολοσειρών. Προκειμένου, όμως, να προσδιοριστεί μία πιο συγκεκριμένη μέθοδος επιλογής αλγορίθμου χρειάζονται εκτεταμένοι πειραματισμοί.

Κεφάλαιο 4.

Υλοποίηση των αλγορίθμων αναζήτησης συμβολοσειρών.

Στον πυρήνα του TUCNIDS βρίσκονται οι αλγόριθμοι αναζήτησης, με τους οποίους γίνεται ο έλεγχος των εισερχόμενων πακέτων για ύποπτο περιεχόμενο. Καθώς η αναζήτηση του περιεχομένου καταλαμβάνει σημαντικό ποσοστό του χρόνου εκτέλεσης, είναι απαραίτητο οι αλγόριθμοι αναζήτησης να είναι όσο το δυνατόν γρηγορότεροι.

Στο κεφάλαιο που ακολουθεί περιγράφεται ο τρόπος με τον οποίο υλοποιήθηκαν οι τρεις αλγόριθμοι αναζήτησης που χρησιμοποιούνται από το TUCNIDS, ενώ αναλύονται και οι διάφορες σχεδιαστικές επιλογές που έγιναν στον κάθε ένα.

4.1 Υλοποίηση του Aho-Corasick.

Ο αλγόριθμος Aho-Corasick ήταν ο πρώτος που υλοποιήσαμε. Καθώς πρόκειται για τον πιο περίπλοκο από τους τρεις αλγορίθμους, η υλοποίησή του χρειάστηκε να περάσει από αρκετά στάδια εξέλιξης μέχρι την τελική μορφή. Σε κάθε εξέλιξη του αλγορίθμου ο αποκλειστικός στόχος μας ήταν η βελτίωση της ταχύτητας του αλγορίθμου.

Από την πρώτη μέχρι την τελευταία υλοποίηση χρησιμοποιήθηκε μια λογική δύο επιπέδων σχετικά με τον τρόπο κατασκευής της μηχανής αναζήτησης.

Στο πρώτο επίπεδο, το γενικότερο, χρησιμοποιήθηκαν πολύπλοκες και λεπτομερείς δομές για την αναπαράσταση της μηχανής πεπερασμένων καταστάσεων, των κόμβων και των υπολοίπων στοιχείων του αλγορίθμου. Στο στάδιο αυτό γίνεται η προσθήκη των συμβολοσειρών αναζήτησης και η κατασκευή της μηχανής πεπερασμένων καταστάσεων που θα μπορεί να εκτελεί την επιθυμητή αναζήτηση.

Για το δεύτερο επίπεδο χρησιμοποιούμε τη μηχανή πεπερασμένων καταστάσεων που κατασκευάστηκε προηγουμένως για να εξάγουμε την απλούστερη δυνατή μηχανή η οποία μπορεί να εκτελεί την ισοδύναμη αναζήτηση. Αυτή η απλούστερη μηχανή είναι που θα χρησιμοποιηθεί αργότερα για την εξέταση των πακέτων καθώς θα είναι γρηγορότερη από την ισοδύναμη αλλά πιο πολύπλοκη αρχική ΜΠΚ.

Ο λόγος που έγινε η συγκεκριμένη επιλογή είναι για να μπορέσουμε να συνδυάσουμε την ευκολία διαχείρισης των δεδομένων και της αυξημένες δυνατότητες που μας προσφέρει ένα πολύπλοκο μοντέλο με την ταχύτητα ενός απλούστερου μοντέλου. Επιπλέον, με το διαχωρισμό σε δύο επίπεδα γίνεται ευκολότερη η τροποποίηση του αλγόριθμου με στόχο την απόδοση: καθώς το πρώτο επίπεδο αναλαμβάνει τον φόρτο της δημιουργίας της μηχανής πεπερασμένων καταστάσεων, ακριβώς όπως αυτή περιγράφεται από τον αλγόριθμο Aho-Corasick, μπορούμε απλά να αλλάζουμε τον τρόπο μετάφρασης της πολύπλοκης ΜΠΚ σε απλή και να αποκτάμε διαφορετικές υλοποιήσεις με την ίδια λειτουργικότητα.

4.1.1 Διαφορετικές υλοποιήσεις.

Όπως αναφέρθηκε προηγουμένως, στη λογική δύο επιπέδων που χρησιμοποιήσαμε το πρώτο επίπεδο παρέμεινε ουσιαστικά αμετάβλητο ανά τις διάφορες υλοποιήσεις που δοκιμάστηκαν και αυτό που άλλαζε ήταν ο τρόπος εξαγωγής του δεύτερου επιπέδου από το πρώτο. Για το λόγο αυτό θα αναφερθούμε μόνο στις διαφορετικές υλοποιήσεις του δευτέρου επιπέδου. Άλλωστε, η μηχανή πεπερασμένων καταστάσεων του δεύτερου επιπέδου θέλουμε να είναι όσο γίνεται πιο γρήγορη καθώς αυτή χρησιμοποιείται για την εξέταση των πακέτων. Κάθε υλοποίηση που αναφέρουμε βρήκαμε ότι ήταν γρηγορότερη από τις προηγούμενες. Άλλες υλοποιήσεις που δοκιμάσαμε αλλά δεν βελτίωναν την ταχύτητα δεν αναφέρονται.

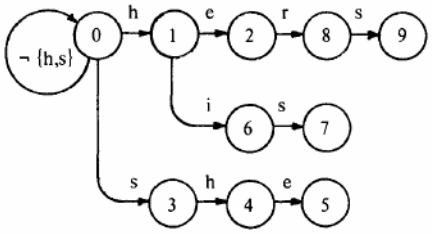
Στο σημείο αυτό πρέπει να αναφερθεί πως οι υλοποιήσεις που έγιναν δεν βρέθηκαν σε κάποια βιβλιογραφία αλλά προέκυψαν από δικές μας ιδέες. Στη βιβλιογραφία στηριχτήκαμε μόνο για τη θεωρία του αλγορίθμου. Παρόλο που μπορεί να

υπάρχουν συγκεκριμένες υλοποιήσεις σε δημοσιεύσεις ή στο διαδίκτυο, οι υλοποιήσεις που κάναμε προέκυψαν καθαρά από τη δική μας λογική και διαίσθηση.

Βασική υλοποίηση: Η συγκεκριμένη υλοποίηση έγινε με την πρόθεση να τεθούν τα θεμέλια για τις εξελίξεις που θα ακολουθούσαν και όχι με στόχο να επιτευχθούν μεγάλες ταχύτητες. Πρακτικά εκτελούσαμε μόνο μια μικρή περικοπή πληροφοριών από το πρώτο επίπεδο μέχρι το δεύτερο.

Η κυρίαρχη δομή στην υλοποίηση αυτή είναι η δομή που αντιπροσώπευε μία κατάσταση της μηχανής πεπερασμένων καταστάσεων. Για το σύνολο μίας οποιασδήποτε πεπερασμένης μηχανής χρειάζεται ένας συγκεκριμένος αριθμός καταστάσεων, έστω N, οι οποίες είναι αριθμημένες από 0 μέχρι N-1 και αποθηκεύονται σε έναν μονοδιάστατο πίνακα καταστάσεων. Κατά σύμβαση, η κατάσταση 0 ισοδυναμεί πάντα με την αρχική κατάσταση.

Κάθε κατάσταση περιέχει και έναν πίνακα μεταβάσεων σε επόμενες καταστάσεις. Στην υλοποίηση αυτή, κάθε μετάβαση περιλαμβάνει δύο στοιχεία: τον χαρακτήρα εισόδου (τιμή ενός byte) για τον οποίο γίνεται η συγκεκριμένη μετάβαση και τον αριθμό της επόμενης κατάστασης (δηλαδή τη θέση της μέσα στον πίνακα). Για να γίνει αυτό πιο εύκολα αντιληπτό, ας θεωρήσουμε το παράδειγμα των τεσσάρων ζητούμενων λέξεων “he”, “she”, “his” και “hers”. Για το συγκεκριμένο παράδειγμα, το δέντρο μεταβάσεων που προκύπτει μετά από την εισαγωγή των συμβολοσειρών φαίνεται στην εικόνα *4α*, ενώ οι μεταβάσεις από κάθε κατάσταση της τελικής μηχανής πεπερασμένων καταστάσεων βρίσκονται στην εικόνα *4β*.



(a) Goto function.

Εικόνα 4α) Το δέντρο μεταβάσεων ύστερα από την εισαγωγή των συμβολοσειρών (πριν από τη μετατροπή σε ντετερμινιστικό αυτόματο)

		input symbol	next state
	state 0:	h	1
		s	3
		·	0
	state 1:	e	2
		i	6
		h	1
		s	3
		·	0
state 9: state 7:	state 3:	h	4
		s	3
		·	0
state 5:	state 2:	r	8
		h	1
		s	3
		·	0
	state 6:	s	7
		h	1
		·	0
	state 4:	e	5
		i	6
		h	1
		s	3
		·	0
	state 8:	s	9
		h	1
		·	0

Εικόνα 4β) οι μεταβάσεις της μηχανής πεπερασμένων καταστάσεων για τον εντοπισμό των λέξεων "he", "she", "his" και "hers".

Από την εικόνα 4β διαπιστώνουμε πως η κατάσταση 0 θα έχει ένα πίνακα με δύο μεταβάσεις, τις (*h*, 1) και (*s*,3), η κατάσταση 1 θα έχει ένα πίνακα τεσσάρων μεταβάσεων και ούτω καθεξής.

Στη μηχανή πεπερασμένων καταστάσεων που δημιουργείται υπάρχουν συγκεκριμένες καταστάσεις, τις οποίες ονομάζουμε «τελικές», που ισοδυναμούν με την αναγνώριση μίας ή περισσοτέρων από τις ζητούμενες συμβολοσειρές. Στο παράδειγμα που αναφέραμε οι τελικές καταστάσεις είναι η 2 (αντιστοιχεί στη λέξη "he"), η κατάσταση 5 (αντιστοιχεί στις λέξεις "he" και "she"), η 7 ("his") και η 9 ("hers"). Σε μια μηχανή N καταστάσεων οι k από αυτές, όπου $k \leq N$, θα είναι τελικές καταστάσεις. Για λυθεί το πρόβλημα της αντιστοίχησης καταστάσεων με συμβολοσειρές, δημιουργούμε έναν πίνακα k+1 θέσεων τον οποίο ονομάζουμε *signatures*. Οι θέσεις του *signatures* είναι αριθμημένες από 0 μέχρι k και κάθε θέση του πίνακα αυτού να αντιστοιχεί σε μία λίστα από συμβολοσειρές. Στη συνέχεια ορίζουμε σε κάθε κόμβο μία ακέραια μεταβλητή, η οποία έστω ότι ονομάζεται *final*. Σε περίπτωση που η *final* μίας κατάστασης έχει τιμή ίση με 0, τότε η κατάσταση αυτή δεν είναι τελική. Διαφορετικά, δηλαδή εάν η *final* μίας κατάστασης δεν είναι μηδενική αλλά κυμαίνεται ανάμεσα στο 1 και το k, τότε η κατάσταση είναι τελική και οι συμβολοσειρές που της αντιστοιχούν περιέχονται στη λίστα που βρίσκεται στη θέση *final* του πίνακα *signatures*.

Στο σημείο αυτό θα περιγράψουμε συνοπτικά τον τρόπο λειτουργίας της μηχανής αναγνώρισης. Κάθε φορά που εξετάζουμε ένα κείμενο εισόδου ξεκινάμε από την κατάσταση 0 και εξετάζουμε πρώτο χαρακτήρα του κειμένου. Στην κατάσταση που

βρισκόμαστε ελέγχουμε σειριακά τις θέσεις του πίνακα μεταβάσεων, με στόχο να βρούμε τη μετάβαση για το συγκεκριμένο χαρακτήρα εισόδου. Σε περίπτωση που βρεθεί μετάβαση για το συγκεκριμένο χαρακτήρα, η οποία έστω ότι οδηγεί στην κατάσταση *s*, τότε η *s* γίνεται η τρέχουσα κατάσταση του συστήματος. Εάν εξετάσουμε όλες τις μεταβάσεις και δε βρούμε κάποια να ταιριάζει, τότε πηγαίνουμε πίσω στην αρχική κατάσταση. Όποια από τις δύο περιπτώσεις και να ισχύει, στη συνέχεια εξετάζουμε μήπως η κατάσταση στην οποία βρισκόμαστε αντιστοιχεί σε ζητούμενες συμβολοσειρές, δηλαδή εάν η μεταβλητή *final* της τρέχουσας κατάστασης είναι μη μηδενική, οπότε θα πρέπει να εμφανίσουμε τα κατάλληλα δεδομένα στην έξοδο. Στη συνέχεια προχωράμε στον επόμενο χαρακτήρα του κειμένου και επαναλαμβάνουμε τη διαδικασία μέχρι να εξετάσουμε με τον τρόπο αυτό ολόκληρο το κείμενο.

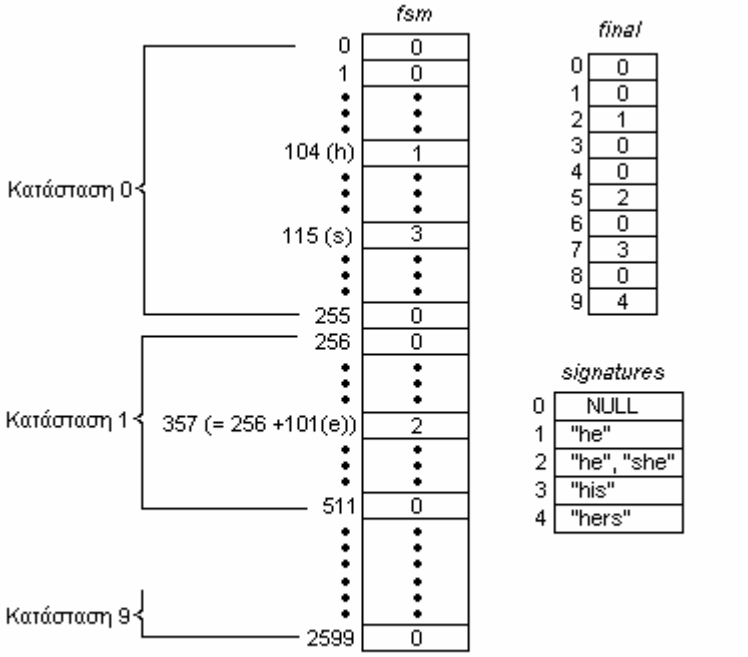
Βελτιστοποίηση 1: Είναι προφανές πως ο σειριακός έλεγχος των μεταβάσεων σε μία κατάσταση δεν είναι καθόλου αποδοτικός. Για να επιταχυνθεί η διαδικασία φροντίζουμε τα στοιχεία του πίνακα μεταβάσεων να είναι ταξινομημένα ως προς την τιμή του χαρακτήρα εισόδου. Με τον τρόπο αυτό μπορούμε να χρησιμοποιούμε έναν αλγόριθμο δυαδικής αναζήτησης (binary search) με παράμετρο το χαρακτήρα του κειμένου για να βρίσκουμε πιο γρήγορα την επόμενη μετάβαση. Τα υπόλοιπα στοιχεία της πρώτης υλοποίησης παραμένουν ως έχουν.

Βελτιστοποίηση 2: Καθώς ο αριθμός των ζητούμενων συμβολοσειρών είναι συχνά μεγάλος, για παράδειγμα μεγαλύτερος από 100, ο αριθμός των μεταβάσεων που ξεκινάνε από κάθε κατάσταση μπορεί να είναι αρκετά μεγάλος και η αναζήτηση της κατάλληλης μετάβασης για ένα συγκεκριμένο χαρακτήρα εισόδου χρονοβόρα. Καθώς ο χαρακτήρας εισόδου ενός byte μπορεί να πάρει 256 διαφορετικές τιμές, αντικαθιστούμε τον πίνακα μεταβάσεων κάθε κατάστασης σε πίνακα 256 θέσεων με τέτοιο τρόπο ώστε κάθε θέση *c* του πίνακα να περιέχει την επόμενη κατάσταση σε περίπτωση που ο χαρακτήρας εισόδου έχει τιμή *c*.

Βελτιστοποίηση 3: Στην υλοποίηση αυτή καταργείται η χρήση μίας δομής για κάθε κατάσταση της μηχανής. Αντίθετα, χρησιμοποιείται ένας ενιαίος πίνακας μεγέθους

256*N στοιχείων, όπου N ο συνολικός αριθμός καταστάσεων και κάθε στοιχείο είναι μια ποσότητα 16 bit. Ο πίνακας αυτός κατασκευάζεται με τέτοιον τρόπο ώστε για τις μεταβάσεις από την κατάσταση s να αντιστοιχεί το μπλοκ τιμών από 256*s μέχρι 256*s+255. Τώρα πια, σε περίπτωση που κατά τη διάρκεια της εξέτασης του κειμένου βρισκόμαστε στην κατάσταση με αριθμό s και ο χαρακτήρας εισόδου έχει τιμή c, ο αριθμός της επόμενης κατάστασης βρίσκεται στη θέση (256*s)+c του ενιαίου πίνακα.

Οι πληροφορίες για την αντιστοιχία καταστάσεων και ζητούμενων συμβολοσειρών που προηγουμένως βρισκόταν στην μεταβλητή *final* κάθε κατάστασης, πλέον αποθηκεύονται σε έναν πίνακα *final* N θέσεων με παρόμοιο τρόπο όπως στις προηγούμενες υλοποιήσεις. Δηλαδή, αν η θέση s αυτού του πίνακα έχει τιμή 0 τότε η κατάσταση δεν είναι τελική, ενώ σε περίπτωση που έχει τιμή y τότε οι ζητούμενες συμβολοσειρές που αντιστοιχούν στην τελική κατάσταση s βρίσκονται στη θέση y του πίνακα *signatures*, δηλαδή του πίνακα με τις λίστες από συμβολοσειρές. Έτσι, για το παράδειγμα των συμβολοσειρών “he”, “she”, “his” και “hers”, για το οποίο ισχύουν οι μεταβάσεις της εικόνας *4β*, προκύπτουν οι πίνακες της εικόνας *4γ*.



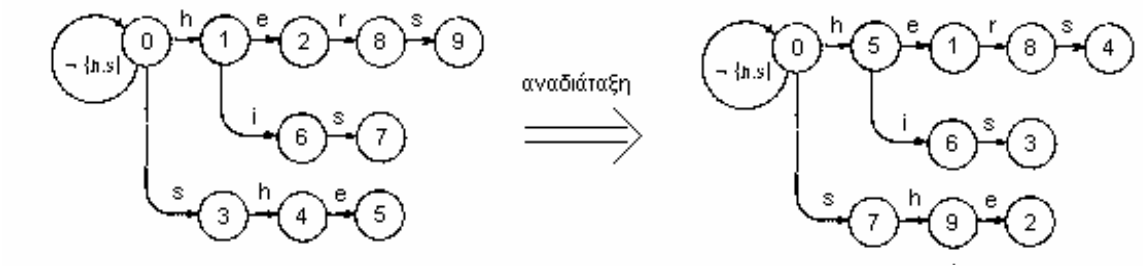
Εικόνα 4γ) Οι τρεις πίνακες που χρησιμοποιούνται από την βελτιστοποίηση 3 του Aho-Corasick.

Το γεγονός ότι χρησιμοποιούνται ποσότητες 16 bit για την αποθήκευση της επόμενης κατάστασης συνεπάγεται ότι ο μέγιστος αριθμός καταστάσεων που μπορούμε να δευθυνσιοδοτήσουμε είναι 2^{16} = 65536 καταστάσεις. Η επιλογή αυτή είναι σκόπιμη και έγινε με βάση τα πειραματικά δεδομένα. Ο αλγόριθμος αυτός χρησιμοποιείται για την επαλήθευση συνόλων από κανόνες, όπου κάθε κανόνας αντιστοιχεί και σε μια ύποπτη συμβολοσειρά που θα αναζητηθεί μέσα στα εισερχόμενα πακέτα. Από την ανάλυση και ομαδοποίηση των κανόνων, όπως αυτή περιγράφεται στις παραγράφους 5.1.1 και 5.1.2, προκύπτει οτι το μεγαλύτερο μέγεθος συνόλου κανόνων είναι περίπου 310 κανόνες, ενώ το μέσο μήκος ανά κανόνα είναι περίπου 8 bytes. Ο αριθμός καταστάσεων μίας μηχανής είναι πάντα μικρότερος η ίσος από το άθροισμα των μηκών όλων των συμβολοσειρών προς αναζήτηση, επομένως για τη μηχανή πεπερασμένων καταστάσεων που θα μπορεί να αναζητεί όλους τους κανόνες αυτού του συνόλου χρειάζονται το πολύ 310*8 = 2480 καταστάσεις, τιμή αρκετά μικρότερη από το όριο των 65536 καταστάσεων. Το όφελος της χρήσης αριθμών πλάτους 16 bit αντί για το σύνηθες πλάτος των 32 bit είναι ότι μειώνονται οι απαιτήσεις σε μνήμη και, το σημαντικότερο, αυξάνεται η τοπικότητα της cache με αποτέλεσμα σημαντικά οφέλη στην ταχύτητα.

Πέρα από τα οφέλη του μικρότερου μεγέθους, ο δεύτερος λόγος για τον οποίο αυξάνεται σημαντικά η ταχύτητα με τη συγκεκριμένη υλοποίηση είναι επειδή μειώνονται οι προσβάσεις στη μνήμη. Το γεγονός αυτό θα γίνει εμφανές αν εξετάσουμε την περίπτωση να βρισκόμαστε στην κατάσταση *s* και ο χαρακτήρας εισόδου είναι ο *c*. Στην προηγούμενη υλοποίηση ήταν απαραίτητες δύο προσβάσεις στη μνήμη σε κάθε βήμα, η πρώτη πρόσβαση για να βρούμε από τον πίνακα καταστάσεων την *s* και η δεύτερη για να βρούμε την μετάβαση από την *s* που αντιστοιχεί στον χαρακτήρα *c*. Αντίθετα, σε αυτή την υλοποίηση ο αριθμός της επόμενης κατάστασης θα βρεθεί με την πρόσβαση στη θέση (s<<8)+256 του πίνακα, όπου s<<8 η πράξη του αριστερού shift κατά 8 θέσεις η οποία ισοδυναμεί με τον πολλαπλασιασμό του *s* με 256.

Βελτιστοποίηση 4: Σε αυτή την υλοποίηση, η οποία είναι και η ταχύτερη, γίνεται μια μικρή αναδιάταξη των δεδομένων στο πρώτο επίπεδο της υλοποίησης παράλληλα με κάποιες αλλαγές στο δεύτερο επίπεδο. Συγκεκριμένα, στο πρώτο επίπεδο κάνουμε μια αναδιάταξη των καταστάσεων της ΜΠΚ έτσι ώστε κάθε μία από της k τελικές

καταστάσεις να έχει αριθμό κατάστασης ίσο με τη θέση του πίνακα *signatures* που αντιστοιχεί στη συγκεκριμένη κατάσταση. Δηλαδή, αν η κατάσταση *s* είναι τελική κατάσταση και οι συμβολοσειρές που της αντιστοιχούν βρίσκονται στη θέση *s'* του πίνακα *signatures*, τότε ανταλλάσσουμε τους αριθμούς των καταστάσεων *s* και *s'*. Από τη στιγμή σε κάθε τελική κατάσταση έχουμε αναθέσει ένα διαφορετικό αριθμό από 1 μέχρι k δεν υπάρχει πρόβλημα αλληλοεπικάλυψης καταστάσεων. Ένα παράδειγμα αναδιάταξης καταστάσεων φαίνεται στην ακόλουθη εικόνα, στην οποία γίνεται αναδιάταξη των καταστάσεων του συντακτικού δέντρου της εικόνας *4α* (*στην πραγματικότητα, η αναδιάταξη καταστάσεων δε γίνεται στο συντακτικό δέντρο αλλά στο ντετερμινιστικό αυτόματο. Απλά χρησιμοποιούμε στο παράδειγμα το συντακτικό δέντρο επειδή ο γράφος της ΜΠΚ είναι αρκετά πιο περίπλοκος. Η ιδέα παραμένει ίδια και στις δύο περιπτώσεις*).



Εικόνα 4δ) ένα παράδειγμα αναδιάταξης καταστάσεων.

Στο σημείο αυτό πρέπει να σημειωθεί το εξής: στο πρώτο επίπεδο της υλοποίησης όλες οι καταστάσεις περιέχονται και σε ένα μονοδιάστατο πίνακα καταστάσεων μεγέθους N. Η δομή που χρησιμοποιείται για την αναπαράσταση κάθε κατάστασης περιέχει μία μεταβλητή για την αποθήκευση της ετικέτας της, δηλαδή της θέσης της κατάστασης μέσα στον πίνακα. Επιπλέον η μετάβαση από μια συγκεκριμένη κατάσταση χρησιμοποιεί *δείκτη* στην επόμενη κατάσταση και *όχι τη θέση* της επόμενης κατάστασης μέσα στον πίνακα καταστάσεων. Έτσι, όταν ανταλλάσσουμε τις θέσεις και τις ετικέτες των s και s', οι μεταβάσεις που οδηγούσαν στις s και s' παραμένουν σωστές και δείχνουν στις νέες θέσεις των αντίστοιχων καταστάσεων. Η λειτουργία της μηχανής παραμένει απολύτως ίδια.

Από την αναδιάταξη αυτή προκύπτει στο πρώτο επίπεδο μια μηχανή πεπερασμένων καταστάσεων της οποίας η κατάσταση 0 είναι η αρχική κατάσταση, οι

καταστάσεις 1 ως k είναι τελικές καταστάσεις και όλες οι υπόλοιπες είναι ενδιάμεσες. Δεδομένης της μορφή αυτής μπορούμε στο δεύτερο επίπεδο υλοποίησης να απαλείψουμε τον πίνακα *final* και να γλιτώσουμε από το κόστος της πρόσβασης σε αυτόν σε κάθε κύκλο εκτέλεσης του αλγορίθμου. Πλέον, συγκρίνοντας απλά τον αριθμό της τρέχουσας κατάστασης με την σταθερή τιμή k μπορούμε να ξέρουμε αν μια κατάσταση είναι τελική η όχι. Έτσι, όταν κατά τη διάρκεια της αναζήτησης ο αριθμός της τρέχουσας κατάστασης είναι μεγαλύτερος από 0 και μικρότερος ή ίσος με k, τότε γνωρίζουμε οτι βρισκόμαστε σε μια τελική κατάσταση και ανατρέχουμε στην ίδια θέση του πίνακα με τις λίστες συμβολοσειρών ώστε να τις εμφανίσουμε στην έξοδο.

Καθώς η συγκεκριμένη υλοποίηση είναι αυτή που χρησιμοποιήθηκε τελικά, θα παραθέσουμε στο σημείο αυτό τον ψευδοκώδικα που περιγράφει τον αλγόριθμο αναζήτησης (Σημείωση: ο παρακάτω ψευδοκώδικας καθώς και όλοι οι υπόλοιποι που παρουσιάζονται στο κεφάλαιο αυτό αφορούν την αναζήτηση με διαχωρισμό κεφαλαίων και μικρών γραμμάτων).

Αλγόριθμος linear_AC: Υλοποίηση τυπικού αλγόριθμου αναζήτησης Aho-Corasick.

Είσοδος: Πίνακας *fsm* που αντιπροσωπεύει τη μηχανή πεπερασμένων καταστάσεων, πίνακας *signatures* με τις λίστες από συμβολοσειρές για κάθε τελική κατάσταση, κείμενο εισόδου *text*.

Έξοδος: Οι συμβολοσειρές που βρέθηκαν μέσα στο κείμενο και η θέση κάθε στιγμιοτύπου.

```
begin
        i ← 0
        state ← 0
fast_loop:
        while ( i ≤ length(text) ) do
        begin
                c ← text[i]
                state ← fsm[c]
slow_loop:
                i ← i + 1
                if state == 0 then goto fast_loop
                else if state ≤ final_state_count then output(signatures[state], i )
                if (i > length(text) ) then break
                c ← text[i]
                state ← fsm[ (state << 8) + c ]
```

```
            goto slow_loop
        end
end
```

Ο βρόχος *fast_loop* περιλαμβάνει τον κώδικα μέχρι και το πρώτο **if** και εκτελείται όσο παραμένουμε στην κατάσταση 0. Αντίστοιχα, ο βρόχος slow_loop περιλαμβάνει τον κώδικα μέχρι το "**goto** *slow_loop*" και εκτελείται για τις υπόλοιπες καταστάσεις. Ο διαχωρισμός σε δύο βρόχους έγινε επειδή μας επιτρέπει να αποφύγουμε μία πράξη shift και μία πρόσθεση κάθε φορά που βρισκόμαστε στην αρχική κατάσταση 0. Ανάλογα με την περίπτωση, η ΜΠΚ μπορεί να βρίσκεται αρκετές φορές στην κατάσταση 0.

4.1.2 Εναλλακτική υλοποίηση - Aho-Corasick μαζί με Boyer-Moore.

Η υλοποίηση αυτή προέκυψε από τροποποίηση της βελτιστοποίησης 4 του Aho-Corasick. Η νέα υλοποίηση αποτελεί έναν υβριδικό αλγόριθμο Aho-Corasick και Boyer-Moore, καθώς χρησιμοποιεί έναν πίνακα μετατοπίσεων «κακού χαρακτήρα» αντίστοιχο με εκείνο του αλγορίθμου Boyer-Moore με σκοπό την υπερπήδηση κάποιων χαρακτήρων χωρίς εξέταση.

Πρέπει να αναφέρουμε πως στην βιβλιογραφία βρέθηκαν αναφορές [19] για υβριδικούς αλγορίθμους Aho-Corasick και Boyer-Moore, όμως εμείς δεν βασιστήκαμε σε κάποιον από αυτούς τους υβριδικούς αλγορίθμους. Η συγκεκριμένη υλοποίηση μας προέκυψε ως αποτέλεσμα της εφαρμογής της ιδέας του αλγορίθμου Boyer-Moore στον Aho-Corasick.

Θα περιγράψουμε πρώτα την κατασκευή του πίνακα μετατοπίσεων. Έχουμε το σύνολο των n ζητούμενων συμβολοσειρών και έστω ότι το μήκος της μικρότερης συμβολοσειράς μέσα στο σύνολο είναι m. Ο πίνακας μετατοπίσεων έχει 256 θέσεις, όσες και οι πιθανές τιμές των χαρακτήρων εισόδου, και για την κατασκευή του μας ενδιαφέρουν μόνο οι m αρχικοί χαρακτήρες όλων των συμβολοσειρών. Σε κάθε θέση c του πίνακα μετατοπίσεων αποθηκεύουμε την τιμή m-j, όπου j η δεξιότερη θέση μέσα στους πρώτους m χαρακτήρες όλων των συμβολοσειρών στην οποία εμφανίζεται ο χαρακτήρας με τιμή c. Συνεπώς, αν ο c εμφανίζεται στη θέση m σε κάποια συμβολοσειρά η τιμή μετατόπισης θα είναι 0, αν δεν εμφανίζεται στους πρώτους m χαρακτήρες σε

καμία από τις συμβολοσειρές η μετατόπιση είναι *m*, αν εμφανίζεται μόνο στην πρώτη θέση η μετατόπιση είναι *m*-1 και λοιπά.

Προκειμένου να αξιοποιηθεί ο πίνακας αναζήτησης είναι απαραίτητη η τροποποίηση του αλγόριθμου. Με τον νέο αλγόριθμο, η αναζήτηση ξεκινά από τον *m*-οστό χαρακτήρα του κειμένου. Στο πρώτο βήμα εξετάζουμε την τιμή του πίνακα μετατόπισης για τον χαρακτήρα *c* του κειμένου στην θέση που βρισκόμαστε. Αν η μετατόπιση είναι μη-μηδενική, έστω *j*, τότε μετακινούμαστε δεξιά κατά *j* θέσεις και πηγαίνουμε πίσω στο πρώτο βήμα του αλγορίθμου.

Σε περίπτωση, όμως, που η μετατόπιση είναι μηδενική τότε τα πράγματα είναι πιο περίπλοκα καθώς ίσως να έχει βρεθεί κάποια συμβολοσειρά. Αρχικά πηγαίνουμε *πίσω*, δηλαδή αριστερά, κατά *m*-1 θέσεις καθώς στο σημείο εκείνο θα βρίσκεται η αρχή της πιθανής συμβολοσειράς. Από τη νέα θέση ξεκινάμε την εφαρμογή της μηχανής πεπερασμένων καταστάσεων του αλγόριθμου Aho-Corasick, δηλαδή ξεκινάμε από την αρχική κατάσταση και βρίσκουμε την επόμενη κατάσταση για τον χαρακτήρα που βρίσκεται στη συγκεκριμένη θέση του κειμένου. Όσο η νέα κατάσταση είναι διαφορετική από την αρχική κατάσταση της μηχανής (δηλαδή την κατάσταση 0) μετακινούμαστε δεξιά κατά μια θέση και συνεχίζουμε τον έλεγχο. Σε περίπτωση που βρεθούν κάποιες συμβολοσειρές τότε αυτές εμφανίζονται στην έξοδο, ενώ η λειτουργία της μηχανής πεπερασμένων καταστάσεων συνεχίζεται χωρίς διακοπή. Η λειτουργία της ΜΠΚ σταματάει μόνο όταν η μηχανή επιστρέψει στην αρχική της κατάσταση, στην οποία περίπτωση μετακινούμαστε δεξιά κατά *m* θέσεις και επιστρέφουμε στο πρώτο βήμα του αλγορίθμου, δηλαδή στο βήμα που χρησιμοποιείται ο πίνακας μετατόπισης.

Ακολουθεί ο ψευδοκώδικας που περιγράφει την αναζήτηση συμβολοσειρών μέσω του υβριδικού αλγορίθμου.

Αλγόριθμος AC_BM: Υλοποίηση υβριδικού αλγόριθμου αναζήτησης Aho-Corasick και Boyer-Moore.

Είσοδος: Πίνακας *fsm* που αντιπροσωπεύει τη μηχανή πεπερασμένων καταστάσεων, πίνακας *signatures* με τις λίστες από συμβολοσειρές για κάθε τελική κατάσταση, πίνακας μετατοπίσεων *shift*, ελάχιστο μήκος συμβολοσειρών *m*, κείμενο εισόδου *text*.

Έξοδος: Οι συμβολοσειρές που βρέθηκαν μέσα στο κείμενο και η θέση κάθε στιγμιοτύπου.

begin

$i \leftarrow 0$

$state \leftarrow 0$

```
fast_loop:
        while ( i ≤ length(text) ) do
        begin
                c ← text[i]
                if shift[c] > 0 then
                begin
                        i ← i + shift[c]
                        goto fast_loop
                end
                i ← i – m + 1
                c ← text[i]
                state ← fsm[c]
slow_loop:
                if state == 0 then
                begin
                        i ← i + m
                        goto fast_loop
                end
                else if state ≤ final_state_count then output(signatures[state], i + 1)
                i ← i + 1
                if (i > length(text) ) then break
                c ← text[i]
                state ← fsm[ (state << 8) + c ]
                goto slow_loop
        end
end
```

Ο διαχωρισμός σε δύο βρόχους γίνεται για βελτίωση της απόδοσης. Ο βρόχος *fast_loop* περιλαμβάνει τον κώδικα μέχρι το πρώτο "**goto** *fast_loop*". Αντιστοιχεί στην περίπτωση όπου ο πίνακας *shift* μας επιτρέπει να υπερπηδήσουμε χαρακτήρες. Ο βρόχος slow_loop περιλαμβάνει τον κώδικα μέχρι το "**goto** *slow_loop*" και αντιστοιχεί στην επεξεργασία ενός χαρακτήρα από την μηχανή πεπερασμένων καταστάσεων.

Η χρήση ενός πίνακα μετατοπίσεων τύπου Boyer-Moore μπορεί να επιταχύνει σημαντικά τον αλγόριθμο Aho-Corasick, καθώς στην καλύτερη περίπτωση μπορούμε να εξετάσουμε ολόκληρο πακέτο μήκους *n* εκτελώντας μόνο *n/m* ελέγχους χαρακτήρων.

Ακόμα και σε περίπτωση που m=2 το κέρδος είναι σημαντικό από τη στιγμή που ο κανονικός αλγόριθμος Aho-Corasick εκτελεί πάντα n ελέγχους.

Δυστυχώς προκύπτουν και σημαντικά μειονεκτήματα από τη χρήση του πίνακα μετατοπίσεων. Αρχικά, ο αλγόριθμος αυτός είναι καλύτερος από τον τυπικό Aho-Corasick μόνο για περιορισμένο πλήθος συμβολοσειρών. Αυτό συμβαίνει επειδή όσο περισσότερες συμβολοσειρές υπάρχουν μέσα στο σύνολο τόσο αυξάνονται οι πιθανότητες να έχουμε μηδενική ή μοναδιαία μετατόπιση, με αποτέλεσμα η ταχύτητα του αλγορίθμου να μειώνεται σημαντικά. Όπως παρατηρήθηκε από τις πειραματικές μετρήσεις, ο αλγόριθμος αυτός είναι καλύτερος από τους εναλλακτικούς μόνο όταν το πλήθος των συμβολοσειρών σε ένα σύνολο είναι μικρότερο περίπου του 40 ή του 50, ανάλογα με την περίσταση. Το δεύτερο μειονέκτημα του ως προς τον κλασσικό Aho-Corasick είναι ότι, στη χειρότερη περίπτωση, ο αλγόριθμος με τον πίνακα μετατόπισης εκτελεί 2n συγκρίσεις χαρακτήρων εκεί που ο κλασσικός χρειάζεται n συγκρίσεις.

Ένα απλό παράδειγμα χειρότερης απόδοσης του συνδυασμένου αλγορίθμου είναι το ακόλουθο. Έστω ότι έχουμε μόνο μία συμβολοσειρά προς αναζήτηση, την abc, και το κείμενο εισόδου είναι το cccccc…ccc. Στην πρώτη επανάληψη του αλγορίθμου εξετάζεται ο χαρακτήρας στην θέση 3 και επειδή η μετατόπιση για το χαρακτήρα ‘c’ είναι μηδέν πηγαίνουμε 2 θέσεις πίσω, στον χαρακτήρα 1, και εφαρμόζουμε τη μηχανή πεπερασμένων καταστάσεων. Για χαρακτήρα εισόδου c η μηχανή παραμένει στην αρχική κατάσταση, επομένως ο έλεγχος απέτυχε και μετακινούμαστε δεξιά κατά 3 θέσεις και πηγαίνουμε στην 4. Δηλαδή μέχρι τώρα έχουμε τις εξής προσβάσεις ανά χαρακτήρα:

```
cccccc…ccc
101000…000
```

Εξετάζουμε τον πίνακα μετατόπισης για τη θέση 4 και βρίσκουμε μηδέν, οπότε πηγαίνουμε στη θέση 2. Εξετάζουμε το χαρακτήρα, ο έλεγχος αποτυγχάνει και μετακινούμαστε δεξιά κατά τρεις θέσεις, στην θέση 5. Η μετατόπιση γαι τη θέση 5 είναι μηδέν, πηγαίνουμε πίσω στη θέση 3 και από εκεί, λόγω της αποτυχίας του ελέγχου, μετακινούμαστε στη θέση 6. Στο σημείο αυτό έχουν γίνει οι εξής προσβάσεις:

```
cccccc…ccc
112110…000
```

Η διαδικασία αυτή θα επαναλαμβάνεται μέχρι να φτάσουμε στο τέλος του πακέτου και προκύπτει ότι σχεδόν σε όλους τους χαρακτήρες του πακέτου θα γίνουν δύο προσβάσεις, επομένως το κόστος χειρότερης περίπτωσης του αλγορίθμου είναι $2n$. Παρόλα αυτά, στη γενική περίπτωση μπορούμε να περιμένουμε πως θα υπάρχει ένας βαθμός τυχαιότητας στους χαρακτήρες που εμφανίζονται μέσα σε ένα πακέτο, επομένως ο αλγόριθμος αυτός θα είναι πιο γρήγορος από τον κλασσικό Aho-Corasick.

4.1.3 Υποστήριξη αναζήτησης χωρίς διαφοροποίηση μικρών-κεφαλαίων.

Στις περισσότερες περιπτώσεις αναζήτησης μας ενδιαφέρει να βρίσκουμε συμβολοσειρές μέσα σε ένα κείμενο εισόδου οι οποίες είναι ακριβώς ίδιες με τις συμβολοσειρές προς αναζήτηση. Όταν ισχύει αυτό λέμε ότι εκτελούμε αναζήτηση με διαχωρισμό κεφαλαίων και μικρών γραμμάτων ή αλλιώς case sensitive αναζήτηση συμβολοσειράς. Υπάρχει, όμως, το ενδεχόμενο να θέλουμε να αναζητήσουμε ορισμένες συμβολοσειρές του συνόλου χωρίς να κάνουμε διαχωρισμό μεταξύ κεφαλαίων και μικρών γραμμάτων. Στη δεύτερη περίπτωση λέμε ότι έχουμε μια case insensitive αναζήτηση συμβολοσειράς. Για παράδειγμα, ας θεωρήσουμε ότι η συμβολοσειρά προς αναζήτηση είναι η “Foo”. Στην περίπτωση που εκτελούμε αναζήτηση με διαχωρισμό, μας ενδιαφέρει μόνο να βρούμε σε ποιες θέσεις του κειμένου εμφανίζεται η συμβολοσειρά “Foo”. Εάν, όμως, θέλουμε η αναζήτηση να γίνει χωρίς διαχωρισμό, τότε μας ενδιαφέρει να βρούμε τις θέσεις του κειμένου όπου εμφανίζεται η συμβολοσειρά “Foo”, η “fOo”, η “FOO”, και γενικά τις συμβολοσειρές που προκύπτουν από όλους τους δυνατούς συνδυασμούς κεφαλαίων και μικρών γραμμάτων.

Κάθε αλγόριθμος αναζήτησης από τη φύση του εκτελεί μόνο αναζητήσεις με διαχωρισμό ανάμεσα σε κεφαλαία και μικρά γράμματα. Παρόλα αυτά, στα συστήματα ανίχνευσης εισβολών δικτύου είναι απαραίτητη η υποστήριξη αναζήτησης για συμβολοσειρές χωρις διαφοροποίηση κεφαλαίων-μικρών.

Ας υποθέσουμε ότι έχουμε ένα σύνολο από συμβολοσειρές, εκ των οποίων ορισμένες θέλουμε να τις αναζητήσουμε με case sensitive τρόπο, ενώ τις υπόλοιπες με

case insensitive τρόπο. Γενικά, για κάθε αλγόριθμο η υποστήριξη case insensitive αναζητήσεων μπορεί να γίνει με τις εξής δύο μεθοδολογίες.

1) Η αναζήτηση κάθε συμβολοσειράς γίνεται με διαχωρισμό κεφαλαίων-μικρών. Σε περίπτωση που έχουμε κάποια συμβολοσειρά *χωρίς* διαχωρισμό, τότε ο αλγόριθμος εκτελεί αναζήτηση *με* διαχωρισμό για κάθε πιθανή παραλλαγή της συμβολοσειράς που προκύπτει από διαφορετικούς συνδυασμούς κεφαλαίων και μικρών. Ας θεωρήσουμε το παράδειγμα αναζητησης της συμβολοσειρας "foo" χωρίς διαχωρισμό κεφαλάιων-μικρών. Σύμφωνα με αυτή τη μεθοδολογία, ο αλγόριθμος θα εκτελεί αναζήτηση *με* διαχωρισμό κεφαλαίων-μικρών των συμβολοσειρών "foo", "foO", "fOo", "fOO", "Foo", "FoO", "FOo" και "FOO". Το πλεονέκτημα αυτής της μεθόδου είναι ότι δεν απαιτούνται επιπλέον εντολές κατά τη διάρκεια της αναζήτησης με τον αλγόριθμο. Υπάρχει, όμως, το μειονέκτημα ότι αυξάνονται σημαντικά οι απαιτήσεις σε μνήμη: συχνά η αύξηση είναι εκθετική ως προς το μήκος των case insensitive συμβολοσειρών.

2) Σε περίπτωση που υπάρχει έστω και μια συμβολοσειρά χωρίς διαφοροποίηση μέσα στο σύνολο τότε η αναζήτηση όλων των συμβολοσειρών γίνεται με χωρίς διαφοροποίηση και σε περίπτωση που ίσως έχουμε βρεί μια συμβολοσειρά με διαφοροποίηση εξετάζουμε ξανά τους χαρακτήρες του κειμένου για επαληθεύση. Το θετικό αυτής της μεθόδου είναι πως απαιτεί ελάχιστο ή και καθόλου επιπλέον χώρο (ως προς την περίπτωση που έχουμε μόνο συμβολοσειρές με διαχωρισμό κεφαλαίων-μικρών). Επιπλέον, είναι ευκολότερη στην υλοποίηση από την πρώτη μέθοδο. Το αρνητικό είναι πως χρειάζεται επανεξέταση χαρακτήρων σε περίπτωση που βρίσκουμε μια συμβολοσειρά χωρίς διαχωρισμό, ενώ πιθανόν να χρειάζονται επιπλέον εντολές για την επεξεργασία των χαρακτήρων που διαβάζονται από το κείμενο εισόδου.

Για τον αλγόριθμο Aho-Corasick εξετάσαμε τρεις διαφορετικές περιπτώσεις για την αντιμετώπιση του προβλήματος αναζήτησης συμβολοσειρών χωρίς διαχωρισμό κεφαλαίων και μικρών γραμμάτων.

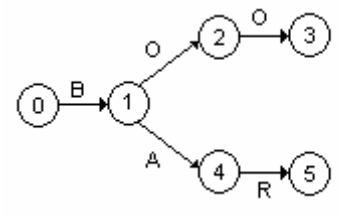
α) Αρχικά στηριχτήκαμε στην πρώτη μεθοδολογία για την υποστήριξη αναζητήσεων χωρίς διαχωρισμό. Για να γίνει αυτό μετατρέψαμε την κάθε συμβολοσειρά που θέλουμε να αναζητηθεί *χωρίς* διαχωρισμό στο ισοδύναμο *σύνολο* συμβολοσειρών για αναζήτηση *με* διαχωρισμό.

Με τον τρόπο αυτό δεν υπάρχει η πιθανότητα ψευδούς αναγνώρισης μίας case sensitive συμβολοσειράς. Προκύπτει, όμως, το πρόβλημα ότι αυξάνονται σημαντικά οι απαιτήσεις σε χώρο. Ας θεωρήσουμε ότι έχουμε μια συμβολοσειρά χωρίς διαχωρισμό, η οποία αποτελείται από m γράμματα. Για την αναζήτηση όλων των πιθανών παραλλαγών της συμβολοσειράς αυτής χρειάζονται 2^{m+1}-1 καταστάσεις στην μηχανή πεπερασμένων καταστάσεων ενώ για μια αντίστοιχου μήκους συμβολοσειρά με διαχωρισμό κεφαλαίων και μικρών απαιτούνται μόλις m καταστάσεις. Δεδομένου του περιορισμού που έχουμε εισάγει στις υλοποιήσεις μας για μέγιστο αριθμό καταστάσεων ίσο με 65536, με τη μέθοδο αυτή προέκυπτε για ορισμένα σύνολα κανόνων ότι ο απαιτούμενος αριθμός καταστάσεων ήταν εκτός ορίου.

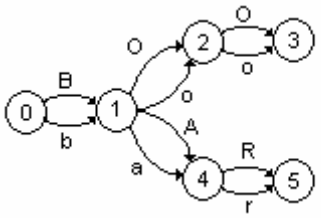
β) Χρησιμοποιώντας τη δεύτερη μεθοδολογία είναι δυνατή η υποστήριξη case insensitive αναζητήσεων χωρίς καμία αύξηση στον αριθμό των απαιτούμενων καταστάσεων. Για να επιτευχθεί το αποτέλεσμα αυτό χρησιμοποιείται η ακόλουθη διαδικασία κατά την κατασκευή της ΜΠΚ: αρχικά αποθηκεύουμε κατάλληλα τις πληροφορίες για το ποιες συμβολοσειρές είναι case sensitive και ποιες όχι και μετατρέπουμε τα γράμματα όλων των συμβολοσειρών στην ίδια μορφή, δηλαδή όλα κεφαλαία ή όλα μικρά. Στην υλοποίησή μας όλα τα γράμματα μετατρέπονται σε κεφαλαία. Στο επόμενο βήμα εισάγουμε όλες τις συμβολοσειρές στο αρχικά άδειο δέντρο μεταβάσεων. Όταν έχουν εισαχθεί όλες οι συμβολοσειρές εξετάζουμε κάθε κατάσταση και σε περίπτωση που βρίσκουμε μετάβαση προς μια άλλη κατάσταση s για κάποιο γράμμα c τότε προσθέτουμε άλλη μια μετάβαση από την κατάσταση που βρισκόμαστε προς την s για το γράμμα c', το οποίο είναι το αντίστροφο του c (δηλαδή, αν το c είναι κεφαλαίο γράμμα τότε το c' είναι το αντίστοιχο μικρό γράμμα). Στη συνέχεια το δέντρο μετατρέπεται σε ντετερμινιστικό αυτόματο ακριβώς

όπως περιγράφεται στον αλγόριθμο Aho-Corasick. Το μόνο που χρειάζεται να έχουμε υπόψη μας κατά τη διάρκεια της αναζήτησης είναι ότι, σε περίπτωση που βρεθούμε σε μια κατάσταση που σηματοδοτεί το τέλος μίας συμβολοσειράς με διαχωρισμό γραμμάτων (case sensitive), χρειάζεται επαληθεύσουμε το κομμάτι του κειμένου που διαβάστηκε με την αρχική μορφή της συμβολοσειράς για να επιβεβαιώσουμε ότι πράγματι βρέθηκε.

Για να γίνει πιο εύκολα αντιληπτός ο τρόπος κατασκευής του δέντρου σύμφωνα με τη δεύτερη μεθοδολογία, ας θεωρήσουμε την περίπτωση όπου θέλουμε να αναζητήσουμε τη συμβολοσειρά "bOo" με διαχωρισμό γραμμάτων και τη συμβολοσειρά "bar" χωρίς διαχωρισμό. Αρχικά μετατρέπουμε όλα τα γράμματα στην ίδια μορφή, έστω σε κεφαλαία, και εισάγουμε τις συμβολοσειρές στο άδειο δέντρο μεταβάσεων. Η μορφή του δέντρου στο σημείο αυτό φαίνεται στην παρακάτω εικόνα.



Στη συνέχεια εξετάζουμε κάθε κόμβο του δέντρου και σε περίπτωση που βρούμε μετάβαση για κάποιο γράμμα εισόδου σε έναν κόμβο τότε προσθέτουμε μετάβαση για το αντίστροφο γράμμα προς τον ίδιο κόμβο με προηγουμένως. Η μορφή που θα αποκτήσει το δέντρο μετά από αυτό το στάδιο φαίνεται παρακάτω.

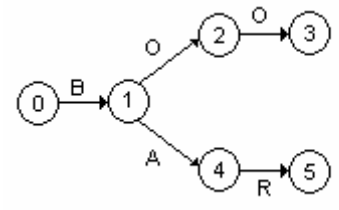


Από το σημείο αυτό και μετά χρησιμοποιούνται οι κλασσικοί αλγόριθμου του αλγορίθμου Aho-Corasick για την μετατροπή του δέντρου σε ντετερμινιστικό αυτόματο. Το μόνο που πρέπει να προσέχουμε όταν εξετάζουμε κάποιο κείμενο εισόδου είναι ότι, στην περίπτωση που βρεθούμε στην κατάσταση 3 (γεγονός που σημαίνει ότι βρήκαμε κάποια παραλλαγή της "bOo"), θα πρέπει να συγκρίνουμε

ξανά τους τελευταίους χαρακτήρες τρεις που διαβάστηκαν από το κείμενο με την συμβολοσειρά “bOo” για να επιβεβαιώσουμε την ύπαρξή της.

Η περίπτωση αυτή είναι που επιλέξαμε τελικά για τη χρήση στον αλγόριθμο Aho-Corasick.

γ) Η τρίτη και τελευταία περίπτωση που εξετάσαμε αποτελεί παραλλαγή της *β)*, επομένως στηρίζεται και αυτή στην δεύτερη μεθοδολογία. Το αρχικό στάδιο για την περίπτωση αυτή είναι ίδιο με εκείνο της *β)*, δηλαδή μετατρέπουμε όλα τα γράματα για όλες τις συμβολοσειρές σε κεφαλαία και τις εισάγουμε στο αρχικά άδειο δέντρο μεταβάσεων. Δηλαδή, για το παράδειγμα των δύο συμβολοσειρών που αναφέραμε στην περίπτωση *β)* προκύπτει το ακόλουθο δέντρο μεταβάσεων.



Αντίθετα με την *β)*, στην περίπτωση αυτή δεν προσθέτουμε καινούργιες μεταβάσεις. Όταν εισαχθούν όλες οι συμβολοσειρές με τον τρόπο αυτό στο δέντρο μεταβάσεων, κατασκευάζουμε το ντετερμινιστικό αυτόματο χρησιμοποιώντας τους κλασσικούς αλγορίθμους που περιγράφουν οι Aho-Corasick.

Το μόνο ππου χρειάζεται τώρα είναι μία αλλαγή στον τρόπο που γίνεται η αναζήτηση σε ένα κείμενο. Συγκεκριμένα, κάθε γράμμα που διαβάζουμε από το κείμενο το μετατρέπουμε σε κεφαλαίο και μετά το εισάγουμε στην ΜΠΚ. Η αναγνώριση των συμβολοσειρών γίνεται όπως στη *β)*, δηλαδή κάθε φορά που η μηχανή βρισκεται σε τελική κατάσταση που αντιστοιχεί σε συμβολοσειρά με διαχωρισμό γραμμάτων συγκρίνουμε την κανονική μορφή της συμβολοσειράς (όπως είναι πριν μετατρέψουμε όλα της τα γράμματα σε κεφαλαία) με τους χαρακτήρες που βρίσκονται στις τελευταίες θέσεις που εξετάσαμε.

Η μετατροπή των γραμμάτων σε κεφαλαία μπορεί να γίνει εύκολα με τη χρήση ενός πίνακα 256 θέσεων. Για τις θέσεις από 0 εώς 96 και από 123 μέχρι 255, ο πίνακας περιέχει τιμή ίση με τη θέση του πίνακα. Για παράδειγμα, στη θέση [5] του πίνακα θα βρίσκεται η τιμή 5. Αντίθετα, για τις τιμές από 97 μέχρι και 122

αποθηκεύουμε την θέση μείον 32. Ο λόγος που γίνεται αυτό είναι επειδή οι τιμές 97-122 αντιστοιχούν σε μικρά γράμματα, τα οποία μετατρέπονται στο αντίστοιχο κεφαλαίο γράμμα όταν αφαιρέσουμε 32. Με τον τρόπο αυτό, όταν διαβάζουμε έναν χαρακτήρα *x* από το κείμενο χρησιμοποιούμε τα περιεχόμενα της θέσης [*x*] του πίνακα σαν είσοδο στην ΜΠΚ.

Η υλοποίηση αυτή προσφέρει βελτιωμένη τοπικότητα ως προς την *β*), αφού για κάθε κατάσταση οι μεταβάσεις που αντιστοιχούν στα 26 μικρά γράμματα του αλφαβήτου δεν εξετάζονται ποτέ. Υπάρχει, όμως, το πρόσθετο κόστος της επιπλέον πρόσβασης στον πίνακα μετατροπής των γραμμάτων για κάθε χαρακτήρα που εξετάζεται.

Πρέπει να αναφέρουμε ότι την περίπτωση *γ)* τελικά δεν την υλοποιήσαμε ώστε να τη συγκρίνουμε άμεσα με την *β)*. Ο λόγος που δεν προχωρήσαμε στη υλοποίηση είναι ότι δεν περιμένουμε βελτίωση της ταχύτητας. Παρόλο που ο πίνακας μετατροπής των χαρακτήρων για τη *γ)* είναι πολύ μικρός και θα χωράει στην cache (άρα θα είναι γρήγορες οι προσβάσεις σε αυτόν) δεν μπορεί να αγνοηθεί το γεγονός ότι χρειάζεται μία επιπλέον πρόσβαση για κάθε χαρακτήρα που ελέγχεται. Επιπλέον, η αύξηση της τοπικότητας της περίπτωσης *γ)* ως προς τη *β)* είναι περίπου 10%, (δεδομένου ότι αποφεύγουμε τις προσβάσεις σε 26 από τις 256 μεταβάσεις κάθε κατάστασης) τιμή αρκετά μικρή για να αντισταθμίσει το κόστος της επιπλέον πρόσβασης ανά χαρακτήρα.

4.2 Υλοποίηση του Boyer-Moore.

Αρχικά πρέπει να αναφέρουμε ότι επιλέχθηκε να μην υλοποιήσουμε τον τυπικό αλγόριθμο Boyer-Moore, αλλά την παραλλαγή Boyer-Moore-Horspool. Όπως αναφέραμε στην παράγραφο 3.2.5, ο αλγόριθμος του Horspool αποτελεί απλοποίηση του Boyer-Moore καθώς αγνοεί την ευρετική «καλής κατάληξης» και χρησιμοποιεί για τις μετατοπίσεις μόνο την ευρετική «κακού χαρακτήρα».

Η επιλογή για τη χρήση του αλγορίθμου του Horspool έγινε με βάση το γεγονός ότι χρησιμοποιείται ένα αλφάβητο 256 χαρακτήρων και η συμβολοσειρές προς αναζήτηση σπάνια περιέχουν κατάληξη που να επαναλαμβάνεται στο εσωτερικό της συμβολοσειράς. Αυτό συνεπάγεται ότι σχεδόν πάντα θα χρησιμοποιείται η ευρετική «κακού χαρακτήρα» για τη μετατόπιση, επομένως ο Horspool είναι ιδανικός. Επίσης, η πρόθεσή μας ήταν να χρησιμοποιούμε τον συγκεκριμένο αλγόριθμο στις περιπτώσεις που κάποια σύνολα περιέχουν μόνο έναν κανόνα, επομένως επιλέχτηκε να μην υπάρχει υποστήριξη για περισσότερες από μία συμβολοσειρές.

Η υλοποίηση του αλγορίθμου Boyer-Moore-Horspool ήταν αρκετά απλή και δεν χρειάστηκε να περάσει από στάδια εξέλιξης όπως έγινε με τον Aho-Corasick. Η μόνη τροποποίηση ως προς τον κλασσικό τρόπο υλοποίησης του αλγορίθμου είναι ότι η ζητούμενη συμβολοσειρά αποθηκεύεται σε έναν πίνακα ανάποδα, δηλαδή από το τέλος προς την αρχή. Η διαφοροποίηση αυτή προέκυψε με βάση τον τρόπο που αποθηκεύονται και με τον οποίο γίνεται πρόσβαση στα δεδομένα.

Ας θεωρήσουμε σαν παράδειγμα την περίπτωση όπου έχουμε μία ζητούμενη συμβολοσειρά p μήκους m. Σε μία τυπική υλοποίηση οι χαρακτήρες της p θα είναι αποθηκευμένοι υπό μορφή πίνακα, έτσι ώστε το στοιχείο $p[0]$ να αντιστοιχεί στον πρώτο χαρακτήρα της συμβολοσειράς και το $p[m\text{-}1]$ στον τελευταίο χαρακτήρα. Η αρχή του συγκεκριμένου πίνακα θα βρίσκεται σε μία διεύθυνση A της μνήμης, η οποία θα περιέχει και το πρώτο στοιχείο του πίνακα. Κατά τη διάρκεια της αναζήτησης εξετάζεται συνέχεια ο τελευταίος χαρακτήρας, ο m-1, της συμβολοσειράς. Προκειμένου να γίνει η πρόσβαση στον τελευταίο χαρακτήρα χρειάζεται να προσθέσουμε στην τιμή A την m-1 ώστε να βρούμε τη διεύθυνση του τελευταίου χαρακτήρα και να τον διαβάσουμε από τη μνήμη. Σε περίπτωση, όμως, που η συμβολοσειρά είναι αποθηκευμένη ανάποδα, τότε ο γλιτώνουμε σε κάθε επανάληψη την πρόσθεση καθώς ο τελευταίος χαρακτήρας, ο οποίος και μας ενδιαφέρει, βρίσκεται με άμεση πρόσβαση στη θέση A.

Ακολουθεί ο ψευδοκώδικας που περιγράφει την υλοποίηση μας για τον αλγόριθμο Boyer-Moore-Horspool.

Αλγόριθμος BMH: Υλοποίηση αλγόριθμου αναζήτησης Boyer-Moore-Horspool.

Είσοδος: Σύμβολοσειρά προς αναζήτηση *p* αποθηκευμένη με αντίστροφη σειρά, πίνακας μετατοπίσεων *shift*, μήκος συμβολοσειράς *m*, κείμενο εισόδου *text*.

Έξοδος: Η θέση κάθε στιγμιοτύπου της συμβολοσειράς.

```
begin
        i ← m
        while (i ≤ length(text) )
        begin
                c ← text[i]
                if c ≠ p[0] then i ← i + shift[c]
                else
                begin
                        j ← 1
                        while (text[ i - j] == p[j] ) && ( j < m ) do j ← j+1
                        if  j==m then output( i )
                        i ← i + shift[c]
                end
        end
end
```

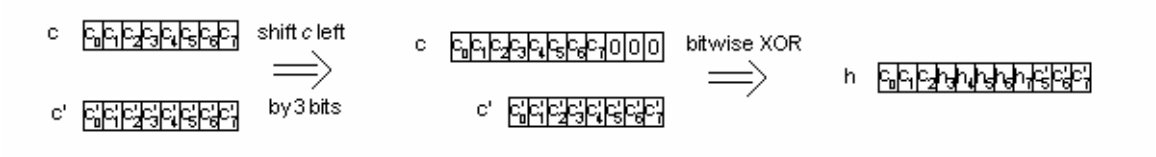
4.2.1 Υποστήριξη αναζητήσεων χωρίς διαφοροποίηση.

Αρκετά απλός είναι και ο τρόπος με τον οποίο υποστηρίζεται η αναζήτηση συμβολοσειράς χωρίς διαχωρισμό ανάμεσα σε κεφαλαία και μικρά γράμματα. Αρχικά μετατρέπουμε όλα τα γράμματα της συμβολοσειράς σε μία μορφή, έιτε όλα μικρά είτε όλα κεφαλαία. Στην υλοποίησή μας επιλέξαμε να τα μετατρέπουμε όλα σε κεφαλαία.. Έπειτα, κατά τη διάρκεια της αναζήτησης, εξετάζουμε κάθε φορά τον χαρακτήρα *c*, που μόλις διαβάσαμε από το κείμενο, μήπως είναι μικρό γράμμα. Σε περίπτωση που όντως είναι μικρό γράμμα τον μετατρέπουμε στο αντίστοιχο κεφαλαίο. Στη συνέχεια γίνεται η σύγκριση με τον τελευταίο χαρακτήρα της συμβολοσειράς και σε περίπτωση ισότητας διαβάζουμε τον προηγούμενο χαρακτήρα, τον μετατρέπουμε σε κεφαλαίο σε περίπτωση που είναι μικρό γράμμα, τον συγκρίνουμε με τον αντίστοιχο χαρακτήρα της συμβολοσειράς και ούτω καθεξής.

4.3 Υλοποίηση του αλγορίθμου Wu-Manber.

Σχετικά με τη λειτουργία και την υλοποίηση σου αλγορίθμου Wu-Manber, σημαντικό ρόλο κατέχουν οι τιμές των σταθερών παραμέτρων του αλγορίθμου. Για τις παραμέτρους *B* και *B'*, δηλαδή τον αριθμό των χαρακτήρων που ελέγχονται στο τέλος των συμβολοσειρών και στην αρχή αντίστοιχα, χρησιμοποιήθηκαν οι τιμές 2 και 1. Συνεπώς εξετάζονται και γίνονται hash 2 χαρακτήρες του κειμένου για να βρούμε την θέση που μας ενδιαφέρει μέσα στον πίνακα μετατοπίσεων SHIFT, ενώ για τη σύγκριση των προθεμάτων χρησιμοποιείται μόνο ένας χαρακτήρας. Ουσιαστικά επιλέξαμε τις μικρότερες δυνατές τιμές έτσι ώστε να μπορούμε να έχουμε μια απάντηση εξετάζοντας το μικρότερο δυνατό αριθμό χαρακτήρων από το κείμενο.

Σαν συνάρτηση hash χρησιμοποιήθηκε η πράξη bitwise XOR, η οποία γινόταν με τέτοιο τρόπο ανάμεσα στους δύο χαρακτήρες κατάληξης ώστε να μας δώσει μία τιμή πλάτους *b* bits, όπου το *b* παίρνει τιμές ανάμεσα στο 8 και το 16 και καθορίζεται από το χρήστη. Η τιμή *b* επηρεάζει άμεσα το μέγεθος που των τριών πινάκων του αλγορίθμου, καθώς και οι τρεις έχουν μέγεθος 2^b.Προκειμένου κάνουμε hash δύο χαρακτήρες, έστω *c* και *c'*, κάνουμε αριστερό shift τον χαρακτήρα *c* κατά *b*-8 θέσεις και ύστερα τον κάνουμε bitwise XOR με τον *c'*. Ένα παράδειγμα αυτής της διαδικασίας για *b*=11 φαίνεται στην ακόλουθη εικόνα.



Εικόνα 4β) η διαδικασία hashing των c και c' για τη δημιουργία του h.

Από δοκιμές που έγιναν με τα πειραματικά δεδομένα προέκυψε ότι η καλύτερη τιμή του *b*, ως προς το θέμα του χρόνου εκτέλεσης, είναι 10 bits.

Πέρα από τις επιλογές των παραμέτρων, η υλοποίηση του αλγορίθμου έγινε χωρίς επιπλοκές και σημαντικές τροποποιήσεις. Η μόνη διαφορά σε σχέση με μία τυπική υλοποίηση είναι στον τρόπο με τον οποίο αποθηκεύονται οι συμβολοσειρές προς αναζήτηση.

Ας θεωρήσουμε την περίπτωση που έχουμε ένα σύνολο k συμβολοσειρών $p_0, p_1, p_2, \ldots, p_{k-1}$, μήκους m_j η κάθε μία. Σε μία τυπική υλοποίηση θα υπάρχει ένας πίνακας k θέσεων με δείκτη σε κάθε συμβολοσειρά ξεχωριστά. Στην περίπτωση που θέλουμε να εξετάσουμε μία συμβολοσειρά i, γεγονός που συμβαίνει κάθε φορά που οι χαρακτήρες του κειμένου ταιριάζουν με την κατάληξη και το πρόθεμα της i και χρειάζεται να εξετάσουμε και τους υπόλοιπους χαρακτήρες, είναι απαραίτητο να γίνουν δύο προσβάσεις στη μνήμη: η πρώτη πρόσβαση γίνεται στον πίνακα με τις συμβολοσειρές προκειμένου να βρεθεί ο δείκτης στη συμβολοσειρά i και η δεύτερη γίνεται στη θέση που υποδεικνύει ο δείκτης για να προσπελάσουμε τη συμβολοσειρά.

Στην υλοποίησή μας χρησιμοποιήθηκε διαφορετική λογική. Δημιουργούμε έναν πίνακα με k μπλοκ χαρακτήρων, όπου κάθε μπλοκ είναι μεγέθους 2^z με το z να είναι μία ακέραια τιμή τέτοια ώστε $2^z \geq m_j$ για κάθε j από 0 μέχρι k-1. Δηλαδή, κάθε μπλοκ είναι αρκετά μεγάλο ώστε να χωράει και τη μεγαλύτερη συμβολοσειρά. Με τον τρόπο αυτό, όταν θέλουμε να προσπελάσουμε τη συμβολοσειρά i απλά εκτελούμε ένα (shift) του i προς τα αριστερά κατά z bits και πηγαίνουμε στη συγκεκριμένη θέση του πίνακα. Συνεπώς αποφεύγεται μία πρόσβαση στη μνήμη. Το αρνητικό με αυτή τη μέθοδο αποθήκευσης των συμβολοσειρών είναι ότι μπορεί να υπάρχει σημαντική σπατάλη χώρου, για παράδειγμα στην περίπτωση που έχουμε ένα σύνολο από πολλές μικρές συμβολοσειρές και μία μεγάλη.

Ο παρακάτω ψευδοκώδικας περιγράφει την υλοποίηση μας για τον αλγόριθμο Wu-Manber.

Αλγόριθμος WM: Υλοποίηση αλγόριθμου αναζήτησης Wu-Manber.

Είσοδος: Πίνακές *shift*, *hash* και *prefix*, εύρος πινάκων *b*, ελάχιστο μήκος συμβολοσειρων *m*, κείμενο εισόδου *text*.

Έξοδος: Οι συμβολοσειρές που βρέθηκαν μέσα στο κείμενο και η θέση κάθε στιγμιοτύπου.

```
begin
        i ← m
        while (i ≤ length(text) )
        begin
                h ← (text[i] << (b-8) ) XOR text[i-1]
                if shift[h] > 0  then i ← i + shift[h]
                else
```

```
            begin
                c = text[i-m+1]
                for every signature p in hash[h] do
                begin
                    if prefix[p] == c then
                    begin
                        compare p with text[i-m+1…i]
                        if equal then output(p, i )
                    end
                end
                i ← i+1
            end
        end
end
```

4.3.1 Υποστήριξη αναζητήσεων χωρίς διαφοροποίηση.

Το πρόβλημα της αναζήτησης συμβολοσειρών χωρίς διαχωρισμό κεφαλαίων και μικρών γραμμάτων λύθηκε χρησιμοποιώντας έλεγχο χωρίς διαχωρισμό για όλες τις συμβολοσειρές και επαλήθευση με δεύτερο έλεγχο σε περίπτωση που βρούμε παραλλαγή κάποιας case sensitive συμβολοσειρές.

Ας υποθέσουμε πως έχουμε ένα σύνολο συμβολοσειρών από το οποίο κάποιες συμβολοσειρές θέλουμε να αναζητηθούν με διαχωρισμό γραμμάτων και κάποιες άλλες χωρίς διαχωρισμό. Αρχικά, στη φάση της προεπεξεργασίας, χρησιμοποιούμε όλες τις συμβολοσειρές ακριβώς όπως μας έχουν δοθεί για να γεμίσουμε τις θέσεις του πίνακα SHIFT (ακολουθώντας τη διαδικασία του hashing των χαρακτήρων ανά ζεύγη για τη δημιουργία του δείκτη *h* και της ενημέρωσης των τιμών των πινάκων στη θέση *h*). Στις αντίστοιχες θέσεις του πίνακα HASH για κάθε συμβολοσειρά αποθηκεύουμε την ίδια συμβολοσειρά αφού πρώτα μετατρέψουμε όλα της τα γράμματα στην ίδια μορφή, δηλαδή είτε όλα κεφαλαία είτε όλα μικρά (στην περίπτωσή μας κεφαλαίά). Για παράδειγμα, αν έχουμε τη συμβολοσειρά “abCd” βρίσκουμε το *h* κάνοντας hash τους χαρακτήρες ‘C’ και ‘d’ και στην αντίστοιχη θέση του πίνακα HASH βάζουμε την

συμβολοσειρά “ABCD”. Με παρόμοιο τρόπο συμπληρώνονται η θέσεις του πίνακα PREFIX, δηλαδή στο παράδειγμά μας το PREFIX για την “abCd” είναι ‘A’.

Το επόμενο βήμα αφορά μόνο τις case insensitive συμβολοσειρές: βρίσκουμε όλους τους εναλλακτικούς, ως προς το μέγεθος, συνδυασμούς των χαρακτήρων κατάληξης, δημιουργούμε για κάθε εναλλακτικό συνδυασμό την αντίστοιχη τιμή h και ενημερώνουμε κατάλληλα τις θέσεις των τριών πινάκων SHIFT, HASH και PREFIX. Οι εναλλακτικοί συνδυασμοί για τους χαρακτήρες μπορεί να είναι από 0, σε περίπτωση που κανένας από τους δύο χαρακτήρες δεν είναι γράμμα, μέχρι 3, εάν και οι δύο χαρακτήρες είναι γράμματα. Για παράδειγμα, σε περίπτωση που η συμβολοσειρά “abCd” είναι case insensitive θα προσθέσουμε τα στοιχεία της σε τρεις επιπλέον θέσεις στους πίνακες, στις θέσεις που αντιστοιχούν στα ζεύγη χαρακτήρων “cd”,”cD” και “CD”. Η διαδικασία της εύρεσης και υπολογισμού των εναλλακτικών συνδυασμών επαναλαμβάνεται και για τα ενδιάμεσα ζεύγη χαρακτήρων ώστε να γεμίσουν κατάλληλα οι θέσεις του πίνακα SHIFT. Επομένως, για την ίδια συμβολοσειρά στις θέσεις που αντιστοιχούν στους χαρακτήρες “bc”,”Bc,” και ”BC” θα έχουμε τιμή του πίνακα SHIFT ίση με 1 ενώ για τους χαρακτήρες “aB”, “Ab” και “AB” η τιμή θα είναι 2. Με τον τρόπο αυτό ολοκληρώνεται η φάση της προεπεξεργασίας.

Αλλαγές γίνονται και στη φάση της αναζήτησης μέσα στο κείμενο. Το πρώτο βήμα του αλγορίθμου αναζήτησης, στο οποίο κάνουμε hash δύο γειτονικούς χαρακτήρες και εξετάζουμε την αντίστοιχη τιμή h αν είναι 0 ή όχι, παραμένει ίδιο. Σε όλα τα υπόλοιπα βήματα του αλγορίθμου, κάθε γράμμα που διαβάζεται από το κείμενο μετατρέπεται στη μορφή που είναι τα γράμματα στους πίνακες PREFIX και HASH. Επομένως, στην υλοποίηση μας όταν είμαστε στη φάση που συγκρίνουμε έναν χαρακτήρα του κειμένου με το αντίστοιχο πρόθεμα συμβολοσειράς πρέπει πρώτα να μετατρέψουμε τον χαρακτήρα σε κεφαλαίο γράμμα, σε περίπτωση που είναι μικρό. Η ίδια μετατροπή γίνεται σε περίπτωση που έχουμε ταιριάξει την κατάληξη και το πρόθεμα μίας συμβολοσειράς και εξετάζουμε έναν-έναν τους ενδιάμεσους χαρακτήρες.

Σε περίπτωση που διαπιστώσουμε ότι ίσως έχει βρεθει μια case sensitive συμβολοσειρά, χρειάζεται να συγκρίνουμε ξανά τους χαρακτήρες που μόλις διαβάσαμε με την αρχική μορφή της συμβολοσειράς, ώστε να αποφύγουμε το ενδεχόμενο ψευδούς αναγνώρισης.

4.4 Σύνοψη.

Στο κεφάλαιο αυτό περιγράψαμε τις υλοποιήσεις που έγιναν για κάθε έναν από τους τρεις αλγορίθμους. Επίσης αναφέρθηκαν και αιτιολογήθηκαν οι σχεδιαστικές επιλογές που έγιναν σε κάθε περίπτωση. Τέλος, για κάθε αλγόριθμο έγινε ειδική αναφορά στον τρόπο με τον οποίο υποστηρίζουμε την αναζήτηση συμβολοσειρών χωρίς διαχωρισμό κεφαλαίων και μικρών γραμμάτων.

Σε κάθε περίπτωση, ο στόχος μας ήταν η επίτευξη της μέγιστης δυνατής ταχύτητας για κάθε αλγόριθμο. Οι υλοποιήσεις στις οποίες καταλήξαμε μας δίνουν αρκετά καλές ταχύτητες, όμως είναι πιθανό να υπάρχουν και άλλα περιθώρια βελτίωσης τα οποία παραβλέψαμε.

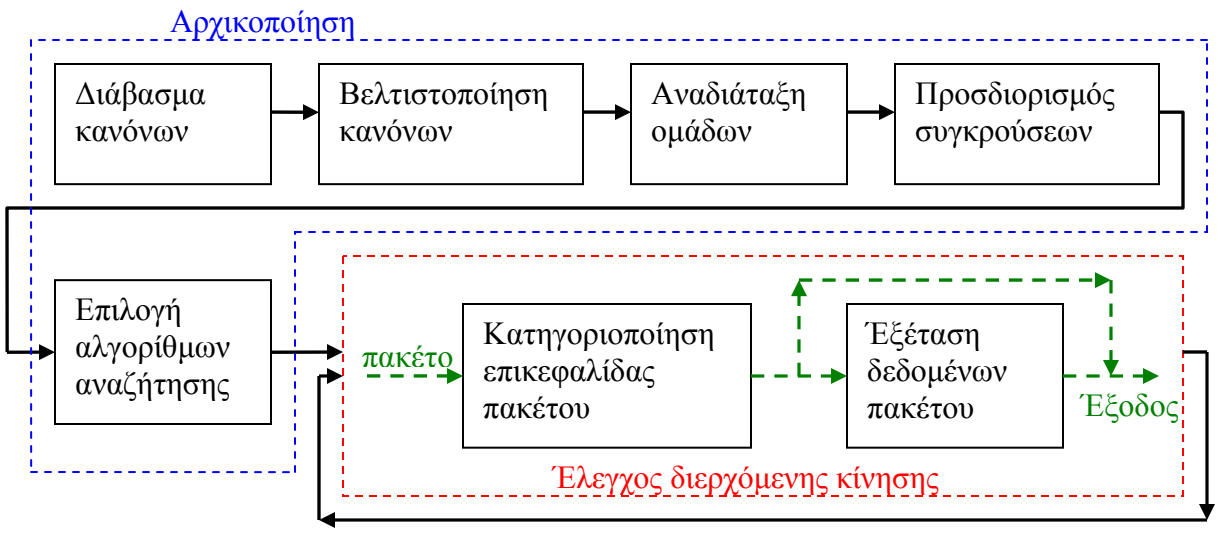
Κεφάλαιο 5.

Περιγραφή του συνολικού συστήματος TUCNIDS και των φίλτρων υλικού.

Όπως αναφέραμε στην εισαγωγή, το κίνητρο της εργασίας αυτής ήταν να διερευνήσουμε κατά πόσο θα μπορούσε η χρήση σχετικά απλών φίλτρων σε υλικό να επιταχύνει και να διευκολύνει τη λειτουργία ενός συστήματος ανίχνευσης εισβολών δικτύου. Παράλληλα θέλαμε να διαπιστώσουμε εάν και κατά πόσο μπορεί να βελτιωθεί η ταχύτητα ενός συστήματος ανίχνευσης εισβολών μέσω καλύτερων υλοποιήσεων των αλγορίθμων αναζήτησης.

Απαραίτητη προϋπόθεση για την επίτευξη των παραπάνω στόχων ήταν η υλοποίηση ενός συστήματος ανίχνευσης εισβολών το οποίο θα παρείχε τη στοιχειώδη λειτουργικότητα, δηλαδή τη δυνατότητα να εξετάζονται διερχόμενα πακέτα και να ανιχνεύονται ύποπτες συμβολοσειρές. Το σύστημα ανίχνευσης εισβολών δικτύου που υλοποιήσαμε ονομάστηκε TUCNIDS (Technical University of Crete Network Intrusion Detection Sustem).

Σε γενικές γραμμές, η λειτουργία του TUCNIDS μπορεί να περιγραφεί από το παρακάτω διάγραμμα. Κάθε ορθογώνιο του διαγράμματος μπορεί να θεωρηθεί σαν ένα ανεξάρτητο υποσύστημα, ενώ τα μαύρα βέλη συμβολίζουν τη ροή του ελέγχου.



Εικόνα 5α) Οι διαφορετικές φάσεις τις λειτουργίας του TUCNIDS

Η λειτουργία του TUCNIDS χωρίζεται σε δύο λογικά στάδια: στο στάδιο αρχικοποίησης, κατά το οποίο εκτελούνται οι απαραίτητες ενέργειες για την προετοιμασία του συστήματος, και στο στάδιο ελέγχου της εισερχόμενης κίνησης δικτύου.

Το υπόλοιπο κεφάλαιο χωρίζεται σε δύο ενότητες. Η πρώτη ενότητα περιλαμβάνει μια πιο αναλυτική περιγραφή των υπολειτουργιών του TUCNIDS που εκτελούνται κατά τη διάρκεια της αρχικοποίησης και του ελέγχου της διερχόμενης κίνησης.

Στη δεύτερη ενότητα του κεφαλαίου περιγράφεται η θεωρητική λειτουργία των φίλτρων υλικού που θα μπορούσαν να χρησιμοποιηθούν για την υποβοήθηση και βελτίωση της συνολικής απόδοσης ενός συστήματος ανίχνευσης. Επίσης, παρουσιάζουμε μια θεωρητική εκτίμηση της επίδρασης που θα έχει η χρήση των φίλτρων υλικού στο συνολικό σύστημα.

5.1 Η αρχικοποίηση του TUCNIDS.

Όπως φαίνεται στο διάγραμμα *5α*, η αρχικοποίηση του TUCNIDS ολοκληρώνεται σε πέντε διαφορετικά στάδια, των οποίων η εκτέλεση γίνεται σειριακά και με την ακολουθία που φαίνεται στο σχήμα. Η διαδικασία της αρχικοποίησης, όπως είναι αναμενόμενο, εκτελείται μόνο μία φορά κατά την εκκίνηση της λειτουργίας του TUCNIDS.

5.1.1 Το διάβασμα των κανόνων.

Η πρώτη ενέργεια που εκτελείται κατά την εκκίνηση της λειτουργίας του TUCNIDS είναι το διάβασμα όλων των αρχείων με τους κανόνες που θα χρησιμοποιηθούν για την εξέταση των εισερχόμενων πακέτων. Τα αρχεία κανόνων που χρησιμοποιεί το TUCNIDS είναι ακριβώς ίδια με εκείνα του Snort. Παρόλα αυτά, δεν υπάρχει πλήρης υποστήριξη όλων των λειτουργιών του Snort καθώς ο στόχος μας ήταν η κατασκευή ενός συστήματος ανίχνευσης εισβολών με την στοιχειώδη λειτουργικότητα, έτσι ώστε να μπορούμε να αξιολογήσουμε τις επιδόσεις και την παράλληλη χρήση υλικού-λογισμικού.

Υπενθυμίζουμε ότι κάθε κανόνας του Snort αποτελείται από δύο λογικά τμήματα, την επικεφαλίδα και τις ρυθμίσεις (options). Όσον αφορά τις επικεφαλίδες των κανόνων, υποστηρίζονται σχεδόν όλες οι επιλογές που υπάρχουν και στο Snort. Μία από τις επιλογές που δεν υλοποιήθηκαν είναι η αναγραφή των διευθύνσεων IP υπό μορφή λίστας, όπως για παράδειγμα [100.100.100.0/24,200.200.0.0/16]. Συμβολικές διευθύνσεις του τύπου ‘$HOME_NET’ υποστηρίζονται, όμως πρέπει να δηλώνονται από το χρήστη σε κατάλληλο σημείο του κώδικα πριν από την εκτέλεση του προγράμματος. Σε περίπτωση που βρεθεί σε κάποιον κανόνα συμβολική διεύθυνση που δεν έχει οριστεί, η εκτέλεση τερματίζεται. Η δεύτερη και τελευταία λειτουργία που δεν υποστηρίζεται, όσον αφορά τις επικεφαλίδες, είναι η αποθήκευση του είδους των κανόνων, δηλαδή αν κάποιος κανόνας είναι alert, log ή pass. Στο TUCNIDS όλοι οι κανόνες αντιμετωπίζονται με τον ίδιο τρόπο.

Σχετικά με τις ρυθμίσεις των κανόνων, η υποστήριξη από το TUCNIDS είναι αρκετά περιορισμένη καθώς το ενδιαφέρον μας επικεντρώθηκε μόνο στο θέμα της αναζήτησης συμβολοσειρών. Για το λόγο αυτό, οι μόνες ρυθμίσεις των κανόνων που χρησιμοποιούνται είναι η 'content' και η 'sid'. Η εντολή 'content' μας υποδεικνύει μια συμβολοσειρά προς αναζήτηση, ενώ η εντολή 'sid' υποδεικνύει τον αριθμό-ταυτότητα του συγκεκριμένου κανόνα. Από τις επιπλέον εντολές που τροποποιούν τη λειτουργία της 'content', η μόνη που υποστηρίζεται είναι η 'nocase' (δηλαδή ότι η συμβολοσειρά είναι case insensitive). Οι υπόλοιπες εντολές τροποποίησης της 'content', όπως είναι η 'distance' ή η 'offset', δεν υλοποιήθηκαν.

Οι περισσότερες από τις επιπλέον εντολές των κανόνων θα μπορούσαν να υλοποιηθούν σχετικά απλά, χωρίς να έχουν σημαντική επίπτωση στην ταχύτητα του συστήματος. Υπάρχουν, όμως, και ορισμένες εντολές, όπως η 'pcre' που εκτελεί αναζητήσεις κανονικών εκφράσεων, των οποίων η υποστήριξη θα απαιτούσε σημαντική επεξεργαστική ισχύ και θα καθυστερούσε το συνολικό σύστημα.

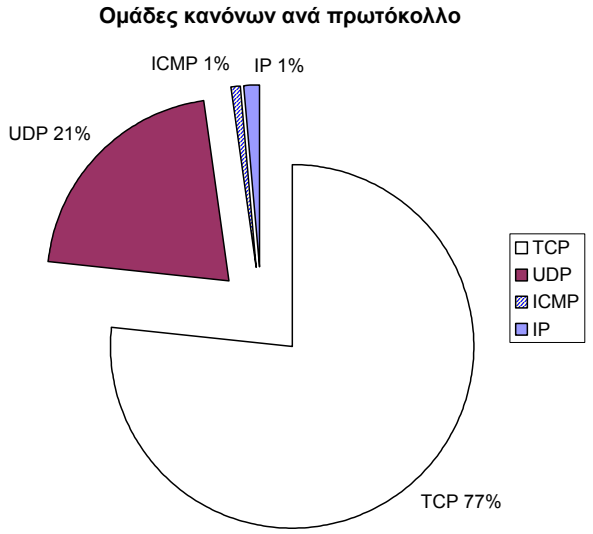
Στο σημείο αυτό πρέπει να αναφέρουμε ότι σε περίπτωση που δεν βρεθεί εντολή content σε έναν κανόνα, ο κανόνας αυτός αγνοείται. Επιπλέον, το TUCNIDS χρησιμοποιεί μόνο την πρώτη εντολή 'content' που εντοπίζει σε έναν κανόνα και αγνοεί τα επιπλέον 'content', εφόσον αυτά υπάρχουν. Επίσης αγνοούνται όλες οι υπόλοιπες ρυθμίσεις που μπορεί να υπάρχουν σε έναν κανόνα του Snort, όπως για παράδειγμα η εντολή 'pcre' ή η 'uricontent'.

Ο διαχωρισμός των κανόνων σε ομάδες γίνεται με βάση τις παραμέτρους που έχουν στις επικεφαλίδες τους, δηλαδή το πρωτόκολλο, τις διευθύνσεις αποστολέα και παραλήπτη και, για τα πρωτόκολλα TCP και UDP, τις θύρες αποστολέα και παραλήπτη. Οι ομάδες κανόνων αποθηκεύονται σε τέσσερις διαφορετικούς και αρχικά άδειους πίνακες, έναν πίνακα για κάθε πρωτόκολλο (TCP, UDP, ICMP και IP).

Καθώς διαβάζεται κάποιος κανόνας από ένα αρχείο, εξετάζουμε πρώτα το πρωτόκολλο του και βρίσκουμε τον πίνακα από ομάδες κανόνων στον οποίο θα εισαχθεί. Στη συνέχεια αποθηκεύουμε τις διευθύνσεις IP και -σε περίπτωση που έχουμε κανόνα TCP ή UDP- τις θύρες του κανόνα. Αμέσως μετά εξετάζουμε όλες τις ομάδες του πίνακα που επιλέχθηκε προηγουμένως για να βρούμε μήπως υπάρχει ομάδα με ακριβώς τις ίδιες παραμέτρους. Σε περίπτωση που βρεθεί κάποια ομάδα τότε ο κανόνας –επομένως και η

συμβολοσειρά προς αναζήτηση- εισάγεται στην ομάδα αυτή, διαφορετικά δημιουργείται ένα καινούργιο σύνολο κανόνων το οποίο αρχικά περιέχει μόνο τον κανόνα που μόλις διαβάσαμε.

Με τη χρήση όλων των αρχείων κανόνων που περιλαμβάνονται στη έκδοση 2.1.3 του Snort προκύπτουν συνολικά 222 ομάδες κανόνων. Η κατανομή των ομάδων κανόνων ανά πρωτόκολλο φαίνεται στην εικόνα *5β*.



Εικόνα 5β) Η κατανομή των 222 ομάδων κανόνων ανά πρωτόκολλο.

Μόλις ολοκληρωθεί το διάβασμα όλων των κανόνων από τα αντίστοιχα αρχεία περνάμε στη δεύτερη φάση της προεπεξεργασίας, στην οποία γίνεται η βελτιστοποίηση των συνόλων από κανόνες.

5.1.2 Η βελτιστοποίηση των κανόνων.

Η βελτιστοποίηση των κανόνων στο TUCNIDS εξυπηρετεί τον ίδιο σκοπό με τον Βελτιστοποιητή Κανόνων του Snort, δηλαδή να μετασχηματίσει τις ομάδες των κανόνων σε τέτοια μορφή ώστε για κάθε εισερχόμενο πακέτο να χρειάζεται η εξέταση μίας και μόνο ομάδας (και κατ' εξαίρεση η εξέταση μίας δεύτερης ομάδας).

Το πλεονέκτημα που προκύπτει από τη βελτιστοποίηση των κανόνων είναι πως για τη συντριπτική πλειοψηφία των πακέτων εκτελούμε το πολύ έναν έλεγχο του

περιεχομένου. Σε περίπτωση που δεν γινόταν η βελτιστοποίηση, θα έπρεπε να χρησιμοποιούμε τους αλγόριθμους αναζήτησης τόσες φορές όσες και η ομάδες κανόνων με τις οποίες ταιριάζουν οι παράμετροι ενός πακέτου, με συνέπεια τον πολλαπλασιασμό του απαιτούμενου χρόνου.

Η βελτιστοποίηση των κανόνων γίνεται σύμφωνα με την παρακάτω διαδικασία: επιλέγουμε μία ομάδα κανόνων, έστω Α, και τη συγκρίνουμε με όλες τις υπόλοιπες ομάδες κανόνων του ίδιου πρωτοκόλλου. Ας ονομάσουμε Β την ομάδα με την οποία γίνεται η σύγκριση σε κάποια στιγμή. Το ζητούμενο της σύγκρισης είναι να βρούμε τις περιπτώσεις όπου η ομάδα Α είναι υποσύνολο της Β. Στην περίπτωση που βρεθεί σχέση υποσυνόλου ανάμεσα στην Α και τη Β, τότε όλοι οι κανόνες που περιέχονται στην ομάδα Β αντιγράφονται και στην Α. Η διαδικασία αυτή επαναλαμβάνεται για κάθε ομάδα κανόνων που περιέχεται σε καθε ένα από τα τέσσερα πρωτόκολλα.

Όταν λέμε ότι μία ομάδα κανόνων Α είναι υποσύνολο μίας ομάδας Β εννοούμε ότι οι παράμετροι των δύο ομάδων είναι τέτοιες ώστε *όλα* τα πακέτα που ικανοποιούν τις παραμέτρους της ομάδας Α να ικανοποιούν και τις παραμέτρους της ομάδας Β. Επομένως, αφού αντιγράφουμε όλους τους κανόνες της Β στην Α, αρκεί να εξετάσουμε τα δεδομένα του εισερχόμενου πακέτου που ταιριάζει στην ομάδα Α *μόνο* με τους κανόνες της Α.

Η σύγκριση των ομάδων γίνεται με βάση τις παραμέτρους επικεφαλίδας. Για τις ομάδες IP και ICMP χρησιμοποιούνται οι διευθύνσεις αποστολέα και παραλήπτη για τη σύγκριση, ενώ για πρωτόκολλα TCP και UDP χρησιμοποιούνται και οι θύρες αποστολέα και παραλήπτη. Ικανή και αναγκαία συνθήκη προκειμένου μία ομάδα Α να αποτελεί υποσύνολο μίας άλλης ομάδας Β είναι *κάθε μία* από τις δύο ή τις τέσσερις (ανάλογα με το πρωτόκολλο) παραμέτρους επικεφαλίδας της Α να είναι υποσύνολο της αντίστοιχης παραμέτρου της Β.

Στο σημείο αυτό παρουσιάζουμε λεπτομερώς τις περιπτώσεις όπου μία παράμετρος της ομάδας Α είναι υποσύνολο της αντίστοιχης παραμέτρου στη Β.

Για τις διευθύνσεις IP αποστολέα και παραλήπτη:

- εάν η διεύθυνση της Β είναι ‘any’.

- αν με εφαρμογή της bitmask της B στη διεύθυνση της A το αποτέλεσμα είναι ίδιο με τη διεύθυνση της B. Για παράδειγμα αν η A είναι 147.27.3.71 και η B είναι 147.27.3.0/24.
- αν οι διευθύνσεις A και B είναι και οι δύο ανεστραμμένες και με εφαρμογή της bitmask της A στη διεύθυνση της B το αποτέλεσμα είναι ίδιο με τη διεύθυνση της A. για παράδειγμα αν η A είναι !147.27.3.0/24 και η B είναι !147.27.3.1.
- τέλος, εάν η διεύθυνση της A είναι σε κανονική μορφή, η B είναι ανεστραμμένη η εφαρμογή της bitmask της B στη διεύθυνση της A δίνει αποτέλεσμα διαφορετικό από τη διεύθυνση της B. Για παράδειγμα αν η A είναι 147.27.3.71 και η B είναι !50.100.150.0/24.

Για τις θύρες αποστολέα και παραλήπτη:

- εάν η θύρα της B είναι 'any'.
- αν η θύρα της A είναι μία τιμή ή ένα πεδίο τιμών τέτοιο ώστε να είναι ίσο ή να περιέχεται μέσα στο πεδίο τιμών της B. Για παράδειγμα αν η A είναι 10:20 και η B είναι 0:20.
- αν οι τιμές των θυρών είναι ανεστραμμένες για την A και την B και οι δύο τιμές είναι ίδιες ή το πεδίο τιμών της A εμπεριέχει το πεδίο τιμών της B. Για παράδειγμα αν η A είναι !0:20 και η B είναι !10:20.
- τέλος, εάν η τιμή ή το πεδίο τιμών της θύρας A είναι σε κανονική τιμή, η τιμή ή το πεδίο τιμών της θύρας της B είναι σε ανεστραμμένη μορφή και δεν υπάρχει κανένα κοινό σημείο ανάμεσα στα δύο πεδία τιμών. Για παράδειγμα αν η A έιναι 10:20 και η B είναι !30:60.

Η βελτιστοποίηση των κανόνων αποτελεί απαραίτητο κομμάτι της λειτουργίας του TUCNIDS καθώς επιτρέπει την εξέταση κάθε εισερχόμενου πακέτου με μία μόνο ομάδα κανόνων. Έχει ενδιαφέρον, όμως, να εξετάσουμε και την επίδραση που έχει η διαδικασία της βελτιστοποίησης σε ορισμένα στατιστικά χαρακτηριστικά των ομάδων κανόνων. Στον παρακάτω πίνακα παρουσιάζονται τα διάφορα χαρακτηριστικά πριν και μετά την βελτιστοποίηση, όταν χρησιμοποιούνται όλα τα αρχεία κανόνων που περιλαμβάνονται στην έκδοση 2.1.3 του Snort.

ΠΙΝΑΚΑΣ 5.1

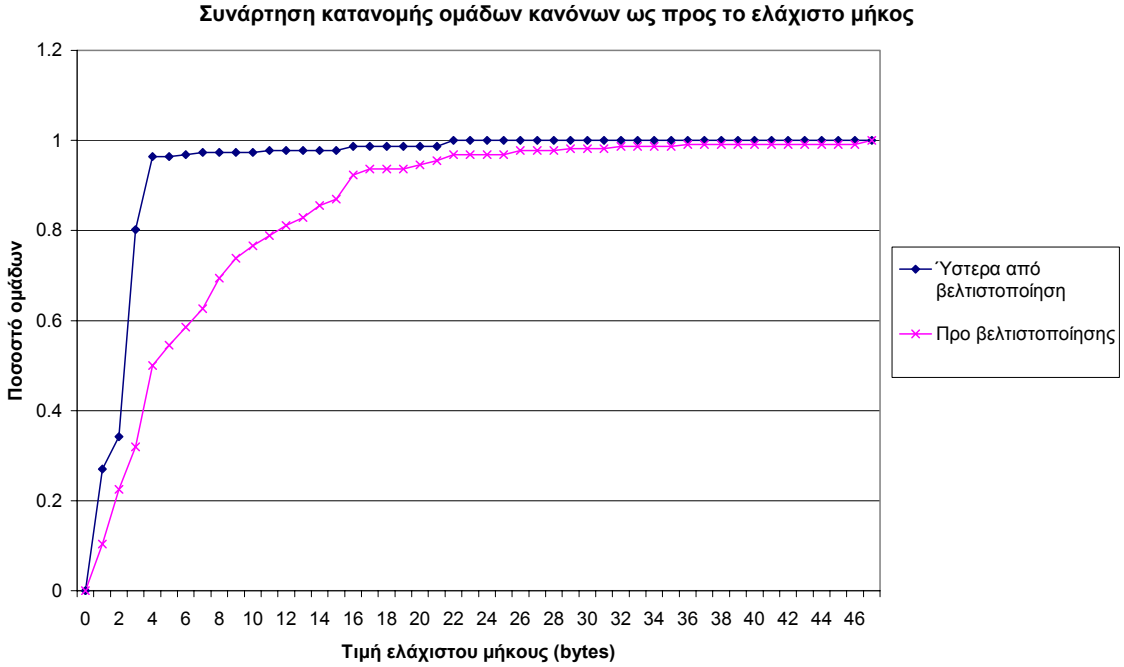
Στατιστικά χαρακτηριστικά πριν και μετά από τη βελτιστοποίηση

	Προ βελτιστοποίησης	Ύστερα από βελτιστοποίηση
Πλήθος ομάδων κανόνων	222	222
Πλήθος συμβολοσειρών	1187	6544
Μ.Ο. συμβολοσειρών ανά ομάδα	5,347	29,478
Διασπορά συμβολοσειρών ανά ομάδα	379,1	546,6
Διακύμανση μεγέθους ομάδας	1 – 264	1 – 294
Μέσο μήκος συμβολοσειρών	9,7	6,6
Διασπορά μήκους	90,2	38,4
Εύρος μήκους	1 – 122	1 – 122
Μέσος όρος ελάχιστου μήκους των ομάδων	7,5	3,0

Το πρώτο που παρατηρούμε είναι πως έχει αυξηθεί σημαντικά, σχεδόν εξαπλασιάστηκε, ο αριθμός των συμβολοσειρών ανά ομάδα κανόνων. Το γεγονός αυτό, όμως, δεν μας ανησυχεί ιδιαίτερα καθώς η ταχύτητα των αλγόριθμων αναζήτησης εξαρτάται σε πολύ μικρό βαθμό από το πλήθος των συμβολοσειρών προς αναζήτηση.

Το δεύτερο -και σημαντικότερο- που διαπιστώνουμε είναι μια μεγάλη μείωση του μέσου όρου των ελαχίστων μηκών για τις ομάδες κανόνων, από 7,5 bytes σε 3 bytes. Υπενθυμίζουμε ότι το ελάχιστο μήκος μίας ομάδας κανόνων είναι το μήκος της μικρότερης συμβολοσειράς που περιέχεται στη συγκεκριμένη ομάδα. Ο συγκεκριμένος μέσος όρος προκύπτει αθροίζοντας το ελάχιστο μήκος όλων των ομάδων και διαιρώντας με το πλήθος. Η μεγάλη μείωση του ελάχιστου μήκους έχει σημαντικό αντίκτυπο στους αλγορίθμους αναζήτησης, καθώς η ταχύτητα των τριών από τους τέσσερις υλοποιημένους αλγορίθμους –του Wu-Manber, του Boyer-Moore και του υβριδικού Aho-Corasick με Boyer-Moore– είναι αντιστρόφως ανάλογη του ελαχίστου μήκους.

Προκειμένου να γίνει πιο εύκολα αντιληπτή η μεταβολή του ελαχίστου μήκους λόγω της βελτιστοποίησης, παρουσιάζουμε στην εικόνα *5γ* το γράφημα για τη συνάρτηση κατανομής των ομάδων ως προς το ελάχιστο μήκος πριν και μετά από τη βελτιστοποίηση.



1.

Εικόνα 5γ) Η συνάρτηση κατανομής των ομάδων κανόνων ως προς το ελάχιστο μήκος.

Καθώς η μέση τιμή του ελαχίστου μήκους μειώνεται, το κόστος των τριών αλγόριθμων πλησιάζει ή ακόμα και υπερβαίνει το εγγυημένα γραμμικό κόστος του παραδοσιακού αλγορίθμου Aho-Corasick. Επομένως, είναι λογικό να περιμένουμε πως ο απλός Aho-Corasick θα χρησιμοποιείται για ένα μεγάλο αριθμό ομάδων από κανόνες. Η διαπίστωση αυτή θα μας φανεί χρήσιμη στην αξιολόγηση των αποτελεσμάτων από την αντιστοίχηση ενός αλγορίθμου αναζήτησης σε κάθε ομάδα κανόνων, αποτελέσματα που παρουσιάζονται στην παράγραφο 5.1.5.

5.1.3 Αναδιάταξη των ομάδων από κανόνες.

Προκειμένου να γίνεται αργότερα ο έλεγχος των παραμέτρων της επικεφαλίδας ενός πακέτου με σκοπό την επιλογή της κατάλληλης ομάδας κανόνων, είναι αναγκαία η

ύπαρξη ενός μηχανισμού επιλογής για τις ομάδες. Για την υλοποίηση του TUCNIDS θέλαμε ο μηχανισμός επιλογής ομάδας να είναι αρκετά απλός. Συγκεκριμένα, θέλαμε να γίνεται ένας απλός ακολουθιακός έλεγχος των ομάδων του πίνακα που αντιστοιχεί στο πρωτόκολλο του πακέτου και να σταματάμε στην πρώτη ομάδα κανόνων της οποίας οι παράμετροι ικανοποιούνται από τα στοιχεία του πακέτου.

Οποιοσδήποτε μηχανισμός επιλογής και αν χρησιμοποιηθεί, χρειάζεται να ικανοποιεί μια απαραίτητη προϋπόθεση για τη σωστή λειτουργία του συστήματος, δεδομένου ότι θέλουμε να εξετάζουμε κάθε πακέτο για τις συμβολοσειρές που περιέχονται σε μία μόνο ομάδα κανόνων. Πρέπει πρώτα να εξετάζουμε μήπως το εισερχόμενο πακέτο ικανοποιεί τις ομάδες από κανόνες που έχουν πολύ συγκεκριμένες παραμέτρους (για παράδειγμα η ομάδα “$HOME_NET 20 -> $EXTERNAL_NET :1023”) και στη συνέχεια να ελέγχουμε ομάδες με πιο γενικές παραμέτρους (για παράδειγμα, είναι προφανές ότι η ομάδα με παραμέτρους “any any -> any any” πρέπει να εξετάζεται τελευταία).

Για το λόγο αυτό ορίζουμε σε κάθε ομάδα κανόνων έναν «δείκτη γενικότητας». Ο δείκτης αυτός είναι απλά μία ακέραια τιμή, της οποίας η τιμή είναι ανάλογη με το πόσο γενικές είναι οι παράμετροι της συγκεκριμένης ομάδας. Μία ομάδα κανόνων με συγκεκριμένες παραμέτρους θα έχει μικρή τιμή για τον δείκτη γενικότητας, ενώ σε μια γενικότερη ομάδα θα αντιστοιχεί μεγαλύτερη τιμή.

Από τη στιγμή που έχουμε υπολογίσει τον δείκτη γενικότητας για όλες τις ομάδες κανόνων ενός πρωτοκόλλου, χρησιμοποιούμε έναν απλό αλγόριθμο ταξινόμησης ώστε να μαζέψουμε τις ομάδες με μικρή τιμή του δείκτη γενικότητας στην αρχή του πίνακα με τις ομάδες κανόνων.

5.1.4 Στατικός προσδιορισμός πιθανών συγκρούσεων.

Δεν πρέπει να αγνοήσουμε το ενδεχόμενο σύγκρουσης, δηλαδή τη περίπτωση κατά την οποία το εισερχόμενο πακέτο ικανοποιεί τις παραμέτρους περισσότερων από μίας ομάδων και δεν υπάρχει σχέση υποσυνόλου ανάμεσα στις συγκρουόμενες ομάδες. Για παράδειγμα, θεωρούμε την περίπτωση δύο ομάδων A και B όπου η ομάδα A

περιγράφεται από τις παραμέτρους “$HOME_NET 20 -> $EXTERNAL_NET any” ενώ η Β από τις παραμέτρους “$HOME_NET any -> $EXTERNAL_NET 80”. Είναι εμφανές ότι η Α δεν είναι υποσύνολο της Β και η Β δεν είναι υποσύνολο της Α. Εάν, όμως, το εισερχόμενο πακέτο είναι της μορφής “$HOME_NET 20 -> $EXTERNAL_NET 80” τότε πρέπει να εξεταστεί και με τις δύο ομάδες.

Η σειρά των ομάδων από κανόνες, όπως έχει προκύψει μετά από την αναδιάταξη μέσα στον πίνακα ενός πρωτοκόλλου, είναι και η σειρά με την οποία αργότερα ελέγχονται οι ομάδες μήπως ταιριάζει σε κάποια ένα εισερχόμενο πακέτο. Η πρώτη ομάδα που θα βρεθεί να ικανοποιείται από τις παραμέτρους του πακέτου είναι και η ομάδα με την οποία θα γίνει η εξέταση των δεδομένων του πακέτου. Από την άλλη, οι παράμετροι κάθε ομάδας μπορούν να χρησιμοποιηθούν για τον προσδιορισμό των άλλων ομάδων με τις οποίες υπάρχει περίπτωση σύγκρουσης. Με βάση τα δύο αυτά γεγονότα μπορούμε να προσδιορίσουμε στη φάση της προεπεξεργασίας το σύνολο των ομάδων με τις οποίες πιθανόν να συγκρούεται μία συγκεκριμένη ομάδα έτσι ώστε, σε περίπτωση που ένα πακέτο ταιριάξει σε μία ομάδα αλλά δεν βρεθεί ύποπτο περιεχόμενο, να γνωρίζουμε ποιες από τις υπόλοιπες ομάδες έχει νόημα να εξεταστούν για σύγκρουση.

Στην υλοποίηση του TUCNIDS ορίζουμε σε κάθε ομάδα κανόνων μία λίστα, στην οποία αποθηκεύονται οι ομάδες με τις οποίες ίσως να υπάρχει σύγκρουση. Ο τρόπος δημιουργίας τις λίστας είναι αρκετά απλός.

Έστω ότι μας ενδιαφέρει να βρούμε τη λίστα πιθανών συγκρούσεων για την ομάδα στη θέση j του πίνακα. Η λίστα είναι αρχικά κενή. Συγκρίνουμε την ομάδα στη θέση j διαδοχικά με όλες τις ομάδες στις θέσεις από j+1 μέχρι n. Συμβολίζουμε με Α την ομάδα j και με Β την ομάδα με την οποία γίνεται η σύγκριση ανά πάσα στιγμή. Σε κάθε σύγκριση ελέγχονται οι παρακάτω προτάσεις:

- η Α *δεν* είναι υποσύνολο της Β.
- καμία από τις ομάδες που έχουν βρεθεί μέχρι τώρα ως πιθανές συγκρούσεις της Α *δεν* είναι υποσύνολο της Β.
- συγκρίνουμε όλες τις παραμέτρους της Α με τις αντίστοιχες της Β. Για κάθε παράμετρο Ρ ισχύει είτε i) οι P_A και P_B είναι ίσες είτε ii) η P_A είναι υποσύνολο της P_B είτε iii) η P_B είναι υποσύνολο της P_A.

Σε περίπτωση που αληθεύουν και οι τρεις παραπάνω προτάσεις για τη συγκεκριμένη σύγκριση, η ομάδα Β προστίθεται στη λίστα των πιθανών συγκρούσεων της Α.

5.1.5 Επιλογή αλγόριθμου αναζήτησης.

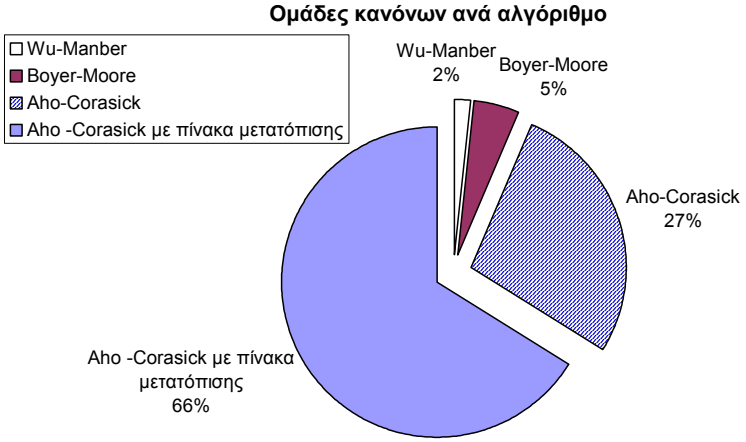
Αποτελεί το τελευταίο βήμα της προεπεξεργασίας. Στο σημείο αυτό χρησιμοποιούμε δύο χαρακτηριστικά, τον αριθμό των περιεχόμενων συμβολοσειρών και το ελάχιστο μήκος συμβολοσειράς, με σκοπό να επιλέξουμε τον καταλληλότερο αλγόριθμο για κάθε ομάδα κανόνων.

Ύστερα από διάφορους πειραματισμούς με τα δεδομένα μας καταλήξαμε στην ακόλουθη μέθοδο επιλογής αλγορίθμου, η οποία μας έδινε το μικρότερο συνολικό χρόνο επεξεργασίας πακέτων.

- Εάν το ελάχιστο μήκος συμβολοσειράς είναι 1 τότε επιλέγεται ο αλγόριθμος Aho-Corasick.
- Αλλιώς, εάν το ελάχιστο μήκος συμβολοσειράς είναι μικρότερο ή ίσο με 3 και το πλήθος συμβολοσειρών είναι μεγαλύτερο από 60 τότε επιλέγεται ο αλγόριθμος Aho-Corasick.
- Αλλιώς, εάν το ελάχιστο μήκος συμβολοσειράς είναι μικρότερο ή ίσο με 3 τότε επιλέγεται ο αλγόριθμος Aho-Corasick που χρησιμοποιεί πίνακα μετατόπισης Boyer-Moore.
- Αλλιώς, εάν στην ομάδα περιέχεται 1 συμβολοσειρά, τότε επιλέγεται ο αλγόριθμος Boyer-Moore.
- Αλλιώς, εάν το πλήθος των συμβολοσειρών στην ομάδα είναι μικρότερο ή ίσο με 50 τότε επιλέγεται ο αλγόριθμος Aho-Corasick που χρησιμοποιεί πίνακα μετατόπισης Boyer-Moore.
- Αλλιώς επιλέγεται ο αλγόριθμος Wu-Manber.

Η μέθοδος αυτή πιθανό να μην είναι η βέλτιστη, όμως μας έδωσε τα καλύτερα αποτελέσματα από τις εναλλακτικές που δοκιμάστηκαν. Στο παρακάτω γράφημα

φαίνεται το ποσοστό των ομάδων από κανόνες που αντιστοιχεί σε κάθε αλγόριθμο όταν χρησιμοποιείται η μέθοδος επιλογής που μόλις περιγράψαμε. Ο συνολικός αριθμός ομάδων από κανόνες που προκύπτει από τη χρήση όλων των αρχείων είναι 222.



Εικόνα 5δ) Η κατανομή των 222 ομάδων κανόνων ανά αλγόριθμο αναζήτησης.

Παρατηρούμε ότι οι δύο υλοποιήσεις του Aho-Corasick χρησιμοποιούνται για το μεγαλύτερο ποσοστό των ομάδων. Το αποτέλεσμα αυτό είναι λογικό αν σκεφτούμε ότι ένας μεγάλος αριθμός ομάδων έχει ελάχιστο μήκος συμβολοσειράς μικρότερο ή ίσο με 3, για τις οποίες περιπτώσεις ο αλγόριθμος Wu-Manber έχει τουλάχιστον γραμμικό κόστος και, επομένως, θα είναι πιο αργός από τον Aho-Corasick.

Αφού γίνει η επιλογή του αλγορίθμου για κάθε ομάδα κανόνων, η προεπεξεργασία ολοκληρώνεται με την αρχικοποίηση του αλγορίθμου κάθε ομάδας σύμφωνα με τα περιεχόμενα της.

5.2 Ο έλεγχος της διερχόμενης κίνησης δικτύου

Μόλις ολοκληρωθεί η αρχικοποίηση, το TUCNIDS είναι έτοιμο να εξετάσει την διερχόμενη κίνηση δικτύου. Σε αυτό το στάδιο λειτουργίας το TUCNIDS δέχεται αρχεία εισόδου υπό μορφή bytestream, δηλαδή αρχεία που περιέχουν συνεχόμενα πακέτα IP με τη μορφή που αυτά θα έρχονταν από το δίκτυο, χωρίς καμία αποκωδικοποίηση ή

επιπλέον πληροφορία. Για κάθε πακέτο αρχικά γίνεται κατηγοριοποίηση με βάση την επικεφαλίδα του και στη συνέχεια, σε περίπτωση που κριθεί απαραίτητο, ελέγχονται και τα δεδομένα του πακέτου για ύποπτα περιεχόμενα.

5.2.1 Κατηγοριοποίηση επικεφαλίδας πακέτου.

Στη συγκεκριμένη υλοποίηση του TUCNIDS υποστηρίζονται μόνο πακέτα τύπου IPv4. Σε κάθε πακέτο που ελέγχουμε, εξετάζονται από τις επικεφαλίδες τα πεδία που περιέχουν πληροφορίες για το πρωτόκολλο, το μήκος των δεδομένων και των επικεφαλίδων, τις διευθύνσεις αποστολέα και παραλήπτη και, εφόσον το πρωτόκολλο είναι TCP ή UDP, τις θύρες αποστολέα και παραλήπτη.

Στο σημείο αυτό πρέπει να αναφέρουμε ότι κάθε εισερχόμενο πακέτο θεωρείται ολοκληρωμένο και σωστό. Για το λόγο αυτό δε γίνεται έλεγχος στα checksum των επικεφαλίδων, στα flags ή σε άλλες παραμέτρους από τις επικεφαλίδες που περιγράφουν την ορθότητα του πακέτου ενώ, επίσης, δεν γίνεται έλεγχος για κατακερματισμό πακέτων. Καθώς το ενδιαφέρον επικεντρώνεται αποκλειστικά στο θέμα της απόδοσης του μηχανισμού αναζήτησης συμβολοσειρών, θεωρήσαμε αυτές τις επιπλέον λειτουργίες δευτερεύουσας σημασίας.

Αφού διαβάσουμε τις πληροφορίες που μας ενδιαφέρουν από την επικεφαλίδα του πακέτου, μπορούμε να προχωρήσουμε στην εύρεση της ομάδας κανόνων με την οποία θα εξεταστεί το πακέτο. Με βάση το πρωτόκολλο του πακέτου επιλέγεται ο αντίστοιχος πίνακας από ομάδες κανόνων και ξεκινάει η σύγκριση διαδοχική σύγκριση όλων των ομάδων με τις παραμέτρους του πακέτου.

Σε περίπτωση που δε βρεθεί καμία ομάδα κανόνων στην οποία να ταιριάζει το πακέτο, τότε προφανώς το πακέτο είναι ασφαλές (με βάση το σύνολο των κανόνων που χρησιμοποιούνται) και προχωράμε στην εξέταση του επόμενου πακέτου. Εάν, όμως, οι παράμετροι του πακέτου ταιριάζουν με κάποια ομάδα κανόνων, τότε προχωράμε στην εξέταση των δεδομένων (payload) του πακέτου.

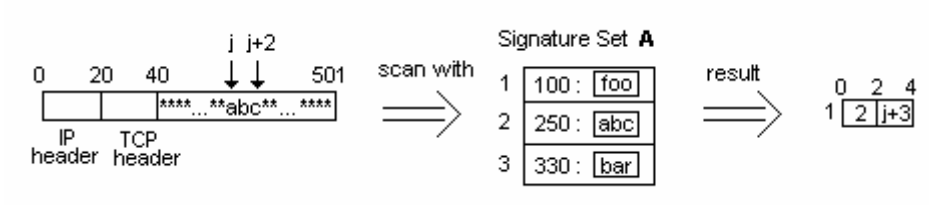
5.2.2 Εξέταση δεδομένων πακέτου

Σε κάθε ομάδα κανόνων αντιστοιχεί και ένας αλγόριθμος αναζήτησης, όπως περιγράφουμε και στην παράγραφο 5.1.5. Όταν η λειτουργία της κατηγοριοποίησης μας απαντήσει πως η επικεφαλίδα του πακέτου ταιριάζει σε κάποια ομάδα κανόνων, τότε χρησιμοποιείται ο αντίστοιχος αλγόριθμος για να εξεταστούν τα δεδομένα του πακέτου. Ο έλεγχος των δεδομένων γίνεται για την ανεύρεση των ύποπτων συμβολοσειρών που επαληθεύουν τους κανόνες της συγκεκριμένης ομάδας.

Το αποτέλεσμα της εξέτασης των δεδομένων είναι μία λίστα 32 bit τιμών, στην οποία μπορούν να περιέχονται από 0 (κενή λίστα) μέχρι και 256 τιμές. Κάθε στοιχείο της λίστας ισοδυναμεί με την εύρεση του περιεχόμενου που υποδηλώνει ένας κανόνας σε κάποιο σημείο των δεδομένων. Η κάθε τιμή 32 bit που περιέχεται στη λίστα αναλύεται σε δύο τιμές 16 bit ως εξής: τα 16 πιο σημαντικά bit (MSB) ισοδυναμούν με τον αύξων αριθμό του κανόνα της ομάδας που επαληθεύτηκε (όχι το sid του κανόνα), ενώ τα 16 λιγότερο σημαντικά bit (LSB) αντιστοιχούν στη θέση από την αρχή των *δεδομένων* του πακέτου (όχι από την αρχή ολόκληρου του πακέτου) που βρίσκεται αμέσως μετά το τέλος της συμβολοσειράς που βρέθηκε.

Προκειμένου να γίνει πιο κατανοητός ο τρόπος με τον οποίο αποθηκεύονται τα αποτελέσματα των αλγορίθμων, ας θεωρήσουμε το ακόλουθο παράδειγμα. Έχουμε ένα πακέτο TCP/IP συνολικού μεγέθους 500 bytes, στο οποίο η επικεφαλίδα IP είναι μεγέθους 20 bytes και η επικεφαλίδα TCP είναι άλλα 20 bytes. Όλοι οι χαρακτήρες των δεδομένων (payload) του πακέτου είναι '*' με εξαίρεση την συμβολοσειρά "abc" που βρίσκεται στις θέσεις j ως j+2 από την αρχή του πακέτου. Η ομάδα κανόνων, έστω A, που αντιστοιχεί στο συγκεκριμένο πακέτο περιέχει τρεις κανόνες: ο πρώτος κανόνας της ομάδας έχει sid=100 και περιεχόμενο "foo", ο δεύτερος κανόνας έχει sid=250 και περιεχόμενο "abc", ενώ ο τρίτος κανόνας έχει sid=330 και περιεχόμενο "bar". Μετά τον έλεγχο των δεδομένων από τον αλγόριθμο αναζήτησης θα επιστραφεί μία λίστα με έναν μόνο αριθμό, του οποίου τα 16 MSB έχουν την τιμή 2 καθώς επαληθεύτηκε ο δεύτερος από τους τρεις κανόνες τις ομάδας, ενώ τα 16 LSB είναι ίσα με j+3-40: j+3 επειδή αυτή η θέση *του πακέτου* βρίσκεται αμέσως μετά το τέλος της συμβολοσειράς και -40 επειδή οι

δύο επικεφαλίδες του πακέτου έχουν συνολικό μήκος 40 bytes. Το παράδειγμα που μόλις περιγράψαμε φαίνεται στην ακόλουθη εικόνα.



Εικόνα 5ε) Παράδειγμα αποτελέσματος μίας αναζήτησης στο TUCNIDS.

Ο λόγος που επιλέξαμε κάθε επιτυχής αναζήτηση να μας επιστρέφει μία λίστα με τις εμφανίσεις των συμβολοσειρών είναι ώστε να μπορούμε με ευκολία στο μέλλον να επεκτείνουμε τη λειτουργικότητα του TUCNIDS προσθέτοντας πιο περίπλοκους ελέγχους. Προς το παρόν, πάντως, δεν γίνεται κάποια επιπλέον επεξεργασία πάνω στα αποτελέσματα της αναζήτησης, απλά σημειώνουμε ότι βρέθηκε ένα ακόμη πακέτο που ενεργοποιεί κάποιον κανόνα.

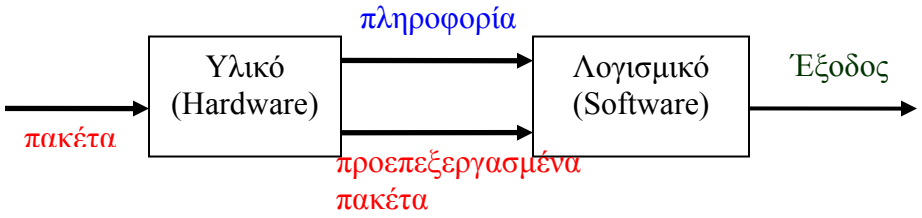
Εάν κάποιο πακέτο ταιριάζει σε μία ομάδα κανόνων Α αλλά ο έλεγχος του περιεχομένου του πακέτου δεν εντοπίσει καμία από τις συμβολοσειρές, δηλαδή η λίστα αποτελεσμάτων είναι άδεια, τότε εξετάζουμε το ενδεχόμενο να υπάρχει σύγκρουση ομάδων για το συγκεκριμένο πακέτο. Για να γίνει ο έλεγχος σύγκρουσης, συγκρίνουμε το πακέτο με κάθε ομάδα που περιέχεται στη λίστα των πιθανών συγκρούσεων της ομάδας Α. Σε περίπτωση που το πακέτο ταιριάζει με κάποια ομάδα Β από τις ομάδες πιθανής σύγκρουσης τότε γίνεται δεύτερη εξέταση των περιεχομένων του πακέτου, αυτή τη φορά με τους κανόνες που περιέχονται στη Β. Εάν και πάλι δεν βρεθεί ύποπτο περιεχόμενο, συνεχίζουμε την σύγκριση του πακέτου με τα περιεχόμενα της λίστας πιθανών συγκρούσεων για την ομάδα Α, όχι για την Β, καθώς υπάρχει και η περίπτωση σύγκρουσης ανάμεσα σε περισσότερες από δύο ομάδες. Τριπλή σύγκρουση μπορεί να υπάρξει, για παράδειγμα, ανάμεσα στις ομάδες Α “any 80 -> any 120”, Β “$HOME_NET any -> any any” και Γ “any any-> $EXTERNAL_NET any” σε περίπτωση που οι παράμετροι του πακέτου είναι “$HOME_NET 80 -> $EXTERNAL_NET 120”. Το φαινόμενο της τριπλής σύγκρουσης είναι πολύ σπάνιο, παρόλα αυτά δεν πρέπει να αγνοηθεί.

5.3 Συνδυασμός του TUCNIDS με φίλτρα σε υλικό (hardware).

Ένα σύστημα ανίχνευσης εισβολών δικτύου υλοποιημένο αποκλειστικά σε λογισμικό θα είναι σίγουρα ευέλικτο και ισχυρό, όμως ανακύπτει το ζήτημα της ταχύτητας. Καθώς οι ταχύτητες σύνδεσης αυξάνονται με πολύ γρηγορότερους ρυθμούς από ότι αυξάνεται ταχύτητα με την οποία ένας υπολογιστής μπορεί να ελέγχει τα δεδομένα, το λογισμικό από μόνο του σύντομα θα δυσκολεύεται να ανταπεξέλθει στο ζητούμενο έργο, ειδικά κάτω από συνθήκες έντονου φόρτου.

Από την άλλη πλευρά, το υλικό σαφώς είναι κατά πολύ ταχύτερο. Έρευνες [23,24,25,26,27,28] που έχουν γίνει σχετικά με την αναζήτηση συμβολοσειρών μέσω υλικού δείχνουν ότι μπορούν να επιτευχθούν ρυθμοί επεξεργασίας αρκετών Gigabit ανά δευτερόλεπτο. Ανακύπτει, όμως, το ζήτημα των περιορισμών του χώρου. Η υλοποίηση ενός ολόκληρου συστήματος ανίχνευσης εισβολών είναι αρκετά περίπλοκη και έχει μεγάλες απαιτήσεις σε χώρο. Έτσι, ακόμα και στην περίπτωση που κάποιος πληρώσει για την αγορά ενός συστήματος γρήγορου και με επαρκή χώρο, είναι πολύ πιθανό στο μέλλον να αυξηθούν οι απαιτήσεις λόγω της προσθήκης νέων κανόνων με αποτέλεσμα να μη χωράει πλέον το σύστημα ανίχνευσης στο υλικό.

Έχοντας υπόψη μας τα δύο αυτά γεγονότα, αναζητήσαμε τρόπους με τους οποίους θα μπορούσαν να συνδυαστούν σχετικά απλές υλοποιήσεις-φίλτρα σε υλικό με ένα σύστημα ανίχνευσης εισβολών σε λογισμικό με στόχο τη μείωση του υπολογιστικού φόρτου και την αύξηση της συνολικής απόδοσης. Μια γενική άποψη του συνδυάσμού ανάμεσα στο υλικό και το λογισμικό παρουσιάζεται στην εικόνα *5στ*.



Εικόνα 5στ) Μια γενική συνδεσμολογία υλικού και λογισμικού.

Ανάλογα με τις δυνατότητες και τον τρόπο λειτουργίας του υλικού, διακρίνουμε τρεις γενικές περιπτώσεις ως προς τον τρόπο συνεργασίας και υποβοήθησης του λογισμικού από το υλικό:

α) *Το υλικό επιτρέπει τη διέλευση από το λογισμικό όλων των πακέτων και παρέχει χρήσιμη πληροφορία στο λογισμικό*. Το ζητούμενο στην περίπτωση αυτή είναι να παρέχεται στο λογισμικό οι κατάλληλες πληροφορίες ώστε να αποφευχθεί κάποια χρονοβόρα επεξεργασία.

β) *Το υλικό έχει τη δυνατότητα να παρακάμπτει το λογισμικό για ορισμένα πακέτα που σίγουρα δεν πληρούν τους κανόνες, ενώ για τα πακέτα που φτάνουν στο λογισμικό δεν παρέχονται επιπλέον πληροφορίες*. Ο χρόνος που χρειάζεται το λογισμικό για την εξέταση της διερχόμενης κίνησης εξαρτάται σε μεγάλο βαθμό από το μέγεθος της κίνησης. Η μείωση του αριθμού των πακέτων που εξετάζει το λογισμικό συνεπάγεται και μείωση του απαιτούμενου χρόνου.

γ) *Το υλικό έχει τη δυνατότητα να παρακάμπτει το λογισμικό για ορισμένα πακέτα που σίγουρα δεν πληρούν τους κανόνες, ενώ παρέχονται επιπλέον πληροφορίες για τα πακέτα που φτάνουν στο λογισμικό*. Αποτελεί συνδυασμό των προηγουμένων δύο περιπτώσεων, συνδυάζοντας τα οφέλη τους.

Ύστερα από μελέτη του προβλήματος και των λειτουργιών που θα μπορούσε να μας παρέχει το υλικό, καταλήξαμε σε δύο συστήματα υλικού τα οποία περιγράφονται στη συνέχεια. Τα αποτελέσματα από την προσομοίωση της λειτουργίας των δύο αυτών συστημάτων παρουσιάζονται στο κεφάλαιο 6.

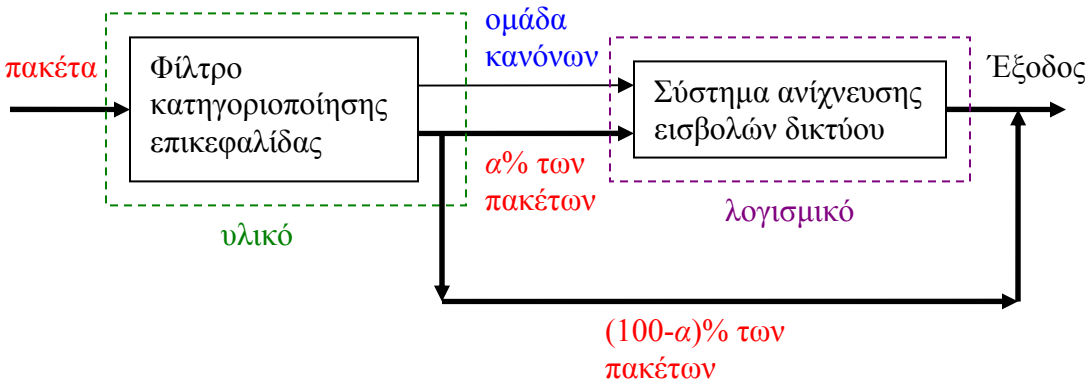
5.3.1 Φίλτρο κατηγοριοποίησης επικεφαλίδας πακέτου (header classification).

Εκτός από την αναζήτηση συμβολοσειρών μέσα στα δεδομένα του πακέτου, ένα σημαντικό ποσοστό του χρόνου επεξεργασίας καταναλώνεται στην εξέταση των παραμέτρων της επικεφαλίδας του πακέτου με της διάφορες ομάδες κανόνων, μέχρι να επιλεγεί η κατάλληλη. Οι πειραματικές μετρήσεις που παρουσιάζουμε στην παράγραφο 6.3.1 δείχνουν ότι η κατηγοριοποίηση για μέσο μήκος πακέτων 300 bytes καταλαμβάνει

περίπου το 58.7% του συνολικού χρόνου επεξεργασίας ενός πακέτου. Επιπλέον, αν σκεφτούμε ότι κάτω από τυπικές συνθήκες λειτουργίας αρκετά από τα εισερχόμενα πιθανόν να μην αντιστοιχούν σε καμία ομάδα κανόνων, το ποσοστό του χρόνου που αντιστοιχεί στην κατηγοριοποίηση της επικεφαλίδας του πακέτου αυξάνεται σημαντικά.

Δεδομένου ότι οι ομάδες κανόνων δεν αλλάζουν κατά τη διάρκεια της λειτουργίας ενός συστήματος ανίχνευσης, μπορούμε να κατασκευάσουμε σε υλικό ένα φίλτρο που θα εκτελεί το έργο της εξέτασης των παραμέτρων του πακέτου και της επιλογής της ομάδας κανόνων που αντιστοιχεί στο πακέτο.

Ο τρόπος με τον οποίο συνδέονται του φίλτρου υλικού με το λογισμικό φαίνεται στην ακόλουθη εικόνα. Το ποσοστό *α*% που αναγράφεται στην εικόνα *5ζ* είναι ενδεικτικό: η ακριβής αριθμός των πακέτων περιγράφεται από την εξίσωση 5.1 που βρίσκεται στην υποενότητα 5.3.1.



Εικόνα 5ζ) Η συνδεσμολογία του φίλτρου κατηγοριοποίησης με το λογισμικό

Το φίλτρο κατηγοριοποίησης επικεφαλίδας υπάγεται στην τρίτη κατηγορία, δηλαδή μειώνει το ποσοστό πακέτων που προοθούνται στο λογισμικό ταυτόχρονα παρέχει χρήσιμη πληροφορία.

Η λειτουργία του φίλτρου είναι απλή. Από κάθε εισερχόμενο πακέτο αρχικά διαβάζονται οι παράμετροι της επικεφαλίδας του με τις οποίες θα γίνει η κατηγοριοποίηση. Οι παράμετροι αυτοί είναι το πρωτόκολλο, οι διευθύνσεις IP αποστολέα και παραλήπτη, οι θύρες αποστολέα και παραλήπτη (εάν το πακέτο είναι TCP ή UDP), το είδος του ICMP (σε περίπτωση πακέτου ICMP) και ο αριθμός του

πρωτοκόλλου μεταφοράς (για πακέτα που χρησιμοποιούν διαφορετικό πρωτόκολλο από TCP, UDP και ICMP).

Το φίλτρο έχει καταχωρημένες τις παραμέτρους για όλες τις ομάδες κανόνων, ενώ η διάταξη των ομάδων για κάθε πρωτόκολλο είναι ακριβώς η ίδια όπως στο σύστημα του λογισμικού. Έτσι, το φίλτρο κατηγοριοποίησης επικεφαλίδας μπορεί να συγκρίνει τις παραμέτρους που μόλις διαβάστηκαν με τις παραμέτρους όλων των ομάδων για το συγκεκριμένο πρωτόκολλο ώστε να επιλέξει την κατάλληλη ομάδα.

Η σύγκριση με τις ομάδες μπορεί να γίνεται είτε σειριακά, όπως γίνεται στο λογισμικό, οπότε το φίλτρο σταματάει στην πρώτη ομάδα που ταιριάζει (δηλαδή την ομάδα με τις πιο συγκεκριμένες παραμέτρους), είτε με παράλληλο τρόπο, στην οποία περίπτωση είναι απαραίτητη μία μέθοδος κωδικοποίησης προτεραιότητας ώστε να επιλέγεται η ομάδα με τον μικρότερο αύξων αριθμό. Η ύπαρξη αυτού του μηχανισμού προτεραιότητας είναι αναγκαία καθώς σε συγκεκριμένες τιμές των παραμέτρων του πακέτου μπορούν να αντιστοιχούν πολλές ομάδες, αλλά εμείς ενδιαφερόμαστε για την πιο συγκεκριμένη.

Η έξοδος του φίλτρου κατηγοριοποίησης είναι ο αύξων αριθμός της ομάδας που βρέθηκε να ταιριάζει με τις παραμέτρους του πακέτου και αποστέλλεται στο λογισμικό μαζί με το πακέτο. Με τον τρόπο αυτό, το λογισμικό μπορεί αμέσως να ξεκινήσει την αναζήτηση μέσα στα δεδομένα του πακέτου για τις συμβολοσειρές της συγκεκριμένης ομάδας, εξετάζοντας από την επικεφαλίδα του πακέτου μόνο το πρωτόκολλο και το μήκος της επικεφαλίδας. Σε περίπτωση που δε βρεθεί καμία ομάδα κανόνων, τότε το πακέτο είναι σίγουρα ασφαλές και μπορούμε να παρακάμψουμε τελείως το σύστημα ανίχνευσης του λογισμικού, μειώνοντας, έτσι, τον φόρτο του συστήματος.

Στο σημείο αυτό πρέπει να αναφέρουμε ότι ο έλεγχος για τυχόν συγκρούσεις ανάμεσα σε διαφορετικές ομάδες κανόνων παραμένει αρμοδιότητα του λογισμικού και γίνεται όπως και προηγουμένως. Επίσης, σε περίπτωση που θέλαμε να εκτελούμε περισσότερους ελέγχους στις παραμέτρους της επικεφαλίδας του πακέτου, για παράδειγμα έλεγχο του checksum ή του TTL, μπορούμε να εντάξουμε και τις λειτουργίες αυτές στο φίλτρο κατηγοριοποίησης.

5.3.2 Φίλτρο προσεγγιστικής αναγνώρισης συμβολοσειρών (approximate string matching).

Η αναζήτηση των ύποπτων συμβολοσειρών αποτελεί το πλέον χρονοβόρο κομμάτι στη διαδικασία της εξέτασης του πακέτου. Επιπλέον, είναι αρκετά πολύπλοκη διαδικασία ώστε να είναι δύσκολα υλοποιήσιμη σε υλικό. Όμως, το υλικό μπορεί εύκολα να χρησιμοποιηθεί για την υλοποίηση ενός φίλτρου που χρησιμοποιεί μεθόδους προσεγγιστικής αναγνώρισης συμβολοσειρών, όπως είναι για παράδειγμα τα bloom filters [16].

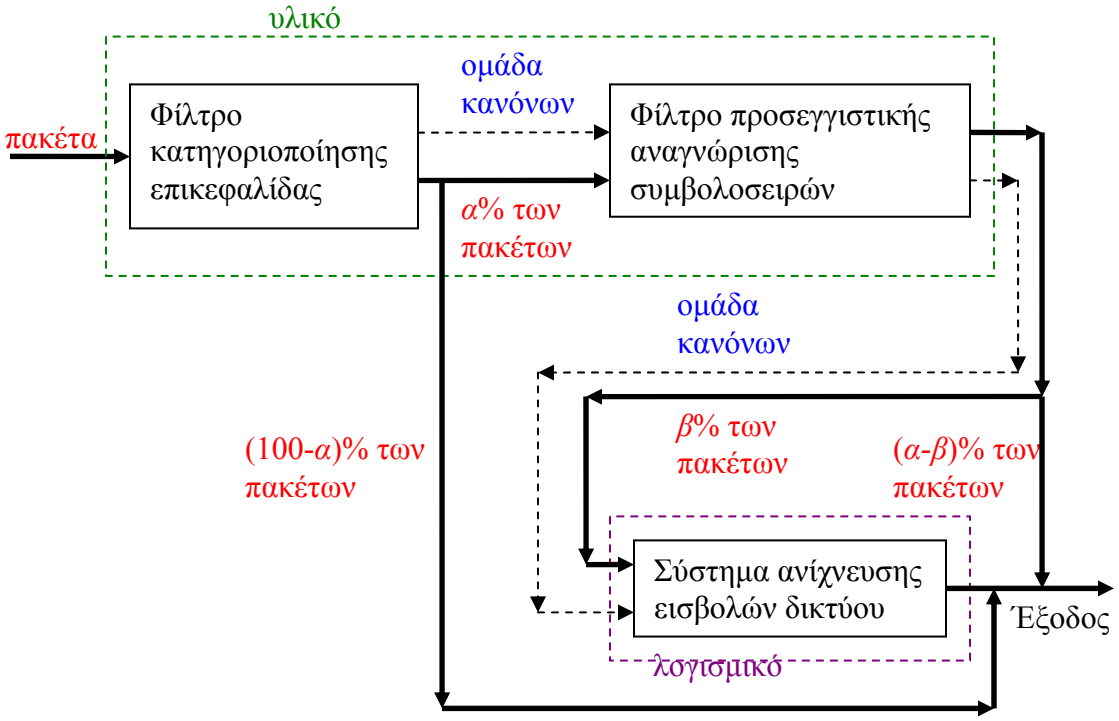
Μια μέθοδος προσεγγιστικής αναγνώρισης συμβολοσειρών δέχεται σαν είσοδο ένα σύνολο από συμβολοσειρές προς αναζήτηση και ένα κείμενο και επιστρέφει με απόλυτη βεβαιότητα αρνητική απάντηση, δηλαδή ότι δεν υπάρχει καμία από τις συμβολοσειρές μέσα στο κείμενο, ή επιστρέφει θετική απάντηση, που σημαίνει ότι υπάρχει κάποια από τις συμβολοσειρές μέσα στο κείμενο. Υπάρχει, επίσης, η πιθανότητα να λάβουμε θετική απάντηση ακόμη και όταν το κείμενο δεν περιέχει καμία από τις συμβολοσειρές. Στην περίπτωση αυτή λέμε πως έχουμε ψευδή αναγνώριση (false positive). Πάντως, δεν υπάρχει καμία περίπτωση ψευδούς απόρριψης (false negative).

Αν λάβουμε υπόψην μας ότι στην τυπική περίπτωση λειτουργίας ενός συστήματος ανίχνευσης εισβολών τα περισσότερα εισερχόμενα πακέτα δεν θα περιέχουν ύποπτες συμβολοσειρές, είναι προφανές πως ένα φίλτρο προσεγγιστικής αναγνώρισης μπορεί να μειώσει σημαντικά τον αριθμό των πακέτων που ελέγχονται από το λογισμικό.

Υποθέτουμε πως για να είναι αποδοτική η χρήση ενός φίλτρου προσεγγιστικής αναγνώρισης συμβολοσειρών σε ένα σύστημα ανίχνευσης εισβολών δικτύου, θα πρέπει να χρησιμοποιείται σε συνδυασμό με το φίλτρο κατηγοριοποίησης επικεφαλίδας που περιγράφηκε προηγουμένως. Η υπόθεση αυτός οφείλεται στο γεγονός ότι ο αριθμός κανόνων που μπορούν να επαληθευτούν για μία ομάδα είναι σαφώς μικρότερος από τον συνολικό αριθμό κανόνων. Επίσης, υποθέτουμε ότι το φίλτρο υλικού έχει τη δυνατότητα να επιλέγει ανάμεσα σε N διαφορετικά σύνολα συμβολοσειρών για προσεγγιστικής αναγνώρισης, όπου N ο συνολικός αριθμός από ομάδες κανόνων.

Η συνδεσμολογία των δύο φίλτρων και του συστήματος λογισμικού φαίνεται στην ακόλουθη εικόνα. Τα ποσοστά α% και β% της εικόνας 5η είναι ενδεικτικά: ο

ακριβής αριθμός των πακέτων σε κάθε περίπτωση περιγράφονται από τις εξισώσεις 5.1 και 5.2 αντίστοιχα.



Εικόνα 5η) Η συνδεσμολογία των δύο φίλτρων και του λογισμικού

Το φίλτρο προσεγγιστικής αναγνώρισης συμβολοσειρών δέχεται σαν είσοδο δύο δεδομένα: το πακέτο και τον αριθμό της ομάδας κανόνων που ταιριάζει στο πακέτο αυτό. Η δεύτερη πληροφορία μας έρχεται ως αποτέλεσμα της κατηγοριοποίησης της επικεφαλίδας από το αντίστοιχο φίλτρο.

Ανάλογα με το πρωτόκολλο του πακέτου και την πληροφορία από το φίλτρο κατηγοριοποίησης, επιλέγεται ένα από τα N σύνολα συμβολοσειρών του φίλτρου και εκτελείται μία προσεγγιστική αναγνώριση στα δεδομένα του πακέτου. Ο έλεγχος θα επιστρέψει είτε αρνητικό αποτέλεσμα, οπότε το πακέτο είναι σίγουρα καθαρό και δε χρειάζεται να ελεγχθεί από το λογισμικό, είτε θετικό αποτέλεσμα, γεγονός που σημαίνει πως το πακέτο πρέπει να εξεταστεί από το λογισμικό προκειμένου να είμαστε σίγουροι.

Όπως αναφέραμε, η επιλογή ενός από τα N σύνολα γίνεται με βάση το πρωτόκολλο και την ομάδα κανόνων που μας υποδεικνύει το φίλτρο κατηγοριοποίησης. Οι συμβολοσειρές που περιέχονται στο σύνολο αυτό είναι όλες οι συμβολοσειρές που

περιέχονται στην ομάδα κανόνων που μας υπέδειξε το φίλτρο κατηγοριοποίησης *αλλά και όλες οι συμβολοσειρές που περιλαμβάνονται στις ομάδες που ίσως να έχουν σύγκρουση με τη συγκεκριμένη ομάδα*. Η αναζήτηση των συμβολοσειρών που περιέχονται σε όλες τις πιθανές συγκρουόμενες ομάδες κανόνων είναι απαραίτητη αν θέλουμε να εγγυόμαστε για την ασφάλεια του συστήματος ανίχνευσης. Σε διαφορετική περίπτωση, υπάρχει ο κίνδυνος να θεωρηθεί ασφαλές ένα πακέτο του οποίου το περιεχόμενο δεν επαληθεύει κανέναν από τους κανόνες της ομάδας που επιλέχθηκε από το φίλτρο κατηγοριοποίησης, αλλά επαληθεύει τους κανόνες κάποιας από τις συγκρουόμενες ομάδες.

Η αποδοτική λειτουργία του φίλτρου προσεγγιστικής αναγνώρισης στηρίζεται στην υπόθεση ότι το υλικό μπορεί να υποστηρίζει N διαφορετικά σύνολα συμβολοσειρών για προσεγγιστική αναζήτηση. Όμως, ένα τέτοιο σύστημα μπορεί να είναι υπερβολικά περίπλοκο για να υλοποιηθεί αποδοτικά σε υλικό. Στην περίπτωση που το υλικό δεν έχει τη δυνατότητα υποστήριξης N συνόλων, υπάρχει εναλλακτική λύση: να σχεδιαστεί το υλικό έτσι ώστε να εκτελεί προσεγγιστική αναγνώριση για ένα μόνο σύνολο συμβολοσειρών, στο οποίο θα περιέχονται οι συμβολοσειρές *όλων* των κανόνων.

Με τη χρήση ενός συνόλου για όλους τους κανόνες η υλοποίηση γίνεται αρκετά πιο απλή. Η αρνητική συνέπεια της συγκεκριμένης απλοποίησης είναι ότι θα υπάρχει ένα σημαντικό ποσοστό «ημι-ψευδών» αναγνωρίσεων. Με τον όρο «ημι-ψευδής αναγνώριση» εννοούμε την περίπτωση όπου το υλικό πράγματι εντοπίζει κάποια από τις συμβολοσειρές μέσα στο πακέτο, όμως αυτή η συμβολοσειρά δεν περιλαμβάνεται σε κάποια ομάδα κανόνων που αντιστοιχεί στο συγκεκριμένο πακέτο και, επομένως, δεν θεωρείται ύποπτη.

Η τελική επιλογή για τον τρόπο υλοποίησης του φίλτρου προσεγγιστικής αναγνώρισης συμβολοσειρών, δηλαδή για το αν θα χρησιμοποιείται 1 ή N σύνολα, θα πρέπει να γίνει με βάση τις δυνατότητες του υλικού.

5.4 Θεωρητική ανάλυση της απόδοσης των φίλτρων υλικού.

Στο σημείο αυτό θα προσπαθήσουμε να εκφράσουμε μαθηματικά την επίδραση που μπορεί να έχει η χρήση των δύο φίλτρων υλικού τόσο στον αριθμό των πακέτων που προωθούνται στο λογισμικό όσο και στον χρόνο επεξεργασίας των πακέτων.

Αρχικά υποθέτουμε ότι η εισερχόμενη κίνηση δικτύου μπορεί να περιγραφεί στατιστικά μέσω δύο πιθανοτήτων: η πρώτη τιμή, την οποία συμβολίζουμε με h, εκφράζει την πιθανότητα οι παράμετροι της επικεφαλίδας ενός διερχόμενου πακέτου να ταιριάζουν με κάποιας ομάδα κανόνων (δηλαδή την πιθανότητα να έχουμε header match). Η δεύτερη πιθανότητα, με το συμβολισμό p, εκφράζει την *δεσμευμένη πιθανότητα* ένα πακέτο να ικανοποιεί το περιεχόμενο κάποιου κανόνα μίας ομάδας *δεδομένου ότι το πακέτο ταιριάζει με την ομάδα αυτή* (δηλαδή την πιθανότητα το πακέτο που έκανε header match να κάνει **και** content match).

Επίσης, θεωρούμε την πιθανότητα f με την οποία γίνεται ψευδής αναγνώριση στο φίλτρο προσεγγιστικής αναγνώρισης συμβολοσειρών. Καθώς τα h, p και f είναι πιθανότητες, ισχύει ότι $0 \leq h,p,f \leq 1$. Τέλος, υποθέτουμε ότι η διερχόμενη κίνηση δικτύου αποτελείται από N πακέτα μέσου μήκους l_{mean}.

Στην πράξη, οι πιθανότητες h και p δεν εξαρτώνται μόνο από τη διερχόμενη κίνηση αλλά και από το σύνολο των χρησιμοποιούμενων κανόνων. Για παράδειγμα, εάν χρησιμοποιούμε πολύ γενικούς κανόνες του τύπου “any any -> any any” η πιθανότητα h θα είναι ίση με τη μονάδα. Παρόλα αυτά, για ένα σταθερό σύνολο κανόνων μπορούμε να δεχτούμε ότι η διερχόμενη κίνηση δικτύου περιγράφεται μακροσκοπικά από αυτές τις δύο πιθανότητες.

5.4.1 Μείωση του αριθμού των πακέτων που προωθούνται στο λογισμικό.

Όπως αναφέρθηκε στην υποενότητα 5.3, τα φίλτρα υλικού μπορούν να επιλέξουν να μην προωθήσουν ορισμένα πακέτα στο λογισμικό ανάλογα με τα αποτελέσματα που θα επιστρέψει ο έλεγχος που εκτελούνε. Θα προσπαθήσουμε να εκτιμήσουμε τον αριθμό

των πακέτων που τελικά προωθούνται στο λογισμικό χρησιμοποιώντας ορισμένες σταθερές παραμέτρους.

Αρχικά εξετάζουμε την περίπτωση που χρησιμοποιείται μόνο το φίλτρο κατηγοριοποίησης. Καθώς η πιθανότητα h ισούται με την πιθανότητα ένα πακέτο να ταιριάζει με κάποια ομάδα κανόνων, το φίλτρο κατηγοριοποίησης θα παρακάμψει το λογισμικό για τα $(1-h)*N$ πακέτα που δεν ταιριάζουν σε καμία ομάδα κανόνων. Τα υπόλοιπα $h*N$ πακέτα προωθούνται στο λογισμικό για περεταίρω εξέταση. Το αποτέλεσμα αυτό το ονομάζουμε εξίσωση 5.1.

$$N' = h*N$$

Εξίσωση 5.1: Ο αριθμός των πακέτων που τελικά εξετάζονται από το λογισμικό όταν χρησιμοποιείται φίλτρο κατηγοριοποίησης σε υλικό.

Ας δούμε τι γίνεται στην περίπτωση που χρησιμοποιείται ο συνδυασμός των δύο φίλτρων υλικού. Σε πρώτο στάδιο το φίλτρο κατηγοριοποίησης διαπιστώνει, όπως περιγράψαμε προηγουμένως, ότι τα $(1-h)*N$ πακέτα είναι ασφαλή και προωθεί τα υπόλοιπα $h*N$ πακέτα στο φίλτρο προσεγγιστικής αναγνώρισης για επιπλέον ελέγχο.

Σε κάθε πακέτο που φτάνει στο φίλτρο προσεγγιστικής αναγνώρισης, η πιθανότητα να περιέχεται ύποπτο περιεχόμενο είναι ίση με p. Συνεπώς, από τα $h*N$ πακέτα που φτάνουν στο δεύτερο φίλτρο, τα $p*(h*N)$ [i] θα προωθηθούν σίγουρα στο λογισμικό καθώς περιέχουν ύποπτο περιεχόμενο και θα αναγνωριστούν σωστά από το φίλτρο. Για κάθε ένα από τα πακέτα που φτάνουν στο δεύτερο φίλτρο αλλά δεν περιέχουν ύποπτο πακέτο υπάρχει πιθανότητα f ψευδούς αναγνώρισης και προώθησης στο λογισμικό, συνεπώς ο αριθμός των πακέτων στα οποία γίνεται ψευδής αναγνώριση και προώθηση από το δεύτερο φίλτρο είναι $f*(1-p)*(h*N)$ [ii].

Αθροίζοντας τις δύο εξισώσεις [i] και [ii] καταλήγουμε στην εξίσωση 5.2.

$$N' = (p + f - f*p)*h*N$$

Εξίσωση 5.2: ο αριθμός των πακέτων που τελικά εξετάζονται από το λογισμικό όταν χρησιμοποιείται ο συνδυασμός των δύο φίλτρων υλικού.

Ας θεωρήσουμε ένα αυθαίρετο δικτύου όπου το 50% των εισερχόμενων πακέτων ταιριάζει με κάποια ομάδα κανόνων, το 20% των πακέτων που ταιριάζουν σε κάποια ομάδα περιέχουν ύποπτο περιεχόμενο και το φίλτρο προσεγγιστικής αναγνώρισης συμβολοσειρών έχει 10% πιθανότητα ψευδούς αναγνώρισης. Για το δίκτυο αυτό, η χρήση των δύο φίλτρων υλικού θα μειώσει τον αριθμό των πακέτων που εξετάζονται από το λογισμικό κατά έναν παράγοντα 1/[(0.2 + 0.1 - 0.1*0.2)*0.5] = 7.14.

5.4.2 Μείωση του συνολικού χρόνου επεξεργασίας των πακέτων.

Οι εξισώσεις 5.1 και 5.2 με τις οποίες υπολογίζεται ο αριθμός των πακέτων που εξετάζονται από το λογισμικό είναι αρκετά απλές και μπορούν να χρησιμοποιηθούν για μια πρόχειρη εκτίμηση της επιτάχυνσης που μας προσφέρει η χρήση των φίλτρων. Σε περίπτωση, όμως, που ενδιαφερόμαστε για ένα πιο ακριβές μοντέλο της απόδοσης των φίλτρων υλικού είναι απαραίτητη μία λεπτομερής ανάλυση.

Αρχικά, συμβολίζουμε με t_p τον χρόνο που απαιτείται αποκλειστικά για την επεξεργασία ενός πακέτου. Στον χρόνο αυτό δεχόμαστε ότι δεν περιλαμβάνεται καθόλου χρόνος για I/O.

Ο χρόνος t_p μπορεί γενικά να εκφραστεί ως το άθροισμα των δύο συνιστωσών t_{header} και $t_{payload}$. Ο χρόνος t_{header} αναπαριστά τον απαραίτητο χρόνο για την κατηγοριοποίηση της επικεφαλίδας του πακέτου και ο χρόνος $t_{payload}$ είναι ο απαιτούμενος χρόνος για τον έλεγχο των δεδομένων του πακέτου. Ο χρόνος t_{header} για ένα πακέτο είναι ανεξάρτητος του μήκους του πακέτου αφού οι επικεφαλίδες είναι σταθερού μεγέθους και οι ζητούμενες πληροφορίες βρίσκονται πάντοτε στις ίδιες θέσεις. Αντίθετα, ο χρόνος $t_{payload}$ εξαρτάται άμεσα από το μήκος του πακέτου.

Κάθε πακέτο που διέρχεται από το σύστημα ανήκει σε μία από τις τρεις ακόλουθες κατηγορίες.

Α) *Πακέτα που δεν ταιριάζουν σε καμία ομάδα κανόνων*. Αποτελούν το (1-h)*100% των διερχομένων πακέτων. Εφόσον τα πακέτα αυτά δεν ταιριάζουν σε καμία ομάδα, δεν γίνεται έλεγχος των δεδομένων τους, επομένως ο χρόνος $t_{payload}$ για αυτά τα πακέτα

θα είναι μηδέν. Από την άλλη πλευρά, για να διαπιστώσουμε ότι κάποιο πακέτο δεν ταιριάζει με καμία ομάδα κανόνων πρέπει να συγκρίνουμε τις παραμέτρους του με όλες τις ομάδες, επομένως ο χρόνος t_{header} θα έχει τη μέγιστη τιμή t_{header_max}.

Β) *Πακέτα που ταιριάζουν σε μια ομάδα κανόνων αλλά δεν επαληθεύουν το περιεχόμενο καποιου από τους κανόνες της ομάδας*. Στην κατηγορία αυτή υπάγεται το $h*(1-p)*100\%$ των διερχομένων πακέτων. Για τα πακέτα αυτά ο χρόνος κατηγοριοποίησης t_{header} θα έχει μία μέση τιμή t_{header_avg}, ενώ ο χρόνος για τον έλεγχο των δεδομένων θα έχει μία μέση τιμή $t_{payload_avg}$.

Γ) *Πακέτα που ταιριάζουν σε μια ομάδα κανόνων και επαληθεύουν το περιεχόμενο κάποιου από τους κανόνες της ομάδας*. Σε αυτή την κατηγορία ανήκει το υπόλοιπο $h*p*100\%$ των πακέτων. Για τα πακέτα αυτά ο χρόνος κατηγοριοποίησης της επικεφαλίδας είναι t_{header_avg}, όπως και στην προηγούμενη κατηγορία. Όσον αφορά τον χρόνο εξέτασης των δεδομένων, υποθέτουμε πως η θετική απαντήση που επιστρέφει ο αλγόριθμος αναζήτησης αυξάνει τον απαιτούμενο χρόνο κατά Δt, με αποτέλεσμα ο χρόνος εξέτασης των δεδομένων να δίνεται από το άθροισμα $t_{payload_avg} + \Delta t$.

Σύμφωνα με τον παραπάνω διαχωρισμό των πακέτων, ο συνολικός χρόνος T που χρειάζεται το λογισμικό για την επεξεργασία μίας ακολουθίας πακέτων χωρίς να χρησιμοποιούνται τα φίλτρα υλικού μπορεί να υπολογιστεί αθροίζοντας τους επιμέρους χρόνους επεξεργασίας για τα πακέτα των τριών κατηγοριών. Το τελικό αποτέλεσμα καταγράφεται στην εξίσωση 5.3.

$$\boxed{T = [(1-h)*\ t_{header_max} + h*(\ t_{header_avg} + t_{payload_avg}\ ) + h*p*\Delta t]\ *\ N}$$

Εξίσωση 5.3: ο συνολικός χρόνος που χρειάζεται το λογισμικό για την επεξεργασία μίας ακολουθίας πακέτων σε περίπτωση που δεν χρησιμοποιούνται φίλτρα υλικού.

Σε περίπτωση που χρησιμοποιείται το φίλτρο κατηγοριοποίησης, η λειτουργία της κατηγοριοποίησης παύει να εκτελείται από το λογισμικό με αποτέλεσμα να εξαφανίζονται οι όροι t_{header_max} και t_{header_avg} από την εξίσωση 5.3. Επομένως, η εξίσωση

για τον χρόνο επεξεργασίας όταν χρησιμοποιείται το φίλτρο κατηγοριοποίησης θα είναι η ακόλουθη.

$$T = h*(t_{payload_avg} + p*\Delta t) * N$$

Εξίσωση 5.4: ο χρόνος που δαπανά το λογισμικό για την επεξεργασία των πακέτων όταν χρησιμοποιείται φίλτρο κατηγοριοποίησης υλοποιημένο σε υλικό.

Το μόνο που απομένει είναι να υπολογίσουμε τον χρόνο επεξεργασίας των πακέτων στην περίπτωση που χρησιμοποιείται ο συνδυασμός των δύο φίλτρων υλικού. Όπως αναφέραμε στην παράγραφο 5.4.1, στο φίλτρο προσεγγιστικής αναγνώρισης συμβολοσειρών εισέρχονται $h*N$ πακέτα, εκ των οποίων τα $p*h*N$ ικανοποιούν την επικεφαλίδα **και** το περιεχόμενο κάποιου κανόνα. Για κάθε ένα από αυτά τα $p*h*N$ πακέτα απαιτείται χρόνος ($t_{payload_avg}+\Delta t$) για τον έλεγχο των δεδομένων.

Τα υπόλοιπα $(1-p)*h*N$ πακέτα που έχουν εισέλθει στο φίλτρο προσεγγιστικής αναγνώρισης δεν περιέχουν ύποπτο περιεχόμενο. Για το f *100% των πακέτων αυτών θα γίνει ψευδής αναγνώρισης από το φίλτρο, γεγονός που σημαίνει ότι $f*(1-p)*h*N$ πακέτα που δεν περιέχουν ύποπτο περιεχόμενο θα προωθηθούν στο λογισμικό για έλεγχο των δεδομένων τους. Για κάθε ένα από τα ψευδώς αναγνωρισμένα πακέτα, το λογισμικό θα χρειαστεί χρόνο $t_{payload_avg}$. Ο όρος Δt δεν εμφανίζεται στην συγκεκριμένη περίπτωση καθώς τα πακέτα αυτά δεν επιστρέφουν θετική απάντηση.

Τέλος, τα $(1-f)*(1-p)*h*N$ πακέτα που δεν περιέχουν ύποπτα δεδομένα αναγνωρίζονται με επιτυχία ως ασφαλή από το φίλτρο προσεγγιστικής αναγνώρισης και δεν προωθούνται στο λογισμικό. Συνεπώς, για τα συγκεκριμένα πακέτα το λογισμικό δεν ξοδεύει καθόλου χρόνο.

Αθροίζοντας τους επιμέρους χρόνους για τις διάφορετικές περιπτώσεις καταλήγουμε στην εξίσωση 5.5.

$$T = [p*(t_{payload_avg} + \Delta t) + (f*(1-p)*t_{payload_avg})] *h*N$$

Εξίσωση 5.5: ο χρόνος που δαπανά το λογισμικό για την επεξεργασία των πακέτων όταν χρησιμοποιείται ο συνδυασμός των δύο φίλτρων υλικού.

5.5 Ανακεφαλαίωση

Στο συγκεκριμένο κεφάλαιο ασχοληθήκαμε με τις σημαντικότερες λεπτομέρειες σχετικά με τη λειτουργία του TUCNIDS και των φίλτρων υλικού. Ένα γενικό σχεδιάγραμμα των υποσυστημάτων του TUCNIDS παρουσιάστηκε στην εικόνα *5α*, ενώ η ανάλυση του σταδίου της αρχικοποίησης και του σταδίου ελέγχου διερχόμενης κίνησης παρουσιάζεται στις υποενότητες 5.1 και 5.2 αντίστοιχα.

Στην υποενότητα 5.3 παρουσιάστηκε η θεωρητική λειτουργία των δύο προτεινόμενων φίλτρων υλικού, δηλαδή του φίλτρου κατηγοριοποίησης επικεφαλίδας και του φίλτρου προσεγγιστικής αναγνώρισης.

Στην τελευταία υποενότητα, την 5.4, κάναμε μία θεωρητική ανάλυση της επίδρασης που θα έχουν τα φίλτρα υλικού στη λειτουργία του συστήματος ανίχνευσης εισβολών. Το αποτέλεσμα της ανάλυσης είναι οι μαθηματικές εξισώσεις 5.1 μέχρι και 5.5, με τις οποίες μπορούμε να εκτιμήσουμε τον αριθμό των πακέτων που θα καταλήγουν τελικά στο λογισμικό για επεξεργασία και καθώς και τον χρόνο που θα ξοδεύει το λογισμικό για την επεξεργασία των πακέτων.

Κεφάλαιο 6.

Πειραματικές μετρήσεις και αποτελέσματα.

Στο κεφάλαιο αυτό γίνεται η αξιολόγηση της υλοποίησης του TUCNIDS καθώς και η σύγκριση του TUCNIDS με το Snort. Επιπλέον, αξιολογούνται τα αποτελέσματα που προκύπτουν από τη συνεργασία υλικού και λογισμικού για την ανίχνευση εισβολών δικτύου.

Προκειμένου να αξιολογήσουμε το σύστημα ανίχνευσης εισβολών δικτύου που υλοποιήσαμε και να το συγκρίνουμε με το υπάρχον σύστημα Snort, είναι απαραίτητο να χρησιμοποιηθεί ένα κοινό σύνολο κανόνων και για τα δύο συστήματα. Επίσης, χρειάζεται μία ακολοθία εισόδων η οποία να περιέχει, σε κάποιο ποσοστό, μερική ή ολική συμφωνία με κάποιους από τους χρησιμοποιούμενους κανόνες. Οι λεπτομέρειες για τους χρησιμοποιούμενους κανόνες, τις εισόδους των συστήματων και, γενικότερα, για το περιβάλλον διεξαγωγής των πειραμάτων παρουσιάζονται στο πρώτο κομμάτι του κεφαλαίου αυτού.

Στη συνέχεια του κεφαλαίου συγκρίνουμε τα δύο συστήματα ανίχνευσης εισβολών δικτύου που είναι υλοποιημένα σε λογισμικό, δηλαδή το TUCNIDS και το Snort. Τα δύο συστήματα συγκρίνονται ως προς την ακρίβειά τους, δηλαδή ως προς τον αριθμό των πακέτο με ύποπτο περιεχόμενο που εντοπίζουν επιτυχώς, αλλά και ως προς την την ταχύτητα τους. Η σύγκριση της ταχύτητας γίνεται με βάση το συνολικό χρόνο (συμπεριλαμβανομένου και το χρόνου I/O) που χρειάζονται τα δύο συστήματα για να εξετάσουν τις ίδιες ακολουθίες πακέτων. Επίσης, εξετάζεται η επίδραση που έχουν οι διαφορετικές είσοδοι στον χρόνο που απαιτείται αποκλειστικά για την εξέταση των πακέτων στο σύστημα TUCNIDS.

Τέλος, παρουσιάζονται τα πειραματικά αποτελέσματα από τη χρήση των φίλτρων υλικού για την υποβοήθηση του TUCNIDS, τόσο στην περίπτωση που μετράμε μόνο τον χρόνο που χρησιμοποιείται αποκλειστικά για την επεξεργασία των πακέτων όσο και στην περίπτωση που μετράμε το συνολικό χρόνο εκτέλεσης (συμπεριλαμβανομένου και του χρόνου πρόσβασης στο δίσκο).

6.1 Οι συνθήκες των πειραμάτων.

Οι συνθήκες υπό τις οποίες εκτελέστηκαν οι δοκιμές και λήφθηκαν οι μετρήσεις περιγράφονται ακολούθως.

- *Χαρακτηριστικά υπολογιστή*:

- Οι μετρήσεις για όλα τα δείγματα έγιναν στον ίδιο υπολογιστή (επεξεργαστής AMD Athlon στα 1,2 GHz, 256 KB μνήμη cache, 256 MB SDRAM με Windows XP. Όλες οι δοκιμές έγιναν υπο συνθήκες πολύ ελαφρού φόρτου, με περισσότερο από το 98% του επεξεργαστή να είναι διαθέσιμο.

- *Οι ρυθμίσεις των συστημάτων ανίχνευσης εισβολών δικτύου*:

- Τόσο στο Snort όσο και στο TUCNIDS χρησιμοποιούνται τα αρχεία κανόνων που συμπεριλαμβάνονται στην έκδοση 2.1.3 του Snort. Όλα τα αρχεία κανόνων έχουν επεξεργαστεί κατάλληλα ώστε σε κάθε κανόνα να κρατάμε μόνο το ένα πεδίο content. Επιπλέον πεδία content καθώς και οι υπόλοιπες επιλογές, όπως η επιλογή "uricontent" ή η "pcre", αφαιρέθηκαν από τους κανόνες. Επίσης αφαιρέθηκαν τελείως οι κανόνες που δεν περιέχουν κανένα πεδίο content.
- Και τα δύο συστήματα ανίχνευσης εισβολών χρησιμοποιούν όλα τα αρχεία κανόνων.
- Το Snort έχει ρυθμιστεί ώστε να μην εκτελεί έλεγχο checksum και έχουν απενεργοποιηθεί όλοι οι προεπεξεργαστές του. Οι λειτουργίες αυτές είναι ιδιαίτερα χρήσιμες αλλά εκτιμούμε ότι οι χρονικές τους απαιτήσεις θα είναι παρόμοιες σε όλες τις αποδοτικές υλοποιήσεις, οπότε δεν επιρρεάζουν την αξιολόγηση και τα συμπεράσματα που ακολουθούν. Επίσης, ορισμένοι προεπεξεργαστές επιτρέπουν στο Snort την αποφυγή εξέτασης ορισμένων πακέτων ανάλογα με τις παραμέτρους των πακέτων (π.χ ο "stream4" προεπεξεργαστής απορρίπτει πακέτα TCP από ροές που δεν έχουν δημιουργηθεί μέσω 3-way handshake).
- Στις περιπτώσεις που ενδιαφερόμαστε να μετρήσουμε την ταχύτητα αναζήτησης του Snort, η λειτουργία του γίνεται σε quiet mode, χωρίς να γίνεται καταγραφή

των πακέτων που επαληθεύουν κανόνες ή των παραγόμενων alerts. Ο λόγος είναι επειδή η καταγραφή των alerts ή των πακέτων δημιουργεί αρχεία εξόδου στο δίσκο, γεγονός που καθυστερεί σημαντικά τη λειτουργία του συστήματος.

- *Οι είσοδοι των συστημάτων*:

- Χρησιμοποιήθηκαν 3 διαφορετικά σύνολα ψευδοτυχαίων δειγμάτων (traces). Κάθε σύνολο δειγμάτων αντιπροσωπεύει και μια διαφορετική τιμή για το μέσο μήκος των πακέτων του δείγματος (300 bytes, 560 bytes και 1000 bytes). Το μήκος κάθε πακέτου που περιέχεται σε κάποιο δείγμα ενός συνόλου κυμαίνεται ομοιόμορφα σε τιμές ± 50 bytes από το μέσο μήκος που αντιστοιχεί στο σύνολο αυτό. Για παράδειγμα, τα πακέτα σε ένα δείγμα του συνόλου των 300 bytes θα έχουν μήκος από 250 μέχρι 350 bytes.
- Κάθε σύνολο δειγμάτων αποτελείται από 15 διαφορετικά δείγματα (traces). Σε κάθε δείγμα περιέχονται 250.000 πακέτα με αναλογία 70% πακέτα TCP και 30% πακέτα UDP. Δε χρησιμοποιήθηκαν πακέτα άλλων πρωτοκόλλων καθώς οι ομάδες κανόνων για τα πρωτόκολλα TCP και UDP αποτελούν το 98% του συνολικού αριθμού ομάδων, όπως φαίνεται και στην *εικόνα 5α*.
- Τα περιεχόμενα κάθε δείγματος περιγράφονται συνοπτικά μέσω δύο πιθανοτήτων: η πρώτη πιθανότητα συμβολίζεται με h και εκφράζει την πιθανότητα με την οποία ένα εισερχόμενο πακέτο ταιριάζει με την επικεφαλίδα κάποιας ομάδας κανόνων. Η δεύτερη πιθανότητα συμβολίζεται με p και αποτελεί τη δεσμευμένη (ως προς την πιθανότητα h) πιθανότητα σύμφωνα με την οποία ένα πακέτο θα περιέχει το content που υποδηλώνει ένας κανόνας **δεδομένου ότι** το πακέτο ταιριάζει με την επικεφαλίδα του κανόνα αυτού. Για παράδειγμα, σε περίπτωση που έχουμε σε ένα δείγμα τις δυο τιμές h και p να είναι 75% και 50% αντίστοιχα, τότε μπορούμε να περιμένουμε ότι το το 75% των συνολικών πακέτων (δηλαδή περίπου 187500 πακέτα) ταιριάζουν με την επικεφαλίδα κάποιων κανόνων. Από αυτά τα 187500 πακέτα που ταιριάζουν με τις επικεφαλίδες των κανόνων, το 50% (δηλαδή περίπου 93750 πακέτα) περιέχουν **και** το content που υποδεικνύει ο αντίστοιχος κανόνας. Οι δύο αυτές παράμετροι περιγράφουν την επικινδυνότητα της διερχόμενης κίνησης.

- Πιο συγκεκριμένα, σε κάθε σύνολο δειγμάτων χρησιμοποιούνται τα εξής 15 ζεύγη των πιθανοτήτων *h* και *p* (σε κάθε ζεύγος η πρώτη τιμή είναι η πιθανότητα *h* και η δεύτερη η *p*): (0%,0%), (25%,0%), (25%,50%), (25%,100%), (50%,0%), (50%,50%), (50%,100%), (75%,0%), (75%,50%), (75%,100%), (100%,0%), (100%,25%) (100%,50%), (100%,75%) και (100%,100%)
- Τα διάφορα δείγματα βρίσκονται αποθηκευμένα σε αρχεία στον σκληρό δίσκο, από όπου και διαβάζονται κατά τη διάρκεια της λειτουργίας των συστημάτων. Επειδή τα αρχεία εισόδου είναι αρκετά μεγάλα (από 73 μέχρι 245 MB το καθένα) και τυχόν κατακερματισμός (fragmentation) του σκληρού δίσκου μπορεί να αλλοιώσει σημαντικά τα αποτελέσματα, πριν την εκτέλεση των μετρήσεων έγινε defragment του δίσκου.
- Το Snort δέχεται σαν είσοδο αρχεία σε μορφή tcpdump, ενώ το TUCNIDS χρησιμοποιεί αρχεία υπό μορφή bytestream. Καθώς θέλουμε τα δείγματα να είναι ακριβώς ίδια για τα δύο συστήματα, χρησιμοποιούμε την εξής διαδικασία: όποτε παράγουμε ένα δείγμα υπό μορφή bytestream για το TUCNIDS, παράγουμε το ίδιο ακριβώς δείγμα και υπό μορφή hexdump (στην οποία κάθε byte συμβολίζεται ως xy, όπου τα x και y είναι δεκαεξαδικές τιμές, από 0 μέχρι F, και ανάμεσα σε κάθε διαδοχικά bytes υπάρχει κενό). Στη συνέχεια χρησιμοποιούμε το υποπρόγραμμα text2pcap του προγράμματος Ethereal [30] για να μετατρέψουμε το δείγμα hexdump σε μορφή tcpdump κατάλληλη για το Snort. Τα αρχεία tcpdump είναι ελαφρά μεγαλύτερα σε μέγεθος από τα αντίστοιχα bytestream αρχεία (5,6% μεγαλύτερο για μέσο μήκος πακέτου 300 bytes, 2,78% για μέσο μήκος 560 bytes και 1,6% για μέσο μήκος 1000 bytes).

- *Οι μετρήσεις*:

- Όλες οι μετρήσεις των χρόνων επαναλήφθηκαν 10 φορές. Οι χρόνοι που αναφέρονται σε κάθε περίπτωση είναι οι μέσοι χρόνοι. Οι διασπορές ήταν αρκετά μικρές (<0.5%) σε κάθε περίπτωση.
- Κάθε φορά που βρίσκαμε μια τιμή με σημαντική απόκλιση από το μέσο όρο την απορρίπταμε και παίρναμε νέα μέτρηση.

6.1.1 Κατανομή του μήκους των πακέτων.

Στο σημείο αυτό αξίζει να αναφερθούμε στην επιλογή που έγινε για την κατανομή του μήκους των πακέτων στα δείγματα που κατασκευάστηκαν. Εκτενείς μελέτες στην κίνηση του Διαδικτύου έχουν δείξει ότι το μήκος των πακέτων ακολουθεί την τριφασική (tri-modal) κατανομή, με τις τρεις φάσεις να είναι περίπου στα 40 bytes, τα 560 bytes και τα 1500 bytes [21]. Επιπλέον, ο μέσος όρος του μήκους είναι κοντά στα 380 bytes και παρουσιάζει τάσεις αύξησης.

Εμείς χρησιμοποιήσαμε μια πολύ απλή κατανομή για τον υπολογισμό του μήκους των πακέτων. Ισχυριζόμαστε ότι ο απαιτούμενος χρόνος για την επεξεργασία ενός δείγματος δεν εξαρτάται από την ακριβής κατανομή του μήκους των πακέτων, αλλά εξαρτάται μόνο από τη μέση τιμή του μήκους. Ο ισχυρισμός αυτός στηρίζεται πάνω σε δύο υποθέσεις.

Υπόθεση 1: Δεν υπάρχει συσχέτιση ανάμεσα στο μήκος του πακέτου και τις πιθανότητες h και p. Δηλαδή, στο ίδιο δείγμα η πιθανότητα h ή η p για ένα πακέτο είναι ίδιες είτε το πακέτο έχει μήκος 50 bytes είτε έχει μήκος 1000 bytes. Επομένως, αν σε ένα δείγμα έχουμε, για παράδειγμα, μέση τιμή μήκους 300 bytes και μια τυχαία κατανομή από 40 μέχρι 1500, τότε το μέσο μήκος των πακέτων που ταιριάζουν σε κάποια ομάδα κανόνων (δηλαδή το $100*h\%$ των συνολικών πακέτων) θα είναι επίσης 300 bytes. Η υπόθεση αυτή είναι απαραίτητη καθώς οι πιθανότητες h και p καθορίζουν το ποσοστό των πακέτων που θα επεξεργαστούν τελικά από το λογισμικό.

Υπόθεση 2: Ο χρόνος για τον έλεγχο των δεδομένων ενός πακέτου είναι ανάλογος με το μήκος του πακέτου. Το γεγονός αυτό είναι προφανές καθώς το κόστος σε όλους τους χρησιμοποιούμενους αλγορίθμους είναι γραμμικά εξαρτώμενο από το μήκους n του ελεγχόμενου πακέτου.

Συνεπώς, αν χρησιμοποιούσαμε μια κατανομή με μεγαλύτερο εύρος και διαφορετική μορφή τότε τα πακέτα με μικρό μήκος θα χρειάζονταν λιγότερο χρόνο επεξεργασίας από τα πακέτα με μήκος κοντά στο μέσο όρο. Αντίστοιχα τα πακέτα με μεγαλύτερο μήκος θα απαιτούσαν περισσότερο χρόνο. Το αποτέλεσμα στην περίπτωση αυτή θα είναι ο χρόνος για τα μικρά και τα μεγάλα πακέτα να εξισορροπείται και, τελικά,

ο συνολικός χρόνος εκτέλεσης για όλο το δείγμα να είναι σχεδόν ίδιος με την περίπτωση όπου όλα τα πακέτα έχουν μήκος ίσο με τον μέσο όρο.

6.2 Σύγκριση της ακρίβειας του TUCNIDS και του Snort.

Προτού ασχοληθούμε με την *ταχύτητα* του TUCNIDS ως προς το Snort, είναι απαραίτητο να συγκρίνουμε την *ακρίβεια* των δύο συστημάτων, δηλαδή των αριθμό των επικίνδυνων πακέτων που εντοπίζει το κάθε σύστημα στα διάφορα κοινά δείγματα. Υπήρξε, όμως, το πρόβλημα ότι το Snort μετράει τον αριθμό των alerts που καταγράφονται (ένα πακέτο μπορεί να δημιουργήσει περισσότερα του ενός alerts), όχι τον αριθμό των πακέτων που βρέθηκαν να έχουν ύποπτο περιεχόμενο. Για το λόγο αυτό φτιάξαμε ένα μικρό πρόγραμμα με το οποίο εξετάζουμε το αρχείο με τα alerts που παράγει το Snort και να υπολογίζουμε τον αριθμό των πακέτων που προκάλεσαν τουλάχιστον ένα alert. Τα αποτελέσματα για τα διάφορα δείγματα φαίνονται στον πίνακα 6.1.

ΠΙΝΑΚΑΣ 6.1

Αριθμός επικίνδυνων πακέτων που εντοπίστηκαν ανά δείγμα και ανά μέσο μήκος. Για το Snort καταγράφεται ο απόλυτος αριθμός πακέτων που βρέθηκαν, ενώ για το TUCNIDS καταγράφεται η σχετική διαφορά ως προς τον αριθμό των πακέτων που βρήκε το Snort. Μια τιμή +5 υποδηλώνει ότι το TUCNIDS βρήκε 5 επιπλέον πακέτα.

	Μέσο Μήκος Πακέτων					
	300 bytes		**560 bytes**		**1000 bytes**	
Παράμετροι Δείγματος (h, p)	*Snort*	*TUCNIDS*	*Snort*	*TUCNIDS*	*Snort*	*TUCNIDS*
(0%, 0%)	0	+0	6	+0	0	+0
(25%, 0%)	2	+0	0	+0	1	+0
(25%, 50%)	30.773	+31	31.363	+37	31.365	+31
(25%, 100%)	62.584	+62	62.631	+64	62.367	+68
(50%, 0%)	0	0	0	+0	2	+0
(50%, 50%)	62.578	+72	62.661	+67	62.477	+61
(50%, 100%)	124.437	+126	124.635	+118	125.070	+138

(75%, 0%)	2	+0	3	+0	3	+0
(75%, 50%)	93.449	+77	93.793	+107	93.641	+100
(75%, 100%)	187.054	+215	187.466	+177	181.278	+195
(100%, 0%)	3	+0	1	+0	3	+0
(100%, 25%)	62.550	+66	62.389	+66	62.666	+55
(100%, 50%)	124.674	+131	124.715	+147	125.306	+128
(100%, 75%)	187.708	+196	187.269	+211	187.492	+176
(100%, 100%)	249.727	+271	249.740	+260	249.731	+264

Παρατηρούμε ότι ο αριθμός των πακέτων που εντοπίζονται από το TUCNIDS είναι σε όλες τις περιπτώσεις ίσος ή μεγαλύτερος από τον αντίστοιχο αριθμό του Snort. Το αποτέλεσμα αυτό είναι σίγουρα ενθαρρυντικό όσον αφορά την ακρίβεια του TUCNIDS, όμως δε σημαίνει απαραίτητα πως το Snort έχει πρόβλημα στην ακρίβειά του. Όπως αναφέρεται στο documentation του Snort σχετικά με το θέμα των συγκρούσεων ανάμεσα σε διαφορετικές ομάδες κανόνων, υπάρχουν τρεις μέθοδοι αντιμετώπισης των πακέτων. Καθώς δεν βρήκαμε κάποια επιλογή που να υποδεικνύει στο Snort τον τρόπο με τον οποίο θα αντιμετωπίζει τις συγκρούσεις, το πιθανότερο είναι πως η μικρή διαφορά που διαπιστώθηκε στον αριθμό των επικίνδυνων πακέτων οφείλεται στην μη εξέταση όλων των πιθανών συγκρούσεων από το Snort.

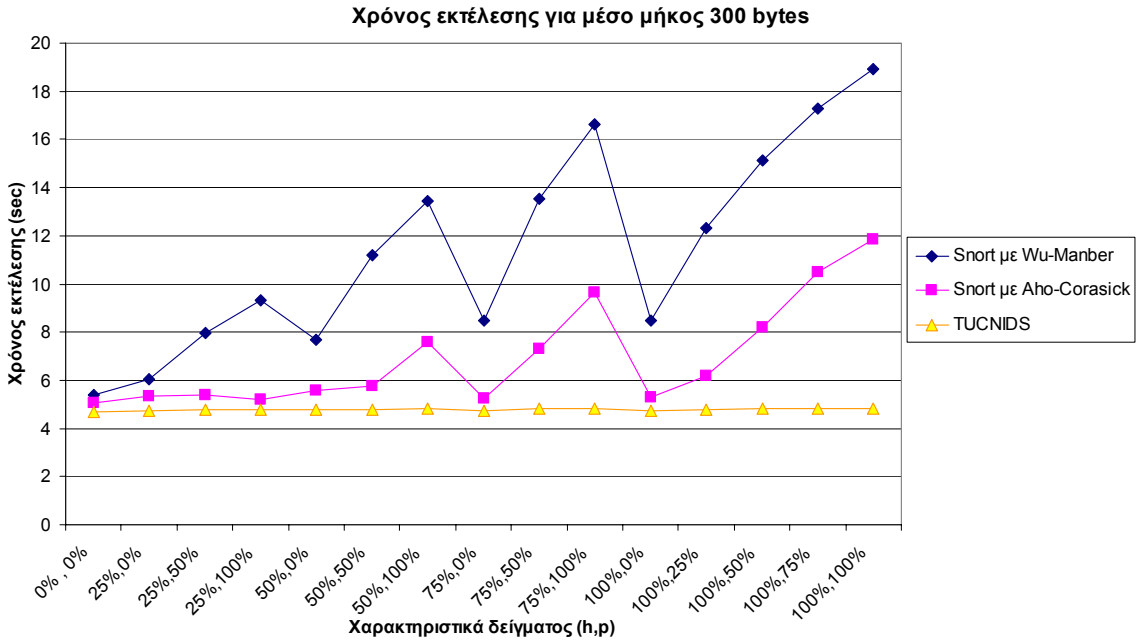
Προκειμένου να εξετάσουμε καλύτερα το θέμα της ακρίβειας των συστημάτων, τροποποιήσαμε το TUCNIDS κατάλληλα ώστε να μην εξετάζει για πιθανές συγκρούσεις αλλά να σταματάει στην πρώτη ομάδα κανόνων με την οποία ταιριάζουν οι παράμετροι κάθε πακέτου. Τα αποτελέσματα που προέκυψαν από την αλλαγή αυτή δεν θα τα παρουσιάσουμε λεπτομερώς, αξίζει απλά να αναφέρουμε πως ο αριθμός των πακέτων που εντοπίστηκαν ήταν σε κάθε περίπτωση λίγο πάνω ή κάτω από τον αντίστοιχο αριθμό πακέτων για το Snort. Υπήρχαν, δηλαδή, περιπτώσεις όπου η διαφορά των πακέτων που εντόπισε το TUCNIDS ως προς τα πακέτα του Snort είναι αρνητική. Ο λόγος για το φαινόμενο αυτό είναι απλά ότι η σύγκριση ενός πακέτου με τις ομάδες κανόνων δεν γίνεται με την ίδια σειρά στα δύο συστήματα. Επομένως, είναι λογικό σε ορισμένες περιπτώσεις να εντοπίζει μόνο το Snort με την πρώτη προσπάθεια την κατάλληλη ομάδα κανόνων για ένα πακέτο, ενώ σε άλλες περιπτώσεις την κατάλληλη ομάδα κανόνων να την εντοπίζει μόνο το TUCNIDS.

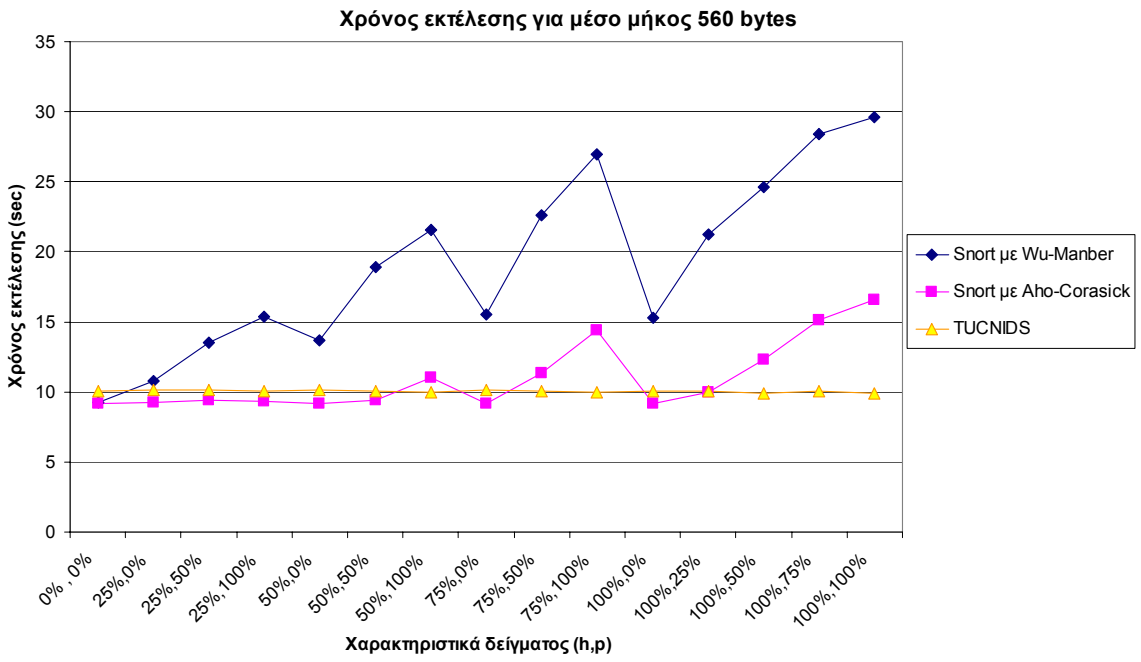
6.3 Σύγκριση της ταχύτητας του TUCNIDS και του Snort.

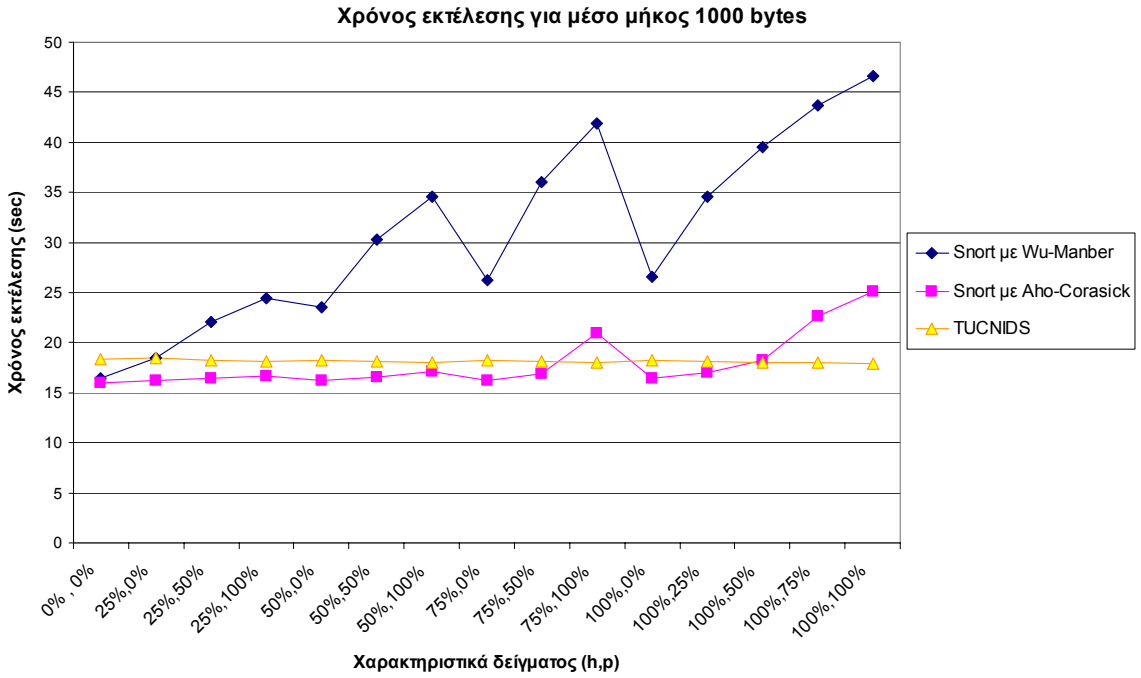
Στο σημείο αυτό θα εξετάσουμε την ταχύτητα των δύο προγραμμάτων συγκρίνοντας τους χρόνους εκτέλεσης για τα διάφορα δείγματα. Οι μετρήσεις για το TUCNIDS αφορούν καθαρά τον χρόνο εκτέλεσης, δηλαδή το χρόνο που χρειάζεται για την πλήρη επεξεργασία κάθε δείγματος, από την αρχή της εκτέλεσης του προγράμματος μέχρι το τέλος του. Στο χρόνο αυτό δε συμπεριλαμβάνεται ο χρόνος προεπεξεργασίας.

Για τις μετρήσεις του Snort χρησιμοποιούμε τους χρόνους που μας επιστρέφει το ίδιο το πρόγραμμα μετά από την ολοκλήρωση της επεξεργασίας ενός δείγματος. Καθώς δεν αναφέρεται ρητά αν στο χρόνο αυτό συμπεριλαμβάνεται ο χρόνος προεπεξεργασίας ή όχι, όταν χρησιμοποιήσαμε σαν είσοδο στο Snort ένα πολύ μικρό δείγμα (με μόνο 3 πακέτα) ο χρόνος εκτέλεσης ήταν μηδενικός, επομένως ούτε στο Snort περιλαμβάνεται ο χρόνος προεπεξεργασίας στις μετρήσεις.

Για το Snort μετρήσαμε τους χρόνους εκτέλεσης και για την περίπτωση που χρησιμοποιείται ο αλγόριθμος Wu-Manber αλλά και όταν χρησιμοποιείται ο Aho-Corasick. Οι μετρήσεις που πήραμε για όλα τα δείγματα φαίνονται στις ακόλουθες γραφικές.







Εικόνα 6α) Από πάνω προς τα κάτω: οι χρόνοι εκτέλεσης για τα διάφορα δείγματα όταν το μέσο μήκος είναι 300, 560 και 1000 bytes αντίστοιχα.

Παρατηρώντας τις τρεις παραπάνω γραφικές, διαπιστώνουμε ότι σχεδόν πάντα το TUCNIDS είναι ταχύτερο από το Snort που χρησιμοποιείται ο αλγόριθμος Wu-Manber, ενώ βρίσκεται αρκετά κοντά στους χρόνους του Snort όταν χρησιμοποιείται ο αλγόριθμος Aho-Corasick. Επίσης, διαπιστώνουμε ότι το Snort με τον αλγόριθμο Aho-Corasick συμπεριφέρεται γενικώς καλύτερα από το TUCNIDS καθώς αυξάνεται το μέσο μήκος των πακέτων: για μέσο μήκος 300 το TUCNIDS είναι ταχύτερο από το Snort με Aho – Corasick, ενώ για μέσο μήκος 560 και 1000 το TUCNIDS είναι σε αρκετές περιπτώσεις λίγο πιο αργό.

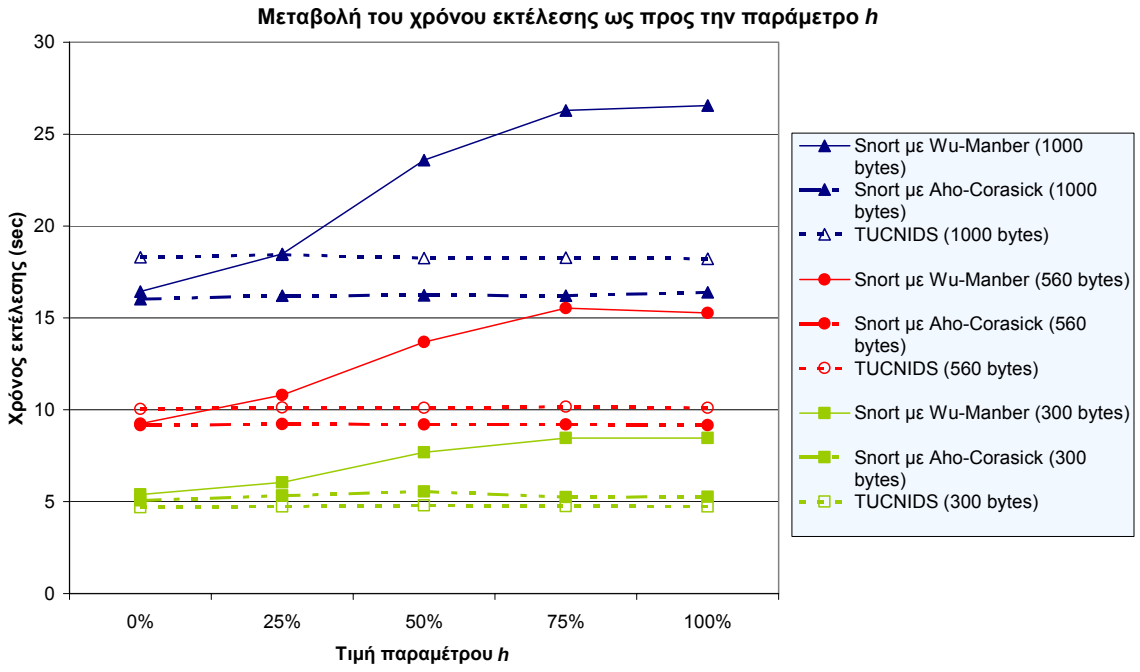
Η άλλη διαπίστωση που προκύπτει από τις γραφικές της εικόνας 6α είναι ότι ο χρόνος εκτέλεσης για το TUCNIDS δεν παρουσιάζει πρακτικά καμία μεταβολή ως προς τις τιμές των πιθανοτήτων *h* και *p*. Το γεγονός αυτό αποτελεί σημαντικό πλεονέκτημα καθώς υποδηλώνει ότι η συμπεριφορά του TUCNIDS είναι σταθερή και απόλυτα προβλέψιμη.

Στο σημείο αυτό αξίζει να αναφερθούμε και στις απαιτήσεις μνήμης που έχουν οι διαφορετικές μέθοδοι. Το Snort όταν χρησιμοποιεί τον αλγόριθμο Wu-Manber χρειάζεται περίπου 35 MB μνήμης, με τον αλγόριθμο Aho-Corasick χρειάζεται περίπου 180 MB μνήμης, ενώ το TUCNIDS χρειάζεται μόλις 16 MB μνήμης. Δηλαδή, με το TUCNIDS πετυχαίνουμε ταχύτητες αρκετά κοντά σε εκείνες του Snort χρησιμοποιώντας υποδεκαπλάσια μνήμη!

Στις γραφικές της εικόνας *6α* καταγράφονται οι χρόνοι για όλα τα δείγματα που δοκιμάσαμε. Από τους χρόνους αυτούς μπορούμε να μελετήσουμε τη συμπεριφορά του TUCNIDS και του Snort καθώς μεταβάλλονται οι παραμέτροι *h* και *p*, γεγονός που θα μας οδηγήσει σε αρκετά ενδιαφέροντα αποτελέσματα.

6.3.1 Μεταβολή του χρόνου εκτέλεσης ως προς την πιθανότητα *h*.

Διατηρώντας το *p* (πιθανότητα να υπάρχει συμβολοσειρά στο πακέτο δεδομένου ότι το πακέτο ταιριάζει σε μία ομάδα) σταθερό στο 0 και παίρνοντας τιμές του *h* (πιθανότητα το πακέτο να ταιριάζει σε μία ομάδα) από 0 μέχρι 100% προκύπτει η ακόλουθη γραφική.



Εικόνα 6β) Μεταβολή του χρόνου εκτέλεσης ως προς την παράμετρο h όταν η παράμετρος p είναι μηδέν.

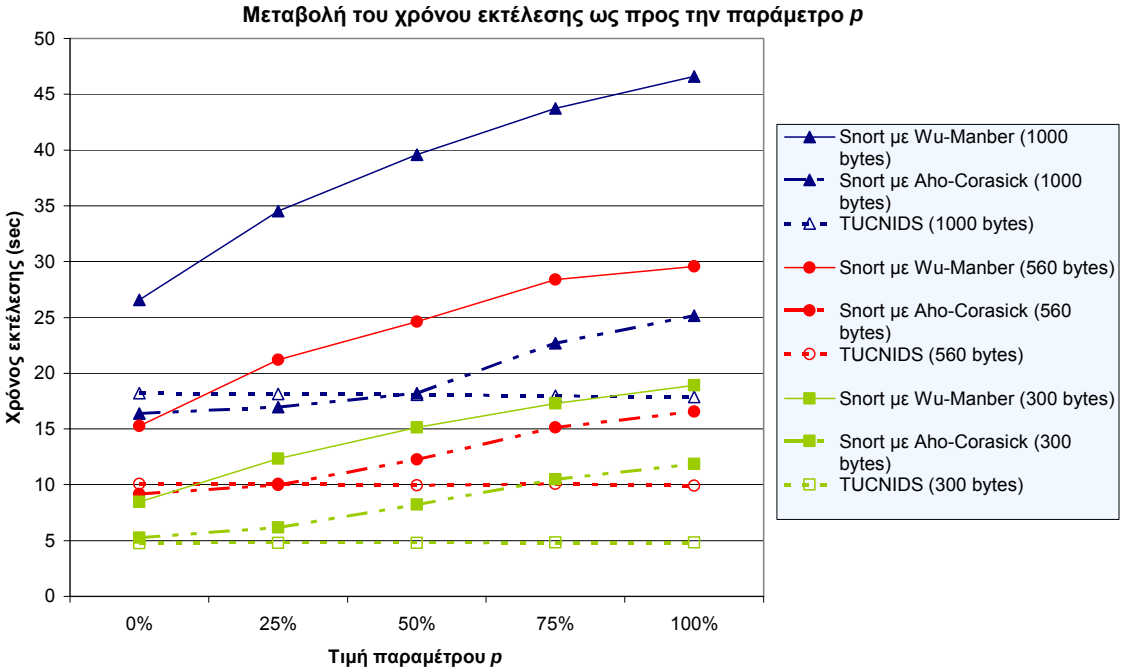
Εξετάζοντας τη γραφική παρατηρούμε μια συμπεριφορά που φαίνεται αρχικά παράδοξη: τόσο ο χρόνος του TUCNIDS όσο και ο χρόνος του Snort με τον αλγόριθμο Aho-Corasick είναι ανεξάρτητοι της παραμέτρου *h*. Το παράδοξο της υπόθεσης είναι πως η παράμετρος *h* εκφράζει την πιθανότητα να ταιριάζει ένα πακέτο με την επικεφαλίδα κάποιου κανόνα, γεγονός που συνεπάγεται ότι το πακέτο θα πρέπει να εξεταστεί με τους αλγόριθμους αναζήτησης για ύποπτα περιεχόμενα. Αντίθετα, εμείς βλέπουμε τους χρόνους του TUCNIDS και του Snort με τον Aho-Corasick να παραμένουν πρακτικά σταθεροί, είτε χρειάζεται να εξετάσουμε τα περιεχόμενα όλων των πακέτων σε ένα δείγμα είτε δε χρειάζεται να εξετάσουμε κανένα.

Η παράδοξη αυτή συμπεριφορά εξηγείται από το γεγονός ότι κάθε δείγμα είναι ένα μεγάλο αρχείο αποθηκευμένο στο σκληρό δίσκο. Καθώς η πρόσβαση στο δίσκο είναι πάρα πολύ «ακριβή» σε θέμα χρόνου, η λογική εξήγηση είναι ότι ο χρόνος εξέτασης του περιεχομένου ενός πακέτου επικαλύπτεται και γίνεται παράλληλα με το διάβασμα επόμενων πακέτων από το δίσκο, τόσο για το TUCNIDS όσο και για το Snort με τον αλγόριθμο Aho-Corasick. Επομένως, ακόμα και αν υπάρχει κάποια περιορισμένη αύξηση στον χρόνο που χρειάζεται για την επεξεργασία των πακέτων στην περίπτωση του TUCNIDS και του Snort με τον αλγόριθμο Aho-Corasick, ο χρόνος πρόσβασης στο

δίσκο είναι τόσο μεγάλος ώστε να υπερκαλύπτει τις αλλαγές στον χρόνο επεξεργασίας των πακέτων.

6.3.2 Μεταβολή του χρόνου εκτέλεσης ως προς την πιθανότητα *p*.

Εκτός από τη μεταβολή του χρόνου ως προς την πιθανότητα h, είναι σημαντικό να εξετάσουμε τη συμπεριφορά των συστημάτων ως προς τις μεταβολές της πιθανότητας *p*. Για το σκοπό αυτό, θεωρούμε την πιθανότητα h ίση με 100% και μεταβάλλουμε την *p* από 0 μέχρι 100%, οπότε προκύπτει η γραφική της *εικόνας 6γ*.



Εικόνα 6γ) Μεταβολή του χρόνου εκτέλεσης ως προς την παράμετρο p όταν η πιθανότητα h είναι ίση με 1 .

Από την παραπάνω γραφική διαπιστώνουμε πως ο χρόνος εκτέλεσης στην περίπτωση του Snort μπορεί να αυξηθεί σημαντικά καθώς αυξάνεται η πιθανότητα *p*. Αυτό σημαίνει πως το Snort ξοδεύει σημαντικό χρόνο όχι μόνο στους αλγόριθμους αναζήτησης συμβολοσειρών αλλά και στην επεξεργασία των αποτελεσμάτων (δηλαδή

των κανόνων που επαληθεύονται) που επιστρέφουν οι αλγόριθμοι αυτοί. Συγκεκριμένα, παρατηρούμε πως ο χρόνος που δαπανάται για τον έλεγχο των πακέτων και την επεξεργασία των αποτελεσμάτων, δηλαδή ο χρόνος επεξεργασίας των πακέτων, είναι αρκετά μεγάλος ώστε να κυριαρχεί επί του χρόνου πρόσβασης στο δίσκο.

Η τελευταία παρατήρηση βασίζεται πάνω σε δύο γεγονότα. Το πρώτο είναι πως τα αρχεία εισόδου έχουν πρακτικά σταθερό μέγεθος για όλα τα δείγματα σε κάθε μία από τις τρεις τιμές που χρησιμοποιήσαμε για το μέσο μήκος πακέτων. Αυτό σημαίνει πως ο απαιτούμενος χρόνος για την πρόσβαση στο δίσκο είναι πρακτικά σταθερός σε κάθε δείγμα ενός συγκεκριμένου συνόλου.

Το δεύτερο γεγονός είναι πως η επεξεργασία ενός πακέτου γίνεται παράλληλα με το διάβασμα από τον δίσκο επόμενων πακέτων. Στο συμπέρασμα αυτό οδηγηθήκαμε από την παρατήρηση της συμπεριφοράς του TUCNIDS και του Snort με τον αλγόριθμο Aho-Corasick στις μεταβολές της πιθανότητας h.

Συνεπώς, το φαινόμενο της αύξησης του χρόνου εκτέλεσης για το Snort καθώς αυξάνεται η πιθανότητα p μας οδηγεί στο συμπέρασμα ότι ο χρόνος που απαιτείται για την επεξεργασία των πακέτων είναι πλέον μεγαλύτερος από τον χρόνο πρόσβασης στο δίσκο και καταλαμβάνει το σημαντικότερο ποσοστό του χρόνο εκτέλεσης.

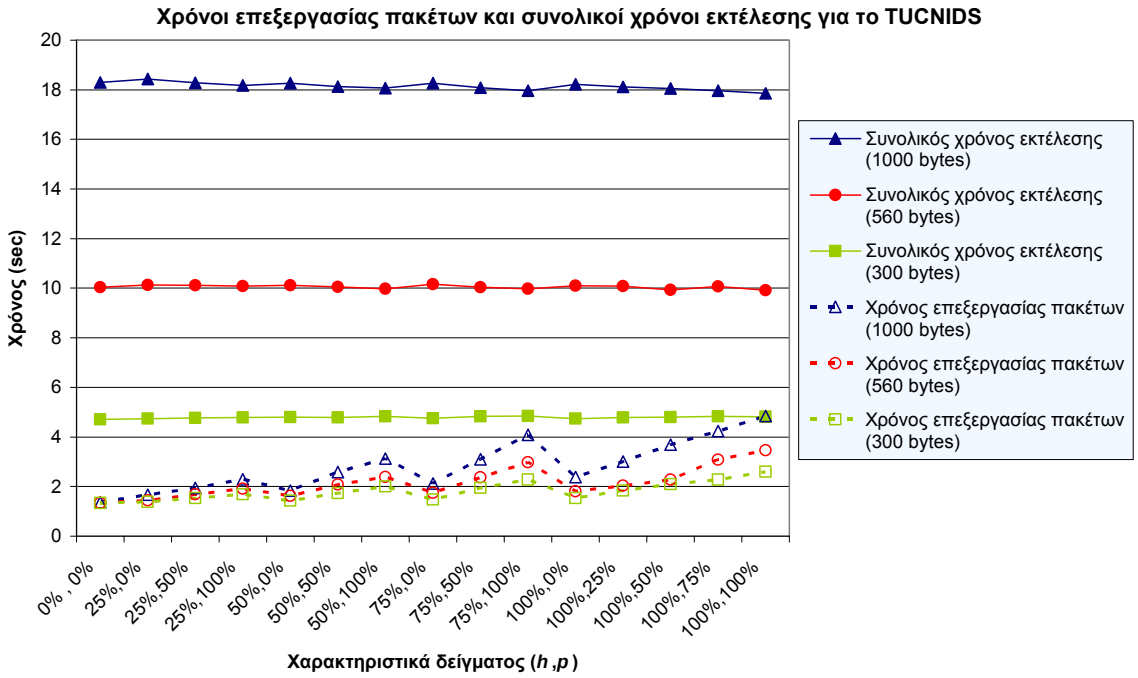
Αντίθετα, στο TUCNIDS παρατηρούμε μια σταθερή συμπεριφορά, ανεξάρτητη από την τιμή της παραμέτρου p. Καθώς στο TUCNIDS δεν γίνεται κάποια χρονοβόρα επεξεργασία των αποτελεσμάτων που επιστρέφουν οι αλγόριθμοι αναζήτησης, είναι αναμενόμενο ο χρόνος να μεταβάλεται σε μικρό βαθμό με την αύξηση της παραμέτρου p. Όπως συνέβη και με την παράμετρο h, οι προσβάσεις στον δίσκο είναι τόσο αργές ώστε να υπερκαλύπτουν τις πιθανές μεταβολές στον χρόνο επεξεργασίας των πακέτων.

6.3.3 Μετρήσεις του χρόνου επεξεργασίας πακέτων για το TUCNIDS.

Προκειμένου να είναι ακριβέστερη η σύγκριση, θα έπρεπε να μπορούμε να συγκρίνουμε το χρόνο *επεξεργασίας* του κάθε συστήματος, δηλαδή τον χρόνο που ξοδεύει κάθε σύστημα από τη στιγμή έχει διαβαστεί ένα πακέτο μέχρι να ολοκληρωθεί η

εξέταση του. Δυστυχώς, το Snort μπορεί να επιστρέφει μόνο το συνολικό χρόνο εκτέλεσης. Μπορούμε, όμως, να τροποποιήσουμε κατάλληλα το TUCNIDS ώστε να διαβάζει μεγάλα μπλοκ από τα αρχεία εισόδου και να καταγράφουμε με ακρίβεια τον χρόνο που χρειάζεται αποκλειστικά για την επεξεργασία των μπλοκ αυτών.

Μετρώντας το χρόνο επεξεργασίας του TUCNIDS για τα διάφορα δείγματα προκύπτει η ακόλουθη γραφική παράσταση.



Εικόνα 6δ) Οι χρόνοι επεξεργασίας και οι συνολικοί χρόνοι εκτέλεσης του TUCNIDS για τα διάφορα δείγματα.

Παρατηρούμε μία απόλυτα λογική και αναμενόμενη συμπεριφορά. Ο χρόνος επεξεργασίας αυξάνεται ανάλογα με την πιθανότητα *p*, την πιθανότητα *h* αλλά και με το μέσο μήκος των πακέτων. Επίσης, διαπιστώνουμε ότι η γενική μορφή της καμπύλης του χρόνου επεξεργασίας μοιάζει αρκετά με την καμπύλη του χρόνου εκτέλεσης για το Snort όταν χρησιμοποιείται ο αλγόριθμος Wu-Manber, όπως αυτή φαίνεται στις γραφικές της εικόνας *6α*. Η συγκεκριμένη διαπίστωση ενισχύει την υπόθεσή μας πως για το Snort με τον αλγόριθμο Wu-Manber ο χρόνος επεξεργασίας ενός πακέτου είναι τόσο μεγάλος

ώστε να μην είναι δυνατή η επικάλυψή του από τον χρόνο που απαιτείται για το διάβασμα επόμενων πακέτων από το δίσκο.

Μία άλλη παρατήρηση από την εικόνα *6δ* είναι πως ο χρόνος επεξεργασίας των πακέτων είναι σε όλες τις περιπτώσεις κατά πολύ μικρότερος του συνολικού χρόνου εκτέλεσης. Η διαπίστωση αυτή ενισχύει την υπόθεσή μας ότι στο συνολικό χρόνο εκτέλεσης που μετράμε για το TUCNIDS, ο χρόνος πρόσβασης στο δίσκο είναι αυτός που κυριαρχεί και επικαλύπτει τον χρόνο επεξεργασίας των πακέτων.

Ιδιαίτερο ενδιαφέρον παρουσιάζει το δείγμα (0%,0%) καθώς στην περίπτωση αυτή οι χρόνοι επεξεργασίας και για τις τρεις τιμές του μέσου μήκους είναι πρακτικά ίσοι. Δεδομένου ότι στο δείγμα αυτό πρακτικά κανένα πακέτο δεν αντιστοιχεί σε κάποια ομάδα κανόνων (η πιθανότητα h είναι 0), ουσιαστικά δεν γίνεται καμία εξέταση σε περιεχόμενα πακέτου. Επιπλέον, το γεγονός ότι κανένα πακέτο δεν αντιστοιχεί σε κάποια ομάδα κανόνων σημαίνει πως οι παράμετροι του κάθε πακέτου έχουν συγκριθεί με όλες τις ομάδες κανόνων (στην περίπτωση που ένα πακέτο ταιριάζει σε κάποια ομάδα κανόνων, η διαδικασία της κατηγοριοποίησης σταματάει στη συγκεκριμένη ομάδα κανόνων χωρίς να εξεταστούν οι επόμενες).

Από τα γεγονότα αυτά γίνεται εμφανές ότι ο χρόνος του δείγματος (0%,0%) ισούται αποκλειστικά και μόνο με τον μέγιστο δυνατό χρόνο κατηγοριοποίησης των πακέτων, δηλαδή με το χρόνο που χρειάζεται για να εξετάσουμε τις επικεφαλίδες κάθε πακέτου με όλες τις ομάδες κανόνων. Αφού οι επικεφαλίδες των πακέτων δεν επιρεάζονται από το μήκος των πακέτων, είναι λογικό ο χρόνος κατηγοριοποίησης των πακέτων να είναι ανεξάρτητος από το μέσο μήκος τους.

Συγκρίνοντας το χρόνο του δείγματος (0%,0%) με τους χρόνους των τριών δειγμάτων (100%,100%), δηλαδή με τον απαιτούμενο χρόνο στη χειρότερη περίπτωση, διαπιστώνουμε πως ο έλεγχος των επικεφαλίδων των διερχομένων πακέτων καταλαμβάνει σημαντικό κομμάτι του συνολικού χρόνου *επεξεργασίας*. Συγκεκριμένα, για μέσο μήκος πακέτων ίσο με 300 bytes ο χρόνος για τον έλεγχο των επικεφαλίδων αποτελεί τουλάχιστον το 58.7% του συνολικού χρόνου επεξεργασίας, για μέσο μήκος 560 bytes καταλαμβάνει τουλάχιστον το 38.4% και, τέλος, για μέσο μήκος 1000 bytes ο έλεγχος επικεφαλίδων αποτελεί το 27.4% του συνολικού χρόνου επεξεργασίας.

6.3.4 Μέγιστοι ρυθμοί επεξεργασίας.

Χρησιμοποιώντας τους χρόνους που έχουμε μετρήσεις, μπορούμε να εκτιμήσουμε το μέγιστο ρυθμό επεξεργασίας που μας παρέχει κάθε ένα από τα συστήματα ανίχνευσης.

Καθώς δεν γνωρίζουμε τις τιμές των πιθανοτήτων h και p για την «τυπική» κίνηση δικτύου (οι πιθανότητες αυτές εξαρτώνται σε μεγάλο βαθμό από το σύνολο των κανόνων που χρησιμοποιούνται), μπορούμε να υπολογίσουμε τους ρυθμούς επεξεργασίας για το «καλύτερο» και την «χειρότερο» είδος κίνησης. Η «καλύτερη» περίπτωση συνεπάγεται ότι το λογισμικό δεν εξετάζει τα περιεχόμενα κανενός πακετου, γεγονός που συμβαίνει μόνο όταν και οι δύο πιθανότητες είναι 0. Αντίθετα, η «χειρότερη» περίπτωση προκύπτει όταν και οι δύο πιθανότητες είναι 100%, οπότε το λογισμικό εξετάζει όλα τα περιεχόμενα και όλοι οι έλεγχοι μας επιστρέφουν θετικό αποτέλεσμα.

Προκειμένου να υπολογίσουμε τον μέγιστο ρυθμό επεξεργασίας σε κάθε περίπτωση, διαιρούμε τον αριθμό των bits που επεξεργάστηκαν με το συνολικό χρόνο που χρειάστηκε για την επεξεργασίας τους.

Για το Snort μετρήσαμε τους μέγιστους ρυθμούς επεξεργασίας όταν χρησιμοποιείται ο αλγόριθμος Wu-Manber και όταν χρησιμοποιείται ο Aho-Corasick.

Στην περίπτωση του TUCNIDS υπολογίσαμε δύο μέγιστους ρυθμους επεξεργασίας. Ο πρώτος ρυθμός που υπολογίσαμε αφορά την περίπτωση όπου τα πακέτα διαβάζονται από το δίσκο (για τον υπολογισμό του ρυθμού αυτού χρησιμοποιούμε τους *χρόνους εκτέλεσης* του TUCNIDS). Για τον δεύτερο ρυθμό υποθέτουμε ότι δε γίνεται καθόλου πρόσβαση στο δίσκο αλλά τα πακέτα προς επεξεργασία βρίσκονται στη μνήμη (ο ρυθμός αυτός υπολογίζεται χρησιμοποιώντας τους *χρόνους επεξεργασίας πακέτων*).

Τα αποτελέσματα που βρήκαμε για τους ρυθμούς επεξεργασίας καταγράφονται στον πίνακα 6.2. Οι ρυθμοί που βρέθηκαν είναι σε Megabits ανά δευτερόλεπτο (Mbps).

ΠΙΝΑΚΑΣ 6.2

Μέγιστοι ρυθμοί επεξεργασίας (σε Mbps) για τα συστήματα ανίχνευσης εισβολών. Με έντονα γράμματα τονίζονται οι βέλτιστοι ρυθμοί στις περιπτώσεις που έχουμε I/O.

	Χειρότερη περίπτωση (p=h=1)			**Καλύτερη περίπτωση** (p=h=0)		
	300 bytes	*560 bytes*	*1000 bytes*	*300 bytes*	*560 bytes*	*1000 bytes*
Snort με Wu-Manber (με I/O από δίσκο)	31,7	37,9	42,9	111,2	121,2	121,7
Snort με Aho-Corasick (με I/O από δίσκο)	50,5	67,7	79,51	118,6	**122,3**	**124,7**
TUCNIDS (με I/O από δίσκο)	**124,5**	**112,9**	**111,6**	**127,6**	111,5	109,3
TUCNIDS (χωρίς I/O από δίσκο)	230,8	323,7	412,3	447,7	842,1	1.470,6

Η πρώτη διαπίστωση που προκύπτει από τους μέγιστους ρυθμούς επεξεργασίας είναι ότι το TUCNIDS είναι κατά πολύ γρηγορότερο του Snort για την χειρότερη περίπτωση εισόδου (2,46 φορές πιο γρήγορο για μέσο μήκος 300 bytes, 1,66 φορές για 560 bytes και 1,4 φορές για 1000 bytes). Το γεγονός αυτό είναι ιδιαίτερα σημαντικό καθώς σημαίνει πως το TUCNIDS είναι πιο ανθεκτικό σε επιθέσεις στέρησης εξυπηρέτησης (DoS attacks).

Όσον αφορά την καλύτερη περίπτωση εισόδου, διαπιστώνουμε ότι οι ρυθμοί του TUCNIDS είναι αρκετά κοντά με εκείνους του Snort που χρησιμοποιεί τον αλγόριθμο Aho-Corasick, έστω και αν το TUCNIDS είναι λίγο πιο αργό σε ορισμένες περιπτώσεις (η μεγαλύτερη διαφορά διαπιστώνεται για μέσο μήκος πακέτου 1000 bytes, στην οποία περίπτωση το Snort είναι γρηγορότερο κατά 1,14 φορές).

Το γεγονός ότι το TUCNIDS είναι ελαφρά πιο αργό από το Snort σε ορισμένες περιπτώσεις δε μας πτοεί καθώς δεν παύει να είναι μία πολύ καλή υλοποίηση ενός συστήματος ανίχνευσης εισβολών δικτύου. Από τη στιγμή που η απόδοση του TUCNIDS είναι αρκετά καλή, έχει νόημα να εξετάσουμε την επίδραση που θα έχουν τα φίλτρα υλικού στην απόδοση του TUCNIDS και, κατ'επέκταση, στην απόδοση οποιουδήποτε καλού συστήματος ανίχνευσης εισβολών.

6.4 Χρήση φίλτρου κατηγοριοποίησης επικεφαλίδων.

Το ενδιαφέρον μας εστιάζεται στην ιδανική λειτουργία ενός φίλτρου κατηγοριοποίησης επικεφαλίδων υλοποιημένου σε υλικό, όχι στη λειτουργία κάποιας συγκεκριμένης υλοποίησης. Για το λόγο αυτό θεωρούμε πως το φίλτρο υλικού είναι αρκετά γρήγορο ώστε να μπορεί να επεξεργάζεται τα εισερχόμενα πακέτα με ρυθμό μεγαλύτερο από ότι αυτά φτάνουν στο σύστημα. Επομένως, θεωρούμε ότι το φίλτρο δεν εισάγει καθόλου καθυστέρηση στο σύστημα, ενώ η πληροφορία από το φιλτράρισμα γίνεται διαθέσιμη στο λογισμικό ταυτόχρονα με την άφιξη του πακέτου.

Η προσομοίωση του φίλτρου αυτού είναι αρκετά απλή. Αρχικά κάνουμε μία προεπεξεργασία ενός δείγματος, στην οποία διαβάζουμε ένα-ένα τα περιεχόμενα πακέτα του δείγματος. Για κάθε πακέτο που διαβάζουμε χρησιμοποιούμε την ίδια μέθοδο κατηγοριοποίησης που χρησιμοποιούμε όταν κάνουμε έλεγχο διερχόμενης κίνησης, με σκοπό να βρούμε τον αύξων αριθμό της ομάδας κανόνων που ταιριάζει στο συγκεκριμένο πακέτο. Σε περίπτωση που βρεθεί κάποια ομάδα να ταιριάζει στο πακέτο, τότε καταγράφουμε σε ένα αρχείο τον αύξων αριθμό της ομάδας, ενώ αντιγράφουμε το πακέτο που διαβάστηκε σε ένα δεύτερο αρχείο εξόδου (το οποίο θα χρησιμοποιηθεί αργότερα για τη μέτρηση του χρόνου εκτέλεσης). Εάν δε βρεθεί καμία ομάδα κανόνων να ταιριάζει στο πακέτο, τότε δεν υπάρχει λόγος να σταλεί το πακέτο στο λογισμικό και απλά το πετάμε.

Η παραπάνω προεπεξεργασία εκτελείται μια φορά για κάθε δείγμα, ενώ δεν μας ενδιαφέρει ο χρόνος για την ολοκλήρωσή της καθώς υποθέτουμε ότι το υλικό είναι αρκετά γρήγορο ώστε να επεξεργάζεται κάθε πακέτο πριν φτάσει το επόμενο και να τα προωθεί μέσω pipeline στο λογισμικό.

Μετά από την προεπεξεργασία, χρησιμοποιούμε σαν είσοδο στη φάση του έλεγχου της εισερχόμενης κίνησης τα δυο αρχεία που παράχθηκαν προηγουμένως, δηλαδή το φιλτραρισμένο δείγμα (το οποίο περιέχει μικρότερο ή ίσο αριθμό πακέτων με το αρχικό δείγμα) και το αρχείο με τους αριθμούς των ομάδων. Για κάθε πακέτο που διαβάζουμε από το φιλτραρισμένο δείγμα αρχικά εξετάζουμε από την επικεφαλίδα του τα πεδία του πρωτοκόλλου και του μήκους. Στη συνέχεια διαβάζουμε από το δεύτερο αρχείο τον αριθμό της ομάδας κανόνων που ταιριάζει στο συγκεκριμένο πακέτο,

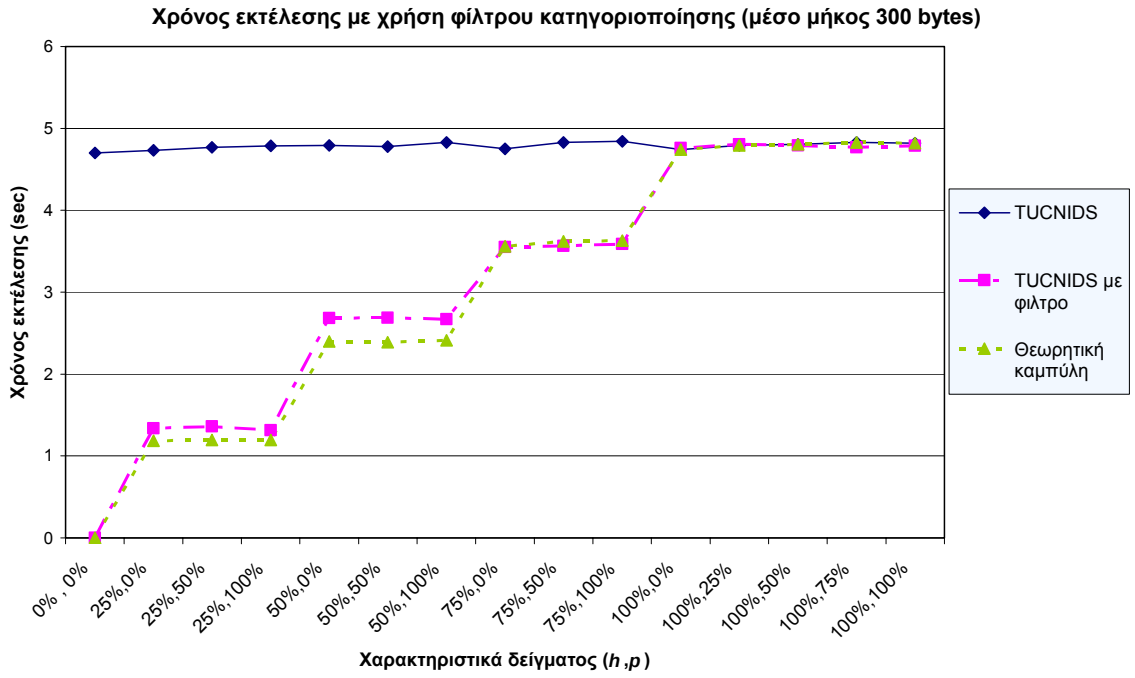
πηγαίνουμε στη συγκεκριμένη ομάδα του αντίστοιχού πρωτοκόλλου και εκτελούμε την αναζήτηση των συμβολοσειρών. Σε περίπτωση που δεν βρούμε καμία συμβολοσειρά εξετάζουμε για πιθανές συγκρούσεις με τον ίδιο τρόπο που χρησιμοποιούμε όταν έχουμε μόνο το λογισμικό.

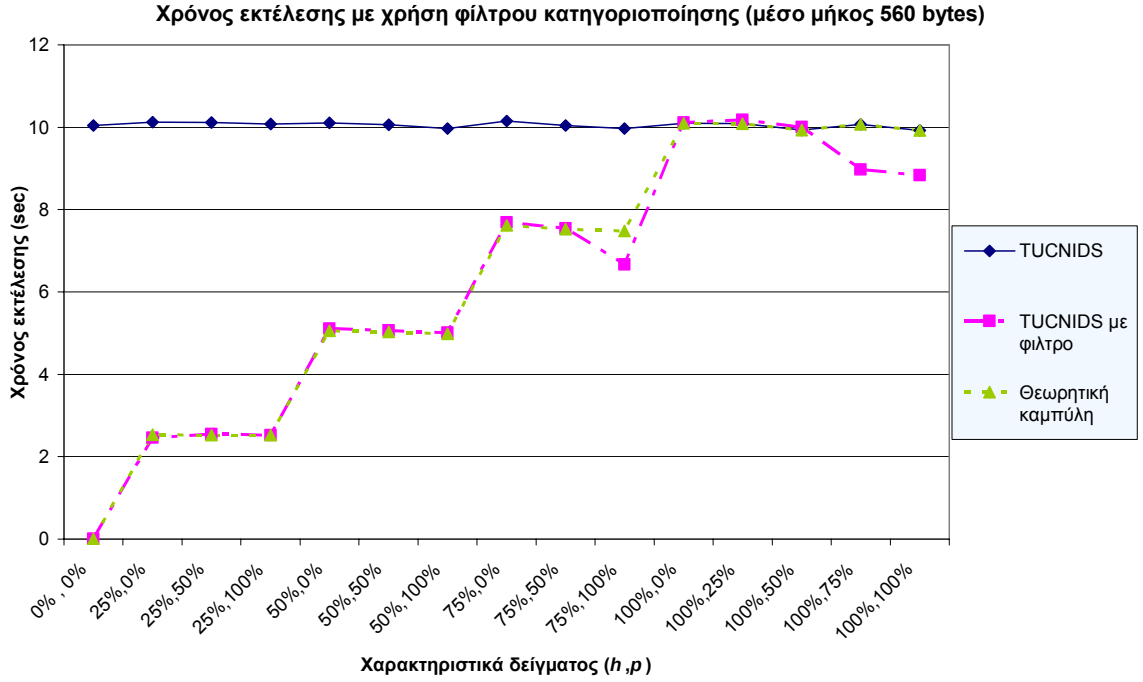
6.4.1 Επίδραση του φίλτρου κατηγοριοποίησης στον χρόνο εκτέλεσης.

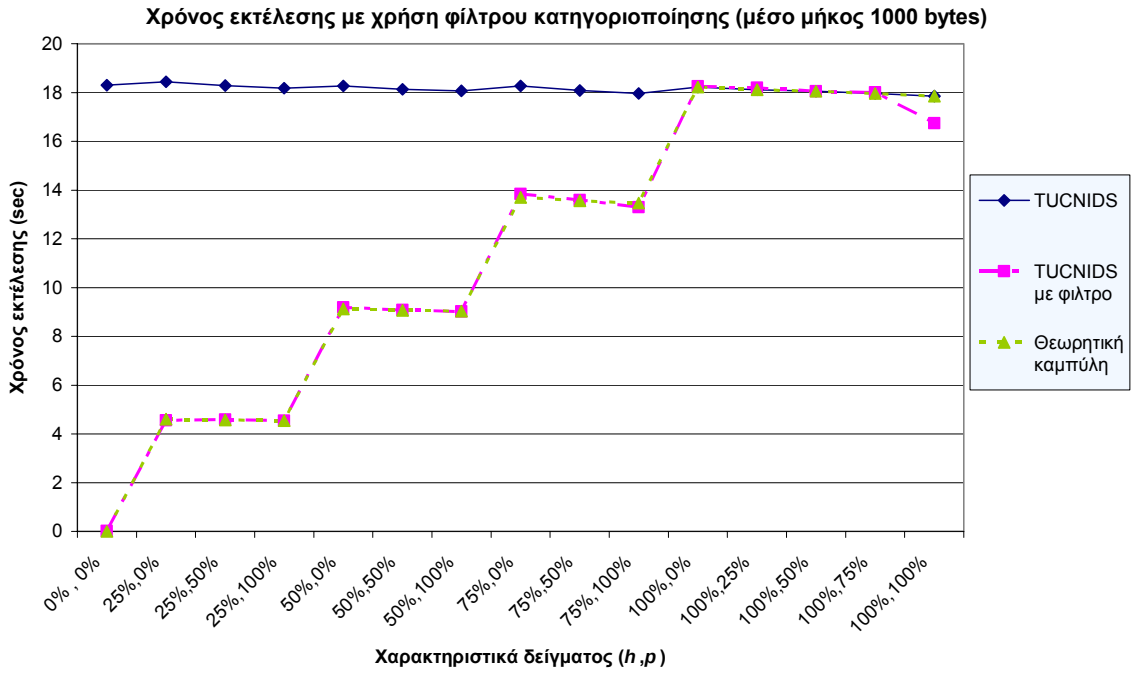
Στις ακόλουθες γραφικές παραστάσεις παρουσιάζεται ο χρόνος εκτέλεσης όταν χρησιμοποιείται μόνο το TUCNIDS καθώς και ο χρόνος που προκύπτει αν χρησιμοποιούμε και το φίλτρο κατηγοριοποίησης επικεφαλίδων.

Σε κάθε γραφική παράσταση συμπεριλαμβάνεται και η θεωρητική καμπύλη $h*t$, όπου t ο χρόνος που χρειάζεται το TUCNIDS χωρίς το φίλτρο υλικού για την επεξεργασία του αντίστοιχου δείγματος. Ο τύπος για την θεωρητική καμπύλη προέκυψε από την εξίσωση 5.1, με την οποία εκφράζουμε τη μείωση των πακέτων που προωθούνται στο λογισμικό.

Βασιστήκαμε στην εξίσωση 5.1 καθώς υποθέτουμε ότι σε κάθε πακέτο αντισοιχεί περίπου ίδιο μερίδιο του χρόνου εκτέλεσης και ότι ο χρόνος εκτέλεσης που προκύπτει με τη χρήση του φίλτρου κατηγοριοποίησης είναι απολύτως ανάλογος με τον αριθμό των πακέτων που τελικά προωθούνται στο λογισμικό. Οι συγκεκριμένες υπόθεσεις οφείλονται στη διαπίστωσή μας ότι στον χρόνο εκτέλεσης κυριαρχεί ο χρόνος που απαιτείται για την πρόσβαση στο δίσκο. Καθώς ο χρόνος πρόσβασης στο δίσκο εξαρτάται από το μήκος του πακέτου και τα πακέτα έχουν περιορισμένη διακύμανση στο μήκος τους, σε κάθε πακέτο αντιστοιχεί περίπου ίσο μερίδιο του συνολικού χρόνου εκτέλεσης.


Χρόνος εκτέλεσης με χρήση φίλτρου κατηγοριοποίησης (μέσο μήκος 300 bytes)
Χρόνος εκτέλεσης (sec)
0
1
2
3
4
5
6
0% , 0%
25%,0%
25%,50%
25%,100%
50%,0%
50%,50%
50%,100%
75%,0%
75%,50%
75%,100%
100%,0%
100%,25%
100%,50%
100%,75%
100%,100%
Χαρακτηριστικά δείγματος (h ,p)
TUCNIDS
TUCNIDS με φιλτρο
Θεωρητική καμπύλη


Χρόνος εκτέλεσης με χρήση φίλτρου κατηγοριοποίησης (μέσο μήκος 560 bytes)
Χρόνος εκτέλεσης (sec)
0
2
4
6
8
10
12
0% , 0%
25%,0%
25%,50%
25%,100%
50%,0%
50%,50%
50%,100%
75%,0%
75%,50%
75%,100%
100%,0%
100%,25%
100%,50%
100%,75%
100%,100%
Χαρακτηριστικά δείγματος (h ,p)
TUCNIDS
TUCNIDS με φιλτρο
Θεωρητική καμπύλη



Εικόνα 6ε) Από πάνω προς τα κάτω: Ο χρόνος εκτέλεσης του TUCNIDS κανονικά και με τη χρήση φίλτρου κατηγοριοποίησης για μέσο μήκος πακέτου 300, 560 και 1000 bytes.

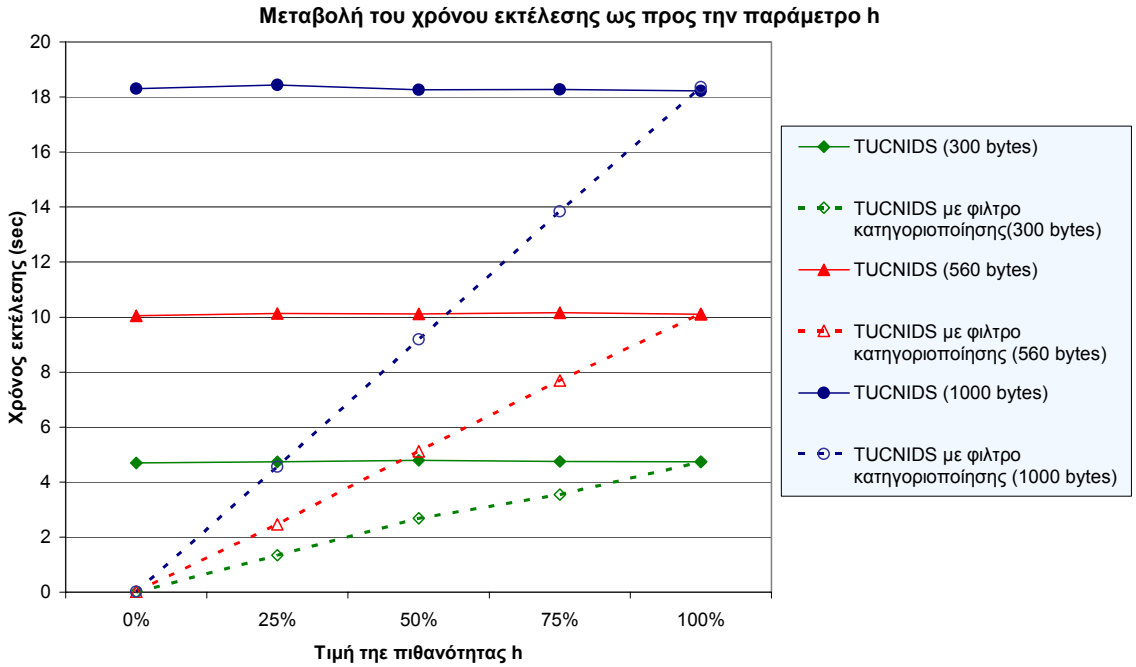
Από τις γραφικές της εικόνας *6ε* διαπιστώνουμε πως η σύμπτωση των πραγματικών μετρήσεων και της θεωρητικής ανάλυσης είναι εντυπωσιακή. Δηλαδή παρατηρούμε ότι το φίλτρο κατηγοριοποίησης μειώνει το χρόνο εκτέλεσης αντιστρόφως ανάλογα με την τιμή της πιθανότητας h με την οποία τα πακέτα ταιριάζουν με κάποια ομάδα κανόνων. Ο βαθμός κατά τον οποίο συμπίπτουν οι θεωρητικές και οι πειραματικές τιμές επιβεβαιώνει τις υποθέσεις που κάναμε στην αρχή της συγκεκριμένης υποενότητας σχετικά με τη μορφή της θεωρητικής καμπύλης.

Οι αισθητές διαφορές που παρατηρούνται σε ορισμένα δείγματα οφείλονται, κατά πάσα πιθανότητα, σε διακυμάνσεις στη λειτουργία του δίσκου.

6.4.2 Μεταβολή του χρόνου εκτέλεσης ως προς την πιθανότητα *h*.

Προκειμένου να έχουμε μια πιο ξεκάθαρη εικόνα της συμπεριφοράς του φίλτρου κατηγοριοποίησης ως προς την πιθανότητα h, κρατάμε σταθερή την τιμή της p, έστω στο

0, και παίρνουμε διάφορες τιμές για την πιθανότητα *h*. Η νέα γραφική που προκύπτει φαίνεται στην παρακάτω εικόνα.



Εικόνα 6στ) Η μεταβολή του χρόνου εκτέλεσης ως προς την πιθανότητα h όταν η p είναι 0.

Είναι προφανές ότι τα οφέλη που παίρνουμε από το φίλτρο κατηγοριοποίησης είναι αντιστρόφως ανάλογα με την πιθανότητα *h*, όπως ακριβώς υπέδειξε η θεωρητική ανάλυση. Το γεγονός αυτό είναι απόλυτα λογικό, καθώς με τη χρήση του φίλτρου κατηγοριοποίησης το λογισμικό χρειάζεται να εξετάσει μόνο το 100**h*%, όπου *h* ανάμεσα στο 0 και το 1, των εισερχόμενων πακέτων. Δεδομένου ότι ο χρόνος εκτέλεσης του λογισμικού είναι άμεση συνάρτηση του αριθμού των πακέτων που εξετάζονται, ο χρόνος εκτέλεσης από t που είναι χωρίς το φίλτρο κατηγοριοποίησης γίνεται t'= *h**t, δηλαδή έχουμε επιτάχυνση του συστήματος κατά 1/*h*.

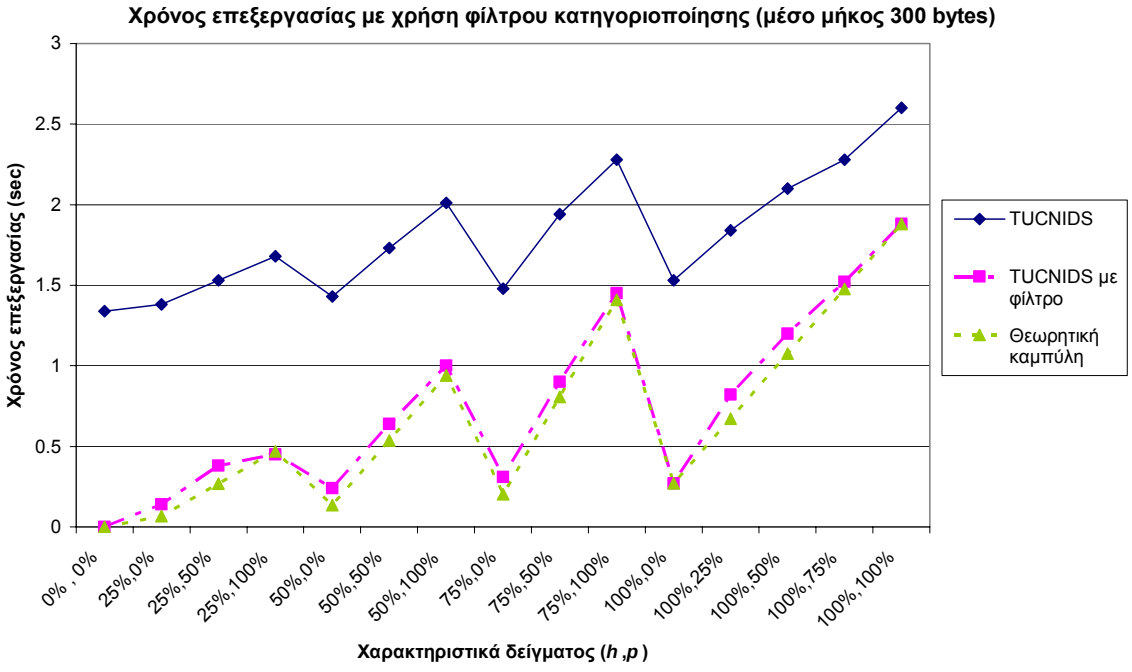
6.4.3 Επίδραση φίλτρου κατηγοριοποίησης στον χρόνο επεξεργασίας πακέτων.

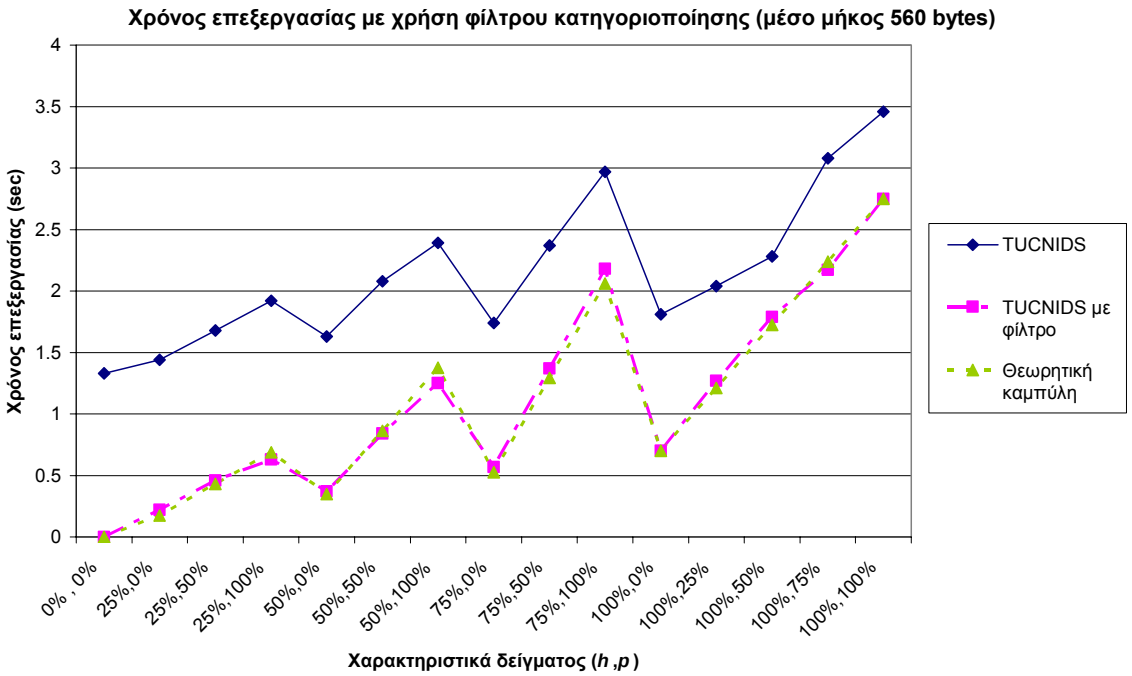
Εκτός από την επίδραση που έχει η εφαρμογή του φίλτρου στον χρόνο *εκτέλεσης*, αξίζει να παρατηρήσουμε την επίδραση που έχει το φίλτρο και στον χρόνο *επεξεργασίας*

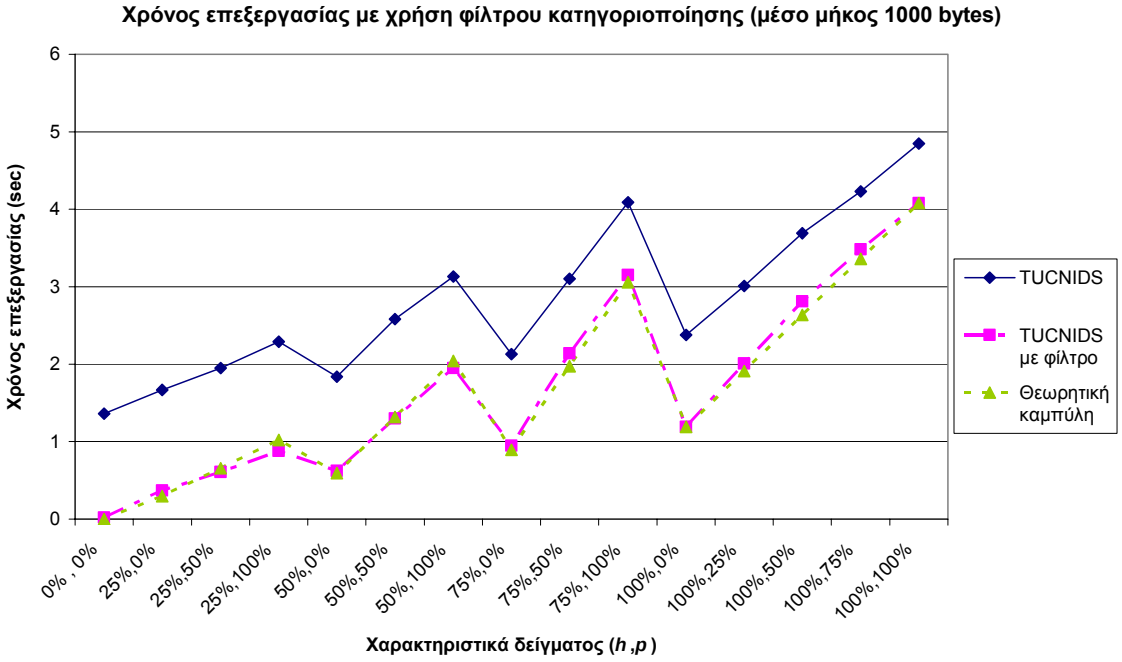
των πακέτων, χωρίς να υπάρχει η παρεμβολή του χρόνου πρόσβασης στο δίσκο. Οι γραφικές που προκύπτουν για το χρόνο επεξεργασίας φαίνονται στην εικόνα *6ζ*.

Στις γραφικές φαίνεται επίσης η θεωρητική καμπύλη, η οποία προκύπτει από την εξίσωση 5.4. Την πρώτη άγνωστη παράμετρο της εξίσωσης, τον χρόνο '$t_{payload_avg}$', την υπολογίζουμε διαιρώντας τον χρόνο που μας δίνει *το TUCNIDS μαζί με το φίλτρο κατηγοριοποίησης* για το δείγμα (100%,0%) με το πλήθος των πακέτων στην ακολουθία: στο δείγμα (100%,0%) γίνεται έλεγχος των δεδομένων σε όλα τα πακέτα χωρίς να έχουμε θετικές απαντήσεις, επομένως η διαίρεση του χρόνου με το πλήθος των πακέτων μας δίνει μια καλή εκτίμηση για τον μέσο χρόνο εξέτασης των δεδομένων όταν δεν υπάρχουν θετικές απαντήσεις.

Η παράμετρος 'Δt' της εξίσωσης υπολογίζεται αν πάρουμε τη διαφορά των χρόνων *του TUCNIDS μαζί με το φίλτρο κατηγοριοποίησης* ανάμεσα στα δείγματα (100%,100%) και (100%,0%) με το πλήθος των πακέτων στην ακολουθία: στο δείγμα (100%,100%) όλα τα πακέτα επιστρέφουν θετική απάντηση, επομένως ο χρόνος επεξεργασίας κάθε πακέτου είναι ($t_{payload_avg}$+ Δt), ενώ για το κάθε πακέτο του δείγματος (100%,0%) απαιτείται χρόνος $t_{payload_avg}$, άρα η διαφορά τους μας δίνει τη μέση τιμή του χρόνου Δt.







Εικόνα 6ζ) Από πάνω προς τα κάτω: η επίδραση του φίλτρου κατηγοριοποίησης στον χρόνο επεξεργασίας για μέσο μήκος πακέτων 300, 560 και 1000 bytes αντίστοιχα.

Από τις γραφικές τις εικόνας *6ζ* είναι προφανές ότι το φίλτρο κατηγοριοποίησης επικεφαλίδας μειώνει αισθητά τον απαιτούμενο χρόνο για όλες τις πιθανές εισόδους. Μια

δεύτερη διαπίστωση που προκύπτει εάν συγκρίνουμε τις γραφικές της εικόνας *6ζ* με εκείνες τις εικόνας *6ε* είναι ότι στην πραγματικότητα το φίλτρο κατηγοριοποίησης μειώνει τον χρόνο επεξεργασίας για *όλες* τις διαφορετικές εισόδους, γεγονός το οποίο δεν γινόταν εμφανές στην περίπτωση που παρατηρούσαμε τον συνολικό χρόνο εκτέλεσης (στον οποίο υπενθυμίζουμε ότι κυριαρχεί ο χρόνος πρόσβασης στο δίσκο).

Τέλος, παρατηρούμε σημαντική ομοιότητα ανάμεσα στη θεωρητική καμπύλη και στην καμπύλη των πειραματικών τιμών. Παρόλο που χρησιμοποιήσαμε δύο από τις πειραματικές μετρήσεις για να υπολογίσουμε τις παραμέτρους της θεωρητικής καμπύλης, το γεγονός ότι προσεγγίσαμε με μεγάλη ακρίβεια μια καμπύλη δεκαπέντε σημείων χρησιμοποιώντας μόλις δύο εκ των σημείων της επιβεβαιώνει πως η θεωρητική ανάλυση που κάναμε για τη λειτουργία του φίλτρου είναι ορθή.

6.5 Χρήση φίλτρου προσεγγιστικής αναγνώρισης συμβολοσειρών σε συνδυασμό με το φίλτρο κατηγοριοποίησης.

Η προσομοίωση της χρήσης των δύο φίλτρων, δηλαδή του φίλτρου κατηγοριοποίησης μαζί με το φίλτρο προσεγγιστικής αναγνώρισης, γίνεται με αντίστοιχο τρόπο όπως έγινε η προσομοίωση του συστήματος με το ένα φίλτρο. Το αποτέλεσμα της προεπεξεργασίας είναι ένα νέο αρχείο με λιγότερα πακέτα από το αρχικό δείγμα και ένα δεύτερο αρχείο με την επιπλέον πληροφορία για την κατηγοριοποίηση των πακέτων.

Αρχικά γίνεται η προεπεξεργασία κάθε δείγματος. Διαβάζουμε ένα-ένα τα πακέτα του δείγματος και για κάθε πακέτο χρησιμοποιούμε την μέθοδο κατηγοριοποίησης του λογισμικού για να βρούμε εάν το πακέτο ταιριάζει σε κάποια ομάδα κανόνων. Όσα πακέτα βρεθούν να μην ταιριάζουν με καμία ομάδα απλά αγνοούνται. Διαφορετικά, ελέγχουμε τα περιεχόμενα του πακέτου χρησιμοποιώντας τον αλγόριθμο που αντιστοιχεί στη συγκεκριμένη ομάδα.

Σε περίπτωση που το πακέτο βρεθεί να περιέχει συμβολοσειρές που ορίζουν οι κανόνες της ομάδας με την οποία ταίριαξε ή κάποιας συγκρουόμενης με αυτήν ομάδας, τότε αντιγράφουμε το πακέτο στο δείγμα εξόδου και καταγράφουμε σε ένα δεύτερο

αρχείο τον αριθμό της ομάδας με την οποία ταίριαξε το πακέτο. Εάν κάποιο πακέτο ταιριάξει με μία ομάδα αλλά ο έλεγχος του περιεχομένου με την συγκεκριμένη ομάδα και με όλες τις συγκρουόμενες είναι αρνητικός, τότε εξετάζεται το ενδεχόμενο ψευδούς αναγνώρισης: δημιουργούμε χρησιμοποιώντας κατάλληλη γεννήτρια έναν τυχάιο αριθμό ομοιόμορφα κατανεμημένο στο διάστημα από 0 μέχρι 1. Σε περίπτωση που ο αριθμός είναι μικρότερος ή ίσος με την πιθανότητα ψευδούς αναγνώρισης *f,* την οποία προσδιορίζουμε εμείς, τότε έχουμε ψευδή αναγνώριση και το πακέτο αντιγράφεται στο δείγμα εξόδου, ενώ καταγράφουμε και την πληροφορία κατηγοριοποίησης στο δεύτερο αρχείο. Όταν ο αριθμός είναι μεγαλύτερος από την πιθανότητα *f* απλά αγνοούμε το πακέτο.

Η προεπεξεργασία που περιγράψαμε εκτελείται μία φορά για κάθε χρησιμοποιούμενο δείγμα. Ο χρόνος που απαιτείται για την προεπεξεργασία δε μας ενδιαφέρει, καθώς για την προσομοίωσή μας θεωρούμε πως το υλικό είναι αρκετά γρήγορο ώστε να ολοκληρώνει την επεξεργασία κάθε πακέτου πριν την άφιξη του επόμενου, ενώ τα πακέτα πρωθούνται μέσο pipeline στο λογισμικό.

Η φάση του ελέγχου της κίνησης γίνεται ακριβώς όπως όταν έχουμε μόνο το φίλτρο κατηγοριοποίησης. Δηλαδή, το λογισμικό παίρνει σαν εισόδους το προεπεξεργασμένο δείγμα, από το οποίο διαβάζει ένα-ένα τα πακέτα, καθώς και το αρχείο με τις πληροφορίες για την κατηγοριοποίηση των πακέτων. Για κάθε πακέτο που διαβάζεται το λογισμικό εξετάζει μόνο τα πεδία του πρωτοκόλλου και του μήκους. Στη συνέχεια διαβάζεται από το δεύτερο αρχείο ο αύξων αριθμός της ομάδας κανόνων που ταιριάζει με το πακέτο και το λογισμικό ελέγχει τα περιεχόμενα του πακέτου χρησιμοποιώντας τον αλγόριθμο της συγκεκριμένης ομάδας. Σε περίπτωση που ο έλεγχος δεν επιστρέψει θετικό αποτέλεσμα εξετάζονται οι ομάδες με τις οποίες υπάρχει πιθανότητα σύγκρουσης.

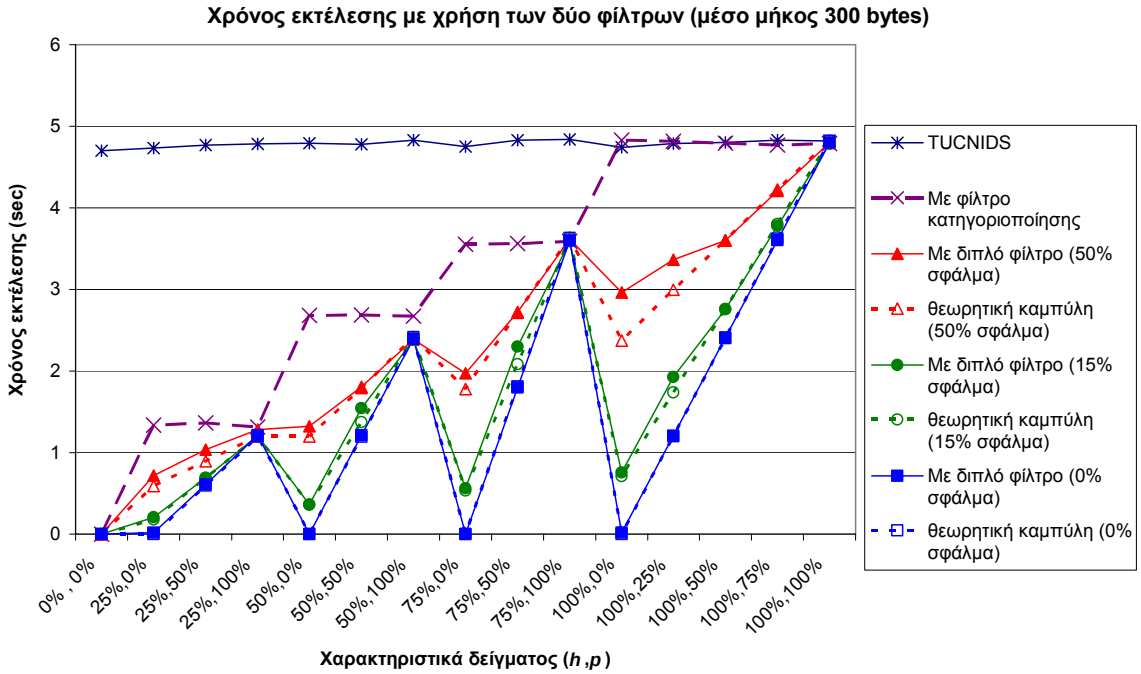
6.5.1 Επίδραση των δύο φίλτρων στον χρόνο εκτέλεσης.

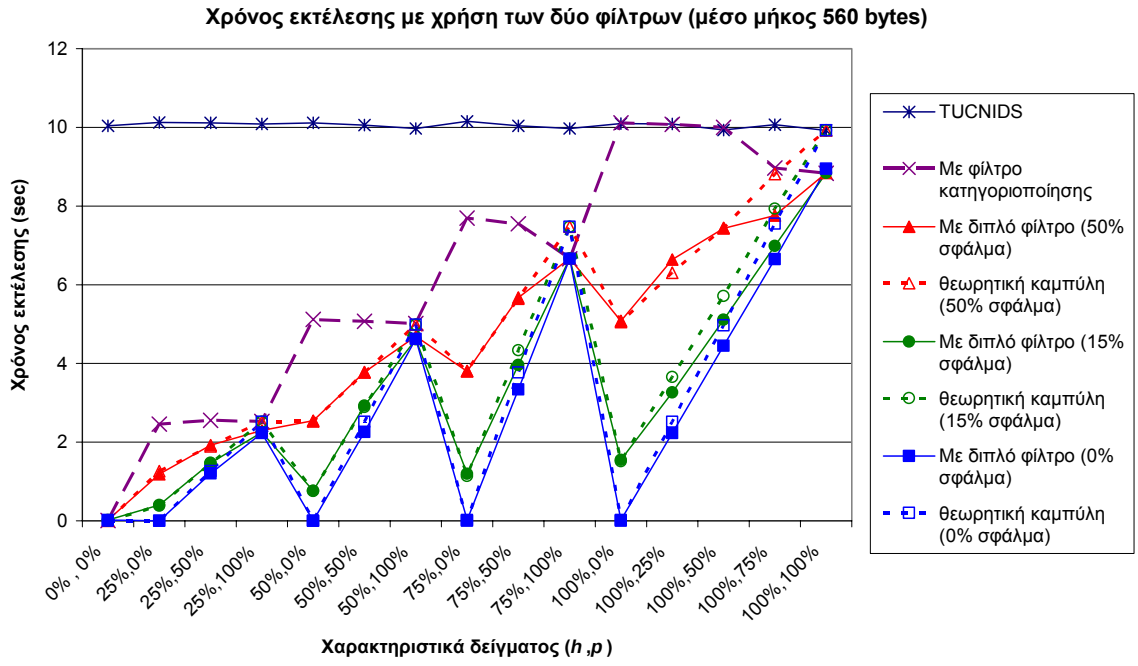
Για την πιθανότητα ψευδούς αναγνώρισης, την οποία ονομάζουμε *f*, εξετάσαμε τρεις διαφορετικές περιπτώσεις. Η πρώτη, που είναι και η ιδανική, είναι όταν η *f* είναι 0,

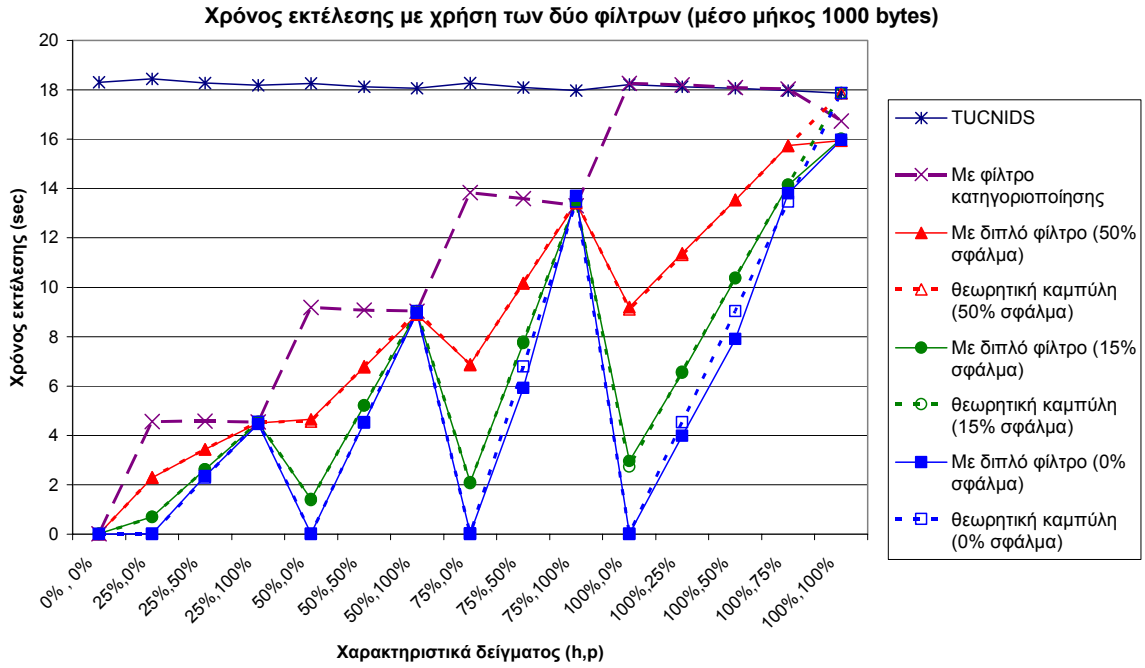
δηλαδή όταν η προσεγγιστική αναγνώριση συμβολοσειρών μας δίνει απόλυτα ακριβή αποτελέσματα. Πρακτικά μπορεί να μην είναι εφικτή μια τέτοια τιμή, όμως χρησιμεύει για τον υπολογισμό της μέγιστης δυνατής επιτάχυνσης. Η δεύτερη τιμή που θεωρήσαμε είναι το 15%, η οποία αντιστοιχεί σε μία μέτρια υλοποίηση, ενώ η τρίτη και τελευταία τιμή είναι το 50%, σε περίπτωση που έχουμε μια μάλλον κακή υλοποίηση.

Στις γραφικές παραστάσεις που ακολουθούν συγκρίνουμε το χρόνο εκτέλεσης (συμπεριλαμβανομένου του χρόνου I/O) για το TUCNIDS χωρίς φίλτρα υλικού, το χρόνο εκτέλεσης όταν χρησιμοποιούμε φίλτρο κατηγοριοποίησης και το χρόνο εκτέλεσης για τις τρεις περιπτώσεις υλοποίησης του φίλτρου προσεγγιστικής αναγνώρισης. Στην παράγραφο 6.5.3 θα δείξουμε τις αντίστοιχες γραφικές για τους χρόνους επεξεργασίας πακέτων.

Στις γραφικές συμπεριλαμβάνονται και οι θεωρητικές καμπύλες που αντιστοιχούν στις τρεις διαφορετικές τιμές της πιθανότητας ψευδούς αναγνώρισης f. Οι θεωρητικές καμπύλες σχεδιάστηκαν με βάση τη συνάρτηση $[(p+f-f*p)*h] * t$, όπου t ο χρόνος του TUCNIDS για το αντίστοιχο δείγμα. Η συγκεκριμένη θεωρητική καμπύλη προέκυψε από την εξίσωση 5.2. Οι λόγοι για τους οποίους επιλέχθηκε η εξίσωση 5.2 είναι ακριβώς οι ίδιοι με εκείνους που περιγράφονται στην παράγραφο 6.4.1 για τη θεωρητική καμπύλη του χρόνου εκτέλεσης όταν χρησιμοποιούμε το φίλτρο κατηγοριοποίησης







Εικόνα 6η) Από πάνω προς τα κάτω: οι χρόνοι εκτέλεσης για μέσο μήκος πακέτων 300, 560 και 1000 bytes αντίστοιχα.

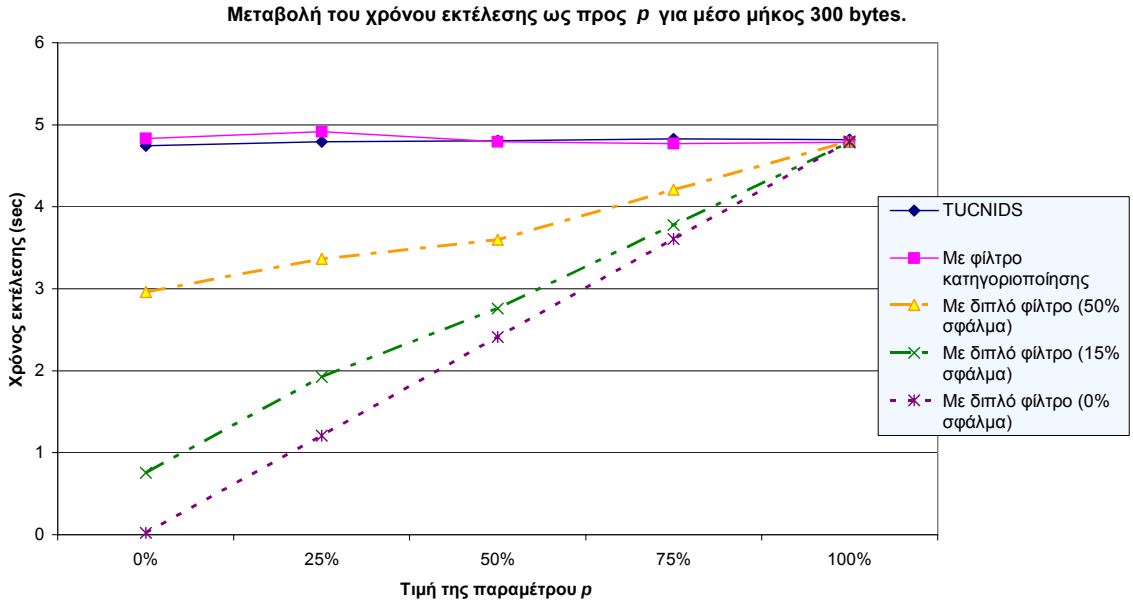
Από τις γραφικές προκύπτει ότι η χρήση του φίλτρου προσεγγιστικής αναγνώρισης συμβολοσειρών σε συνδυασμό με το φίλτρο κατηγοριοποίησης μπορεί να

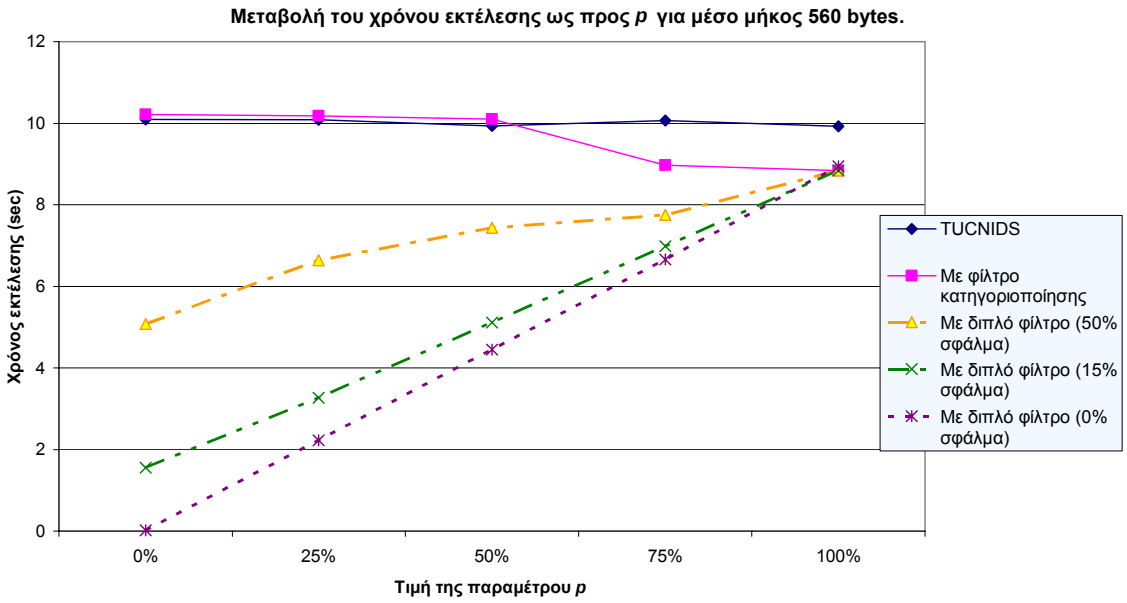
βοηθήσει σημαντικά στην ταχύτητα εκτέλεσης του προγράμματος, ειδικά στις περιπτώσεις που η πιθανότητα *p*, κατά την οποία τα πακέτα που ταιριάζουν στην επικεφαλίδα κάποιου κανόνα ικανοποιούν και το περιεχόμενο που επαληθεύει τον κανόνα, είναι αρκετά μικρή. Το γεγονός αυτό είναι απόλυτα λογικό, αφού όταν η πιθανότητα p είναι μικρή τότε πολλά πακέτα δεν περιέχουν ύποπτο περιεχόμενο, με αποτέλεσμα αρκετά από αυτά να εντοπιστούν από το φίλτρο προσεγγιστικής αναγνώρισης και να μην προωθηθούν στο λογισμικό.

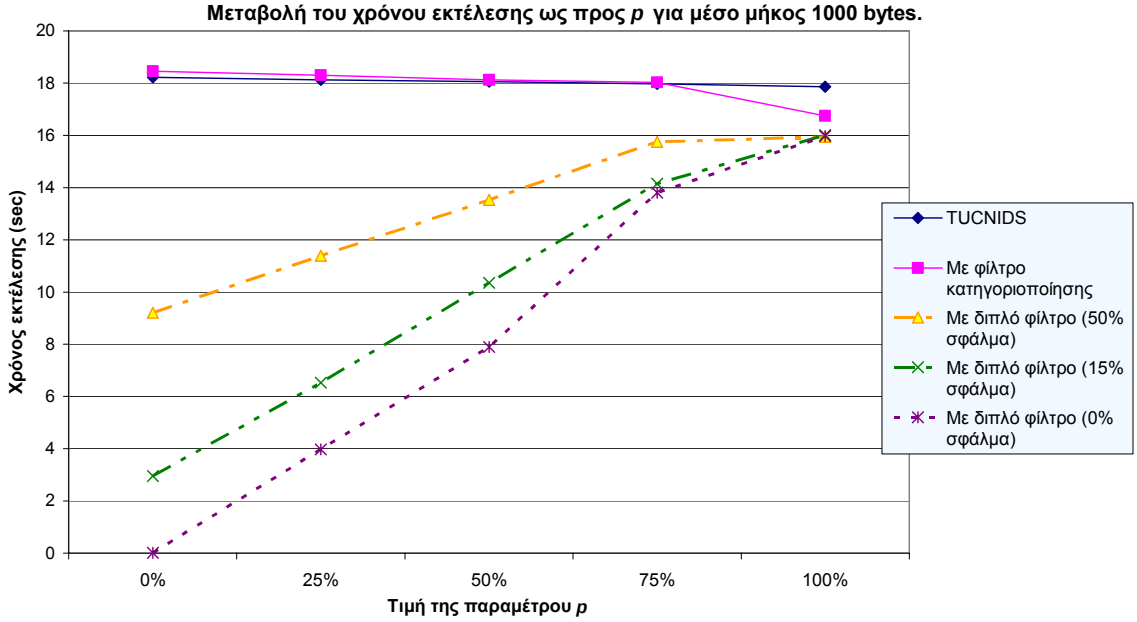
Επίσης, παρατηρούμε πως η μαθηματική σχέση με την οποία υπολογίζεται θεωρητικά ο χρόνος ύστερά από τη χρήση των δύο φίλτρων είναι, στις περισσότερες περιπτώσεις, πολύ κοντά στις πραγματικές πειραματικές μετρήσεις. Όμως υπάρχουν ορισμένα σημεία, ειδικά στην περίπτωση όπου το μέσου μήκους των πακέτων είναι 300 bytes, για τα οποία διαπιστώνουμε σημαντικές αποκλίσεις.

6.5.2 Μεταβολή του χρόνου εκτέλεσης ως προς την πιθανότητα *p*.

Προκειμένου να παρατηρήσουμε καλύτερα την εξάρτηση του χρόνου εκτέλεσης όταν χρησιμοποιούνται τα δύο φίλτρα από την παράμετρο *p*, διατηρούμε σταθερή την τιμή *h* στο 100% και μεταβάλλουμε την *p*. Με τον τρόπο αυτό προκύπτουν οι ακόλουθες γραφικές παραστάσεις.







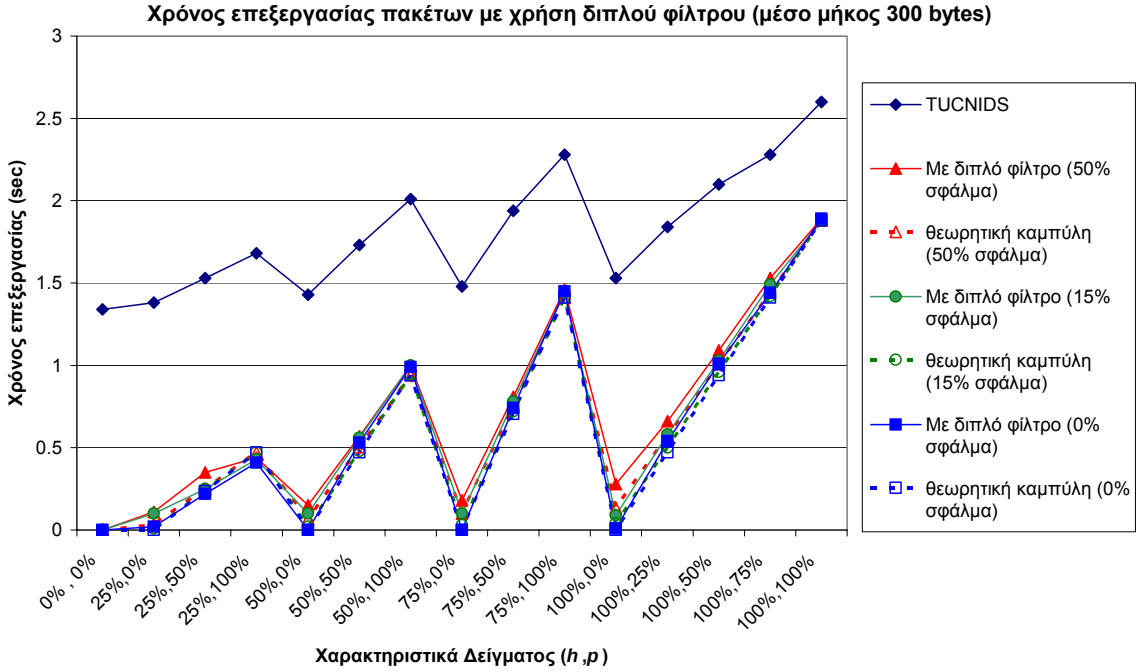
Εικόνα 6θ) Από πάνω προς τα κάτω: η μεταβολή του χρόνου εκτέλεσης ως προς την πιθανότητα p όταν η h είναι 0 για 300, 560 και 1000 bytes αντίστοιχα.

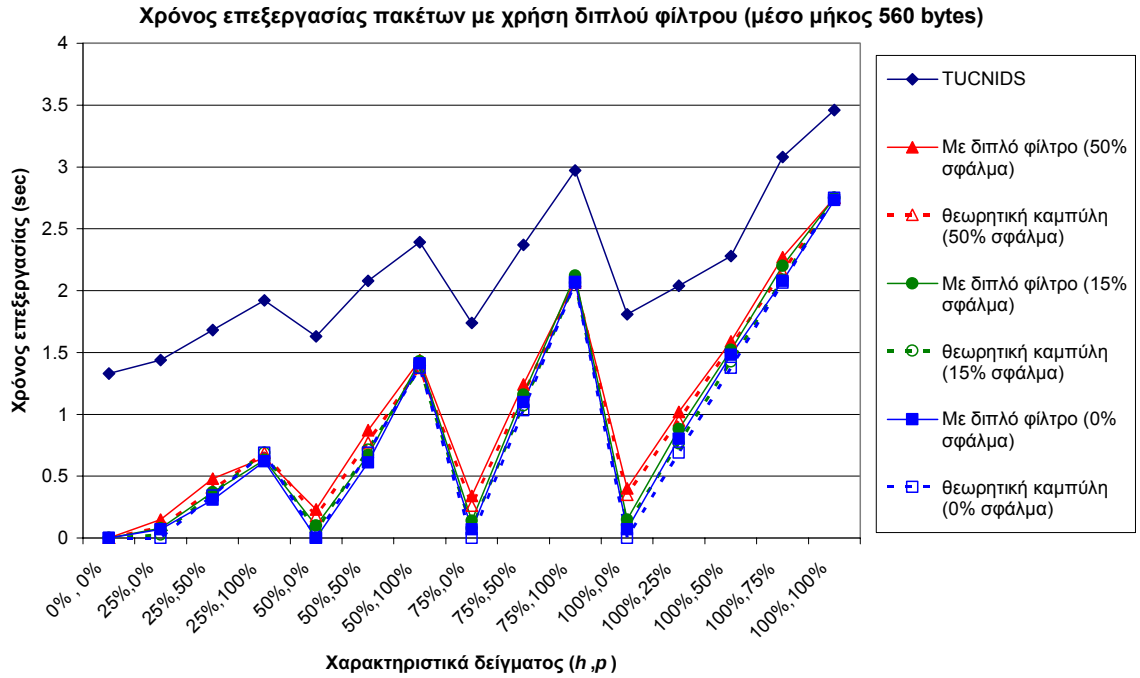
Από τις γραφικές παραστάσεις των *εικόνων 6η* και *6θ* διαπιστώνουμε πως η εφαρμογή του συνδυασμού των δύο φίλτρων υλικού μας προσφέρει μείωση του χρόνου εκτέλεσης, συνεπώς και επιτάχυνση, αντιστρόφως ανάλογη της πιθανότητας *p* και της

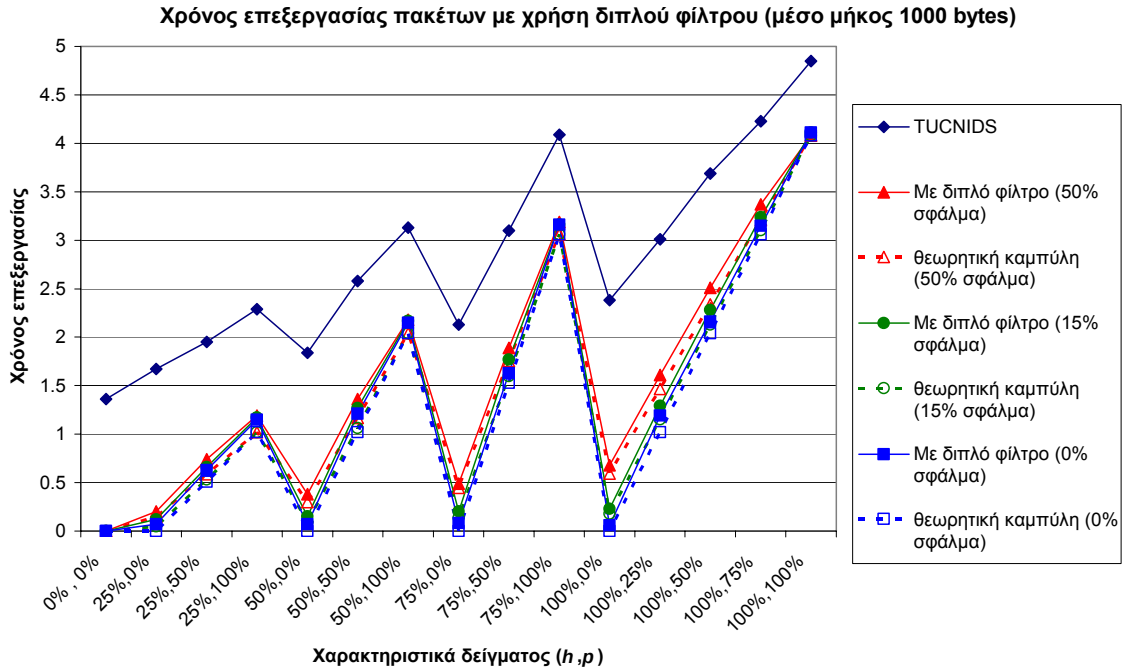
πιθανότητας *h*. Δηλαδή, όσο μικρότερες είναι οι δύο πιθανότητες τόσο μεγαλύτερα είναι τα οφέλη από τη χρήση των δύο φίλτρων. Επίσης, η απόδοση του φίλτρου προσεγγιστικής αναγνώρισης εξαρτάται από την ακρίβεια του φίλτρου, με την ταχύτητα εκτέλεσης να αυξάνεται καθώς μειώνεται η πιθανότητα ψευδούς αναγνώρισης.

6.5.3 Επίδραση των δύο φίλτρων στον χρόνο επεξεργασίας πακέτων.

Θεωρούμε σημαντικό να παρατηρήσουμε και την επίδραση που έχει η χρήση των δύο φίλτρων πάνω στο χρόνο *επεξεργασίας* των πακέτων, χωρίς να παρεμβάλλεται χρόνος πρόσβασης στο δίσκο ή άλλου είδους I/O. Στις ακόλουθες γραφικές παραστάσεις φαίνονται οι μετρήσεις που πήραμε για τους χρόνους επεξεργασίας καθώς επίσης και οι θεωρητικές καμπύλες που προκύπτουν με βάση την εξίσωση 5.5. Οι τιμές για τις παραμέτρους $t_{payload_avg}$ και Δt είναι οι ίδιες που χρησιμοποιήθηκαν στις γραφικές της εικόνας *6ζ*.







Εικόνα 61) Από πάνω προς τα κάτω: η επίδραση της χρήσης του διπλού φίλτρου στον χρόνο επεξεργασίας πακέτων για μέσο μήκος 300, 560 και 1000 bytes αντίστοιχα.

Το πρώτο πράγμα που διαπιστώνουμε από τις γραφικές είναι πως η χρήση του συνδυασμού των δύο φίλτρων μπορεί να προσφέρει σημαντική μείωση του χρόνου

επεξεργασίας ακόμα και για το χειρότερο είδος κίνησης, δηλαδή στην περίπτωση που οι πιθανότητες *p* και *h* είναι 100%.

Και σε αυτές τις γραφικές διαπιστώνουμε πως οι θεωρητικές καμπύλες παρουσιάζουν μικρές μόνο αποκλίσεις από τους χρόνους που μετρήσαμε πειραματικά. Σε κάθε περίπτωση που εμφανίζεται απόκλιση, παρατηρούμε πως η θεωρητική τιμή είναι μικρότερη από τον αντίστοιχο χρόνο που μετρήθηκε πειραματικά. Ένας λόγος για τον οποίο οι θεωρητικές τιμές είναι μικρότερες από τις πραγματικές είναι πιθανότατα το γεγονός ότι δε λάβαμε υπόψην μας στο θεωρητικό μοντέλο τον έλεγχο που γίνεται από το λογισμικό για πιθανές συγκρούσεις ανάμεσα στις ομάδες κανόνων. Το φαινόμενο των συγκρούσεων, όμως, είναι αρκετά δύσκολο να συμπεριληφθεί στο θεωρητικό μοντέλο. Παρόλα αυτά, το θεωρητικό μοντέλο ως έχει διαπιστώσαμε ότι μας δίνει με αρκετά μεγάλη ακρίβεια ένα κάτω όριο για τον χρόνο επεξεργασίας των πακέτων από το λογισμικό

6.6 Ποσοστιαία συνεισφορά κάθε φίλτρου.

Οι πειραματικές μετρήσεις μας έδειξαν ότι τα φίλτρα υλικού συμβάλλουν σημαντικά στη μείωση του χρόνου που χρειάζεται το λογισμικό για τον έλεγχο μίας ακολουθίας πακέτων. Στο σημείο αυτό θεωρούμε ενδιαφέρον να εξετάσουμε το βαθμό με τον οποίο κάθε φίλτρο συμβάλλει στη μείωση του χρόνου επεξεργασίας των πακέτων.

Για τον παραπάνω λόγο σχεδιάζουμε τη γραφική παράσταση που μας δείχνει το ποσοστό μείωσης του καθαρού χρόνου επεξεργασίας των διάφορων δείγματων για κάθε ένα από τα δύο φίλτρα ξεχωριστά. Τα ποσοστά προκύπτουν από τους χρόνους επεξεργασίας, στους οποίους δεν περίλαμβάνεται ο χρόνος πρόσβασης στο δίσκο.

Για τον υπολογισμό του ποσοστού μείωσης του χρόνου που οφείλεται στη χρήση του φίλτρου κατηγοριοποίησης χρησιμοποιούμε τη σχέση:

$$gain_{header_filter}\% = 100*(t-t_{header_filter}) / t$$

όπου ‘$gain_{header_filter}\%$’ είναι το ποσοστό κατά το οποίο μειώθηκε ο χρόνος επεξεργασίας του δείγματος εξαιτίας της λειτουργίας του φίλτρου κατηγοριοποίησης επικεφαλίδων, ‘t’

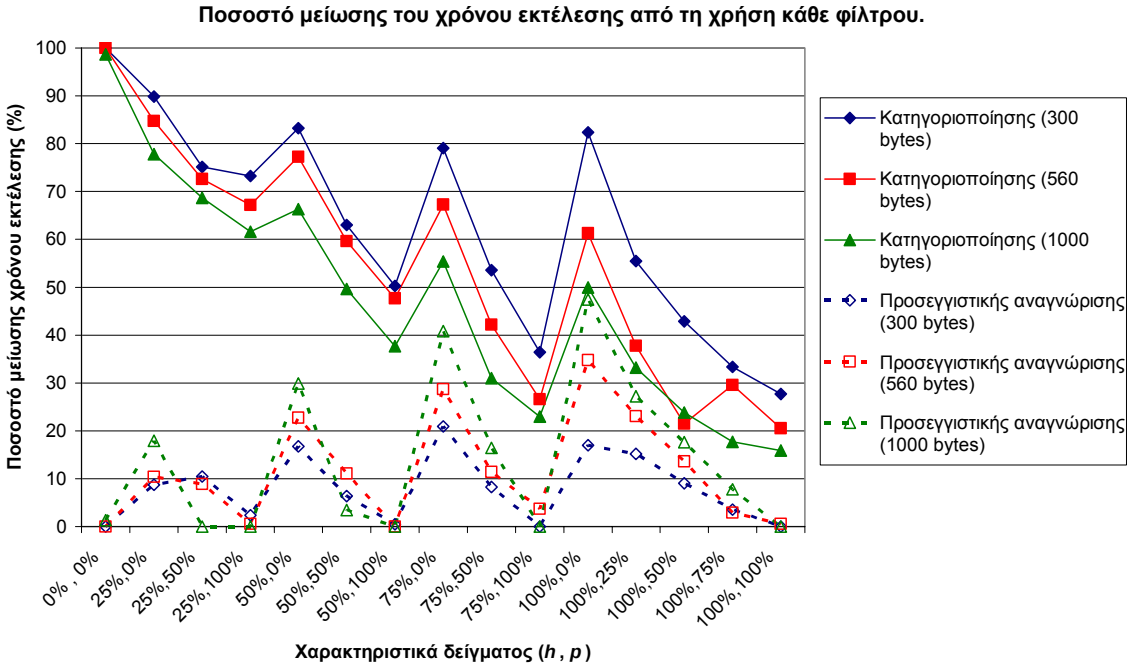
ο χρόνος που απαιτείται για την επεξεργασία του δείγματος όταν δε χρησιμοποιούνται φίλτρα υλικού και, τέλος, 't_{header_filter}' ο χρόνος που απαιτείται για την επεξεργασία του ίδιου δείγματος όταν χρησιμοποιείται το φίλτρο κατηγοριοποίησης.

Αντίστοιχα, για τον υπολογισμό της ποσοστιαίας μείωσης που οφείλεται αποκλειστικά στο φίλτρο προσεγγιστικής αναζήτησης χρησιμοποιούμε τη σχέση:

$$gain_{approximate_filter}\% = 100*(t_{header_filter} - t_{approximate_filter}) / t$$

όπου '$gain_{approximate_filter}\%$' το ποσοστό κατά το οποίο μειώνεται ο *αρχικός* χρόνος επεξεργασίας (δηλαδή ο χρόνος επεξεργασίας όταν δεν χρησιμοποιείται κανένα φίλτρο) λόγω της χρήσης του φίλτρου προσεγγιστικής αναζήτησης, ενώ '$t_{approximate_filter}$' είναι ο χρόνος επεξεργασίας του δείγματος όταν χρησιμοποιείται ο συνδυασμός των δύο φίλτρων.

Για το ποσοστό μείωσης λόγω του φίλτρου προσεγγιστικής αναζήτησης εξετάζουμε μόνο την ιδανική περίπτωση, κατά την οποία η πιθανότητα *f* ψευδούς αναγνώρισης είναι 0. Τα αποτελέσματα παρουσιάζονται στην εικόνα *6ια*.



Εικόνα 6ια) Το ποσοστό κατά το οποίο μειώνεται ο χρόνος επεξεργασίας των δειγμάτων από τη χρήση των δύο φίλτρων υλικού.

Το άθροισμα των δύο ποσοστών για το ίδιο δείγμα μας δίνει το συνολικό ποσοστό κατά το οποίο μειώθηκε ο χρόνος επεξεργασίας του συγκεκριμένου δείγματος.

Παρατηρούμε ότι σε κάθε περίπτωση το ποσοστό κατά το οποίο μειώνεται ο χρόνος επεξεργασίας λόγω του φίλτρου κατηγοριοποίησης είναι μεγαλύτερο από το αντίστοιχο ποσοστό για το φίλτρο προσεγγιστικής αναζήτησης. Το αποτέλεσμα αυτό μας δείχνει πως η λειτουργία μόνο του φίλτρου κατηγοριοποίησης συμβάλει σημαντικά στην επιτάχυνση ενός συστήματος ανίχνευσης εισβολών δικτύου. Παρόλα αυτά, η χρήση του συνδυασμού των δύο φίλτρων προσφέρει ακόμα καλύτερη απόδοση.

6.7 Συμπεράσματα.

Από τις πειραματικές μετρήσεις που έγιναν προκύπτει ότι η χρήση των δύο φίλτρων υλικού που περιγράψαμε στην ενότητα 5.3, δηλαδή του φίλτρου κατηγοριοποίησης επικεφαλίδων και του φίλτρου προσεγγιστικής αναγνώρισης συμβολοσειρών, σε συνδυασμό με ένα σύστημα ανίχνευσης εισβολών δικτύου υλοποιημένο σε λογισμικό μπορεί να βοηθήσει σημαντικά στην απόδοση του λογισμικού.

Επίσης, οι χρόνοι που μετρήσαμε πειραματικά επιβεβαιώνουν τα θεωρητικά μοντέλα που αναπτύχθηκαν για τον υπολογισμό της απόδοσης των φίλτρων υλικού και περιγράφονται από τις συναρτήσεις 5.1 μέχρι και 5.5. Το γεγονός αυτό σημαίνει ότι θα μπορούμε να έχουμε μια καλή εκτίμηση της βελτίωσης που θα παρέχει σε ένα σύστημα ανίχνευσης εισβολών η χρήση των φίλτρων υλικού.

Επίλογος

Με την πάροδο του χρόνου, η απαίτηση για γρήγορα και αποδοτικά συστήματα ανίχνευσης εισβολών δικτύου θα γίνεται ολοένα και εντονότερη. Παράλληλα, τα τωρινά δεδομένα υποδεικνύουν πως η δυνατότητα ενός συστήματος ανίχνευσης εισβολών σε λογισμικό να ανταπεξέρχεται σε καταστάσεις υψηλού φόρτου και σε κινήσεις αυξημένης επικινδυνότητας συνεχώς θα περιορίζεται.

Όπως φαίνεται από τα αποτελέσματα της εργασίας αυτής, η χρήση κατάλληλων διατάξεων υλοποιημένων σε υλικό για την επεξεργασία των διερχόμενων πακέτων μπορεί να μειώσει σημαντικά τον αριθμό των πακέτων που επεξεργαζονται από το λογισμικό αλλά και τον απαιτούμενο χρόνο για την επεξεργασία τους.

Προτείναμε δύο φίλτρα υλοποιημένα σε υλικό, ένα φίλτρο που θα κατηγοριοποιεί την επικεφαλίδα των εισερχόμενων πακέτων και ένα φίλτρο που θα εκτελεί προσεγγιστική αναγνώριση συμβολοσειρών. Τα συγκεκριμένα φίλτρα θα συνεργάζονται κατάλληλα με το λογισμικό και θα ελέγχουν τα πακέτα καθώς αυτά έρχονται από τις ζεύξεις, πρωτού να φτάσουν στο λογισμικό. Τα φίλτρα θα προωθούν στο λογισμικό ανίχνευσης εισβολών δικτύου μόνο τα πακέτα που χρειάζονται λεπτομερή έλεγχο, ενώ παράλληλα θα μεταφέρουν χρήσιμη πληροφορία στο λογισμικό ώστε να αποφεύγονται περιττοί έλεγχοι.

- Το φίλτρο κατηγοριοποίησης επικεφαλίδων εκτελεί τη διαδικασία του ελέγχου της επικεφαλίδας του πακέτου και της επιλογής της ομάδας κανόνων με την οποία ταιριάζει το πακέτο, διαδικασία που κανονικά εκτελείται από το λογισμικό. Η υλοποίησή του φίλτρου θα είναι σχετικά απλή καθώς χρειάζεται μόνο η συγκρίση συγκεκριμένων παράμετρων με καθορισμένες τιμές, ενώ η χρήση του μπορεί να μειώσει σημαντικά το χρόνο που χρειάζεται το λογισμικό. Όπως έδειξαν τα αποτελέσματα, για μέσο μήκος πακέτου 300 bytes (τιμή αρκετά κοντά στο μέσο μήκος πακέτων του διαδικτύου) το φίλτρο κατηγοριοποίησης μειώνει τον απαιτούμενο χρόνο από 28% μέχρι και 100%, ενώ για τις περισσότερες ακολουθίες πακέτων έχουμε μείωση κατά περισσότερο από 50%. Το ακριβές όφελος είναι αντίστροφα ανάλογο της πιθανότητας p. Συνεπώς, το φίλτρο κατηγοριοποίησης από

μόνο του μπορεί να αποτελέσει ένα σχετικά απλό και αποτελεσματικό συμπλήρωμα ενός λογισμικού ανίχνευσης εισβολών δικτύου.

- Το φίλτρο προσεγγιστικής αναζήτησης συμβολοσειρών μπορεί να χρησιμοποιηθεί σε συνδυασμό με το φίλτρο κατηγοριοποίησης. Η λειτουργία του φίλτρου είναι να εκτελεί μία προσεγγιστική αναζήτηση των ύποπτων συμβολοσειρών και να μας απαντάει αν το πακέτο είναι ασφαλές ή όχι. Η υλοποίησή του είναι σαφώς δυσκολότερη και πιο απαιτητική σε πόρους από το φίλτρο κατηγοριοποίησης ενώ τα οφέλη είναι σε όλες τις περιπτώσεις μικρότερα, προσφέροντας μας μείωση του αρχικού χρόνου επεξεργασίας κατά 21% στην καλύτερη περίπτωση (για μέσο μήκος πακέτων 300 bytes). Το ακριβές όφελος είναι ανάλογο της πιθανότητας h και αντιστρόφως ανάλογο της p, ενώ αυξάνεται σε περίπτωση που αυξηθεί το μέσο μήκος των πακέτων και μπορεί να φτάσει μέχρι το 48% για μέσο μήκος πακέτων 1000 bytes. Δεδομένης της πολυπλοκότητας, του κόστους υλοποίησης και του περιορισμένου κέρδους που μας προσφέρει, το φίλτρο προσεγγιστικής αναζήτησης αποτελεί ωφέλιμη λύση μόνο σε περιπτώσεις όπου ενδιαφερόμαστε αποκλειστικά για τη μείωση του χρόνου, χωρίς κανέναν περιορισμό κόστους.

Επιπλέον, αναπτύξαμε ένα αρκετά λεπτομερές μοντέλο της απόδοσης των δύο φίλτρων υλικού, το οποίο επιβεβαιώθηκε με αρκετά μεγάλη ακρίβεια από τους χρόνους που μετρήσαμε πειραματικά.

Το θεωρητικό μοντέλο που αναπτύχθηκε θα ισχύει με την προϋπόθεση ότι τα φίλτρα είναι υλοποιημένα σε υλικό αρκετά γρήγορο ώστε η επεξεργασία κάθε πακέτου να ολοκληρώνεται πριν από την άφιξη του επόμενου, έτσι ώστε να μη συσσωρεύονται πακέτα και δημιουργούνται καθυστερήσεις.

Προκειμένου να είναι αποδοτική και από οικονομικής άποψης η χρήση των προτεινόμενων φίλτρων υλικού, η υλοποιησή τους θα πρέπει να γίνεται πάνω σε κάποιου είδους επαναπρογραμματιζόμενη λογική. Καθώς οι κανόνες που χρησιμοποιεί το σύστημα ανίχνευσης εισβολών αλλάζουν πολύ εύκολα, θα ήταν υπερβολικά ασύμφορη η κατασκευή νέων ολοκληρωμένων κυκλώματων κάθε φορά που τροποποιείται κάποιος κανόνας ή προστίθεται ένας νέος. Το γεγονός αυτό καθιστά απαραίτητη και την κατασκευή κατάλληλων προγραμμάτων που θα εξάγουν από τα δεδομένα ενός

συστήματος ανίχνευσης τον απαραίτητο κώδικα για τον προγραμματισμό της λογικής στην οποία υλοποιούνται τα φίλτρα.

Το ακριβές όφελος της χρήσης των φίλτρων υλικού εξαρτάται από τις στατιστικές παραμέτρους h και p της διερχόμενης κίνησης, οι οποίες εξαρτώνται και από το σύνολο των κανόνων που χρησιμοποιούνται, καθώς και από την πιθανότητα f που περιγράφει την ακρίβεια του φίλτρου προσεγγιστικής αναγνώρισης συμβολοσειρών. Είναι λογικό, όμως, να υποθέσουμε πως για τη διερχόμενη κίνηση ενός τυπικού δικτύου και με τη χρήση ενός λογικού συνόλου κανόνων (δηλαδή ενός συνόλου που δεν ενεργοποιείται για όλα τα πιθανά πακέτα) οι πιθανότητες αυτές δε θα είναι ιδιαίτερα κοντά στη μονάδα. Επιπλέον, μια καλή υλοποίηση του φίλτρου προσεγγιστικής αναγνώρισης θα έχει πιθανότητα σφάλματος f μικρότερη από 10%. Υπό αυτές της συνθήκες γίνεται προφανές πως τα φίλτρα υλικού μπορούν να προσφέρουν σημαντική υποβοήθηση στο λογισμικό.

Βιβλιογραφικές αναφορές.

[1] A. Aho and M. Corasick. Fast pattern matching: an aid to bibliographic search. *Commun. ACM*, 18(6):333-340, June 1975.

[2] R. Boyer and J. Moore. A fast string matching algorithm. *Commun. ACM*, 20(10): 762-772, October 1977.

[3] S. Wu and U. Manber. A fast algorithm for multi-pattern searching. Technical Report TR-94-17, University of Arizona, 1994.

[4] Knuth, D.E., Morris, J.H., Jr., and Pratt, V.R. Fast pattern matching in strings. TR CS-74-40, Stanford University, Stanford,California, 1974.

[5] R. Cole. Tight bounds on the complexity of the Boyer-Moore pattern matching algorithm, *SIAM Journal on Computing* 23 (5): 1075-1091

[6] R. N. Horspool**.** Practical fast searching in strings, *Software - Practice & Experience*, 10(6):501-506, 1980.

[7] A. Apostolico and R.Giancarlo. The Boyer-Moore-Galil string searching strategies revisited, *SIAM Journal on Computing*, 15(1):98-105, 1986

[8] M. Crochemore, A. Czumaj, L. Gasienec, S. Jarominek, T. Lecroq, W. Plandowski and W. Rytter. Deux méthodes pour accélérer l'algorithme de Boyer-Moore, in *Théorie des Automates et Applications, Actes des 2e Journées Franco-Belges*, D. Krob ed., Rouen, France, 1991, pp 45-63, PUR 176, Rouen, France.

[9] K. G. Anagnostakis, E. P. Markatos, S. Antonatos, and M. Polychronakis. E2xB: A domainspecific string matching algorithm for intrusion detection. *Proceedings of the 18th IFIP International Information Security Conference (SEC2003)*, May 2003.

[10] E.P Markatos, S. Antonatos, M. Polychronakis and K.G Anagnostakis. ExB: Exclusion-based signature matching for intrusion detection. *Proceedings of the IASTED International Conference on Communications and Computer Networks (CCN)*, pp. 146-152, Cambridge, USA, November 2002.

[11] Mark Norton & Dan Roelker of Sourcefire. Snort 2.0 Rule Optimizer. Διαθέσιμο από την ιστοσελίδα www.sourcefire.com/technology/whitepapers.htm, June 2004

[12] Mark Norton & Dan Roelker of Sourcefire. Snort 2.0 Protocol Flow Analyzer. Διαθέσιμο από την ιστοσελίδα www.sourcefire.com/technology/whitepapers.htm, June 2004.

[13] Mark Norton & Dan Roelker of Sourcefire. Snort 2.0 Multi-Rule Inspection Engine. Διαθέσιμο από την ιστοσελίδα www.sourcefire.com/technology/ whitepapers.htm, June 2004

[14] Mark Norton & Dan Roelker of Sourcefire. Snort 2.0 Detection Revisited. Διαθέσιμο από την ιστοσελίδα www.sourcefire.com/technology/whitepapers.htm, June 2004

[15] S. Antonatos, K.G. Anagnostakis, E.P. Markatos & M. Polychronakis; *Performance Analysis of Content Matching Intrusion Detection Systems*; Proceedings of the IEEE/IPSJ Symposium on Applications and the Internet (SAINT 2004), Tokyo, 26-30 January 2004.

[16] Burton Bloom. Space/time trade-offs in hash coding with allowable errors. Communications of the ACM, 13(7):422-426, 1970.

[17] C.J Coit, S. Staniford and J. McAlerney. Towards faster pattern matching for intrusion detection, or exceeding the speed of Snort. Proceedings of the 2nd DARPA Information Survivability Conference and Exposition (DISCEX II), 2002.

[18] R. A. Baeza-Yates, 'Improved string searching', *Software Practice & Experience,* 19**,** 257–271 (1989).

[19] M. Fisk and G. Varghese. An analysis of fast string matching applied to content-based forwarding and intrusion detection. Technical Report CS2001-0670 (updated version), University of California - San Diego, 2002.

[20] D. Gusfield. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. University of California Press, 1997.

[21] Raja Epsilon, Jun Ke, and Carey Williamson, "Analysis if ISP IP/ATMNetwork Traffic Measurements", *Performance Evaluation Review*, vol. 27, no. 2, pp. 15–24, 1999.

[22] A. Dollas, D. Pnevmatikatos, N. Aslanides, S. Kavvadias, E. Sotiriades, S. Zogopoulos, K. Papademetriou, N. Chrysos, K. Harteros, E. Antonidakis, N. Petrakis, *Architecture and Applications of PLATO, a Reconfigurable Active Network Platform*. Proceedings, 9th International IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM-2001), Rohnert Park, CA, April 30 - May 2, 2001.

[23] I. Sourdis and D. Pnevmatikatos. Fast, large-scale string match for a 10Gbps FPGAbased network intrusion detection system. *Proceedings of 13th International Conference on Field Programmable Logic and Applications*, Lisbon, Portugal, September 2003.

[24] I. Sourdis and D. Pnevmatikatos. Pre-decoded CAMs for efficient and high-speed nids pattern matching. In *IEEE Symposium on Field-Programmable Custom Computing Machines*. Napa, CA, USA, April 2004.

[25] M. Attig, S. Dharmapurikar, and J. Lockwood. Implementation results of bloom filters for string matching. *IEEE Symposium on Field-Programmable Custom Computing Machines*, Napa, CA, USA, April 2004.

[26] C. R. Clark and D. E. Schimmel. Efficient reconfigurable logic circuit for atching complex network intrusion detection patterns. *Proceedings of 13th International Conference on Field Programmable Logic and Applications*, Lisbon, Portugal, September 2003.

[27] M. Gokhale, D. Dubois, A. Dubois, M. Boorman, S. Poole, and V. Hogsett. Granidt: Towards gigabit rate network intrusion detection technology. *Proceedings of 12th International Conference on Field Programmable Logic and Applications*, France, 2002.

[28] Long Bu and John A. Chandy. FPGA based network intrusion detection using content addressable memories. *IEEE Symposium on Field-Programmable Custom Computing Machines*. Napa, CA, USA, April 2004.

Διαδικτυακές αναφορές.

[29] www.snort.org
[30] www.ethereal.com
[31] www.winpcap.polito.it
[32] www.windump.polito.it
[33] www.tcpdump.org
[34] www.freesoft.org/CIE/Course/Section4/8.htm
[35] www.zytrax.com/tech/protocols/tcp.html
[36] www-igm.univ-mlv.fr/~lecroq/string/node14.html#SECTION00140
[37] www-igm.univ-mlv.fr/~lecroq/string/node15.html#SECTION00150
[38] www-igm.univ-mlv.fr/~lecroq/string/node16.html#SECTION00160
[39] www-igm.univ-mlv.fr/~lecroq/string/node18.html#SECTION00180
[40] www.ics.forth.gr/carv/acs/ACS/Activities/Projects/ids.html
[41] www.snort.org/docs/snort-win2k.htm
[42] http://portal.acm.org/portal.cfm
[43] http://lists.sourceforge.net/lists/listinfo/snort-users
[44] www.sourcefire.com/technology/whitepapers.htm
[45] www.postel.org/pipermail/end2end-interest/2003-April/002941.html
[46] www.mhl.tuc.gr