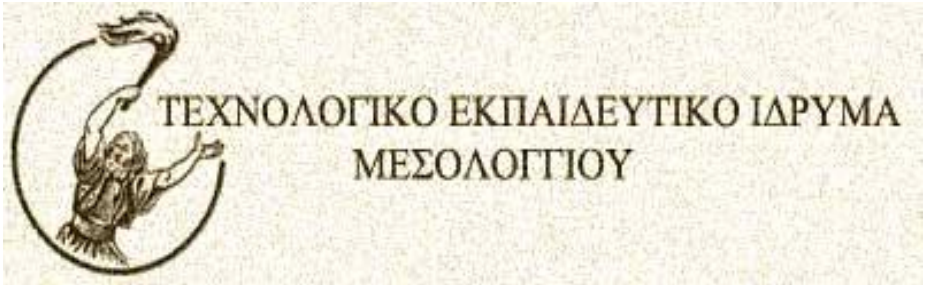


Προπτυχιακή Διπλωματική εργασία

Επιθέσεις Denial of Service (DoS) στις δομές πρόβλεψης διακλαδώσεων (branch predictor) σε σύγχρονους επεξεργαστές

Καραμάνης Παναγιώτης
ΑΜ : 00311

Επιβλέπων Καθηγητής :Κεραμιδάς Γεώργιος

Περίληψη

Στα πλαίσια αυτής της εργασίας θα μελετηθεί και θα αναλυθεί η συμπεριφορά του GShare (Global History with Index Sharing) καθώς και του perceptron branch predictor. Οι κώδικες που εξομοιώνουν την λειτουργία του GShare branch predictor και του perceptron είναι γραμμένοι σε C++ και δημιουργήθηκαν στα πλαίσια του CBP-3 (Championship Branch Prediction 3). Εμείς θα μελετήσουμε τις παραμέτρους των δύο υλοποιήσεων ώστε να βρούμε τις τιμές για τις οποίες παρουσιάζουν την καλύτερη συμπεριφορά (μικρότερος αριθμός από miss-predictions) και το MPPKI (Misprediction Penalty Per Kilo Instructions) . Ενώ παράλληλα θα επιτεθούμε σε αυτούς δημιουργώντας ακολουθίες εντολών που "μπερδεύουν" τους branch predictors.

Αρχικά θα δούμε εν συντομία πως είναι δομημένος ο εξομοιωτής και θα εστιάσουμε αναλυτικότερα στο format που έχουν τα trace files και πως αυτά διαβάζονται από τον εξομοιωτή. Έπειτα θα αναλυθούν οι branch predictors καθώς και η συμπεριφορά αυτών (πίνακες - διαγράμματα), για τις διάφορες τιμές των παραμέτρων τους. Και τέλος θα αναφέρουμε την τεχνική με την οποία δημιουργήσαμε τα δικά μας trace files τα οποία παρουσιάζουν ~100% αστοχία στο branch predictor.

Abstract

In the frames of this work will be studied and analyzed the behavior of GShare (Global History with Index Sharing) and perceptron branch predictor. The codes that assimilate the operation of GShare and perceptron branch predictor are written in C++ and were created in the frames the cbp-3 (Championship Branch Prediction 3). We will study the parameters of these two concretisations so that we find the prices for which they present the better behavior (smaller number than miss-predictions) and the MPPKI (Misprediction Penalty Per Kilo Instructions). While at the same time we attack in them creating sequences of commands that "tangle" branch predictors.

Initially we will see brevity how the simulator is structured and we will focus more analytically in the format that they trace files have and that these are read by simulator. Then they will be analyzed branch predictors as well as the behavior of these (tables - diagrams), for the various prices of their parameters. And finally we will report the technique with which we created ours trace files that present ~100% misprediction in branch predictor.

Περιεχόμενα

Ορισμός

Στην αρχιτεκτονική υπολογιστών, ένας branch predictor είναι το μέρος ενός επεξεργαστή που καθορίζει εάν ένας κλάδος στη ροή οδηγίας ενός προγράμματος είναι πιθανό να ληφθεί ή όχι. Αυτό καλείται πρόβλεψη κλάδων. Οι branch predictors είναι κρίσιμοι στους σημερινούς σύγχρονους, superscalar επεξεργαστές για την επίτευξη της υψηλής επίδοσης. Επιτρέπουν στους επεξεργαστές να προσκομίσουν και να εκτελέσουν τις οδηγίες χωρίς αναμονή για έναν κλάδο που επιλύεται. Ο σκοπός του branch predictor είναι να βελτιώσει τη ροή στις εντολές διασωλήνωσης. Οι branch predictors είναι κρίσιμοι στους σημερινούς μικροεπεξεργαστές διασωλήνωσης για την επίτευξη της υψηλής επίδοσης.

Όλοι οι διοχετευμένοι επεξεργαστές κάνουν κάποιας μορφής πρόβλεψη διακλάδωσης, επειδή πρέπει να υποθέσουν τη διεύθυνση της επόμενης οδηγίας που προσκομίζει προτού να εκτελεσθεί η τρέχουσα οδηγία. Νωρίτερα η ΚΜΕ δε χρειάζεται να κάνει κάποια πρόβλεψη διακλάδωσης.

1 Εισαγωγή

Οι σύγχρονοι επεξεργαστές στηρίζονται στους περίπλοκους μηχανισμούς για να εκμεταλλευτούν τον παραλληλισμό επιπέδων οδηγίας (ILP instruction level parallelism). Παραδείγματος χάριν, οι επεξεργαστές επιλύουν τους κινδύνους ελέγχου με την προσπάθεια να προβλεφθεί εάν ένας κλάδος θα ληφθεί ή όχι. Αυτή η τεχνική, η οποία καλείται πρόβλεψη διακλαδώσεων, η οποία μειώνει το εκμεταλλεύσιμο stalls προκαλείται από τις εξαρτήσεις ελέγχου.
Αλλά καθώς οι σχεδιαστές υλικού στρώνουν το δρόμο για ακόμα περισσότερες ILP μέσω της διοχέτευσης και άλλων δυναμικών σχεδίων, ο αντίκτυπος των εξαρτήσεων ελέγχου στην αποτελεσματικότητα ενός επεξεργαστή αυξάνεται. Επιπλέον, καθώς το βάθος της σωλήνωσης αυξάνεται, η ποινή για μια λανθασμένη πρόβλεψη γίνεται αυστηρότερη, αυξάνοντας την απαίτηση για τις ακριβείς προβλέψεις. Εντούτοις, καθώς ο κύκλος ζωής ρολογιών μειώνεται και οι ταχύτητες επεξεργαστών αυξάνονται η δυνατότητα χρήσης περίπλοκης λογικής για να προβλέψει αυτούς τους κλάδους σε έναν κύκλο είναι μειωμένη (eliminate). Για αυτούς τους λόγους , η γρήγορη και ακριβής πρόβλεψη διακλαδώσεων γίνεται εξαιρετικά σημαντική. Λαμβάνοντας υπόψη τη τρέχουσα κατάσταση της βιομηχανίας η πρόβλεψη διακλαδώσεων είναι ο αριθμός bottleneck σε ένα ILP σήμερα.

Διάφοροι παράγοντες επηρεάζουν πόσο καλά οι κλάδοι μπορούν να προβλεφθούν: πόσο συχνά λαμβάνονται, (taken) και εάν οι κοντινοί κλάδοι συσχετίζονται ή όχι. Αυτοί οι παράγοντες καθορίζουν ποιές ανταλλαγές είναι σημαντικές για μια δεδομένη μηχανή πρόβλεψης διακλαδώσεων, και με την κατανόηση αυτών των παραγόντων μπορούμε να σχεδιάσουμε τους branch predictors για τις συγκεκριμένες κατηγορίες προβλημάτων.

Οι τωρινές εργασίες στη πρόβλεψη διακλαδώσεων εστιάζουν στη χρήση των μετρητών bits (bit counters). Χειρίζοντας αυτούς τους μετρητές και ενσωματώνοντας τις ιδέες που περιλαμβάνουν την ιστορία πρόβλεψης, την επικόλληση διευθύνσεων, και το συσχετισμό κλάδων οδήγησαν στο σχέδιο των branch predictors. Εκτός από αυτά τα απλά πρότυπα, τα υβριδικά πρότυπα δημιουργήθηκαν με το συνδυασμό δύο ή περισσότερων από αυτά τα απλά πρότυπα. Αυτά τα υβριδικά πρότυπα συμπεριφέρονται καλύτερα από τα ενιαία πρότυπα. Εντούτοις, σε πρόσφατα ερευνητικά συμπεράσματα μια τεχνική που βασίζεται σε perceptron, ένα εποπτευμένο πρότυπο εκμάθησης, έχει ξεπεράσει τους προκατόχους του.

Μόλις η σύσταση για τις απλές προβλέψεις εγκαθιδρύθηκε με τους (two-bit counters) και perceptrons, η φυσική πρόοδος είναι να εφαρμοστεί ένα υβρίδιο που εκμεταλλεύεται το γεγονός ότι οι δυνάμεις μερικών προσεγγίσεων θα μπορούσαν να καλύψουν αδυναμίες άλλων. Κατά συνέπεια, πιστεύουμε ότι αυτά τα υβριδικά σχέδια πρόβλεψης θα ξεπεράσουν τις υπάρχουσες τεχνικές βασισμένες σε ένα πλαίσιο ή ένα άλλο. Έχουμε εφεύρει τέσσερα τέτοια υβρίδια για να εξετάσουμε αυτήν την υπόθεση, Αυτά περιλαμβάνουν

- 2bc-pgskew
- 2bc-ppskew
- Weighted Perceptron (Σταθμισμένο Perceptron)
- Meta Perceptron

Στην παράγραφο 2, παρουσιάζουμε αρκετές από τις συνηθέστερα εφαρμοσμένες προβλέψεις βασισμένες στους two-bit counters. Εισάγουμε επίσης τα κύρια προβλήματα που τα διακλαδιζόμενα σχέδια πρόβλεψης πρέπει να

υπερνικήσουν για να είναι αποτελεσματικά και πώς κάθε ένας από αυτούς τους μετρητές bit προσπαθούν να αντιμετωπίσουν αυτά τα προβλήματα. Στην παράγραφο 3, εισάγουμε την ιδέα της χρησιμοποίησης perceptron για να κάνουμε δυναμική πρόβλεψη κλάδων. Το υλικό αναφοράς perceptron, φαίνεται στο παράρτημα Α.

1.1 Το πρόβλημα: Καθυστέρηση στους Branch Predictors

Παίρνει την εργασία για να προβλέψει ακριβώς τους κλάδους. Το χρονικό διάστημα που είναι διαθέσιμο για να κάνει αυτήν την εργασία ασκεί μεγάλη επίδραση στην ακρίβεια του branch predictor. Η καθυστέρηση πρόσβασης στον branch predictor είναι ένα κρίσιμο συστατικό στον καθορισμό της απόδοσης του επεξεργαστή, δεδομένου ότι ασκεί επίδραση και στη συχνότητα του ρολογιού (clock rate) και στη ρυθμαπόδοση εντολών. Αυτή η καθυστέρηση επηρεάζεται από τις τάσεις στην τεχνολογία. Σε αυτό το τμήμα, εξηγούμε την πηγή καθυστέρησης των branch predictor και τις συνέπειες αδιαφορίας της καθυστέρησης.

Οι σύγχρονοι branch predictors είναι βασισμένοι στη δύο επιπέδων προσαρμοστική τεχνική πρόβλεψης κλάδων (two-level adaptive branch prediction) που παρουσιάσθηκε από τον Yeh και τον Patt [18]. Αυτό το σχέδιο χρησιμοποιεί έναν πίνακα από μετρητές για να βρει τους συσχετισμούς μεταξύ των προηγούμενων εκβάσεων διακλαδωτών για να κάνει μια πρόβλεψη. Για να είναι ο branch predictor ακριβής, αυτός ο πίνακας πρέπει να έχει εκατοντάδες ή χιλιάδες καταχωρήσεις, αναγκάζοντας τον να μοιάζει με μια μικρή κρυφή μνήμη. Στις προηγούμενες τρεις δεκαετίες, οι βελτιώσεις απόδοσης στους μικροεπεξεργαστές έχουν οδηγηθεί σε μεγάλο μέρος από βελτιώσεις στη τεχνολογική διαδικασία, δηλαδή, τη διαδικασία με την οποία οι μικροεπεξεργαστές κατασκευάζονται από ταινίες (wafers) του πυριτίου. Καθώς η τεχνολογία βελτιώνεται, τα μεγέθη και οι καθυστερήσεις των transistors και των καλωδίων σε μια μείωση μικροεπεξεργαστών, επιτρέπει στους αρχιτέκτονες υπολογιστών να συμπιέσουν περισσότερη λειτουργία επάνω στο τσιπ, και να τρέξουν το τσιπ σε μια υψηλότερη συχνότητα ρολογιού. Πρόσφατες μελέτες, εντούτοις, έχουν δείξει ότι καθώς τα χαρακτηριστικά μεγέθη έχουν συρρικνωθεί στην τρέχουσα και μελλοντική τεχνολογική διαδικασία, αυξάνονται όλο και περισσότερο επιθετικά τα ποσοστά ρολογιών και οι μεγαλύτερες καθυστερήσεις καλωδίων θα οδηγήσουν στην πρόσβαση πολυ-κύκλων (multi-cycle) στις μεγάλες δομές των chip [19] όπως οι γρήγορες μνήμες (cache) και οι branch predictor.

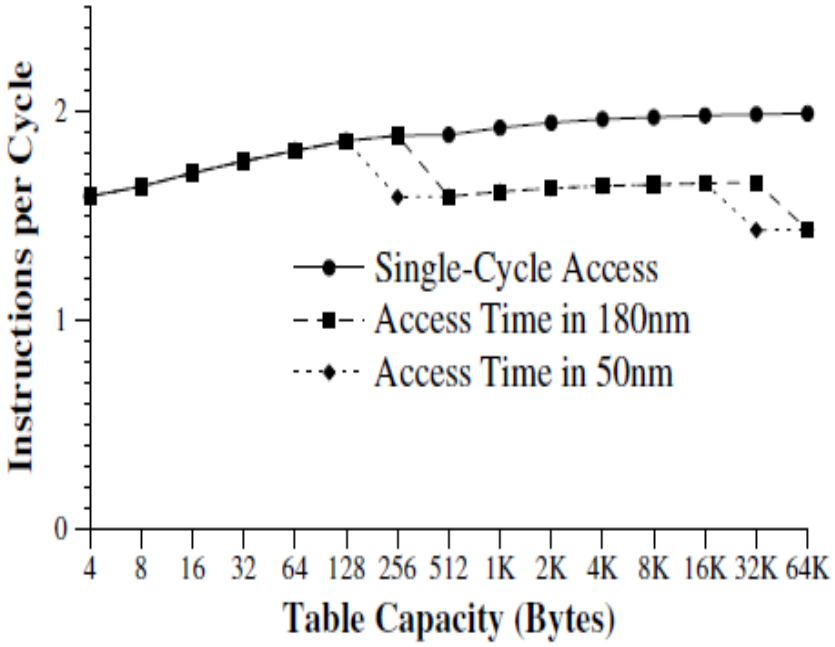
Μέχρι τώρα, το τεράστιο θέμα της πρόβλεψης διακλαδώσεων έχει εστιάσει σε μόνο δύο διαστάσεις του προβλήματος.

- της περιοχής (coverage) και
- της ακρίβειας (accuracy)

και έχει διαπιστώσει ότι οι μεγαλύτεροι προϋπολογισμοί υλικού παράγουν την υψηλότερη ακρίβεια για δύο λόγους: Επιτρέπουν τη χρήση μιας πιο longer history lengths, και μειώνουν την αντανάκλαση, (aliasing) το οποίο συμβαίνει όταν δυο μη σχετικοί διακλαδωτές καταστροφικά μοιράζονται τους ίδιους πόρους πρόβλεψης διακλαδωτών υλικού. Πράγματι, ένα μεγάλο μέρος της πρόσφατης εργασίας έχει εστιάσει στις μεθόδους για μείωση της αντανάκλασης [20,21,22,23,24]. Με την αύξηση της χωρητικότητας των chip, η εστίαση της ερευνητικής κοινότητας στην περιοχή και την ακρίβεια έχει οδηγήσει στους μεγάλους επιμελημένους predictors, μερικοί από τους οποίους απαιτούν από 16K μέχρι 64K byte [25]και η σύνθετη

πρόβλεψη σχεδίων προσθέτει επίπεδα λογικής στην καταστρεπτική αντανάκλαση (aliasing) [22,24].

Δεδομένου ότι οι δυναμικοί branch predictors χρησιμοποιούν μεγάλους πίνακες για να βρουν συσχετισμούς και να κάνουν προβλέψεις, οι μελλοντικοί branch predictors θα πρέπει να εξετάσουν μια τρίτη διάσταση : τη καθυστέρηση. Χρησιμοποιώντας μια εξιδανικευμένη καθυστέρηση δηλαδή έναν κύκλο για να έχει πρόσβαση στον πίνακα ιστορίας πρότυπων pattern history table (PHT), ο gshare predictor [21] εμφανίζει βελτιωμένες εντολές ανά κύκλο (IPC) λόγω της βελτιωμένης ακρίβειας πρόβλεψης καθώς το μέγεθος του πίνακα ιστορίας πρότυπων (PHT) αυξάνεται. Σε αντίθεση, με μια επιθετική συχνότητα ρολογιών (1.9 Ghz) και ένα ρεαλιστικό πρότυπο καθυστέρησης για τη σημερινή τεχνολογία των 180 νανόμετρων, οι πτώσεις καμπυλών πέφτουν στα 512 bytes όπου το PHT απαιτεί δύο κύκλους για να έχει πρόσβαση, και πέφτει πάλι στα 64KB όπου η καθυστέρηση γίνεται τρεις κύκλοι. Αυτό το πρόβλημα θα επιδεινωθεί από τις μικρότερες τεχνολογικές διαδικασίες του μέλλοντος, όπως παρουσιάζεται από την καμπύλη για την τεχνολογία 50nm και ένα προβαλλόμενο ποσοστό ρολογιών στα 6.9 Ghz. Σε αυτήν την τεχνολογία, η καθυστέρηση καλωδίων προκαλεί καθυστέρηση στο χρόνο πρόσβασης στο πίνακα επάνω από έναν κύκλο ακόμα νωρίτερα.



Σχήμα 1.1: Ρυθμαπόδοση εντολών εναντίον της ικανότητας για τον gshare predictor

Για παράδειγμα μιας πραγματικής περίπτωσης αυτού του προβλήματος, ο branch predictor για το μικροεπεξεργαστή AMD Athlon αντιπροσωπεύει ένα βήμα πίσω σε σύγκριση με τον προκάτοχό του, τον K6. Ενώ ο K6 έχει έναν υπερβολικά ακριβή 8K-εισόδων, GAs predictor, ο Athlon χρησιμοποιεί έναν λιγότερο ακριβή 2K-εισόδων GAs predictor [26]. Αυτή η αλλαγή μειώνει τη καθυστέρηση και το κόστος ακίνητης περιουσίας (real estate costs) του branch predictor και θα μπορούσε να είναι ένας

λόγος για τον οποίο ο Athlon είναι σε θέση να επιτύχει ένα επιθετικό ποσοστό ρολογιών της τάξεως του 1.4 Ghz. Εντούτοις, ο Athlon έχει μειώσει την απόδοση από την άποψη του IPC λόγω του λιγότερο ακριβούς branch predictor.

Τα υψηλότερα ποσοστά ρολογιών αυξάνουν επίσης την ανάγκη για την υψηλότερη ακρίβεια πρόβλεψης κλάδων. Δεδομένου ότι οι διασωληνώσεις γίνονται βαθύτερες για να δημιουργήσουν τη λιγότερη εργασία ανά κύκλο, η ποινή μιας άστοχης πρόβλεψης (misprediction) αυξάνεται. Παραδείγματος χάριν, ο Pentium 4 έχει ένα misprediction 20 σταδίων διασωλήνωσης [27].

Εν συντομία, το πρόβλημα είναι αυτό: Η χρησιμοποίηση της μικρότερης πρόβλεψης κλάδων παρουσιάζει τα αποτελέσματα στη χαμηλότερη IPC λόγω της χαμηλότερης ακρίβειας. Είναι αφελές να χρησιμοποιούμε τους μεγαλύτερους πίνακες σε μελλοντικές τεχνολογίες αποτελεσμάτων σε ακόμα χαμηλότερες εντολές ανά κύκλο (IPC) λόγω των επιθετικών διαβαθμίσεων τάσεων ρολογιών και της αυξανόμενης σχετικής καθυστέρησης καλωδίων. Η ερώτηση είναι πώς μπορούμε να πάρουμε και την υψηλή ακρίβεια και τη χαμηλή λανθάνουσα κατάσταση;

1.2 Διαστάσεις του προβλήματος

Δεδομένου ότι τα σχέδια μικροαρχιτεκτονικής εξελίσσονται και η τεχνολογική διαδικασία βελτιώνεται, διάφορες διαστάσεις του προβλήματος καθυστέρησης branch predictor προκύπτουν. Σε αυτό το τμήμα, εξερευνούμε αυτές τις διαστάσεις, και υποβάλουμε σημαντικές ερωτήσεις για το μέλλον των branch predictors.

1.2.1 Επεκτείνοντας τους παραδοσιακούς predictors στο κοντινό μέλλον

Στο κοντινό μέλλον, δηλ. τα επόμενα λίγα χρόνια, θα επιθυμούσαμε να είμαστε σε θέση να συνεχίσουμε τους παραδοσιακούς branch predictors που έχουν παράσχει τέτοια καλή εκτέλεση στο παρελθόν. Οι πίνακες που είναι βασισμένοι στους branch predictors έχουν ερευνηθεί έντονα. Οι βιομηχανικοί και ακαδημαϊκοί ερευνητές έχουν πολύ καλές ιδέες για το πώς να πετύχουν υψηλή ακρίβεια χρησιμοποιώντας τις παραλλαγές των δύο επιπέδων προσαρμοστικών predictors; (ερευνάμε αρκετές από αυτές τις προσπάθειες στο κεφάλαιο 2). Εντούτοις, η εντυπωσιακή απόδοση αυτών των predictors έρχεται με το κόστος της υψηλής καθυστέρησης πρόσβασης. Δεδομένου ότι οι σωληνώσεις εμβαθύνουν για να υποστηρίξουν τα όλο και περισσότερο επιθετικά ποσοστά ρολογιών (aggressive clock rates) στο κοντινό μέλλον, η βιωσιμότητα αυτών των σχεδίων απειλείται από την καθυστέρηση που επιβάλλουν. Δεν μπορούμε απλά να πετάξουμε μακριά αυτά τα σχέδια χωρίς να κατέχουμε κάτι για να τα αντικαταστήσουμε. Κατά συνέπεια, είμαστε ιδιαίτερα – παρακινημένοι για να βρούμε τρόπους γύρω από το πρόβλημα καθυστέρησης, έτσι ώστε να μπορούμε να επεκτείνουμε τη χρησιμότητα αυτών των predictors στο μέλλον και να συνεχίσουμε να χτίζουμε τους παραδοσιακούς πυρήνες με τις βαθύτερες σωληνώσεις και τα υψηλότερα ποσοστά ρολογιών.

1.2.2 Αυξανόμενη ακρίβεια παρά την καθυστέρηση

Η στήριξη των παραδοσιακών branch predictors δεν είναι ικανοποιητική, ειδικά από τότε που η αστοχία πρόβλεψης (mispredictions) άρχισε να γίνεται δαπανηρότερη.

Πώς μπορούμε όμως να καταστήσουμε τον branch predictor ακριβέστερο εάν έχει το λιγότερο χρόνο; Όπως έχουμε σημειώσει, μια έρευνα μεγάλου ποσού έχει γίνει για να βελτιώσει την ακρίβεια των βασισμένων σε πίνακα (table-based) branch predictor. Εντούτοις, πιστεύουμε ότι αυτή η ερευνητική προσπάθεια έχει τελειώσει. Πιστεύουμε ότι υπάρχουν πολλές περισσότερες ιδέες ακόμα που ανακαλύπτονται. Πράγματι, εισάγουμε μια τέτοια δική μας τεχνική στο κεφάλαιο 5. Πώς μπορούν οι υπερβολικά ακριβείς predictor με υψηλή πρόσβαση στις καθυστερήσεις να χρησιμοποιούνται σε επεξεργαστές με τις πολύ μικρές χρονικές περιόδους ρολογιών; Ομοίως, υπάρχουν τρόποι να χρησιμοποιηθούν οι table-based predictors που οδηγούν στην υψηλή ακρίβεια αλλά είναι πιο οικονομικοί με το χρόνο τους;

1.2.3 Εξέταση της διαβάθμισης τεχνολογίας

Καθώς τα όρια της CMOS διεργασίας τεχνολογίας προσεγγίζονται στην επόμενη δεκαετία, η καθυστέρηση και η δύναμη καλωδίων θα γίνουν κυρίαρχες δυνάμεις που διαμορφώνουν το σχέδιο επεξεργαστών. Λόγω της καθυστέρησης καλωδίων, ο χρόνος που χρειάζεται για να έχει πρόσβαση σε έναν εύλογα μεγάλο branch predictor μπορεί να είναι ένα σημαντικό μέρος του χρόνου που παίρνει μια εντολή για να διαβεί τη διασωλήνωση. Κατά συνέπεια, οι παραδοσιακοί table-based branch predictors μπορούν να γίνουν ανέφικτοι ή ακόμα και άχρηστοι σε αυτήν την νέα ρύθμιση, και θα αναγκαστούμε να ψάξουμε για κάτι νέο. Υπάρχουν τρόποι να προβλεφθούν ακριβώς οι κλάδοι χωρίς τους πίνακες ή άλλα ακριβά συστατικά;

Microprocessor Integer	Integer Pipeline Depth	Clock Frequency (MHz)
PowerPC 7400	4	733
HP PA-8700	7	800
Alpha 21264	7	833
AMD Athlon	9	1400
Intel Pentium 4	20	1760

Πίνακας 1.1: Βάθος σωληνώσεων εναντίον της συχνότητας ρολογιών για τους διάφορους επεξεργαστές.

1.3 Οι λύσεις μας

Η λύση μας στο πρόβλημα της καθυστέρησης στους branch predictors είναι να διαιρέσουμε την εργασία πρόβλεψης σε μέρη με διαφορετικές καθυστερήσεις. Κατά τη διάρκεια ενός μέρους της πρόβλεψης, ένας γρήγορος branch predictor αναπτύσσει δραστηριότητες σε έναν ενιαίο κύκλο. Κατά τη διάρκεια ενός άλλου μέρους της πρόβλεψης, είτε offline μέσω profiling , είτε on-line μέσω του υλικού, ξοδεύεται περισσότερος χρόνος στο να καταστήσει τον predictor ακριβέστερο. Ερευνάμε δύο κύριες τεχνικές για την διαίρεση της εργασίας:

Ιεραρχικοί Predictors Προτείνουμε τις ιεραρχικές οργανώσεις για τους branch predictors. Περιγράφουμε τρεις οργανώσεις branch predictor κάθε μια με το κοινό στόχο να συνδυάσει έναν γρήγορο predictor με έναν πιο αργό αλλά ακριβέστερο predictor για να επιτύχουμε την ακριβή πρόβλεψη σε έναν ενιαίο κύκλο.

Εφαρμόζουμε αυτές τις ιδέες σε δύο πλαίσια:

- Καταδεικνύουμε πώς αυτές οι τεχνικές μπορούν να εφαρμοστούν στους συμβατικούς predictors η καθυστέρηση των οποίων προέρχεται από το χρόνο πρόσβασης στον πίνακα Κατά συνέπεια, επιδεικνύουμε πώς οι παραδοσιακοί predictor μπορούν να επεκταθούν στα επόμενα χρόνια της διαβάθμισης ρολογιού και της βελτίωσης της τεχνολογίας.
- Ερευνάμε το διάστημα των πιο σύνθετων predictors που ειδάλλως θα ήταν ανέφικτοι λόγω της καθυστέρησης: περιγράφουμε έναν νέο branch predictor βασισμένο σε μια νευρωνική τεχνική εκμάθησης. Αυτός ο perceptron predictor έχει μοναδικές ιδιότητες που του επιτρέπουν να παραγάγει με υψηλή ακρίβεια. Χρησιμοποιώντας τις τεχνικές που περιγράφονται ανωτέρω, αυτός ο περίπλοκος multi-cycle predictor μπορεί να χρησιμοποιηθεί ως συστατικό ενός γρήγορου ευαίσθητου στη καθυστέρηση (delay-sensitive) predictor. Κατά συνέπεια, δείχνουμε ότι πάντα οι πιο σύνθετοι και ακριβείς predictors είναι ακόμα εφικτοί, ακόμη και παρά το πρόβλημα της καθυστέρησης στους branch predictor.

Συνεταιριστικοί Predictors Ένας άλλος τρόπος να ανεχτούν τη καθυστέρηση είναι να ξεφορτωθούν μερικές από τις εργασίες πρόβλεψης στο μεταγλωττιστή, με τη σκιαγράφηση (profiling). Κατά αυτόν τον τρόπο, ο μεταγλωττιστής και το υλικό συνεργάζονται για να παραγάγουν την πρόβλεψη. Η εργασία πρόβλεψης πραγματοποιείται σε δύο στάδια: Κατ' αρχάς, ένας off-line profiling αλγόριθμος αναλύει τη συμπεριφορά του προγράμματος σε μια εισαγωγή κατάρτισης. Ο μεταγλωττιστής διαβιβάζει τις σχεδιασμένες πληροφορίες στο μικροεπεξεργαστή χρησιμοποιώντας τις επεκτάσεις στην αρχιτεκτονική συνόλου οδηγίας (Instruction Set Architecture ISA), προσδιορίζοντας πώς να εκτελέσει την πρόβλεψη διακλαδώσεων με την υψηλή ακρίβεια. Δεύτερον, το σύνολο οδηγίας διαβιβάζει τους υπαινιγμούς στον branch predictor στο τρέχον πρόγραμμα έτσι ώστε η πρόβλεψη να είναι γρήγορη. Περιγράφουμε δύο νέες τεχνικές:

- Το μονοπάτι διακλάδωσης ξανά-αντανακλά (re-aliasing), μια τεχνική που κινεί την πολυπλοκότητα μακριά, της κρίσιμης πορείας για την παραγωγή μιας πρόβλεψης μέσα στο μεταγλωττιστή. Αυτή η τεχνική αυξάνει την ακρίβεια με τη μείωση του καταστρεπτικού aliasing κατά τη διάρκεια του λιγότερο κρίσιμου σταδίου των αναπροσαρμογών. Αυτή η τεχνική επιτρέπει σε μας να μειώσουμε την καθυστέρηση των branch predictor με τη συρρίκνωση ενός branch predictor από μια γενεά στην επόμενη χωρίς να θυσιάσουμε την ακρίβεια.

Αυτή η τεχνική είναι συγκεκριμένη για μια ιδιαίτερη οικογένεια των branch predictors.

● Ένας branch predictor στον οποίο μια φάση σκιαγράφησης (profiling phase) βρίσκεται μια λειτουργία, χρησιμοποιείται για να εκτελέσει την πρόβλεψη διακλαδώσεων για κάθε κλάδο. Κάθε λειτουργία κωδικοποιείται ως τύπος Boolean στην διακλάδωση εντολών. Το PHT αποβάλλεται, έτσι ώστε να μην είναι άλλο πια μια πηγή καθυστέρησης. Αυτός ο τύπος Boolean predictor έχει μια μικρή και γρήγορη εφαρμογή υλικού και θα εργαστεί σε λιγότερο από έναν κύκλο ακόμη και στις μικρές τεχνολογίες και τα επιθετικά ποσοστά ρολογιών για τις οποίες οι συμβατικοί table-based predictors είναι ανέφικτοι.

1.3.1 Πεδίο και περιορισμοί της έρευνας

Σε αυτήν την διατριβή, εστιάζουμε κυρίως στα αποτελέσματα της τεχνολογίας που διαβαθμίζονται πάνω στην κατεύθυνση πρόβλεψης διακλαδώσεων. Κατά συνέπεια, μελετάμε κυρίως την κατεύθυνση branch prediction στην απομόνωση, υποθέτοντας χάριν της απλότητας ότι άλλες μικροαρχιτεκτονικές δομές δεν επηρεάζονται με τη διαβάθμιση της τεχνολογίας. Εν συντομία εξετάζουμε άλλες σχετικές πτυχές της μικροαρχιτεκτονικής, όπως είναι η καθυστέρηση target buffer κλάδων η καθυστέρηση κρυφών εντολών (instruction cache delay), και η δύναμη του branch predictor και δεν προτείνουμε τις πλήρεις λύσεις σε αυτά τα προβλήματα.

Αυτή η μεθοδολογία επιτρέπει σε μας να κάνουμε ισχυρότερες δηλώσεις για το μέλλον των branch predictors χωρίς στήριξη στις προβλέψεις άλλων συστατικών εντούτοις, χωρίς να λάβει υπόψη αυτά τα άλλα συστατικά, είναι δυσκολότερο να ερμηνευθούν οι εξομειούμενοι αριθμοί απόδοσής μας. Είναι σημαντικό να σημειωθεί ότι άλλα προβλήματα μπορούν να διαμορφώσουν πιο σημαντικά bottlenecks, όπως είναι η αυξανόμενη διαφορά μεταξύ των ταχυτήτων DRAM και KME, και ότι τα αποτελέσματα της IPC μπορούν να είναι αισιόδοξα υποθέτοντας ότι αυτά τα προβλήματα δεν θα γίνουν καθόλου χειρότερα.

2 Ιστορία της πρόβλεψης διακλαδώσεων

Η πρόβλεψη διακλαδώσεων έρχεται με δύο διαφορετικές μορφές: στατική και δυναμική. Μερικοί από την Reduced Instruction Set Computers χρησιμοποιούν την τετριμμένη στατική πρόβλεψη διακλαδώσεων για τη βελτίωση της ρυθμαπόδοσης (throughput) του προγράμματος. Αυτές οι αρχιτεκτονικές χρησιμοποιούν έναν στατικό not taken branch predictor, ο οποίος προσκομίζει την επόμενη διαδοχική οδηγία υποθέτοντας ότι ο κλάδος δεν θα ληφθεί (not taken). Άλλη προφανής εναλλακτική λύση είναι να προβλέψει με συνέπεια ότι θα παρθεί (taken). Και τα δύο αυτά σχέδια αποτυγχάνουν να προβλέψουν τους κλάδους με ένα ποσοστό ακρίβειας αποδεκτό για τους υπάρχοντες επεξεργαστές, που συχνά αποδίδουν στην ακρίβεια μόνο 50-60% [2].

Μια πιο περίπλοκη στατική μέθοδος πρόβλεψης διακλαδώσεων αποκαλούμενη Backward Taken Forward Not Taken (BTFNT), προβλέπει ότι οι κλάδοι με τους στόχους κλάδων που βρίσκονται πριν από τον κλάδο θα ληφθούν, ειδάλλως δεν θα ληφθούν. Οι ληφθέντες προβλέψεις λειτουργούν καλύτερα για βρόγχους επειδή όλες εκτός από μια από τις προβλέψεις (τη τελευταία) θα είναι σωστή. Οι μπροστινοί κλάδοι συνήθως προσδιορίζουν κάποιο τύπο εξαίρεσης που προκαλεί ένα πρόγραμμα για να πηδήσει από έναν φραγμό κώδικα (code block) νωρίς.

Αυτές οι καταστάσεις συμβαίνουν σπάνια, και έτσι ένας not taken predictor είναι εντάξει. Επιπλέον ο μεταγλωττιστής μπορεί να χρησιμοποιήσει μια σκιαγράφηση τεχνικών για να παραχθούν οι οδηγίες κλάδων έτσι ώστε η συνηθέστερα ληφθείσα πορεία ελέγχου να είναι πάντα στη ληφθείσα πορεία. Κατά συνέπεια, ο BTFNT διακλαδωτής συνδυάζει τις δυνάμεις και των δύο τετριμμένων predictors. Ενώ οι στατικοί predictors είναι πολύ απλό να εφαρμοστούν, τα οφέλη απόδοσης από αυτά τα σχέδια είναι πολύ χλωμά σε σύγκριση με τα δυναμικά σχέδια που εισάγουμε. Εντούτοις, όταν καμία δυναμική πληροφορία πρόβλεψης δεν είναι διαθέσιμη ακόμα, πολλοί δυναμικοί predictors προκαθορίζουν (default back) τα στατικά σχέδια.

Οι δυναμικοί branch predictors χρησιμοποιούν τα προηγούμενα αποτελέσματα κλάδων για να αυξήσουν την ακρίβεια πρόβλεψης κλάδων. Με τη χρησιμοποίηση της διεύθυνσης του κλάδου, που συνδέεται με μια ιστορία των πρόσφατων προβλέψεων, ένας branch predictor μπορεί ακριβώς να προβλέψει τις εκβάσεις κλάδων. Η προϋπόθεση αυτών των μεθόδων είναι ότι ένα σχέδιο που θα προκύψει μπορεί να βοηθήσει στο μέλλον τις προβλέψεις. Παρουσιάζουμε μια επιλογή πολλών διαφορετικών σχεδίων πρόβλεψης διακλαδώσεων κατά σειρά αυξανόμενης πολυπλοκότητας.

Υπάρχουν τρεις διαφορετικές κατηγορίες branch predictors: τοπικός (local), σφαιρικός (global), και ο συνδυασμός τοπικού και σφαιρικού. Οι τοπικές εργασίες πρόβλεψης διακλαδώσεων λειτουργούν καλά για την επαναλαμβανόμενη συμπεριφορά όπως βρόχοι, επειδή ένας συγκεκριμένος βρόχος θα συμπεριφερθεί συνήθως με τον ίδιο τρόπο κάθε φορά που εκτελείται. Από την άλλη πλευρά, η σφαιρική πρόβλεψη είναι σημαντική στην κατανόηση του συσχετισμού μεταξύ των κλάδων μέσα στη στενή εγγύτητα, επειδή οι κλάδοι που είναι κοντά ο ένας στον άλλον συμπεριφέρονται συχνά ομοίως κάθε φορά που ο φραγμός (block) εκτελείται.

2.1 One-Bit Counters

Ένας απλός one-bit counter [4] είναι ο απλούστερος δυναμικός predictor. Αυτή η μέθοδος λειτουργεί υπό την υπόθεση ότι οποιοσδήποτε κλάδος δοθεί σε ένα πρόγραμμα, είναι καλύτερο να μιμηθεί την έκβαση για τον πιο πρόσφατο κλάδο. Παραδείγματος χάριν, αυτός ο τύπος predictor λείπει μόνο δύο φορές για οποιοδήποτε δεδομένο βρόχο, μια φορά στην αρχή ενός βρόχου και μια φορά στο τέλος. Εντούτοις, η παρουσία άλλων διακλαδωτών μέσα στο βρόχο θα μπορούσε να είναι καταστρεπτική. Υποθέστε ότι υπάρχει ένας κλάδος μέσα στο βρόχο που πολύ σπάνια λαμβάνεται (taken). Κατόπιν κάθε φορά που εισάγεται ο βρόχος, το bit ιστορικού θα μπορούσε να τεθεί σε ένα,"1" επειδή ο κλάδος βρόχων λήφθηκε, και ο predictor θα υπέθετε ότι ο κλάδος μέσα στο βρόχο πρέπει να ληφθεί. Εντούτοις, δεν λαμβάνεται (non taken) , και το history bit τίθεται μηδέν. Τώρα ο επεξεργαστής φθάνει στο τέλος του βρόχου και το history bit τίθεται μηδέν "0" έτσι ο επεξεργαστής προβλέπει (non taken). Όπως κάποιος μπορεί να δει, αυτό το σχέδιο εναλλάσσεται (flip-flop) μέσω ολόκληρης της εκτέλεσης του βρόχου, με την ακρίβεια πρόβλεψης κοντά σε μηδέν.

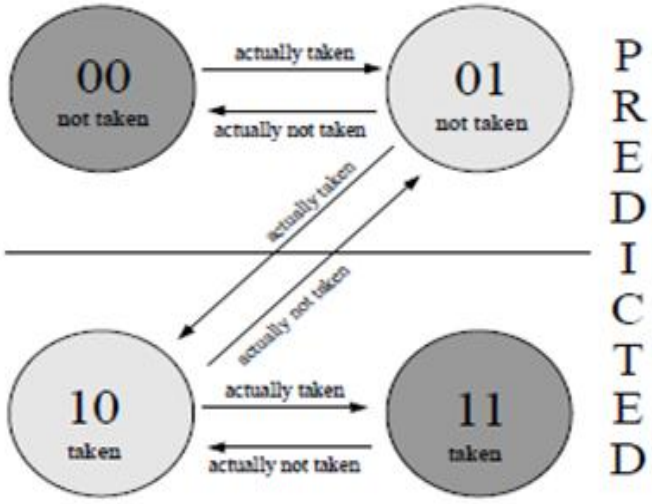
Μια επέκταση αυτής της ιδέας περιλαμβάνει ένα history bit για κάθε κλάδο στο πρόγραμμα [4]. Αυτό θα απέβαλλε το flip-flop πρόβλημα που περιγράφεται ανωτέρω. Εντούτοις, για τα πολύ μεγάλα προγράμματα αυτός ο πίνακας θα μπορούσε να είναι τεράστιος. Αντ' αυτού κάθε διεύθυνση χρησιμοποιείται για να κομματιαστεί (hash) σε έναν μικρότερο πίνακα ιστορίας. Είναι έπειτα σημαντικό ότι οι κατακερματισμένες λειτουργίες χωρίζουν τις διευθύνσεις που είναι αρκετά κοντά έτσι ώστε να μη συγκρούονται μέσα στο πίνακα. Σε περίπτωση που δύο κοντινοί κλάδοι συγκρούονται, ο τύπος ξυλοφορτώματος, συντριβής (thrashing) που φαίνεται στο ανωτέρω παράδειγμα δεν αποφεύγεται. Αυτή η μέθοδος είναι ένα παράδειγμα ενός τοπικού branch predictor, όπου οι μεμονωμένες πληροφορίες κλάδων χρησιμοποιούνται.

2.2 Two-Bit Counters

Οι Two-Bit Counters [4, 2, 1] είναι λογικές επεκτάσεις των μετρητών one-bit.

Το σχήμα 1.2 επεξηγεί αυτήν την μέθοδο.

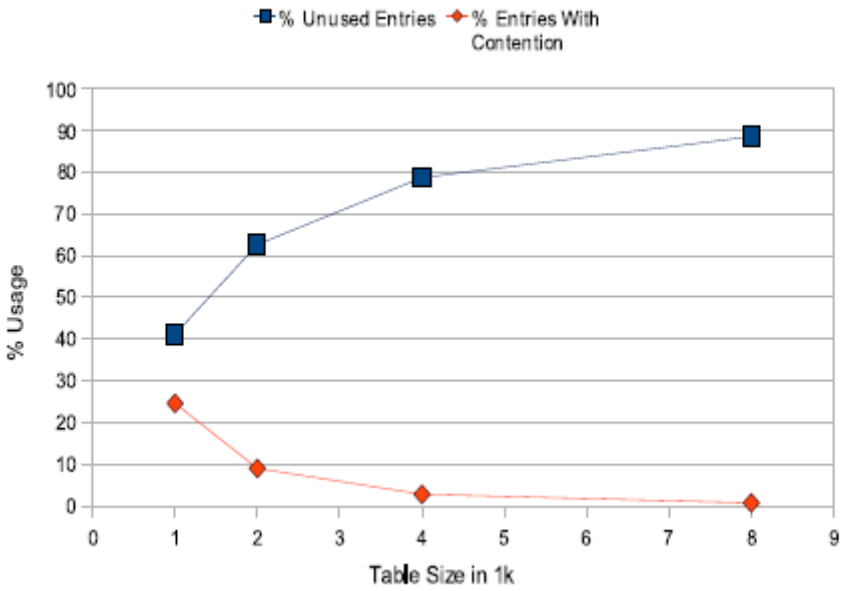
A finite state machine for the bi-mod predictor



Εάν ένας κλάδος λαμβάνεται (taken), ο predictor αυξάνει ένα μετρητή (που αρχικοποιείται σε 01). Υπάρχουν τέσσερις διαφορετικές καταστάσεις weakly taken, strongly taken, weakly not-taken, και strongly not taken. Η κατάσταση που ο predictor είναι αυτήν την περίοδο (κατά την διάρκεια που ο κλάδος) καθορίζει ποια θα είναι η πρόβλεψη. Κάθε πρόβλεψη προτρέπει μια αλλαγή στην κατάσταση που εξαρτάται από το εάν η πρόβλεψη ήταν σωστή. Εάν η πρόβλεψη ήταν σωστή, η κατάσταση γίνεται από weakly taken ή not-taken σε strongly taken ή not-taken. Αντιθέτως, οι ανακριβείς προβλέψεις κινούν τη πρόβλεψή μακριά από τη σωστή εκτίμηση. Χρησιμοποιώντας αυτό το σχέδιο, ο predictor χαλαρώνει την απάντησή του σε ανακριβείς προβλέψεις λαμβάνοντας υπόψη δύο ανακριβείς προβλέψεις πριν υποθέσει την αντίθετη επιλογή. Η ειδοποίηση ότι αυτό χρησιμοποιεί έναν saturating counter όπου η χαμηλότερη τιμή μπορεί να είναι μηδέν και η μεγαλύτερη τιμή περιορίζεται σε τρία.

2.3 Ο Bi-mod Predictor

Ο Bi-mod Predictor [2, 4] είναι ένα παράδειγμα ενός τοπικού σχεδίου πρόβλεψης που χρησιμοποιεί έναν πίνακα μετρητή two-bit. Ο πίνακας συντάσσεται από τη διεύθυνση του κλάδου, που καθιστά κάθε είσοδο τοπική σε εκείνο τον ιδιαίτερο κλάδο. Εντούτοις, οι συγκρούσεις υπάρχουν εάν δύο διευθύνσεις κατακερματίζονται (hash) στο ίδιο σημείο του πίνακα. Αυτό το πρόβλημα αναφέρεται ως αντανάκλαση (aliasing) και θα συζητηθεί λεπτομερέστερα αργότερα δεδομένου ότι είναι μια κύρια πηγή προβλημάτων στην πρόβλεψη κλάδων. Τέτοιοι predictors είναι περισσότερο ακριβής από τον one bit predictor για τους βρόχους, δεδομένου ότι λείπουν μόνο μια φορά ανά βρόχο εάν αρχικοποιείται σε weakly taken κατάσταση.



Σχήμα 2.3: Average Bimodal Predictor Table Usage

2.4 Two-Level Adaptive Predictor

Ένα άλλο παράδειγμα ενός τοπικού σχεδίου πρόβλεψης είναι μια πιο περίπλοκη έκδοση του bi-mod predictor. Αυτό το σχέδιο χρησιμοποιεί δύο επίπεδα πρόβλεψης. Ο πρώτος πίνακας επιπέδων, χρησιμοποίει το αρχείο διακλάδωσης διευθύνσεων, K bit σε κάποιο αριθμό branch history καταχωρητών όπου κάθε νέο bit μετατοπίζει το λιγότερο πρόσφατο bit. Ο δεύτερος πίνακας επιπέδων, ο πίνακας ιστορίας σχεδίων (PHT), συντάσσεται βασισμένος στο πρώτο σχέδιο ιστορίας διακλαδώσεων επιπέδων, όπου κάθε είσοδος στο PHT έχει τον δικό της two-bit counter. Ο two-bit counter επιλέγεται βασισμένος σε αυτόν τον δείκτη, έπειτα

χρησιμοποιείται για να προβλέψει τον κλάδο, και έπειτα αυτός ο μετρητής ενημερώνεται αφότου καθορίζεται η έκβαση κλάδων.

2.5 GA: Πρόβλεψη βασισμένη στην ιστορία

Ένας Global Adaptive (GA) predictor [2, 1] είναι ένας σφαιρικός δύο επιπέδων προσαρμοστικός predictor. Αυτή η μέθοδος χρησιμοποιεί μόνο ένα branch history καταχωρητή για να καταγράψει την έκβαση όλων των κλάδων. Ο καταχωρητής ιστορίας κλάδων χρησιμοποιείται έπειτα ως δείκτης σε ένα PHT. Αυτή η τεχνική εκμεταλλεύεται τους συσχετισμούς μεταξύ του τρέχοντος κλάδου και των άλλων κλάδων στην ιστορία για να προβλέψει το αποτέλεσμα. Ψάχνει για τα επαναλαμβανόμενα σχέδια των σφαιρικών εκβάσεων κλάδων σε αντιδιαστολή με την παρακολούθηση των μεμονωμένων ιστοριών κλάδων.

2.6 PA: Πρόβλεψη βασισμένη στη διεύθυνση

Ένας predictor PA [2, 1] είναι ένας δύο επιπέδων προσαρμοστικός predictor ανά-διεύθυνση. Αυτή η μέθοδος χρησιμοποιεί ένα καταχωρητή ιστορίας κλάδων ανά κλάδο. Για έναν δεδομένο κλάδο, ο καταχωρητής ιστορίας κλάδων χρησιμοποιείται ως δείκτης σε ένα PHT ακριβώς όπως στον GA predictor. Ο επιλεγμένος predictor είναι τοπικός στον κλάδο και έτσι λειτουργεί πολύ καλά για τους κλάδους που επαναλαμβάνουν τον ίδιο τύπο συμπεριφοράς. Για παράδειγμα, εάν κάθε πέμπτος κλάδος δεν λαμβάνεται (non taken), και άλλες φορές λαμβάνεται taken, ο ανά διεύθυνση predictor θα απέδιδε καλά λόγω αυτής της επαναλαμβανόμενης συμπεριφοράς.

2.7 Gshare: Πρόβλεψη βασισμένη και στη διεύθυνση & στην ιστορία

Ο Gshare predictor [2, 1] είναι μια παραλλαγή του GA predictor και παρουσιάζεται στο (σχήμα 2). Ο Gshare παίρνει το σφαιρικό branch history καταχωρητή και τον περνά από μια XOR που κάνει για να υπολογίσει το δείκτη (index) στο PHT. Αυτή η μέθοδος λειτουργεί καλύτερα από ένα GA predictor και μειώνει την πιθανότητα ότι δύο ίδια σχέδια που παράγονται από τους διαφορετικούς φραγμούς του κώδικα θα συγκρουστούν στο PHT.

Αυτό επιτρέπει μια πιο ομοιόμορφη χρήση του table space από το bimodal predictor, ενώ ακόμα αρχικοποιεί την ενδεχόμενη σύγκρουση διακλαδώσεων στις χωριστές εισόδους. Ο βασικός συλλογισμός πίσω από το συνδυασμό του PC και της

σφαιρικής ιστορίας είναι να χρησιμοποιήσει και τις χρονικές και τις πρόσθετες ιδιότητες του κλάδου, μειώνοντας κατά συνέπεια το επιτραπέζιο μέγεθος (table size) που απαιτείται για να επιτύχει το ίδιο ποσοστό πρόβλεψης.

Παρομοίος με άλλους predictors, τα περισσότερα από τα οφέλη παράγονται στις πρώτες αυξήσεις του επιτραπέζιου μεγέθους. Ο ισχυρισμός, στην περίπτωση του gshare, αποβάλλεται γενικά από το ίδιο το σχέδιο ευρετηρίασης, το οποίο επηρεάζεται πολύ από τη σφαιρική ιστορία. Διαπιστώνουμε ότι αντ' αυτού η αύξηση επιτραπέζιου μεγέθους είναι μόνο χρήσιμη στη σύλληψη των συσχετισμών με τα ευρύτερα σύνολα εργασίας κλάδων δεδομένου ότι διαπιστώθηκε ότι ο ισχυρισμός στο PHT του gshare διαδίδεται ομοιόμορφα μεταξύ όλων των καταχωρήσεων με μια χαμηλή σταθερή απόκλιση.

Table 2.7: Gshare Miss Rates

Table size	Avg. Miss Rate	Std. Dev.
1K	6.76%	5.48
2K	6.12%	5.40
4K	5.77%	5.31
8K	5.66%	5.25

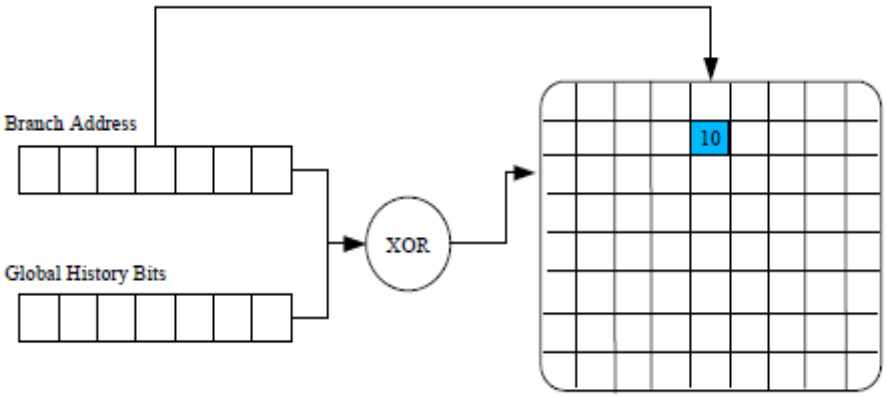
Δεδομένου ότι μερικοί κλάδοι είναι πολύ εύκολο να προβλεφθούν, είναι λογικό να εξετάσουμε επίσης την επίδραση της αλλαγής στο επιτραπέζιο μέγεθος στους κορυφαίους 10 χαμένους κλάδους. Ξέρουμε ότι συνολικά το χαμένο ποσοστό μειώνεται καθώς το επιτραπέζιο μέγεθος (table size) και το μήκος ιστορίας (history length) αυξάνεται εναλλακτικά θα εξετάσουμε τώρα σε ποιο κλάδο τα PC είναι παρόντα στους κορυφαίους 10 χαμένους κλάδους. Δύο διαφορετικές στατιστικές συλλέγονται για να επιτρέψουν μια κατάλληλη σύγκριση 1k, 2k, 4k, και 8k gshare διαμορφώσεων. Ο πρώτος ήταν μια αρίθμηση πόσων νέων κλάδων ήταν παρών και παρουσιάζονταν στους κορυφαίους 10 καταλόγους (lists) του gshare ενός μεγέθους N και 2N, δηλ. πόσοι κλάδοι αντικαταστάθηκαν με τους νέους κλάδους όταν διπλασιάστηκε το επιτραπέζιο μέγεθος. Ο δεύτερος ήταν μια αρίθμηση των πόσων κλάδων που μετατοπίστηκαν από την ταξινομημένη top 10 λίστα χάθηκαν. Είχαμε διαπιστώσει ότι κατά μέσο όρο υπάρχουν 3 νέοι κλάδοι (1.6, 1.0, 1.1)[3] στους κορυφαίους 10 κλάδους επίσης διαπιστώσαμε ότι κατά μέσον όρο οι κλάδοι μετατοπίζουν τη θέση τους στους κορυφαίους 10 κλάδους (4.1, 4.2, 3.4). Αυτές οι τιμές, μειώνουν τις απώλειες των κορυφαίων 10 κλάδων συνδυάζοντας το νέο κλάδο και μετατοπίζοντας την αρίθμηση κλάδων, αποκαλύπτουν ότι, γενικά οι κορυφαίοι χαμένοι κλάδοι είναι πιθανότερο να παραμείνουν ιδιαίτερα χαμένοι (missed) κατά την αύξηση του επιτραπέζιου μεγέθους.

Σε κάθε αύξηση του επιτραπέζιου μεγέθους, το μήκος ιστορίας αυξάνεται κατά 1 bit. Περαιτέρω, ο αριθμός κομματιών που χρησιμοποιούνται στο δείκτη στον πίνακα από το PC αυξάνεται επίσης κατά 1. Αυτά τα γεγονότα παρέχουν για 2 σχέδια συμπεριφοράς (patterns of behavior). Το πρώτο επιτρέπει στους κλάδους που χωρίζονται κατά διαστήματα να παρέχει την πρόβλεψη “hints” για τους τωρινούς προσκομισμένους κλάδους. Αυτό επεκτείνεται πέρα από τον ίδιο κλάδο που εκτελείται μέσα στις n εντολές για να αποτελέσει μια μερίδα του καταλόγου ιστορίας του gshare, εξομοιώνοντας κατά συνέπεια την τοπική πρόβλεψη στα μικρά σύνολα εργασίας.

……...

[3]Η μορφοποίηση (a, b, c) χρησιμοποιείται για να παρουσιάσει τα δεδομένα όπου a είναι μια τιμή που προέρχεται από τη σύγκριση του gshare table μεγέθους 1k and 2k,το b προέρχεται από table sizes 2k and 4k, and c is προέρχεται από table sizes 4k and 8k.

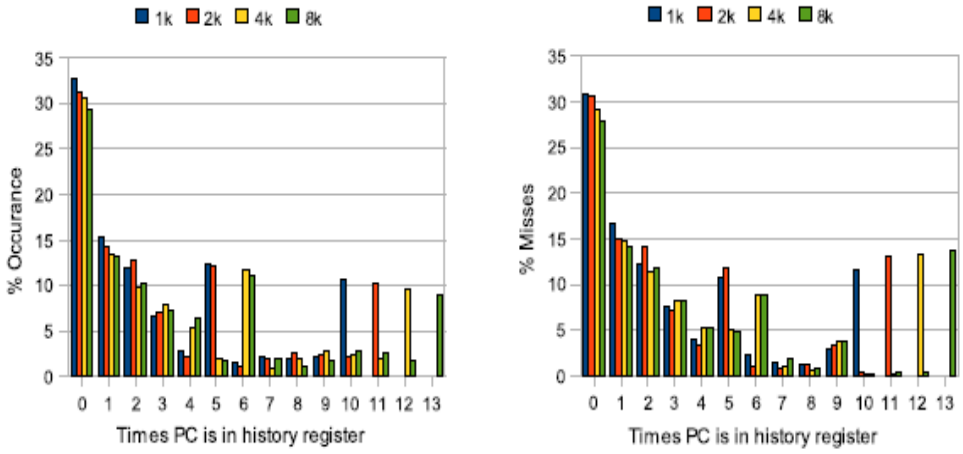
Σχήμα 2 ο Gshare predictor



Εκείνος ο ίδιος κλάδος «οδηγεί» τον predictor στη σωστή είσοδο όταν συνδυάζεται με το PC εκείνου του ίδιου κλάδου.

Ο δεύτερος επιτρέπει σε δύο κλάδους με συμπεριφοριστικά σχέδια (behavioral patterns) που μπορούν να είναι πάρα πολύ μακροχρόνια ή χάνονται λόγω του μεγάλου κλάδου που λειτουργεί, να εγκαθιδρύονται για να διακριθούν λόγω των πρόσθετων τοπικών bits (PC) που χρησιμοποιούνται για την αρχικοποίηση (indexing). Έπειτα, ερευνάμε την επίδραση του πλάτους καταλόγων ιστορίας. Το Gshare οργανώθηκε σε όλες τις συγκριτικές μετρήσεις επιδόσεων με μια μικρή τροποποίηση: κρατήσαμε μια ένωση του PC που συνέβαλε στον κατάλογο ιστορίας σε κάθε θεωρητική αναπροσαρμογή και το κομμάτι στον κατάλογο ιστορίας, (σχήμα 2.7. a.)

Σχήμα 2.7: Distribution of occurrences and misses of fetched PCs in gshare's global history register.



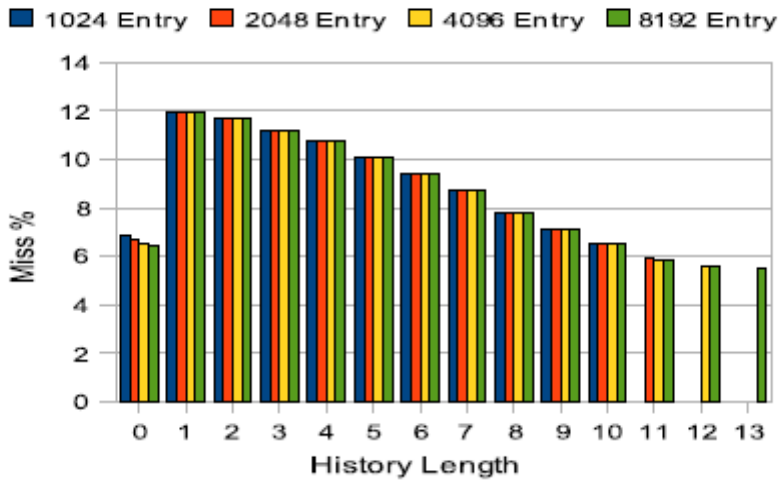
(a) Gshare PC Distribution (b) Gshare Miss Distribution

Όταν εξετάζονται, διαπιστώνουμε ότι κατά προσέγγιση 30% των προβλέψεων γίνονται όταν το τρέχον PC που προβλέπεται δεν είναι παρόν στο σφαιρικό κατάλογο ιστορίας. Περαιτέρω, βρίσκουμε επίσης ότι οι περισσότερες χαμένες εντολές εμφανίζονται στα άκρα, έχοντας είτε μηδέν περιπτώσεις ενός δεδομένου PC στον κατάλογο ιστορίας , είτε έχοντας συμπληρωμένο τον καταχωρητή με το δεδομένο PC (σχήμα 2.7 b). Παρατηρώντας το σχήμα 2.7.c , δύο ανωμαλίες είναι προφανείς. Το πρώτο και το ευκολότερο είναι ότι για κάθε history length, φαίνεται να υπάρχει κατά προσέγγιση 10% των προσβάσεων που έχουν κάθε bit του καταλόγου που προέρχεται από το PC που προβλέπεται. Αυτό προκαλείται από το πολύ σφιχτό, ενιαίο backward κλάδο που λαμβάνεται πάντα και το προσωρινά τοπικό κλάδο (temporal locality).

Η δεύτερη ανωμαλία είναι το 12% των προσβάσεων στη $5^{η}$ στήλη για 1k και 2k (10 και 11 μπιτ), και η $6^{η}$ στήλη για 4k και 8k (12 και 13 μπιτ). Αυτές οι ακίδες εμφανίζονται όταν δυο σφιχτά διακλαδιζόμενοι βρόχοι έχουν υψηλή προσβασιμότητα σε αυτόν τον βρόχο και κατά συνέπεια είναι πιθανό ότι ένας από αυτούς τους κλάδους πρέπει να είναι ένας πάντα ληφθείς προς τα πίσω (backward) αναγκάζοντας το βρόχο να το επαναλάβει αρκετές φορές ώστε να γίνει στάση των χαρακτηριστικών του και ο άλλος να μπορεί να έχει οποιοδήποτε χαρακτηριστικό καθώς και οι δυο αποτυγχάνουν και ο στόχος παραμένει μέσα έχοντας ο βρόχος ληφθεί ως strongly taken.

Περαιτέρω, εξερευνάμε πόση ιστορία πρέπει να διατηρηθεί. Ενώ θεωρείται ότι η πιο μακροχρόνια ιστορία παράγει την καλύτερη πρόβλεψη, καμία μελέτη δεν έχει γίνει για το gshare που αποκρυπτογραφεί ποια PC ή history, bits συμβάλλουν περισσότερο στο βελτιωμένο ποσοστό πρόβλεψης όταν διπλασιάζουμε το επιτραπέζιο μέγεθος (table size). Κάθε μια από τις (1k, 2k, 4k, και 8k) gshare διαμορφώσεις εκτελέσθηκε με όλα τα πιθανά πλάτη

καταλόγων ιστορίας και χάνει το ποσοστό που καταγράφηκε για κάθε μια στο σχήμα 2.7.c .



Σχήμα 2.7.c: Average miss rate of gshare predictor with variable history register width.

Η είσοδος 0 είναι αυτή που αντιστοιχεί σε καμία ιστορία (bimodal indexing scheme). Διαπιστώσαμε ότι *όταν* χρησιμοποι*ούμε* λιγότερα από 10 bits της ιστορίας με τις δεδομένες συγκριτικές μετρήσεις επιδόσεων, θα *ήταν* καλύτερα με τον bimodal predictor. Για κάθε bit *που προστέθηκε μετά από το* πρώτο, κατορθώσαμε να πάρουμε την καλύτερη πρόβλεψη. Όπως αναμένεται, η βελτίωση ανά bit έγινε όλο και λιγότερο σημαντική δεδομένου ότι περισσότερα bit ήταν ήδη παρόντα στο σφαιρικό κατάλογο ιστορίας. Η μετάβαση από το 12^{o} έως το 13^{o} bit στη διαμόρφωση 8k gshare μείωσε το ποσοστό αστοχίας μόνο κατά 0.08%, και μπορεί να αναμένεται ότι ένα άλλο bit σε έναν μεγαλύτερο πίνακα θα είχε αμελητέα μείωση ποσοστού χαμένων εντολών στις συγκριτικές μετρήσεις επιδόσεων.

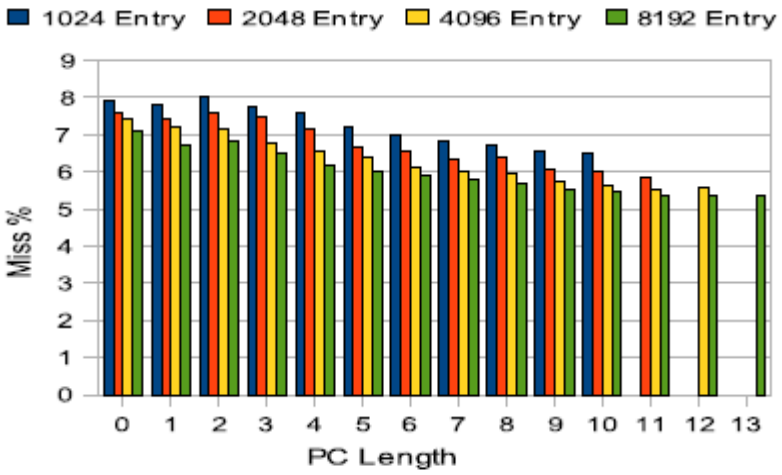


Figure 2.7.d: Average miss rate of gshare predictor with variable PC bits used for indexing.

Τέλος, εξετάζουμε τη σχετικότητα των PC bit που χρησιμοποιούνται στο gshare. Για αυτήν την μελέτη επιλέξαμε τη 8k είσοδο gshare με 13 bit της σφαιρικής ιστορίας. Κάθε μια γύρα που οργανώνεται θα αύξανε τον αριθμό PC bits που χρησιμοποιούνται στο δείκτη από 0 στα 13 bit.

Gshare

	1k	2k	4k	8k
Miss Rate:	24.75%	24.17%	23.35%	22.79%

2.8 Agree predictors

Ένας predictor κατεύθυνσης κλάδων μπορεί να ενισχύεται με τη χρησιμοποίηση του μηχανισμού συμφωνίας (agree mechanism) [28]. Παρά το συσχετισμό με την έκβαση κλάδων, οι είσοδοι PHT σε έναν predictor συμφωνίας παρακολουθούν εάν μια έκβαση κλάδων θα συμφωνήσει με ένα προκατειλημμένο κομμάτι (bias bit) που τίθεται στις εντολές διακλαδώσεων. Ο μηχανισμός συμφωνίας (agree mechanism) γίνεται καταστροφική παρέμβαση στην εποικοδομητική παρέμβαση, που αυξάνει την ακρίβεια. Εντούτοις, καθώς οι εντολές διακλάδωσης opcode πρέπει να διαβαστούν και να συνδυαστούν με την πρόβλεψη PHT, η κρυφή μνήμη (cache) είναι στην κρίσιμη πορεία για την πρόβλεψη κλάδων. Σημειώστε ότι οι προβλέψεις διακλαδώσεων μπορούν να μαθευτούν και να αποθηκευτούν σε έναν branch target buffer παρά να διακλαδιστούν ανά εντολή.

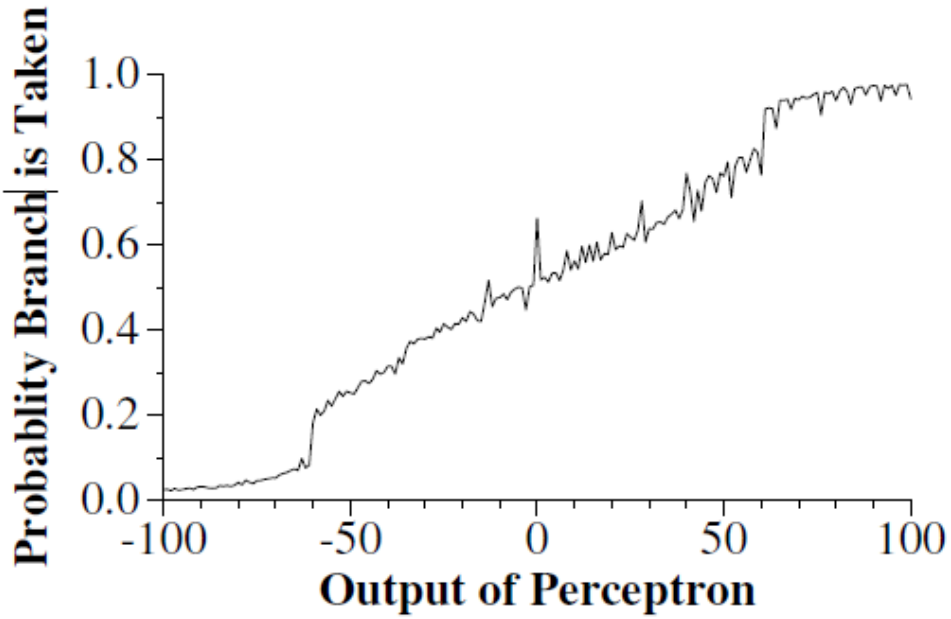
2.9 O Perceptron Predictor

Οι perceptron predictor κρατούν έναν πίνακα βάρους, που συντάσσεται από τη διεύθυνση κλάδων. Κάθε γραμμή κρατά τα βάρη που αντιστοιχούν στην εισαγωγή του απαραίτητου predictor στον υπολογισμό της εξόδου. Για να εφαρμόσει τον predictor στο chip, είναι σημαντικό να σημειωθεί ότι το προϊόν σημείων (dot product) των διανυσμάτων βάρους και ιστορίας θα υπολογίσει ένα ποσό των βαρών που επιλέγονται από τα κομμάτια ιστορίας. Για να αθροίσει αυτές τις τιμές παράλληλα, χρησιμοποιείται ένα δέντρο αθροιστών Wallace [8]. Η λειτουργία ενημέρωσης (update) μπορεί να εφαρμοστεί αργότερα στην διασωλήνωση, αφότου η συνθήκη διακλάδωσης έχει υπολογιστεί.

Η αρχική εργασία του Jimenez [7] πρότεινε μόνο τη σφαιρική (global) ιστορία για την εισαγωγή στον predictor. Αλλά ένας perceptron predictor μπορεί να πάρει τη σφαιρική ιστορία, την τοπική ιστορία, ή έναν συνδυασμό των δύο ως είσοδο. Ο perceptron είναι σε θέση να μάθει ποια bits είναι πιο πολύ σημαντικά για την πρόβλεψη και ανάπτυξη ενός υπερπλάνου (hyperplane) δηλαδή εκείνων των bits που συσχετίζονται αποτελεσματικά με τους κλάδους όταν ενημερώνεται ο perceptron. Αυτό το χαρακτηριστικό γνώρισμα επιτρέπει στο percepton να κάνει χρήση του μακροχρόνιου ιστορικού που χρησιμοποιείται από τους two bit counter perdictors. Αυτό είναι επειδή η ιστορία γίνεται είσοδος στο perceptron αντί να επιλέξει ένα μετρητή όπως γίνεται στον Gshare.

Η πρόβλεψη διακλαδώσεων με perceptrons έχει άλλα πλεονεκτήματα πέρα από τις προηγούμενες μεθόδους. Μια perceptron έξοδος μπορεί να δώσει μια εμπιστοσύνη στην πρόβλεψη. Το διάνυσμα βάρους μπορεί να χρησιμοποιηθεί για να βρει τους συσχετισμούς μεταξύ των κλάδων, έτσι αυτή η μέθοδος μπορεί να χρησιμοποιηθεί στην προσομοίωση για να αναλύσει τη συμπεριφορά ενός προγράμματος.

Ανάθεση της εμπιστοσύνης στις αποφάσεις. Ο predictor μας μπορεί να παρέχει ένα επίπεδο εμπιστοσύνης στις προβλέψεις του που μπορούν να είναι χρήσιμες στην καθοδήγηση της κερδοσκοπίας υλικού. Η έξοδος y από το perceptron predictor δεν είναι μια Boolean τιμή, αλλά ένας αριθμός που ερμηνεύουμε όπως θα λαμβανόταν εάν $y \geq 0$. Η τιμή του y παρέχει σημαντικές πληροφορίες για τον κλάδο καθώς η απόσταση του y από το 0 είναι ανάλογη προς τη βεβαιότητα ότι ο κλάδος θα ληφθεί [32]. Αυτή η εμπιστοσύνη μπορεί να χρησιμοποιηθεί, παραδείγματος χάριν, για να επιτρέψει σε μια μικροαρχιτεκτονική να εκτελέσει υποθετικά και τις δύο πορείες κλάδων όταν η εμπιστοσύνη είναι χαμηλή , και για να εκτελέσει μόνο την προβλεφθήσα πορεία όταν η εμπιστοσύνη είναι υψηλή. Μερικά σχέδια πρόβλεψης κλάδων ρητά υπολογίζουν μια εμπιστοσύνη στις προβλέψεις τους [33], αλλά στον predictor μας αυτές οι πληροφορίες έρχονται δωρεάν. Έχουμε παρατηρήσει πειραματικά ότι η πιθανότητα ότι ένας κλάδος θα ληφθεί μπορεί να υπολογιστεί ακριβώς ως γραμμική λειτουργία της παραγωγής του perceptron predictor. Το σχήμα 2.9 παρουσιάζει μια μέτρηση της πιθανότητας των δειγμάτων ότι ένας κλάδος λαμβάνεται ως λειτουργία της perceptron εξόδου για τις συγκριτικές μετρήσεις επιδόσεων ΠΡΟΔΙΑΓΡΑΦΩΝ (Standard Performance Evaluation Corporation-SPEC).



Σχήμα 2.9.: Πιθανότητα ο κλάδος να είναι Taken σαν να είναι μια λειτουργία της perceptron εξόδου

Ανάλυση της συμπεριφοράς κλάδων με perceptrons. Τα perceptrons μπορούν να χρησιμοποιηθούν για να αναλύσουν τους συσχετισμούς μεταξύ των κλάδων. Ο perceptron predictor ορίζει σε κάθε bit στην ιστορία κλάδων ένα βάρος. Όταν ένα ιδιαίτερο bit συσχετίζεται έντονα με μια ιδιαίτερη έκβαση κλάδων, το μέγεθος του βάρους είναι υψηλότερο από όταν δεν υπάρχει λιγότερος ή κανένας συσχετισμός. Κατά συνέπεια, ο perceptron predictor μαθαίνει να αναγνωρίζει τα bits στην ιστορία ενός ιδιαίτερου κλάδου που είναι σημαντικό για την πρόβλεψη, και μαθαίνει να αγνοεί τα ασήμαντα bits. Αυτή η ιδιοκτησία του perceptron predictor μπορεί να χρησιμοποιείται με τη σκιαγράφηση που παρέχει ανατροφοδότηση για άλλα σχέδια πρόβλεψης κλάδων.

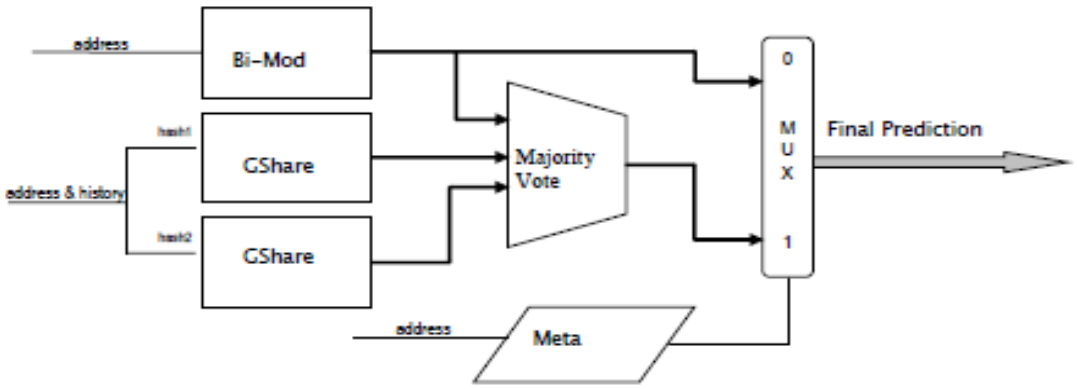
2.10 De-Aliasing Predictors

Οι de-aliasing predictors προσπαθούν να λύσουν το πρόβλημα του κατακερματισμού των πολλαπλών κλάδων στην ίδια είσοδο σε έναν πίνακα. Για να αποφύγουμε αυτήν την σύγκρουση, μπορούμε να διαστρέψουμε δύο ίδιους predictors που κατακερματίζουν την είσοδο σε αυτούς τους predictors σε διαφορετικά σημεία στο PHT. Κατά συνέπεια, εάν δύο διευθύνσεις ταιριάζουν (map) στην ίδια είσοδο σε

έναν predictor, μπορούμε να είμαστε αρκετά σίγουροι ότι δεν θα ταιριάξουν (map) στην ίδια είσοδο στο δεύτερο predictor.

Ο 2bc-gskew [14] χρησιμοποιεί δύο Gshare predictors , και ένα bi-mod predictor όπως φαίνεται στο σχήμα 3. Η εισαγωγή στα δυο Gshares είναι η ίδια σφαιρική πληροφορία ιστορίας, εντούτοις, η ιστορία κατακερματίζεται στο PHT χρησιμοποιώντας δύο διαφορετικές λειτουργίες κατακερματισμού (hashing functions). Οι προβλέψεις των bi-mod και των gshare predictors υπόκεινται έπειτα στην ψηφοφορία με πλειοψηφία (majority voting). Εάν και οι τρεις predictors δεν συμφωνούν, η πρόβλεψη με δύο ψήφους είναι αυτή που εμπιστεύεται (trusted). Εάν οι gshare predictors διαφωνούν, αυτό σημαίνει ότι ένας από τους predictors πρέπει να έχει μια σύγκρουση στον πίνακα, και έτσι ο bi-mod predictor θα είναι η ψηφοφορία απόφασης (deciding vote). Ένας meta predictor είναι επίσης παρών στην εκτέλεση καθηκόντων μεταξύ της εξόδου του ψήφου πλειοψηφίας (output of majority vote) και του bi-mod predictor. Ο meta predictor χρησιμοποιείται για να λύσει το πρόβλημα λανθάνουσας κατάστασης στους gshare predictors. Δεδομένου ότι αυτοί οι predictors είναι βασισμένοι στη σφαιρική ιστορία (global history), αυτό παίρνει τον αριθμό K κλάδων για την ιστορία που γεμίζει, όπου το K είναι ο αριθμός bits ιστορίας που χρησιμοποιούνται.

Σχήμα 3: Ο 2bc-gskew predictor

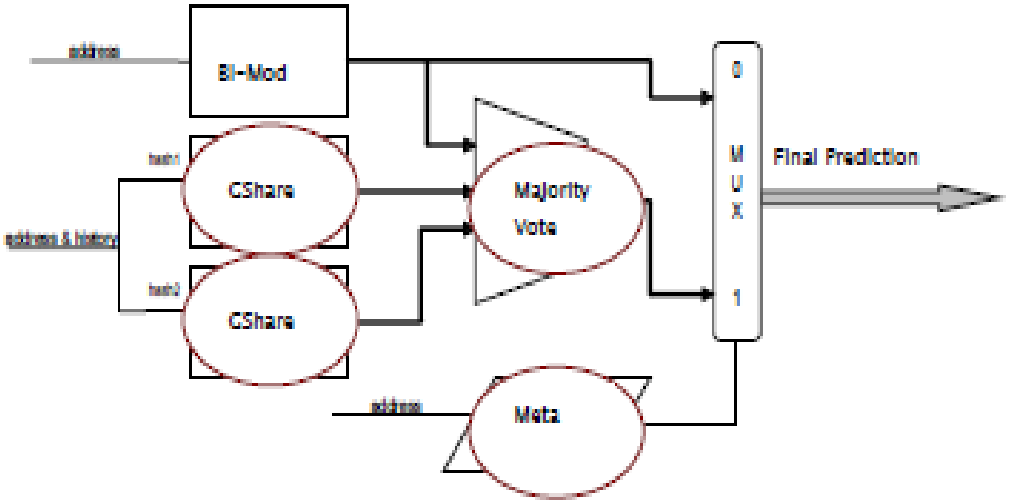


2.11 Υβριδικοί Predictors

Οι υβριδικοί predictors κεφαλαιοποιούνται (capitalize) σε όλους τους predictors που συζητούνται μέχρι αυτό το σημείο. Κάθε ένας διαφορετικός predictor έχει διαφορετικές δυνατότητες και αδυναμίες. Οι υβριδικοί predictors προσπαθούν να συμπληρώσουν το χάσμα με το συνδυασμό απλούστερων predictors. Εδώ να σημειώσουμε ότι ο 2bc-gskew είναι ένας hybird predictor. Εντούτοις ένα πρόβλημα των predictors είναι η εφικτότητα (feasibility) δηλαδή το πώς να συλλάβει κανείς τις καλύτερες πτυχές διάφορων predictors, ενώ συγχρόνως να τηρεί τις υπολογιστικές και hardware δαπάνες στο ελάχιστο. Για παράδειγμα, συνδυάζοντας όλους τους

predictors που συζητήθηκαν μέχρι αυτό το σημείο πιθανόν να αποδίδει την καλύτερη πρόβλεψη, αλλά πόσο καιρό θα έπαιρνε για να κάνει μια πρόβλεψη, και μήπως η ακριβής προσπάθεια θα ήταν ωφέλιμη σε σχέση με το τι θα πετύχουμε από μια πολύ γρηγορότερη τεχνική πρόβλεψης; Αυτή είναι η κατεύθυνση που η δυναμική πρόβλεψη κλάδων έχει μετακινηθεί, και ερευνάμε και προτείνουμε διάφορα νέα υβριδικά σχέδια.

Σχήμα 2.11: Η δομή του 2bc-gskew predictor και των περικυκλωμένων συστατικών (encircled components) όπου τα perceptrons μπορούν να χρησιμοποιηθούν.



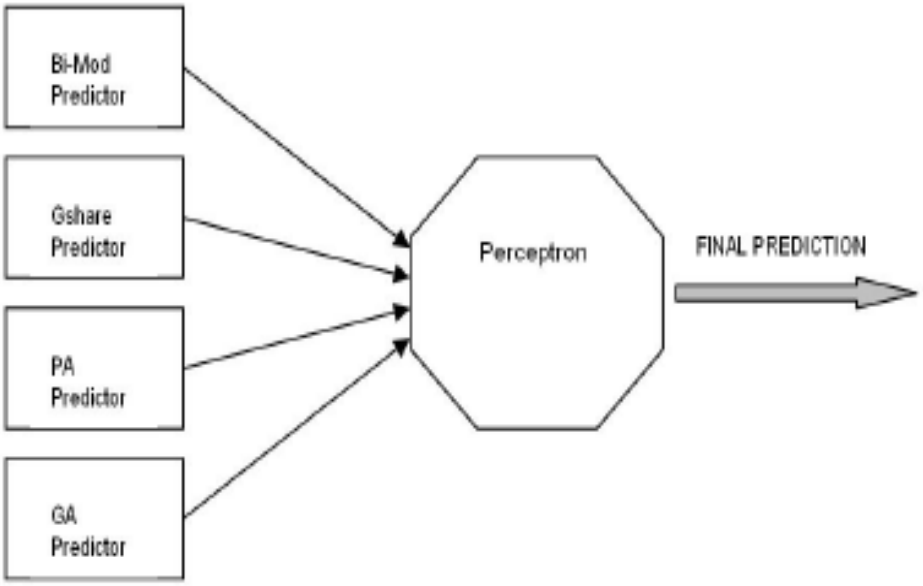
Εδώ συνδυάζουμε τα καθιερωμένα bit counter σχέδια με το νέο perceptron πρότυπο. Υποθέτουμε ότι ο συνδυασμός και των δύο ιδεών θα παραγάγει ένα υβρίδιο των predictor που θα έχει την καλύτερη ακρίβεια πρόβλεψης. Συγκεκριμένα, χειριζόμαστε το καλύτερο two-bit counter predictor, τον 2bc-gskew, ο οποίος εισήχθη από τον Seznec [14]. Προτείνουμε τέσσερα συστατικά όπου τα perceptrons μπορούν να εισαχθούν, (βλέπουμε το σχήμα 2.11).

2.11.α Ο weighted-voting predictor

Ο πρώτος προτεινόμενος weighted predictor, αντικαθιστά το μηχανισμό πλειοψηφίας με ψηφοφορία 2bc-gskew με perceptron. Το κίνητρο για αυτόν τον predictor είναι ότι για κάθε κλάδο, ένας predictor θα είναι ακριβέστερος. Επομένως, χρησιμοποιούμε το perceptron που προσδιορίζει ποιος predictor ταιριάζει καλύτερα στον κλάδο. Σαν αναλογία, εξετάστε την κατάσταση όπου ένα πρόσωπο πρέπει να λάβει μια απόφαση και αναζητά τη σύσταση από μια ομάδα. Είναι κοινό να δοθεί περισσότερη σημασία στο πρόσωπο που έχει δώσει τις σωστές συμβουλές στο

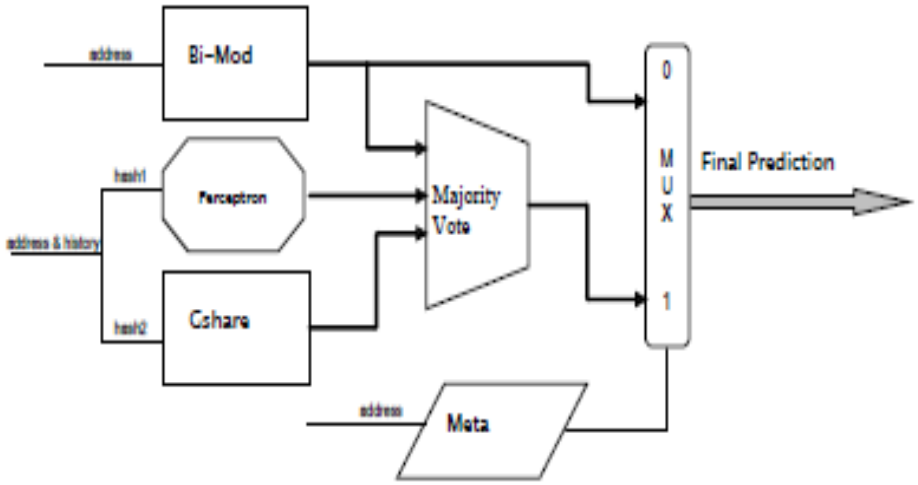
παρελθόν. Στον weighted predictor, όπως πριν, κάθε predictor δίνει την πρόβλεψή του αλλά περισσότερη προσοχή δίνεται στον predictor που είναι ακριβέστερος στις προηγούμενες προβλέψεις. Σημειώστε ότι ένας predictor μπορεί να αποδίδει καλά σε έναν κλάδο, αλλά να αποδίδει κακώς σε έναν άλλον. Αυτή η κατάσταση συσχετίζει το να πάρεις καλές συμβουλές από έναν καθηγητή μαθηματικών για τον υπολογισμό και τη λήψη κακών συμβουλών από το ίδιο πρόσωπο για το χορό salsa. Για να συλλάβει αυτές τις πληροφορίες, το perceptron ρυθμίζει τα βάρη του έτσι ώστε να δώσει τα κατάλληλα βάρη σε κάθε predictor που βασίζεται στην ακρίβεια. Όπως φαίνεται στο σχήμα 2.11.α, κάθε predictor δίνει την πρόβλεψή του στον perceptron, και το perceptron καθορίζει την τελική έκβαση.

Σχήμα 2.11.α: Η δομή του ζυγισμένου predictor.



2.11.β Ο pgskew predictor

Σχήμα 2.11.β : Η δομή του pgskew predictor



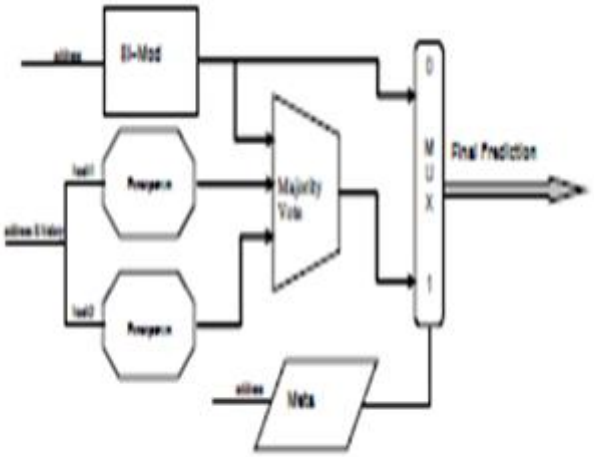
Ο δεύτερος προτεινόμενος predictor είναι ο pgskew predictor. Αυτός ο predictor είναι βασισμένος στον 2bc-gskew predictor, αλλά δεν χρησιμοποιούμε δύο gshare predictors που διαστρέφουν τις κατακερματισμένες διευθύνσεις τους. Αντ' αυτού, χρησιμοποιούμε έναν gshare predictor και έναν perceptron predictor. Δεδομένου ότι τα perceptrons ξεπερνούν τους two-bit counters, το δόσιμο στο perceptron μιας ψήφου μπορεί να κάνει το υβρίδιο να αποδώσει καλύτερα.

Ο resulting predictor είναι ένα υβρίδιο που αποτελείται από το bi-mod, το gshare, και τους perceptron predictors, όλοι αυτοί κάνουν μεμονωμένες προβλέψεις. Η τελική έκβαση καθορίζεται από την κυρίαρχη πρόβλεψη. Χρησιμοποιώντας έναν ψήφο πλειοψηφίας για να επιλεχτεί η έκβαση, τα κοινά mispredictions θα απορριφθούν, αυξάνοντας την ακρίβεια της πρόβλεψης [14].

Αυτός ο συνδυασμός ενσωματώνει τις δυνάμεις κάθε predictor. Ο Bi-mod λειτουργεί συγκεκριμένα καλά με τους προκατειλημμένους κλάδους (biased branches) που είναι συνήθως οι ίδιοι (taken ή not taken), και έχει μια χαμηλή λανθάνουσα κατάσταση εκμάθησης. Εντούτοις, αποτυγχάνει να έχει καλή απόδοση για τους κλάδους που έχουν ένα εναλλασσόμενο σχέδιο των taken ή non-taken διακλαδώσεων. Το Gshare αφ' ενός, μπορεί να ανιχνεύσει ένα τέτοιο σχέδιο, αλλά έχει aliasing προβλήματα. Ο perceptron predictor μπορεί να ανιχνεύσει αυτό το σχέδιο, αλλά το χειροκίνητο aliasing είναι καλύτερο. Ενώ ο bi-mod εκμεταλλεύεται την τοπική ιστορία, το perceptron αλλά και το gshare χρησιμοποιούν και τοπική αλλά και global history, αυτοί οι τελευταίοι δύο predictor έχουν αποδειχθεί ότι γενικά αποδίδουν ακριβέστερα. Εντούτοις η πρόβλεψη bi-mod έχοντας χαμηλή καθυστέρηση εκμάθησης , θα χρησιμοποιηθεί έως ότου τα άλλα δύο να είναι έτοιμα. Στην περίπτωση του δεσμού (tie), ο bi-mod θα είναι η ψηφοφορία απόφασης όπως πριν.

2.11.γ Ο ppskew predictor

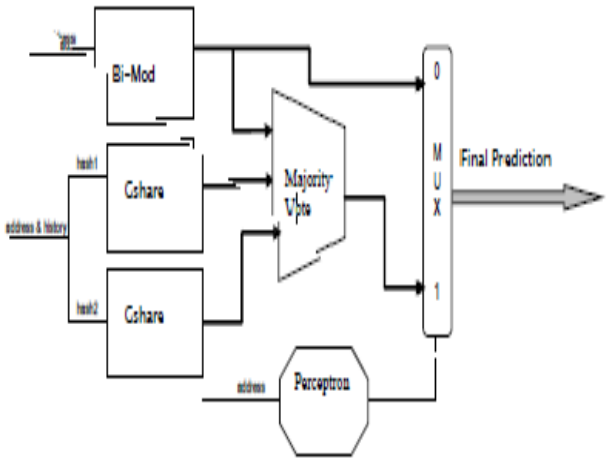
Σχήμα 2.11.γ: Η δομή του pgskew predictor.



Ο Ppskew είναι ο τρίτος predictor, που βασίζεται στην ίδια δομή 2bc-gskew αλλά με perceptrons εν αντιθέση των gshare predictors. Ο Gshare χρησιμοποιήθηκε σε 2bc-gskew επειδή ήταν ο καλύτερος σε απόδοση two-bit counter predictor μέχρι εκείνη τη στιγμή, αλλά τώρα με τα perceptrons έχει αποδειχθεί ότι μπορεί να ξεπεράσει τους προκατόχους του. Οι μηχανικοί του υβριδικού predictor παραμένουν το ίδιο. Οι προβλέψεις Bi-mod χρησιμοποιούνται ενώ τα perceptrons προθερμαίνονται. Ο meta predictor λειτουργεί όπως προηγουμένως συζητήθηκε. Οι δύο perceptron predictor είναι λοξοί (skewed), έτσι ώστε να αποφύγουν την ίδια είσοδο των βαρών για κάθε perceptron predictor. Αυτό προκαλεί συγκρούσεις (collisions) για να χτυπήσει διαφορετικά perceptrons σε κάθε perceptron predictor. Ένα σημαντικό χαρακτηριστικό γνώρισμα για το ppskew, σε αντιδιαστολή με το pgskew είναι ότι το perceptron δεν μπορεί να πλειοψηφήσει δηλαδή να κερδίσει την πλειοψηφία από το μετρητή bit. Το γεγονός μπορεί να εμφανιστεί στο pgskew, όπου το perceptron παράγει μια σωστή πρόβλεψη, αλλά διαφωνεί και με τις προβλέψεις του bi-mod αλλά και με αυτές του gshare. Αυτή η κατάσταση αποφεύγεται στο ppskew.

2.11.δ Ο meta-perceptron predictor

Σχήμα 2.11.δ: Η δομή του meta-perceptron predictor



Αυτός ο τελευταίος predictor, ο meta-perceptron predictor, χρησιμοποιεί perceptron αντί ενός metapredictor. Ο ρόλος του είναι να επιλέξει μεταξύ του bi-mod και των προβλέψεων ψήφου πλειοψηφίας, με το στόχο να μάθει ακριβέστερα τον ακριβή χρόνο όταν ο gshare είναι έτοιμος να παραγάγει τα ακριβή αποτελέσματα. Στον 2bc-gskew και σε αυτούς τους άλλους προτεινόμενους predictors, δεν υπάρχει κανένας συγκεκριμένος κανόνας που προσδιορίζει πόσα history bits πρέπει να δει ο gshare προτού να παραγάγει τις καλές προβλέψεις. Αυτό εξαρτάται ιδιαίτερα από το σχέδιο κλάδων, και το κάνει δύσκολο να εικαστεί όταν η φάση προθέρμανσης τελειώνει. Στις εφαρμογές μας, αποφασίσαμε να δώσουμε στο gshare τουλάχιστον δύο bit ιστορίας προτού να εξεταστεί από το meta-predictor. Με το perceptron που λειτουργεί ως meta-predictor, δεν είναι απαραίτητο να λάβουμε την απόφαση. Ο perceptron θα προσδιορίσει βασιζόμενος στην ακρίβεια, πότε ο gshare πρέπει να γίνει έμπιστος. Δεδομένου ότι ο ψήφος πλειοψηφίας δίνει τις ακριβέστερες προβλέψεις, τα κατάλληλα βάρη θα ρυθμιστούν. Μια σημαντική ανησυχία για την οργάνωση αυτού του υβριδικού predictor είναι ο αριθμός εισαγωγών perceptron. Έχοντας μόνο δύο εισόδους όπου κάθε μια έχει μόνο δύο πιθανές τιμές, που καθορίζουν ένα ξεχωριστό υπερπλάνο (hyperplane) για όλες τις εισαγωγές κλάδων είναι απίθανη. Αλλά για την ολοκλήρωση, εφαρμόζουμε αυτόν τον predictor.

2.12 Branch Predictors στις σημερινές CPUs

Οι τρέχοντες μικροεπεξεργαστές χρησιμοποιούν τους δύο επιπέδων branch predictors.. Τα εξής είναι τρία ξεχωριστά παραδείγματα:

●Οι AMD K6 και K7 (Athlon) επεξεργαστές χρησιμοποιούν τους GAs predictors [26].
● Το HP-PA 8700 χρησιμοποιεί μια 2048 είσοδο GAs με το μηχανισμό συμφωνίας (agree mechanism) [29, 30].
● Ο Άλφα πυρήνας 21264 χρησιμοποιεί έναν hybrid predictor που αποτελείται από δύο two-level predictors [31]: Ένας 4K-εισόδων GAg συντάσσεται από μια σφαιρική ιστορία κλάδων δώδεκα bit ενώ μια 1K-είσοδος PHT τριών bit που διαποτίζει τους μετρητές αρχικοποιείται από τη μια από τις 1024 τοπικές ιστορίες κλάδων των 10 bit. Η τελική πρόβλεψη κλάδων επιλέγεται αρχικοποιώντας έναν τρίτο predictor που παρακολουθεί την σχετική ακρίβεια των δύο predictors για μια ιδιαίτερη σφαιρική ιστορία. Ο Άλφα predictor είναι πράγματι πολύ ακριβής, είναι ο ακριβέστερος των εφαρμοσμένων branch predictors που έχουμε παρατηρήσει. Εντούτοις, η πολυπλοκότητα εφαρμογής του έρχεται με ένα κόστος. Ο Άλφα branch predictor αγνοεί έναν λιγότερο ακριβή predictor εντολών κρύπτης οδηγίας(cache line), εισάγοντας μια φυσαλίδα ενιαίος-κύκλων (single-cycle bubble) στη σωλήνωση όποτε τα δύο διαφωνούν [31].

2.13 Απόδοση

Κάθε μία από αυτές τις bit counter τεχνικές : bi-mod, PA's, GA's, Gshare, και 2bc-gskew προσπαθούν όλες να λύσουν ένα διαφορετικό πρόβλημα στην πρόβλεψη διακλαδώσεων. Ενώ μερικοί χρησιμοποιούν τις τοπικές τεχνικές πρόβλεψης, μερικοί άλλοι χρησιμοποιούν τις σφαιρικές (global), τεχνικές και οι predictors όπως είναι ο Gshare και ο 2bc-gskew χρησιμοποιούν και τις δύο τεχνικές. Ενώ ο bi-mod είναι πιο αποδοτικός και πολύ γρήγορος, είναι ο λιγότερο ακριβής predictor. Ο Gshare και ο 2bc-gskew μπορεί να χρησιμοποιούν περισσότερο hardware για να εφαρμοστούν, αλλά είναι ακριβέστεροι. Η ακρίβεια αυτών των predictors είναι κάτι σαν συγκεκριμένη εφαρμογή, αλλά γενικά, ο bi-mod αποδίδει χειρότερα, σε σχέση με τον gshare και τον 2bcgskew που αποδίδουν καλύτερα. Αυτό είναι διαισθητικό επειδή και ο Gshare αλλά και ο 2bc-gskew έχουν περισσότερες πληροφορίες στη διάθεσή τους. Οι predictors PA και GA είναι κάπου στο ενδιάμεσο [2].

3 Πώς να χρησιμοποιούμε το Championship Branch Prediction Evaluation Framework

3.α Εγκατάσταση του Simulation Infrastructure

1.Κατεβάζουμε τον **cbp3_framework.tar.gz** simulator από το

http://www.jilp.org/jwac-2/cbp3_framework_instructions.html

2. Κάνουμε unpack

tar -xzvf cbp3_framework.tar.gz

cd framework

3. Κατεβάζουμε τα traces και ξεζιπάρουμε και τα δύο αρχεία ως εξής :

tar -xzvf cbp3_traces1.tar.gz

tar -xzvf cbp3_traces2.tar.gz

Για να ελέγξουμε ότι το αρχείο έχει κατέβει σωστά, ελέγχουμε το MD5SUM των traces σε σχέση με εκείνο στο αρχείο MD5SUMS:

4. Διαμορφώνουμε το Makefile: αφού σιγουρευτούμε ότι είμαστε στο framework/ directory και ότι το μονοπάτι του gcc compiler έχει εγκατασταθεί σωστά. Διαμορφώνουμε την έκτη γραμμή του Makefile για να ταιριάξει με τη διαμόρφωση του συστήματος. Για να μπούμε στη διαμόρφωση (configuration), γράφουμε

"uname -a"

Τώρα εάν το σύστημά σας είναι 32-bit (IA-32, i686), τροποποιήστε την έκτη γραμμή του Makefile σε

FORMAT = 32

Αλλιώς δηλαδή εάν το σύστημά σας είναι 64-bit (x86_64), τροποποιήστε την έκτη γραμμή του Makefile σε

FORMAT = 64

5. Για να τρέξουμε το simulator πρέπει να σιγουρευτούμε ότι βρισκόμαστε στο framework/ directory, και εκεί γράφουμε

make

3.β Πώς τρέχουμε τον εξομοιωτή:

Για να τρέξουμε τον εξομοιωτή στο αναδημιουργημένο αρχείο κάνουμε το εξής από το CRC/ directory):

```
mkdir runs

cd runs/
```

Η command διαμόρφωση είναι:

../cbp3 -t [trace file name] > [output file name]

Παραδείγματος χάριν, για να εξομοιώσουμε ένα δείγμα trace, τρέχουμε

../cbp3 -t ../trace_test.bz2 > trace_test.out

Μπορούμε να εξετάσουμε τις στατιστικές αυτές , από αυτό το τρέξιμο, με το άνοιγμα του trace_test.out. Το αρχείο output περιέχει τις στατιστικές για τους κύκλους , τις εντολές , τα mispredictions κλάδων, το misprediction κλάδων penalites, και ένα τελικό αποτέλεσμα και για τους υπό όρους και έμμεσους κλάδους.

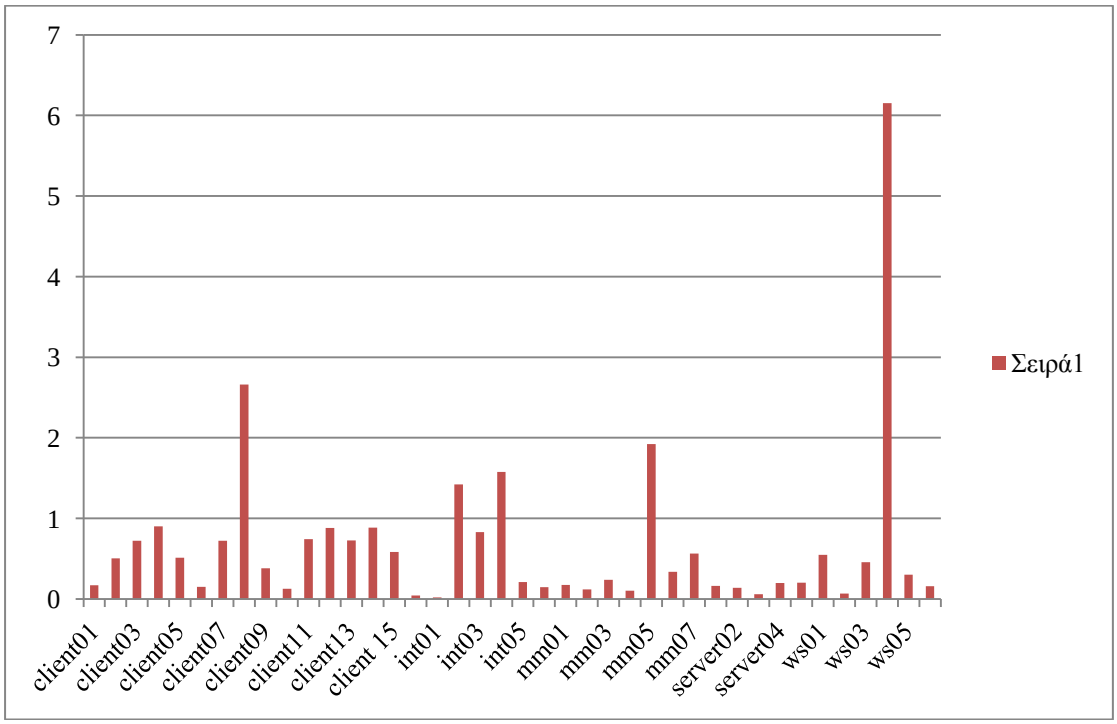
3.1 Testing Framework

Αφότου έχουμε εφαρμόσει διάφορους δημοσιευμένους (published) predictors και τους προτεινόμενους predictors, θα χρησιμοποιήσουμε το Championship Branch Prediction (CBP) framework για να εξετάσουμε την ακρίβεια κάθε predictor. Οι συγκριτικές μετρήσεις επιδόσεων παρέχονται με 40 trace αρχεία που ταξινομούνται σε 5 κατηγορίες: CLIENT, INT (Integer), MM (Multimedia), SERVER και WS (Workstation). Ενώ κάθε trace αρχείο έχει εκατό χιλιάδες εντολές δεν υποστηρίζουμε αυτά τα traces για να αντιπροσωπεύσουμε όλους τους τύπους προγραμμάτων. Εντούτοις, είναι μια εύλογη βάση για να συγκρίνουμε την εργασία μας με τους δημοσιευμένους (published) predictors.

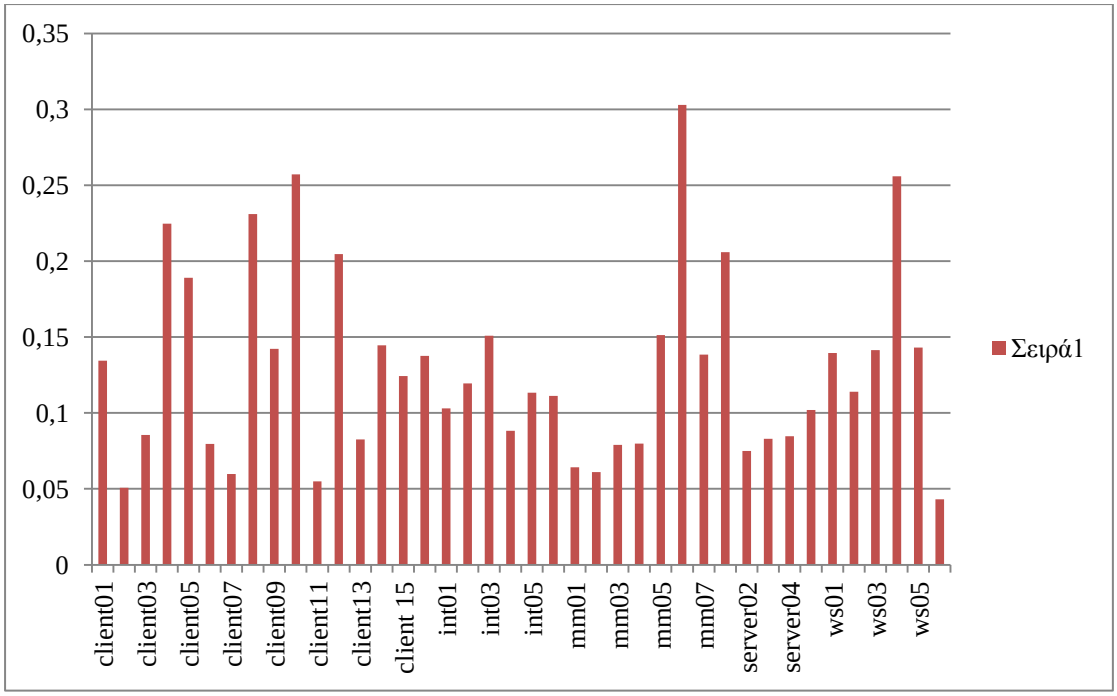
3.2 Μετρικές

Εξετάζουμε δύο τυποποιημένες μετρικές για τις συγκρίσεις μας . Μετράμε τον αριθμό mispredictions ανά 1.000 εντολές (MPKI) και μετράμε την αναλογία των mispredictions στο συνολικό αριθμό κλάδων. Η πρώτη μετρική είναι η τυποποιημένη μετρική που χρησιμοποιείται στην επίσημη μελέτη, αλλά η δεύτερη μετρική δίνει μια πιο αισθητή αξιολόγηση των πόσων περισσότερων (ή λιγότερων) κλάδων μπορούμε ακριβώς να προβλέψουμε. Εκθέτουμε των μέσο όρο των 40 traces που δίνονται αλλά και το γενικό μέσο όρο. Ορισμένοι predictors μπορούν να εκτελέσουν το καλύτερο με ορισμένες εφαρμογές, και θα τις αναγνωρίσουμε υπό αυτήν τη μορφή, αλλά οι μελέτες μας εστιάζουν στον Gshare predictor που θα ήταν καταλληλότερος για μια γενικής χρήσης μηχανή. (σχήμα 3.1)

Προκειμένου να παραμείνουμε σύμφωνοι με το 3rd CBP , θα χρησιμοποιήσουμε τη μετρική Misprediction Penalty per Kilo Instructions (MPPKI) που προτάθηκε για το 3rd CBP Στην πράξη, για τους predictor που αναφέρονται σε αυτό το έγγραφο, αυτή η μετρική είναι συνολικά ανάλογη προς τον αριθμό των misprediction παρόλο που η μέση ποινή αστοχίας (misprediction penalty) ποικίλλει μεταξύ των συγκριτικών μετρήσεων επιδόσεων. (σχήμα 3.2)



Σχήμα 3.1 . Miss prediction statistics για 40 benchmarks
(CL: Client, INT: Integer, MM: Multi-media, SER: Server, WS: Workstation)



Σχήμα 3.2 . Penalty prediction statistics για 40 benchmarks
(CL: Client, INT: Integer, MM: Multi-media, SER: Server, WS: Workstation)

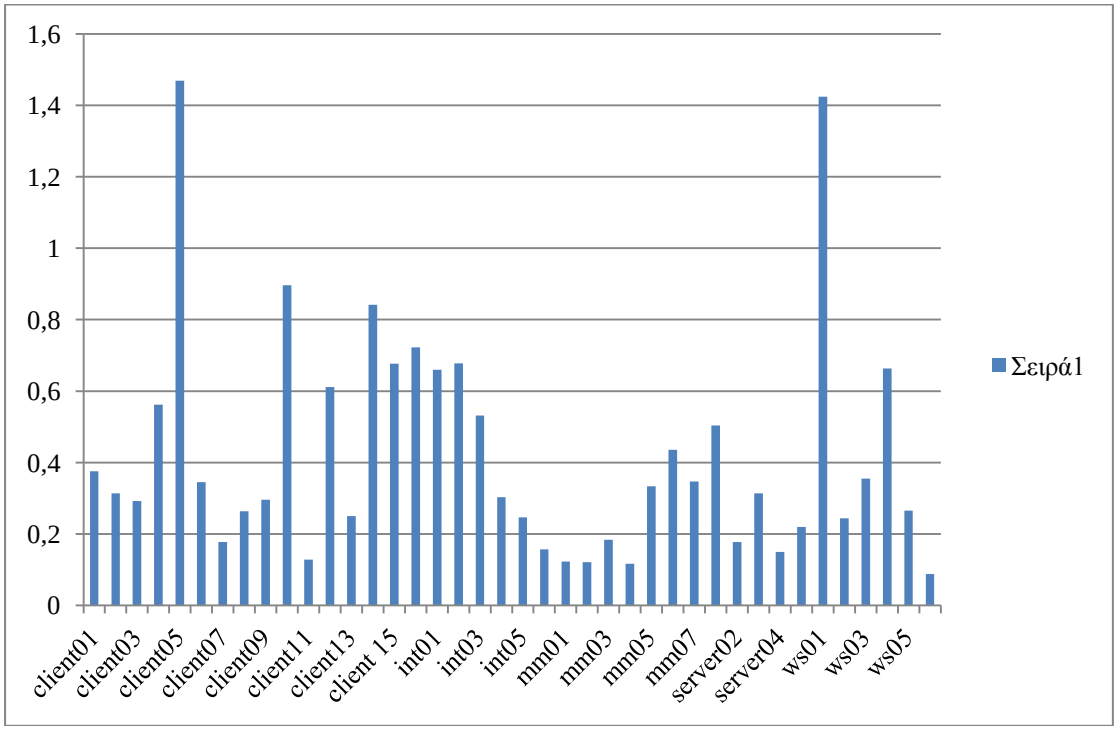
3.3 *Καθορισμένα χαρακτηριστικά συγκριτικής μέτρησης επιδόσεων (Benchmark Set Characteristics)*

3.3.1 Gshare branch predictor

Την υψηλότερη αστοχία πρόβλεψης περιλαμβάνουν τα παρακάτω αρχεία: CLIENT02 , CLIENT08 , και WS04. Αυτές οι 3 συγκριτικές μετρήσεις επιδόσεων αντιπροσωπεύουν περίπου τα 3/5 των αστοχιών πρόβλεψης (mispredictions) των συγκριτικών μετρήσεων επιδόσεων. Όλο αυτό τα benchmarks εκθέτουν περισσότερες από 500.000 αστοχίες πρόβλεψης (δηλ. περισσότερα από 10 mispredictions ανά kilomicrooperations) για τον predictor αναφοράς. Μεταξύ αυτής της κατηγορίας, ο CLIENT02 είναι η μόνη συγκριτική μέτρηση επιδόσεων που όταν υπόκειται σε ένα χαμηλό ποσοστό misprediction απλά αυξάνει το μέγεθος του predictor στο μη ρεαλιστικό προϋπολογισμό αποθήκευσης των (256 Mbits).

3.3.2 Perceptron branch predictor

Την υψηλότερη αστοχία πρόβλεψης περιλαμβάνουν τα παρακάτω αρχεία: CLIENT05 , CLIENT10 και WS01 (σχήμα 3.3). Αυτές οι 3 συγκριτικές μετρήσεις επιδόσεων αντιπροσωπεύουν περίπου τα 3/5 των αστοχιών πρόβλεψης (mispredictions) των συγκριτικών μετρήσεων επιδόσεων.



Σχήμα 3.3 . Miss prediction statistics για 40 benchmarks
(CL: Client, INT: Integer, MM: Multi-media, SER: Server, WS: Workstation)

3.4 Βελτιστοποιημένες ρυθμίσεις

Για να συγκρίνουμε τους προτεινόμενους predictors με το bit-counter, έχουμε εφαρμόσει το gshare predictor. Ο gshare είναι ο καλύτερος branch predictor 2 επιπέδων.

Παρατηρούμε ότι ο gshare παρουσιάζει την καλύτερη δυνατή ακρίβεια για μέγεθος πίνακα 227 (λίγο λιγότερο από το μέγιστο επιτρεπτό του contest 230). Παράλληλα παρατηρούμε ότι το history length παίζει σημαντικό ρόλο στην απόδοση του gshare. Βλέπουμε ότι για μικρά μεγέθη έχουμε κακή απόδοση, ενώ για τιμές μεγαλύτερες του 14 η απόδοση δεν βελτιώνετε καθόλου, εν αντιθέσει για πολύ μεγάλες τιμές (κοντά στο 30), η απόδοση πέφτει κιόλας. Συνεπώς τα νούμερα με την καλύτερη απόδοση είναι *HistoryLength=14* και *TableBits=27* (TableSize=227), τιμές οι οποίες επαληθεύονται και από την έρευνα του McFarilng [+].

Ο gshare predictor εφαρμόστηκε με έναν σταθερό προϋπολογισμό 64 Kb για να λειτουργήσει. Εντούτοις, μια σημαντική ανησυχία είναι να βρεθεί η βέλτιστη

δρομολόγηση του προϋπολογισμού ανά predictor για να επιτρέψει σε κάθε predictor να αποδώσει στο καλύτερό του. Έπρεπε να διευθύνουμε τις πρόσθετες δοκιμές ανά predictor προκειμένου να βρεθούν οι βέλτιστες ρυθμίσεις, και αυτό φαίνεται στον ακόλουθο πίνακα.

Σχήμα 9: Βελτιστοποιημένη δρομολόγηση υλικού ανά predictor

PREDICTOR	*gshare A table size*	*gshare B table size*	*bimod table size*	*meta table size*	*perceptron A table size*	*perceptron A history bits*	*perceptron B table size*	*perceptron B history bits*
gshare	32,000	-	-	-	-	-	-	-
2bc-gskew	8192	8192	8192	8192	-	-	-	-
pweighted	8192	8192	4096	-	1000	3	-	-
pgskew	8192	-	2048	2048	300	24	-	-
ppskew	-	-	8192	4096	124	20	124	20
meta-perceptron	8192	8192	2048	-	1695	8	-	-
perceptron	-		-	-	250	32	-	-

3.5 Η ποινή (penalty) σε ένα predictor

Η σχεδίαση σε ένα branch predictor στρέφεται χαρακτηριστικά μόνο στην ελαχιστοποίηση του ποσοστού misprediction, ενώ αγνοεί την ποινή σε μια άστοχη πρόβλεψη (misprediction penalty). Ποινή (penalty) ορίζεται ως το χρονικό διάστημα που το σύστημα δεν προσκομίζεται κατά μήκος της σωστής πορείας , περιλαμβάνει τη διασωλήνωση που ξαναγεμίζει το χρόνο συν οποιαδήποτε καθυστέρηση στην αξιολόγηση του όρου κλάδων, ίσως λόγω των εξαρτήσεων. Επειδή η ποινή στην αστοχία πρόβλεψης ποικίλλει ευρέως από κλάδο σε κλάδο, η απόδοση μπορεί να βελτιωθεί με τη χρησιμοποίηση ενός predictor που καταβάλλει μεγαλύτερη προσπάθεια να προβλέψει τους κλάδους με την υψηλότερη ποινή εις βάρος κάποιου άλλου κλάδου, ακόμα κι αν ο συνολικός αριθμός των mispredictions δεν αλλάζει.

4 Πρόβλεψη διακλαδώσεων με perceptrons

Στην πρόσφατη έρευνα ο perceptron predictor έχει αποδειχθεί ότι είναι ο ακριβέστερος branch predictor [10]. Ο perceptron αλγόριθμος εκμάθησης, μια μέθοδος για την εποπτευμένη τεχνική εκμάθησης, επιτρέπει σε ένα perceptron να μάθει εάν ένας κλάδος λαμβάνεται (taken) ή δεν λαμβάνεται (non taken). Για περισσότερες λεπτομέρειες θα δούμε το παράρτημα Α. Μετά από την αρχική πρόταση από το Jimenez και Lin, ένας μεγάλος αριθμός συνδυασμών perceptron predictor έχει αναπτυχθεί. Αυτό το τμήμα συζητά και τα ζητήματα με την εφαρμογή του αρχικών perceptron predictor και των συνδυασμών του perceptron predictor.

4.1 Εφαρμογή ενός perceptron predictor

Οι perceptron predictor κρατούν έναν πίνακα βάρους, που συντάσσεται από τη διεύθυνση κλάδων. Κάθε γραμμή κρατά τα βάρη που αντιστοιχούν στην εισαγωγή του απαραίτητου predictor στον υπολογισμό της εξόδου. Για να εφαρμόσει τον predictor στο chip, είναι σημαντικό να σημειωθεί ότι το προϊόν σημείων (dot product) των διανυσμάτων βάρους και ιστορίας θα υπολογίσει ένα ποσό των βαρών που επιλέγονται από τα κομμάτια ιστορίας. Για να αθροίσει αυτές τις τιμές παράλληλα, χρησιμοποιείται ένα δέντρο αθροιστών Wallace [8]. Η λειτουργία ενημέρωσης (update) μπορεί να εφαρμοστεί αργότερα στην διασωλήνωση, αφότου η συνθήκη διακλάδωσης έχει υπολογιστεί.

Η αρχική εργασία του Jimenez [7] πρότεινε μόνο τη σφαιρική (global) ιστορία για την εισαγωγή στον predictor. Αλλά ένας perceptron predictor μπορεί να πάρει τη σφαιρική ιστορία, την τοπική ιστορία, ή έναν συνδυασμό των δύο ως είσοδο. Ο perceptron είναι σε θέση να μάθει ποια bits είναι πιο πολύ σημαντικά για την πρόβλεψη και ανάπτυξη ενός υπερπλάνου (hyperplane) δηλαδή εκείνων των bits που συσχετίζονται αποτελεσματικά με τους κλάδους όταν ενημερώνεται ο perceptron. Αυτό το χαρακτηριστικό γνώρισμα επιτρέπει στο percepton να κάνει χρήση του μακροχρόνιου ιστορικού που χρησιμοποιείται από τους two bit counter perdictors που παρουσιάζονται στην παράγραφο 2. Αυτό είναι επειδή η ιστορία γίνεται είσοδος στο perceptron αντί να επιλέξει ένα μετρητή όπως γίνεται στον Gshare.

4.2 Bit Counters εναντίον Perceptrons

Και οι bit counter schemes αλλά και οι perceptrons προσπαθούν να προβλέψουν ακριβώς το μέλλον ενός κλάδου βασισμένου στην προηγούμενη ιστορία του, π.χ. οι two-bit counters προβλέπουν βασισμένοι στις τελευταίες τρεις εκβάσεις ενός κλάδου, ενώ οι perceptrons εκπαιδεύονται (train) σε ολόκληρη την ιστορία ενός κλάδου. Αυτές οι διαφορές επηρεάζουν πώς τα διαφορετικά παραδείγματα χειρίζονται δύο σημαντικά προβλήματα πρόβλεψης διακλαδώσεων: συσχετισμός διακλάδωσης και αντανάκλαση διακλάδωσης.

4.2.1 Συσχετισμός κλάδων

Οι δύο επιπέδων bit counter schemes χρησιμοποιούν μεθόδους για να διακρίνουν τον κλάδο σύμφωνα με τη σφαιρική ή και τοπική ιστορία του και να προβλέψουν έπειτα τον τρέχοντα κλάδο βασισμένο στις τελευταίες δύο εκβάσεις του κλάδου σε μια συγκεκριμένη κατάσταση. Αυτή η στρατηγική στηρίζεται στην κατοχή του διακεκριμένου κλάδου που μαθαίνει την κατάσταση όπου ο κλάδος συσχετίζεται με έναν άλλο κλάδο και έπειτα αφήνει αυτόν τον διακεκριμένο κλάδο να δείξει ένα σημείο στο PHT. Αυτή η μέθοδος είναι πολύ ανεπαρκής. Θεωρείστε έναν gshare predictor που χρησιμοποιεί δεκαπέντε bits όλου του ιστορικού (global history) για να εξετάσει το δεύτερο επίπεδο του PHT. Εάν υπάρχουν μόνο τρία bits από τα δεκαπέντε που συσχετίζονται με τον κλάδο, κατόπιν οι τιμές των δώδεκα άλλων bits εντούτοις δεν πρέπει να μας πειράξει , παρόλα αυτά ο κλάδος θα μπορούσε να αντιπροσωπευθεί με 2^{12} διαφορετικούς τρόπους και να τοποθετηθεί σε διαφορετικές εγκοπές (slots) στο PHT.

Οι perceptrons, από την άλλη πλευρά χρησιμοποιούν έναν αλγόριθμο εκμάθησης για να μάθουν σε τι πρόβλεψη πρέπει να βασιστεί αυτό είτε είναι σφαιρική (global history) είτε είναι τοπική ιστορία (local history). Οι κλάδοι που αντιστοιχούν τα διαφορετικά bits στην εισαγωγή θα ζυγιστούν κατάλληλα. Στο ανωτέρω παράδειγμα, ο κλάδος θα χρειαστεί μόνο μια γραμμή του πίνακα βάρους για να μάθει τους συσχετισμούς.

4.2.2 Αντανάκλαση-Aliasing

Για έναν two-bit counter, δύο κλάδοι μπορούν να παρεμποδιστούν με τον άλλον, όταν χαρτογραφηθούν στο ίδιο σημείο στο PHT. Αυτό κάνει την ανάκλαση (aliasing) ένα μη-επιδιορθώσιμο λάθος. Εάν αυτές οι συγκρούσεις παράγουν λανθασμένη πρόβλεψη, αυτό θα αναπαραγάγει εκείνη την λανθασμένη πρόβλεψη κάθε φορά. Για το de-alias ενός bit counter predictor, υπάρχουν δύο μηχανισμοί:

- ► το να προσθέσεις περισσότερες αυλακώσεις στο PHT ή
- ► να διαστρέψεις δύο predictors .

Η προσθήκη περισσότερων slots μειώνει τον αριθμό συγκρούσεων και με το να διαστρέψει τις συγκρούσεις μπορεί να παραμεριστεί από άλλους predictors που ενδεχομένως δεν έχουν την ίδια σύγκρουση. Τα perceptrons είναι λιγότερο ευαίσθητα στα προβλήματα aliasing επειδή το perceptron μαθαίνει την εισαγωγή και των δύο κλάδων. Εφ' όσον οι εκβάσεις (λαμβάνονται ή δεν λαμβάνονται) μεταξύ της σύγκρουσης και οι κλάδοι είναι γραμμικά ευδιαχώριστοι, το perceptron θα είναι σε θέση να μάθει τους όρους και για τους δύο κλάδους. Αυτή η επιρροή είναι η προφανέστερη στον πιο ακριβή 64Kb perceptron predictor με 250 γραμμές στον πίνακα βάρους της σε αντιδιαστολή με το gshare που έχει 32.000 slots στο PHT της.

4.3 Perceptrons σε άλλους predictors

Οι perceptron predictors έχουν δύο σημαντικά μειονεκτήματα: την καθυστέρηση στον αλγόριθμο εκμάθησης και την πολυπλοκότητα του προστιθέμενου υλικού. Αλλά ακόμη και με αυτά τα μειονεκτήματα, τα perceptrons μπορούν γενικά να ξεπεράσουν και το gshare και το 2bc-gskew στους παρόμοιους προϋπολογισμούς υλικού (hardware) [8].

Τα perceptrons έχουν χρησιμοποιηθεί πρόσφατα μέσα σε διάφορους άλλους predictors [11, 3, 13, 16, 9] και έχουν εμφανιστεί σε όλους τους predictors στον διαγωνισμό πρόβλεψης κλάδων πρωταθλήματος (Championship Branch Prediction Competition) [15]. Ενώ τα perceptron είναι μια σχετικά νέα ιδέα στην πρόβλεψη διακλαδώσεων, η ένωση των predictors δεν είναι. Μια φυσική επέκταση του perceptron predictor είναι να δει με πόσους τρόπους μπορεί να χρησιμοποιηθεί στους υβριδικούς predictor και το πώς αυτός θα επηρεάσει και τα αποτελέσματα των predictor και τις δαπάνες υλικού.

Μια ενδιαφέρουσα χρήση perceptrons είναι να χρησιμοποιηθεί ένας καλύτερος μηχανισμός στάθμισης. Αυτήν την περίοδο, οι υβριδικοί predictor συνδυάζονται χρησιμοποιώντας είτε το ψήφο πλειοψηφίας είτε έναν meta predictor [2] που χρησιμοποιεί τους μετρητές για να προβλέψει ποιος predictor θα είναι σωστός. Προτείνουμε τη χρήση διάφορων predictor που τροφοδοτούν την είσοδο σε έναν perceptron, τα οποία ζυγίζουν αυτές τις προβλέψεις και προβλέπουν αναλόγως. Ο Loh και ο Harry [10], περιγράφουν έναν τέτοιο predictor. Μια άλλη ενδιαφέρουσα προσέγγιση είναι να χρησιμοποιηθούν οι gshare predictors με τις σφαιρικές ιστορίες (global history) των διάφορων μηκών ως είσοδο των perceptron [13].

Τα perceptrons έχουν χρησιμοποιηθεί σε διάφορα υβρίδια [9, 11]. Αυτοί οι predictors ελαφρύνουν το aliasing, τα οποία αυξάνουν την πιθανότητα για μη-γραμμικό διαχωρισμό (non-linearly separable). Ένας τέτοιος predictors όπως είναι ο Frankenpredictor [9] επιτυγχάνει το de-aliasing με το να διαστρέψει την τεχνική και με το συνδυασμό με άλλους predictors. Με τη χρησιμοποίηση διαφορετικών εισαγωγών όπως είναι η διεύθυνση και η ιστορία, το πρόβλημα των μη-γραμμικά ευδιαχώριστων στοιχείων μπορεί να ανακουφιστεί ελαφρώς από τη στιγμή που τα δεδομένα μπορούν να είναι γραμμικά ευδιαχώριστα όταν χρησιμοποιείται μια άλλη είσοδος [11].

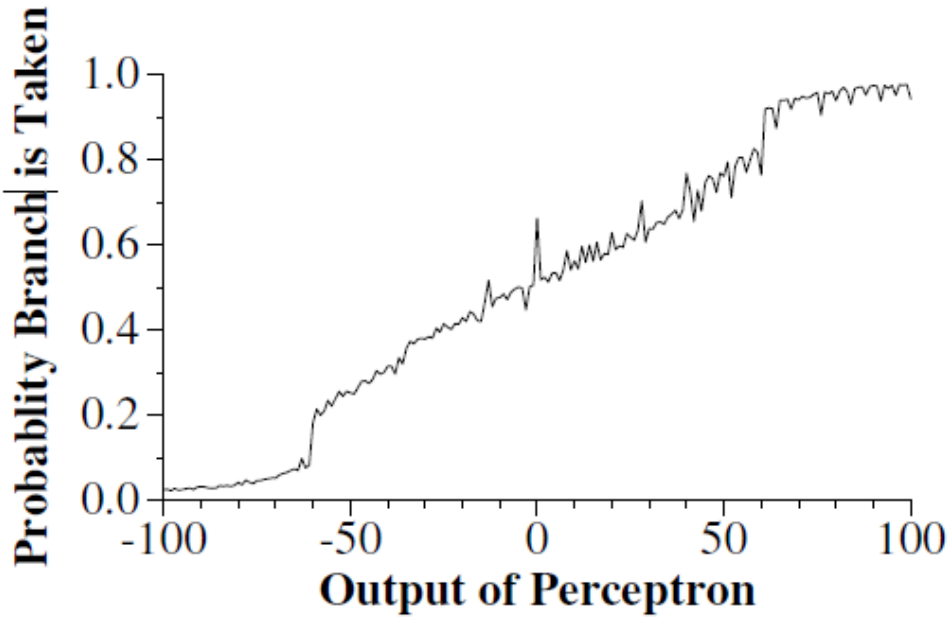
Για ορισμένους προϋπολογισμούς υλικού, κάθε μια από αυτές τις μεθόδους έχει κάνει τις βελτιώσεις πέρα από τον αρχικό perceptron predictor. Προτείνουμε τους predictors για να χρησιμοποιήσουμε μερικούς από αυτούς τους διαφορετικούς τρόπους συνδυασμού των perceptrons για να αναπτύξουμε ένα καλύτερο σχέδιο πρόβλεψης.

4.4 Πλεονεκτήματα του perceptron predictor

Αυτή η ενότητα περιγράφει δύο οφέλη από τη χρήση perceptrons για να εκτελεστεί η πρόβλεψη κλάδων.

Η πρόβλεψη διακλαδώσεων με perceptrons έχει άλλα πλεονεκτήματα πέρα από τις προηγούμενες μεθόδους. Μια perceptron έξοδος μπορεί να δώσει μια εμπιστοσύνη στην πρόβλεψη. Το διάνυσμα βάρους μπορεί να χρησιμοποιηθεί για να βρει τους συσχετισμούς μεταξύ των κλάδων, έτσι αυτή η μέθοδος μπορεί να χρησιμοποιηθεί στην προσομοίωση για να αναλύσει τη συμπεριφορά ενός προγράμματος.

Ανάθεση της εμπιστοσύνης στις αποφάσεις. Ο predictor μας μπορεί να παρέχει ένα επίπεδο εμπιστοσύνης στις προβλέψεις του που μπορούν να είναι χρήσιμες στην καθοδήγηση της κερδοσκοπίας υλικού. Η έξοδος y από το perceptron predictor δεν είναι μια Boolean τιμή, αλλά ένας αριθμός που ερμηνεύουμε όπως θα λαμβανόταν εάν $y \geq 0$. Η τιμή του y παρέχει σημαντικές πληροφορίες για τον κλάδο καθώς η απόσταση του y από το 0 είναι ανάλογη προς τη βεβαιότητα ότι ο κλάδος θα ληφθεί [32]. Αυτή η εμπιστοσύνη μπορεί να χρησιμοποιηθεί, παραδείγματος χάριν, για να επιτρέψει σε μια μικροαρχιτεκτονική να εκτελέσει υποθετικά και τις δύο πορείες κλάδων όταν η εμπιστοσύνη είναι χαμηλή , και για να εκτελέσει μόνο την προβλεφθήσα πορεία όταν η εμπιστοσύνη είναι υψηλή. Μερικά σχέδια πρόβλεψης κλάδων ρητά υπολογίζουν μια εμπιστοσύνη στις προβλέψεις τους [33], αλλά στον predictor μας αυτές οι πληροφορίες έρχονται δωρεάν. Έχουμε παρατηρήσει πειραματικά ότι η πιθανότητα ότι ένας κλάδος θα ληφθεί μπορεί να υπολογιστεί ακριβώς ως γραμμική λειτουργία της παραγωγής του perceptron predictor. Το σχήμα 3.5 παρουσιάζει μια μέτρηση της πιθανότητας των δειγμάτων ότι ένας κλάδος λαμβάνεται ως λειτουργία της perceptron εξόδου για τις συγκριτικές μετρήσεις επιδόσεων ΠΡΟΔΙΑΓΡΑΦΩΝ (Standard Performance Evaluation Corporation-SPEC).



Σχήμα 3.5.: Πιθανότητα ο κλάδος να είναι Taken σαν να είναι μια λειτουργία της perceptron εξόδου.

Ανάλυση της συμπεριφοράς κλάδων με perceptrons. Τα perceptrons μπορούν να χρησιμοποιηθούν για να αναλύσουν τους συσχετισμούς μεταξύ των κλάδων. Ο perceptron predictor ορίζει σε κάθε bit στην ιστορία κλάδων ένα βάρος. Όταν ένα ιδιαίτερο bit συσχετίζεται έντονα με μια ιδιαίτερη έκβαση κλάδων, το μέγεθος του βάρους είναι υψηλότερο από όταν δεν υπάρχει λιγότερος ή κανένας συσχετισμός. Κατά συνέπεια, ο perceptron predictor μαθαίνει να αναγνωρίζει τα bits στην ιστορία ενός ιδιαίτερου κλάδου που είναι σημαντικό για την πρόβλεψη, και μαθαίνει να αγνοεί τα ασήμαντα bits. Αυτή η ιδιοκτησία του perceptron predictor μπορεί να χρησιμοποιείται με τη σκιαγράφηση που παρέχει ανατροφοδότηση για άλλα σχέδια πρόβλεψης κλάδων.

4.5 Περίληψη

Σε αυτό το κεφάλαιο έχουμε εισαγάγει έναν νέο branch predictor που χρησιμοποιεί τα νευρωνικά δίκτυα perceptron πιο συγκεκριμένα ως βασικό μηχανισμό πρόβλεψης. Τα perceptrons είναι ελκυστικά επειδή μπορούν να χρησιμοποιήσουν τα μήκη μεγάλης ιστορίας (long history lengths) χωρίς απαίτηση των εκθετικών πόρων. Μια πιθανή αδυναμία των perceptrons είναι η αυξανόμενη υπολογιστική πολυπλοκότητά τους σε σύγκριση με τους two-bit counters, αλλά έχουμε επιδείξει πώς ένας perceptron predictor μπορεί να εφαρμοστεί αποτελεσματικά με τη χρησιμοποίηση των καθυστέρηση-κρύβοντας (delay-hiding) ιεραρχικών οργανώσεων. Μια άλλη αδυναμία των perceptrons είναι η ανικανότητά

τους να μάθουν γραμμικά τις αδιάσπαστες λειτουργίες. Εντούτοις, ο perceptron predictor αποδίδει καλά, επιτυγχάνοντας ένα χαμηλότερο ποσοστό misprediction, σε όλους τους προϋπολογισμούς υλικού, από τους γνωστούς σφαιρικούς predictors στην SPEC 2000 ακέραιων αριθμών. Οι κλάδοι που εκθέτουν γραμμικά το αναπόσπαστο (inseparability) είναι δύσκολο να προβλεφθούν γενικά, αλλά όχι πολύ δύσκολο για τα perceptrons.

Στην εισαγωγή, ο αναγνώστης μπορεί προς στιγμήν να είχε ανησυχήσει ότι η εποχή των προτάσεων για την αυξανόμενη πολυπλοκότητα πάνω στους branch predictors τελειώνει. Ο αναγνώστης μπορεί να είναι σίγουρος ότι , με τις ιεραρχικές οργανώσεις, οι ερευνητές είναι ελεύθεροι να ερευνήσουν τις ακριβέστερες και τις πιο εξωπραγματικές λύσεις στο πρόβλημα της αυξανόμενης ακρίβειας των branch predictor. Εντούτοις, πρέπει να αναρωτηθούμε πότε αυτή η προσέγγιση μπορεί να στηριχτεί αόριστα, και πότε υπάρχουν απλούστερες ιδέες ότι η διεύθυνση των branch predictors, η καθυστέρηση και η ακρίβεια χωρίς αύξηση της πολυπλοκότητας πρέπει να εξεταστεί από τη μικροαρχιτεκτονική.

4.6 Συμπεράσματα

Η πρόβλεψη κλάδων είναι ο σημαντικότερος περιοριστικός παράγοντας στον παραλληλισμό επιπέδων (Instruction Level Parallelism) οδηγίας σήμερα. Επομένως, η γρήγορη και η ακριβής πρόβλεψη διακλαδώσεων είναι κρίσιμη στο σχεδιασμό ενός επεξεργαστή. Αν και οι παλαιότεροι μετρητές bit έχουν την αξία δεδομένου ότι είναι πολύ γρήγοροι, δεν είναι τόσο ακριβείς όσο οι αναδυόμενοι perceptron predictors.

Επειδή οι μετρητές bit και οι perceptrons είναι βασισμένοι στις πολύ διαφορετικές τεχνολογίες, κάθε ένας χειρίζεται το aliasing και το συσχετισμό με διαφορετικούς τρόπους. Οι perceptrons είναι οι πιο αποτελεσματικοί, η εποπτευμένη τεχνική εκμάθησής τους είναι σε θέση να μάθει το συσχετισμό μεταξύ των κλάδων και να εκπαιδεύσει σε διάφορους κλάδους που μειώνουν αμέσως το λάθος από aliasing συγκρούσεις. Κατά συνέπεια, ο perceptron είναι ο ακριβέστερος διαθέσιμος predictor μέχρι και σήμερα.

Προτείναμε τα υβρίδια αυτών των δύο σχεδίων, δηλαδή τους μετρητές bit και perceptrons, θεωρώντας ότι αυτός ο συνδυασμός θα παρήγαγε έναν ακριβέστερο predictor. Δυστυχώς, οι μετρητές bit εμποδίζουν το perceptron με το να πάρει μαζί τα hardware bits για το perceptron στη χρήση και στο δόσιμο τους στο μετρητή bit στα υβρίδια, και με την πλειοψηφία του perceptron στις καταστάσεις ψήφου πλειοψηφίας. Αυτό υποβιβάζει την απόδοση των υβριδικών predictors, και τους καθιστά λιγότερο ακριβείς από έναν perceptron predictor, όπως φαίνεται σε όλα τα αποτελέσματά μας.

Αν και οι δοκιμές μας δεν είναι με κανένα τρόπο αποφασιστικές που κανένας μετρητής bit και perceptron υβρίδιο δεν θα εκτελεστεί τόσο καλά όσο το perceptron από έναν δεδομένο προϋπολογισμό υλικού (hardware budget), τα στοιχεία είναι ισχυρά ότι το perceptron θα ξεπεράσει τα περισσότερα υβρίδια.

Σε μελλοντική εργασία, μερικές περιοχές της βελτίωσης περιλαμβάνουν τη βελτίωση του perceptron, που αξιολογεί τους προϋπολογισμούς υλικού (hardware budget) για την εφαρμογή των υβριδίων που μπορεί να είναι οικονομικώς πιο αποδοτικοί από το perceptron, και τρέχοντας τις περισσότερες δοκιμές για την ολοκλήρωση της βελτιστοποίησης πάνω από τους υβριδικούς μας predictors.

5 Πόσο σημαντικοί είναι οι branch predictors

Χωρίς τη πρόβλεψη κλάδων, ο επεξεργαστής θα έπρεπε να περιμένει έως ότου η εντολή για το conditional jump έχει περάσει το στάδιο εκτέλεσης προτού να μπορέσει να εισαγάγει την επόμενη εντολή στο στάδιο προσκόμισης στη διασωλήνωση. Ο branch predictor προσπαθεί να αποφύγει αυτό το χάσιμο χρόνου με την προσπάθεια να υποθέσει εάν το υπό όρους άλμα είναι πλέον πιθανό να ληφθεί ή να μην ληφθεί. Ο κλάδος που υποθέτεται για να είναι ο πλέον πιθανός προσκομίζεται έπειτα, και εκτελείται υποθετικά. Εάν αργότερα ανιχνεύεται ότι η πρόβλεψη έκανε λάθος έπειτα οι υποθετικά εκτελεσμένες ή μερικώς εκτελεσμένες εντολές απορρίπτονται και η διασωλήνωση ξεκινά με το σωστό κλάδο, προκύπτει μια καθυστέρηση.

Ο χρόνος που σπαταλιέται σε περίπτωση misprediction κλάδων είναι ίσος με τον αριθμό των σταδίων της διασωλήνωσης από το στάδιο προσκόμησης μεχρι το στάδιο εκτελεσης. Οι σύγχρονοι μικροεπεξεργαστές τείνουν να έχουν τις αρκετά μακριές σωληνώσεις έτσι ώστε η καθυστέρηση misprediction να είναι μεταξύ 10 και 20 κύκλων ρολογιών. Όσο μεγαλύτερη είναι η διασωλήνωση τόσο υψηλότερη είναι η ανάγκη για έναν καλό branch predictor.

Τη πρώτη φορά που μια conditional jump εντολή αντιμετωπίζεται, δεν υπάρχουν πολλές πληροφορίες για να βασιστεί μια πρόβλεψη. Αλλά ο branch predictor διατηρεί αρχεία εάν οι κλάδοι λαμβάνονται ή δεν λαμβάνονται. Όταν αντιμετωπίζει ένα conditional jump που έχει δει αρκετές φορές νωρίτερα τότε μπορεί να βασίσει την πρόβλεψη στην ιστορία. Ο branch predictor μπορεί, παραδείγματος χάριν, να αναγνωρίσει ότι το conditional jump λαμβάνεται περισσότερο συχνά από το να μην λαμβάνεται, ή ότι λαμβάνεται κάθε δεύτερη φορά.

Το Branch target prediction δεν είναι το ίδιο με το branch prediction. Στην αρχιτεκτονική υπολογιστών, ένας Branch target predictor είναι το μέρος ενός επεξεργαστή που προβλέπει το στόχο ενός ληφθέντος conditional branch ή μιας unconditional branch εντολής προτού να υπολογιστεί ο στόχος της εντολής διακλάδωσης από τη μονάδα εκτέλεσης του επεξεργαστή. Η πρόβλεψη κλάδων προσπαθεί να υποθέσει εάν ένας υπό όρους κλάδος θα ληφθεί ή όχι.

Στα περισσότερα παράλληλα σχέδια επεξεργαστών, καθώς η λανθάνουσα κατάσταση εντολών αυξάνεται περισσότερο και το πλάτος προσκόμησης γίνεται ευρύτερο, η εξαγωγή στόχων κλάδων γίνεται bottleneck.

> Οι οδηγίες στο φραγμό ανιχνεύονται για να προσδιορίσουν τους κλάδους.

> Ο πρώτος προβλεφθείς ληφθείς κλάδος προσδιορίζεται.

> Ο στόχος εκείνου του κλάδου υπολογίζεται.

> Ο κύκλος προσκόμησης ξαναξεκινά στο στόχο διακλάδωσης (branch target).

Στις μηχανές όπου αυτή η επανάληψη παίρνει δύο κύκλους, η μηχανή χάνει έναν πλήρη κύκλο της προσκόμησης μετά από κάθε προβλεφθέντα ληφθέντα κλάδο. Καθώς οι προβλεφθέντες κλάδοι συμβαίνουν κάθε 10 εντολές, αυτό μπορεί να αναγκάσει μια ουσιαστική πτώση στο εύρος ζώνης. Μερικές μηχανές με μακροχρόνιες λανθάνουσες καταστάσεις θα είχαν μια ακόμα μεγαλύτερη απώλεια. Για να βελτιώσει την απώλεια, μερικές μηχανές εφαρμόζουν την πρόβλεψη στόχων κλάδων: λαμβάνοντας υπόψη τη διεύθυνση ενός κλάδου, προβλέπουν το στόχο εκείνου του κλάδου. Ένας καθαρισμός της ιδέας προβλέπει την έναρξη ενός

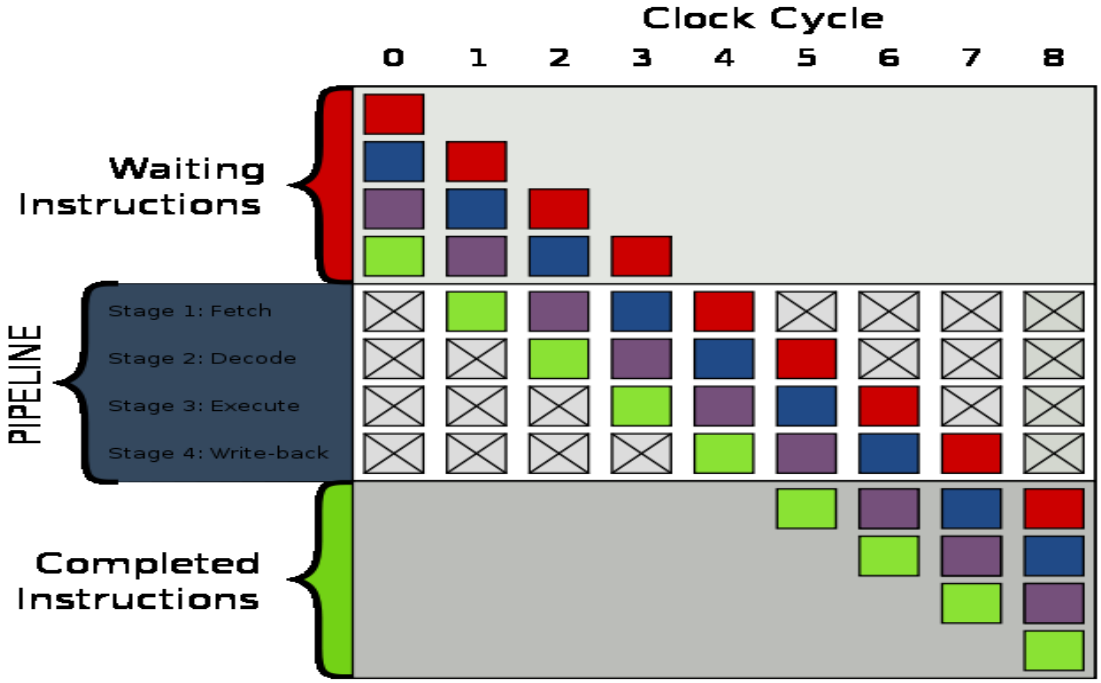
διαδοχικού τρεξίματος των εντολών δεδομένης της διεύθυνσης της έναρξης του προηγούμενου διαδοχικού τρεξίματος των εντολών.

Αυτός ο predictor μειώνει την επανάληψη ως εξής:

> Κατακερματίζοντας τη διεύθυνση της πρώτης οδηγίας στο τρέξιμο

> Προσκομίζοντας την πρόβλεψη για τις διευθύνσεις των στόχων των κλάδων σε εκείνο το τρέξιμο των οδηγιών

> Επιλέγοντας τη διεύθυνση που αντιστοιχεί στον προβλεφθέντα κλάδο που λαμβάνεται



Σχήμα 5: Διασωλήνωση 4-stage. Τα χρωματιστά κουτάκια αναπαριστουν εντολές αλληλοεξαρτόμενες.

Η κατανόηση πώς μπορούμε να μειώσουμε τη δύναμη των branch predictor απαιτεί μια προσεκτικότερη ματιά στην ελλοχεύουσα αρχή της λειτουργίας για τους στατικούς state-of-the-art predictors. Συγκεκριμένα, οι σύγχρονοι predictors είναι βασισμένοι στην ιστορία. Καταγράφουν τη συμπεριφορά των κλάδων υποθέτοντας σιωπηρά ότι εκείνη η προηγούμενη συμπεριφορά είναι ένας καλός δείκτης της μελλοντικής συμπεριφοράς. Οι επεξεργαστές συμπληρώνουν τους πίνακες των branch predictor μόλις αποφασίζεται η έκβαση μιας εντολής κλάδων αποτελεσματικά μαθαίνοντας πώς ο συγκεκριμένος κλάδος συμπεριφέρεται (αυτό καλείται φάση εκμάθησης). Κατόπιν, υπό τον όρο ότι ο κλάδος εκθέτει σχετικά σταθερή συμπεριφορά, η υψηλά ακριβής πρόβλεψη είναι δυνατή, απλά με το να ανατρέξει στις προηγούμενες εκβάσεις.

Μια βασική παρατήρηση για αυτήν την εργασία είναι ότι υπάρχουν πολλοί καλά συμπεριφερόμενοι κλάδοι για τους οποίους μόλις τελειώνει η φάση εκμάθησης, οι συλλεχθείσες πληροφορίες παραμένουν ουσιαστικά αμετάβλητες για μια περίοδο πολύ πιο μεγάλη από τη φάση εκμάθησης. Αναφερόμαστε σε αυτή τη φάση μετα-εκμάθησης ως φάση σταθερής κατάστασης (steady state phase). Τα υπάρχοντα σχέδια στηρίζονται στη συνεχή χρήση όλων των predictor δομών ακόμη και στη σταθερή κατάσταση. Αυτό είναι περιττό. Με την ανίχνευση αυτών των κλάδων είναι πιθανόν να αποφύγουμε την ενημέρωση του συνδυασμένου predictor. Επιπλέον, επειδή τα χαρακτηριστικά προγράμματα εκθέτουν τη χρονική τοποθεσία στο ρεύμα κλάδων

τους, το να προβλέψει κλάδους που συμπεριφέρονται καλά πρέπει να είναι δυνατόν μέσω μιας μικρής δομής χωρίς απώλεια της ακρίβειας.

Εκτός από τις ενεργειακές ανεπάρκειες στο χειρισμό των κλάδων με μια μακροχρόνια φάση σταθερής κατάστασης οι υπάρχοντες predictor είναι ανεπαρκείς στο τρόπο που χειρίζονται μια άλλη κατηγορία εντολών κλάδων. Συγκεκριμένα ,
ενώ μερικοί κλάδοι δεν εκθέτουν τις μακροχρόνιες φάσεις σταθερού κόστους η συμπεριφορά τους είναι αρκετά απλή ώστε να συλληφθεί από έναν μόνο από τους δύο ελλοχεύοντες predictors. Ο προσδιορισμός τέτοιων κλάδων θα επέτρεπε σε μας να αποφύγουμε δύο από τους τρεις ελέγχους που μειώνουν έτσι τη δύναμη ακόμα και όταν ένας κλάδος δεν συμπεριφέρεται καλά.

Λεπτομερέστερα, το ξεπέρασμα της ισχύς (power overheads) που υφίστανται από τους υπάρχοντες συνδυασμένους branch predictors είναι:

a) Ο συνδυασμένος predictor χρησιμοποιεί τρεις ελλοχεύοντες υπο-predictors για να επιτύχει την υψηλή ακρίβεια πρόβλεψης. Δύο από τους υπο-predictors παράγουν τις προβλέψεις για τους κλάδους και είναι χαρακτηριστικά συντονισμένοι για τις διαφορετικές συμπεριφορές κλάδων. Ο τρίτος υπο-predictor είναι ο επιλογέας και κρατά τη διαδρομή των δύο υπο-predictors που εργάζονται καλύτερα ανά κλάδο. Οι χαρακτηριστικές διαμορφώσεις, χρησιμοποιούν τους bi-modal predictors για έναν από τους υπο-predictors, τον επιλογέα, και έναν βασισμένο στο σχέδιο predictor όπως το gshare για τον τελευταίο υπο-predictor. Οι υπο-predictor χρησιμοποιούν έναν saturating μετρητή στις πληροφορίες αρχείων. Ένας χαρακτηριστικός μηχανισμός εκμάθησης είναι η αύξηση ή η μείωση του σχετικού μετρητή εάν ο κλάδος λαμβάνεται ή όχι. Για να μειώσουν τον αριθμό των bits που απαιτούνται, χρησιμοποιούνται οι μικροί μετρητές (π.χ., μετρητές 2 bits). Μόλις η τιμή του μετρητή φτάσει στο μέγιστο οι εκβάσεις των ληφθέντων κλάδων (taken) δεν θα αυξήσουν πλέον το μετρητή. Ομοίως, μόλις φθάσει ο μετρητής στο ελάχιστό του, οι εκβάσεις των μη ληφθέντων κλάδων (not taken) δεν θα αλλάξουν τη κατάσταση του predictor. Αργότερα, οι μετρητές εξετάζονται για να προβλέψουν την έκβαση κλάδων. Εάν η τιμή του μετρητή είναι περισσότερο από ένα κατώτατο όριο (π.χ., ένα για τους 2 bits μετρητές) ο κλάδος προβλέπεται παρμένος (taken). Διαφορετικά, ο κλάδος προβλέπεται να μην ληφθεί.
b) Οι εντολές κλάδων εκθέτουν ισχυρή χρονική τοποθεσία. Δηλαδή κοιτάζοντας κατά τη διάρκεια μικρών χρονικών διαστημάτων, υπάρχει ένα μικρό σύνολο κλάδων που αποτελούν τη μεγάλη πλειοψηφία των προβλέψεων. Έχουμε παρατηρήσει ότι κατά μέσον όρο, περίπου 83% των κλάδων εμφανίζονται μέσα στους τελευταίους 64 κλάδους που προσκομίζονται.
c) Μερικοί κλάδοι τείνουν να χρησιμοποιήσουν τον ίδιο υπο-predictor επανειλημμένα στο συνδυασμένο predictor. Κατά μέσον όρο, 95% του χρόνου, οι εντολές διακλαδώσεων χρησιμοποιούν τον ίδιο υπο-predictor που χρησιμοποίησαν την τελευταία φορά που αντιμετωπίστηκαν.
d) Τέλος, κατά το κύκλο προσκόμισης έχουμε πρόσβαση στο BTB για να ελέγξουμε εάν περιέχει τη διεύθυνση κλάδων. Αυτό απαιτεί τις αποθηκευμένες (storing) διευθύνσεις κλάδων και τη διεύθυνση στόχων τους στον buffer. Συνεπώς, ενημερώνουμε το BTB συχνά και μόλις ξέρουμε τη διεύθυνση στόχων είμαστε σίγουροι ότι ο κλάδος λαμβάνεται. Εντούτοις, η μελέτη μας δείχνει ότι περισσότερο από 99% των αναπροσαρμογών του BTB είναι περιττά δεδομένου ότι η σχετική διεύθυνση κλάδων αποθηκεύεται ήδη

στο BTB. Εντούτοις, πολλοί επεξεργαστές, αποκτούν πρόσβαση στο BTB για κάθε ληφθέντα κλάδο.

5.1 Motivation (Κίνητρο)

° Η πρόβλεψη αξίας μπορεί να σπάσει τις αλυσίδες εξάρτησης, που αυξάνουν τον παραλληλισμό επιπέδων οδηγίας (instruction level parallelism ILP).

° Οι τεχνικές πρόβλεψης αξίας υλικού είναι αποτελεσματικές αλλά ακριβός λόγω του μεγάλου ποσού ο predictor δηλώνει που πρέπει να αποθηκευτεί.

° Τέλος η πρόβλεψη αξίας λογισμικού είναι ανεπαρκής επειδή δεν υπάρχει καμία εμπιστοσύνη. Μόνο οι πιο προβλέψιμοι υποψήφιοι μπορούν να επιλεχτούν, μειώνοντας την κάλυψη.

5.2 Λύσεις

• Η πρόσθεση εμπιστοσύνης στον compiler με την ελεγχόμενη τιμή πρόβλεψης και με τη χρησιμοποίηση της υπάρχουσας κάλυψης αυξήσεων predictor αυξάνει τη κάλυψη (coverage) κρατώντας την ακρίβεια υψηλή.
• Η καθοδηγημένη σκιαγράφηση επιλογής υποψηφίων βασισμένη στην κριτική διάθεση των οδηγιών στο δυναμικό παράθυρο οδηγίας εκθέτει τον πρόσθετο παραλληλισμό.
• Η εφαρμογή ενός νέου κλάδου που αγνοεί τα mispredictions κατά το εκτέλεση μιας μη-βελτιστοποιημένης, αλλά σωστής, διαδρομής , μειώνει τα mispredictions κλάδων.

5.3 Συμπεράσματα

• Η βασισμένη στον predictor εμπιστοσύνη κλάδων βελτιώνει την κάλυψη και την απόδοση για την μεταγλώττιση-ελεγχόμενη πρόβλεψη αξίας.
• Η πρόβλεψη αξίας μόνο στο λογισμικό εκθέτει το μέτριο speedup , σε σημαντικές βελτιώσεις πιθανόν με την ελάχιστη υποστήριξη υλικού ώστε να μειώσουν τα γενικά έξοδα.
• Η επιλογή αξίας των υποψηφίων πρόβλεψης βασισμένων στην κριτική διάθεση δίνει περισσότερη απόδοση, αλλά μόνο στα προγράμματα με τις μακριές δυναμικές αλυσίδες εξάρτησης
• Μια απλή βελτιστοποίηση κλάδων αποβάλλει τις ποινές για τα αβλαβή mispredictions.

6 Οι Denial of Service Επιθέσεις.

Μια από τις επικινδυνότερες επιθέσεις που πραγματοποιούνται σε δίκτυα υπολογιστών είναι οι επιθέσεις άρνησης εξυπηρέτησης (Denial of Service ή DoS). Σε αυτή την κατηγορία επιθέσεων, ο επιτιθέμενος δεν εκμεταλλεύεται αδυναμίες προγραμμάτων ή πρωτοκόλλων για να διεισδύσει σε ένα δίκτυο ή υπολογιστικό σύστημα. Αντίθετα, χρησιμοποιεί όλα τα νόμιμα μέσα που του παρέχει το δίκτυο ή το υπολογιστικό σύστημα σε τόσο μεγάλο βαθμό έτσι ώστε κανείς άλλος χρήστης να μην μπορεί να τα χρησιμοποιήσει. Ο σκοπός δηλαδή σε μια επίθεση τύπου DoS είναι να αποτρέψει τους χρήστες από το να χρησιμοποιήσουν τις υπηρεσίες ενός διακομιστή. Αυτός είναι και ο λόγος για τον οποίο οι επιθέσεις αυτές είναι τόσο δύσκολο να εντοπιστούν και να αποτραπούν. Επιπλέον, με τη συνεχή αύξηση των πόρων του δικτύου ιδιαίτερα προς τους τελικούς χρήστες, τέτοιου είδους επιθέσεις γίνονται όλο και πιο συνηθισμένες.

6.1 Ιστορία των DoS επιθέσεων.

Στα μέσα της δεκαετίας του 90 αρχίζουν να κάνουν την εμφάνιση τους οι πρώτες επιθέσεις DoS. Για να τρέχεις τα κατάλληλα προγράμματα χρειαζόσουν έναν καλό υπολογιστή και κάποιο γρήγορο δίκτυο όποτε οι περισσότεροι χρησιμοποιούσαν το δίκτυο των πανεπιστήμιων.

Αργότερα το 1996 ανακαλύφθηκε μια “τρυπά” στο TCP/IP πρωτόκολλο που επέτρεπε τον μεγάλο αριθμό SYN πακέτων(SYN flood).

Το 1997 μεγάλες Dos επιθέσεις ξεκινάνε να γίνονται σε IRC δίκτυα. Σε μια επίθεση ατέλειες σε windows συστήματα ο επιτιθέμενος μπορούσε απευθείας να crashαρει στα συστήματα IRC χρηστών με προγράμματα όπως το teardrop, boink, bonk. Τα προβλήματα αυτά διορθώθηκαν με διάφορα patches αλλά και άλλες τεχνικές ανακαλύφθηκαν όπως η Smurf attack. Και ενώ μέχρι εκείνη την στιγμή ο αποστολέας εκμεταλλευόταν κάποιο πρόβλημα αργότερα απλά έστελναν πολλά πακέτα σε κάποιον χρήστη. Αν ο χρήστης χρησιμοποιούσε κάποια dial-up σύνδεση και ο αποστολέας μπορούσε να χρησιμοποιήσει το δίκτυο κάποιο πανεπιστημίου μπορούσαν να στείλουν πακέτα προκαλώντας DoS. Το 1998 ενώ οι συνδέσεις άρχιζαν να μεγαλώνουν, οι συνδέσεις και οι υπολογιστές να γίνονται πιο γρήγοροι, έτσι οι επιθέσεις άρχισαν να γίνονται πιο συχνές. Αργότερα εμφανίστηκε ένα άλλο είδος DoS επιθέσεων, οι DDoS επιθέσεις όπου εκεί χρησιμοποιούνταν μεγάλα δίκτυα υπολογιστών για να σταλούν τα πακέτα.

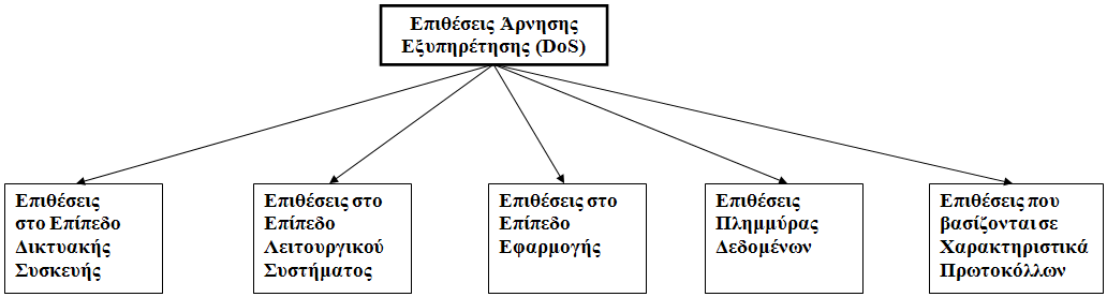
Ενδεικτικά αναφέρονται κάποιες DDoS γνωστές επιθέσεις :

- Οι επιθέσεις σε μεγάλες εταιρίες (Yahoo, eBay, Buy.com, Amazon.com) τον Ιανουάριο του 2000 που τέθηκαν εκτός λειτουργίας για μερικές ώρες.
- Η επίθεση στον βρετανικό παροχέα υπηρεσιών Cloud Nine που οδήγησε σε ολική διακοπή εργασιών στις 22 Ιανουαρίου 2002.
- Η επίθεση του Οκτωβρίου 2002 στους Εξυπηρετητές Δικτυακής Ονοματολογίας Ρίζας (Root Name Servers) πού απέτυχε λόγω υπερεπάρκειας πόρων , άλλα θα μπορούσε να οδηγήσει σε διακοπή παροχής υπηρεσιών DNS σε όλο το διαδίκτυο.

Πρώτη Φάση (δεκαετίατου'90) : DoS Επιθέσεις Άρνησης Υπηρεσίας •Αρχικά εκμετάλλευση προβλημάτων (bugs) ή αδυναμιών λογισμικού •Πρώτοι στόχοι: Single hosts -single services •Σε κάποιες περιπτώσεις αρκεί ένα μοναδικό, κατάλληλα κατασκευασμένο, πακέτο
Δεύτερη Φάση (1996-2000) •Κλήσεις εξυπηρέτησης από πολλές πηγές για κατανάλωση πόρων •Οι υποδομές του Internet χρησιμοποιούνται για "ενίσχυση" της έντασης των επιθέσεων
Τρίτη Φάση (μετά το2000) : Distributed DoS Κατανεμημένες Επιθέσεις Άρνησης Υπηρεσίας •Στόχο αποτελεί το δικτυακό εύρος (Bandwidth) •Χρήση πολλαπλών ελεγχόμενων υπολογιστών, σε πολλαπλά στάδια επίθεσης με κλιμάκωση της επίθεσης

6.2 Κατηγοριοποίηση Επιθέσεων DoS.

Οι επιθέσεις DoS μπορούν να κατηγοριοποιηθούν σε πέντε κατηγορίες με βάση το επίπεδο του πρωτοκόλλου στο οποίο πραγματοποιείται η επίθεση, όπως απεικονίζεται στο παρακάτω σχήμα :



Κατηγοριοποίηση Επιθέσεων Άρνησης Εξυπηρέτησης

●Οι επιθέσεις Άρνησης Εξυπηρέτησης (DoS) στο Επίπεδο Δικτυακής Συσκευής (Network Device Level) περιλαμβάνουν επιθέσεις που μπορεί να προκληθούν είτε αν ο επιτιθέμενος εκμεταλλευτεί λάθη ή αδυναμίες στο λογισμικό, είτε αν προσπαθήσει να εξαντλήσει τους υλικούς πόρους των δικτυακών συσκευών. Ένα παράδειγμα μίας αδυναμίας συσκευών δικτύου είναι αυτό που προκαλείται από ένα σφάλμα υπερχείλισης μνήμης στη διαδικασία ελέγχου των συνθηματικών. Εκμεταλλευόμενοι τέτοιου είδους αδυναμίες συγκεκριμένοι δρομολογητές Cisco 7xx

μπορούν να διακόψουν την λειτουργία τους αν ο επιτιθέμενος συνδεθεί με τους δρομολογητές μέσω telnet και εισάγει ιδιαίτερα μεγάλα συνθηματικά.

●Στο επίπεδο Λειτουργικού Συστήματος (OS level) οι επιθέσεις DoS εκμεταλλεύονται τους τρόπους με τους οποίους τα λειτουργικά συστήματα υλοποιούν τα διάφορα πρωτόκολλα. Ένα παράδειγμα αυτής της κατηγορίας επιθέσεων DoS είναι η επίθεση Ping of Death. Σε αυτή την επίθεση, στέλνονται στο θύμα στόχο αιτήσεις ηχούς ICMP που έχουν συνολικό μέγεθος δεδομένων μεγαλύτερο από το μέγιστο μέγεθος με βάση το πρότυπο IP. Όταν στέλνονται τέτοιου είδους πακέτα τμηματοποιούνται και στη συνέχεια επανενώνονται στον προορισμό. Πολλά λειτουργικά συστήματα όμως αποτυγχάνουν να δεσμεύσουν αρκετή μνήμη για τα υπερμεγέθη επανενωμένα πακέτα ICMP με αποτέλεσμα την υπερχείλιση της προσωρινής μνήμης.

●Οι επιθέσεις στο επίπεδο εφαρμογής (application-based attacks) προσπαθούν να θέσουν μία μηχανή ή μία υπηρεσία εκτός λειτουργίας είτε εκμεταλλευόμενοι συγκεκριμένα λάθη στις εφαρμογές δικτύων που "τρέχουν" στον κόμβο στόχο, είτε χρησιμοποιώντας τέτοιες εφαρμογές προκειμένου να εξαντλήσουν τους πόρους του θύματός τους. Είναι επίσης πιθανό ο επιτιθέμενος να βρει σημεία υψηλής αλγοριθμικής πολυπλοκότητας και να τα εκμεταλλευτεί προκειμένου να καταναλώσει όλους τους διαθέσιμους πόρους σε έναν απομακρυσμένο κόμβο. Ένα παράδειγμα επίθεσης που βασίζεται στο επίπεδο εφαρμογής είναι η επίθεση finger bomb. Ένας κακόβουλος χρήστης μπορεί να προκαλέσει την επαναλαμβανόμενη εκτέλεση της ρουτίνας finger στον κόμβο-θύμα, οδηγώντας πιθανότατα στην εξάντληση των πόρων των δικτύων.

●Στις επιθέσεις πλημμύρας δεδομένων (data flooding), ο επιτιθέμενος προσπαθεί να χρησιμοποιήσει το διαθέσιμο εύρος ζώνης σε έναν κόμβο ή συσκευή δικτύου στο μεγαλύτερο δυνατό βαθμό, στέλνοντας μαζικές ποσότητες δεδομένων και προκαλώντας την επεξεργασία ιδιαίτερα μεγάλων ποσοτήτων δεδομένων.

●Οι επιθέσεις DoS που βασίζονται σε χαρακτηριστικά πρωτοκόλλων εκμεταλλεύονται συγκεκριμένα χαρακτηριστικά των πρωτοκόλλων. Για παράδειγμα διάφορες επιθέσεις εκμεταλλεύονται το γεγονός ότι μπορεί να παραποιηθούν οι διευθύνσεις πηγής IP. Διάφορα είδη επιθέσεων DoS έχουν επικεντρωθεί στην υπηρεσία διάθεσης ονομάτων και διευθύνσεων που χρησιμοποιούνται στο Διαδίκτυο (Domain Name Service (DNS)). Πολλές από αυτές περιλαμβάνουν την επίθεση στη γρήγορη μνήμη των εξυπηρετητών ονομάτων. Ένα πρόβλημα που υπάρχει σε πολλές υλοποιήσεις των DNS, είναι ότι δεν ελέγχεται η ορθότητα των απαντήσεων που λαμβάνουν σε αιτήσεις. Ένας παραβιασμένος εξυπηρετητής ονομάτων μπορεί να ανταποκριθεί σε μία αίτηση με ψευδείς πληροφορίες, οι οποίες μπορούν να αποθηκευτούν στον εξυπηρετητή ονομάτων που λαμβάνει την απάντηση της αίτησης. Ένας επιτιθέμενος που έχει παραβιάσει έναν εξυπηρετητή ονομάτων μπορεί να αναγκάσει ένα θύμα να αποθηκεύει λανθασμένες εγγραφές ρωτώντας το θύμα για το δικτυακό τόπο του ίδιου του επιτιθέμενου. Αυτό θα έχει σαν αποτέλεσμα ένα ευπαθές θύμα εξυπηρετητή ονομάτων να αναφέρεται στον απατεώνα εξυπηρετητή και θα αποθηκεύει την απάντηση, η οποία πιθανότατα θα είναι πλαστή.

6.3 Κίνητρα των Επιθέσεων DoS και Προβλήματα αντιμετώπισής τους.

Υπάρχουν πολλά κίνητρα για την πραγματοποίηση επιθέσεων DoS. Συγκεκριμένα άτομα συχνά εκκινούν επιθέσεις DoS προκειμένου να τραβήξουν την προσοχή και να γίνουν δημοφιλείς. Άλλες επιθέσεις έχουν πολιτικά κίνητρα. Ιστοσελίδες που ανήκουν σε επίμαχες οντότητες συχνά έγιναν στόχοι επιθέσεων άρνησης εξυπηρέτησης. Προσωπικοί λόγοι είναι ένα άλλο κίνητρο για τις επιθέσεις DoS. Άλλα άτομα μπορεί να πραγματοποιήσουν επιθέσεις με σκοπό να προκληθεί κάποια ταπείνωση ή απλά σαν αστείο. Αυτές οι επιθέσεις γενικά δεν είναι πολύ ισχυρές και συνήθως δεν διαρκούν πολύ. Οι επιθέσεις DoS έχουν κάποια χαρακτηριστικά που κάνουν ακόμα πιο δύσκολη την αντιμετώπισή τους.
Για αυτό το λόγο, στη συνέχεια παρουσιάζουμε κάποια θέματα που εξηγούν γιατί η προστασία από τις επιθέσεις DoS είναι πολύ δύσκολη.

Η ασφάλεια του Διαδικτύου είναι αλληλεξαρτώμενη: Το Διαδίκτυο έχει λίγους ενσωματωμένους μηχανισμούς προστασίας προκειμένου να αντιμετωπιστούν οι επιθέσεις DoS. Ο σχεδιασμός τους δημιουργεί κενά ασφάλειας τα οποία μπορεί να εκμεταλλευτούν οι επιτιθέμενοι. Είναι σημαντικό να σημειώσουμε ότι ανεξάρτητα από το πόσο ασφαλής είναι ένας κόμβος, είναι πάντα υπό απειλή αφού το υπόλοιπο Διαδίκτυο δεν είναι ασφαλές.

Οι επιθέσεις DoS είναι από τη φύση τους δύσκολο να ανιχνευθούν: Η ανίχνευση της πηγής των επιθέσεων DoS είναι αρκετά δύσκολη. Εκμεταλλευόμενοι την ασταθή φύση του Διαδικτύου, οι επιτιθέμενοι χρησιμοποιούν παραποιημένες διευθύνσεις πηγής IP προκειμένου να κρύψουν την ταυτότητα τους πίσω από άλλες μηχανές που έχουν θέσει υπό τον έλεγχο τους. Επιπλέον, οι ροές των πακέτων DoS δεν παρουσιάζουν κοινά χαρακτηριστικά, με αποτέλεσμα να καθιστούν ιδιαίτερα δύσκολη την ανίχνευση τους και ακόμα πιο δύσκολη τη διαφοροποίηση των πακέτων επίθεσης από τα νόμιμα πακέτα.

Περιορισμένοι πόροι: Ο υψηλός ρυθμός πακέτων ο οποίος χρειάζεται για να δημιουργηθούν μαζικές επιθέσεις DoS απαιτεί μεγάλο αριθμό πόρων. Τα συστήματα και τα δίκτυα που αποτελούν το Διαδίκτυο έχουν περιορισμένους πόρους οι οποίοι μπορεί εύκολα να εξαντληθούν κατά τη διάρκεια της ανίχνευσης των επιθέσεων.

Αυτοματοποιημένα εργαλεία: Τα εργαλεία DoS τα οποία είναι διαθέσιμα στο Διαδίκτυο συνοδεύονται από οδηγίες οι οποίες επιτρέπουν την εύκολη και αποτελεσματική χρήση τους ακόμα και από όχι τεχνικά καταρτισμένους χρήστες. Οι επιτιθέμενοι συνεχώς προσπαθούν να αναπτύξουν πιο αποτελεσματικά εργαλεία προκειμένου να ξεπεράσουν τα συστήματα ασφαλείας που αναπτύσσονται από τους ερευνητές.

Ένα περιβάλλον γεμάτο στόχους: Υπάρχει ένας μεγάλος αριθμός κόμβων και δικτύων στο Διαδίκτυο που είναι ευπαθή, τα οποία μπορεί να τα εκμεταλλευτούν και τα οποία παρέχουν γόνιμο έδαφος προκειμένου να πραγματοποιηθούν επιθέσεις DoS. Υπάρχουν επίσης πολλοί χρήστες του Διαδικτύου οι οποίοι δεν έχουν την απαιτούμενη τεχνική κατάρτιση προκειμένου να προστατέψουν τα συστήματά τους από επιθέσεις DoS. Επιπλέον, ο σχεδιασμός ενός αποτελεσματικού συστήματος αμύνης απέναντι στις επιθέσεις DoS αντιμετωπίζει πολλές προκλήσεις, γιατί οι απαιτήσεις για μία αποτελεσματική απόκριση στις επιθέσεις DoS είναι πολλαπλές:

• Ένα από τα κύρια χαρακτηριστικά των συστημάτων προστασίας απέναντι στις επιθέσεις DoS είναι η υψηλή ασφάλεια. Πρέπει να επιβεβαιωθεί ότι το σύστημα προστασίας δεν μπορεί να χρησιμοποιηθεί σαν θύμα μίας επίθεσης DoS.

• Ένα σύστημα προστασίας απέναντι στις επιθέσεις DoS πρέπει να είναι αξιόπιστο στην ανίχνευση επιθέσεων DoS και να μην εμφανίζει λανθασμένους θετικούς συναγερμούς. Αυτό μπορεί να έχει σαν αποτέλεσμα υψηλό κόστος, επομένως ίσως θα ήταν καλύτερο να υπάρχει μεγάλη αυστηρότητα ως προς αυτή την απαίτηση.

• Ένα σύστημα προστασίας απέναντι στις επιθέσεις DoS πρέπει να είναι αποτελεσματικό στην ανίχνευση και την απόκριση σε μία επίθεση DoS προκειμένου να περιορίσει την αποτελεσματικότητα της επίθεσης.

• Ένας μηχανισμός προστασίας από επιθέσεις DoS πρέπει να είναι ρεαλιστικός στο σχεδιασμό του και να μπορεί να εφαρμοστεί στις υπάρχουσες υποδομές ασφάλειας, χωρίς να απαιτεί σημαντικές αλλαγές στην υποδομή του Διαδικτύου.

• Ένας μηχανισμός προστασίας από επιθέσεις DoS δεν πρέπει να απαιτεί πολλούς πόρους και πρέπει να έχει μειωμένο κόστος απόδοσης, προκειμένου να αποφύγει τη μείωση της απόδοσης του δικτύου το οποίο δέχεται την επίθεση.

6.4 Απλές DoS Επιθέσεις.

Στόχος των DoS επιθέσεων είναι να αποτρέψουν την πρόσβαση σε υπηρεσίες και πόρους κάποιου εξυπηρετητή (server) από εξουσιοδοτημένους χρήστες. Η επίθεση στον εξυπηρετητή-θύμα επιτυγχάνεται συνήθως με την συνεχή αποστολή σε αυτόν πακέτων δεδομένων. Τα πακέτα μεταδίδονται σε υψηλούς ρυθμούς, έτσι ώστε ο εξυπηρετητής να μην δύναται να ανταποκριθεί στον μεγάλο φόρτο εργασίας και να καταρρεύσει (crash).

Οι DoS επιθέσεις λαμβάνουν χώρα στο Διαδίκτυο, επομένως χρησιμοποιούν το πρωτόκολλο IP ως πρωτόκολλο επιπέδου δικτύου. Αντίθετα, στο επίπεδο μεταφοράς χρησιμοποιούνται πρωτόκολλα που ποικίλουν ανάλογα με το είδος της επίθεσης. Τα πρωτόκολλα ICMP, TCP και UDP είναι αυτά που χρησιμοποιούνται συνήθως και επομένως μπορούμε να διαχωρίσουμε τις επιθέσεις σε ICMP, TCP και UDP επιθέσεις, ανάλογα με το πρωτόκολλο επιπέδου μεταφοράς που χρησιμοποιούν. Μερικοί από τους πιο γνωστούς τρόπους DoS επιθέσεων είναι οι ακόλουθοι :

1. Ping of death.
2. ICMP flood.
3. Smurf attack.
4. TCP SYN flood.
5. UDP flood.
6. Teardrop attack.
7. Fork Bombs.

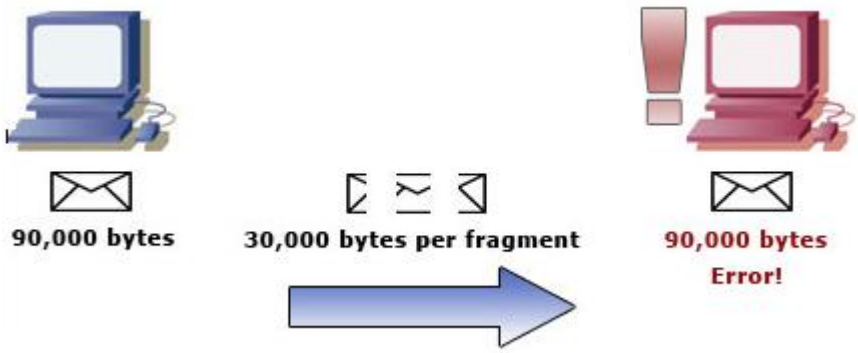
8. Επιθέσεις τύπου Web DoS.
9. Email bomb.
10. DNS amplification attack.

6.4.1 Ping of death.

Το Ping of Death είναι ένας τύπος επίθεσης σε έναν ηλεκτρονικό υπολογιστή. Η επίθεση Ping of Death συντελείται όταν ένας ηλεκτρονικός υπολογιστής στέλνει κακοσχηματισμένα πακέτα ping σε έναν άλλο υπολογιστή με σκοπό να τον θέσει εκτός λειτουργίας.

Ένα πακέτο ping έχει κανονικά μέγεθος 64 bytes (ή 84 bytes εάν προστεθεί και η κεφαλίδα που προσθέτει το πρωτόκολλο IP). Πολλοί τύποι ηλεκτρονικών υπολογιστών δεν μπορούν να χειριστούν πακέτα ping που έχουν μέγεθος μεγαλύτερο από 65535 bytes, δηλαδή το μέγιστο επιτρεπτό από το πρωτόκολλο IP. Κατά συνέπεια, η επίθεση Ping of Death περιλαμβάνει την συνεχή αποστολή μεγάλων πακέτων ping σε κάποιον υπολογιστή μέχρι ο τελευταίος να τεθεί εκτός λειτουργίας.

Σύμφωνα με τα πρωτόκολλα του διαδικτύου, η αποστολή ενός πακέτου ping μεγαλύτερου των 65535 bytes είναι παράνομη και δεν προβλέπεται, δεδομένου ότι στην κεφαλίδα IP προβλέπονται μονάχα 16 bits για την καταχώρηση του μεγέθους του πακέτου (2^16-1 = 65535). Παρόλα αυτά ένας υπολογιστής μπορεί να σπάσει το πακέτο ping σε δύο τμήματα και να το στείλει ως δύο ξεχωριστά πακέτα IP.



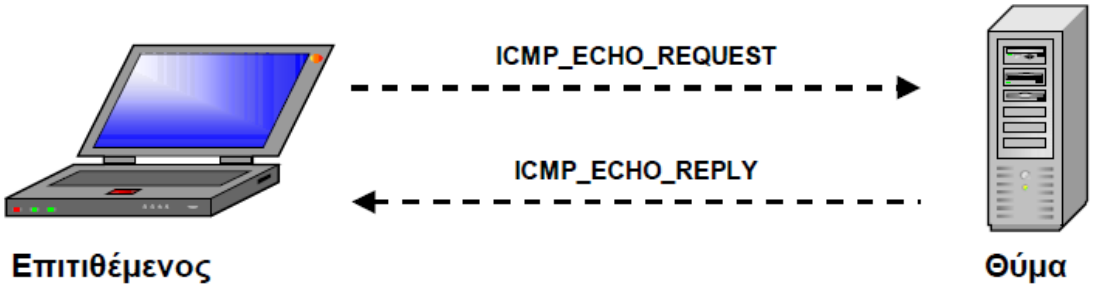
Παράδειγμα Ping of Death attack

Όταν ο υπολογιστής-στόχος παραλάβει τα δύο πακέτα, θα τα συνθέσει και θα δημιουργήσει ένα μεγάλο πακέτο ping, το οποίο στην συνέχεια ενδέχεται να δημιουργήσει σφάλματα του τύπου buffer overflow, τα οποία συνήθως οδηγούν σε δυσλειτουργία ολόκληρου του υπολογιστή (computer crash).

6.4.2 ICMP flood.

Τα ICMP (Internet Control Message Protocol) πακέτα μεταφέρουν ειδικά μηνύματα ελέγχου που χρησιμοποιούνται από το δίκτυο για θέματα συνδεσιμότητας. Όταν εκτελείται μια εντολή ping στέλνονται στον παραλήπτη ICMP πακέτα με κωδικό «ECHO_REQUEST» και ο παραλήπτης απαντάει με μηνύματα «ECHO_REPLY».

Όταν εκτελείται μια «ICMP flood» επίθεση, ο εξυπηρετητής-θύμα «βομβαρδίζεται» με «ECHO_REQUEST» πακέτα απασχολώντας τον από την ωφέλιμη εργασία του, αφού θα πρέπει να απαντάει με «ECHO_REPLY» μηνύματα για κάθε «ECHO_REQUEST» που λαμβάνει.

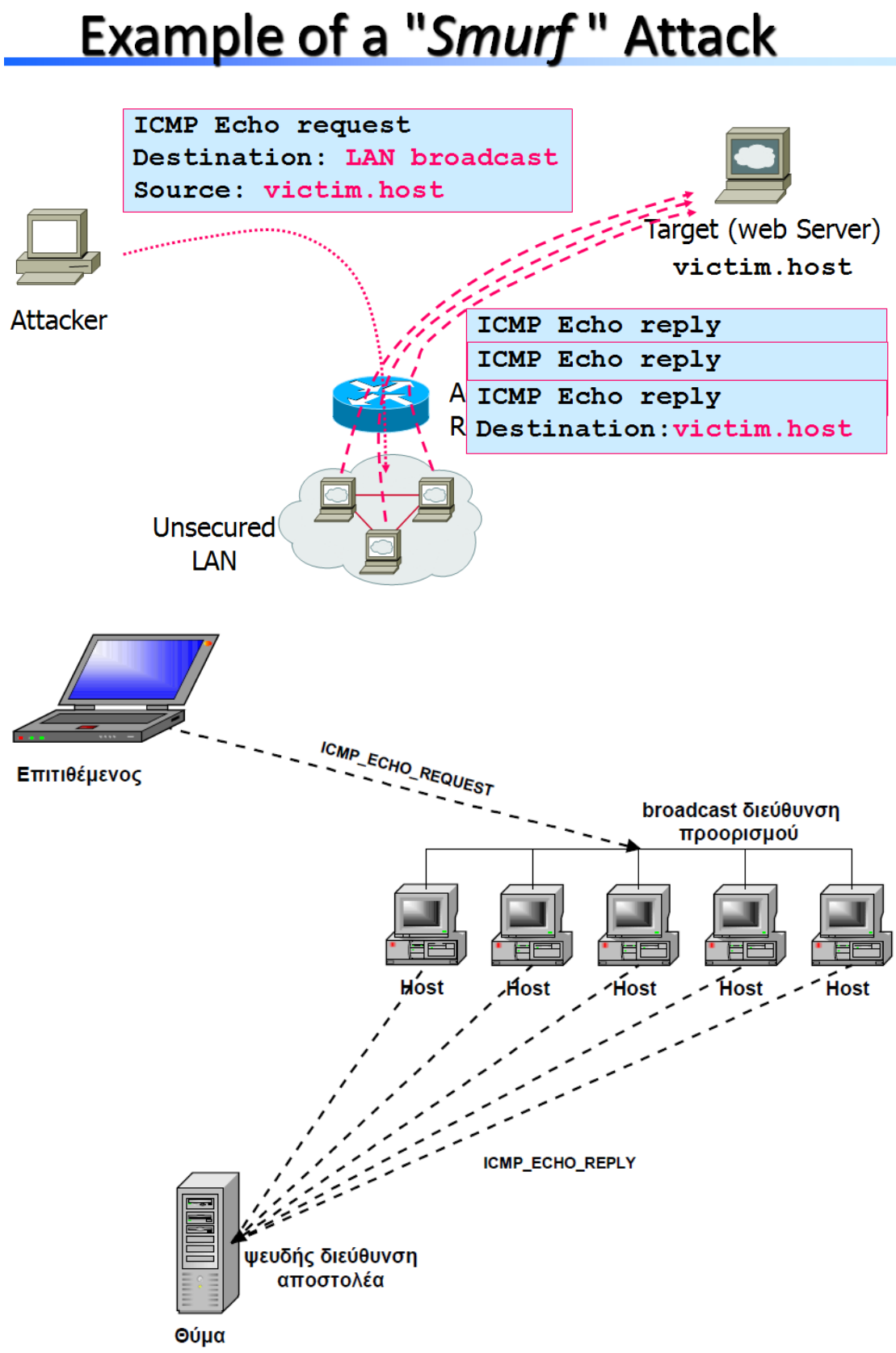


Παράδειγμα ICMP flood attack

Η επίθεση μπορεί να έχει αποτέλεσμα μόνο εφόσον το εύρος ζώνης μεταξύ επιτιθέμενου και θύματος είναι αρκετά μεγάλο.

6.4.3 Smurf attack.

Οι «Smurf» επιθέσεις είναι όμοιες με τις «ICMP flood» επιθέσεις, με τη διαφορά ότι χρησιμοποιούν broadcast διευθύνσεις για τον παραλήπτη των πακέτων και ψευδείς (spoofed) διευθύνσεις για τον αποστολέα.
Συγκεκριμένα, στέλνονται «ECHO_REQUEST» πακέτα σε broadcast διευθύνσεις, χρησιμοποιώντας ως διεύθυνση αποστολέα την διεύθυνση του εξυπηρετητή-θύμα αντί γι' αυτήν του επιτιθέμενου. Αυτό έχει ως αποτέλεσμα να απαντήσουν όλοι οι υπολογιστές των εκάστοτε τοπικών δικτύων στον εξυπηρετητή-θύμα με μηνύματα «ECHO_REPLY».

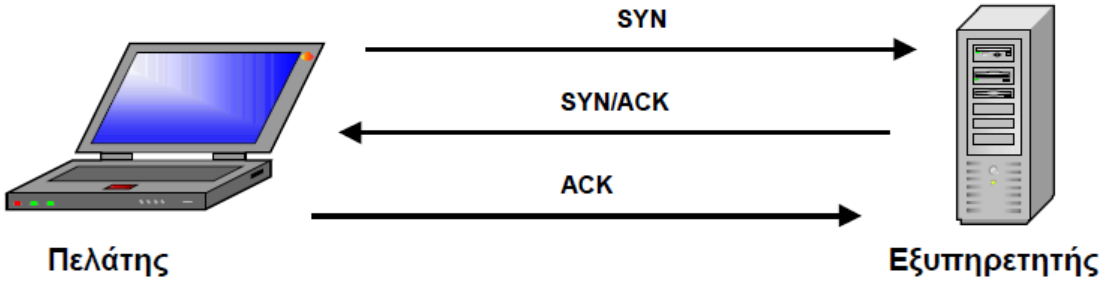


Παραδείγματα Smurf attack

Οι επιθέσεις αυτές είναι πιο αποτελεσματικές από τις «ICMP flood», όμως μπορούν εύκολα να αποτραπούν με έναν απλό firewall που θα απορρίπτει πακέτα που αποστέλλονται σε broadcast διευθύνσεις. Βέβαια, αν χρησιμοποιηθούν για την επίθεση πολλές διαφορετικές broadcast διευθύνσεις, τότε θα πρέπει σε κάθε μία από αυτές να υπάρχει εγκατεστημένος κάποιος firewall που να απορρίπτει τα πακέτα της επίθεσης

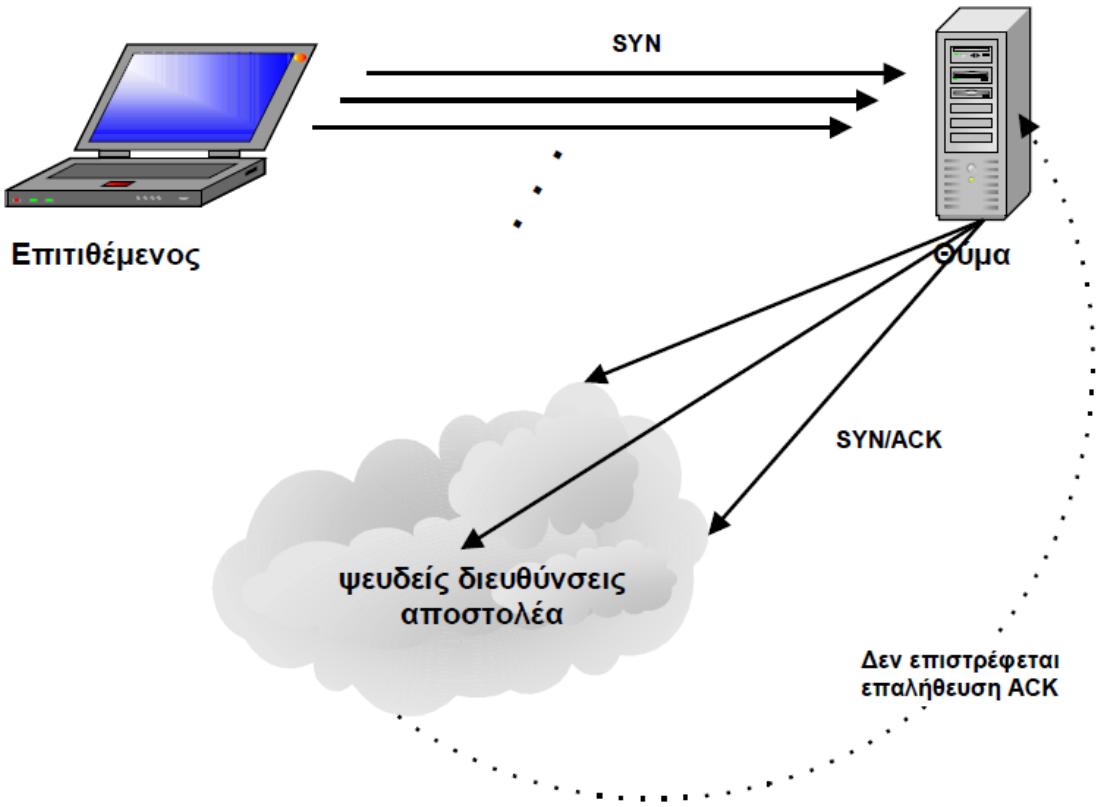
6.4.4 TCP SYN flood.

Οι «TCP SYN flood» επιθέσεις δεν στοχεύουν στην κατανάλωση του εύρους ζώνης, αλλά στην κατανάλωση πόρων του συστήματος. Για να γίνει κατανοητό το πώς ακριβώς γίνονται οι «TCP SYN flood» επιθέσεις πρέπει πρώτα να αναλύσουμε την διαδικασία σύνδεσης δύο υπολογιστών με το πρωτόκολλο TCP. Στα πλαίσια αυτής της επίθεσης, αποστέλλεται μεγάλο πλήθος καθ' όλα νόμιμων αιτήσεων σύνδεσης SYN της υπηρεσίας TCP. Για κάθε μια από αυτές το σύστημα δεσμεύει πόρους και δηλώνει τη διαθεσιμότητα του αποστέλλοντας πακέτο SYN/ACK στα πλαίσια των προδιαγραφών του TCP (three way handshake) όπως φαίνεται στο παρακάτω σχεδιάγραμμα:



Three way handshake

Φυσικά, ο επιτιθέμενος δεν έχει καμία πρόθεση να ολοκληρώσει τη διαδικασία σύνδεσης και η διεύθυνση επιστροφής που περιέχει το SYN πακέτο είναι επίτηδες παραποιημένη-spoofed (με raw socket εφαρμογή) και τυχαία παραγόμενη , ώστε συνήθως να μην αντιστοιχεί σε έγκυρη IP διεύθυνση. Αποτέλεσμα είναι να δεσμευτούν πόροι του TCP server και να αποσταλούν μια σειρά από SYN/ACK πακέτα στην προσπάθεια να γίνει η σύνδεση , όπως φαίνεται στο παρακάτω σχεδιάγραμμα:



Παράδειγμα TCP SYN flood attack

Οι διαθέσιμοι πόροι κατά τον τρόπο αυτό περιορίζονται και το σύστημα από κάποιο σημείο θα αρχίσει να αποκρίνεται με μειωμένη ταχύτητα στις πραγματικές νέες κλήσεις ή στη χειρότερη περίπτωση θα πάψει να ανταποκρίνεται τελείως.

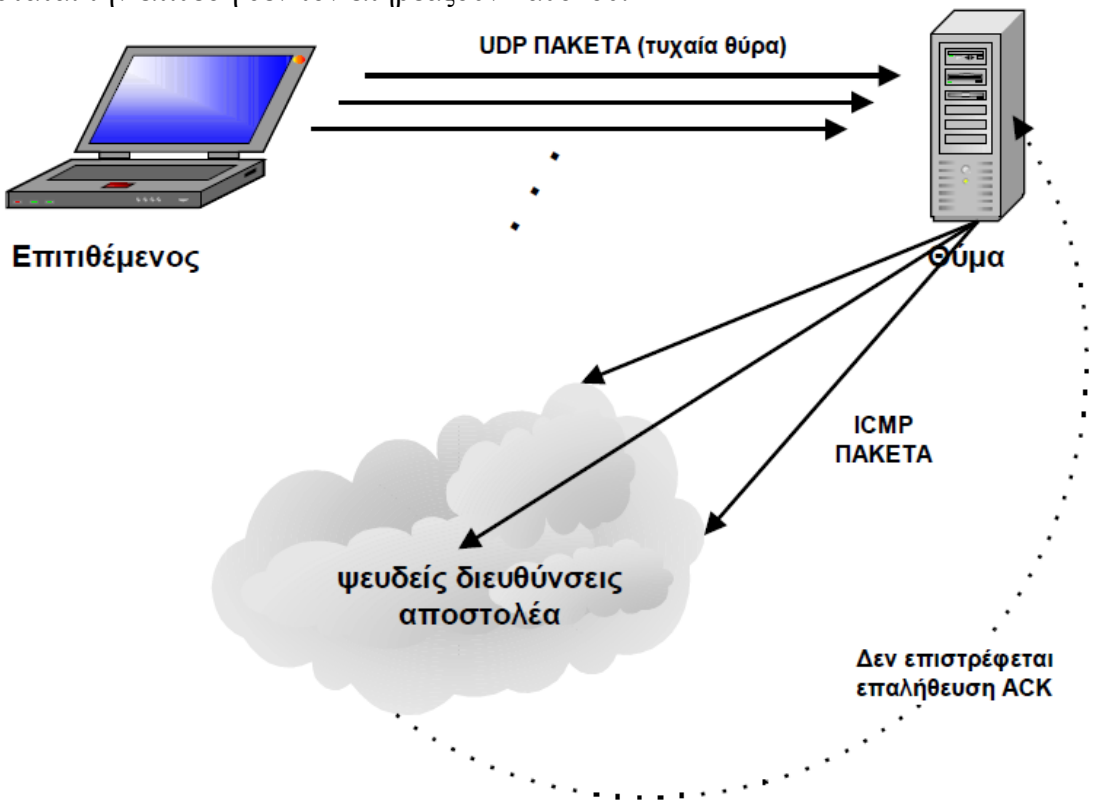
6.4.5 UDP flood.

Η επίθεση UDP flood (UDP flood attack) είναι μία υποπερίπτωση των επιθέσεων άρνησης υπηρεσιών (Denial of Service - DOS) στην οποία χρησιμοποιούνται πακέτα UDP. Η αντίστοιχη μορφή επίθεσης υπάρχει και για πακέτα TCP και μάλιστα είναι πολύ πιο συνηθισμένη.

Μία επίθεση UDP flood περιλαμβάνει την αποστολή ενός πολύ μεγάλου αριθμού UDP πακέτων σε τυχαίες πόρτες ενός υπολογιστή. Ο υπολογιστής που δέχεται την επίθεση θα πρέπει αρχικά να διαπιστώσει εάν κάποια από τις υπηρεσίες του ακούει στην συγκεκριμένη πόρτα και εάν δεν ακούει να απαντήσει με ένα πακέτο ICMP Destination Unreachable. Άρα λοιπόν, η εισροή μεγάλου αριθμού UDP πακέτων στον υπολογιστή που υφίσταται την επίθεση τον αναγκάζει να απαντήσει με εξίσου μεγάλο αριθμό πακέτων ICMP, γεγονός που τελικά εμποδίζει άλλους απλούς χρήστες από το να χρησιμοποιήσουν τις υπηρεσίες του υπό επίθεση υπολογιστή.

Ο επιτιθέμενος μπορεί στο πεδίο Source Address των πακέτων UDP να μην χρησιμοποιήσει την δικιά του διεύθυνση IP, αλλά κάποια άλλη τυχαία διεύθυνση. Με τον τρόπο αυτό παραμένει ανώνυμος και ο υπολογιστής που δέχεται την επίθεση δεν

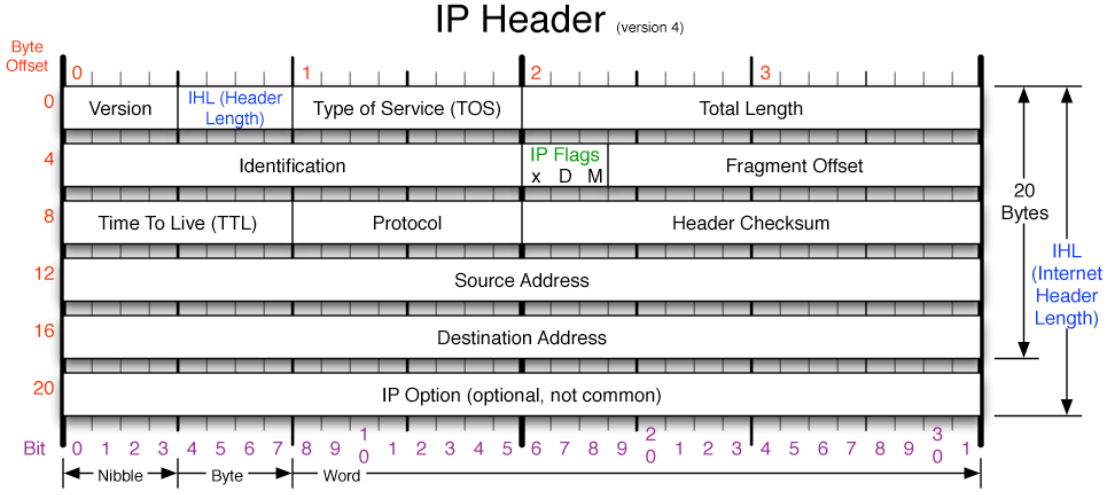
μπορεί να τον εντοπίσει. Επιπροσθέτως τα πακέτα ICMP που στέλνει ο υπολογιστής που υφίσταται την επίθεση δεν τον επηρεάζουν καθόλου.



Παράδειγμα UDP flood attack

6.4.6 Teardrop attack.

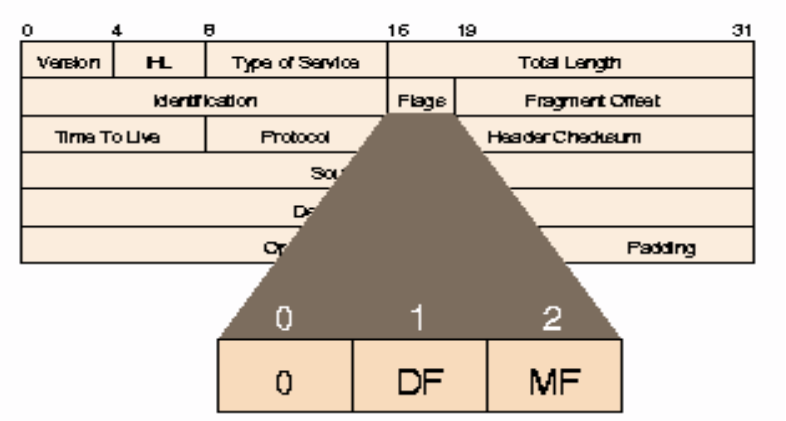
Καταρχήν για να καταλάβουμε αυτού του είδους την επίθεση ας πούμε μερικά πράγματα για το πρωτόκολλο TCP/IP. Κάθε πακέτο σε κάποιο δίκτυο έχει ένα καθορισμένο μέγεθος. Αυτό ονομάζεται MTU (Maximum Transmission Unit). Αυτό το μέγεθος είναι το μεγαλύτερο που μπορεί το δίκτυο να στείλει. Έτσι εάν θέλουμε να στείλουμε ένα πακέτο που έχει μέγεθος μεγαλύτερο από το επιτρεπτό MTU θα πρέπει να το χωρίσουμε σε μικρότερα μέρη.



Τα στοιχεία που χρειάζονται για τον χωρισμό των πακέτων είναι τα:

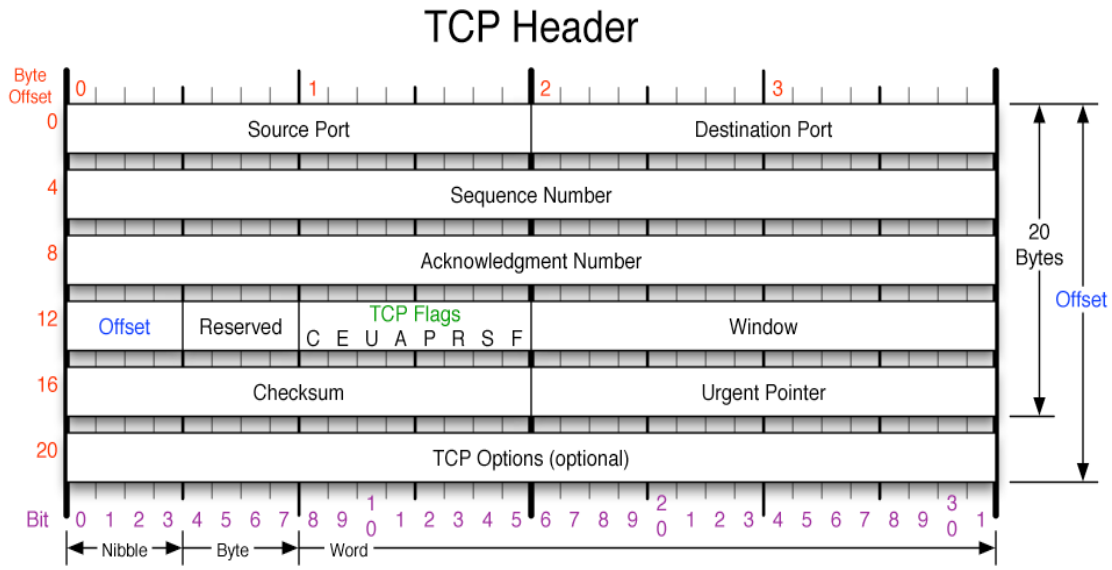
Identification = ταυτότητα αναγνώρισης : 16bits
Flags = σημαια : 3 bits, από αυτά τα 3 bits έχουμε:

1^{ο} bit	δεν το χρησιμοποιούμε
2^{ο} bit (DF)	αν έχει την τιμή 0 πρέπει να χωρίσουμε το πακέτο αν έχει την τιμή 1 δεν πρέπει
3^{ο} bit (MF)	αν έχει την τιμή 0 είναι η τελευταία τμηματοποίηση αν έχει την τιμή 1 υπάρχουν και άλλα πακέτα



Fragment Offset = μετατόπιση τμήματος : 13bits

Το στοιχείο αυτό δείχνει που ανήκει το κάθε πακέτο χωριστά. Δηλαδή με ποια σειρά πάνε τα πακέτα. Το fragment offset μετράται με βάση το μέγεθος των δεδομένων δια 8 (64 bits). Αυτά τα 3 στοιχεία χρησιμοποιούνται για την καθορισμό των επόμενων πακέτων μετά τον χωρισμό.

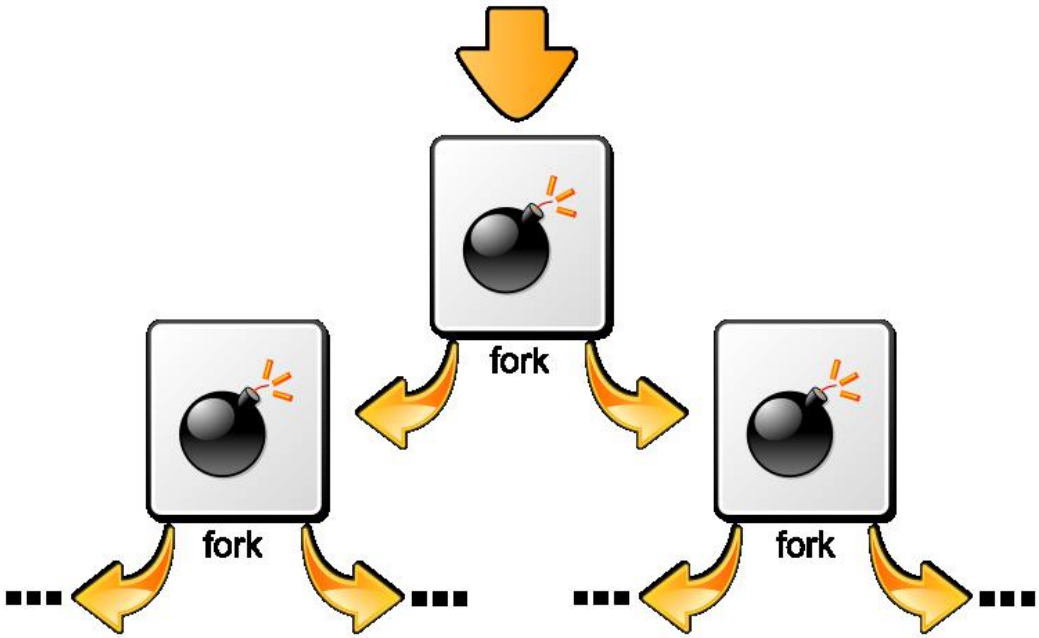


Έστω ότι είμαστε σε ένα Ethernet δίκτυο όπου το MTU είναι ίσο με 1500 και θέλουμε να στείλουμε ένα πακέτο που έχει μέγεθος 2.500 bytes.

Έτσι το πρώτο πακέτο θα έχει μέγεθος 1.500 bytes. 20 bytes για το IP header ,24 bytes για το TCP header και 1.456 bytes για τα δεδομένα. Η flag θα έχει τιμή DF=0 κ MF=1 ώστε ο υπολογιστής περιμένει κ άλλο πακέτο ενώ το Fragmentation Offset θα έχει την τιμή 0. Το δεύτερο πακέτο θα έχει μέγεθος 1.088 bytes. 20 bytes για το IP header, 24 bytes για το TCP header και 1.044 bytes για τα δεδομένα. Η flag θα έχει τιμή DF=0 κ MF=1 ώστε ο υπολογιστής να μην περιμένει κ άλλο πακέτο ενώ το Fragmentation Offset θα έχει την τιμή 182(1456/8=132).
Σκεφτείτε λοιπόν να στέλνεις συνεχεία πακέτα με αλλαγμένα αυτά τα 3 στοιχεία σε έναν υπολογιστή. Σε κάθε πακέτο το MF να είναι ίσο με ένα δηλαδή να περιμένει πάντα πακέτα και Fragmentation Offset να έχει άκυρες τιμές έτσι ώστε να μην μπορεί να δημιουργήσει το αρχικό πακέτο. Έτσι θα γεμίσει όλη η buffer μνήμη του server και δεν θα μπορεί να δέχεται άλλα πακέτα από κανέναν δηλαδή μια Dos attack. Και μετά από σύντομο διάστημα θα έχει ως αποτέλεσμα να κάνει reboot και να χάσει δεδομένα.

6.4.7 Fork Bombs.

Fork bombs είναι ένα είδος επίθεσης DoS η οποία έχει ως σκοπό την σπατάλη όλη της μνήμης RAM σε ένα σύστημα. Αυτό το πετυχαίνει τρέχοντας πολλές διεργασίες η οποίες με την σειρά τους τρέχουν άλλες διεργασίες κτλ. Συνήθως αυτό γίνεται με ένα ατέρμονο βρόγχο (infinite loop) ο οποίος δεν σταματά ποτέ. Το αποτέλεσμα, πολύ απλά δεν θα μπορέσουμε να τρέξουμε οποιοδήποτε process στο σύστημα όσο το fork bomb και όλα τα sub-process παραμένουν ενεργά στο σύστημα. Fork bombs δεν είναι απαραίτητο να έχουν δημιουργηθεί για να προκαλέσουν DoS. Συνήθως απροσεξίες των προγραμματιστών μπορούν να δημιουργήσουν ένα fork bomb.



Fork bombs

Fork bombs μπορούν να δημιουργηθούν σε όλες τις γλώσσες εμείς θα δούμε σε μερικές έτσι ώστε να κατανοήσουμε πως λειτουργούν τα fork bombs αλλά και να δούμε πόσο εύκολο είναι να δημιουργηθεί ένα.

Fork bombs σε ms-dos batch αρχεία:
Ανοίγουμε το notepad και γράφουμε αυτό:

```
%0 | %0
```

Τι κάνει; Απλώς κάνει συνεχώς Pipelining με τον εαυτό του δημιουργώντας fork bomb. (Το %0 δηλώνει το τρέχων όνομα του αρχείου, %1 είναι πρώτο command-line argument κτλ.)
Ας δούμε και κάτι λίγο πιο σύνθετο:

```
:start
start %0
goto start
```

Αυτό απλώς κάνει ένα είδος ατέρμονου goto loop αφού θα εκτελεί τον εαυτό του κάνοντας και άλλα sub-processes του εαυτού του.

Φυσικά αυτό μπορεί να παραμετροποιηθεί και αντί για 'start %0' να βάζαμε όποιο αρχείο-εντολή θέλαμε.

Ας δούμε τώρα τι γίνεται σε Linux systems. Εδώ τα πράγματα είναι εξίσου εύκολα. Για να γίνει πιο κατανοητό εκτελούμε το κάτω script στο terminal:

```
: () { :|:& } ; :
```

Επεξήγηση του script

:()	define ':' -- whenever we say ':', do this:
{	beginning of what to do when we say ':'
:	load another copy of the ':' function into memory...
|	...and pipe its output to...
:	...another copy of ':' function, which has to be loaded into memory (therefore, ':|:' simply gets two copies of ':' loaded whenever ':' is called)
&	disown the functions -- if the first ':' is killed, all of the functions that it has started should NOT be auto-killed
}	end of what to do when we say ':'
;	Having defined ':', we should now...
:	...call ':', initiating a chain-reaction: each ':' will start two more.

Φυσικά μπορούμε να κάνουμε κάτι αντίστοιχο με το PipeLining στο Ms-dos και εδώ:

```
$0 | $0
```

Ένα πολύ ωραίο παράδειγμα fork bomb είναι στην C/C++:

```
#include <unistd.h>
int main(void)
{
  for(;;)
    fork();
  return 0;
}
```

Όπου συνεχώς με ένα for loop ανοίγει ένα νέο fork. Βέβαια μπορούμε να το παραμετροποιήσουμε με μια while loop κάπως έτσι:

```
#include <unistd.h>
int main(void)
{
  while(1)
    fork();
  return 0;
}
```

Πάντως σε όποια γλώσσα και να υλοποιηθεί θα υπάρχουν τα ίδια αποτελέσματα. Οι πιο πρόσφατοι Kernel είναι προστατευμένοι από τέτοιου είδους επιθέσεις, περιορίζοντας τον αριθμό των εφαρμογών που κάθε χρήστης μπορεί να

"παράγει" ταυτόχρονα. Σε Linux συστήματα μπορούμε να βάλουμε κάποια limitations έτσι ώστε π.χ. να ελέγχουμε πόσα processess μπορεί ένας χρήστης να "τρέξει" χρησιμοποιώντας την εντολή *ulimit*. Η σύνταξη είναι η εξής: *ulimit -u <max of processes>*
Άλλη μέθοδος που μπορεί να χρησιμοποιηθεί σε Linux & BSD λειτουργικά συστήματα είναι μέσω editing του */etc/security/limits.conf*

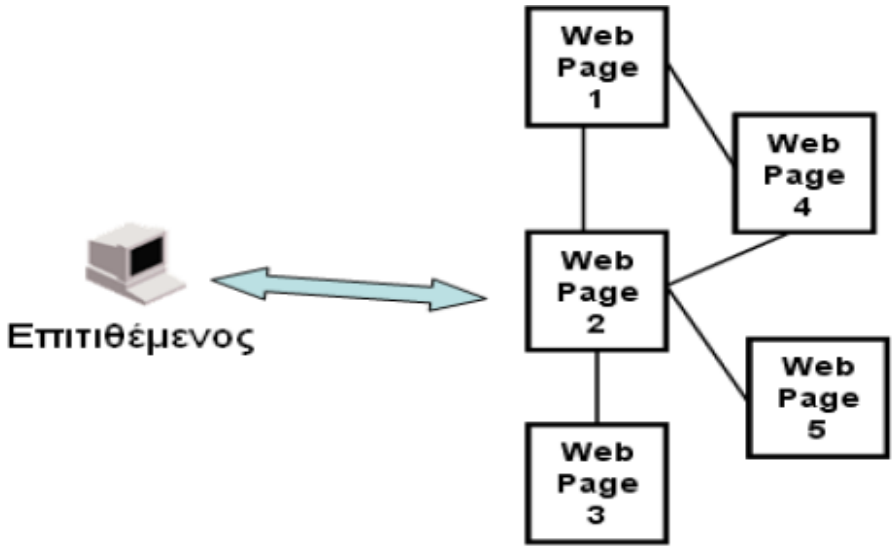
6.4.8 Επιθέσεις τύπου Web DoS ή HTTP Flood.

Οι επιθέσεις τύπου Web DoS αποτελούν ένα νέο είδος επιθέσεων άρνησης εξυπηρέτησης που παρουσιάζουν σημαντικές ποιοτικές διαφορές από τις SYN Flood επιθέσεις που προαναφέρθηκαν. Οι Web DoS επιθέσεις αφορούν την υπηρεσία του παγκόσμιου ιστού (World Wide Web) και έχουν δύο στόχους:

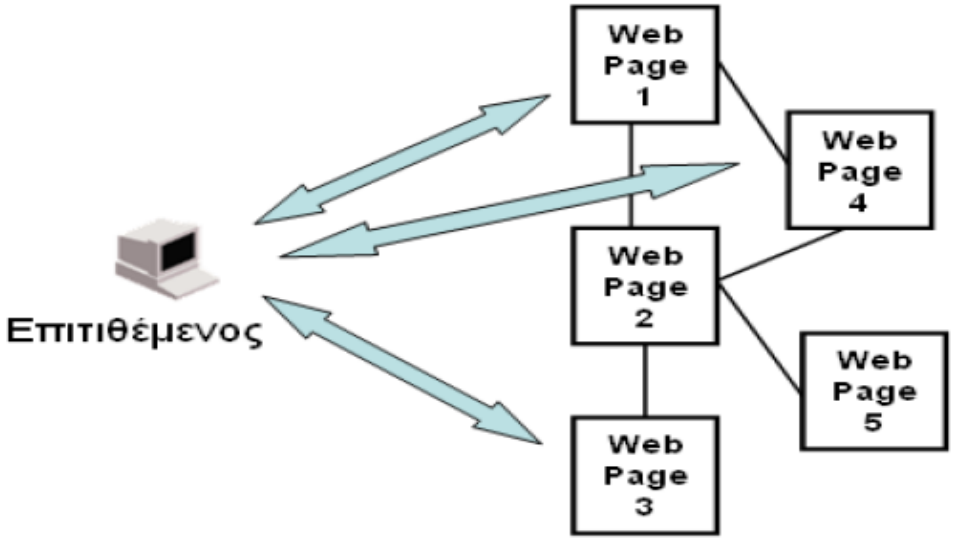
- Να δεσμεύσουν τους διαθέσιμους χρήστες που μπορεί να εξυπηρετήσει ο διακομιστής χωρίς να μπορεί έτσι ένας κανονικός χρήστης να χρησιμοποιήσει την υπηρεσία του παγκόσμιου ιστού.
- Να μειώσουν συστηματικά τον πραγματικό αριθμό των χρηστών που μπορούν να εξυπηρετηθούν καθώς και την ταχύτητα πρόσβασης στις ιστοσελίδες μειώνοντας έτσι την ποιότητα των υπηρεσιών που παρέχονται.

Ο πρώτος στόχος γίνεται εύκολα αντιληπτός αφού ο διακομιστής θα παράγει κάποια προειδοποίηση ή κάποιο σφάλμα. Ο δεύτερος στόχος όμως δεν είναι απαραίτητο πως θα παράγει κάποια προειδοποίηση και έτσι η επίθεση μπορεί να πραγματοποιείται για μεγάλα χρονικά διαστήματα χωρίς να γίνεται αντιληπτή. Στην απλή μορφή τους και σε χαμηλό επίπεδο οι Web DoS επιθέσεις δεν έχουν καμία διαφορά από τις SYN Flood επιθέσεις. Οι Web DoS επιθέσεις που θα εξεταστούν, αποτελούν μια αρκετά σύνθετη μορφή επιθέσεων στην οποία το πρόγραμμα το οποίο εξαπολύει την επίθεση προσπαθεί να μιμηθεί με τον καλύτερο δυνατό τρόπο έναν πραγματικό χρήστη εκτελώντας μια πλοήγηση στον εκάστοτε δικτυακό τόπο. Οι Web DoS επιθέσεις μπορούν να χωριστούν σε τρεις κατηγορίες:

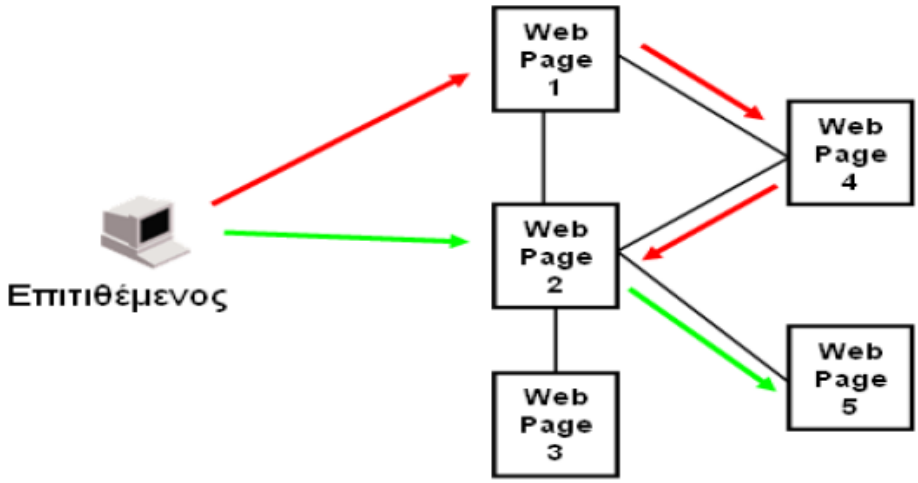
• Επιθέσεις που ζητούν συνέχεια την ίδια σελίδα (Τύπου-1).



• Επιθέσεις που ζητούν συνέχεια τυχαίες σελίδες (Τύπου-2).



• Επιθέσεις που μιμούνται την πλοήγηση κανονικών χρηστών (Τύπου-3).



6.4.9 Email bomb.

Ο όρος email bomb (βόμβα email) στην επιστήμη υπολογιστών αναφέρεται σε ένα είδος επίθεσης κατά την οποία ο επιτιθέμενος στέλνει μία τεράστια ποσότητα ηλεκτρονικών μηνυμάτων σε μία διεύθυνση ηλεκτρονικού ταχυδρομείου με σκοπό να γεμίσει τον διαθέσιμο χώρο στον δίσκο και να προκαλέσει δυσλειτουργία στον mail server.

Μία μορφή email bomb που είναι αρκετά συνηθισμένη ονομάζεται ZIP bomb και βασίζεται στο γεγονός ότι πολλοί από τους σύγχρονους mail servers διαθέτουν προγράμματα ελέγχου των email για τον εντοπισμό ιών. Εάν για παράδειγμα κάποιο email περιλαμβάνει ως επισύναψη ένα συμπιεσμένο αρχείο (.zip, .rar κοκ), τότε πολλοί από τους σύγχρονους mail servers θα αποσυμπιέσουν το αρχείο και θα ελέγξουν το περιεχόμενό του για ιούς ή δούρειους ίππους.

Μία ZIP bomb είναι ένα email που περιέχει ένα συμπιεσμένο αρχείο ως επισυναπτόμενο. Αυτό το συμπιεσμένο αρχείο περιλαμβάνει ένα τεράστιο αρχείο κειμένου αρκετών GB, το οποίο αποτελεί ουσιαστικά συνεχή επανάληψη ενός γράμματος (πχ α). Ένα τέτοιο αρχείο έχει το εξής χαρακτηριστικό: Όταν είναι συμπιεσμένο καταλαμβάνει ελάχιστο χώρο, αλλά όταν αποσυμπιεστεί ο χώρος που δεσμεύει είναι τεράστιος. Άρα λοιπόν, όταν ο mail server προσπαθήσει να αποσυμπιέσει το αρχείο για να ελέγξει το περιεχόμενό του, τότε το αποσυμπιεσμένο αρχείο θα δεσμεύσει μία τεράστια ποσότητα υπολογιστικής ισχύος, μνήμης RAM, και σκληρού δίσκου. Αυτό έχει πολλές φορές ως συνέπεια το πάγωμα του υπολογιστή. Παρόλα αυτά οι σύγχρονοι mail servers είναι στην πλειοψηφία τους άτρωτοι σε ZIP bombs διότι αφενός είναι σε θέση να τις αναγνωρίζουν και αφετέρου διαθέτουν αρκετά υψηλές δυνατότητες (μεγάλη ταχύτητα επεξεργαστή, αρκετή μνήμη κοκ) για να μπορέσουν να συνεχίσουν ομαλά την λειτουργία τους ακόμη και όταν λάβουν μία τέτοια βόμβα.

Υπάρχουν δύο τρόποι διακίνησης email bombs.

- ✓ Ο πρώτος τρόπος συνίσταται στην μαζική αποστολή ηλεκτρονικών μηνυμάτων στον ίδιο παραλήπτη. Ο σχεδιασμός προγραμμάτων που θα επιτελούν αυτήν την λειτουργία είναι αρκετά απλός, αλλά τέτοιου είδους

βόμβες εντοπίζονται εύκολα από φίλτρα spam και τελικά δεν πετυχαίνουν τον στόχο τους. Πολλές φορές οι χάκερ χρησιμοποιούν υπολογιστές zombie για να ξεκινήσουν μία επίθεση DDoS - Distributed Denial of Service. Κατά την επίθεση αυτή, ο χάκερ δίνει εντολή στους υπολογιστές zombie να στείλουν δισεκατομμύρια emails προς έναν συγκεκριμένο στόχο με σκοπό να τον γεμίσουν με emails και έτσι να παρεμποδίσουν την σωστή λειτουργία του. Η επίθεση αυτή είναι πιο δύσκολο να αντιμετωπιστεί σε σχέση με το απλό email bombing διότι αυτήν την φορά τα emails προέρχονται από δεκάδες διαφορετικούς υπολογιστές zombie.

- ✓ Ο δεύτερος τρόπος διακίνησης email bombs περιλαμβάνει την εγγραφή της ηλεκτρονικής διεύθυνσης του θύματος σε διάφορες διαδικτυακές υπηρεσίες (mailing lists, newsletters κοκ). Εάν ο επιτιθέμενος καταφέρει να εγγράψει το θύμα σε πολλές τέτοιες υπηρεσίες, τότε το θύμα θα παραλαμβάνει δεκάδες email από κάθε υπηρεσία, γεμίζοντας με τον τρόπο αυτό τον σκληρό δίσκο του mail server. Για την αποφυγή τέτοιων επιθέσεων έχει καθιερωθεί πλέον η τακτική της αποστολής ενός email επιβεβαίωσης πριν οριστικοποιηθεί η εγγραφή του χρήστη σε μία διαδικτυακή υπηρεσία.

6.4.10 DNS Amplification attack.

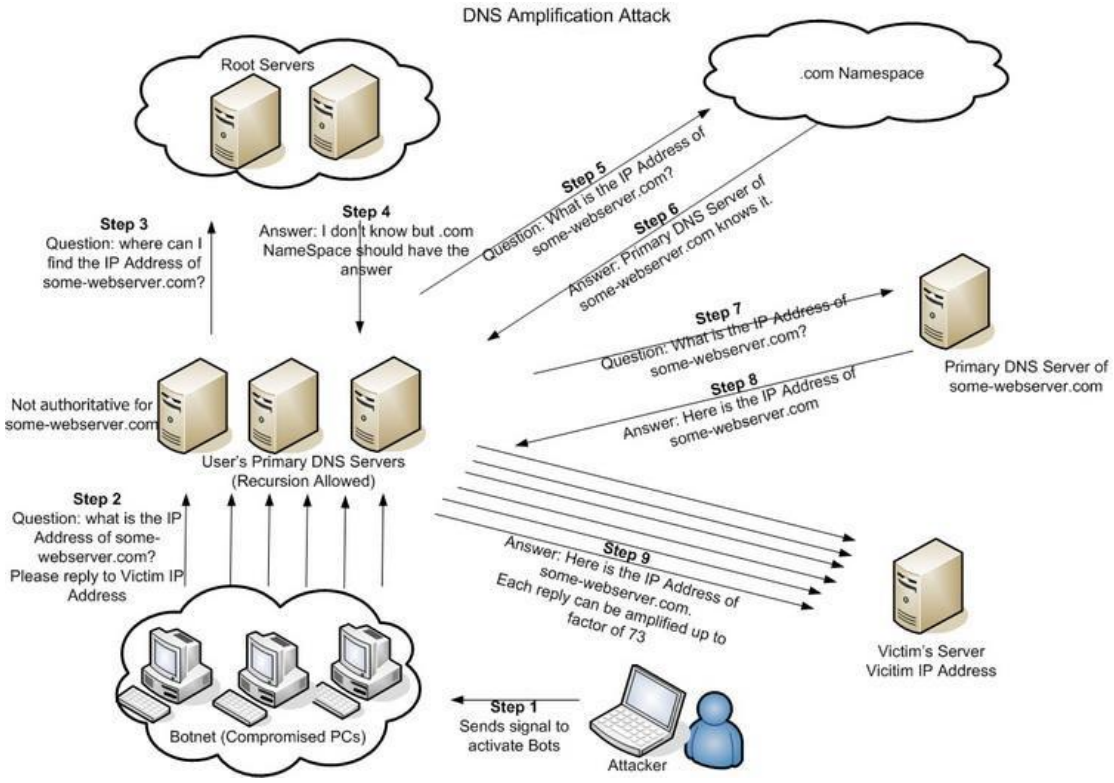
Το Domain Name System (DNS) είναι υπεύθυνο για την μετάφραση των ονομάτων των εξυπηρετητών στις IP διευθύνσεις (και αντίστροφα) και είναι κρίσιμη υπηρεσία για την ομαλή λειτουργία των διασυνδεδεμένων σε δίκτυο συσκευών.

Ένας από τους μεγαλύτερους φόβους των διεθνών αρχών είναι ότι στόχος θα είναι η λειτουργία του internet παγκοσμίως. Ο φόβος είναι δικαιολογημένος, γιατί αυτό έχει ήδη συμβεί τουλάχιστον δύο φορές, το 2002 και το 2007. Και οι δύο επιθέσεις στόχευαν στην καρδιά του internet, στο Σύστημα Ονομάτων Τομέα (DNS, Domain Name System). Οι διευθύνσεις που πληκτρολογούμε (όπως www.unipi.gr) αντιστοιχούν σε πολύπλοκες ομάδες με 10 ψηφία, που είναι ο κώδικας κάθε ονόματος τομέα. Όταν συνδεόμαστε, τα γράμματα μετατρέπονται σε ψηφία, χωρίς όμως εμείς να βλέπουμε κάτι τέτοιο. Μόλις 13 server σε όλο τον κόσμο διατηρούν τον επίσημο κατάλογο των ενεργών ονομάτων τομέα. Αποτελούν το κλειδί για την παγκόσμια διασύνδεση και, αν έπεφταν, το internet θα κατέρρεε αμέσως.

Στις 6 Φεβρουαρίου του 2007 κάποιος προσπάθησε να προκαλέσει αυτό το τρομερό παγκόσμιο μπλακάουτ του ψηφιακού κόσμου. Η επίθεση προήλθε από την περιοχή της Ασίας-Ειρηνικού Ωκεανού και είχε δύο στάδια: το πρώτο διήρκησε δυόμισι ώρες, στη συνέχεια υπήρξε μια διακοπή τρεισήμισι ωρών και έπειτα η επίθεση ξανάρχισε για πέντε συνεχόμενες ώρες. Η τυπολογία ήταν κατανεμημένη επίθεση άρνησης υπηρεσίας μέσω υπολογιστών ζόμπι. Η επίθεση εξαπολύθηκε στους έξι από τους 13 server των ονομάτων τομέα και δύο από αυτούς υπέστησαν σοβαρές ζημιές. Οι επιτιθέμενοι γνώριζαν τι έκαναν, παρ' όλο που δεν πέτυχαν το στόχο τους. Πιο σημαντικό ήταν όμως το επεισόδιο της 21 Οκτωβρίου του 2002, την ημέρα που το internet έφτασε στο χείλος της κατάρρευσης, με τους επιτιθέμενους να αφήνουν νοκ άουτ εννέα από τους 13 server.

Οι επιθέσεις στους DNS servers αποσκοπούν στην διακοπή των υπηρεσιών αντιστοίχησης των ονομάτων του διαδικτύου σε IP διευθύνσεις προκαλώντας το ίδιο αποτέλεσμα αφού ο χρήστης δεν θα μπορεί να επικοινωνήσει με την υπηρεσία μιας εταιρίας αν ο υπολογιστής του δεν μπορεί να βρει το συγκεκριμένο IP. Ο server βομβαρδίζεται με πολλά αιτήματα πρόσβασης και τον οδηγούν έτσι σε υπερφόρτωση

(flooding). Τα αιτήματα προέρχονται συχνά από τους υπολογιστές ανυποψίαστων χρηστών που έχουν μολυνθεί με ιούς.



DNS Amplification attack

7 Προληπτικοί μηχανισμοί και μέτρα προστασίας.

Για να θεωρείται ασφαλές ένα σύστημα υπολογιστών ή και δικτύων αυτών οφείλει να παρέχει τις ακόλουθες υπηρεσίες :

- Δυνατότητα πρόσβασης σε δεδομένα μόνο από εξουσιοδοτημένους χρήστες (data confidentiality).
- Εξασφάλιση της ακεραιότητας των δεδομένων και των μεθόδων επικοινωνίας αυτών (data and communication integrity).
- Διασφάλιση από περιστατικά άρνησης υπηρεσιών (denial of service).

Ειδικά όσον αφορά στην περίπτωση των περιστατικών άρνησης υπηρεσίας , θεωρείται ότι ένα τέτοιο περιστατικό λαμβάνει χώρα εφόσον το εύρος επικοινωνίας (throughput) μειωθεί κάτω από μια προκαθορισμένη τιμή κατωφλίου ή εφόσον ή επικοινωνία με μια (απομακρυσμένη) οντότητα καταστεί αδύνατη. Ενώ τέτοια περιστατικά δεν μπορεί ποτέ να αποκλειστούν τελείως, είναι επιθυμητό να μειωθεί η πιθανότητα εμφάνισης τους κάτω από ένα συγκεκριμένο όριο.

7.1 Αντιδραστικοί μηχανισμοί.

Οι Αντιδραστικοί μηχανισμοί (γνωστοί και ως early warning systems-συστήματα έγκαιρης προειδοποίησης) προσπαθούν να ανιχνεύσουν την επίθεση και να απαντήσουν σε αυτήν άμεσα. Ως εκ τούτου, περιορίζουν τον αντίκτυπο της επίθεσης πάνω στο θύμα. Και πάλι όμως, υπάρχει ο κίνδυνος του χαρακτηρισμού μιας νόμιμης σύνδεσης ως επίθεση. Για αυτόν τον λόγο είναι απαραίτητο για τους ερευνητές να είναι πολύ προσεκτικοί.

Οι κύριες στρατηγικές ανίχνευσης είναι ανίχνευση-υπογραφής, ανίχνευση-ανωμαλίας και υβριδικά συστήματα. Οι μέθοδοι που είναι βασισμένες στην ανίχνευση-υπογραφής αναζητούν πρότυπα (υπογραφές) πάνω στην παρατηρηθείσα κίνηση του δικτύου που ταιριάζουν με γνωστές υπογραφές επίθεσης μιας βάσης δεδομένων. Το πλεονέκτημα αυτών των μεθόδων είναι ότι μπορούν εύκολα και αξιόπιστα να ανιχνεύσουν γνωστές επιθέσεις, αλλά δεν μπορούν να αναγνωρίσουν νέες επιθέσεις. Επιπλέον, η βάση υπογραφών πρέπει να ενημερώνεται τακτικά προκειμένου να διατηρηθεί η αξιοπιστία του συστήματος.

Οι μέθοδοι που είναι βασισμένες στην ανίχνευση-ανωμαλίας συγκρίνουν τις παραμέτρους της παρατηρηθείσας κίνησης του δικτύου με την κανονική κίνηση. Ως εκ τούτου, είναι δυνατό και νέες επιθέσεις να ανιχνευθούν. Εντούτοις, προκειμένου να αποτραπεί ένας ψεύτικος συναγερμός από το σύστημα, το πρότυπο της "κανονικής κίνησης" πρέπει να διατηρείται πάντα ενημερωμένο και τα όρια ταξινόμησης μιας ανωμαλίας πρέπει πάντα να ρυθμίζονται κατάλληλα.

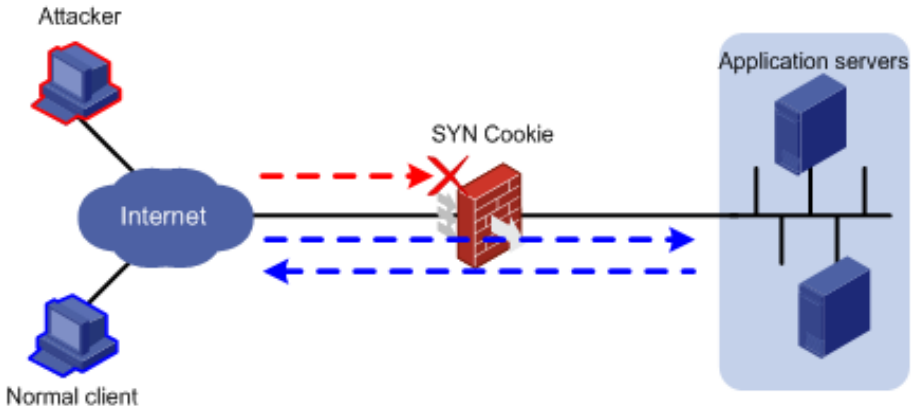
Τέλος, τα υβριδικά συστήματα συνδυάζουν και τις δύο ανωτέρω μεθόδους. Αυτά τα συστήματα ενημερώνουν τη βάση υπογραφών τους με επιθέσεις που ανιχνεύονται με βάση την ανίχνευση ανωμαλίας. Και πάλι ο κίνδυνος είναι μεγάλος καθώς ένας επιτιθέμενος μπορεί να κοροϊδέψει το σύστημα οδηγώντας το στο

χαρακτηρισμό μιας κανονικής κίνησης ως επίθεση. Σε αυτήν την περίπτωση το IDS (σύστημα ανίχνευσης εισβολής) σύστημα γίνεται ένα εργαλείο επίθεσης. Κατά συνέπεια οι σχεδιαστές IDS συστημάτων πρέπει να είναι πολύ προσεκτικοί επειδή η έρευνά τους μπορεί να γυρίσει μπούμερανγκ.

Μετά την ανίχνευση της επίθεσης, οι αντιδραστικοί μηχανισμοί απαντούν σε αυτή. Η ανακούφιση από τον αντίκτυπο της επίθεσης είναι ο πρωταρχικός στόχος. Μερικοί μηχανισμοί αντιδρούν περιορίζοντας το ποσοστό της αποδεχόμενης κίνησης. Αυτό σημαίνει ότι η νόμιμη κίνηση εμποδίζεται επίσης. Σε αυτήν την περίπτωση η λύση έρχεται με τις traceback τεχνικές που προσπαθούν να προσδιορίσουν τον επιτιθέμενο. Εάν ο επιτιθέμενος προσδιοριστεί, παρά τις προσπάθειές του να αλλοιώσει τη διεύθυνσή του, τότε είναι εύκολο να φιλτραριστεί η κίνησή του. Το φιλτράρισμα είναι αποδοτικό μόνο εάν η ανίχνευση του επιτιθεμένου δεν είναι λανθασμένη. Σε οποιαδήποτε άλλη περίπτωση το φιλτράρισμα μπορεί να μετατραπεί σε εργαλείο επίθεσης.

7.2 SYN cookies.

Τα SYN cookies είναι το βασικό στοιχείο της τεχνικής που χρησιμοποιείται για την προστασία έναντι των SYN flood επιθέσεων. Ειδικότερα, η χρήση των SYN Cookies επιτρέπει σε ένα διακομιστή να αποφευχθούν διακοπές των συνδέσεων, όταν η ουρά SYN γεμίζει. Αντί αυτού, ο server συμπεριφέρεται ως εάν η ουρά SYN να είχε διευρυνθεί. Ο server στέλνει πίσω την κατάλληλη SYN + ACK απάντηση προς τον πελάτη, αλλά απορρίπτει την SYN είσοδο στην ουρά. Εάν ο διακομιστής λαμβάνει έπειτα μια μεταγενέστερη απάντηση ACK από τον πελάτη, ο διακομιστής είναι σε θέση να ανακατασκευάσει την SYN είσοδο στην ουρά, χρησιμοποιώντας τις πληροφορίες που κωδικοποιούνται με τον αριθμό ακολουθίας TCP.



Προκειμένου να ξεκινήσει μια σύνδεση TCP, ο πελάτης στέλνει ένα πακέτο TCP SYN στον διακομιστή. Σε απάντηση, ο server στέλνει ένα TCP SYN + ACK πακέτο πίσω στον πελάτη. Μία από τις τιμές σε αυτό το πακέτο είναι ένας αύξων αριθμός, ο οποίος χρησιμοποιείται από το πρωτόκολλο TCP να επανασυναρμολογήσει - reassemble τη ροή των δεδομένων. Σύμφωνα με τις προδιαγραφές του πρωτοκόλλου TCP, ο πρώτος αύξων αριθμός που αποστέλλονται από ένα τελικό σημείο μπορεί να είναι οποιαδήποτε τιμή, όπως αποφασίστηκε από το εν λόγω τελικό σημείο.

Όταν ένας πελάτης στέλνει πίσω ένα TCP ACK πακέτο στον server σε απάντηση του SYN + ACK πακέτου του διακομιστή, ο client πρέπει (σύμφωνα με το TCP specification) να χρησιμοποιήσει n+1 πακέτα στο Acknowledgement number , όπου n είναι ο αρχικός αριθμός ακολουθίας που στάλθηκε από τον server. Ο server στη συνέχεια αφαιρεί 1 από το Acknowledgement number για να αποκαλύψει το SYN Cookie που στάλθηκε στον πελάτη. Οι εφαρμογές αυτού του τύπου βασίζονται κυρίως σε λειτουργικά συστήματα Solaris και Linux.

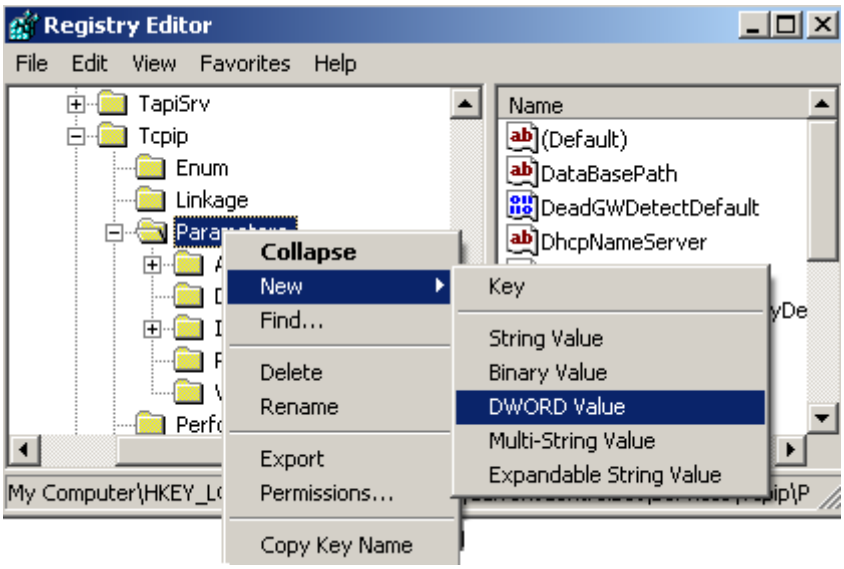
Για να ενεργοποιήσουμε την TCP SYN Cookie προστασία στο Linux πληκτρολογούμε την παρακάτω εντολή :

```
[root@deep] /# echo 1 > /proc/sys/net/ipv4/tcp_syncookies
```

Για να ενεργοποιήσουμε την TCP SYN Cookie προστασία στο Free BSD αφού πάμε στο sysctl.conf file (vi /etc/sysctl.conf) πληκτρολογούμε την τιμή 1 (για disable βάζουμε 0) :

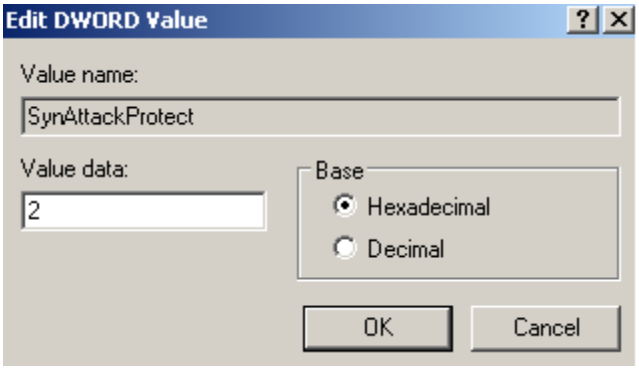
```
# Enable TCP SYN Cookie Protection
net.ipv4.tcp_syncookies = 1
```

Για να ενεργοποιήσουμε την TCP SYN Cookie προστασία στα Windows προσθέτουμε την τιμή DWORD 'SynAttackProtect' στην παρακάτω εγγραφή της registry :



HKLM\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters

Όταν η τιμή 'SynAttackProtect' ρυθμίζεται 1, ο αριθμός των retransmissions μειώνεται και η δημιουργία μιας εγγραφής δρομολόγησης καθυστερείται μέχρι να εγκατασταθεί η σύνδεση.



Η προτεινόμενη τιμή 'SynAttackProtect' είναι 2, που καθυστερεί επιπρόσθετα μέχρι να ολοκληρωθεί το τριπλό handshake.

7.3 Συμπεράσματα

Το διαδίκτυο δεν είναι σταθερό αλλά αλλάζει μορφές πολύ γρήγορα. Αυτό σημαίνει ότι τα DoS αντίμετρα ξεπερνιούνται πολύ γρήγορα. Νέες υπηρεσίες προσφέρονται μέσω του διαδικτύου και νέες επιθέσεις εξαπολύονται προκειμένου να αποτραπούν οι πελάτες από την πρόσβαση σε αυτές τις νέες υπηρεσίες. Παρόλα αυτά, το βασικό θέμα είναι κατά πόσο οι DoS επιθέσεις αντιπροσωπεύουν ένα δικτυακό πρόβλημα ή ένα πρόβλημα μεμονωμένου χρήστη ή και τα δύο.

Αναμφισβήτητα, οι επιθέσεις DoS πρέπει να αντιμετωπιστούν σαν ένα σοβαρό πρόβλημα στο Διαδίκτυο καθώς ο μεγάλος ρυθμός ανάπτυξής τους και η ευρεία αποδοχή τους προκαλεί το γενικό κοινό, τις δύσπιστες κυβερνήσεις και τις επιχειρήσεις. Είναι προφανές ότι το κύμα των επιθέσεων DoS θα συνεχίσει να αποτελεί μία σημαντική απειλή, καθώς όσο ανακαλύπτονται νέα μέτρα αντιμετώπισης οι επιθέσεις DoS εξελίσσονται. Αφού οι επιθέσεις DoS είναι πολύπλοκες και δύσκολο να αντιμετωπιστούν, δεν υπάρχει μοναδική λύση, όλοι είναι αδύναμοι απέναντι σε αυτή την επίθεση και όλων η ασφάλεια είναι αλληλένδετη. Οποιοσδήποτε δηλώνει ότι έχει κατορθώσει μόνος του να αντικρούσει πλήρως τις επιθέσεις DoS λέει ψέματα. Η λύση θα προκύψει από το συνδυασμό και δικτυακών και μεμονωμένων αντιμέτρων.

8 DOS Attack Trace Files

Κατά την διάρκεια των πειραμάτων μας παρατηρήσαμε ότι στο gshare branch predictor και για το σύνολο των τιμών των παραμέτρων των αλγορίθμων, οι αποδόσεις των μετρο-προγραμμάτων CLIENT08 και WS04 , ήταν οι χειρότερες. Έπειτα από πειράματα βρήκαμε κομμάτια κώδικα (ακολουθίες εντολών) από τα δύο traces στα οποία έχουμε πολύ μικρό ποσοστό επιτυχημένων προβλέψεων από τους branch predictors.

Για να βρούμε τα κομμάτια των προγραμμάτων με τη χειρότερη συμπεριφορά εργαστήκαμε με έναν hex editor ως εξής: Αρχικά βρήκαμε το είδος της συμπίεσης των traces με βάση το πρώτο byte των δύο trace file και τα αποσυμπιέσαμε. Έπειτα ψάξαμε για κομμάτια κώδικα τα οποία παρουσίαζαν την χειρότερη απόδοση (με τυχαία δείγματα). Απομονώσαμε αυτά τα κομμάτια και βρήκαμε μέσα σε αυτά μικρές ακολουθίες εντολών και διευθύνσεων οι οποίες "μπερδεύουν" τους branch predictor.

Στα επισυναπτόμενα αρχεία υπάρχουν τα αρχεία CLIENT08.out.bz2 και WS04.out.bz2 (και τα δύο είναι συμπιεσμένα με bzip2), τα οποία αποτελούν τα traces που δημιουργήσαμε. Επειδή δεν ήταν εύκολο να εντοπίσουμε ακολουθίες εντολών και διευθύνσεων που προκαλούν miss στο branch predictor, τα δύο traces περιέχουν περίπου 20.000.000 εντολές συνολικά το καθένα (με αναπαραγωγή των ίδιων κομματιών δεν θα είχαμε το επιθυμητό αποτέλεσμα διότι θα είχαμε hit λόγο της ύπαρξης του global history των branch predictor). Εμείς εκτυπώνουμε ανά 1.000.000 εντολές).

Αναλυτικότερα για τις καλύτερες παραμέτρους των branch predictor πετύχαμε τα εξής:

CLIENT08 output

```
********************  CBP3  Start ***********************
Trace:            ./traces/CLIENT08.bz2
Uops to simulate: Whole Trace

Trace is compressed by bzip2.

Loading trace...
Trace: start_cycle:2020348 end_cycle:23171888 start fetch_q:63
alloc_q:63
Trace loaded.

Reader max_mem: 167.88 M
Reader info_mem: 1.20 M

*********  RUN  1   *********
Predictor:gshare
config: 262144 counters, 64 KB cost
nnnnnnn cycles 0   num_cond_br_dfff  0 num_ind_br_dfff  0
msp_cond_br_dfff  0 msp_ind_cond_br_dfff  0
nnnnnnn cycles 1000000   num_cond_br_dfff  180449 num_ind_br_dfff
54402 msp_cond_br_dfff  17643 msp_ind_cond_br_dfff  54402
nnnnnnn cycles 2000000   num_cond_br_dfff  94257 num_ind_br_dfff
13461 msp_cond_br_dfff  11050 msp_ind_cond_br_dfff  13462
nnnnnnn cycles 3000000   num_cond_br_dfff  148020 num_ind_br_dfff
40913 msp_cond_br_dfff  12239 msp_ind_cond_br_dfff  40913
nnnnnnn cycles 4000000   num_cond_br_dfff  114117 num_ind_br_dfff
24167 msp_cond_br_dfff  5288 msp_ind_cond_br_dfff  24166
nnnnnnn cycles 5000000   num_cond_br_dfff  181302 num_ind_br_dfff
24216 msp_cond_br_dfff  5575 msp_ind_cond_br_dfff  24217
nnnnnnn cycles 6000000   num_cond_br_dfff  173901 num_ind_br_dfff
49546 msp_cond_br_dfff  14854 msp_ind_cond_br_dfff  49546
nnnnnnn cycles 7000000   num_cond_br_dfff  166428 num_ind_br_dfff
46853 msp_cond_br_dfff  13687 msp_ind_cond_br_dfff  46854
nnnnnnn cycles 8000000   num_cond_br_dfff  168873 num_ind_br_dfff
47143 msp_cond_br_dfff  13430 msp_ind_cond_br_dfff  47141
nnnnnnn cycles 9000000   num_cond_br_dfff  161733 num_ind_br_dfff
38201 msp_cond_br_dfff  10834 msp_ind_cond_br_dfff  38202
nnnnnnn cycles 10000000   num_cond_br_dfff  151389
num_ind_br_dfff  35524 msp_cond_br_dfff  8267
msp_ind_cond_br_dfff  35523
nnnnnnn cycles 11000000   num_cond_br_dfff  200825
num_ind_br_dfff  28078 msp_cond_br_dfff  5617
msp_ind_cond_br_dfff  28078
nnnnnnn cycles 12000000   num_cond_br_dfff  189076
num_ind_br_dfff  51947 msp_cond_br_dfff  13489
msp_ind_cond_br_dfff  51947
nnnnnnn cycles 13000000   num_cond_br_dfff  163331
num_ind_br_dfff  46396 msp_cond_br_dfff  13611
msp_ind_cond_br_dfff  46397
nnnnnnn cycles 14000000   num_cond_br_dfff  168962
num_ind_br_dfff  49094 msp_cond_br_dfff  14225
msp_ind_cond_br_dfff  49093
nnnnnnn cycles 15000000   num_cond_br_dfff  167628
num_ind_br_dfff  47107 msp_cond_br_dfff  13457
msp_ind_cond_br_dfff  47107
```

```
nnnnnnn cycles 16000000   num_cond_br_dfff  144421
num_ind_br_dfff  30360 msp_cond_br_dfff  9163
msp_ind_cond_br_dfff  30360
nnnnnnn cycles 17000000   num_cond_br_dfff  188955
num_ind_br_dfff  25539 msp_cond_br_dfff  3881
msp_ind_cond_br_dfff  25539
nnnnnnn cycles 18000000   num_cond_br_dfff  177244
num_ind_br_dfff  41328 msp_cond_br_dfff  12793
msp_ind_cond_br_dfff  41328
nnnnnnn cycles 19000000   num_cond_br_dfff  151873
num_ind_br_dfff  31515 msp_cond_br_dfff  10309
msp_ind_cond_br_dfff  31515
nnnnnnn cycles 20000000   num_cond_br_dfff  157570
num_ind_br_dfff  35531 msp_cond_br_dfff  8710
msp_ind_cond_br_dfff  35531
nnnnnnn cycles 21000000   num_cond_br_dfff  210374
num_ind_br_dfff  32027 msp_cond_br_dfff  6730
msp_ind_cond_br_dfff  32028
Trace signature self checking: passed!

Total cycles:                          21151541

Num_Inst:                              28288713
Num_Uops:                              50000001

Num_cond_br:                           3488924
Mispred_cond_br:                       225849
Mispred_penalty_cond_br:               20372754
Conditional_MPKI:                      7.9837
Conditional_MR:                        0.0647
Final Score Run1_Conditional_MPPKI:    720.1725

Num_ind_br:                            799106
Mispred_ind_br:                        799106
Mispred_penalty_cond_br:               63734624
Indirect_MPKI:                         28.2482
Indirect_MR:                           1.0000
Final Score Run1_Indirect_MPPKI:       2253.0054

Simulation Time: 32 Seconds

********************  CBP3  End   ************************
Total Simulation Time: 38 Seconds
```

WS04 output

```
********************  CBP3  Start ************************
Trace:            ./traces/WS04.bz2
Uops to simulate: Whole Trace

Trace is compressed by bzip2.

Loading trace...
Trace: start_cycle:3250697 end_cycle:26546631 start fetch_q:63
alloc_q:63
Trace loaded.
```

```
Reader max_mem: 226.88 M
Reader info_mem: 1.04 M

*********  RUN  1   *********
Predictor:gshare
config: 262144 counters, 64 KB cost
nnnnnnn cycles 0   num_cond_br_dfff  0 num_ind_br_dfff  0
msp_cond_br_dfff  0 msp_ind_cond_br_dfff  0
nnnnnnn cycles 1000000   num_cond_br_dfff  229782 num_ind_br_dfff
0 msp_cond_br_dfff  91491 msp_ind_cond_br_dfff  0
nnnnnnn cycles 2000000   num_cond_br_dfff  141476 num_ind_br_dfff
2952 msp_cond_br_dfff  17064 msp_ind_cond_br_dfff  2956
nnnnnnn cycles 3000000   num_cond_br_dfff  182565 num_ind_br_dfff
28236 msp_cond_br_dfff  54554 msp_ind_cond_br_dfff  28232
nnnnnnn cycles 4000000   num_cond_br_dfff  207657 num_ind_br_dfff
538 msp_cond_br_dfff  58221 msp_ind_cond_br_dfff  540
nnnnnnn cycles 5000000   num_cond_br_dfff  106618 num_ind_br_dfff
30924 msp_cond_br_dfff  6549 msp_ind_cond_br_dfff  30922
nnnnnnn cycles 6000000   num_cond_br_dfff  238792 num_ind_br_dfff
43 msp_cond_br_dfff  86569 msp_ind_cond_br_dfff  43
nnnnnnn cycles 7000000   num_cond_br_dfff  152685 num_ind_br_dfff
1576 msp_cond_br_dfff  21025 msp_ind_cond_br_dfff  1576
nnnnnnn cycles 8000000   num_cond_br_dfff  171426 num_ind_br_dfff
29628 msp_cond_br_dfff  47300 msp_ind_cond_br_dfff  29628
nnnnnnn cycles 9000000   num_cond_br_dfff  217187 num_ind_br_dfff
417 msp_cond_br_dfff  64589 msp_ind_cond_br_dfff  417
nnnnnnn cycles 10000000   num_cond_br_dfff  97717 num_ind_br_dfff
21422 msp_cond_br_dfff  5261 msp_ind_cond_br_dfff  21422
nnnnnnn cycles 11000000   num_cond_br_dfff  194868
num_ind_br_dfff  10148 msp_cond_br_dfff  64764
msp_ind_cond_br_dfff  10148
nnnnnnn cycles 12000000   num_cond_br_dfff  192362
num_ind_br_dfff  767 msp_cond_br_dfff  47009 msp_ind_cond_br_dfff
767
nnnnnnn cycles 13000000   num_cond_br_dfff  133105
num_ind_br_dfff  30435 msp_cond_br_dfff  21675
msp_ind_cond_br_dfff  30435
nnnnnnn cycles 14000000   num_cond_br_dfff  240088
num_ind_br_dfff  43 msp_cond_br_dfff  86681 msp_ind_cond_br_dfff
43
nnnnnnn cycles 15000000   num_cond_br_dfff  127461
num_ind_br_dfff  11054 msp_cond_br_dfff  6437
msp_ind_cond_br_dfff  11058
nnnnnnn cycles 16000000   num_cond_br_dfff  197400
num_ind_br_dfff  20400 msp_cond_br_dfff  62233
msp_ind_cond_br_dfff  20396
nnnnnnn cycles 17000000   num_cond_br_dfff  194496
num_ind_br_dfff  717 msp_cond_br_dfff  48361 msp_ind_cond_br_dfff
717
nnnnnnn cycles 18000000   num_cond_br_dfff  130996
num_ind_br_dfff  30471 msp_cond_br_dfff  20233
msp_ind_cond_br_dfff  30471
nnnnnnn cycles 19000000   num_cond_br_dfff  241533
num_ind_br_dfff  0 msp_cond_br_dfff  85545 msp_ind_cond_br_dfff
0
```

```
nnnnnnn cycles 20000000   num_cond_br_dfff  121197
num_ind_br_dfff  5052 msp_cond_br_dfff  6232 msp_ind_cond_br_dfff
5057
nnnnnnn cycles 21000000   num_cond_br_dfff  186324
num_ind_br_dfff  26200 msp_cond_br_dfff  57100
msp_ind_cond_br_dfff  26195
nnnnnnn cycles 22000000   num_cond_br_dfff  204273
num_ind_br_dfff  585 msp_cond_br_dfff  55588 msp_ind_cond_br_dfff
585
nnnnnnn cycles 23000000   num_cond_br_dfff  119461
num_ind_br_dfff  30646 msp_cond_br_dfff  12938
msp_ind_cond_br_dfff  30646
Trace signature self checking: passed!

Total cycles:                          23295935

Num_Inst:                              42192243
Num_Uops:                              50000001

Num_cond_br:                           4100952
Mispred_cond_br:                       1053733
Mispred_penalty_cond_br:               71877121
Conditional_MPKI:                      24.9746
Conditional_MR:                        0.2569
Final Score Run1_Conditional_MPPKI:    1703.5625

Num_ind_br:                            282254
Mispred_ind_br:                        282254
Mispred_penalty_cond_br:               22354313
Indirect_MPKI:                         6.6897
Indirect_MR:                           1.0000
Final Score Run1_Indirect_MPPKI:       529.8204

Simulation Time: 36 Seconds

********************  CBP3  End   ************************
Total Simulation Time: 45 Seconds
```

- Gshare

./cbp3 -t ../traces/CLIENT08.bz2
Misspredictions: 990338 Instructions: 21151541 → 23,0954% miss

Για το CLIENT08 παρατηρούμε ότι οι περισσότερες λανθασμένες αποφάσεις παίρνονται μεταξύ των εντολών:

[0-1.000.000] → 30,676%
[5.000.000-6.000.000] → 28,821%
[13.000.000-14.000.000] → 29,037%

Παρατηρούμε ότι μεταξύ των παραπάνω εντολών το misspredict αυξάνεται σε 29,511%

Ομοίως για το WS04 έχουμε:

./cbp3 -t ../traces/WS04.bz2
Misspredictions: 1121267 Instructions: 23295935 → 25,581% miss

Για το WS04 παρατηρούμε ότι οι περισσότερες λανθασμένες αποφάσεις παίρνονται μεταξύ των εντολών:

[0-1.000.000] → 39.816%
[2.000.000-3.000.000] → 39.314%
[20.000.000-21.000.000] → 39,193%

Στο WS04 Trace ο αριθμός των misspredict̲ion αυξάνεται σε 39,441%

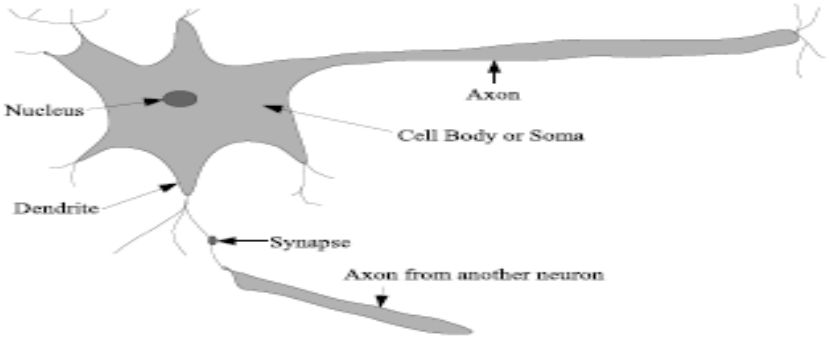
9 Παράρτημα

A. Perceptrons

Το 2001, ο D. Jimenez πρότεινε τη χρησιμοποίηση ενός perceptron για να αντικαταστήσει τους συνηθισμένους μετρητές για την πρόβλεψη κλάδων [8]. Η βασική ιδέα είναι να μαθευτεί η συμπεριφορά των κλάδων καθώς ένα πρόγραμμα εκτελείται. Τα perceptrons είναι μια μέθοδος εκμάθησης που υιοθετείται από την τεχνητή νοημοσύνη.

Το perceptron προτάθηκε αρχικά από τον McCullough και τον Pitts το 1947 για να ξανακατασκευάσει πανομοιότυπα την πυρκαγιά των νευρώνων στον εγκέφαλο [12]. Ένας νευρώνας, ή το κύτταρο νεύρων, είναι το κύριο κύτταρο στο ανθρώπινο νευρικό σύστημα. Κάθε νευρώνας αποτελείται από τρία βασικά μέρη: ένα σώμα ή ένα σώμα κυττάρων, που αποτελεί τον πυρήνα του τους δενδρίτες, μέσω των οποίων λαμβάνει σήματα από γειτονικούς νευρώνες (σημεία εισόδου) και από ένα νευρίτη, που είναι η έξοδος του νευρώνα και το μέσο σύνδεσης του με άλλους νευρώνες όπως φαίνεται στο σχήμα 15. Οι νευρώνες διαδίδουν τα σήματα από έναν νευρίτη σε έναν άλλον νευρώνα δενδρίτη. Σε κάθε δενδρίτη υπάρχει ένα απειροελάχιστο κενό που ονομάζεται σύναψη, η οποία μέσω μιας ηλεκτροχημικής αντίδρασης παράγει ένα σήμα που περνά μέσω του δενδρίτη στο σώμα. Επάνω στη λήψη των σημάτων από τους δενδρίτες, ο νευρώνας πιθανόν να στείλει ή πιθανόν να μη στείλει ένα σήμα μέσω του νευρίτη του που συνδέεται πιθανότατα με τις συνάψεις άλλων νευρώνων. Όταν αυτό το σήμα στέλνεται ο νευρώνας λέγεται ότι είναι «πυρκαγιά.» Ενώ δεν είναι σαφές πώς ο νευρώνας μαθαίνει πότε είναι σε «πυρκαγιά.» (αυτό συμβαίνει όταν τα ηλεκτροχημικά σήματα που εισέρχονται στο σώμα μέσω των δενδριτών συνδυάζονται και το αποτέλεσμα τους ξεπερνά κάποια τιμή κατωφλίου και έτσι το σήμα διαδίδεται με τη βοήθεια του άξονα προς άλλους νευρώνες) και πότε δεν είναι πολλά πρότυπα έχουν προταθεί για να ξαναδιπλώσουν αυτήν την δράση, η απλούστερη είναι η ύπαρξη ενός perceptron [12].

Σχήμα 15: Τραχύ σκίτσο των μερών ενός νευρώνα



A.1 Καθορισμός του perceptron

Το perceptron είναι η πιο απλή τροπολογία δικτύου με απλή τροφοδότηση και αποτελεί ιστορικά μια πρώτη προσέγγιση τεχνητών νευρωνικών δικτύων. Πρόκειται ουσιαστικά για ένα και μοναδικό τεχνητό νευρώνα, όμοιο με αυτόν που απεικονίζεται στο σχήμα 16 ο οποίος χρησιμοποιεί ως συνάρτηση κατωφλίου τη βηματική συνάρτηση. Η μάθηση στο perceptron συνίσταται στην επιλογή κατάλληλων τιμών βαρών έτσι ώστε δεδομένου ενός διανύσματος εισόδου να παραχθεί η επιθυμητή έξοδος. Πρόκειται δηλαδή για μια απλή μορφή μάθησης υπό επίβλεψη. Ο αλγόριθμος μεταβολής των βαρών έχει ως εξής:

1. Εάν για τη συγκεκριμένη είσοδο παράγεται το επιθυμητό αποτέλεσμα τότε δε γίνεται καμιά μεταβολή
2. Εάν το αποτέλεσμα είναι 1 ενώ θα έπρεπε να είναι 0, τότε μειώνονται τα βάρη των ενεργών γραμμών (εκείνες που στο συγκεκριμένο πρότυπο έχουν είσοδο 1) κατά μια τιμή d, η οποία ονομάζεται ρυθμός μάθησης (learning rate).
3. Εάν το αποτέλεσμα είναι 0 ενώ θα έπρεπε να είναι 1, τότε αυξάνονται τα βάρη των ενεργών γραμμών κατά μια τιμή d.

Το perceptron πρότυπο, όπως φαίνεται στο σχήμα 16, υποθέτει ότι κάθε σήμα που μπαίνει στο νευρώνα μπορεί μόνο να πάρει κάποιο μικρό αριθμό τιμών, και ότι ο νευρώνας θα βάλει «φωτιά» όταν ένας γραμμικός συνδυασμός αυτών των τιμών υπερβαίνει κάποιο κατώτατο όριο. Για να καθορίσει αυτόν τον γραμμικό συνδυασμό, το perceptron κρατά μια τιμή βάρους, WI (weight), για κάθε τιμή ο ρόλος της οποίας είναι αντίστοιχος της σύναψης του βιολογικού εγκεφάλου (η τιμή βάρους μπορεί να είναι είτε θετική είτε αρνητική σε αντιστοιχία με την επιβραδυντική ή επιταχυντική λειτουργία της σύναψης) και, VI, όταν λαμβάνει. Το perceptron αθροίζει τους χρόνους βάρους την αξία και εφαρμόζουν μια λειτουργία κατώτατων ορίων "**Θ**" για να καθορίσουν την παραγωγή για perceptron, το οποίο παίρνει την τιμή 1όταν είναι έτοιμο για «πυρκαγιά.» και 0 όταν δεν είναι για «πυρκαγιά». Κατά συνέπεια για perceptron με n εισαγωγές έχουμε την έξοδο, α:

$$a = \Theta\left(w_0 + \sum_{i=1}^{n} w_i \cdot v_i\right)$$

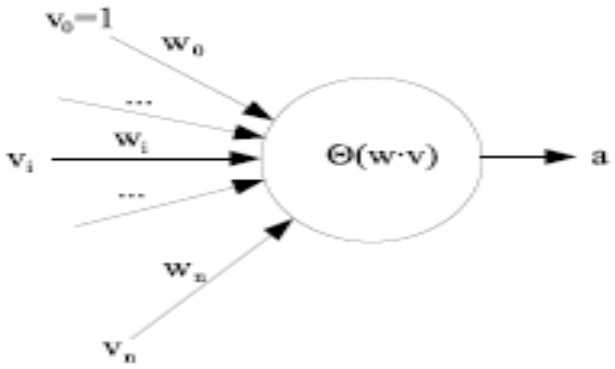
Συχνά το w_0 καλείται bias weight (προκατειλημμένο βάρος) και περιλαμβάνεται στο προϊόν ρυθμίζοντας το v0 = 1. Για τους σκοπούς της πρόβλεψης διακλαδώσεων, η λειτουργία κατώτατων ορίων είναι συνήθως:

$$\Theta(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

Όπου το «1» δείχνει τη λήψη του κλάδου και το «0» δείχνει ότι δεν έχει πάρει τον κλάδο. Κατά συνέπεια ο perceptron αποτελείται απλώς από ένα διάνυσμα των βαρών « W» και τη λειτουργία κατώτατων ορίων του, "**Θ**". Η εισαγωγή του perceptron είναι ένα διάνυσμα των τιμών <V>. Η διαδικασία εκμάθησης του perceptron

πραγματοποιείται όπως τα βάρη ενημερώνονται μετά από κάθε έξοδο. Εάν το perceptron δώσει τη σωστή έξοδο, κατόπιν δεν είναι απαραίτητο να ενημερωθούν τα βάρη.

Σχήμα 16: Το perceptron πρότυπο ενός νευρώνα.



Εάν το perceptron είναι ανακριβές, κατόπιν το διάνυσμα βάρους ενημερώνεται ως εξής:

$$w = \begin{cases} w + v & \text{if } a = 0 \\ w - v & \text{if } a = 1 \end{cases}$$

Μέσω αυτής της διαδικασίας ενημέρωσης, perceptron θα φθάσει τελικά στο σωστό διάνυσμα βάρους που παράγει τη σωστή έξοδο σε κάθε εισαγωγή, εάν η εισαγωγή είναι γραμμικά ευδιαχώριστη.

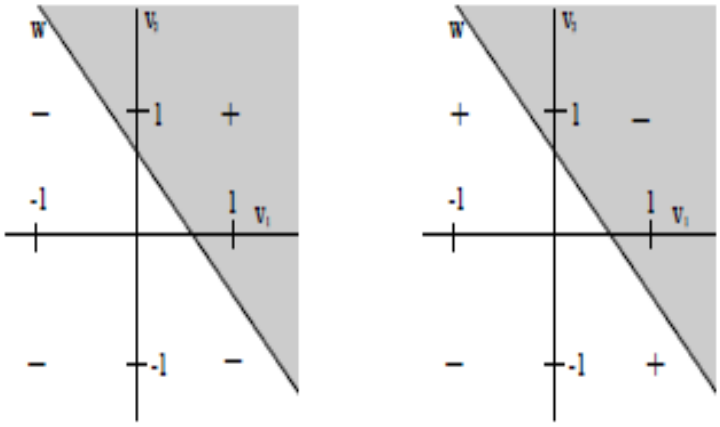
A.2 Οι χρήσεις του perceptron

Το μαθηματικό πρότυπο που παρουσιάζεται για το perceptron επιτρέπει την ταξινόμηση της εισαγωγής στη μια από δύο κλάσεις. Το διάνυσμα βάρους καθορίζει ένα υπερπλάνο (hyperplane) σε n διαστάσεις, και η λειτουργία κατώτατων ορίων του διανύσματος βάρους που διαχωρίζεται με τις τιμές εισαγωγής λέει ποια πλευρά του hyperplane η είσοδος εμφανίζεται, όπως φαίνεται στο σχήμα 17 (a) για παράδειγμα. Σε αυτήν την περίπτωση το bias weight δίνει στο καθορισμένο υπερπλάνο τη δυνατότητα να μετατοπιστεί μακριά από την προέλευση, ταξινομώντας κατά

συνέπεια περισσότερους τύπους λειτουργιών. Όποτε το perceptron δίνει τη λανθασμένη απάντηση, οι ενημερώσεις μετατοπίζουν το υπερπλάνο, το μετατοπισμένο υπερπλάνο θα είναι σε θέση έπειτα να ταξινομήσει σωστά το σημείο διατηρώντας τη σωστή ταξινόμηση από τα προηγούμενα σημεία.

Αυτό το πρότυπο δεν μπορεί να μάθει τα μη γραμμικά ευδιαχώριστα στοιχεία. Η κλασική περίπτωση των μη γραμμικά ευδιαχώριστων στοιχείων είναι η αποκλειστική λειτουργία, OR ή XOR, όπως φαίνεται στο σχήμα 17 (b).

Σχήμα 17: Δύο παραδείγματα perceptron που ταξινομεί τις λειτουργίες με ν = 2 17(a). Η And λειτουργία είναι γραμμικά ευδιαχώριστη και έτσι σωστά ταξινομημένη 17(b)
Η λειτουργία XOR δεν είναι γραμμικά ευδιαχώριστη και όχι σωστά ταξινομημένη.



(a) Example of AND function.

(b) Example of XOR function.

Η λειτουργία XOR δεν μπορεί να ταξινομηθεί από ένα ενιαίο hyperplane και έτσι ούτε από ένα perceptron. Η λύση είναι να χρησιμοποιηθεί ένα δίκτυο από perceptrons. Ένα δίκτυο perceptrons έχει ένα στρώμα perceptrons όπου οι έξοδοί τους τροφοδοτούνται στο επόμενο στρώμα perceptrons μέχρι την έξοδο. Δεδομένου ότι κάθε παραγωγή εξαρτάται από τα βάρη και τις τιμές των perceptrons στο προηγούμενο στρώμα, δεν υπάρχει ένας καλός κανόνας ενημερώσεων που θα εγγυηθεί ότι το δίκτυο μαθαίνει τα δεδομένα. Ο πιο συνηθισμένος ευρετικός αλγόριθμος είναι ο πίσω αλγόριθμος διάδοσης, (Back Propagation Algorithm) ως αλγόριθμος βασισμένος στη μέθοδο μέγιστης καθόδου. Λόγω της πολυπλοκότητας αυτού του αλγορίθμου, δεν είναι κατάλληλος για την πρόβλεψη διακλαδώσεων δεδομένου ότι όλοι οι υπολογισμοί πρέπει να πραγματοποιηθούν στο ίδιο chip.

B. Framework

Παρακάτω ακολουθεί ο κώδικας εξομοίωσης για όλα τα traces αρχεία. Εμείς εκτυπώνουμε τα cycles, τα Mispred cond_br, τα Mispred_ind_br καθώς και το συνολικό αριθμό cond_br και ind_br. Επειδή τα cycles είναι εκατομμύρια (πάνω από 20.000.000) και επειδή όπως αναφέραμε δεν ήταν εύκολο να εντοπίσουμε και να μελετήσουμε ακολουθίες εντολών και διευθύνσεων που προκαλούν mispred στο branch predictor αποφασίσαμε να εκτυπώνουμε ανά 1.000.000 εντολές.

```
#include <stdio.h>
#include <stdlib.h>
#include <cassert>
#include <string.h>
#include <inttypes.h>
#include <fstream>
#include <time.h>

using namespace std;
#include "cbp3_def.h"
#include "cbp3_reader.h"
#include "cbp3_framework.h"
#include "predictor.h"

uint32_t num_run = 0;
void run(int sim_len);

int main ( int argc, char *argv[] )
{
    time_t time_start = time(NULL);

// process command inputs
    char tfile[500];
    int sim_len = -1; // to the end of trace
    if ( argc != 3 && argc != 5) {
        printf( "usage: %s -t /path-to/trace [-u num-uops-to-
simulate]\n", argv[0]);
        exit (1);
    }else {
        int targ = -1, uarg = -1;
        bool wrong = false;
        for (int i = 1; i <= argc - 2; i += 2) {
            if (strcmp(argv[i], "-t") == 0) targ = i + 1;
            else if (strcmp(argv[i], "-u") == 0) uarg = i + 1;
            else
                wrong = true;
        }
        if (targ == -1 || wrong)  {
            printf( "usage: %s -t /path-to/trace [-u num-uops-to-
simulate]\n", argv[0]);
            exit (1);
        }
```

```
        if (uarg != -1) {
            sim_len = atoi(argv[uarg]);
            if (sim_len <= 0) sim_len = -1;
        }
        assert(strlen(argv[targ]) < 500);
        strcpy(tfile, argv[targ]);
    }

// load the trace into memory
    printf("********************  CBP3  Start
**********************\n");
    printf("Trace:            %s\n", tfile);
    if (sim_len == -1)
        printf("Uops to simulate: Whole Trace\n\n");
    else
        printf("Uops to simulate: %i\n\n", sim_len);

    // check whether trace is compressed by bzip2
    bool bzip2 = false;
    if (strlen(tfile) > 3)
        bzip2 = (tfile[strlen(tfile) - 1] == '2' &&
                tfile[strlen(tfile) - 2] == 'z' &&
                tfile[strlen(tfile) - 3] == 'b');
    if (bzip2) {
        printf("Trace is compressed by bzip2.\n");
    }

    char cmd[600];
    sprintf (cmd, "%s %s", bzip2 ? "bzip2 -dc" : "cat", tfile);
    FILE *tracefp = popen (cmd, "r");
    if (!tracefp) {
        printf("ERROR: could not open trace: %s\n", tfile);
        exit (1);
    }
    printf("\nLoading trace...\n");
    ReaderLoadTrace(tracefp, true, false, 17, 16);
    printf("Trace loaded.\n\n");
    fclose(tracefp);

// initialize predictors
    PredictorInit();

    printf("Reader max_mem: %.2f M\n",
(double)ReaderMaxMem()/(1024.0*1024.0));
    printf("Reader info_mem: %.2f M\n",
(double)ReaderTraceInfoSize()/(1024.0*1024.0));

// keep running if rewind_marked is set
// the length of each run is decided by sim_len
//    while (num_run == 0 || rewind_marked) {
        time_t run_start = time(NULL);
        run(sim_len);
        printf("Simulation Time: %ld Seconds\n", (time(NULL) -
run_start));
        ReaderRewind();
//    }

// the end...
```

```
    printf("\n********************  CBP3  End
**********************\n");
    printf("Total Simulation Time: %ld Seconds\n", (time(NULL) -
time_start));
    PredictorExit();

    return 1;
}

// oracle info about an uop from the reader
cbp3_uop_dynamic_t uop_info[FETCHQ_SIZE + ROB_SIZE];

// live uop info that will be availabe to the predictor
// depending on the stage of the uop, fields are copied from
uop_info array
cbp3_queue_entry_t live_uop[FETCHQ_SIZE + ROB_SIZE];

uint32_t reg_value[NUM_REGS];      // architectural reg file
filled at retire
bool     reg_valid[NUM_REGS];

// renaming table
// it is indexed by logical reg number and pointing to a rob
entry that holds the producer uop of this reg
int      rename_table[NUM_REGS];
int      rename_table_flags;

bool     rewind_marked;
uint32_t cycle;
cbp3_cycle_activity_t cycle_info;

uint8_t  allocate_fetchq;    // fetch q ptr of the uop that will
be allocated next
bool     first_fetch;

// stats
uint32_t total_cycle;
uint32_t num_insts;
uint32_t num_uops;
uint32_t num_cond_br;
uint32_t num_ind_br;
uint32_t penalty_cond_br;    // misprediction penalty cycles of
conditional branches
uint32_t penalty_ind_br;     // misprediction penalty cycles of
indirect branches
uint32_t msp_cond_br;        // number of mispredictions of
conditional branches

 uint32_t   num_cond_br_my ;
 uint32_t   num_ind_br_my ;
 uint32_t   msp_cond_br_my ;
 uint32_t   msp_ind_br_my ;
```

```
uint32_t msp_ind_br;         // number of mispredictions of
indirect branches

void reset() {
    for (int i = 0; i < (FETCHQ_SIZE + ROB_SIZE); i++) {
        live_uop[i].reset();
        uop_info[i].reset();
    }
    for (int i = 0; i < NUM_REGS; i++) {
        reg_value[i] = 0;
        reg_valid[i] = false;
        rename_table[i] = -1; // -1 means producer is not in rob
        rename_table_flags = -1;
    }

    rewind_marked = false;
    cycle = 0;
    allocate_fetchq = 0;
    first_fetch = true;

    total_cycle = 0;
    num_insts = 0;
    num_uops = 0;
    num_cond_br = 0;
    num_ind_br = 0;
    penalty_cond_br = 0;
    penalty_ind_br = 0;
    msp_cond_br = 0;
    msp_ind_br = 0;



    num_cond_br_my = 0;
    num_ind_br_my = 0;
    msp_cond_br_my = 0;
    msp_ind_br_my = 0;







    PredictorReset();
}

// for debug: compare two uops
bool uop_check(cbp3_uop_dynamic_t *u1, cbp3_uop_dynamic_t *u2) {
    for (int i = 0; i < UOP_SRCS_MAX; i++) {
        if (u1->srcs[i] != u2->srcs[i]) return false;
        if (u1->src_data[i] != u2->src_data[i]) return false;
        if (u1->srcs_static[i] != u2->srcs_static[i]) return
false;
    }
    for (int i = 0; i < 4; i++)
        if (u1->ldst_data[i] != u2->ldst_data[i]) return false;
    if (u1->dst != u2->dst ||
        u1->dst_data != u2->dst_data ||
        u1->wflags != u2->wflags ||
```

```
        u1->rflags != u2->rflags)
        return false;
    if (u1->vaddr != u2->vaddr ||
        u1->br_target != u2->br_target ||
        u1->br_taken != u2->br_taken ||
        u1->uop_id != u2->uop_id ||
        u1->inst_size != u2->inst_size)
        return false;
    if (u1->pc != u2->pc ||
        u1->type != u2->type ||
        u1->opsize != u2->opsize ||
        u1->dst_static != u2->dst_static)
        return false;
    return true;
}

void run(int sim_len) {
    num_run ++;
    printf("\n*********  RUN  %i   *********\n", num_run);
    reset();
    uint16_t num_stages[NUM_STAGES];

    while (ReaderRunACycle(num_stages) && // trace does not end
            (sim_len == -1 ? 1 : num_uops < (uint32_t)sim_len)) {
// in the simulation range
        // get info from the reader
        const cbp3_cycle_info_t *reader = ReaderInfo();

        cycle = reader->cycle;
        // fill cycle_info
        cycle_info.cycle = reader->cycle;
        cycle_info.num_fetch = num_stages[0];
        cycle_info.num_allocate = num_stages[1];
        cycle_info.num_exe = num_stages[2];
        cycle_info.num_retire = num_stages[3];
        cycle_info.num_agu = num_stages[4];
        cycle_info.num_std = num_stages[5];

        // fill in different fields into live_uop array for each
stage

        for (uint32_t i = 0; i < cycle_info.num_retire; i ++) {
            uint8_t q = reader->retire_q[i];
            cycle_info.retire_q[i] = q;
            cbp3_queue_entry_t &uop = live_uop[q + FETCHQ_SIZE];
            // fill reg file
            if (!reg_is_constant(uop.uop.dst)) {
                reg_valid[uop.uop.dst] = true;
                reg_value[uop.uop.dst] = uop.uop.dst_data;
            }

            uop.cycle_retire = cycle;

            uint16_t type = uop.uop.type;
            if (type & IS_EOM)            num_insts ++; // the
uop is end of an inst
            num_uops ++;
            if (type & IS_BR_CONDITIONAL) num_cond_br ++;
```

```
            if (type & IS_BR_INDIRECT) num_ind_br ++;

            //assert(uop_check(&live_uop[q + FETCHQ_SIZE].uop,
&uop_info[q + FETCHQ_SIZE]));

            // uop retired, fix rename_table
            if (!reg_is_constant(uop.uop.dst) &&
rename_table[uop.uop.dst] == q)
                rename_table[uop.uop.dst] = -1;
            // need to check all fp regs because of fxchg
swapping regs
            // reg 35 is MM0
            for (int k = 35; k < 35 + 8; k++) {
                if (rename_table[k] == q) {
                    rename_table[k] = -1;
                }
            }
            if ((type & IS_WFLAGS) && rename_table_flags == q)
                rename_table_flags = -1;
        }

        for (uint32_t i = 0; i < cycle_info.num_exe; i ++) {
            uint8_t q = reader->exe_q[i];
            cycle_info.exe_q[i] = q;
            cbp3_queue_entry_t &uop = live_uop[q + FETCHQ_SIZE];
            cbp3_uop_dynamic_t &reader_uop = uop_info[q +
FETCHQ_SIZE];

            assert(uop.valid);
            for (int s = 0; s < UOP_SRCS_MAX; s++)
                uop.uop.src_data[s] = reader_uop.src_data[s];
            uop.uop.dst_data = reader_uop.dst_data;
            uop.uop.wflags = reader_uop.wflags;
            uop.uop.rflags = reader_uop.rflags;
            for (int s = 0; s < 4; s++)
                uop.uop.ldst_data[s] = reader_uop.ldst_data[s];

            // calculate mispred info
            uint16_t type = uop.uop.type;
            if (type & IS_BR_CONDITIONAL) {
                bool msp = (uop.uop.br_taken != (uop.last_pred >
0));
                if (!uop.cycle_last_pred) { // no prediction
provided
                    msp = true;
                    penalty_cond_br += (cycle - uop.cycle_fetch);
                }else {
                    assert(uop.cycle_last_pred < cycle);
                    penalty_cond_br += ((msp ? cycle :
uop.cycle_last_pred) - uop.cycle_fetch);
                }
                msp_cond_br += msp;
            }
            if (type & IS_BR_INDIRECT) {
                bool msp = (uop.uop.br_target != uop.last_pred);
                if (!uop.cycle_last_pred) {
                    msp = true;
                    penalty_ind_br += (cycle - uop.cycle_fetch);
```

```
                }else {
                    assert(uop.cycle_last_pred < cycle);
                    penalty_ind_br += ((msp ? cycle :
uop.cycle_last_pred) - uop.cycle_fetch);
                }
                msp_ind_br += msp;
            }

            uop.cycle_exe = cycle;
        }

        for (uint32_t i = 0; i < cycle_info.num_agu; i ++) {
            uint8_t q = reader->agu_q[i];
            cycle_info.agu_q[i] = q;
            cbp3_queue_entry_t &uop = live_uop[q + FETCHQ_SIZE];
            cbp3_uop_dynamic_t &reader_uop = uop_info[q +
FETCHQ_SIZE];
            assert(uop.valid);
            uop.uop.vaddr = reader_uop.vaddr;
            uop.cycle_agu = cycle;
        }

        for (uint32_t i = 0; i < cycle_info.num_std; i ++) {
            uint8_t q = reader->std_q[i];
            cycle_info.std_q[i] = q;
            cbp3_queue_entry_t &uop = live_uop[q + FETCHQ_SIZE];
            assert(uop.valid);
            uop.cycle_std = cycle;
        }

        for (uint32_t i = 0; i < cycle_info.num_allocate; i ++) {
            uint8_t q = reader->allocate_q[i];
            // must allocate from fetch q entry pointed by
allocate_fetchq
            cbp3_queue_entry_t &uop = live_uop[allocate_fetchq];
            // rob entry ptr is from trace
            cycle_info.allocate_q[i] = q;
            cbp3_queue_entry_t &uop_rob = live_uop[q +
FETCHQ_SIZE];

            // copy uop_info from fetch queue position to its
corresponding rob position
            uop_info[q + FETCHQ_SIZE] =
uop_info[allocate_fetchq];
            cbp3_uop_dynamic_t &reader_uop = uop_info[q +
FETCHQ_SIZE];

            allocate_fetchq = (allocate_fetchq + 1) %
FETCHQ_SIZE;
            assert(uop.valid && !uop_rob.valid);
            // move uop from fetch queue to rob
            uop_rob = uop;
            // invalid the fetch queue entry
            uop.reset();

            // fill fields that are available at allocation stage
            uop_rob.uop.opsize = reader_uop.opsize;
            uop_rob.uop.dst_static = reader_uop.dst_static;
```

```
            uop_rob.uop.dst = reader_uop.dst;
            for (int s = 0; s < UOP_SRCS_MAX; s++) {
                uop_rob.uop.srcs_static[s] =
reader_uop.srcs_static[s];
                uop_rob.uop.srcs[s] = reader_uop.srcs[s];
            }

            uop_rob.cycle_allocate = cycle;

            if (uop_rob.uop.type & (1 << 13)) {
                int tmp = rename_table[uop_rob.uop.srcs[0]];
                rename_table[uop_rob.uop.srcs[0]] =
rename_table[uop_rob.uop.srcs[1]];
                rename_table[uop_rob.uop.srcs[1]] = tmp;
            }else if (!reg_is_constant(uop_rob.uop.dst)) {
                rename_table[uop_rob.uop.dst] = q;
            }

            if (uop_rob.uop.type & IS_WFLAGS)
                rename_table_flags = q;
        }

        for (uint32_t i = 0; i < cycle_info.num_fetch; i ++) {
            uint8_t q = reader->fetch_q[i];
            cycle_info.fetch_q[i] = q;
            if (first_fetch) {
                allocate_fetchq = q; // allocate will start from
this fetch q
                first_fetch = false;
            }
            cbp3_queue_entry_t &uop = live_uop[q];
            // put oracle uop info in an array and fill into
live_uop later
            uop_info[q] = reader->uopinfo[i];
            cbp3_uop_dynamic_t &reader_uop = uop_info[q];
            // uop_id is not supposed to be used in predictors
            reader_uop.uop_id = 0;

            assert(!uop.valid);
            uop.valid = true;
            // fill fields that are available at fetch stage
            uop.uop.uop_id = reader_uop.uop_id;
            uop.uop.inst_size = reader_uop.inst_size;
            uop.uop.pc = reader_uop.pc;
            uop.uop.type = reader_uop.type;
            // yes, br_target and direction is available for br
history update
            uop.uop.br_target = reader_uop.br_target;
            uop.uop.br_taken = reader_uop.br_taken;
            uop.cycle_fetch = cycle;
        }

        PredictorRunACycle();

        // release rob entries of retired uops
        for (uint32_t i = 0; i < cycle_info.num_retire; i ++) {
            uint8_t q = reader->retire_q[i];
            live_uop[q + FETCHQ_SIZE].reset();
```

```
        }


//    if (  msp_cond_br != msp_cond_br_my)
      if ( total_cycle % 10000000 == 0 )
      {

//    num_cond_br_my = 0;
//    num_ind_br_my = 0;
//    msp_cond_br_my = 0;
//    msp_ind_br_my = 0;


    printf("nnnnnnn cycles %d   num_cond_br_dfff  %d
num_ind_br_dfff  %d msp_cond_br_dfff  %d msp_ind_cond_br_dfff
%d\n", total_cycle, num_cond_br - num_cond_br_my, num_ind_br -
num_ind_br_my, msp_cond_br - msp_cond_br_my, msp_ind_br -
msp_ind_br_my );



//    printf("nnnnnnn cycles %d   num_cond_br  %d  num_ind_br %d
msp_cond_br %d   msp_ind_br   %d\n", total_cycle, num_cond_br -
num_cond_br_my,  num_ind_br, msp_cond_br,  msp_ind_br);



//    printf("mmmmmmmmmm  cycles %d   num_cond_br  %d  num_ind_br
%d   msp_cond_br %d   msp_ind_br   %d\n", total_cycle,
num_cond_br,  num_ind_br, msp_cond_br,  msp_ind_br);


            num_cond_br_my = num_cond_br ;

            num_ind_br_my = num_ind_br ;
            msp_cond_br_my =  msp_cond_br ;
            msp_ind_br_my  =  msp_ind_br ;
      }


//    msp_cond_br_my= msp_cond_br ;

      total_cycle++;

    };

    if (sim_len == -1) {
        // a complete run, check trace signature
        // if the check failed, something was wrong in the reader
        bool pass = ReaderTraceCheck();
        printf("Trace signature self checking: %s\n\n", pass ?
"passed!" : "failed!");
        if (!pass) exit(1);
    }
    // print out stats of a run
    printf("Total cycles:                          %d\n\n",
total_cycle);
```

```
    printf("Num_Inst:                                %d\n",
num_insts);
    printf("Num_Uops:                                %d\n\n",
num_uops);

    printf("Num_cond_br:                             %d\n",
num_cond_br);
    printf("Mispred_cond_br:                         %d\n",
msp_cond_br);
    printf("Mispred_penalty_cond_br:                 %d\n",
penalty_cond_br);
    printf("Conditional_MPKI:                        %.4f\n",
(double)msp_cond_br/(double)num_insts*1000);
    printf("Conditional_MR:                          %.4f\n",
(double)msp_cond_br/(double)num_cond_br);
    printf("Final Score Run%i_Conditional_MPPKI:   %.4f\n\n",
num_run, (double)penalty_cond_br/(double)num_insts*1000);

    printf("Num_ind_br:                              %d\n",
num_ind_br);
    printf("Mispred_ind_br:                          %d\n",
msp_ind_br);
    printf("Mispred_penalty_cond_br:                 %d\n",
penalty_ind_br);
    printf("Indirect_MPKI:                           %.4f\n",
(double)msp_ind_br/(double)num_insts*1000);
    printf("Indirect_MR:                             %.4f\n",
(double)msp_ind_br/(double)num_ind_br);
    printf("Final Score Run%i_Indirect_MPPKI:      %.4f\n\n",
num_run, (double)penalty_ind_br/(double)num_insts*1000);

    PredictorRunEnd();
}

const cbp3_cycle_activity_t *get_cycle_info() {
    return &cycle_info;
}

const cbp3_queue_entry_t *fetch_entry(uint8_t e) {
    assert(e < FETCHQ_SIZE);
    return &live_uop[e];
}

const cbp3_queue_entry_t *rob_entry(uint8_t e) {
    return &live_uop[e + FETCHQ_SIZE];
}

const int rename(uint8_t reg) {
    assert(reg < NUM_REGS);
    return rename_table[reg];
}

const int rename_flags() {
    return rename_table_flags;
}

const uint32_t reg_val(uint8_t reg) {
    assert(reg < NUM_REGS);
```

```
    return reg_valid[reg] ? reg_value[reg] : 0;
}

bool  report_pred(uint8_t ptr, bool in_rob, uint32_t pred) {
    if (!in_rob)
        assert(ptr < FETCHQ_SIZE);

    // get the uop from fetchq or rob
    cbp3_queue_entry_t * uop = &live_uop[in_rob ? (ptr +
FETCHQ_SIZE) : ptr];
    if (!uop->valid)
        return false;

    if (uop->cycle_exe) // uop has been executed
        return false;

    // record the prediction
    uop->cycle_last_pred = cycle;
    uop->last_pred = pred;
    return true;
}

void cbp3_queue_entry_t::reset() {
    uop.reset();
    valid = false;
    cycle_fetch = 0;
    cycle_allocate = 0;
    cycle_agu = 0;
    cycle_std = 0;
    cycle_exe = 0;
    cycle_retire = 0;
    cycle_last_pred = 0;
    last_pred = 0;
}
```

Πίνακας Συντομογραφιών

ΤΝΔ:	Τεχνητά Νευρωνικά Δίκτυα
PHT:	Pattern History Table
ILP:	Instruction Level Parallelism
ISA:	Instruction Set Architecture
CBP:	Championship Branch Prediction
RISC:	Reduced Instruction Set Computers
BTFNT:	Backward Taken Forward Not Taken
SPEC :	Standard Performance Evaluation Corporation
MPKI:	Misprediction Per Kilo Instruction
MPPKI:	Misprediction Penalty Per Kilo Instruction

Βιβλιογραφία - References.

[1] M. Evers, S. Patel, R. Chappell, and Y. Patt. An analysis of correlation and predictability: What makes two-level branch predictors work. ISCA, pages 52–61, 1998.

[2] M. Evers and T.Y. Yeh. Understanding branches and designing branch predictors for high-performance
microprocessors. Proceedings of the IEEE, 89(11):1610–1620, November 2001.

[3] A. Falc´on, J. Stark, A. Ramirez, K. Lai, and M. Valero. Prophet-critic hybrid branch prediction. 31st
Annual International Symposium on Computer Architecture, pages 250–262, June 2004.

[4] J. Hennessy and D. Patterson. Computer Architecture : A Quantitative Approach; third edition. Morgan Kaufmann, 2003.

[5] D. Jimenez. Fast path-based neural branch prediction. Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture, December 2003.

[6] D. Jimenez. Piecewise linear branch prediction. Proceedings of the 32nd International Symposium on
Computer Architecture (ISCA-32), June 2005.

[7] D. Jimenez and C. Lin. Dynamic branch prediction with perceptrons. Proceedings of the Seventh International Symposium on High Performance Computer Architecture, pages 197–206, 2001.

[8] D. Jimenez and C. Lin. Neural methods for dynamic branch prediction. ACM Transactions on Computer
Systems, 20(4):369–397, 2002. New York, NY, USA.

[9] G. Loh. The frankenpredictor: stitching together nasty bits of the other branch predictors. 1st Championship
Branch Prediction Contest (CBP1), pages 1–4, December 2004. Portland, OR, USA.

[10] G. Loh and D. Henry. Predicting conditional branches with fusion-based hybrid predictors. 11th Conference on Parallel Architectures and Compilation Techniques (PACT), pages 165–176, September 2002. Charlottesville, VA, USA.

[11] M. Monchiero and G. Palermo. The combined perceptron branch predictor. Proceedings of Euro-Par’05 – Parallel Computer Architecture and ILP Track, Septmeber 2005.

[12] S. Russell and P. Norvig. Artificial Intelligence: A Modern Approad; second edition. Prentice Hall, 2002.

[13] A. Seznec. Genesis of the ogehl predictor. Journal of Instruction Level Parallelism, April 2005.

[14] A. Seznec and P. Michaud. Dealiased hypbrid branch predictors. Technical Report PI-1229, IRISA,
February 1999.

[15] J. Stark and C. Wilkerson. Introduction to jilp's special addition for finalists of championship branch
prediction competition. Journal of Instruction Level Parallelism, 7:1–4, April 2005.

[16] D. Tarjan and K. Skadron. Merging path and gshare indexing in perceptron branch prediction. ACM
Transactions on Architecture Code Optimization, 2(3):280–300, 2005. New York, NY, USA.

[17] D. Wall. Limits of instruction level parallelism. David W. Wall, Limits of Instruction Level Parallelism,
Proc. 4th ASPLOS, 1991.

[18] T.-Y. Yeh and Yale N. Patt. Two-level adaptive branch prediction. In *Proceedings of the 24th* y *ACM/IEEE Int'l Symposium on Microarchitecture*, November 1991.

[19] V. Agarwal, M.S. Hrishikesh, S. W. Keckler, and D. Burger. Clock rate versus ipc: The end of the road for conventional microarchitectures. In *the 27th Annual International Symposium on Computer Architecture*, pages 248.259, May 2000.

[20] S. Sechrest, C.-C. Lee, and T.N. Mudge. Correlation and aliasing in dynamic branch predictors. In *Proceedings of the 23rd International Symposium on Computer Architecture*, May 1999.

[21] Scott McFarling. Combining branch predictors. Technical Report TN-36m, DigitalWestern Research Laboratory, June 1993.

[22] C.-C. Lee, C.C. Chen, and T.N. Mudge. The bi-mode branch predictor. In *Proceedings of the 30th Annual International Symposium on Microarchitecture*, November 1997.

[23] E. Sprangle, R.S. Chappell, M. Alsup, and Y. N. Patt. The Agree predictor: A mechanism for reducing negative branch history interference. In *Proceedings of the 24th International Symposium on Computer Architecture*, June 1997.

[24] A.N. Eden and T.N. Mudge. The YAGS branch prediction scheme. In *Proceedings of the 31st Annual ACM/IEEE International Symposium on Microarchitecture*, November 1998.

[25] M. Evers, P.-Y. Chang, and Y. N. Patt. Using hybrid branch predictors to improve branch prediction accuracy in the presence of context switches. In *Proceedings of the 23rd International Symposium on Computer Architecture*, May 1996.

[26] Keith Diefendorff. K7 challenges Intel. *Microprocessor Report*, 12(14), October 1998.

[27] Glenn Hinton, Dave Sager, Mike Upton, Darrell Boggs, Doug Carmean, Alan Kyker, and Patrice Roussel. The microarchitecture of the Pentium 4 processor. *Intel Technology Journal Q1*, 2001.

[28] E. Sprangle, R.S. Chappell, M. Alsup, and Y. N. Patt. The Agree predictor: A mechanism for reducing negative branch history interference. In *Proceedings of the 24th International Symposium on Computer Architecture*, June 1997

[29] Gregg Lesartre and Doug Hunt. PA-8500: The continuing evolution of the PA-8000 family. In *42nd IEEE International Computer Conference*, February 1997.

[30] Li C. Tsai. A 1GHz PA-RISC processor. In *Proceedings of the 2001 International Solid State Circuits Conference (ISSCC)*, February 2001.

[31] Richard E. Kessler. The Alpha 21264 microprocessor. *IEEE Micro*, 19(2):24. 36, March/April 1999.

[32] D. A. Jim´enez and N. Walsh. Dynamically weighted ensemble neural networks for classi_cation. In *Proceedings of the 1998 International Joint Conference on Neural Networks*, May 1998.

[33] Erik Jacobsen, Eric Rotenberg, and James E. Smith. Assigning con_dence to conditional branch predictions. In *Proceedings of the 29th Annual International Symposium on Microarchitecture*, December 1996.

[34] http://en.wikipedia.org/wiki/Denial-of-service_attack

[35] http://el.wikipedia.org/wiki/Ping_Of_Death

[36] http://el.wikipedia.org/wiki/UDP_flood_attack

[37] http://el.wikipedia.org/wiki/Email_bomb

[38] http://en.wikipedia.org/wiki/SYN_cookies

[39] http://www.cert.org/advisories/CA-1996-21.html

[40] http://httpd.apache.org/docs/2.3/misc/security_tips.html

[41] DNS Amplification Attacks-Preliminary release,Randal Vaughn and Gadi Evron March 17, 2006.

[42] DNS Amplification Attacks,
http://www.isotf.org/news/DNS-Amplification-Attacks.pdf