ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΕΠΙΚΟΙΝΩΝΙΩΝ, ΗΛΕΚΤΡΟΝΙΚΗΣ & ΣΥΣΤΗΜΑΤΩΝ
ΠΛΗΡΟΦΟΡΙΚΗΣ

Χρησιμοποίηση ενός Content-Addressable Δικτύου Peer-To-Peer σε ένα Σύστημα για την Αντιμετώπιση Επιθέσεων DDoS

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Κυριακή Α. Λεβαντή

Επιβλέπων : Βασίλης Μάγκλαρης
Καθηγητής Ε.Μ.Π.

Εργαστήριο Διαχείρισης και Βέλτιστου Σχεδιασμού Δικτύων – NETMODE

Αθήνα, Ιούλιος 2005

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΕΠΙΚΟΙΝΩΝΙΩΝ, ΗΛΕΚΤΡΟΝΙΚΗΣ & ΣΥΣΤΗΜΑΤΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ

Χρησιμοποίηση ενός Content-Addressable Δικτύου Peer-To-Peer σε ένα Σύστημα για την Αντιμετώπιση Επιθέσεων DDoS

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Κυριακή Α. Λεβαντή

Επιβλέπων : Βασίλης Μάγκλαρης
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την …….. .

.....	.....	.....
Β. Μάγκλαρης	Ε. Συκάς	Σ. Παπαβασιλείου
Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.	Επ. Καθηγητής Ε.Μ.Π.

Αθήνα, Ιούλιος 2005

.....
Κυριακή Α. Λεβαντή

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

ΠΕΡΙΛΗΨΗ

Οι Κατανεμημένες Επιθέσεις Άρνησης Υπηρεσίας (Distributed Denial of Service Attacks) εμφανίζουν ιδιαίτερη διάδοση στο Διαδίκτυο σήμερα. Πρόκειται για τη δημιουργία ροών κίνησης προς δίκτυα ή υπολογιστές στόχους από ελεγχόμενα συστήματα σε όλο το Διαδίκτυο. Για διάφορους τεχνικούς λόγους οι επιθέσεις αυτές δεν είναι εύκολο να αντιμετωπιστούν, ενώ συνήθως έχουν σοβαρότατο αντίκτυπο στα δίκτυα στόχους.

Σε αυτή τη διπλωματική εργασία οι κόμβοι ενός δικτύου peer-to-peer θα χρησιμοποιηθούν για την υλοποίηση ενός Υπερκείμενου Δικτύου (Overlay Network) το οποίο θα παρέχει υπηρεσίες για τη συνεργασία ανάμεσα σε διαφορετικές δικτυακές διαχειριστικές περιοχές (domains). Κατά την εκδήλωση μιας επίθεσης DDoS το υπερκείμενο δίκτυο προσφέρει υπηρεσίες αυτόματης οργάνωσης και επιλέγει κόμβους οι οποίοι θα χρησιμοποιηθούν ως σημεία συγκέντρωσης δεδομένων, διανομής πληροφοριών και συντονισμού της αντίδρασης.

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ

Επιθέσεις DDoS, Υπερκείμενο Δίκτυο, δίκτυα peer-to-peer, δρομολόγηση, GUID (Globally Unique Identifier), κόμβος-ρίζα, μηνύματα IDMEF, DHT (Distributed Hash Table)

ABSTRACT

Distributed Denial of Service (DDoS) attacks are becoming common in the Internet nowadays. They are comprised of high volume traffic flows that, originating from malicious systems dispersed all over the Internet, overwhelm networks or target hosts. For a number of technical reasons, thoroughly analysed in this thesis, these attacks cannot be easily mitigated at a single domain.

One possible solution proposed in this thesis is to use the nodes of a peer-to-peer network for the implementation of a Cooperative Overlay Network against DDoS attacks. This network will provide services which will enable the collaboration between different domains. At the event of a DDoS attack the overlay network organizes the domains involved, automatically selects the nodes which will be used as points of data collection and combines data from a number of sources to achieve path discovery. Overall, it is a framework for the distribution of information and coordination of reaction across many domains.

KEYWORDS

DDoS attacks, Overlay Network, peer-to-peer networks, routing, GUID (Globally Unique Identifier), root node, IDMEF messages, DHT (Distributed Hash Table)

ΕΥΧΑΡΙΣΤΙΕΣ

Θα ήθελα να ευχαριστήσω τον επιβλέποντα καθηγητή κ. Βασίλη Μάγκλαρη για τη δυνατότητα που μου έδωσε να ασχοληθώ με αυτή τη διπλωματική εργασία. Επίσης, θα ήθελα να ευχαριστήσω ιδιαίτερα τους υποψήφιους διδάκτορες Γεώργιο Κουτέπα και Αθανάσιο Μώραλη για την πολύτιμη βοήθεια τους χωρίς την οποία θα ήταν αδύνατο να ολοκληρωθεί αυτή η διπλωματική εργασία. Τέλος, θα ήθελα να ευχαριστήσω όλα τα μέλη του εργαστηρίου Διαχείρισης και Βέλτιστου Σχεδιασμού Δικτύων NETMODE για την ένθερμη υποστήριξή τους.

ΠΕΡΙΕΧΟΜΕΝΑ

Κεφάλαιο 1: Εισαγωγή

Σκοπός αυτής της διπλωματικής εργασίας είναι η υλοποίηση μίας συντονισμένης υποδομής με τη χρήση ενός Δικτύου Peer-to-peer, "Διευθυνσιοδότησης σύμφωνα με το Περιεχόμενο" (Content-Addressable Network - CAN) για την αντιμετώπιση Κατανεμημένων Επιθέσεων Άρνησης Υπηρεσίας (Distributed Denial of Service Attacks – DDoS). Συγκεκριμένα, υλοποιείται ένα Υπερκείμενο Δίκτυο (Overlay Network), το οποίο οργανώνει την επικοινωνία μεταξύ κόμβων, οι οποίοι ενδεχομένως να ανήκουν σε διαφορετικές διαχειριστικές περιοχές αλλά συμμετέχουν όλοι σε ένα δίκτυο peer-to-peer.

Οι επιθέσεις DDoS αποτελούν πλέον ένα αρκετά διαδεδομένο και με ιδιαίτερα δυσάρεστες συνέπειες φαινόμενο του Διαδικτύου. Μία επίθεση DDoS εξαπολύεται με τη δημιουργία αθροιζόμενων ροών κίνησης προς δίκτυα ή υπολογιστές στόχους με σκοπό τον κατακλυσμό των συνδέσεων που τα ενώνουν με το ευρύτερο δίκτυο και τη μετατροπή τους σε δίκτυα ανίκανα να παρέχουν τις υπηρεσίες που φυσιολογικά προσφέρουν. Αυτή η διογκωμένη ροή κίνησης δεν αποτελεί προφανώς φυσιολογική και νόμιμη κίνηση προς το θύμα, αλλά υποκινείται από διεσπαρμένα στο Διαδίκτυο συστήματα, τα οποία έχουν τεθεί υπό κακόβουλο έλεγχο ("trojaned") και δρουν σύμφωνα με τις οδηγίες του επιτιθέμενου. Η κατανεμημένη φύση της επίθεσης, καθώς και το τέχνασμα της απόκρυψης της πραγματικής διεύθυνσης του αποστολέα (Address Spoofing), καθιστούν τις περισσότερες απόπειρες αντιμετώπισης των επιθέσεων DDoS αναποτελεσματικές. Ωστόσο, οι επιπτώσεις τους στα δίκτυα στόχους είναι αρκετά σοβαρές και εκτεταμένες ώστε να επιβάλλεται η διαρκής έρευνα για εξεύρεση λύσεων στο πρόβλημα.

Ένα σύγχρονο δικτυακό φαινόμενο επίσης είναι τα δίκτυα peer-to-peer. Οι προγραμματιστές Διαδικτύου πρότειναν συνεχώς νέες και πρωτοποριακές κατανεμημένες εφαρμογές. Αυτές οι νέες εφαρμογές είχαν μια ποικιλία από απαιτήσεις σε διαθεσιμότητα, αντοχή και επίδοση. Μια τεχνική για την επίτευξη των παραπάνω χαρακτηριστικών θα μπορούσε να είναι η προσαρμογή σε προβλήματα συνδέσεων ή εξοπλισμού ή η κατανομή του φόρτου μέσω μεταφοράς και αντιγραφής δεδομένων και υπηρεσιών. Δυστυχώς, όμως, η δυνατότητα τοποθέτησης αντιγράφων ή η συχνότητα με την οποία μπορούν να μετακινούνται περιορίζεται σημαντικά από τη χρησιμοποιούμενη υποδομή. Ο παραδοσιακός τρόπος για την ανάπτυξη νέων εφαρμογών είναι η προσαρμογή τους - συχνά χωρίς επιτυχία - με κάποιον τρόπο στις υπάρχουσες υποδομές ή η τυποποίηση νέων πρωτοκόλλων Διαδικτύου, τα οποία έχουν να αντιμετωπίσουν σημαντική αδράνεια στην εφαρμογή. Έτσι, έγινε επιτακτική η ανάγκη για ένα ελαστικό αλλά τυποποιημένο "ενδιάμεσο στρώμα" πάνω στο οποίο θα μπορούν να αναπτυχθούν οι νέες εφαρμογές.

Αυτήν την ανάγκη ήρθαν να καλύψουν τα δίκτυα peer-to-peer. Τα δίκτυα peer-to-peer είναι υπερκείμενα δίκτυα με δική τους οργάνωση (διευθυνσιοδότηση, δρομολόγηση, μηχανισμούς διόρθωσης) και με βασικό χαρακτηριστικό το μεταβαλλόμενο αριθμό συμμετεχόντων κόμβων. Επιτρέπουν σε μία ομάδα χρηστών που συνεργάζονται να μοιράζονται το δίκτυο, και τους αποθηκευτικούς και τους υπολογιστικούς πόρους τους και να μην εξαρτώνται από υπηρεσίες κεντρικών εξυπηρετητών. Γνωρίζοντας μονάχα το μοναδικό αναγνωριστικό (κλειδί) της ζητούμενης πληροφορίας ο χρήστης μπορεί να την εντοπίσει χωρίς να χρειάζεται πληροφορίες για τη θέση της.

Η δρομολόγηση χαμηλού επιπέδου των δικτύων peer-to-peer ονομάζεται δρομολόγηση με βάση το κλειδί (Key Based Routing – KBR). Σε υψηλότερο επίπεδο

τα δίκτυα peer-to-peer παρέχουν λειτουργικότητα DHT (Distributed Hash Tables), DOLR (Decentralized Object Location and Routing) ή CAST (group anycast/multicast). Κάθε αντικείμενο στο δίκτυο αποκτά ένα μοναδικό αναγνωριστικό (κλειδί) μέσω μίας συνάρτησης κατακερματισμού. Με τον ίδιο τρόπο αποκτά κάθε κόμβος του δικτύου μοναδική διεύθυνση (ID). Σημειώνουμε ότι κλειδιά και IDs μοιράζονται τον ίδιο χώρο αναγνωριστικών. Έτσι, η αναζήτηση ενός αντικειμένου με κλειδί k ανάγεται σε δρομολόγηση στον κόμβο με ID k. Κατά αυτόν τον τρόπο έχουμε ευρεία διασπορά του περιεχομένου στο Διαδίκτυο, μεγάλη ανθεκτικότητα σε λάθη, υψηλό βαθμό εφεδρείας και κλιμάκωση.
Με αφορμή τα ιδιαίτερα χαρακτηριστικά της οργάνωσης των δικτύων peer-to-peer προτείνεται η οργάνωση ενός Συνεργατικού, Υπερκείμενου Δικτύου που να βασίζεται σε ένα δίκτυο peer-to-peer και να αξιοποιεί τα πλεονεκτήματα που αναφέρθηκαν παραπάνω. Αυτό το υπερκείμενο δίκτυο συντονίζει τη δράση που θα λαμβάνει ένα σύνολο κόμβων κατά την εκδήλωση μιας επίθεσης DDoS για την αντιμετώπισή της. Το ρόλο των αρχείων δεδομένων (τα οποία υπό κανονικές συνθήκες διανέμει ένα δίκτυο peer-to-peer) που διασπείρονται στο δίκτυο παίζουν οι αναφορές για ύποπτη κίνηση από τα Συστήματα Ανίχνευσης Επίθεσης (Intrusion Detection Systems – IDSs) των επιμέρους δικτύων. Αυτή η πληροφορία δεν μπορεί να καταστεί χρήσιμη, εάν προέρχεται από ένα και μόνο σημείο. Ωστόσο, η συγκέντρωσή της από διάφορα σημεία σε έναν κόμβο μπορεί να βοηθήσει στην εδραίωση της βεβαιότητας ανίχνευσης ενός περιστατικού και κατ' επέκταση να ενεργοποιήσει αποτελεσματικούς μηχανισμούς αντιμετώπισης της επίθεσης.
Συνοπτικά, αναφέρουμε ότι οι αναφορές από τα IDSs που αναφέρονται στην ίδια επίθεση DDoS αντιστοιχίζονται στο ίδιο κλειδί peer-to-peer και συγκεντρώνονται με δρομολόγηση στον κόμβο με αναγνωριστικό ίδιο (ή παρόμοιο) με το κλειδί. Αυτός ο κόμβος καλείται κόμβος-ρίζα του συγκεκριμένου κλειδιού. Από την άλλη, οποιοσδήποτε κόμβος θέλει να συλλέξει και να επεξεργαστεί πληροφορίες για μία επίθεση DDoS με γνωστό κλειδί δρομολογεί αίτηση στον κόμβο-ρίζα του κλειδιού. Η επεξεργασία των πληροφοριών καταλήγει στο δέντρο της επίθεσης, το οποίο αποτελεί μία καλή κατά προσέγγιση εικόνα της διαδρομής της επίθεσης.

Στο Κεφάλαιο 2 αναφέρονται βασικά στοιχεία για τις επιθέσεις DDoS (οργάνωση της επίθεσης, δυσκολίες και τακτικές αντιμετώπισης, προτεινόμενες αρχιτεκτονικές). Στο Κεφάλαιο 3 μελετώνται οι σύγχρονες αρχιτεκτονικές peer-to-peer, ενώ στο Κεφάλαιο 4 αναλύεται η αρχιτεκτονική της προτεινόμενης υποδομής. Στο Κεφάλαιο 5 αναλύεται η υλοποίηση της εφαρμογής και τέλος, στο Κεφάλαιο 6 εκφέρονται συμπεράσματα ως προς την αποτελεσματικότητά της με αναφορά στα ζητήματα της οργάνωσης, της συνέπειας και της ασφάλειας, καθώς επίσης προτείνονται μελλοντικές προεκτάσεις.

Κεφάλαιο 2: Επιθέσεις DDoS

Οι Κατανεμημένες επιθέσεις Άρνησης Υπηρεσίας (Distributed Denial of Service attacks – DDoS attacks) αποτελούν μία μεγάλη απειλή για το Διαδίκτυο. Μέχρι τώρα δεν υπάρχει απόλυτα αποτελεσματικός μηχανισμός άμυνας σε επιθέσεις DDoS μεγάλης κλίμακας. Αν και υπάρχουν διάφορα αμυντικά συστήματα για επιθέσεις DDoS, στην πλειοψηφία τους αποδίδουν μόνο για συγκεκριμένα σενάρια με αποτέλεσμα οι επιτιθέμενοι που ξεφεύγουν από αυτά τα σενάρια να αποφεύγουν τα παραπάνω αμυντικά συστήματα. Το κλειδί για την επιτυχημένη αντιμετώπιση των επιθέσεων DDoS είναι η ευρεία εφαρμογή των μέτρων αντιμετώπισης σε περισσότερα του ενός δίκτυα. Ωστόσο αυτή δεν μπορεί σε καμία περίπτωση να θεωρηθεί κάτι εύκολο, ή ακόμα και εφικτό, γιατί εξαρτάται από την αγορά. Μέχρι τώρα καμία τεχνολογία δεν έχει καταφέρει να μονοπωλήσει την αγορά, συνεπώς είναι μάλλον απίθανο κάτι τέτοιο να συμβεί με ένα και μόνο σύστημα αντιμετώπισης επιθέσεων DDoS. [1]

2.1 Βασικά στοιχεία μιας επίθεσης DDoS

Επίθεση DDoS έχουμε, όταν ένας αριθμός από ελεγχόμενα μηχανήματα (agents) παράγουν μεγάλο όγκο κίνησης προς το θύμα με στόχο να καταβάλλουν τους πόρους του. Είναι προφανές, λοιπόν, ότι στις επιθέσεις DDoS περιλαμβάνονται περισσότεροι από ένας επιτιθέμενοι και πιθανώς περισσότερα από ένα μηχανήματα-στόχος. Αυτές οι επιθέσεις καταλήγουν να είναι ιδιαίτερα «αποτελεσματικές» χρησιμοποιώντας την κατανεμημένη υποδομή του Διαδικτύου. Κατευθύνοντας πολλαπλές "επιθετικές" ροές κίνησης προς μία περιοχή δικτύου είναι δυνατό οι συνδέσεις, ακόμα και αν είναι υψηλής χωρητικότητας, να κατακλυστούν και να σταματήσουν να είναι διαθέσιμες για κανονική κίνηση.

Συγκεκριμένα, μια επίθεση DDoS εξαπολύεται με τα ακόλουθα βήματα:

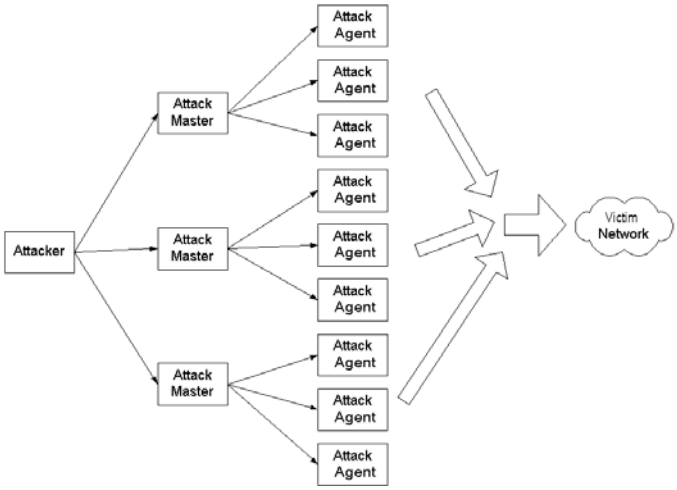
-βήμα $1^{ο}$: οι επιτιθέμενοι διεισδύουν σε ένα αριθμό μηχανημάτων και εγκαθιστούν κώδικα ελέγχου μικρού μεγέθους και αντίκτυπου στην κανονική λειτουργία ("footprint" κώδικας). Αυτά τα μηχανήματα είναι πλέον οι masters πρώτου επιπέδου.

-βήμα $2^{ο}$: οι masters πρώτου επιπέδου οργανώνουν την υποδομή της επίθεσης. Μετά από μια διαδικασία διερεύνησης (scanning) διαφόρων μηχανημάτων (από προσωπικούς υπολογιστές μέχρι μεγάλα συστήματα) για γνωστές αδυναμίες αναγνωρίζουν τα τρωτά μηχανήματα και εγκαθιστούν σε αυτά τον κώδικα που θα παράγει την επίθεση. Αυτά τα μηχανήματα ελέγχονται από τον επιτιθέμενο και ονομάζονται agents της επίθεσης, ενώ όσο τα κακόβουλα προγράμματά τους δεν είναι σε δράση ονομάζονται "zombies" ή "bots".

-βήμα $3^{ο}$: μετά από εντολή του επιτιθέμενου οι agents εξαπολύουν την επίθεση ενεργοποιώντας μεγάλη ροή πακέτων προς το δίκτυο-θύμα. Αυτές οι ροές πακέτων μπορεί μεμονωμένες και κοντά στην πηγή τους να είναι σχετικά μικρές, αλλά αθροιζόμενες κοντά στο δίκτυο-θύμα καταλήγουν να καταλάβουν σημαντικό τμήμα του διαθέσιμου δικτυακού εύρους (bandwidth). Στην Εικόνα 1 φαίνονται τα πολλαπλά επίπεδα της επίθεσης.

Τα εργαλεία που μπορούν να εκκινήσουν επιθέσεις DDoS καλούνται rootkits. Είναι έτοιμα πακέτα εργαλείων hacking που αυτοματοποιούν τις εργασίες της επίθεσης και εγκατάστασης κώδικα στους agents, αφού πρώτα ανιχνευθούν τα τρωτά μηχανήματα-που θα εξυπηρετήσουν τη διάδοση του λογισμικού επίθεσης. Οι κακόβουλες ροές πακέτων που ξεκινούν από τους agents για το δίκτυο-θύμα μπορεί να είναι οποιοδήποτε είδος κίνησης (TCP, UDP, ICMP κ.λπ.) καθώς και συνδυασμοί τους.

Χαρακτηριστικά παραδείγματα rootkits είναι τα “Trinoo”, “Stacheldraht” και “TFN2K”. [2]



Εικόνα 1: Πολλαπλά επίπεδα μιας επίθεσης DDoS. [2]

2.2 Δυσκολίες στην αντιμετώπιση

Για να εξαπολύσει κανείς μια τέτοιου είδους επίθεση δε χρειάζεται ιδιαίτερες γνώσεις ή ικανότητες και εάν μάλιστα το θύμα δεν είναι εφοδιασμένο με αποδοτικούς αμυντικούς μηχανισμούς, θα «υποφέρει» όσο χρονικό διάστημα διαρκεί η επίθεση. Επιπλέον, οι επιτιθέμενοι δεν φοβούνται το ενδεχόμενο εντοπισμού, μιας και είναι πολύ δύσκολο να ανιχνευτεί η επίθεση με βήματα προς τα πίσω - η διαδικασία αναφέρεται ως "Back-tracing" - που θα οδηγήσει στον εντοπισμό των agents, ενώ είναι ακόμα πιο δύσκολο να εντοπιστούν αυτοί που μόλυναν τους agents.

Ορισμένα χαρακτηριστικά των επιθέσεων DDoS που εμποδίζουν την επιτυχημένη αντιμετώπισή τους είναι:

- Φαινομενικά «νόμιμα» πακέτα: τα πακέτα των επιθέσεων μπορεί να είναι ίδια με τα «νόμιμα» πακέτα, αφού ο επιτιθέμενος στοχεύει στο να προκαλέσει πρόβλημα με τον όγκο των πακέτων και όχι το περιεχόμενό τους. Έτσι, το σύστημα ανίχνευσης επιθέσεων δεν μπορεί να διαχωρίσει τα πακέτα σε «κακόβουλα» και «νόμιμα» βασιζόμενο σε μεμονωμένα πακέτα, αλλά θα πρέπει να κρατά στατιστικά, για να συσχετίσει πακέτα και να ανιχνεύσει ανωμαλίες.
- Αυξημένος όγκος κίνησης: τα αθροιζόμενα μονοπάτια επίθεσης σχηματίζουν τόσο μεγάλη ροή που μπορεί να καταρρεύσει μέχρι και το σύστημα ανίχνευσης επιθέσεων, ενώ οι δικτυακοί τόποι που βρίσκονται πάνω στη διαδρομή της επίθεσης θα συνεχίσουν να «υποφέρουν».
- IP spoofing: οι επιτιθέμενοι συνήθως βάζουν ψεύτικη διεύθυνση στο πεδίο IP source των πακέτων της επίθεσης. Κατ’ αυτόν τον τρόπο «κρύβονται» οι agents και δυσχεραίνεται ο εντοπισμός τους.
- Έλεγχος και ευελιξία: ο έλεγχος της επίθεσης από τον επιτιθέμενο έχει πολλαπλά επίπεδα, ξεκινώντας από έναν μικρό αριθμό από masters που αυξάνονται σταδιακά με την πρόσθεση των agents και μπορούν τελικώς να εξαπλωθούν σε ευρεία κλίμακα. Συγχρόνως, ο επιτιθέμενος μπορεί να εναλλάσσει τους στόχους της επίθεσης και τις πηγές της κίνησης, να ενεργοποιεί και απενεργοποιεί τμήματα της κίνησης και να μεταβάλλει τα χαρακτηριστικά των ροών σύμφωνα με τις αντιδράσεις των δικτύων που αντιλαμβάνονται με κάποιον τρόπο την επίθεση. Τέλος, στα πολλαπλά

επίπεδα της επίθεσης μπορούν να προστεθούν επιπλέον επίπεδα κατά την παράδοσή της. Επί παραδείγματι, αντί να στέλνονται τα πακέτα απευθείας από τα υπονομευόμενα μηχανήματα, τα τελευταία στέλνουν νόμιμες αιτήσεις σε εξυπηρετητές Διαδικτύου με διεύθυνση επιστροφής το θύμα.

Τέλος, πρέπει να σημειώσουμε ότι και τα συμβατικά IDS (Intrusion Detection Systems) αποτυγχάνουν στην αντιμετώπιση των επιθέσεων DDoS. Τα IDSs μπορούν να αναγνωρίσουν έγκαιρα μια επίθεση, αλλά δεν έχουν τη δυνατότητα διάσχισης ενός δικτυακού τόπου (domain) και κλιμακούμενης σε μέγεθος αντίδρασης. Επιπλέον, δεν υπάρχει η αναγκαία υποδομή για την ανταλλαγή αιτήσεων και απαντήσεων με άλλους δικτυακούς τόπους. Ωστόσο, ακόμα και αν κάτι τέτοιο ήταν διαθέσιμο, μία τέτοια συναλλαγή θα επέφερε πολλά ζητήματα ασφαλείας και το IDS θα έπρεπε επίσης να γνωρίζει την τοπολογία και τα interfaces των απομακρυσμένων μηχανημάτων στα άλλους δικτυακούς τόπους, για να εκδώσει τις κατάλληλες εντολές. [2]

2.3 Τακτικές αντιμετώπισης

Σκοπός ενός αμυντικού συστήματος DDoS είναι να μειώσει την αρνητική επίδραση της επίθεσης στο θύμα και να συνεχιστεί η καλή εξυπηρέτηση των «νόμιμων» πελατών του θύματος και της «φυσιολογικής» κίνησης κατά τη διάρκεια της επίθεσης. Για να επιτευχθεί αυτό μπορεί να προσεγγίσει κανείς το θέμα της άμυνας σε επιθέσεις DDoS από τρεις πλευρές: 1) παρεμπόδιση της επίθεσης, 2) εφοδιασμός του θύματος με εργαλεία-μεθόδους, ώστε να επιβιώσει από την επίθεση, 3) ανίχνευση και αντιμετώπιση της επίθεσης.

- Για την παρεμπόδιση της επίθεσης είναι αναγκαίο να δοθεί προσοχή στα τρωτά σημεία, τα οποία εκμεταλλεύονται οι επιτιθέμενοι, για να εξαπολύσουν την επίθεση. Παρόλο που μια τέτοια προσέγγιση έχει ελπίδες να συνεισφέρει σημαντικά στην ασφάλεια Διαδικτύου, θα χρειαστεί πολύ χρόνος να αποκτήσει τέτοια πληρότητα και εξάπλωση, ώστε να εμποδίσει τις επιθέσεις DDoS.
- Για την επιβίωση του θύματος κατά την επίθεση πρέπει να αυξηθούν οι πόροι του θύματος αρκετά, ώστε να μπορεί να εξυπηρετεί και «νόμιμη» και κακόβουλη κίνηση. Κάτι τέτοιο θα μπορούσε γενικά να φέρει αποτέλεσμα σε υπηρεσίες, όπως στατικές σελίδες Web, αλλά δε θα μπορούσε να αποτελέσει γενικευμένη λύση, καθώς ο επιτιθέμενος μπορεί να αυξάνει συνεχώς τους κατανεμημένους πόρους του (αύξηση των agents), ενώ το θύμα δε θα έχει ποτέ τα ίδια περιθώρια.
- Για την ανίχνευση και αντιμετώπιση της επίθεσης απαιτείται η υλοποίηση ενός μηχανισμού με δυνατότητα ανίχνευσης της πλειοψηφίας των απειλητικών επιθέσεων, μείωσης της ροής σε λογικά επίπεδα ανεξάρτητα από τον όγκο ή την κατανομή της κίνησης, και διαφοροποίησης της φυσιολογικής από την κακόβουλη κίνηση, ώστε να εξασφαλίζεται η καλή εξυπηρέτηση της νόμιμης κίνησης. Η παράπλευρη ζημιά κατά την αντιμετώπιση της επίθεσης πρέπει να είναι μικρότερη από τη ζημιά που θα προκαλούσε η παντελής έλλειψη μηχανισμού αντιμετώπισης. [1]

Συγκεκριμένα για την αντιμετώπιση μιας επίθεσης DDoS, είναι αναγκαία η συνεργασία μεταξύ των δικτυακών τόπων (sites), αφού η φύση της επίθεσης εμποδίζει την αποτελεσματική αντιμετώπισή της από ένα μεμονωμένο δικτυακό τόπο. Όπως έχει ήδη αναφερθεί, το IP spoofing, η εξάπλωση της επίθεσης σε πολλά επιτιθέμενα δίκτυα και η αδυναμία ενός δικτύου να διαμορφώσει τον όγκο της εισερχόμενης κίνησης (traffic shaping) καθιστούν μη αποτελεσματικές τις προσπάθειες ενός και μόνο δικτυακού τόπου να αντιμετωπίσει την επίθεση.

Η συνεργασία ανάμεσα στους δικτυακούς τόπους μπορεί να λάβει χώρα με τις ακόλουθες ενέργειες:
(α) καθορισμός των χαρακτηριστικών της επίθεσης (χρησιμοποιούμενο πρωτόκολλο, πόρτες κ.α.) σε κάθε σημείο κατά μήκος της διαδρομής της επίθεσης, έτσι ώστε να μπορούν να τεθούν σε λειτουργία τα κατάλληλα φίλτρα αντιμετώπισης,
(β) διάδοση αυτών των χαρακτηριστικών σε όλα τα δίκτυα που βρίσκονται πάνω στο μονοπάτι της επίθεσης και επικοινωνία ανάμεσα στο θύμα και στα δίκτυα προέλευσης της κίνησης για έλεγχο της αποτελεσματικότητας των φίλτρων (προσαρμογή τους στα εναλλασσόμενα μοτίβα επιθέσεων).
Ωστόσο μια τέτοια συνεργατική προσέγγιση έχει να αντιμετωπίσει ένα σημαντικό πρόβλημα. Η αποτελεσματικότητά της εξαρτάται έντονα από τη διαθεσιμότητα και προθυμία των υπεύθυνων των δικτύων προέλευσης της κίνησης, καθώς και από τις πολιτικές λειτουργίας τους. Επιπλέον, δεν προβλέπει μέτρα για την ανακούφιση από τη συμφόρηση, η οποία εξακολουθεί να πλήττει το δίκτυο-θύμα και τα δίκτυα που βρίσκονται πλησιέστερα σε αυτό στο μονοπάτι της επίθεσης. Συνεπώς, χρειάζονται επιπρόσθετα μέτρα, για να μειωθεί το κόστος σε εύρος δικτύου. [2]

2.4 Προτεινόμενες λύσεις

Κατά την προσπάθεια αντιμετώπισης των επιθέσεων DDoS έχουν αναπτυχθεί τρεις βασικές μεθοδολογίες – προσεγγίσεις: *ανίχνευση*, *traceback* (ανακάλυψη πηγής και διαδρομής επίθεσης) και *αντίδραση*. Η διαδικασία της ανίχνευσης μπορεί να γίνεται στο δίκτυο-θύμα με παρακολούθηση της συμφόρησης και ανάλυση της κίνησης ή των απορριφθέντων πακέτων, ή στο μονοπάτι της επίθεσης με συνδυασμό διαφόρων αναφορών από τα δίκτυα που πλήττονται. Το traceback μπορεί να διεξαχθεί με συνεργασία της υποδομής Διαδικτύου ή με απλή κίνηση προς τα πίσω χωρίς προηγούμενη παρατήρηση. Γενικά, μπορεί να γίνει μια εξαιρετικά δύσκολη διαδικασία, εάν η επίθεση χρησιμοποιεί IP spoofing. Τέλος, η αντίδραση μπορεί να είναι προληπτική ή άμεση κατά την επίθεση. Οι άμεσες προσπάθειες αντίδρασης μπορούν να αφορούν ένα και μόνο δικτυακό τόπο ή να αποτελούν κομμάτι μιας συνολικής κατανεμημένης υποδομής με κοινό στόχο την αντιμετώπιση της επίθεσης DDoS. [3]
Στη συνέχεια ακολουθούν λίγα λόγια για τέσσερις προτεινόμενες λύσεις, οι οποίες αντιμετωπίζουν μερικώς τις επιθέσεις DDoS, αλλά παρουσιάζουν μειονεκτήματα:

- Υπερκείμενο Δίκτυο CenterTrack [4]: αυτό το υπερκείμενο δίκτυο έχει ως σκοπό την ανίχνευση των ροών της επίθεσης. Εγκαθίσταται τούνελ από το δρομολογητή συνόρου του δικτυακού τόπου ως ένα συγκεκριμένο σημείο στο δίκτυο και όταν το θύμα δέχεται επίθεση, η κίνηση δρομολογείται σε αυτό μέσω του υπερκείμενου δικτύου. Έτσι, είναι εφικτό να γίνει ανά βήμα traceback στους δρομολογητές συνόρου, από τους οποίους περνάει η κίνηση της επίθεσης. Αυτή η τεχνική από τη μία χρειάζεται διαγνωστικά στοιχεία μόνο στους δρομολογητές συνόρου, από την άλλη η εγκατάσταση των τούνελ απαιτεί την κατανάλωση πόρων και αυξάνει την πολυπλοκότητα διαχείρισης. Προφανώς, χρησιμοποιείται η μεθοδολογία του traceback.
- Pushback πρωτόκολλο [5]: το ειδικό αυτό πρωτόκολλο αποτελεί μια προσπάθεια προτυποποίησης της επικοινωνίας ανάμεσα σε δρομολογητές που συνεργάζονται. Σκοπός είναι ο έλεγχος των υψηλού εύρους δικτύου ροών κίνησης μιας επίθεσης DDoS. Οι δρομολογητές επικοινωνούν μεταξύ τους δεδομένα για τυχόν κακόβουλη κίνηση και κινητοποιούνται για να τη σταματήσουν χρησιμοποιώντας εξειδικευμένα φίλτρα. Κατά αυτόν τον τρόπο ακολουθούν τις επιμέρους ροές προς την πηγή τους και ελέγχουν σε διάφορα

σημεία την κατανομή του εύρους δικτύου. Μειονέκτημα αποτελεί και πάλι η κατανάλωση πόρων για την επεξεργασία των επικοινωνούντων μηνυμάτων σε όλους τους δρομολογητές προς την πηγή. Η μεθοδολογία-προσέγγιση που χρησιμοποιείται είναι το traceback και η αντίδραση.

- Panoptis [6]: ο Panoptis είναι ένα εργαλείο ανοιχτού λογισμικού για τη μέτρηση αλλαγών στα μοτίβα ροής κίνησης στους δρομολογητές συνόρου. Συγκεκριμένα, συσχετίζει πρόσφατα πακέτα με δεδομένα από μετρήσεις κίνησης, για να συμπεράνει ποιο interface ενός δρομολογητή συνόρου εμπλέκεται σε επίθεση. Κατόπιν, με την εφαρμογή κατάλληλων φίλτρων στα interfaces που πλήττονται μπορεί να αντιμετωπιστεί η επίθεση. Αυτή η λύση παρουσιάζει καλή κλιμάκωση, αλλά είναι απαραίτητη μία μεγάλη ποσότητα ιστορικών δεδομένων προκειμένου να γίνει η συσχέτιση και μάλιστα πρέπει όλη η επεξεργασία να γίνεται τοπικά, για να μην καταναλωθεί και άλλο εύρος δικτύου για τη μεταφορά των δεδομένων. Ο Panoptis ανήκει στη μεθοδολογία της ανίχνευσης και της αντίδρασης.
- Υποδομή Cooperative Intrusion Traceback and Response (CITRA): η υποδομή CITRA βασίζεται στην οργάνωση διαφόρων κοινοτήτων σε γειτονιές. Κάθε κοινότητα εκτελεί ανίχνευση εισβολής χαμηλού επιπέδου και περιορίζεται σε συγκεκριμένα όρια. Με την ανίχνευση μίας επίθεσης από μία κοινότητα, η τελευταία διανέμει αναφορές επίθεσης στους γείτονές της, οι οποίοι μπορούν να ανακαλύψουν το μονοπάτι της επίθεσης και να αντιδράσουν. Η CITRA χρησιμοποιεί εντολές αντίδρασης ανεξάρτητες μηχανήματος, και η διαδικασία των αναφορών και ο συντονισμός γίνεται κεντρικά. Τέλος, χρησιμοποιείται το πρωτόκολλο IDIP (Intruder Detection and Isolation Protocol) και η μεθοδολογία ανήκει στην κατηγορία της ανίχνευσης και της αντίδρασης. [3]

Κεφάλαιο 3: Peer-to-peer Υπερκείμενα Δίκτυα

3.1 Υπερκείμενα δίκτυα

Το Διαδίκτυο χρησιμοποιείται σήμερα από εκατομμύρια ανθρώπους. Το να αλλάξει κάτι στους βασικούς μηχανισμούς του (στα βασικά πρωτόκολλά του), ακόμα και αν αυτό είναι η πρόσθεση επιπλέον χαρακτηριστικών σε αυτούς, δεν είναι απλή υπόθεση. Σε αυτήν την περίπτωση θα χρειαστεί να αλλάξει η υλοποίηση των πρωτοκόλλων σε κάθε υπάρχων κόμβο. Για αυτό το λόγο έχουν γίνει τόσο δημοφιλή τα **Υπερκείμενα Δίκτυα.** Τα υπερκείμενα δίκτυα δημιουργούν ένα εικονικό δίκτυο πάνω από την ήδη υπάρχουσα τοπολογία και χρησιμοποιούν τα δικά τους σχήματα δρομολόγησης και καμιά φορά και το δικό τους χώρο διευθύνσεων. Ανήκουν στο στρώμα εφαρμογής του μοντέλου OSI, και τα πιο γνωστά παραδείγματα υπερκείμενων δικτύων είναι τα Ιδιωτικά Εικονικά Δίκτυα (Virtual Private Networks – VPN), καθώς και οι περιπτώσεις των συστημάτων ανταλλαγής αρχείων peer-to-peer.

3.2 Δίκτυα peer-to-peer

3.2.1 Γενικά

Ένα peer-to-peer δίκτυο υπολογιστών δεν έχει σταθερό αριθμών πελατών και εξυπηρετητών, αλλά αποτελείται από ένα μεταβλητό αριθμό αλληλένδετων κόμβων που λειτουργούν και σαν πελάτης και σαν εξυπηρετητής για κάθε άλλο κόμβο του δικτύου. Έτσι, κάθε κόμβος μπορεί να στείλει ή να λάβει αίτηση εξυπηρέτησης.

Τα δίκτυα peer-to-peer αποτελούν μια κατηγορία εφαρμογών που εκμεταλλεύεται τους αποθηκευτικούς πόρους, τους υπολογιστικούς κύκλους, το περιεχόμενο και την ανθρώπινη παρουσία που είναι διαθέσιμη στα διάφορα σημεία που συνδέονται μέσω Διαδικτύου. Ωστόσο, επειδή η πρόσβαση σε αυτούς τους κατανεμημένους πόρους συνεπάγεται λειτουργία σε ένα περιβάλλον με υπολογιστές, που δεν είναι σταθερά συνδεδεμένοι στο Διαδίκτυο, και επιπλέον, οι IP διευθύνσεις τους είναι μη προβλέψιμες, πρέπει οι κόμβοι ενός δικτύου peer-to-peer να λειτουργούν έξω από το σύστημα DNS και να μην εξαρτώνται από κεντρικούς εξυπηρετητές. Τα peer-to-peer υπερκείμενα δίκτυα δεύτερης γενιάς δε χρησιμοποιούν συγκεντρωμένες υπηρεσίες και δε χρειάζονται υποστήριξη από ISPs. Τέλος, χρησιμοποιούν μεγάλο όγκο πόρων χωρίς να απαιτείται κεντρικός σχεδιασμός ή μεγάλες επενδύσεις σε υλικό, εύρος ζώνης.

Στα πρωτοεμφανιζόμενα συστήματα peer-to-peer συγκαταλέγονται προγράμματα επικοινωνίας με την υπηρεσία instant messaging όπως το ICQ [7] και δίκτυα ανταλλαγής αρχείων όπως το Gnutella [8] και ο Napster [9] (ωστόσο και το ICQ και ο Napster χρησιμοποιούν σύστημα client-server για την αναζήτηση). Άλλο παράδειγμα είναι το SETI@Home project, ένα επιστημονικό πείραμα, το οποίο χρησιμοποιεί υπολογιστές συνδεδεμένους στο Διαδίκτυο για την αναζήτηση εξωγήινων επικοινωνιών. Οι υπολογισμοί γίνονται με τεχνικές peer-to-peer, ενώ η κατανομή των δεδομένων ακολουθεί το πρότυπο client-server. Τέλος, θα μπορούσε κανείς να πει ότι το πιο γνωστό σύστημα peer-to-peer είναι η IP δρομολόγηση.

Τα συστήματα peer-to-peer αποτελούν δημοφιλές πεδίο έρευνας στο χώρο της πληροφορικής και των επικοινωνιών, και χρησιμοποιούνται κυρίως για την υλοποίηση εντελώς αποκεντρωμένων δικτύων από αλληλένδετους κόμβους. Τέτοια δίκτυα ενδείκνυνται για εφαρμογές ομαδικής επικοινωνίας, αποθήκευσης αρχείων, κατανομής περιεχομένου και ανταλλαγής πόρων.

Τα πρωτοεμφανιζόμενα συστήματα peer-to-peer ήταν συστήματα ανταλλαγής αρχείων. Ο **Napster** εμφανίστηκε στα μέσα του 1999 και μέχρι το τέλος του 2000 το

λογισμικό του αποκτήθηκε από περισσότερους από 50 εκατομμύρια χρήστες. Σε αυτά τα συστήματα τα αρχεία αποθηκεύονται σε μηχανήματα τελικών χρηστών (peers) αντί σε έναν κεντρικό εξυπηρετητή και σε αντίθεση με το μοντέλο client-server, τα αρχεία ανταλλάσσονται μεταξύ των *peers*. Τα συστήματα ανταλλαγής αρχείων peer-to-peer μπορούν να οδηγήσουν σε νέα μοντέλα κατανομής περιεχομένου για εφαρμογές όπως η διανομή λογισμικού, η ανταλλαγή αρχείων και η στατική διανομή περιεχομένου web. Άλλα συστήματα ανταλλαγής αρχείων είναι τα Scour [10], FreeNet [11], Ohaha [12], Jungle Monkey [13], MojoNation [14].
Ωστόσο, τα παραπάνω συστήματα peer-to-peer δεν είναι κλιμακούμενα ως προς το μέγεθος του δικτύου. Για παράδειγμα, στον Napster ένας κεντρικός εξυπηρετητής αποθηκεύει τον κατάλογο όλων των αρχείων της κοινότητας χρηστών του Napster. Για την ανάκτηση ενός αρχείου ο χρήστης ρωτά τον κεντρικό εξυπηρετητή χρησιμοποιώντας το ευρύτερα γνωστό όνομα του αρχείου και παίρνει ως απάντηση την IP διεύθυνση του μηχανήματος που αποθηκεύει το παραπάνω αρχείο. Έτσι, εάν και ο Napster χρησιμοποιεί μοντέλο επικοινωνίας peer-to-peer για τη μεταφορά του αρχείου, η διαδικασία εντοπισμού του αρχείου εξακολουθεί να είναι κεντρική με αποτέλεσμα να κοστίζει στο σύστημα η αυξομείωση του κεντρικού κατάλογου και να είναι τρωτό, αφού η αποτυχία εξαρτάται από ένα και μόνο σημείο.
Από την άλλη, το **Gnutella** προχωράει ένα βήμα και αποκεντρώνει τη διαδικασία εντοπισμού του αρχείου. Οι χρήστες ενός δικτύου Gnutella οργανώνονται σε ένα πλέγμα επιπέδου εφαρμογής, στο οποίο κατευθύνονται μαζικά οι αιτήσεις για ένα αρχείο με συγκεκριμένο περιεχόμενο. Η πλημμύρα για κάθε αίτηση προφανώς δεν είναι κλιμακούμενη ανάλογα με το μέγεθος του δικτύου και επειδή πρέπει να σταματήσει σε κάποιο σημείο, είναι πιθανό η αναζήτηση να αποτύχει και να μη βρεθεί το ζητούμενο περιεχόμενο στο σύστημα. Το **MojoNation** χρησιμοποιεί ένα online οικονομικό μοντέλο, για να ενθαρρύνει το μοίρασμα των πόρων. Το **Freenet** είναι ένα δίκτυο ανταλλαγής αρχείων σχεδιασμένο για να αντιστέκεται στη λογοκρισία. Ούτε το Gnutella ούτε το Freenet εγγυώνται τον εντοπισμό των αρχείων – ακόμα και σε ένα κανονικό δίκτυο που λειτουργεί χωρίς μεγάλα προβλήματα..

Η δεύτερη γενιά των συστημάτων peer-to-peer είναι τα δομημένα peer-to-peer υπερκείμενα δίκτυα όπως είναι το Tapestry [36,52], το Chord [30], το Pastry [43,51], το CAN [27], το Kademlia [29], το Bamboo [43,49]. Αυτά τα υπερκείμενα δίκτυα υλοποιούν ένα βασικό interface δρομολόγησης με βάση το κλειδί (key-based routing – KBR), το οποίο υποστηρίζει μία ντετερμινιστική δρομολόγηση μηνυμάτων στον κόμβο - υπεύθυνο για το κλειδί προορισμού. Επίσης, μπορούν να υποστηρίξουν interfaces υψηλότερου επιπέδου όπως έναν κατανεμημένο πίνακα κατακερματισμού (Distributed Hash Table – DHT) ή ένα στρώμα κατανεμημένου εντοπισμού και δρομολόγησης αντικειμένων (Distributed Object Location and Routing – DOLR) (βλ. παράγραφο 3.2.3). Αυτά τα συστήματα παρουσιάζουν καλή κλιμάκωση και εγγυώνται ότι οι ερωτήσεις βρίσκουν τα ζητούμενα αντικείμενα υπό φυσιολογικές συνθήκες.
Ένα χαρακτηριστικό που διαφοροποιεί αυτά τα συστήματα μεταξύ τους είναι το εάν λαμβάνουν υπόψη δικτυακές αποστάσεις, όταν φτιάχνουν το υπερκείμενο στρώμα δρομολόγησης. Έτσι, στο CAN και στο Chord ένα υπερκείμενο βήμα μπορεί να είναι όσο η διάμετρος του δικτύου. Και τα δύο πρωτόκολλα δρομολογούν κατά μήκος της μικρότερης διαθέσιμης διαδρομής και χρησιμοποιούν σε χρόνο εκτέλεσης διάφορες τεχνικές για τη βελτιστοποίηση της συμπεριφοράς τους. Από την άλλη, το Tapestry και το Pastry φτιάχνουν τοπικά βελτιστοποιημένους πίνακες δρομολόγησης κατά την

αρχικοποίηση και τους διατηρούν προκειμένου να μειώσουν την έκταση που θα πάρει η δρομολόγηση.
Μια καινούρια γενιά εφαρμογών έχει προταθεί πάνω σε αυτά τα συστήματα peer-to-peer επικυρώνοντάς τα ως υποδομές νέων και πιο εξελιγμένων εφαρμογών. Πολλά συστήματα παρέχουν multicast επιπέδου εφαρμογής: CAN-MC [15] (CAN), Scribe [16] (Pastry) και Bayeux [17] (Tapestry). Επιπλέον, έχουν προταθεί διάφορα αποκεντρωμένα συστήματα αρχείων: CFS [18] (Chord), Mnemosyne [19] (Chord, Tapestry), OceanStore [20,53] (Tapestry) και PAST [21] (Pastry). Τα δομημένα peer-to-peer υπερκείμενα δίκτυα υποστηρίζουν επίσης νέες εφαρμογές, όπως τα δίκτυα αντοχής σε επιθέσεις [22], στρώματα δικτυακής ανακατεύθυνσης [23] και η αναζήτηση ομοιότητας [24].

3.2.2 Τα βασικότερα συστήματα peer-to-peer

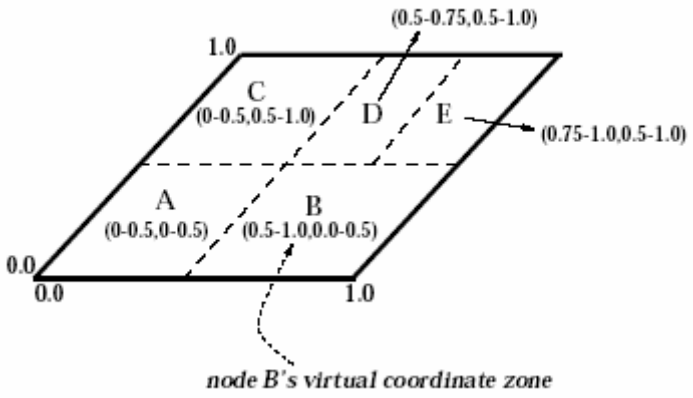
3.2.2.1 CAN

Το μεγαλύτερο πλεονέκτημα για ένα σύστημα peer-to-peer είναι η ευκολία κλιμάκωσής του. Η διαδικασία peer-to-peer μεταφοράς ενός αρχείου είναι από τη φύση της κλιμακούμενη, αλλά η πραγματική δυσκολία έγκειται στην εύρεση του κόμβου που διαθέτει το αρχείο. **Δίκτυα Διευθυνσιοδότησης σύμφωνα με το Περιεχόμενο** (Content-Addressable Networks - CANs) είναι τα δίκτυα που χρησιμοποιούν ένα σχήμα δεικτοδότησης για την αντιστοίχηση ονομάτων αρχείων σε θέση στο σύστημα. Τα CANs, εκτός από τα συστήματα peer-to-peer, βρίσκουν εφαρμογή και σε συστήματα οργάνωσης αποθήκευσης μεγάλης κλίμακας, όπως το OceanStore, το Farsite [25] και το Publius [26]. Και αυτά τα συστήματα απαιτούν αποδοτική εισαγωγή και ανάκτηση περιεχομένου σε μία μεγάλη κατανεμημένη υποδομή αποθήκευσης και έτσι ένας εύκολα κλιμακούμενος μηχανισμός δεικτοδότησης είναι απαραίτητος. Μια άλλη πιθανή εφαρμογή των CANs θα μπορούσε να είναι η κατασκευή υπηρεσιών ανάλυσης ονόματος (name resolution) ευρείας κλίμακας, οι οποίες, σε αντίθεση με το DNS, θα αποσυνδέουν το σχήμα της ονοματοδοσίας από τη διαδικασία της ανάλυσης ονόματος και έτσι θα επιτρέπουν τη χρήση αυθαίρετων και ανεξάρτητων από τη θέση σχημάτων ονοματοδοσίας. Γενικά, σκοπός των CANs (και ειδικότερα της αφαιρετικής δομής ενός πίνακα κατακερματισμού) είναι να χρησιμοποιηθούν σαν σχεδιαστικό εργαλείο από τους σχεδιαστές Διαδικτύου για την ανάπτυξη νέων εφαρμογών και μοντέλων επικοινωνίας.
Ένα CAN αποτελεί μια κατανεμημένη υποδομή, η οποία παρέχει τη λειτουργικότητα ενός πίνακα κατακερματισμού (**hash table**) σε επίπεδο Διαδικτύου. Ένας πίνακας κατακερματισμού είναι μια δομή δεδομένων που αντιστοιχίζει αποδοτικά «κλειδιά» σε «τιμές» και αποτελεί την κύρια δομική μονάδα στις περισσότερες υλοποιήσεις συστημάτων λογισμικού.

Βασικά στοιχεία ενός CAN δικτύου
Τα CANs μοιάζουν με πίνακες κατακερματισμού. Οι βασικές λειτουργίες τους είναι η εισαγωγή, η αναζήτηση και η διαγραφή ζευγών (κλειδί, τιμή). Το CAN αποτελείται από πολλούς μεμονωμένους κόμβους. Κάθε κόμβος CAN αποθηκεύει ένα τμήμα ολόκληρου του πίνακα κατακερματισμού, το οποίο καλείται *ζώνη* (zone). Επιπλέον, κάθε κόμβος κρατά πληροφορίες για ένα μικρό αριθμό «γειτονικών» ζωνών. Αιτήσεις εισαγωγής, αναζήτησης ή διαγραφής για ένα συγκεκριμένο κλειδί δρομολογούνται από ενδιάμεσους CAN κόμβους προς τον κόμβο, του οποίου η ζώνη περιέχει το

κλειδί. Αυτός ο CAN σχεδιασμός είναι εντελώς **κατανεμημένος** (δεν απαιτείται καμιάς μορφής κεντρικός έλεγχος, συνεργασία ή ρύθμιση), **κλιμακούμενος** (οι κόμβοι διατηρούν μονάχα μια μικρή ποσότητα της πληροφορίας ελέγχου, η οποία είναι ανεξάρτητη από τον αριθμό των κόμβων στο σύστημα) και **ανεκτικός σε σφάλματα** (οι κόμβοι μπορούν να συνεχίζουν τη δρομολόγηση παρά τις αποτυχίες). Επίσης, δε χρησιμοποιεί καμιά μορφή ιεραρχικής δομής ονοματοδοσίας (όπως το DNS ή η IP δρομολόγηση), για να πετύχει καλή κλιμάκωση και μπορεί να υλοποιηθεί εξολοκλήρου σε επίπεδο εφαρμογής.

Ένας συγκεκριμένος CAN σχεδιασμός
Το κυριότερο στοιχείο αυτής της σχεδίασης είναι ένας εικονικός d-διάστατος καρτεσιανός χώρος συντεταγμένων. Σε κάθε χρονική στιγμή ολόκληρος ο χώρος κατανέμεται δυναμικά ανάμεσα σε όλους τους κόμβους του συστήματος, έτσι ώστε κάθε κόμβος να έχει τη δική του ζώνη μέσα στο χώρο (Εικόνα 2).

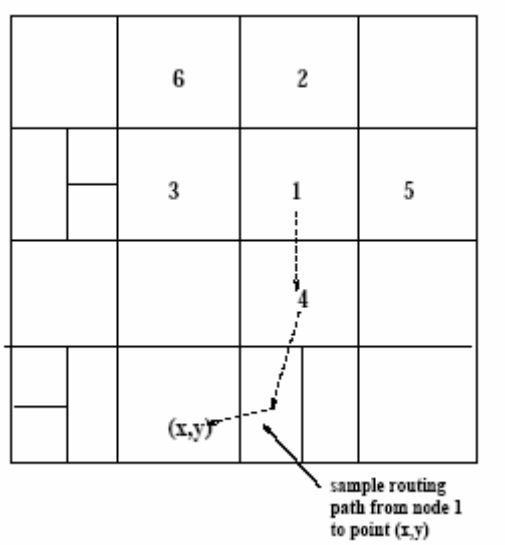


Εικόνα 2: διδιάστατος [0,1]*[0,1] χώρος συντεταγμένων κατανεμημένος ανάμεσα σε 5 κόμβους. [27]

Αυτός ο εικονικός χώρος συντεταγμένων χρησιμοποιείται, για να αποθηκεύσει ζεύγη (κλειδί, τιμή). Για την αποθήκευση του ζεύγους (K1,V1), το κλειδί K1 αντιστοιχίζεται συγκεκριμένα σε ένα σημείο P του χώρου με τη χρήση μιας ομοιόμορφης συνάρτησης κατακερματισμού. Το ζεύγος (K1,V1) θα αποθηκευτεί στον κόμβο, ο οποίος κατέχει τη ζώνη, στην οποία «πέφτει» το σημείο P. Για την ανάκτηση του αντικειμένου με κλειδί K1, ο εκάστοτε κόμβος μπορεί να εφαρμόσει την ίδια συνάρτηση κατακερματισμού, για να αντιστοιχίσει το κλειδί K1 στο σημείο P και μετά να ανακτήσει την αντίστοιχη τιμή από το P. Εάν το σημείο P δεν ανήκει στον κόμβο που κάνει την αναζήτηση ή στους γείτονές του, η αίτηση πρέπει να δρομολογηθεί μέσω της CAN υποδομής μέχρι να φτάσει στον κόμβο με τη ζώνη που περιέχει το P. Συνεπώς, η αποδοτική δρομολόγηση είναι ένα κρίσιμο κομμάτι του CAN.
Οι κόμβοι στο CAN αυτο-διοργανώνονται σε ένα υπερκείμενο δίκτυο, το οποίο αναπαριστά τον εικονικό χώρο συντεταγμένων. Κάθε κόμβος μαθαίνει και διατηρεί τις IP διευθύνσεις των κόμβων που κατέχουν τις ζώνες, οι οποίες γειτονεύουν στο χώρο των συντεταγμένων με τη δική του ζώνη. Αυτό το σύνολο των άμεσων γειτόνων στο χώρο ουσιαστικά είναι ένας πίνακας δρομολόγησης που επιτρέπει τη δρομολόγηση ανάμεσα σε δύο αυθαίρετα σημεία στο χώρο.

Δρομολόγηση στο CAN

Όπως αναφέρθηκε και παραπάνω, ένας κόμβος CAN διατηρεί έναν πίνακα δρομολόγησης με τις IP διευθύνσεις και εικονικές ζώνες όλων των άμεσων γειτόνων του στο χώρο των συντεταγμένων. Σε έναν d-διάστατο χώρο δύο κόμβοι είναι γειτονικοί, εάν τα διαστήματα των συντεταγμένων τους δεν επικαλύπτονται για d - 1 διαστάσεις και συνορεύουν κατά μήκος μίας μόνο διάστασης. Έτσι, στην Εικόνα 3 ο κόμβος 5 είναι γείτονας του κόμβου 1, ενώ ο κόμβος 6 δεν είναι γείτονας του κόμβου 1.



Εικόνα 3: Παράδειγμα διδιάστατου χώρου. Το σύνολο γειτόνων του κόμβου 1 είναι {2, 3, 4, 5}. [27]

Αυτή η πληροφορία είναι αρκετή για τη δρομολόγηση ανάμεσα σε δύο αυθαίρετα σημεία στο χώρο των συντεταγμένων, εφόσον ένα μήνυμα CAN περιλαμβάνει τις συντεταγμένες του προορισμού του. Χρησιμοποιώντας τις συντεταγμένες των γειτόνων του, ένας κόμβος δρομολογεί ένα μήνυμα προς τον προορισμό του προωθώντας το στο γείτονα με τις πλησιέστερες συντεταγμένες στις συντεταγμένες του προορισμού. Στην παραπάνω εικόνα (Εικόνα 3) φαίνεται ένα απλό μονοπάτι δρομολόγησης.

Για ένα χώρο **d** διαστάσεων χωρισμένο σε **n** ίσες ζώνες, το μέσο μήκος μονοπατιού δρομολόγησης είναι **(d / 4)(n $^{1/d}$)** βήματα και κάθε κόμβος διατηρεί 2d γείτονες. Συνεπώς, σε έναν τέτοιο χώρο μπορούμε να αυξήσουμε τον αριθμό των κόμβων (άρα και των ζωνών) χωρίς να αυξάνουμε την πληροφορία ανά κόμβο, ενώ το μέσο μήκος μονοπατιού θα αυξάνει με πολυπλοκότητα O(n $^{1/d}$).

Επιπλέον, υπάρχουν προφανέστατα πολλά διαφορετικά μονοπάτια ανάμεσα σε δύο σημεία στο χώρο, πράγμα που σημαίνει ότι, εάν ο γείτονας ενός κόμβου είναι «νεκρός», ο κόμβος μπορεί αυτόματα να δρομολογήσει την αίτησή του κατά μήκος της επόμενης καλύτερης διαδρομής. Εάν, ωστόσο, ένας κόμβος χάσει όλους τους γείτονές του προς μία συγκεκριμένη κατεύθυνση και οι διορθωτικοί μηχανισμοί που περιγράφονται αργότερα δεν καταφέρουν να γεμίσουν το κενό που δημιουργείται στο χώρο των συντεταγμένων, τότε η «άπληστη» προώθηση μπορεί να αποτύχει προσωρινά. Σε αυτήν την περίπτωση, ο κόμβος μπορεί να χρησιμοποιήσει μια επεκτεινόμενη αναζήτηση δακτυλίου (με ελεγχόμενη πλημμύρα στο unicast υπερκείμενο πλέγμα CAN), για να εντοπίσει έναν κόμβο που είναι πλησιέστερα στον προορισμό από τον εαυτό του. Έπειτα, το μήνυμα προωθείται σε αυτόν τον κόμβο και επαναφέρεται η «άπληστη» προώθηση.

Δόμηση του CAN

Όλοι οι κόμβοι σε ένα CAN κατέχουν τη δική τους ζώνη. Για να επιτραπεί σε ένα CAN να αυξήσει τον αριθμό των κόμβων του, κάθε καινούριος κόμβος που προσχωρεί στο σύστημα θα πρέπει να αποκτήσει τη δικιά του ζώνη στο χώρο των συντεταγμένων. Αυτό μπορεί να γίνει, εφόσον ένας ήδη υπάρχων κόμβος χωρίσει στη μέση τη δική του ζώνη και παραχωρήσει τη μισή στον καινούριο κόμβο.

Συγκεκριμένα, η διαδικασία ολοκληρώνεται σε τρία βήματα:

- Ο νέος κόμβος βρίσκει έναν ήδη υπάρχοντα κόμβο στο CAN :

Ένας νέος κόμβος CAN πρώτα ανακαλύπτει την IP διεύθυνση ενός οποιουδήποτε κόμβου στο σύστημα. Η λειτουργία του CAN δεν εξαρτάται από τον τρόπο που γίνεται αυτό. Υποθέτουμε ότι ένα CAN έχει ένα DNS όνομα δικτύου, το οποίο αναλύεται στην IP διεύθυνση ενός ή περισσοτέρων κόμβων αρχικοποίησης (bootstrap nodes) του CAN. Ένας κόμβος αρχικοποίησης διατηρεί μια μερική λίστα των κόμβων CAN που πιστεύει ότι ανήκουν στο σύστημα. Για να προσχωρήσει στο CAN, ο νέος κόμβος αναζητά το όνομα δικτύου του CAN στο DNS, για να αποκτήσει την IP διεύθυνση ενός κόμβου αρχικοποίησης. Στη συνέχεια, ο κόμβος αρχικοποίησης δίνει τις IP διευθύνσεις μερικών τυχαία επιλεγόμενων κόμβων του συστήματος.

- Με χρήση των μηχανισμών δρομολόγησης CAN βρίσκει έναν κόμβο, του οποίου η ζώνη θα μοιραστεί :

Ο νέος κόμβος επιλέγει τυχαία ένα σημείο P στο χώρο και στέλνει μία JOIN αίτηση με προορισμό το σημείο P. Αυτό το μήνυμα στέλνεται στο CAN μέσω οποιουδήποτε υπάρχοντος κόμβου CAN και προωθείται σύμφωνα με τους μηχανισμούς δρομολόγησης CAN στον κόμβο που είναι υπεύθυνος για το P. Αυτός ο κόμβος χωρίζει τη ζώνη του στα δύο και αναθέτει το ένα κομμάτι στον καινούριο κόμβο. Αυτός ο χωρισμός γίνεται υποθέτοντας μια συγκεκριμένη ταξινόμηση των διαστάσεων για την επιλογή της διάστασης που θα χωριστεί, έτσι ώστε οι ζώνες να μπορούν να ενωθούν, όταν κόμβοι αποχωρούν από το σύστημα. Έτσι, σε ένα διδιάστατο χώρο μια ζώνη πρώτα θα χωριζόταν κατά την X διάσταση, μετά κατά την Y κ.ο.κ. Τα ζεύγη (κλειδί, τιμή) της μισής ζώνης που παραχωρείται μεταφέρονται στον καινούριο κόμβο.

- Οι γείτονες της ζώνης που μοιράστηκε ενημερώνονται, ώστε να συμπεριληφθεί στη δρομολόγηση ο νέος κόμβος :

Αφού έχει αποκτήσει τη ζώνη του, ο νέος κόμβος μαθαίνει τις IP διευθύνσεις των γειτόνων του από τον κόμβο που του παραχώρησε τη μισή ζώνη του. Προφανώς, και αυτός ο κόμβος ανήκει στο σύνολο των γειτόνων του νέου κόμβου και πρέπει και αυτός να ανανεώσει τη λίστα των γειτόνων του και να διαγράψει αυτούς που δεν ανήκουν πλέον σε αυτή. Τέλος, οι γείτονες του νέου και του παλιού κόμβου πρέπει να ενημερωθούν για την αλλαγή στο χώρο των συντεταγμένων. Κάθε κόμβος στο σύστημα στέλνει ένα άμεσο μήνυμα ενημέρωσης στους γείτονές του, ακολουθούμενο από περιοδικές ανανεώσεις, με περιεχόμενο τη ζώνη που του έχει ανατεθεί τη δεδομένη χρονική στιγμή. Αυτές οι "soft-state style" ενημερώσεις εξασφαλίζουν ότι όλοι οι γείτονες θα μάθουν γρήγορα τις τυχόν αλλαγές και θα ενημερώσουν ανάλογα το σύνολο των γειτόνων τους.

Η πρόσθεση καινούριων κόμβων επηρεάζει μόνο ένα μικρό αριθμό κόμβων σε μια μικρή περιοχή του χώρου των συντεταγμένων. Ο αριθμός των γειτόνων ενός κόμβου εξαρτάται μόνο από τις διαστάσεις του χώρου και είναι ανεξάρτητος του συνολικού αριθμού των κόμβων στο σύστημα. Έτσι, η εισαγωγή ενός κόμβου επηρεάζει μόνο O(*αριθμός διαστάσεων*) υπάρχοντες κόμβους, πράγμα πολύ σημαντικό για ένα CAN με πάρα πολλούς κόμβους.

Αποχώρηση κόμβου, ανάκαμψη και διατήρηση του CAN

Όταν κόμβοι αποχωρούν από ένα CAN, είναι απαραίτητο να διασφαλίζεται ότι οι ζώνες που είχαν καταλάβει θα περάσουν υπό την επίβλεψη των κόμβων που παραμένουν. Αυτό γίνεται με την παραχώρηση της ζώνης και της βάσης δεδομένων (κλειδί, τιμή) που συνδέεται με αυτήν σε κάποιον από τους γείτονες του αποχωρούντος κόμβου. Εάν η ζώνη ενός γείτονα μπορεί να ενωθεί με τη ζώνη του αποχωρούντος κόμβου παράγοντας μία έγκυρη ζώνη, τότε γίνεται η ένωση. Εάν αυτό δεν είναι δυνατό, τότε η ζώνη παραχωρείται στο γείτονα με τη μικρότερη ζώνη, ο οποίος προσωρινά θα μεταχειρίζεται δύο ζώνες.

Το CAN πρέπει επίσης να παραμένει σταθερό παρά τις αποτυχίες κόμβων ή δικτύου (όταν ένας ή περισσότεροι κόμβοι χάνουν συνδεσιμότητα). Αυτό ρυθμίζεται με έναν άμεσο αλγόριθμο απόκτησης ελέγχου, ο οποίος διασφαλίζει ότι ένας γείτονας του νεκρού κόμβου θα αναλάβει τη ζώνη του. Ωστόσο, σε αυτήν την περίπτωση τα ζεύγη (κλειδί, τιμή) του νεκρού κόμβου χάνονται μέχρι να ανανεωθεί η κατάσταση από τους κόμβους που κρατούν τα δεδομένα.

Υπό κανονικές συνθήκες ένας κόμβος στέλνει περιοδικά μηνύματα ανανέωσης σε κάθε έναν από τους γείτονές του περικλείοντας τις συντεταγμένες της ζώνης του και τη λίστα των γειτόνων του με τις συντεταγμένες των ζωνών τους. Μία παρατεταμένη απουσία ενός τέτοιου μηνύματος από ένα γείτονα ερμηνεύεται ως αποτυχία κόμβου.

Μόλις ένας κόμβος αποφασίσει ότι ένας γείτονας του είναι νεκρός, ξεκινά το μηχανισμό κατάληψης μαζί με ένα χρονιστή απόκτησης ελέγχου. Κάθε γείτονας του νεκρού κόμβου θα δράσει ανάλογα, με τον χρονιστή ενεργοποιημένο σε αναλογία με το μέγεθος της ζώνης του. Όταν ο χρονιστής αποπνέει, ο κόμβος στέλνει ένα μήνυμα TAKEOVER μαζί με το μέγεθος της ζώνης του σε όλους τους γείτονες του νεκρού κόμβου.

Λαμβάνοντας ένα μήνυμα TAKEOVER, κάθε κόμβος ακυρώνει τον χρονιστή του, εάν το μέγεθος της ζώνης στο μήνυμα είναι μικρότερο από το μέγεθος της δικής του ζώνης. Διαφορετικά, απαντά με το δικό του μήνυμα TAKEOVER. Με αυτόν τον τρόπο, επιλέγεται ο γείτονας-κόμβος που είναι ζωντανός και έχει τη μικρότερη ζώνη σε μέγεθος.

Στην περίπτωση που «πέσουν» ταυτόχρονα πολλοί διπλανοί κόμβοι, είναι δυνατό να ανιχνευτεί η αποτυχία, εάν λιγότεροι από τους μισούς γείτονες του νεκρού κόμβου είναι ακόμα προσιτοί. Εάν ένας κόμβος αναλάβει μια άλλη ζώνη υπό αυτές τις συνθήκες, μπορεί το CAN να γίνει ασυνεπές. Σε αυτήν την περίπτωση, πριν πυροδοτηθεί ο μηχανισμός διόρθωσης, ο κόμβος ξεκινά μια επεκτεινόμενη αναζήτηση δακτυλίου για κόμβους που βρίσκονται πέρα από την περιοχή αποτυχίας και έτσι αποκτά ξανά την απαραίτητη πληροφορία για τους γειτονικούς κόμβους και μπορεί να εκκινήσει πάλι την κατάληψη με ασφάλεια.

Τέλος, και η κανονική διαδικασία αποχώρησης και ο άμεσος αλγόριθμος κατάληψης μπορούν να καταλήξουν σε έναν κόμβο που κρατά περισσότερες από μία ζώνες. Για να αποφευχθεί περαιτέρω τεμαχισμός του χώρου, ένας παρασκηνιακός αλγόριθμος συναρμολόγησης ζωνών τρέχει, για να διασφαλίσει ότι το CAN θα τείνει πάλι σε μία ζώνη ανά κόμβο.

Σχεδιαστικές βελτιώσεις

Ο βασικός αλγόριθμος CAN, όπως περιγράφτηκε παραπάνω, χαρακτηρίζεται από μια ισορροπία ανάμεσα σε πληροφορία ανά κόμβο (**O(d)** για d-διάστατο χώρο) και σε σύντομα μήκη διαδρομής (**O(d $n^{1/d}$)** βήματα για d διαστάσεις και n κόμβους). Αυτά τα βήματα αναφέρονται σε επίπεδο εφαρμογής και όχι σε επίπεδο IP, πράγμα που

σημαίνει ότι η καθυστέρηση σε κάθε βήμα μπορεί να είναι μεγάλη. Δύο κόμβοι, οι οποίοι είναι διπλανοί στο CAN, μπορεί να είναι πολλά χιλιόμετρα και πολλά βήματα IP μακριά. Η μέση συνολική καθυστέρηση μιας αναζήτησης ισούται με το μέσο αριθμό των βημάτων CAN επί τη μέση καθυστέρηση ενός βήματος CAN. Επιθυμητή είναι μια καθυστέρηση αναζήτησης, η οποία να είναι συγκρίσιμη με τις υποκείμενες IP καθυστερήσεις ανάμεσα στον αιτούντα και στον κόμβο CAN που κρατά το κλειδί.
Έτσι, έχουν αναπτυχθεί κάποιες σχεδιαστικές τεχνικές (τις οποίες δε θα αναφέρουμε εδώ, γιατί ξεφεύγουν από το θέμα μας), οι οποίες έχουν σαν κύριο σκοπό τη μείωση της καθυστέρησης της δρομολόγησης CAN. Πολλές από αυτές τις τεχνικές βελτιώνουν παράλληλα την ευρωστία του CAN όσον αφορά τη δρομολόγηση και τη διαθεσιμότητα των δεδομένων. Η προσπάθεια για μείωση της καθυστέρησης ανάγεται στην προσπάθεια μείωσης του μήκους της διαδρομής ή της καθυστέρησης ανά βήμα στο CAN. Τέλος, το CAN έχει εφοδιαστεί και με απλούς μηχανισμούς εξισορρόπησης φορτίου.
Ωστόσο, πρέπει να επισημάνουμε ότι οι επιπλέον σχεδιαστικές βελτιώσεις γίνονται εις βάρος της πληροφορίας ανά κόμβο (παρόλο που αυτή παραμένει ανεξάρτητη του αριθμού των κόμβων στο σύστημα) και ότι αυξάνουν την πολυπλοκότητα του συστήματος. Έτσι, το κατά πόσο αυτές οι τεχνικές αξίζει να εφαρμοστούν περιλαμβάνει μία επιλογή ανάμεσα σε βελτιωμένη επίδοση δρομολόγησης και σταθερότητα συστήματος από τη μία πλευρά, και αυξημένη πληροφορία ανά κόμβο και πολυπλοκότητα συστήματος από την άλλη πλευρά.

Μέτρα απόδοσης και σχεδιαστικοί παράμετροι του CAN
Για την αξιολόγηση της απόδοσης του συστήματος χρησιμοποιούνται κυρίως τα παρακάτω μεγέθη :

- Μήκος διαδρομής: ο αριθμός των βημάτων (επιπέδου εφαρμογής) που απαιτούνται για τη δρομολόγηση ανάμεσα σε δύο σημεία στο χώρο των συντεταγμένων.
- "Neighbor-state": ο αριθμός των CAN κόμβων, για τους οποίους πρέπει κάθε μεμονωμένος κόμβος να κρατά πληροφορία.
- Καθυστέρηση: η καθυστέρηση της συνολικής διαδρομής ανάμεσα σε δύο σημεία στο χώρο των συντεταγμένων και η ανά βήμα καθυστέρηση (η δεύτερη προκύπτει από τη διαίρεση της πρώτης με το μήκος του μονοπατιού).
- Μέγεθος: το μέγεθος της ζώνης, η οποία έχει ανατεθεί σε έναν κόμβο. Είναι ενδεικτικό του φόρτου αίτησης και αποθήκευσης που αναμένεται να πρέπει να μεταχειριστεί ο κόμβος.
- Ανεκτικότητα σε λάθη δρομολόγησης: η διαθεσιμότητα πολλαπλών διαδρομών ανάμεσα σε δύο σημεία στο CAN.
- Διαθεσιμότητα του πίνακα κατακερματισμού: η επαρκής αντιγραφή ενός ζεύγους (κλειδί, τιμή) προκειμένου να είναι ανεκτή η απώλεια ενός αντίγραφου.

Οι κυριότερες σχεδιαστικές παράμετροι που επηρεάζουν την απόδοση του συστήματος είναι :

- ο αριθμός των διαστάσεων του εικονικού χώρου συντεταγμένων : d
- ο αριθμός των συναρτήσεων κατακερματισμού, δηλ. ο αριθμός των σημείων ανά πραγματικότητα στα οποία αποθηκεύεται ένα ζεύγος (κλειδί, τιμή) : k

Τελευταίες παρατηρήσεις
Η παραπάνω σχεδίαση ενός CAN υλοποιεί κλιμακούμενη δρομολόγηση και δεικτοδότηση. Σύμφωνα με τα αποτελέσματα προσομοίωσης, σε ένα CAN με περισσότερους από 260.000 κόμβους μπορούμε να δρομολογήσουμε με καθυστέρηση λιγότερη από τη διπλάσια IP καθυστέρηση.
Ανάμεσα στα προβλήματα που αντιμετωπίζει το CAN είναι η αντοχή του σε επιθέσεις DoS. Αυτό το πρόβλημα είναι ιδιαίτερα δύσκολο στην αντιμετώπιση, γιατί ένας κακόβουλος κόμβος μπορεί να συμπεριφέρεται και σαν πελάτης και σαν εξυπηρετητής. Έτσι, η έρευνα για ασφαλή CAN ακόμα συνεχίζεται. Επιπλέον, γίνεται προσπάθεια, ώστε να επεκταθούν οι CAN αλγόριθμοι και να χειρίζονται και μεταβαλλόμενο περιεχόμενο. [27]

3.2.2.2 Kademlia

Το Kademlia είναι ένα peer-to-peer σύστημα αποθήκευσης και αναζήτησης ζευγών (κλειδί, τιμή). Βασικό χαρακτηριστικό του είναι ότι ελαχιστοποιεί τον αριθμό των μηνυμάτων ρύθμισης που πρέπει να στείλουν οι κόμβοι, για να μάθουν όσα χρειάζεται να ξέρουν για τους άλλους κόμβους. Οι πληροφορίες ρύθμισης εξαπλώνονται αυτόματα, καθώς διεξάγονται οι αναζητήσεις κλειδιών. Επιπλέον, οι κόμβοι έχουν την απαραίτητη γνώση και ευελιξία, ώστε να δρομολογούν ερωτήσεις μέσω διαδρομών χαμηλής καθυστέρησης. Το Kademlia χρησιμοποιεί παράλληλα ασύγχρονα ερωτήματα, για να αποφύγει καθυστερήσεις λήξης χρόνου από νεκρούς κόμβους και ο αλγόριθμος με τον οποίο ενημερώνονται οι κόμβοι για την κατάσταση των άλλων κόμβων στο σύστημα είναι ανθεκτικός απέναντι στη βασική επίθεση DoS.
Τα κλειδιά στο Kademlia είναι λέξεις των 160 bits (π.χ. ο SHA-1 κατακερματισμός μεγαλύτερων δεδομένων). Κάθε ένας από τους συμμετέχοντες στο σύστημα υπολογιστές έχει ένα node ID σε έναν χώρο κλειδιών των 160 bits. Τα ζεύγη (κλειδί, τιμή) αποθηκεύονται σε κόμβους με IDs «κοντά» στο κλειδί. Τέλος, ο αλγόριθμος δρομολόγησης βασίζεται στα node IDs και επιτρέπει τον εντοπισμό των εξυπηρετητών κοντά σε ένα κλειδί-προορισμό από κάθε κόμβο.
Ένα από τα καινοτόμα στοιχεία του Kademlia είναι η XOR τεχνική μέτρησης της απόστασης ανάμεσα σε δύο σημεία στο χώρο των κλειδιών. Η XOR τεχνική είναι συμμετρική και επιτρέπει στους κόμβους του Kademlia να λαμβάνουν ερωτήσεις αναζήτησης από ακριβώς την ίδια κατανομή κόμβων που περιέχεται στους πίνακες δρομολόγησης τους. Ένας κόμβος Kademlia μπορεί να στείλει ερώτηση σε οποιονδήποτε κόμβο εντός ενός διαστήματος και έτσι μπορεί να επιλέξει διαδρομές ανάλογα με την καθυστέρηση ή ακόμα και να στείλει παράλληλα ασύγχρονες ερωτήσεις.
Για τον εντοπισμό κόμβων κοντά σε ένα συγκεκριμένο ID το Kademlia χρησιμοποιεί έναν απλό αλγόριθμο δρομολόγησης από την αρχή μέχρι το τέλος (άλλα συστήματα χρησιμοποιούν έναν αλγόριθμο για να πλησιάσουν το ID-στόχο και έναν άλλο για τα τελευταία λίγα βήματα).

Περιγραφή συστήματος Kademlia
Κάθε κόμβος Kademlia έχει ένα node ID των 160 bits. Κάθε μήνυμα που μεταδίδεται από έναν κόμβο περιλαμβάνει και το node ID του, επιτρέποντας κατ' αυτόν τον τρόπο στον παραλήπτη να καταγράψει την ύπαρξη του αποστολέα.
Τα κλειδιά είναι και αυτά αναγνωριστικά των 160 bits. Για τη δημοσίευση και την εύρεση ζευγών (κλειδί, τιμή) το Kademlia χρησιμοποιεί μια δική του έννοια για την απόσταση ανάμεσα σε δύο αναγνωριστικά. Δεδομένου δύο αναγνωριστικών των 160

bits, x και y, το Kademlia ορίζει ως απόσταση ανάμεσα στα δύο το ανά bit XOR τους εκφρασμένο σαν ακέραιο, **d (x, y) = x XOR y**. Σημειώνουμε ότι το XOR είναι ένα έγκυρο και μη-Ευκλείδειο μέτρο, για το οποίο ισχύει :

- d (x, x) = 0 και d (x, y) > 0, εάν x ≠ y
- ∀ x, y : d (x, y) = d (y, x)
- d (x, z) = d (x, y) XOR d (y, z)
  ∀ a ≥0, b≥0 : a + b ≥ a XOR b =>
- d (x, y) + d (y, z) ≥d (x, z)

Το XOR είναι μονοκατευθυντικό. Για κάθε δεδομένο σημείο x και απόσταση Δ > 0, υπάρχει ένα και μόνο σημείο y, για το οποίο ισχύει d (x, y) = Δ. Η μονοκατευθυντικότητα εξασφαλίζει ότι όλες οι αναζητήσεις για το ίδιο κλειδί θα συγκλίνουν στην ίδια διαδρομή ανεξάρτητα του κόμβου που την ξεκίνησε. Έτσι, η προσωρινή αποθήκευση ζευγών (κλειδί, τιμή) κατά μήκος της διαδρομής αναζήτησης μπορεί να ανακουφίσει τα δημοφιλή ζεύγη. Τέλος, η XOR τοπολογία είναι συμμετρική (∀ x, y: d (x, y) = d (y, x)).

Πληροφορία ανά κόμβο

Οι κόμβοι Kademlia αποθηκεύουν πληροφορίες επικοινωνίας με του άλλους κόμβους, για να δρομολογούν μηνύματα-ερωτήσεις. Για κάθε 0 ≤ i < 160, κάθε κόμβος διατηρεί μια λίστα από τριπλέτες του τύπου (IP διεύθυνση, UDP πόρτα, node ID) για κόμβους με απόσταση 2^i και 2^{i+1} από τον εαυτό του. Αυτές οι λίστες καλούνται *k-κάδοι* (k-buckets). Κάθε k-κάδος κρατείται ταξινομημένος σύμφωνα με το χρόνο τελευταίας συνάντησης. Στην κεφαλή της λίστας βρίσκεται ο κόμβος που «ακούστηκε» νωρίτερα και στην ουρά ο κόμβος που «ακούστηκε» πιο πρόσφατα. Για μικρές τιμές του i, οι k-κάδοι θα είναι γενικά άδειοι, αφού δε θα υπάρχουν οι αντίστοιχοι κόμβοι. Για μεγάλες τιμές του i, οι λίστες μπορούν να φτάσουν μέχρι και μέγεθος k, όπου k είναι η παράμετρος αντιγραφής του συστήματος. Το k επιλέγεται έτσι ώστε οποιοιδήποτε k κόμβοι να μην βρίσκονται μεταξύ τους σε περισσότερο από μία ώρα απόσταση.

Όταν ένας κόμβος Kademlia λαμβάνει ένα μήνυμα (αίτηση ή απάντηση) από έναν άλλο κόμβο, ενημερώνει τον αντίστοιχο k-κάδο με το node ID του αποστολέα. Εάν τα στοιχεία του κόμβου αποστολέα υπάρχουν ήδη στον k-κάδο του παραλήπτη, τότε ο παραλήπτης μεταφέρει την τριπλέτα στην ουρά της λίστας. Εάν δεν υπάρχουν και ο k-κάδος έχει λιγότερες από k στοιχεία, τότε ο παραλήπτης απλά εισάγει τον αποστολέα στην ουρά της λίστας. Αν πάλι ο k-κάδος είναι γεμάτος, τότε ο παραλήπτης κάνει ping στον τελευταίο κόμβο από τον οποίο «άκουσε», για να αποφασίσει τι θα κάνει. Εάν αυτός ο κόμβος δεν απαντήσει, διαγράφεται από τον κάδο και εισάγεται στην ουρά της λίστας ο νέος αποστολέας. Διαφορετικά, μεταφέρεται στην ουρά της λίστας και ο νέος αποστολέας απορρίπτεται από τον κάδο.

Οι k-κάδοι υλοποιούν αποδοτικά μια τεχνική απόρριψης του κόμβου που συναντήθηκε λιγότερο πρόσφατα χωρίς να απομακρύνουν ζωντανούς κόμβους. Αυτή η προτίμηση για παλιές επαφές πηγάζει από μία ανάλυση, κατά την οποία, όσο περισσότερο είναι ζωντανός ένας κόμβος, τόσο περισσότερο πιθανό είναι να παραμείνει ζωντανός για άλλη μια ώρα [28]. Έτσι, κρατώντας τις παλιότερες επαφές στους κάδους οι k-κάδοι μεγιστοποιούν την πιθανότητα να περιέχουν κόμβους που θα παραμένουν συνδεδεμένοι.

Ένα δεύτερο πλεονέκτημα των k-κάδων είναι ότι παρέχουν ασφάλεια απέναντι σε συγκεκριμένες επιθέσεις DoS. Κανείς δεν μπορεί να εξαφανίσει την πληροφορία δρομολόγησης που φυλάει ένας κόμβος πλημμυρίζοντας το σύστημα με νέους

κόμβους, γιατί οι νέοι κακόβουλοι κόμβοι θα εισαχθούν στους k-κάδους μόνο όταν αποχωρήσουν από το σύστημα οι παλιοί κόμβοι.

Πρωτόκολλο Kademlia

Το πρωτόκολλο Kademlia περιλαμβάνει τέσσερις απομακρυσμένες κλήσεις διαδικασίας (Remote Procedure Calls – RPCs): PING, STORE, FIND_NODE και FIND_VALUE. Η *PING* RPC επιχειρεί να μάθει εάν ένας κόμβος είναι συνδεδεμένος. Η *STORE* δίνει την εντολή σε έναν κόμβο να αποθηκεύσει ένα ζεύγος (κλειδί, τιμή), για να ανακτηθεί αργότερα.

Η *FIND_NODE* παίρνει ένα ID των 160 bits σαν όρισμα. Ο παραλήπτης της RPC επιστρέφει μια τριπλέτα (διεύθυνση IP, UDP πόρτα, node ID) για τους k κόμβους που γνωρίζει ότι είναι κοντύτερα στον στόχο-ID. Αυτή η τριπλέτα μπορεί να προέρχεται από έναν και μόνο κάδο, ή μπορεί να προέρχεται από πολλούς κάδους, εάν ο κοντινότερος k-κάδος δεν είναι γεμάτος. Σε κάθε περίπτωση, ο παραλήπτης RPC πρέπει να στείλει k τεμάχια, εκτός εάν υπάρχουν λιγότεροι από k κόμβοι σε όλους συνολικά τους κάδους του, οπότε και στέλνει πληροφορία μόνο για τους κόμβους που γνωρίζει.

Η *FIND_VALUE* συμπεριφέρεται σαν την FIND_NODE –επιστρέφει και αυτή τριπλέτες- με μία εξαίρεση. Εάν ο παραλήπτης RPC έχει λάβει κλήση STORE για το κλειδί, απλά επιστρέφει την αποθηκευμένη τιμή.

Σε όλα τα RPCs, ο παραλήπτης πρέπει να συμπεριλάβει ένα τυχαίο RPC ID των 160 bits, το οποίο παρέχει κάποια αντίσταση σε απόπειρες πλαστογραφίας. Τα PINGs μπορούν επίσης να γίνουν piggy-backed σε απαντήσεις RPC, έτσι ώστε ο παραλήπτης RPC να λάβει μια επιπλέον επιβεβαίωση της διεύθυνσης δικτύου του αποστολέα.

Η πιο σημαντική εργασία για έναν κόμβο Kademlia είναι ο εντοπισμός των k κοντινότερων κόμβων σε ένα δεδομένο ID. Αυτή η εργασία καλείται *αναζήτηση κόμβου*. Το Kademlia εφαρμόζει έναν αναδρομικό αλγόριθμο για τις αναζητήσεις κόμβων. Ο κόμβος – εκκινητής της αναζήτησης – (τον ονομάζουμε *αρχικό*) διαλέγει α κόμβους από τον κοντινότερο μη-άδειο k-κάδο του. Εάν ο κάδος έχει λιγότερα από *α* στοιχεία, παίρνει απλά του κοντινότερους *α* κόμβους που γνωρίζει. Ο αρχικός στέλνει στη συνέχεια παράλληλα και ασύγχρονα FIND_NODE RPCs στους *α* κόμβους που έχει επιλέξει. Το *α* είναι μια παράμετρος ταυτοχρονισμού του συστήματος.

Αλγόριθμος αναζήτησης , προσωρινής αποθήκευσης και εισαγωγή κόμβου

Στο αναδρομικό βήμα ο *αρχικός* ξαναστέλνει FIND_NODE στους κόμβους που γνωρίζει από προηγούμενα RPCs. Αυτή η αναδρομή μπορεί μάλιστα να ξεκινήσει πριν επιστρέψουν και τα *α* προηγούμενα RPCs. Από τους k κόμβους που έχει ακούσει ο αρχικός να είναι πλησιέστερα στο στόχο, διαλέγει *α* που δεν έχει ακόμα ρωτήσει και ξαναστέλνει FIND_NODE RPC σε αυτούς. Οι κόμβοι που δεν απαντούν γρήγορα αγνοούνται μέχρι και εάν απαντήσουν. Εάν ένας κύκλος από FIND_NODEs αποτύχει να επιστρέψει έναν κόμβο κοντύτερα από αυτόν που θεωρείται μέχρι τώρα κοντινότερος, ο αρχικός ξαναστέλνει FIND_NODE σε όλους τους k κοντινότερους κόμβους που δεν έχει ακόμα ρωτήσει. Η αναζήτηση τελειώνει, όταν ο αρχικός έχει ρωτήσει και λάβει απαντήσεις από όλους τους k κοντινότερους κόμβους που έχει «δει». Το Kademlia μπορεί να δρομολογήσει με χαμηλότερη καθυστέρηση, γιατί έχει την ευχέρεια της επιλογής ανάμεσα σε αυτούς τους k κόμβους, για να προωθήσει ένα μήνυμα.

Οι περισσότερες λειτουργίες γίνονται σε σχέση με τον παραπάνω αλγόριθμο αναζήτησης. Για να αποθηκεύσει ένας κόμβος ένα ζεύγος (κλειδί, τιμή), εντοπίζει τους k κοντινότερους κόμβους στο κλειδί και στέλνει STORE RPCs σε αυτούς.

Επιπλέον, κάθε κόμβος δημοσιεύει τα ζεύγη που φυλάει κάθε ώρα. Αυτό εξασφαλίζει στο σύστημα συνέπεια με μεγάλη πιθανότητα. Γενικά, απαιτείται και οι κόμβοι που δημοσίευσαν πρώτοι ένα ζεύγος να το δημοσιεύουν ξανά κάθε 24 ώρες. Διαφορετικά, όλα τα ζεύγη (κλειδί, τιμή) «λήγουν» ύστερα από 24 ώρες από την αρχική τους δημοσίευση προκειμένου να περιοριστεί η παλιά πληροφορία στο σύστημα.
Τέλος, για να υπάρχει συνέπεια στον κύκλο ζωής (δημοσίευση-αναζήτηση) ενός ζεύγους (κλειδί, τιμή), απαιτείται οποτεδήποτε ένα κόμβος x παρατηρεί έναν νέο κόμβο y, ο οποίος είναι κοντύτερα σε κάποια από τα ζεύγη που έχει ο x, να αντιγράφονται αυτά τα ζεύγη του x στον y χωρίς να αφαιρούνται από τη βάση δεδομένων του.
Για να βρει ένας κόμβος ένα ζεύγος (κλειδί, τιμή), ξεκινά με αναζήτηση των k κόμβων με τα πλησιέστερα IDs στο κλειδί. Ωστόσο, οι αναζητήσεις τιμής χρησιμοποιούν FIND_VALUE αντί για FIND_NODE RPCs. Επιπλέον, η διαδικασία διακόπτεται, μόλις κάποιος κόμβος επιστρέψει την τιμή. Για λόγους προσωρινής αποθήκευσης (caching), όταν πετύχει μια αναζήτηση, ο ζητών κόμβος αποθηκεύει το ζεύγος (κλειδί, τιμή) στον κοντινότερο στο κλειδί κόμβο που «είδε» και που δεν επέστρεψε την τιμή.
Λόγω της μονοκατευθυντικότητας της τοπολογίας, μελλοντικές αναζητήσεις για το ίδιο κλειδί αναμένεται να πετύχουν προσωρινά αποθηκευμένες εγγραφές πριν ρωτήσουν τον κοντινότερο κόμβο. Εάν ένα κλειδί γίνει πολύ δημοφιλές, το σύστημα θα καταλήξει να το αποθηκεύει σε πολλούς κόμβους. Για να αποφευχθεί το "over-caching", ο χρόνος «λήξης» ενός ζεύγους στη βάση δεδομένων ενός κόμβου είναι αντίστροφα εκθετικά ανάλογος του αριθμού των κόμβων ανάμεσα στον κόμβο και στον κόμβο με ID πλησιέστερα στο κλειδί. Ενώ μια απλή διαγραφή LRU (Least Recently Used) θα έδινε παρόμοια κατανομή στους χρόνους ζωής, δεν υπάρχει τρόπος να επιλεχτεί το μέγεθος της μνήμης προσωρινής αποθήκευσης (cache), εφόσον οι κόμβοι δε γνωρίζουν εκ των προτέρων πόσες τιμές θα αποθηκεύσει το σύστημα.
Οι κάδοι θα διατηρούνται γενικά «φρέσκοι», καθώς θα ανανεώνουν τα δεδομένα τους από την κίνηση των αιτήσεων στο σύστημα. Για την αποφυγή παθολογικών καταστάσεων, όταν δεν υπάρχει κίνηση, κάθε κόμβος ανανεώνει έναν κάδο του, όταν δεν έχει κάνει κανένας κόμβος αναζήτηση στην περιοχή του για μια ώρα. Η ανανέωση αυτή έγκειται στην επιλογή ενός τυχαίου ID στην περιοχή του κάδου και στην εκκίνηση μιας αναζήτησης για αυτό το ID.
Για να προσχωρήσει στο δίκτυο ένας κόμβος x, πρέπει να έχει επαφή με έναν τουλάχιστον κόμβο του δικτύου, έστω y. Ο x εισάγει τον y στον κατάλληλο k-κάδο του και μετά ξεκινά μια αναζήτηση κόμβου για το δικό του ID. Έτσι, γεμίζει όλους τους k-κάδους του και γνωστοποιεί την παρουσία του στο υπόλοιπο δίκτυο.

Συμπερασματικά
Λόγω της πρωτότυπης τοπολογίας που βασίζεται στο XOR μέτρο απόστασης, το Kademlia είναι ένα σύστημα peer-to-peer που συνδυάζει συνέπεια, απόδοση, δρομολόγηση με ελαχιστοποιημένη καθυστέρηση και συμμετρική, μονοκατευθυντική τοπολογία. Επιπλέον, εισάγει μία παράμετρο ταυτοχρονισμού, α, που επιτρέπει την επιλογή ανάμεσα σε εύρος ζώνης για την ασύγχρονη επιλογή του βήματος με την ελάχιστη καθυστέρηση και σε ανάκαμψη από αποτυχίες ανεξαρτήτως χρόνου. Τέλος, το Kademlia είναι το πρώτο σύστημα peer-to-peer το οποίο εκμεταλλεύεται το γεγονός ότι οι αποτυχίες κόμβων είναι αντίστροφα σχετιζόμενες με το χρονικό διάστημα που οι ίδιοι κόμβοι είναι ζωντανοί. [29]

3.2.2.3 Chord

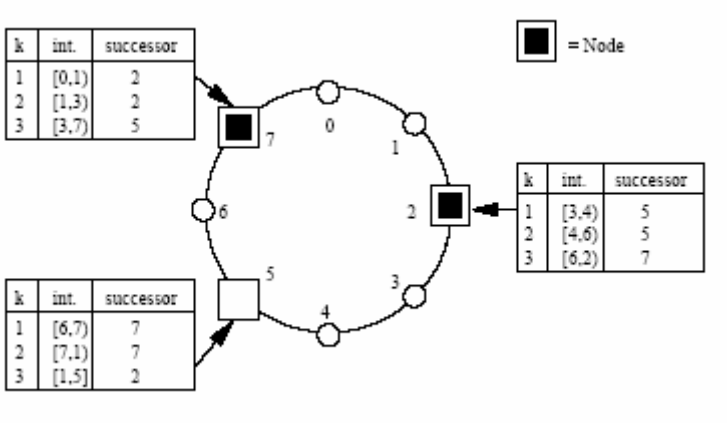
Το Chord είναι μια κατανεμημένη υπηρεσία αναζήτησης, η οποία είναι κλιμακούμενη σε μέγεθος και αποκεντρωμένη, και μπορεί να χρησιμοποιηθεί ως βάση για συστήματα peer-to-peer γενικού σκοπού. Τα πιο πολλά συστήματα peer-to-peer εμφανίζουν κάποια χαρακτηριστικά που ενισχύουν την απόδοση τους, όπως η εφεδρική αποθήκευση, η μονιμότητα και αποδοτική τοποθέτηση των δεδομένων, η επιλογή των κοντινών εξυπηρετητών, η πιστοποίηση και η ιεραρχική ονοματοδοσία. Το Chord δεν υλοποιεί αυτές τις υπηρεσίες άμεσα, αλλά μάλλον επιλέγει μια εύκαμπτη και υψηλής απόδοσης αρχή αναζήτησης, πάνω στην οποία μπορούν να βασιστούν αποδοτικά οι παραπάνω λειτουργικότητες. Με άλλα λόγια, η σχεδιαστική φιλοσοφία έγκειται στο διαχωρισμό του προβλήματος της αναζήτησης από την επιπλέον λειτουργικότητα. Τοποθετώντας επιπλέον χαρακτηριστικά πάνω σε μία υπηρεσία αμιγούς αναζήτησης, το σύστημα κερδίζει σε ευρωστία και δυνατότητα κλιμάκωσης.
Το Chord είναι σχεδιασμένο έτσι ώστε να προσφέρει την απαραίτητη λειτουργικότητα για την υλοποίηση συστημάτων γενικού σκοπού διατηρώντας παράλληλα τη μέγιστη ελαστικότητα. Ουσιαστικά, είναι ένα σύστημα αποδοτικής και κατανεμημένης αναζήτησης που βασίζεται στο συνεχή κατακερματισμό. Παρέχει μία μονοσήμαντη αντιστοίχηση ανάμεσα στο χώρο των αναγνωριστικών και στο σύνολο των κόμβων. Ένας κόμβος μπορεί να είναι ένας υπολογιστής ή μια διεργασία με IP διεύθυνση και αριθμό πόρτας. Κάθε κόμβος συνδέεται με ένα αναγνωριστικό Chord. Το Chord αντιστοιχεί κάθε αναγνωριστικό a στον κόμβο με το μικρότερο αναγνωριστικό που είναι μεγαλύτερο από a. Αυτός ο κόμβος καλείται *διάδοχος του* a.
Το Chord είναι αποδοτικό. Ο καθορισμός του διαδόχου ενός αναγνωριστικού απαιτεί με μεγάλη πιθανότητα την ανταλλαγή **O (log N)** μηνυμάτων όπου N είναι ο αριθμός των εξυπηρετητών στο δίκτυο Chord. Η εισαγωγή ή αφαίρεση ενός εξυπηρετητή από το δίκτυο μπορεί να επιτευχθεί με κόστος O (log N) μηνύματα.

Μία διαστρωματωμένη εφαρμογή Chord
Χρησιμοποιώντας ένα επιπλέον στρώμα μετάφρασης ονομάτων υψηλού επιπέδου σε αναγνωριστικά Chord, το Chord μπορεί να χρησιμοποιηθεί σαν μία ισχυρή υπηρεσία αναζήτησης. Έτσι, πάνω σε ένα στρώμα Chord μπορεί να τοποθετηθεί ένα στρώμα DHASH (Distributed HASH table) και μία peer-to-peer εφαρμογή αποθήκευσης. Παρακάτω φαίνεται η κατανομή της λειτουργικότητας σε μια εφαρμογή αποθήκευσης.

Στρώμα	Λειτουργικότητα
Chord	Αντιστοιχίζει αναγνωριστικά σε κόμβους - διαδόχους
DHASH	Συσχετίζει τιμές με αναγνωριστικά
Εφαρμογή	Παρέχει ένα interface συστήματος αρχείων

Πληροφορία ανά κόμβο
Το Chord χρησιμοποιεί συνεχές κατακερματισμό, για να αντιστοιχήσει κόμβους σε έναν κυκλικό χώρο αναγνωριστικών των m bits. Όπως αναφέραμε και παραπάνω, κάθε αναγνωριστικό a αντιστοιχίζεται στον κόμβο με το μικρότερο αναγνωριστικό που είναι μεγαλύτερο ή ίσο με το a στον κυκλικό χώρο. Αυτός είναι ο *διάδοχος* του a.
Για την υλοποίηση της συνάρτησης διαδόχου όλοι οι κόμβοι διατηρούν έναν πίνακα δρομολόγησης με m εισόδους, ο οποίος καλείται *πίνακας finger*. Αυτός ο πίνακας αποθηκεύει πληροφορίες για άλλους κόμβους στο σύστημα. Κάθε είσοδος περιέχει

ένα αναγνωριστικό κόμβου και τη διεύθυνση δικτύου του (IP διεύθυνση και αριθμό πόρτας). *Η k-ιοστή είσοδος στον πίνακα finger του κόμβου r είναι ο μικρότερος κόμβος s που είναι μεγαλύτερος του r + 2^k*. Ο κόμβος s καλείται επίσης διάδοχος τάξης k του κόμβου r. Ο αριθμός μοναδικών εισόδων στον πίνακα finger είναι O (log N). Ο πίνακας finger μπορεί επίσης να ερμηνευτεί ως m διαστήματα αναγνωριστικών που αντιστοιχίζονται στις m εισόδους του πίνακα: το τάξης k διάστημα ενός κόμβου r ορίζεται το ((r + 2^{k-1}) mod 2^m, (r + 2^k) mod 2^m]. Στην Εικόνα 4, όπου m = 3 και υπάρχουν οι κόμβοι 2, 5 και 7, ο άμεσος διάδοχος του κόμβου 5 είναι ο (5 +2^0) mod 2^3 = 6, δηλαδή ο κόμβος 7.



Εικόνα 4: Εύρεση διαδόχου. [30]

Κάθε κόμβος διατηρεί επίσης ένα δείκτη στον αμέσως προηγούμενο κόμβο. Για συμμετρία ορίζεται ως άμεσος διάδοχος η πρώτη είσοδος στον πίνακα finger. Συνολικά, κάθε κόμβος πρέπει να διατηρεί έναν πίνακα finger με μέγιστο αριθμό εισόδων O (log N). Αυτός ο κατακερματισμός απαλλάσσει τους κόμβους από την ανάγκη να εντοπίσουν σχεδόν όλους τους κόμβους του συστήματος.

Αποτίμηση της συνάρτησης διαδόχου

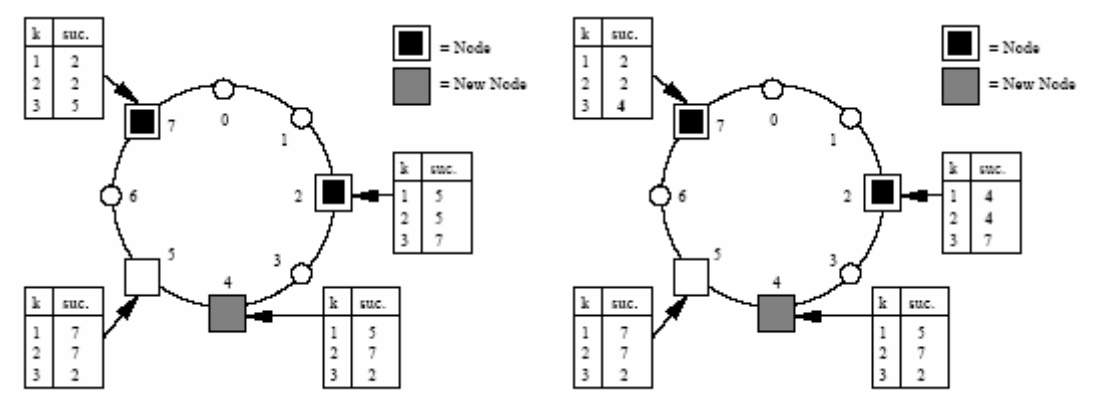
Εφόσον κάθε κόμβος διατηρεί πληροφορία για ένα μικρό υποσύνολο κόμβων του συστήματος, η αποτίμηση της συνάρτησης διαδόχου απαιτεί την επικοινωνία μεταξύ κόμβων σε κάθε βήμα του πρωτοκόλλου. Η αναζήτηση ενός κόμβου προχωράει σταδιακά με την αναγνώριση του διαδόχου σε κάθε βήμα.

Η αναζήτηση του διαδόχου του f ξεκινώντας από τον κόμβο r αρχίζει ελέγχοντας εάν το f είναι ανάμεσα στον r και στον άμεσο διάδοχο του r. Εάν ναι, η αναζήτηση τερματίζεται και επιστρέφεται ο διάδοχος του r. Διαφορετικά, ο r προωθεί την αίτηση αναζήτησης στο μεγαλύτερο κόμβο του πίνακα finger που προηγείται του f (έστω κόμβος s). Η ίδια διαδικασία ακολουθείται από τον s μέχρι να τερματιστεί η αναζήτηση. Επί παραδείγματι, υποθέτουμε ότι το σύστημα είναι σε σταθερή κατάσταση (όλοι οι πίνακες δρομολόγησης περιέχουν σωστές πληροφορίες) και έχει ξεκινήσει από τον κόμβο 2 (βλ. Εικόνα 4) αναζήτηση για το διάδοχο του αναγνωριστικού 6. Ο μεγαλύτερος κόμβος με αναγνωριστικό μικρότερο του 6 είναι ο 5. Ο στόχος της αναζήτησης (6) είναι στο διάστημα ανάμεσα στον 5 και στο διάδοχο του 5 (7). Άρα, 7 είναι η επιστρεφόμενη τιμή.

Όπως σκιαγραφείται και παραπάνω, ο αλγόριθμος έχει αναδρομική μορφή : εάν μία αίτηση αναζήτησης χρειάζεται πολλαπλά βήματα για την ολοκλήρωση της, το n-ιοστό βήμα ξεκινά από το (n-1)-ιοστό κόμβο εκ μέρους του αρχικού κόμβου. Η συνάρτηση αναζήτησης μπορεί να υλοποιηθεί και επαναληπτικά. Στην επαναληπτική υλοποίηση, ο αρχικός κόμβος είναι αυτός που κάνει αιτήσεις για πληροφορίες του

πίνακα finger σε κάθε στάδιο του πρωτοκόλλου. Και οι δύο υλοποιήσεις έχουν πλεονεκτήματα : η επαναληπτική προσέγγιση είναι ευκολότερη στην υλοποίηση και βασίζεται λιγότερο σε ενδιάμεσους κόμβους, ενώ η αναδρομική προσέγγιση ενδείκνυται περισσότερο για επιπλέον λειτουργικότητες, όπως είναι η προσωρινή αποθήκευση και η επιλογή εξυπηρετητή.

Εισαγωγή και αφαίρεση κόμβου
Όταν ένας νέος κόμβος r προσχωρεί στο δίκτυο, πρέπει να αρχικοποιήσει τον πίνακα finger του. Οι υπάρχοντες κόμβοι πρέπει επίσης να ενημερώσουν τους πίνακες τους σύμφωνα με το νέο δεδομένο της ύπαρξης του r.



Εικόνα 5: Αρχικοποίηση και ανανέωση πινάκων δρομολόγησης. [30]

Όταν ένα σύστημα είναι σταθεροποιημένο, ένας νέος κόμβος r μπορεί να αρχικοποιήσει τον πίνακα finger του ρωτώντας έναν ήδη υπάρχοντα κόμβο για τους αντίστοιχους διαδόχους των κατώτερων τιμών των k διαστημάτων στον πίνακα του r. Οι κόμβοι, των οποίων οι πληροφορίες ακυρώνονται από την εισαγωγή του r, μπορούν να βρεθούν από τον πίνακα finger του r ακολουθώντας τους δείκτες των προηγούμενων κόμβων. Αυτοί θα είναι οι κόμβοι που θα ενημερωθούν από τον r πως πρέπει να ανανεώσουν τους πίνακες τους.
Η αφαίρεση ενός κόμβου από το δίκτυο περιλαμβάνει μια σειρά από βήματα παρόμοια με αυτά της εισαγωγής ενός κόμβου στο δίκτυο. Παράλληλες προσχωρήσεις, παράλληλες αποχωρήσεις και αποτυχίες αντιμετωπίζονται με τη διατήρηση της αρχής ότι όλοι οι κόμβοι γνωρίζουν τους άμεσους διαδόχους τους και τους άμεσους προηγούμενους κόμβους τους, και επιτρέποντας στον πίνακα finger να μεταβάλλεται στο χρόνο, για να καταλήξει στη σταθερή κατάσταση. Τέλος, σημειώνουμε ότι η σωστή μεταχείριση των αποτυχιών απαιτεί από τους κόμβους να αποθηκεύουν και τους k διαδόχους και όχι μόνο τον άμεσο διάδοχο.

Το API του Chord
Το API του Chord παρέχει τη βασική λειτουργικότητα του εντοπισμού σε έναν διαστρωματωμένο σχεδιασμό. Δύο αρχές σχεδιασμού διευκολύνουν τη χρήση του Chord σε μία διαστρωματωμένη αρχιτεκτονική : ελάχιστη λειτουργικότητα και ελάχιστη εκτεθειμένη πληροφορία. Ελαχιστοποιώντας την έκταση της λειτουργικότητας που ενσωματώνεται στο Chord (κατώτερο επίπεδο), ελαχιστοποιούνται οι περιορισμοί που τίθενται σε υψηλότερα επίπεδα, τα οποία εξαρτώνται από το Chord. Η χρησιμότητα του Chord API είναι περισσότερο εμφανής κατά το σχεδιασμό των στρωμάτων που τοποθετούνται πάνω στο βασικό στρώμα

Chord και τα οποία είναι απαραίτητα για τη δόμηση μεγάλων peer-to-peer εφαρμογών ανταλλαγής αρχείων.
Ωστόσο τα μεγάλα συστήματα που βασίζονται στο Chord υπόκεινται σε αρκετούς περιορισμούς. Αυτό δε συμβαίνει, γιατί το Chord δεν παρέχει ένα αρκετά ελαστικό σύνολο χαρακτηριστικών, αλλά γιατί τα υψηλότερα στρώματα επιθυμούν πρόσβαση στην εσωτερική κατάσταση του Chord κατά τη διάρκεια του υπολογισμού της. Για να παρέχεται αυτή η πρόσβαση με ταυτόχρονη διατήρηση των αφαιρετικών ορίων, επιτρέπεται σε όλα τα στρώματα να εγγράφονται σε συναρτήσεις "callback" για γεγονότα, για τα οποία ενδιαφέρονται και να αποτιμούν τη συνάρτηση διαδόχου σε κάθε βήμα ξεχωριστά.
Χαρακτηριστικά αναφέρουμε ότι η *next_hop(j, k)* στέλνει ένα μήνυμα στον κόμβο j ρωτώντας τον για τη μικρότερη είσοδο στον finger πίνακά του που είναι μεγαλύτερη του k. Αυτό επιτρέπει στους κόμβους που καλούν τη συνάρτηση να ελέγχουν την εκτέλεση του αλγόριθμου αναζήτησης του Chord βήμα προς βήμα. Επιπλέον, το στρώμα DHASH (βλ. παρακάτω) χρησιμοποιεί το *callback interface*, για τη μεταφορά τιμών όταν κόμβοι προσχωρούν ή αποχωρούν από το σύστημα. Το DHASH αποτιμά επίσης τη συνάρτηση διαδόχου βήμα προς βήμα, για να αποθηκεύει προσωρινά στις διαδρομές αναζήτησης.

Κατανεμημένη υπηρεσία κατακερματισμού
Το Chord δεν είναι σύστημα αποθήκευσης : αντιστοιχίζει κλειδιά με κόμβους (αντί με τιμές). Μια χρήσιμη επέκταση σε αυτό το σύστημα είναι ένας Κατανεμημένος Πίνακας Κατακερματισμού (Distributed Hash Table –DHASH). Οι δύο βασικές κλήσεις του DHASH API είναι:

- DHASH : : insert (key, value)

  εισαγωγή τιμής υπό κλειδί και επιστροφή τιμής σχετιζόμενης με το κλειδί ή NULL, αν δεν υπάρχει το κλειδί. Μπορεί να υλοποιηθεί κατακερματίζοντας το κλειδί key, για να παραχθεί ένα 160-μπιτο αναγνωριστικό Chord των 160 bits, k, και αποθηκεύοντας την τιμή value στο διάδοχο του k.
- DHASH : : lookup (key)

  το κλειδί key κατακερματίζεται και δίνει το k και ο διάδοχος του k ερωτάται για την τιμή που σχετίζεται με το key. Η μεταφορά δεδομένων value από και προς κόμβους διεκπεραιώνεται και με ένα επιπλέον RPC interface που είναι διαφορετικό από αυτό που εξάγεται από το Chord.

Οι τιμές εισάγουν, όμως, μια επιπλοκή: όταν κόμβοι αποχωρούν ή προσχωρούν στο σύστημα, ο διάδοχος κόμβος ενός δεδομένου κλειδιού μπορεί να αλλάξει. Για να διατηρηθεί η αρχή ότι οι τιμές αποθηκεύονται στο διάδοχο των αντίστοιχων κλειδιών τους, το DHASH παρακολουθεί την άφιξη και αναχώρηση κόμβων χρησιμοποιώντας το callback interface που παρέχει το Chord και μετακινεί τις τιμές ανάλογα. Επί παραδείγματι, εάν η τιμή που αντιστοιχεί στο κλειδί 7 αποθηκεύεται στον κόμβο 10 και ο κόμβος 9 εισαχθεί στο σύστημα, η τιμή πρέπει να μετακινηθεί στον κόμβο 9.
Τέλος, επειδή το DHASH βασίζεται στο Chord, κληρονομεί τις επιθυμητές ιδιότητες του Chord. Εκπληρώνει μια αναζήτηση με O (log N) RPCs και δεν απαιτεί κεντρικό έλεγχο. Το στρώμα DHASH θέτει ένα επιπλέον κόστος μεταφοράς των κλειδιών στο σύστημα O (1 / N) κάθε φορά που φεύγει ή έρχεται ένας κόμβος στο σύστημα.

Αξιοπιστία και απόδοση
Το στρώμα DHASH εκμεταλλεύεται στοιχεία του Chord, για να πετύχει μεγαλύτερη αξιοπιστία και απόδοση. Για να εξασφαλίσει ότι οι αναζητήσεις θα πετύχουν ακόμα και στην περίπτωση απρόοπτων αποτυχιών κόμβων, το DHASH αποθηκεύει την τιμή

που αντιστοιχεί σε ένα δεδομένο κλειδί όχι μόνο στον άμεσο διάδοχο αυτού του κλειδιού , αλλά και στους επόμενους r διαδόχους. Η παράμετρος *r* μπορεί να ποικίλει προκειμένου να επιτευχθεί το επιθυμητό επίπεδο εφεδρείας στην αποθήκευση.
Η στενή σύνδεση ανάμεσα στην προσέγγιση του DHASH για την αντιγραφή και του Chord (και τα δύο χρησιμοποιούν τη γνώση των άμεσων διαδόχων ενός κόμβου) είναι ενδεικτική της αλληλεπίδρασης ανάμεσα στο Chord και στα υψηλότερα στρώματα.
Για τη βελτίωση της απόδοσης αναζήτησης του DHASH χρησιμοποιείται η εξής ιδιότητα του αλγορίθμου αναζήτησης του Chord: οι διαδρομές προς αναζήτηση ενός συγκεκριμένου διαδόχου (ξεκινώντας από διαφορετικούς κόμβους) μέσα στο δακτύλιο του Chord είναι πολύ πιθανό να επικαλύπτονται. Αυτές οι επικαλύψεις είναι πιο πιθανό να συμβούν κοντά στο στόχο της αναζήτησης, όπου κάθε βήμα του αλγόριθμου αναζήτησης κάνει ένα μικρότερο ‘βήμα’ στο χώρο των αναγνωριστικών. Εκεί δίνεται η δυνατότητα για προσωρινή αποθήκευση δεδομένων. Σε κάθε επιτυχημένη λειτουργία αναζήτησης του ζεύγους (k, v), η τιμή-στόχος, v, αποθηκεύεται σε κάθε κόμβο στη διαδρομή των κόμβων που διασχίζεται για την εύρεση του διαδόχου του k (πρόκειται για το μονοπάτι που επιστρέφεται από τη συνάρτηση διαδόχου του Chord).
Οι επόμενες αναζητήσεις αποτιμούν τη συνάρτηση διαδόχου βήμα προς βήμα χρησιμοποιώντας τη μέθοδο *next_hop* και ρωτούν κάθε ενδιάμεσο κόμβο για την τιμή v. Η αναζήτηση τερματίζεται γρήγορα, εάν ένας από αυτούς τους κόμβους μπορεί να επιστρέψει τη νωρίτερα αποθηκευμένη τιμή v.
Συνεπώς, οι τιμές «σκορπίζονται» στο δακτύλιο του Chord κοντά στους αντίστοιχους κόμβους διαδόχους. Επειδή η ανάκτηση ενός αρχείου οδηγεί και στην προσωρινή αποθήκευσή του, τα δημοφιλή αρχεία είναι ευρύτερα αποθηκευμένα (cached) από ότι τα μη δημοφιλή. Αυτή είναι μια επιθυμητή παρενέργεια του σχεδιασμού της προσωρινής αποθήκευσης. Η προσωρινή αποθήκευση μειώνει το μήκος διαδρομής που απαιτείται για την εύρεση μιας τιμής, επομένως και τον αριθμό των μηνυμάτων ανά αναζήτηση. Και μια τέτοια μείωση είναι ιδιαίτερα σημαντική δεδομένου ότι η καθυστέρηση επικοινωνίας μεταξύ κόμβων αποτελεί μία σοβαρή στένωση επίδοσης για το σύστημα.

Επιθέσεις άρνησης υπηρεσίας
Η κατανεμημένη φύση του Chord το βοηθά να αντισταθεί σε αρκετές αλλά όχι σε όλες τις επιθέσεις DoS. Επί παραδείγματι, το Chord «αντέχει» επιθέσεις που βγάζουν εκτός λειτουργίας κάποιες γραμμές δικτύου, αφού κόμβοι που βρίσκονται κοντά στο χώρο των αναγνωριστικών είναι απίθανο να εμφανίζουν δικτυακή τοπικότητα. Για τον εμποδισμό και των άλλων επιθέσεων DoS χρειάζονται επιπλέον χειρισμοί.
Ένα σύστημα αποθήκευσης βασισμένο στο Chord μπορεί να δεχτεί επίθεση με την εισαγωγή πολύ μεγάλης ποσότητας άχρηστων δεδομένων στο σύστημα, καθώς θα αποκλειστούν από την αποθήκευση τα «νόμιμα» αρχεία. Παρατηρώντας ότι η πυκνότητα των κόμβων κοντά σε έναν οποιονδήποτε κόμβο δίνει μια εκτίμηση του αριθμού των κόμβων στο σύστημα το Chord μπορεί να προστατευτεί μερικώς από αυτήν την επίθεση περιορίζοντας τον αριθμό των αρχείων που μπορεί να αποθηκεύσει ένα κόμβος του συστήματος. Παίρνεται μια τοπική απόφαση για την αναλογία των αρχείων στο σύστημα βασισμένη στον αριθμό των κόμβων του συστήματος. Κατ’ αυτόν τον τρόπο, κάθε χρήστης του συστήματος αναγκάζεται να τηρήσει αυτήν την αναλογία.
Κόμβοι που μπορούν μόνοι τους να διαλέξουν αναγνωριστικό μπορούν να διαγράψουν κάποιο κομμάτι πληροφορίας από το σύστημα θέτοντας τον εαυτό τους διάδοχο της πληροφορίας και μετά αποτυγχάνοντας να την αποθηκεύσουν, όταν αυτό

τους ζητείται. Αυτή η επίθεση μπορεί να εμποδιστεί απαιτώντας κάποια αντιστοιχία του αναγνωριστικού ενός κόμβου με την κατακερματισμένη IP διεύθυνσή του.
Οι κακόβουλοι κόμβοι μπορεί να αρνούνται να εκτελέσουν ορθά το πρωτόκολλο Chord με αποτέλεσμα να παρουσιάζουν αυθαίρετη και ασυνεπής συμπεριφορά. Ένας κόμβος που δεν εμφανίζει τη σωστή συμπεριφορά μπορεί να ανιχνευτεί επαληθεύοντας τις απαντήσεις του με άλλων κόμβων που θεωρείται ότι είναι συνεργάσιμοι. Για παράδειγμα, εάν ένας κόμβος l αναφέρει ότι διάδοχός του είναι ο s, ερωτάται ο s ποιος είναι ο προηγούμενός του κόμβος. Η απάντηση που αναμένεται είναι προφανώς ο l. Μια ομάδα τέτοιων κόμβων, όπως ο s, μπορεί να συνεργαστεί για τη συγκρότηση ενός δικτύου Chord με «έμπιστους» κόμβους. Δεν υπάρχει κατανεμημένη λύση στο παραπάνω πρόβλημα, αλλά θεωρώντας ότι οι κόμβοι αρχικοποίησης είναι «έμπιστοι» αποφεύγεται μια τέτοια επίθεση.

Σχεδιασμός ενός συστήματος αποθήκευσης
Χρησιμοποιώντας το Chord ως πυρήνα για ένα peer-to-peer σύστημα αποθήκευσης εμφανίζεται το πρόβλημα της αποδοτικής κατανομής του φόρτου ανάμεσα στους κόμβους παρά τις μεγάλες διαφοροποιήσεις στη δημοτικότητα των αρχείων. Ιδιαίτερα σημαντικό στη δόμηση αυτού του συστήματος είναι το πώς θα αντιστοιχισθούν τα αρχεία σε κόμβους.
Μία δυνατή υλοποίηση θα χρησιμοποιούσε το στρώμα DHASH ως peer-to-peer αποθηκευτικό σύστημα. Σε μία τέτοια σχεδίαση τα περιεχόμενα ενός αρχείου εισάγονται κατευθείαν στο σύστημα DHASH αποκτώντας για κλειδί είτε τον κατακερματισμό των περιεχομένων του αρχείου είτε τον κατακερματισμό ενός ανθρωπίνως κατανοητού ονόματος. Ωστόσο εάν ένα αρχείο γίνει πολύ δημοφιλές, το βάρος της παράδοσης ενός τέτοιου αρχείου δε θα κατανεμηθεί. Το σχήμα προσωρινής αποθήκευσης που αναφέρθηκε βοηθά τα μικρά αρχεία, αλλά δεν είναι πρακτικό για τα μεγάλα.
Μια εναλλακτική προσέγγιση χρησιμοποιεί το DHASH σαν στρώμα ανακατεύθυνσης: το DHASH αντιστοιχεί αναγνωριστικά αρχείων σε μια λίστα από IP διευθύνσεις όπου το αρχείο είναι διαθέσιμο. Σε αυτή τη σχεδίαση το DHASH λειτουργεί ανάλογα με το σύστημα DNS, αλλά δε βασίζεται σε ένα ειδικό σύνολο από κεντρικούς εξυπηρετητές όπως το DNS. Μόλις επιλέγεται μια IP διεύθυνση, τα αρχεία ανακτώνται με τη χρήση κάποιου άλλου πρωτοκόλλου μεταφοράς (HTTP, SSL, SFS κ.α.).
Η διατήρηση μιας λίστας, που ανανεώνεται δυναμικά, από δυνατούς εξυπηρετητές για ένα αρχείο λύνει το πρόβλημα των δημοφιλών αρχείων κατανέμοντας το φορτίο ανάμεσα σε όλους τους εξυπηρετητές της λίστας. Ωστόσο, αυτή η σχεδίαση απαιτεί βελτιστοποιήσεις, όπως η προσωρινή και η εφεδρική αποθήκευση, να υλοποιηθούν διπλά: μία στη στοίβα του Chord και μία στο πρωτόκολλο μεταφοράς. Για αυτό το λόγο χρειάστηκε μια λύση για το πρόβλημα του δημοφιλούς αρχείου πιο κοντινή στους μηχανισμούς του πρωτοκόλλου Chord.
Αυτό μπορεί να επιτευχθεί χρησιμοποιώντας το Chord για την αντιστοίχηση τμημάτων αρχείων (blocks) , αντί ολόκληρων αρχείων, σε εξυπηρετητές. Σε αυτό το σχήμα, τα αρχεία χωρίζονται σε blocks και κάθε block εισάγεται στο στρώμα DHASH χρησιμοποιώντας τον κρυπτογραφημένο κατακερματισμό των περιεχομένων του block σαν κλειδί. Ένα κομμάτι από μετα-δεδομένα εισάγεται επίσης στο σύστημα, για να δίνεται στο αρχείο ένα μόνο όνομα. Επιπλέον, αντίστοιχα με ένα σύστημα αρχείων, το σύστημα μπορεί να επεκταθεί, για να περιλαμβάνει και καταλόγους. Αυτή η προσέγγιση σκορπά επιθετικά ένα μεγάλο αρχείο σε πολλούς εξυπηρετητές κατανέμοντας κατά αυτόν τον τρόπο την εξυπηρέτησή του. Ακόμη,

κληρονομεί τις βελτιώσεις αξιοπιστίας και επίδοσης του στρώματος DHASH. Προφανώς, αρχεία μικρότερα από το μέγεθος ενός block εξυπηρετούνται από έναν και μόνο κόμβο. Το σχήμα προσωρινής αποθήκευσης είναι υπεύθυνο για τη διανομή αυτών των αρχείων και για το φόρτο εξυπηρέτησής τους, εάν γίνουν δημοφιλή.
Το κυριότερο μειονέκτημα αυτού του σχεδίου πηγάζει από την ίδια ιδιότητα που το κάνει επιθυμητό: επειδή σκορπίζουμε ένα αρχείο σε πολλούς εξυπηρετητές, για κάθε αρχείο πρέπει να πληρώσουμε το κόστος πολλών αναζητήσεων DHASH και αποτιμήσεων της συνάρτησης διαδόχου. Μια απλή υλοποίηση μπορεί να απαιτήσει **(S × L × log N) / B** δευτερόλεπτα, για να ανακτήσει ένα αρχείο των S byte όπου N είναι ο αριθμός των εξυπηρετητών στο δίκτυο, B είναι το μέγεθος του block και L είναι η μέση καθυστέρηση δικτύου. Η παραπάνω καθυστέρηση επιχειρείται να μειωθεί με επιθετική προ-μεταφορά των δεδομένων και επιλογή ενός εξυπηρετητή από ένα σύνολο εξυπηρετητών που είναι κοντά δικτυακά στον αιτούντα κόμβο.

Αυθεντικότητα
Ένα σύστημα αρχείων βασιζόμενο στο Chord μπορεί να παρέχει εγγυήσεις αυθεντικότητας με χρήση μηχανισμών του εξυπηρετητή SFS Read-Only (SFSRO) [31]. Με τον όρο αυθεντικότητα εννοείται η επιβεβαίωση της ακεραιότητας του αρχείου. Στον SFSRO τα blocks του συστήματος αρχείων ονομάζονται σύμφωνα με τον κρυπτογραφημένο κατακερματισμό των περιεχομένων τους. Αυτό το αναγνωριστικό δεν μπορεί από τη φύση του να πλαστογραφηθεί. Για την ονομασία συστημάτων αρχείων χρησιμοποιούνται ονόματα διαδρομών (pathnames) που πιστοποιούνται μόνα τους: το block που περιέχει τον αρχικό δείκτη (root inode) ενός συστήματος αρχείων ονομάζεται σύμφωνα με το δημόσιο κλειδί του εκδότη και υπογράφεται από αυτό το δημόσιο κλειδί. Το στρώμα DHASH μπορεί να επαληθεύσει ότι ο αρχικός δείκτης έχει υπογραφεί με το κλειδί με το οποίο έχει εισαχθεί. Έτσι, εμποδίζονται οι μη-εξουσιοδοτημένες ανανεώσεις σε ένα σύστημα αρχείων. Τέλος, η ονομασία των συστημάτων αρχείων με δημόσιο κλειδί δεν παράγει ανθρωπίνως κατανοητά ονόματα αρχείων. Ωστόσο αυτό δεν αποτελεί σοβαρό μειονέκτημα σε ένα περιβάλλον hypertext, ή σε ένα περιβάλλον με δείκτες και συμβολικές συνδέσεις.

Προβλήματα του Chord
Οι εφαρμογές που έχουν δομηθεί πάνω στο Chord αντιμετωπίζουν έναν αριθμό προβλημάτων, όπως τα παρακάτω:

- Ο παραπάνω σχεδιασμός ηθελημένα διαχωρίζει ερωτήσεις ανωνυμίας από τη διαδικασία του εντοπισμού. Η ανώνυμη δημοσίευση και το ανώνυμο διάβασμα είναι δύσκολο να προστεθούν σε ένα σύστημα Chord δεδομένης της ισχυρής αντιστοίχησης μεταξύ ενός αρχείου και του κόμβου που είναι υπεύθυνος για την εξυπηρέτηση αυτού του αρχείου. Με ένα υπερκείμενο δίκτυο mix-network [32] πάνω στο Chord θα μπορούσε να υποστηριχτεί η ανώνυμη δημοσίευση και το ανώνυμο διάβασμα.
- Η δόμηση ενός καταλόγου με όλα τα αρχεία που είναι αποθηκευμένα στο Chord είναι μια στρωτή και απλή διαδικασία: ένας δείκτης μπορεί να επισκέπτεται κάθε κόμβο στο σύστημα ακολουθώντας τους δείκτες στους διαδόχους. Ωστόσο, η αποθήκευση ενός δείκτη και η εξυπηρέτηση ερωτήσεων χωρίς την καταφυγή σε μια γενική αρχή παραμένει ένα ανοικτό θέμα. Εναλλακτικά, θα μπορούσε το σύστημα Chord να δίνεται σε έναν WWW gateway και να εξυπηρετείται από τις υπάρχουσες υπηρεσίες καταλόγου.

- Η κατεύθυνση αιτήσεων σε εξυπηρετητές κοντινούς στη δικτυακή τοπολογία είναι σημαντικό για τη μείωση των καθυστερήσεων των αιτήσεων. Για να γίνει αυτό πρέπει να μετρηθεί η απόδοση των εξυπηρετητών στο σύστημα. Ωστόσο, επειδή το Chord διανέμει επιθετικά αρχεία σε άσχετους μεταξύ τους εξυπηρετητές σε ένα μεγάλο δίκτυο δεν είναι πολύ πιθανό να συναντήσουμε τον ίδιο εξυπηρετητή πολλαπλές φορές. Αυτό δυσκολεύει αρκετά τη διατήρηση τεχνικών μέτρησης για την απόδοση των εξυπηρετητών.

Συμπερασματικά
Η επίδοση και η αξιοπιστία των συστημάτων peer-to-peer περιορίζεται από μη ελαστικές αρχιτεκτονικές που επιχειρούν να βρουν λύση σε πολλά προβλήματα. Χρησιμοποιώντας το βασικό στρώμα Chord για το διαχωρισμό των προβλημάτων της κατανομής δεδομένων, της αυθεντικότητας και της ανωνυμίας, τα συστήματα peer-to-peer μπορούν να αποφασίσουν πού να συμβιβαστούν και σαν αποτέλεσμα προσφέρουν καλύτερη επίδοση, μεγαλύτερη αξιοπιστία και ακεραιότητα. [30]

3.2.2.4 Tapestry

Το Tapestry είναι μια επεκτάσιμη υποδομή, η οποία παρέχει αποκεντρωμένο εντοπισμό και δρομολόγηση ενός αντικειμένου (Decentralized Object Location and Routing – DOLR). Το DOLR interface εστιάζει στη δρομολόγηση μηνυμάτων σε τελικά σημεία, όπως είναι οι κόμβοι και τα αντίγραφα αντικειμένων. Το DOLR παρέχει εικονικούς πόρους, αφού τα τελικά σημεία παίρνουν ονόματα αναγνωριστικών κωδικοποιώντας μηδενική πληροφορία για τη φυσική τους θέση. Με αυτήν την υλοποίηση επιτρέπεται η παράδοση μηνυμάτων σε κινητά τελικά σημεία ή αντίγραφα τελικών σημείων κατά την παρουσία αστάθειας στην υποκείμενη υποδομή. Σαν αποτέλεσμα, ένα δίκτυο DOLR παρέχει μια απλή πλατφόρμα πάνω στην οποία μπορούν να υλοποιηθούν κατανεμημένες εφαρμογές, ενώ μπορεί να αγνοηθεί η δυναμικότητα του δικτύου. Ήδη το Tapestry έχει κάνει δυνατή την ανάπτυξη εφαρμογών αποθήκευσης μεγάλης κλίμακας όπως το OceanStore και συστημάτων multicast διανομής όπως το Bayeux.
Το Tapestry χρησιμοποιεί προσαρμοστικούς αλγόριθμους, για να επιδείξει αντοχή σε λάθη, καθώς αλλάζει το σώμα μελών του δικτύου και γίνονται δικτυακά λάθη. Η αρχιτεκτονική του είναι τμηματική και περιλαμβάνει μια εκτεταμένη λειτουργικότητα "upcall" γύρω από έναν απλό, υψηλής απόδοσης δρομολογητή. Αυτό το API επιτρέπει στους προγραμματιστές να αναπτύξουν και να επεκτείνουν την υπερκείμενη λειτουργικότητα, όταν η βασική λειτουργικότητα DOLR είναι ανεπαρκής για την εφαρμογή τους.
Το Tapestry επιτρέπει στις εφαρμογές να τοποθετούν αντικείμενα ανάλογα με τις ανάγκες τους και δε ρυθμίζει, όπως άλλα συστήματα peer-to-peer, τον αριθμό και τη θέση των αντιγράφων των αντικειμένων μέσω ενός DHT interface. Το Tapestry «δημοσιεύει» δείκτες θέσης μέσα στο δίκτυο, για να διευκολυνθεί η δρομολόγηση στα αντικείμενα που έχουν μικρή διασπορά στο δίκτυο. Αυτή η τεχνική δίνει στο Tapestry ιδιότητες τοπικότητας [33]: οι ερωτήσεις για κοντινά αντικείμενα ικανοποιούνται γενικά σε χρόνο ανάλογο της απόστασης ανάμεσα στην πηγή της ερώτησης και στο πλησιέστερο αντίγραφο του αντικειμένου.
Το Tapestry είναι υλοποιημένο σε Java και υπό κανονικές συνθήκες το σχετικό κόστος καθυστέρησης (relative delay penalty – RPD) για τον εντοπισμό δύο κινητών τελικών σημείων είναι το πολύ δύο σε μία μεγάλη περιοχή. Σύμφωνα με τις προσομοιώσεις, οι λειτουργίες του Tapestry είναι επιτυχείς σχεδόν κατά το 100% του

χρόνου, ενώ υπό συνεχείς δικτυακές αλλαγές και υπό μαζικές αποτυχίες ή προσχωρήσεις παρουσιάζουν μικρές περιόδους υποβαθμισμένης επίδοσης (όσο διαρκεί η αυτο-διόρθωση). Αυτά τα αποτελέσματα καταστούν το Tapestry ικανό να λειτουργήσει σαν μία υπηρεσία μακράς διαρκείας σε δυναμικά και επιρρεπή σε λάθη δίκτυα όπως το Διαδίκτυο.

API Δικτύωσης του DOLR

Το Tapestry παρέχει ένα τύπου δεδομενογραφήματος interface επικοινωνίας με επιπλέον μηχανισμούς, για να αξιοποιηθούν οι θέσεις των αντικειμένων. Ακολουθούν κάποιοι σημαντικοί ορισμοί.

Οι κόμβοι Tapestry συμμετέχουν στο υπερκείμενο δίκτυο και τους ανατίθενται *nodeIDs* ομοιόμορφα και τυχαία από ένα μεγάλο χώρο αναγνωριστικών. Περισσότεροι από δύο κόμβοι μπορούν να φιλοξενούνται από το ίδιο φυσικό μηχάνημα. Στα τελικά σημεία μιας συγκεκριμένης εφαρμογής ανατίθενται μοναδικά αναγνωριστικά (*globally unique identifiers – GUID*) επιλεγμένα από τον ίδιο χώρο αναγνωριστικών. Το Tapestry χρησιμοποιεί έναν χώρο αναγνωριστικών των 160 bits με μία γενικά ορισμένη βάση (π.χ. η δεκαεξαδική βάση μας δίνει αναγνωριστικά των 40 ψηφίων). Το Tapestry υποθέτει ότι τα nodeIDs και τα GUIDs είναι γενικά ομοίως κατανεμημένα στο χώρο αναγνωριστικών, πράγμα που επιτυγχάνεται χρησιμοποιώντας έναν αλγόριθμο ασφαλούς κατακερματισμού όπως SHA-1 [34]. Γενικά, ένας κόμβος N έχει nodeID N_{id}, και ένα αντικείμενο O έχει GUID O_G.

Εφόσον η αποδοτικότητα του Tapestry γενικά βελτιώνεται όσο μεγαλώνει το μέγεθος του δικτύου, ωφελεί πολλές εφαρμογές να μοιράζονται το ίδιο μεγάλο υπερκείμενο δίκτυο Tapestry. Για να είναι δυνατή η συνύπαρξη πολλών εφαρμογών, κάθε μήνυμα περιέχει ένα αναγνωριστικό της εφαρμογής A_{id}, το οποίο χρησιμοποιείται για την επιλογή μιας διεργασίας ή εφαρμογής για την παράδοση του μηνύματος στον προορισμό (ομοίως με το ρόλο της πόρτας στο TCP/IP), ή για την επιλογή ενός χειριστή upcall όπου χρειάζεται.

Το API Δικτύωσης του DOLR αποτελείται από τέσσερις βασικές κλήσεις:

1) PUBLISHOBJECT (O_G, A_{id}): δημοσιεύει, ή κάνει διαθέσιμο, το αντικείμενο O στον τοπικό κόμβο. Αυτή η κλήση είναι καλύτερης δυνατής προσπάθειας, και δε λαμβάνει επιβεβαίωση.
2) UNPUBLISHOBJECT (O_G, A_{id}): κλήση καλύτερης δυνατής προσπάθειας για την κατάργηση αντιστοιχήσεων θέσης για το O.
3) ROUTETOOBJECT (O_G, A_{id}): δρομολογεί μήνυμα στη θέση ενός αντικειμένου με GUID O_G
4) ROUTETONODE (N, A_{id}, Exact): δρομολογεί μήνυμα στην εφαρμογή A_{id} στον κόμβο N. Το "Exact" προσδιορίζει εάν το ID του προορισμού πρέπει να ταιριάζει απόλυτα, για να παραδοθεί το περιεχόμενο του μηνύματος.

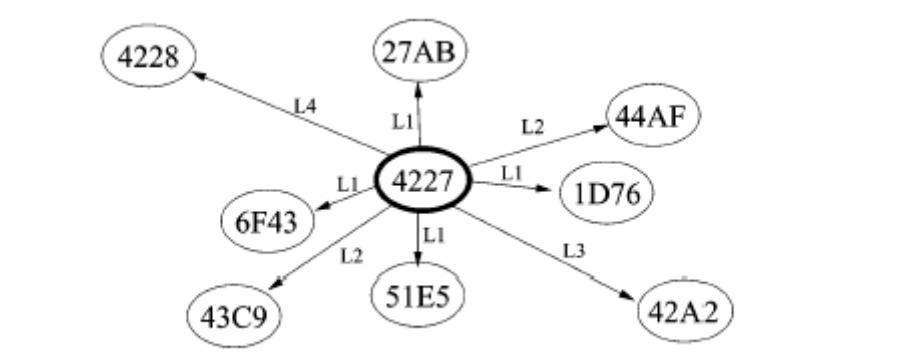
Δρομολόγηση και εντοπισμός αντικειμένου

Το Tapestry αντιστοιχεί δυναμικά κάθε αναγνωριστικό G σε έναν μοναδικό ζωντανό κόμβο, ο οποίος καλείται *ρίζα* του αναγνωριστικού, G_R. Εάν ένας κόμβος N υπάρχει με $N_{id} = G$, τότε αυτός ο κόμβος είναι η ρίζα του G. Για να παραδίδει μηνύματα, κάθε κόμβος διατηρεί έναν πίνακα δρομολόγησης που περιέχει nodeIDs και IP διευθύνσεις των κόμβων με τους οποίους επικοινωνεί. Αυτοί οι κόμβοι καλούνται *γείτονες* του τοπικού κόμβου. Κατά τη δρομολόγηση προς το G_R, τα μηνύματα προωθούνται κατά τις συνδέσεις των γειτόνων προς κόμβους με nodeIDs σταδιακά

πλησιέστερα (σύμφωνα με τη σύγκριση προθεμάτων) στο G στο χώρο των αναγνωριστικών.

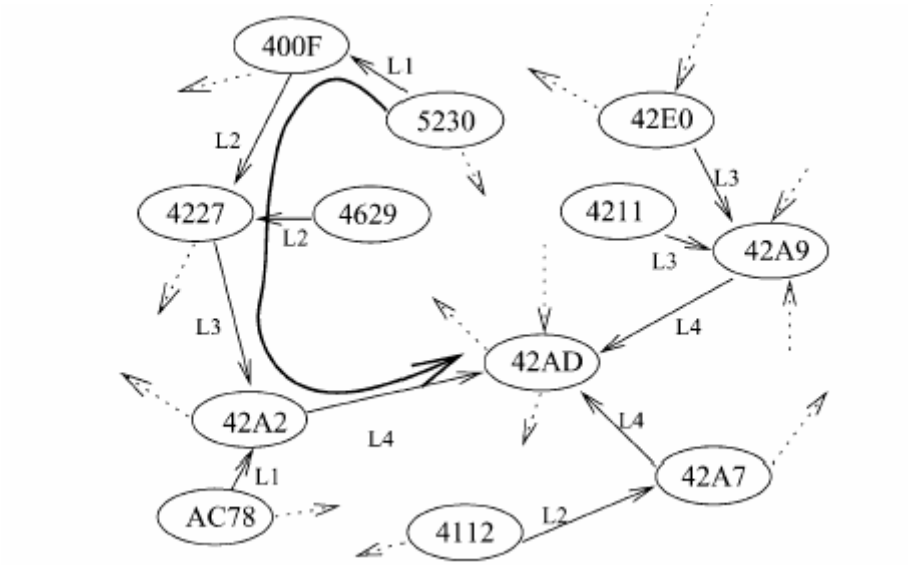
- Δίκτυο δρομολόγησης

Το Tapestry χρησιμοποιεί τοπικούς πίνακες σε κάθε κόμβο, οι οποίοι καλούνται *χάρτες γειτόνων* (neighbor maps), για τη δρομολόγηση υπερκείμενων μηνυμάτων στο ID προορισμού ψηφίο προς ψηφίο (π.χ. 4*** $\Rightarrow$ 42** $\Rightarrow$ 42^{A}* $\Rightarrow$ 42AD, όπου το * αντιπροσωπεύει έναν άγνωστο δεκαεξαδικό αριθμό). Αυτή η προσέγγιση είναι παρόμοια με τη δρομολόγηση μακρύτερου προθέματος που χρησιμοποιείται από την CIDR (Classless InterDomain Routing) ανάθεση IP διεύθυνσης [35]. Ένας κόμβος N έχει έναν χάρτη γειτόνων πολλαπλών επιπέδων, όπου σε κάθε επίπεδο περιέχονται συνδέσεις σε κόμβους που έχουν το ίδιο πρόθεμα μέχρι μια θέση ψηφίου στο ID και ο αριθμός των εγγραφών σε κάθε επίπεδο ισούται με τη βάση του ID. Η i-ιοστή εγγραφή στο j-ιοστό επίπεδο είναι το ID και η θέση του κοντινότερου κόμβου που ξεκινά με πρόθεμα (N, j – 1) + "i" (π.χ. η $9^{η}$ εγγραφή του $4^{ου}$ επιπέδου για τον κόμβο 325AE είναι ο κοντινότερος κόμβος με ID που ξεκινά με 3259). Αυτή η έννοια του «κοντινότερου» κόμβου προσδίδει στο Tapestry ιδιότητες τοπικότητας. Στην Εικόνα 6 φαίνεται το δίκτυο δρομολόγησης του Tapestry από την πλευρά ενός κόμβου. Οι εξερχόμενες γειτονικές συνδέσεις δείχνουν σε κόμβους με μερικά κοινό πρόθεμα. Οι εγγραφές υψηλότερου επιπέδου έχουν περισσότερα κοινά ψηφία με τον τοπικό κόμβο.



Εικόνα 6: Ο τοπικός πίνακας δρομολόγησης ενός κόμβου υπό μορφή δικτύου δρομολόγησης. [36]

Στο επόμενο σχήμα φαίνεται η διαδρομή που μπορεί να πάρει ένα μήνυμα μέσα στην υποδομή Tapestry. Ο δρομολογητής για το n-ιοστό βήμα μοιράζεται ένα πρόθεμα μήκους ≥ n με το ID του προορισμού. Έτσι, το Tapestry, για να δρομολογήσει, ψάχνει στο (n+1) – ιοστό επίπεδο του χάρτη για την εγγραφή που ταιριάζει με το επόμενο ψηφίο του ID του προορισμού. Αυτή η μέθοδος εγγυάται ότι οποιοσδήποτε κόμβος στο σύστημα θα βρεθεί σε το πολύ $\log_{\beta} N$ βήματα επιπέδου εφαρμογής, όταν το σύστημα έχει μέγεθος χώρου αναγνωριστικών N, IDs βάσης β, και υποθέτοντας ότι οι χάρτες γειτόνων είναι συνεπείς. Όταν ένα ψηφίο δεν μπορεί να ταιριάξει, το Tapestry ψάχνει για ένα «κοντινό» ψηφίο στον πίνακα δρομολόγησης. Αυτό καλείται **surrogate routing** [37]. Όταν, δηλαδή, ένα ID που δεν υπάρχει αντιστοιχίζεται σε κάποιο ζωντανό κόμβο με παρόμοιο ID.

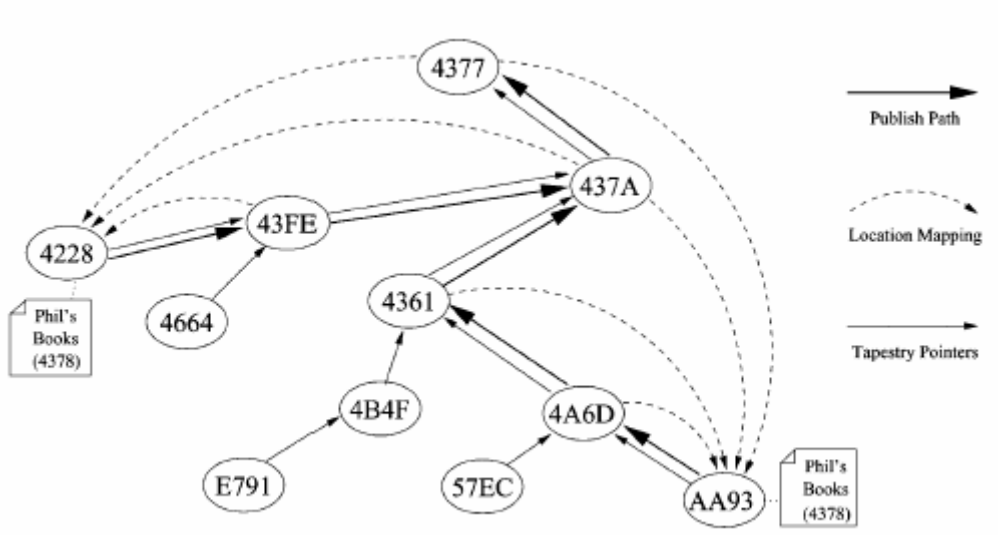


Εικόνα 7: Η διαδρομή ενός μηνύματος που ξεκινά από τον κόμβο 5230 και κατευθύνεται στον κόμβο 42AD. [36]

Κάθε αναγνωριστικό G αντιστοιχίζεται σε έναν μοναδικό κόμβο-ρίζα G_R με συνεχή χρήση της συνάρτησης NEXTHOP, η οποία επιλέγει τον εξερχόμενο σύνδεσμο. Αυτή η συνάρτηση εντοπίζει το επόμενο βήμα προς τη ρίζα δεδομένου του αριθμού του προηγούμενου βήματος, n, και του GUID προορισμού G. Επιστρέφει το επόμενο βήμα ή "self" , εάν ρίζα είναι ο τοπικός κόμβος.
Οι κύριοι σύνδεσμοι γειτόνων, όπως φαίνονται στην Εικόνα 6, έχουν εφεδρικούς συνδέσμους με ίδια προθέματα. Στο n-ιοστό επίπεδο δρομολόγησης, οι c σύνδεσμοι γειτόνων διαφέρουν μόνο στο n-ιοστό ψηφίο. Υπάρχουν $c \times \beta$ δείκτες σε κάθε επίπεδο, και το συνολικό μέγεθος ενός χάρτη γειτόνων είναι $\mathbf{c \times \beta \times \log_{\beta} N}$. Κάθε κόμβος αποθηκεύει επίσης αντίστροφες αναφορές (backpointers) στους κόμβους που δείχνουν σε αυτόν. Ο συνολικός αριθμός τέτοιων εγγραφών είναι $c \times \beta \times \log_{\beta} N$.

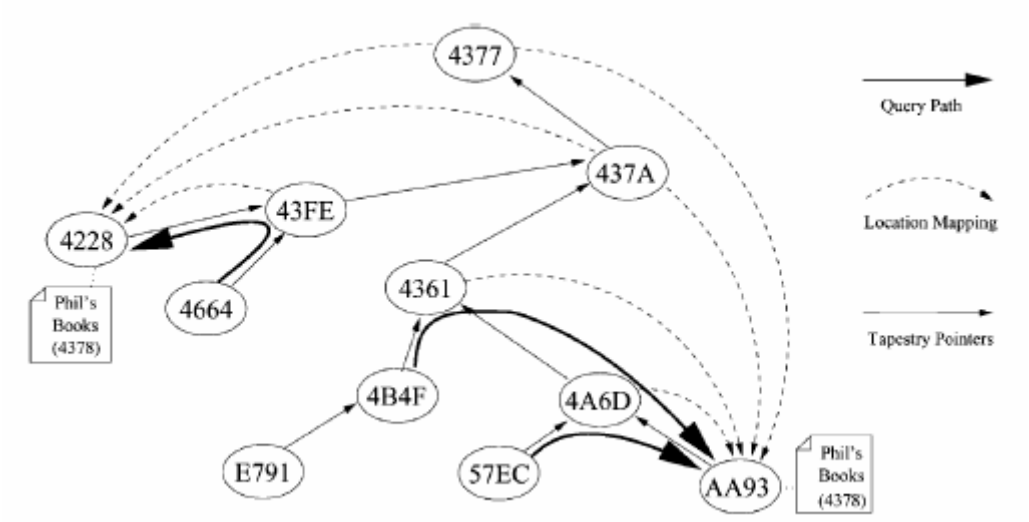
- Εντοπισμός και δημοσίευση αντικειμένου

Όπως προαναφέρθηκε, κάθε αναγνωριστικό G έχει έναν μοναδικό κόμβο-ρίζα G_R που του ανατίθεται από τη διαδικασία δρομολόγησης. Κάθε τέτοιος κόμβος-ρίζα αποκτά ένα μοναδικό δέντρο με μηνύματα από κόμβους-φύλλα να διασχίζουν ενδιάμεσους κόμβους στην πορεία τους για τη ρίζα. Αυτή η ιδιότητα χρησιμεύει για τον εντοπισμό αντικειμένων διανέμοντας "soft-state" πληροφορία καταλόγου στους κόμβους συμπεριλαμβανομένου και της ρίζας του αντικειμένου.
Ένας εξυπηρετητής S που αποθηκεύει ένα αντικείμενο O (με GUID O_G και ρίζα O_R) περιοδικά διαφημίζει ή *δημοσιεύει* αυτό το αντικείμενο δρομολογώντας ένα μήνυμα δημοσίευσης προς τον O_R. Γενικά, το nodeID του O_R είναι διαφορετικό από O_G. Ο O_R είναι ο μοναδικός κόμβος που φτάνει η δρομολόγηση με διαδοχικές κλήσεις της NEXTHOP (*, O_G). Κάθε κόμβος κατά μήκος της διαδρομής δημοσίευσης αποθηκεύει μία αντιστοίχηση δείκτη {O_G, S} αντί για ένα αντίγραφο του αντικειμένου. Όταν υπάρχουν αντίγραφα ενός αντικειμένου σε διαφορετικούς εξυπηρετητές, κάθε εξυπηρετητής δημοσιεύει το αντίγραφό του. Οι κόμβοι Tapestry αποθηκεύουν αντιστοιχήσεις θέσης για αντίγραφα αντικειμένων ταξινομημένες σύμφωνα με τη δικτυακή καθυστέρηση από τον εαυτό τους.



Εικόνα 8: Παράδειγμα δημοσίευσης αντικειμένου στο Tapestry. Δύο αντίγραφα του αντικειμένου 4378 διαφημίζονται / δημοσιεύονται στον κόμβο-ρίζα τους, 4377. Τα μηνύματα δημοσίευσης κατευθύνονται προς τη ρίζα αφήνοντας ένα δείκτη θέσης προς το αντικείμενο σε κάθε βήμα που συναντούν στη διαδρομή τους. [36]

Ένας πελάτης εντοπίζει το Ο δρομολογώντας ένα μήνυμα στο O_R. Κάθε κόμβος στη διαδρομή ελέγχει εάν αποθηκεύει αντιστοίχηση θέσης για το Ο. Εάν ναι, ανακατευθύνει το μήνυμα στον κάτοχο S. Διαφορετικά, προωθεί το μήνυμα παρακάτω προς το O_R. Στην Εικόνα 9 φαίνονται τρεις αναζητήσεις αντικειμένων.



Εικόνα 9: Παράδειγμα δρομολόγησης σε αντικείμενο στο Tapestry. Κόμβοι από διαφορετικά σημεία του δικτύου δρομολογούν μηνύματα προς (για) το αντικείμενο 4378. Όλα τα μηνύματα κινούνται προς τη ρίζα του 4378. Όταν συναντούν το μονοπάτι δημοσίευσης, ακολουθούν το δείκτη θέσης προς το πλησιέστερο αντίγραφο. [36]

Σε κάθε βήμα μειώνονται οι κόμβοι που ικανοποιούν τον περιορισμό του προθέματος κατά έναν παράγοντα ίσο με τη βάση του αναγνωριστικού. Μηνύματα που στέλνονται στον ίδιο προορισμό από δύο κοντινούς κόμβους γενικά θα συναντηθούν

σύντομα κατά την πορεία τους προς τον προορισμό, για τον εξής λόγο: κάθε βήμα αυξάνει το μήκος του προθέματος που απαιτείται για το επόμενο βήμα, η διαδρομή προς τη ρίζα είναι συνάρτηση μόνο του ID του προορισμού, και όχι του nodeID της πηγής (όπως στο Chord), και τα γειτονικά βήματα διαλέγονται με γνώμονα την τοπικότητα του δικτύου, η οποία αποτελεί δυναμικό στοιχείο. Συνεπώς, όσο πιο κοντά σε απόσταση δικτύου είναι ένας πελάτης σε ένα αντικείμενο, τόσο το συντομότερο οι ερωτήσεις του για το αντικείμενο θα συναντήσουν το μονοπάτι δημοσίευσης του αντικειμένου και τόσο πιο γρήγορα θα το φτάσουν. Εφόσον οι κόμβοι ταξινομούν τους δείκτες στα αντικείμενα σύμφωνα με την απόσταση από τους εαυτούς τους, οι ερωτήσεις δρομολογούνται σε κοντινά αντίγραφα των αντικειμένων.

Δυναμικοί αλγόριθμοι του Tapestry

1. Εισαγωγή κόμβου

Υπάρχουν τέσσερα βήματα κατά την εισαγωγή ενός κόμβου Ν σε ένα δίκτυο Tapestry.

α) Οι κόμβοι που χρειάζονται να γνωρίζουν την είσοδο του Ν ενημερώνονται για τον Ν, γιατί ο Ν συμπληρώνει μια άδεια είσοδο στον πίνακα δρομολόγησής τους.

β) Ο Ν μπορεί να γίνει η νέα ρίζα για υπάρχοντα αντικείμενα. Οι αναφορές σε αυτά τα αντικείμενα πρέπει πλέον να δείχνουν προς τον Ν, για να διατηρηθεί η διαθεσιμότητα των αντικειμένων.

γ) Οι αλγόριθμοι πρέπει να φτιάξουν ένα βελτιστοποιημένο πίνακα δρομολόγησης για τον Ν με ιδιότητες τοπικότητας.

δ) Οι κόμβοι κοντά στον Ν ενημερώνονται και μπορεί να χρησιμοποιήσουν τον Ν στους πίνακες δρομολόγησής τους σαν βελτιστοποίηση.

Η εισαγωγή του κόμβου Ν ξεκινά στον S που είναι η ρίζα-κόμβος στην οποία αντιστοιχεί το N_{id} στο υπάρχων δίκτυο. Ο S βρίσκει το p, το μήκος του μεγαλύτερου προθέματος που μοιράζεται το ID του με το N_{id}. Ο S στέλνει ένα μήνυμα *Acknowledged Multicast* που φτάνει σε όλους τους υπάρχοντες κόμβους που μοιράζονται το ίδιο πρόθεμα, όταν διασχίζεται το δέντρο που είναι βασισμένο στα nodeIDs τους. Όταν οι κόμβοι λαμβάνουν το μήνυμα, προσθέτουν τον Ν στους πίνακες δρομολόγησής τους και τροποποιούν αναφορές σε τοπικούς δείκτες όπως χρειάζεται. Έτσι, συμπληρώνονται τα βήματα α) και β).

Οι κόμβοι που λαμβάνουν το multicast μήνυμα επικοινωνούν με τον Ν και γίνονται το πρώτο σύνολο γειτόνων που χρησιμοποιείται για τη δόμηση του πίνακα δρομολόγησης του Ν. Ο Ν διεξάγει μια επαναληπτική αναζήτηση κοντινότερων γειτόνων ξεκινώντας με το επίπεδο δρομολόγησης p. Χρησιμοποιεί το σύνολο των γειτόνων του, για να γεμίσει το επίπεδο δρομολόγησης p, αφήνει στη λίστα τους κοντινότερους k κόμβους, και ζητά από τους k κόμβους να στείλουν σε αυτόν τους δείκτες προς τους προηγούμενούς τους σε αυτό το επίπεδο. Το σύνολο που προκύπτει περιέχει όλους τους κόμβους που δείχνουν σε οποιονδήποτε από τους k κόμβους στο προηγούμενο επίπεδο δρομολόγησης, και γίνεται το επόμενο σύνολο γειτόνων. Στη συνέχεια μειώνεται το p και επαναλαμβάνεται η παραπάνω διαδικασία μέχρι να γεμίσουν όλα τα επίπεδα. Έτσι ολοκληρώνεται το βήμα γ). Οι κόμβοι με τους οποίους έρχεται σε επαφή ο Ν κατά τον επαναληπτικό αλγόριθμο χρησιμοποιούν τον Ν, για να βελτιστοποιήσουν, όπου είναι δυνατό, τους πίνακες δρομολόγησής τους. Συνεπώς, συμπληρώνεται και το βήμα δ).

Για να διασφαλιστεί ότι οι κόμβοι που εισάγονται στο δίκτυο δεν αποτυγχάνουν να ενημερώσουν (και να ενημερωθούν) για την ύπαρξή τους, κάθε κόμβος Α που ανήκει στο multicast δέντρο κρατά πληροφορία για κάθε κόμβο Β που κάνει ακόμα multicast προς τα κάτω σε κάποιον από τους γείτονές του. Αυτή η πληροφορία συνήθως

πληροφορεί κάθε κόμβο C που έχει τον A στο multicast δέντρο του για τον B. Τέλος, το multicast δέντρο αφήνει μια λίστα από κενά στον πίνακα δρομολόγησης του νέου κόμβου. Ο κόμβοι ελέγχουν τους πίνακές τους σε σχέση με τον πίνακα δρομολόγησης του νέου κόμβου και τον ενημερώνουν με εγγραφές που μπορούν να γεμίσουν τα κενά.

2. Εκούσια διαγραφή κόμβου

Εάν ένα κόμβος N εγκαταλείπει το Tapestry ηθελημένα, ενημερώνει το σύνολο D των κόμβων που δείχνουν οι δείκτες προς τους προηγούμενούς του για την πρόθεσή του, στέλνοντας μαζί και έναν κόμβο-αντικαταστάτη από κάθε επίπεδο δρομολόγησης του δικού του πίνακα δρομολόγησης. Οι κόμβοι που ενημερώνονται στέλνουν κίνηση για την αναδημοσίευση του αντικειμένου και στον N και στον αντικαταστάτη του. Εντωμεταξύ, ο N δρομολογεί αναφορές σε αντικείμενα, για τα οποία αποτελεί ρίζα, στις νέες τους ρίζες και στέλνει σήμα στους κόμβους που ανήκουν στο D, όταν τελειώνει.

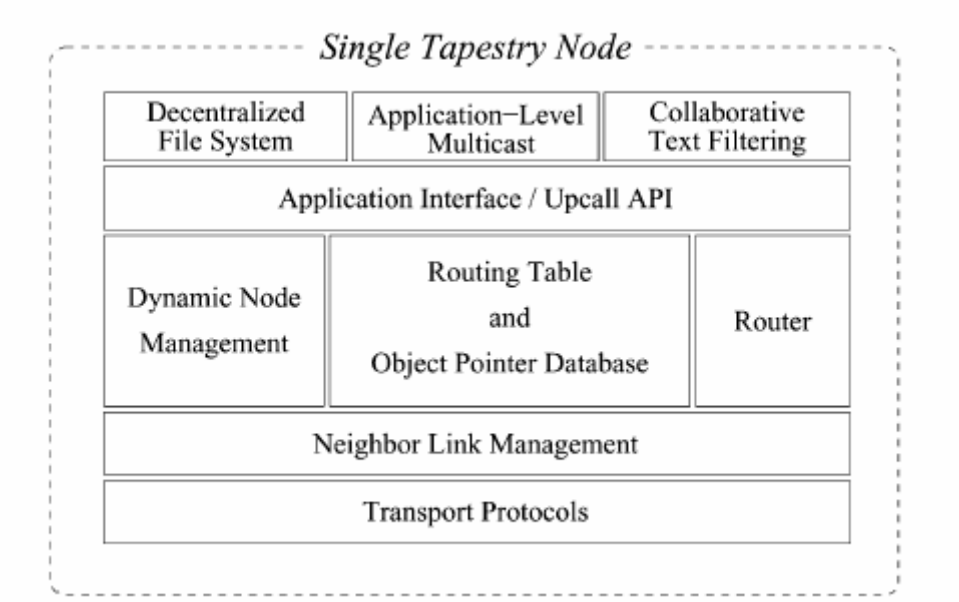
3. Ακούσια διαγραφή κόμβου

Σε ένα δυναμικό, επιρρεπές σε λάθη δίκτυο όπως είναι το Διαδίκτυο οι κόμβοι γενικά εγκαταλείπουν το δίκτυο απροετοίμαστοι και αυτό οφείλεται σε αποτυχίες συνδέσεων ή σε αποτυχίες άλλων τμημάτων του δικτύου. Επιπλέον, κόμβοι μπορεί να μπουν και να βγουν από το δίκτυο πολλές φορές σε ένα μικρό χρονικό διάστημα. Το Tapestry βελτιώνει τη διαθεσιμότητα και δρομολόγηση αντικειμένου σε ένα τέτοιο περιβάλλον προσθέτοντας εφεδρεία στους πίνακες δρομολόγησης και στις αναφορές θέσης αντικειμένων (π.χ. οι c-1 εφεδρικοί δείκτες προώθησης για κάθε είσοδο του πίνακα δρομολόγησης).

Για τη διατήρηση της διαθεσιμότητας και της εφεδρείας, οι κόμβοι χρησιμοποιούν περιοδικούς φάρους (beacons), για να ανιχνεύσουν αποτυχίες κόμβων και εξερχόμενων συνδέσεων. Τέτοια γεγονότα πυροδοτούν διορθώσεις στο δίκτυο δρομολόγησης και εκκινούν αναδιανομή και αντιγραφή αναφορών σε θέσεις αντικειμένων. Επιπλέον, η διαδικασία διόρθωσης εντείνεται από την soft-state αναδημοσίευση των αναφορών σε αντικείμενα. Το Tapestry είναι πολύ αποδοτικό και διατηρεί κοντά ένα 100% ποσοστό επιτυχίας στη δρομολόγηση μηνυμάτων σε κόμβους και αντικείμενα.

Αρχιτεκτονική και υλοποίηση ενός κόμβου Tapestry

Στην παρακάτω εικόνα απεικονίζεται η λειτουργική διαστρωμάτωση ενός κόμβου Tapestry. Στην κορυφή φαίνονται οι εφαρμογές που έρχονται σε επαφή με το υπόλοιπο σύστημα μέσω του Tapestry API. Κάτω από αυτό βρίσκονται τα στρώματα δρομολόγησης (*router*) και δυναμικής διαχείρισης κόμβων (*dynamic node management*).Το πρώτο επεξεργάζεται μηνύματα δρομολόγησης και θέσης, ενώ το δεύτερο χειρίζεται την άφιξη και την αναχώρηση κόμβων στο δίκτυο. Αυτά τα δύο στρώματα επικοινωνούν μέσω του πίνακα δρομολόγησης. Στη βάση βρίσκονται τα στρώματα μεταφοράς (*transport*) και γειτονικών συνδέσεων (*neighbor links*), τα οποία μαζί παρέχουν ένα στρώμα ανταλλαγής μηνυμάτων μεταξύ των κόμβων.



Εικόνα 10: Αρχιτεκτονική επιμέρους τμημάτων του Tapestry. Τα μηνύματα κατευθύνονται προς τα πάνω από τα στρώματα φυσικού δικτύου και προς τα κάτω από τα στρώματα εφαρμογών. Κρίσιμο στρώμα για την επικοινωνία είναι αυτό του δρομολογητή. [36]

- Στρώμα μεταφοράς (Transport layer)

Το στρώμα μεταφοράς παρέχει την αφαίρεση των καναλιών επικοινωνίας ανάμεσα σε έναν υπερκείμενο κόμβο και σε έναν άλλο. Αντιστοιχεί στο στρώμα 4 της OSI αρχιτεκτονικής. Χρησιμοποιώντας τις δυνατότητες του εκάστοτε λειτουργικού συστήματος είναι δυνατές πολλές υλοποιήσεις καναλιών. Αυτή τη στιγμή υποστηρίζονται δύο υλοποιήσεις (TCP/IP και (UDP)/IP).

- Στρώμα γειτονικών συνδέσεων (Neighbor link layer)

Πάνω από το στρώμα μεταφοράς είναι το στρώμα γειτονικών συνδέσεων. Παρέχει ασφαλείς αλλά αναξιόπιστες υπηρεσίες δεδομενογραφήματος στα παραπάνω στρώματα συμπεριλαμβανομένου του τεμαχισμού και της συναρμολόγησης μεγάλων μηνυμάτων. Την πρώτη φορά που ένα υψηλότερο στρώμα επιθυμεί να επικοινωνήσει με έναν άλλο κόμβο πρέπει να δώσει στο στρώμα γειτονικών συνδέσεων τη φυσική διεύθυνση (IP διεύθυνση και πόρτα) του προορισμού. Εάν επιθυμείται ένα ασφαλές κανάλι, πρέπει να δώσει επίσης ένα δημόσιο κλειδί για τον απομακρυσμένο κόμβο. Το στρώμα γειτονικών συνδέσεων χρησιμοποιεί αυτή την πληροφορία, για να εγκαταστήσει μια σύνδεση με τον απομακρυσμένο κόμβο.
Οι συνδέσεις ανοίγονται υπό αίτηση ανώτερων επιπέδων του Tapestry. Για να αποφευχθεί η κατάχρηση πόρων του λειτουργικού συστήματος που δεν αφθονούν, όπως οι δείκτες σε αρχεία, το στρώμα γειτονικών συνδέσεων μπορεί να κλείνει περιοδικά μερικές συνδέσεις. Οι κλεισμένες συνδέσεις ανοίγονται υπό αίτηση.
Μια σημαντική λειτουργία αυτού του στρώματος είναι η συνεχής παρακολούθηση των συνδέσεων και η προσαρμογή. Παρέχει ανίχνευση λαθών μέσω soft-state μηνυμάτων *keep-alive* και εκτιμήσεις καθυστέρησης και ποσοστού απωλειών. Το στρώμα γειτονικών συνδέσεων ενημερώνει τα υψηλότερα στρώματα όποτε τα χαρακτηριστικά ενός συνδέσμου αλλάζουν σημαντικά.
Αυτό το στρώμα βελτιστοποιεί επίσης την επεξεργασία μηνυμάτων, καθώς αναλαμβάνει το parsing των επικεφαλίδων των μηνυμάτων και το deserialising μόνο του περιεχομένου των μηνυμάτων, όταν χρειάζεται. Τέλος, η πιστοποίηση κόμβου και

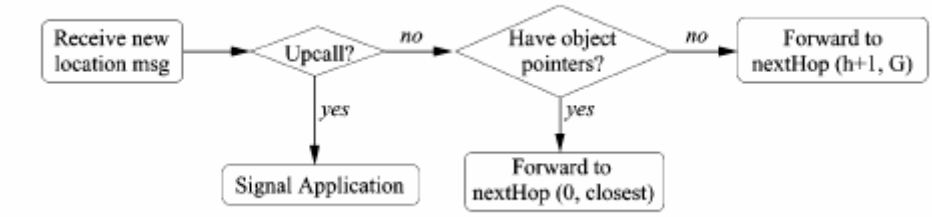
οι κώδικες πιστοποίησης μηνυμάτων (message authentication codes – MACs) μπορούν να ενσωματωθούν σε αυτό το στρώμα για επιπλέον ασφάλεια.

- Στρώμα δρομολογητή (Router layer)

Ενώ το στρώμα γειτονικών συνδέσεων παρέχεις βασικές δικτυακές υπηρεσίες, το στρώμα δρομολογητή παρέχει μία λειτουργικότητα μοναδική στο Tapestry. Σε αυτό το στρώμα περιλαμβάνονται ο πίνακας δρομολόγησης και οι δείκτες τοπικών αντικειμένων.

Όπως έχει ήδη αναφερθεί, το δίκτυο δρομολόγησης είναι μια λίστα από σύμφωνα με το πρόθεμα ταξινομημένους γείτονες, που αποθηκεύονται στον πίνακα δρομολόγησης. Ο δρομολογητής εξετάζει το GUID προορισμού των μηνυμάτων που λαμβάνει και καθορίζει το επόμενο βήμα τους χρησιμοποιώντας τον πίνακα και τους δείκτες τοπικών αντικειμένων. Τα μηνύματα περνάνε στη συνέχεια στο στρώμα γειτονικών συνδέσεων για παράδοση.

Στην Εικόνα 11 βλέπουμε το διάγραμμα ροής της διαδικασίας εντοπισμού ενός αντικειμένου. Τα μηνύματα φτάνουν από το στρώμα γειτονικών συνδέσεων στα αριστερά. Κάποια από τα μηνύματα πυροδοτούν περαιτέρω upcalls και βάζουν αμέσως σε λειτουργία τους χειριστές των upcalls. Διαφορετικά, οι δείκτες στα τοπικά αντικείμενα ελέγχονται μήπως υπάρξει ταίριασμα με το GUID που αναζητείται. Εάν πράγματι βρεθούν δύο GUIDs που να ταυτίζονται, το μήνυμα προωθείται στον κοντινότερο κόμβο από το σύνολο των δεικτών που ταιριάζουν με το αναζητούμενο GUID. Αλλιώς, το μήνυμα προωθείται στο επόμενο βήμα προς τη ρίζα.



Εικόνα 11: Διάγραμμα ροής επεξεργασίας μηνυμάτων. [36]

Πρέπει να σημειώσουμε ότι ο πίνακας δρομολόγησης και η βάση δεδομένων των δεικτών προς αντικείμενα μεταβάλλονται συνεχώς από το στρώμα δυναμικής διαχείρισης κόμβων και το στρώμα γειτονικών συνδέσεων. Επί παραδείγματι, λόγω συνεχών αλλαγών στις καθυστερήσεις συνδέσεων, το στρώμα γειτονικών συνδέσεων μπορεί να αναδιατάξει τις προτιμήσεις που έχουν ανατεθεί στους γείτονες που καταλαμβάνουν την ίδια εγγραφή στον πίνακα δρομολόγησης. Ομοίως, το στρώμα δυναμικής διαχείρισης κόμβων μπορεί να προσθέσει ή να αφαιρέσει δείκτες προς αντικείμενα μετά την άφιξη ή αναχώρηση γειτόνων.

Upcall Interface του Tapestry

Ενώ το DOLR API αποτελεί ένα ισχυρό interface εφαρμογών, άλλες λειτουργίες όπως το multicast απαιτούν περισσότερο έλεγχο πάνω στις λεπτομέρειες της δρομολόγησης και της αναζήτησης αντικειμένου. Για να είναι δυνατός αυτός ο επιπλέον έλεγχος, το Tapestry υποστηρίζει έναν εκτεταμένο **upcall μηχανισμό**. Μάλιστα, αναμένεται με την ωρίμανση της χρήσης των υπερκείμενων υποδομών να δημιουργηθεί η ανάγκη για προσαρμογή σε ένα σύνολο από καλά δοκιμασμένες και συχνά χρησιμοποιούμενες συμπεριφορές δρομολόγησης.

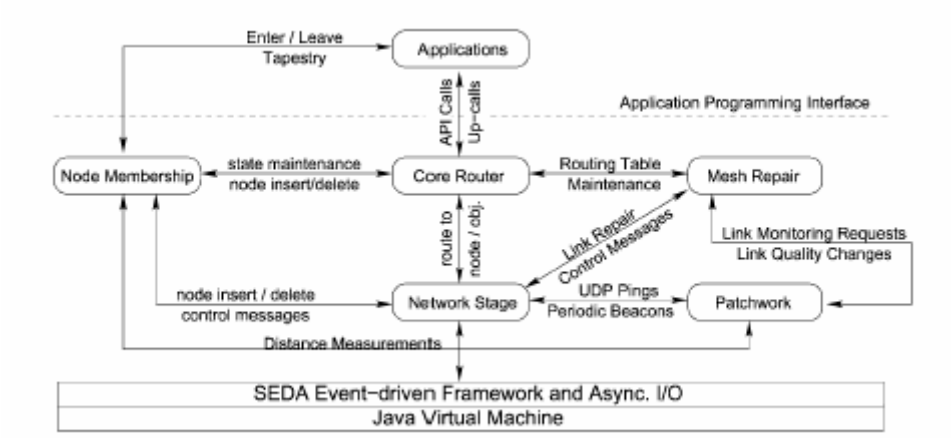
Η αλληλεπίδραση ανάμεσα στο Tapestry και στους χειριστές των εφαρμογών γίνεται μέσω τριών βασικών κλήσεων. Το G είναι ένα γενικευμένο κλειδί – έτσι, θα μπορούσε να είναι ένα nodeID $N_{id,}$ ή ένα GUID O_G.

1. DELIVER (G, A_{id}, Msg): επιφέρεται στα εισερχόμενα μηνύματα που προορίζονται για τον τοπικό κόμβο. Είναι ασύγχρονη κλήση. Η εφαρμογή μπορεί να παράγει περαιτέρω γεγονότα καλώντας τη ROUTE( ).
2. FORWARD (G, Aid, Msg): επιφέρεται σε εισερχόμενα μηνύματα που ενεργοποιούν τα upcalls. Η εφαρμογή πρέπει να καλέσει τη ROUTE(·), για να προωθηθεί και άλλο το μήνυμα.
3. ROUTE (G, Aid, Msg, NextHopNode): επιφέρεται από τον χειριστή εφαρμογής, για να προωθήσει ένα μήνυμα στον NextHopNode.

Επιπλέον interfaces δίνουν πρόσβαση στον πίνακα δρομολόγησης και στη βάση δεδομένων των δεικτών σε αντικείμενα. Όταν φτάνει ένα μήνυμα που ενεργοποιεί τα upcalls, το Tapestry στέλνει το μήνυμα στην εφαρμογή μέσω του FORWARD( ). Ο χειριστής είναι υπεύθυνος για την κλήση του ROUTE( ) με τον τελικό προορισμό. Στο τέλος, το Tapestry καλεί την DELIVER( ) σε μηνύματα που προορίζονται για τον τοπικό κόμβο και έτσι τελειώνει η δρομολόγηση.
Αυτό το upcall interface παρέχει επαρκή λειτουργικότητα, για να υλοποιηθεί – επί παραδείγματι- το Bayeux [17] σύστημα multicast. Τα μηνύματα μαρκάρονται, έτσι ώστε να πυροδοτούν upcalls σε κάθε βήμα και το Tapestry να «προκαλεί» την FORWARD( ) κλήση σε κάθε μήνυμα. Ο χειριστής Bayeux ελέγχει στη συνέχεια τη λίστα μελών, την ταξινομεί σε ομάδες και προωθεί ένα αντίγραφο του μηνύματος σε κάθε εξερχόμενη εγγραφή.

Υλοποίηση του Tapestry
Το Tapestry είναι υλοποιημένο σε Java και αποτελείται από περίπου 57.000 γραμμές κώδικα σε 255 αρχεία πηγής.
- Υλοποίηση ενός κόμβου
Το Tapestry είναι υλοποιημένο ως ένα σύστημα οδηγούμενο από γεγονότα με μεγάλη ρυθμαπόδοση και κλιμάκωση. Το σύστημα χρειάζεται ένα ασύγχρονο στρώμα I/O και ένα αποδοτικό μοντέλο για εσωτερική επικοινωνία και έλεγχο μεταξύ των επιμέρους τμημάτων. Προκειμένου να τηρηθούν αυτές οι απαιτήσεις, χρησιμοποιείται η SEDA (Staged Event-Driven Architecture) [37] αρχιτεκτονική εφαρμογών (βλ. παράγραφο 5.1). Στο SEDA, τα εσωτερικά επιμέρους τμήματα επικοινωνούν μέσω γεγονότων και ενός μοντέλου εγγραφής. Όπως φαίνεται στην παρακάτω εικόνα, αυτά τα επιμέρους τμήματα είναι τα *core router*, *node membership*, *mesh repair*, *patchwork* και *network*.

Το στάδιο *network* αντιστοιχεί σε ένα συνδυασμό του στρώματος γειτονικών συνδέσεων και τμημάτων του στρώματος μεταφοράς από τη γενική αρχιτεκτονική. Υλοποιεί κομμάτι της αφαίρεσης της επικοινωνίας των γειτόνων που δεν παρέχεται από το λειτουργικό σύστημα. Επίσης, είναι υπεύθυνο για την αποθήκευση και το πέρασμα των μηνυμάτων στα υψηλότερα στρώματα του συστήματος. Το στάδιο *network* αλληλεπιδρά στενά με το στάδιο *patchwork* για τη λειτουργία παρακολούθησης και μέτρησης ποσοστών λαθών και καθυστέρησης πληροφορίας για εγκατεστημένα κανάλια επικοινωνίας.



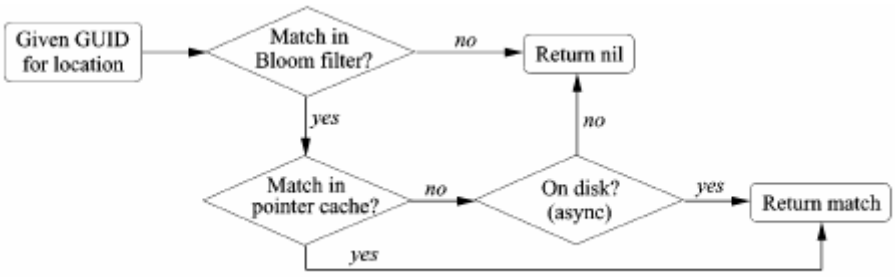
Εικόνα 12: Υλοποίηση του Tapestry. Το Tapestry είναι υλοποιημένο σαν μια σειρά από στάδια, τα οποία χρονοδρομολογούνται ανεξάρτητα και αλληλεπιδρούν μεταξύ τους περνώντας γεγονότα το ένα στο άλλο. [36]

Ο *core router* χρησιμοποιεί τους πίνακες δρομολόγησης και αναφοράς αντικειμένων, για να χειριστεί μηνύματα που οδηγούνται από την εφαρμογή, συμπεριλαμβανομένου της δημοσίευσης αντικειμένου, του εντοπισμού αντικειμένου και της δρομολόγησης μηνυμάτων σε κόμβους-προορισμούς. Αυτό το στάδιο αλληλεπιδρά επίσης με το στρώμα εφαρμογής μέσω του interface της εφαρμογής και των upcalls. Ο *core router* είναι στο κρίσιμο μονοπάτι όλων των μηνυμάτων που εισέρχονται και εξέρχονται από το σύστημα. Αυτή η υλοποίηση είναι αρκετά αποδοτική, ωστόσο οι αλγόριθμοι του Tapestry επιδέχονται βελτιστοποίησης για τη γρηγορότερη διαδρομή. Έτσι, μπορεί να αυξηθεί κι άλλο η ρυθμαπόδοση και να μειωθεί η καθυστέρηση.
Υπάρχουν δύο δυναμικά στάδια, τα οποία υποστηρίζουν το δρομολογητή: το στάδιο *deterministic node membership* και το στάδιο *soft-state mesh repair*. Και τα δύο χρησιμοποιούν τον πίνακα δρομολόγησης και τον πίνακα αναφορών σε αντικείμενα. Το στάδιο *node membership* είναι υπεύθυνο για το χειρισμό της εισαγωγής νέων κόμβων στο δίκτυο Tapestry και της αποχώρησης – ακούσια ή εκούσια – των κόμβων. Είναι δηλαδή υπεύθυνο για την εκκίνηση ενός νέου κόμβου με σωστό (συνεπή και ορθό σε σχέση με την τοπικότητα δικτύου) πίνακα δρομολόγησης.
Εν αντιθέσει, το στάδιο *mesh repair* είναι υπεύθυνο για την προσαρμογή του Tapestry, καθώς αλλάζει το δικτυακό περιβάλλον. Αυτό περιλαμβάνει αντίδραση σε αλλαγές της ποιότητας των δικτυακών συνδέσεων, προσαρμογή σε καταστροφική απώλεια γειτόνων, και ενημέρωση του πίνακα δρομολόγησης, καθώς αλλάζει η καθυστέρηση δικτύου. Η διαδικασία διόρθωσης επίσης ανακατανέμει τους δείκτες των αντικειμένων, όσο αλλάζουν οι συνθήκες δικτύου. Η διαδικασία διόρθωσης μπορεί να θεωρηθεί σαν μια οδηγούμενη από τα γεγονότα προσαρμογή της κατάστασης συνδυαζόμενη με συνεχή αποκατάσταση της ήδη υπάρχουσας πληροφορίας για τη δρομολόγηση και τη θέση των αντικειμένων. Έτσι, παρέχεται προσαρμογή στα περισσότερα λάθη και σε εξελισσόμενες αλλαγές, ενώ εξασφαλίζεται αποκατάσταση από τα περισσότερα αινιγματικά προβλήματα.
Τέλος, το στάδιο *patchwork* χρησιμοποιεί soft-state φάρους, για να εξετάσει τις εξερχόμενες συνδέσεις ως προς την αξιοπιστία τους και την απόδοσή τους. Κατά αυτόν τον τρόπο επιτρέπεται στο Tapestry να ανταποκριθεί σε αποτυχίες και αλλαγές στην τοπολογία δικτύου. Επιπλέον, υποστηρίζει ασύγχρονες μετρήσεις καθυστέρησης προς άλλους κόμβους. Είναι στενά συνδεδεμένο με το δίκτυο, καθώς χρησιμοποιεί

πολλούς μηχανισμούς μεταφοράς του λειτουργικού συστήματος (όπως channel acknowledgments) όπου είναι δυνατό.
Έχουν υλοποιηθεί και τα βασιζόμενα σε TCP και τα βασιζόμενα σε UDP στρώματα δικτύου. Το TCP, ως γνωστόν, υποστηρίζει και έλεγχο ροής και έλεγχο συμφόρησης, και συμπεριφέρεται δίκαια υπό την παρουσία πολλών ροών. Τα μειονεκτήματά του είναι οι μεγάλοι χρόνοι εγκατάστασης και απεγκατάστασης συνδέσεων, η μη ιδανική χρησιμοποίηση του διαθέσιμου εύρους και η κατανάλωση δεικτών σε αρχεία που αποτελεί περιορισμένο πόρο. Από την άλλη, τα μηνύματα UDP μπορούν να σταλούν με μικρές επικεφαλίδες και μπορούν να χρησιμοποιήσουν περισσότερο από το διαθέσιμο εύρος σε μία σύνδεση δικτύου. Ωστόσο, το UDP δεν υποστηρίζει έλεγχο ροής και έλεγχο συμφόρησης, και μπορεί να καταναλώσει κομμάτι εύρους που δεν του αναλογεί προκαλώντας συμφόρηση, εάν χρησιμοποιείται σε μεγάλη έκταση. Για αυτό το λόγο, το στρώμα UDP στο Tapestry χρησιμοποιεί έλεγχο συμφόρησης παρόμοιο με του TCP. Έχοντας μειονεκτήματα και πλεονεκτήματα, κανένα από τα δύο πρωτόκολλα δεν μπορεί να θεωρηθεί καλύτερο. Ωστόσο, το γεγονός ότι το στρώμα UDP δεν καταναλώνει δείκτες σε αρχεία φαίνεται να είναι ένα σημαντικό πλεονέκτημα για τη χρήση του.
- Εικονικότητα κόμβου
Σε ένα φυσικό μηχάνημα μπορούν να τοποθετηθούν πολλαπλά στιγμιότυπα κόμβων Tapestry και έτσι γίνεται δυνατή μία μεγαλύτερη ποικιλία από πειράματα στο δίκτυο Tapestry. Για την ελαχιστοποίηση της μνήμης και της υπολογιστικής επικεφαλίδας κατά την αύξηση του αριθμού των στιγμιότυπων σε ένα φυσικό μηχάνημα τρέχουμε όλα τα στιγμιότυπα κόμβων σε ένα JVM (Java Virtual Machine). Αυτή η τεχνική επιτρέπει την εκτέλεση πολλών ταυτόχρονων στιγμιότυπων του Tapestry σε ένα μόνο μηχάνημα.
Όλοι οι εικονικοί κόμβοι στο ίδιο φυσικό μηχάνημα χρησιμοποιούν ένα και μόνο JVM νήμα εκτέλεσης, πράγμα που σημαίνει ότι μόνο ένας εικονικός κόμβος εκτελεί κάθε φορά. Τα εικονικά στιγμιότυπα μοιράζονται μονάχα κώδικα. Κάθε στιγμιότυπο διατηρεί τα δικά του αποκλειστικά δεδομένα. Μια παρενέργεια των εικονικών κόμβων είναι η καθυστέρηση που εισάγεται από την Κεντρική Μονάδα Επεξεργασίας για τη χρονοδρομολόγηση μεταξύ των κόμβων. Κατά τη διάρκεια περιόδων υψηλού φόρτου για την ΚΜΕ, οι καθυστερήσεις δρομολόγησης μπορούν να έχουν σημαντική επίπτωση στα αποτελέσματα επίδοσης και αυξάνουν τεχνητά την καθυστέρηση δρομολόγησης και εντοπισμού. Αυτή η κατάσταση επιδεινώνεται από τις αφύσικα μικρές αποστάσεις δικτύου μεταξύ των κόμβων στο ίδιο μηχάνημα. Αυτά τα στιγμιότυπα κόμβων μπορούν να ανταλλάσσουν μηνύματα σε λιγότερο από 10μs μετατρέποντας το χρόνο για την επεξεργασία επικεφαλίδας και την καθυστέρηση χρονοδρομολόγησης συγκριτικά πολύ μεγάλο για κάθε υπερκείμενο δίκτυο.

Αύξηση της επίδοσης
Η υπάρχουσα υλοποίηση μπορεί να χειριστεί περισσότερα από 7000μηνύματα/δευτερόλεπτο. Ωστόσο, μια υλοποίηση με εμπορικό προσανατολισμό και την ανάλογη ποιότητα θα μπορούσε να αποδίδει καλύτερα. Σε σχέση με τη βελτιστοποίηση παρατηρήθηκε το εξής σημαντικό γεγονός: παρόλη την προηγμένη λειτουργικότητα του DOLR API, το κρίσιμο μονοπάτι της δρομολόγησης μηνυμάτων επιδέχεται βελτιστοποίηση προς επίτευξη υψηλότερης απόδοσης.
Το κρίσιμο μονοπάτι της δρομολόγησης αποτελείται από δύο ξεχωριστά κομμάτια. Το πιο απλό κομμάτι – υπολογισμός του NEXTHOP – είναι παρόμοιο σε λειτουργικότητα με αυτό των δρομολογητών υλικού: γρήγορη αναζήτηση σε πίνακα. Για ένα δίκτυο ενός εκατομμυρίου κόμβων με βάση 16 (β=16), ο πίνακας

δρομολόγησης με πληροφορία GUID/IP διεύθυνση για κάθε εγγραφή θα είχε μέγεθος μικρότερο από 10kB –μέγεθος πολύ μικρότερο από την προσωρινή μνήμη (cache) της ΚΜΕ. Αποδεικνύεται απλά ότι τα περισσότερα βήματα δικτύου περιλαμβάνουν μία αναζήτηση, ενώ τα τελευταία δύο βήματα απαιτούν το πολύ β/2 = 8 αναζητήσεις.
Συνεπώς, είναι το δεύτερο κομμάτι της δρομολόγησης DOLR – γρήγορη αναζήτηση δεικτών – που αποτελεί τη μεγαλύτερη πρόκληση για δρομολόγηση υψηλής ρυθμαπόδοσης. Κάθε φορά που μία αίτηση **R**OUTE**TOO**BJECT περνάει από έναν δρομολογητή πρέπει αυτός να «περάσει» τον πίνακα δεικτών του. Εάν όλοι οι δείκτες χωρούν στη μνήμη, μια απλή αναζήτηση πίνακα κατακερματισμού δίνει O(1) πολυπλοκότητα για αυτή την αναζήτηση. Ωστόσο, ο αριθμός των δεικτών μπορεί να είναι πολύ μεγάλος σε μία ευρείας κλίμακας εφαρμογή και επιπλέον, οι πόροι γρήγορης μνήμης ενός δρομολογητή υλικού μπορεί να είναι μικρότεροι από τους σταθμούς εργασίας τελευταίας τεχνολογίας.
Εξετάζοντας αυτό το θέμα παρατηρήθηκε ότι τα περισσότερα βήματα δρομολόγησης λαμβάνουν αρνητικά αποτελέσματα αναζήτησης (μόνο ένα λαμβάνει θετικό αποτέλεσμα). Έτσι, η ιδέα είναι το φτιάξιμο ενός **φίλτρου Bloom** [38] πάνω από το σύνολο των δεικτών. Ένα φίλτρο Bloom είναι μια μη ακριβής αναπαράσταση ενός συνόλου, η οποία μπορεί να ανιχνεύσει σχετικά γρήγορα την απουσία ενός μέλους ενός συνόλου. Το μέγεθος ενός φίλτρου Bloom πρέπει να προσαρμοστεί, ώστε να αποφευχθούν τα πολλά "false-positives". Επί παραδείγματι, ένα λογικό μέγεθος ενός φίλτρου Bloom πάνω από P δείκτες είναι περίπου 10P bits. Υποθέτοντας ότι οι δείκτες (με όλη την πληροφορία τους) είναι 100 bytes, το ίχνος στη μνήμη ενός φίλτρου Bloom μπορεί να είναι δύο τάξεις μεγέθους μικρότερο από το συνολικό μέγεθος των δεικτών.
Συνεπώς, προτείνεται η βελτίωση στην αναζήτηση δεικτών, όπως φαίνεται στην Εικόνα 13. Μαζί με ένα φίλτρο Bloom περιλαμβάνεται και μία προσωρινή μνήμη από ενεργούς δείκτες τόσο μεγάλη ώστε να χωράει στη μνήμη του δρομολογητή. Το κύριο σημείο της βελτιωμένης αναζήτησης δείκτη είναι ο χωρισμός της διαδικασίας αναζήτησης σε έναν γρήγορο αρνητικό έλεγχο, ακολουθούμενο από έναν γρήγορο θετικό έλεγχο (για αντικείμενα που είναι ενεργά), ακολουθούμενο από κάτι πιο αργό. Παρόλο που στο διάγραμμα ροής αναφέρεται ο δίσκος, αντί για δίσκο μπορεί να έχουμε μνήμη πάνω σε έναν επεξεργαστή που προσπελάζεται από το δρομολογητή υλικού, όταν όλα τα άλλα αποτυγχάνουν.



Εικόνα 13: Βελτιστοποιημένη αναζήτηση δείκτη. [36]

Tapestry και εφαρμογές
Έχοντας εξετάσει την υλοποίηση και τη συμπεριφορά του Tapestry, καταλήγουμε στο ότι το Tapestry παρέχει ένα σταθερό interface υπό μια ποικιλία συνθηκών δικτύου. Μένει να δούμε πως το Tapestry μπορεί να σταθεί απέναντι στις προκλήσεις που αντιμετωπίζουν οι μεγάλης κλίμακας εφαρμογές.

Με την αυξανόμενη χρησιμοποίηση του Διαδικτύου, οι μηχανικοί εφαρμογών έχουν αρχίσει να εστιάζουν σε εφαρμογές μεγάλης κλίμακας που εκμεταλλεύονται κοινούς πόρους του δικτύου. Παραδείγματα αποτελούν το multicast επιπέδου εφαρμογής, τα συστήματα αποθήκευσης μεγάλης κλίμακας, και τα συστήματα ανακατεύθυνσης κίνησης για ανθεκτικότητα και ασφάλεια. Αυτές οι εφαρμογές μοιράζονται νέες προκλήσεις, όταν πλέον μιλάμε για μεγάλες περιοχές εφαρμογής: θα είναι πιο δύσκολο για τους χρήστες να εντοπίζουν κοντινούς πόρους, καθώς μεγαλώνει το δίκτυο, και η εξάρτηση από περισσότερα κατανεμημένα επιμέρους τμήματα σημαίνει μικρότερος μέσος χρόνος μεταξύ αποτυχιών (mean time between failures – MTBF) για το σύστημα. Επί παραδείγματι, ένας χρήστης ενός συστήματος ανταλλαγής αρχείων θέλει να εντοπίσει και να ανακτήσει ένα κοντινό αντίγραφο ενός αρχείου αποφεύγοντας αποτυχίες εξυπηρετητή ή δικτύου.
Η ασφάλεια είναι επίσης ένα σημαντικό θέμα. Η επίθεση Sybil [39] είναι μια επίθεση όπου ένας χρήστης παίρνει έναν μεγάλο αριθμό από IDs, για να αυξήσει τις επιθέσεις συνωμοσίας (collusion attacks). Το Tapestry αντιμετωπίζει την παραπάνω επίθεση χρησιμοποιώντας μία έμπιστη υποδομή δημόσιου κλειδιού (public-key infrastructure – PKI) για την ανάθεση των nodeIDs. Για να μειωθεί η ζημιά από ελεγχόμενους κόμβους, οι κόμβοι του Tapestry μπορούν να δουλεύουν σε ζευγάρια δρομολογώντας μηνύματα μεταξύ τους μέσω γειτόνων και επαληθεύοντας στη συνέχεια τη διαδρομή που ακολουθήθηκε. Τέλος, το Tapestry υποστηρίζει τη χρήση MACs, για να διατηρήσει την ακεραιότητα της υπερκείμενης κίνησης.
Το Tapestry, εκτός του ότι υποστηρίζει αποδοτική δρομολόγηση των μηνυμάτων σε ονομαζόμενα αντικείμενα ή τελικά σημεία στο δίκτυο, αυξομειώνεται λογαριθμικά με το μέγεθος του δικτύου σε πληροφορία δρομολόγησης ανά κόμβο και σε αναμενόμενο αριθμό υπερκείμενων βημάτων σε μία διαδρομή. Επιπλέον, εμφανίζει ανθεκτικότητα σε αποτυχίες εξυπηρετητών και αποτυχίες δικτύου επιτρέποντας στα μηνύματα να δρομολογούνται γύρω τους σε εφεδρικές διαδρομές. Οι εφαρμογές μπορούν να πετύχουν πρόσθετη αντοχή αντιγράφοντας δεδομένα σε πολλαπλούς εξυπηρετητές και περιμένοντας από το Tapestry να κατευθύνει αιτήσεις πελατών σε κοντινά αντίγραφα.
Μια ποικιλία από διαφορετικές εφαρμογές έχουν σχεδιαστεί, υλοποιηθεί και λειτουργήσει πάνω στο Tapestry. Το OceanStore [20] είναι μία μεγάλης κλίμακας υπηρεσία αποθήκευσης υψηλής διαθεσιμότητας, η οποία έχει δοκιμαστεί στο PlanetLab Testbed. Οι εξυπηρετητές OceanStore χρησιμοποιούν το Tapestry, για να διασκορπίσουν αποδοτικά κωδικοποιημένα τμήματα αρχείων (blocks). Οι πελάτες μπορούν να εντοπίσουν γρήγορα και να ανακτήσουν κοντινά τμήματα αρχείων από το ID τους ανεξάρτητα από αποτυχίες εξυπηρετητών ή αποτυχίες δικτύου. Άλλες εφαρμογές είναι το Mnemosyne [19], ένα σύστημα αρχείων, το Bayeux [17], ένα αποδοτικό αυτο-διοργανούμενο σύστημα multicast επιπέδου εφαρμογής, και το SpamWatch [24], ένα αποκεντρωμένο σύστημα που φιλτράρει το "spamming" και χρησιμοποιεί μία μηχανή αναζήτησης ομοιότητας υλοποιημένη στο Tapestry.

Συμπερασματικά

Η αρχιτεκτονική εντοπισμού και δρομολόγησης του Tapestry είναι μία αυτο-διοργανούμενη, κλιμακούμενη σε μέγεθος και εύρωστη υποδομή ευρείας κλίμακας που δρομολογεί αποτελεσματικά αιτήσεις περιεχομένου υπό την παρουσία υψηλού φορτίου και λαθών δικτύου ή κόμβων. Ένα υπερκείμενο δίκτυο Tapestry μπορεί να δομηθεί αποτελεσματικά, για να υποστηρίξει δυναμικά δίκτυα χρησιμοποιώντας κατανεμημένους αλγορίθμους. Ενώ το Tapestry είναι παρόμοιο με την κατανεμημένη τεχνική αναζήτησης Plaxton [40], έχει πρόσθετους μηχανισμούς που

εκμεταλλεύονται την soft-state πληροφορία και παρέχουν αυτόματη οργάνωση, ευρωστία, κλιμάκωση σε μέγεθος, δυναμική προσαρμογή, και ικανοποιητική υποβάθμιση/ μείωση της απόδοσης παρουσία αποτυχιών και υψηλού φορτίου.
Το Tapestry αποτελεί την πλέον κατάλληλη λύση για δυναμικά, ευρείας κλίμακας συστήματα ονοματοδοσίας αντικειμένου και δρομολόγησης μηνυμάτων, όταν αυτά τα συστήματα πρέπει να παραδώσουν μηνύματα στο πλησιέστερο αντίγραφο των αντικειμένων ή των υπηρεσιών με έναν τρόπο ανεξάρτητο θέσης, χρησιμοποιώντας μόνο τις από σημείο σε σημείο συνδέσεις και χωρίς συγκεντρωμένες υπηρεσίες. Το Tapestry το πετυχαίνει αυτό χρησιμοποιώντας την τυχαιότητα, για να επιτύχει και την κατανομή φορτίου και την τοπικότητα της δρομολόγησης. [36]

3.2.2.5 Pastry

Ένα από τα μεγαλύτερα προβλήματα στις peer-to-peer εφαρμογές μεγάλης κλίμακας είναι η εύρεση αποδοτικών αλγορίθμων για τον εντοπισμό αντικειμένων και τη δρομολόγηση μέσα στο δίκτυο. Το Pastry είναι ένα γενικό σχήμα peer-to-peer εντοπισμού αντικειμένων και δρομολόγησης βασισμένο σε ένα αυτο-διοργανούμενο υπερκείμενο δίκτυο κόμβων που συνδέονται μέσω του Διαδικτύου. Είναι πλήρως αποκεντρωμένο, ανθεκτικό σε λάθη, κλιμακούμενο σε μέγεθος και αξιόπιστο. Επιπλέον, διαθέτει καλές ιδιότητες όσον αφορά την τοπικότητα της δρομολόγησης.
Το Pastry προορίζεται για γενικό υπόστρωμα δόμησης πολλών και διαφορετικών peer-to-peer εφαρμογών Διαδικτύου, όπως η εκτεταμένη ανταλλαγή και αποθήκευση αρχείων, η επικοινωνία ομάδων και τα συστήματα ονοματοδοσίας. Πολλές εφαρμογές έχουν δομηθεί πάνω στο Pastry μέχρι στιγμής, όπως το PAST [21,41], μια υπηρεσία συνεχούς αποθήκευσης και το SCRIBE [16], ένα κλιμακούμενο σύστημα δημοσίευσης / εγγραφής.
Το Pastry παρέχει την ακόλουθη λειτουργικότητα: κάθε κόμβος στο δίκτυο Pastry έχει ένα μοναδικό αναγνωριστικό (*nodeId*). Με ένα μήνυμα και ένα κλειδί ένας Pastry κόμβος δρομολογεί αποδοτικά το μήνυμα στον κόμβο με nodeId που είναι αριθμητικά πλησιέστερα στο κλειδί σε σχέση με όλους τους άλλους ζωντανούς κόμβους Pastry. Ο αναμενόμενος αριθμός βημάτων δρομολόγησης είναι **O (log N)**, όπου N ο αριθμός των κόμβων Pastry στο δίκτυο. Από κάθε κόμβο Pastry που περνάει ένα μήνυμα κατά τη διαδρομή του, ενημερώνεται η εφαρμογή και ενδέχεται να κάνει -ανάλογα με την εφαρμογή- υπολογισμούς σχετικά με το μήνυμα.
Το Pastry λαμβάνει υπόψη του την τοπικότητα δικτύου: προσπαθεί να ελαχιστοποιήσει τις αποστάσεις που ταξιδεύουν τα μηνύματα σύμφωνα με ένα κλιμακούμενο μέτρο εγγύτητας όπως ο αριθμός των IP βημάτων δρομολόγησης. Κάθε κόμβος Pastry είναι ενήμερος για τους άμεσους γείτονές του στο χώρο των nodeIds και πληροφορεί τις εφαρμογές για αφίξεις νέων κόμβων, αποτυχίες και ανακάμψεις κόμβων. Επειδή τα nodeIds ανατίθενται τυχαία, είναι πολύ πιθανό μια ομάδα κόμβων με γειτονικά αναγνωριστικά να βρίσκονται αρκετά μακριά γεωγραφικά. Οι εφαρμογές μπορούν να το εκμεταλλευτούν αυτό, καθώς το Pastry μπορεί να δρομολογήσει σε έναν από k κόμβους που είναι αριθμητικά κοντύτερα στο κλειδί. Μια ευρηματική τεχνική διασφαλίζει ότι ανάμεσα στο σύνολο των k κόμβων με τα κοντινότερα nodeIds στο κλειδί το μήνυμα μάλλον θα φτάσει πρώτο στον κόμβο που είναι «κοντύτερα» (όσον αφορά το μέτρο εγγύτητας) στον κόμβο από τον οποίο προέρχεται το μήνυμα.
Οι εφαρμογές χρησιμοποιούν αυτές τις ιδιότητες με διαφορετικούς τρόπους. Το **PAST** [21,41], για παράδειγμα, χρησιμοποιεί ένα fileId (ο κατακερματισμός του ονόματος και του ιδιοκτήτη του αρχείου) ως κλειδί για το αρχείο. Αντίγραφα του

αρχείου αποθηκεύονται σε k κόμβους Pastry με nodeIds αριθμητικά πλησιέστερα στο fileId. Το αρχείο μπορεί να αναζητηθεί στέλνοντας ένα μήνυμα στο δίκτυο Pastry και χρησιμοποιώντας το fileId σαν κλειδί. Εξ ορισμού η αναζήτηση θα καταλήξει σε έναν κόμβο που αποθηκεύει το αρχείο, αρκεί κάποιος από τους k κόμβους να είναι ζωντανός. Επιπλέον, αναμένεται το μήνυμα να φτάσει πρώτα στον κόμβο που είναι κοντύτερα στον πελάτη. Αυτός ο κόμβος θα παραδώσει το αρχείο και το ταξίδι του μηνύματος τελειώνει. Οι μηχανισμοί ενημέρωσης του Pastry επιτρέπουν στο PAST να διατηρεί αντίγραφα ενός αρχείου στους k πλησιέστερους κόμβους στο κλειδί ανεξάρτητα από αποτυχίες και αφίξεις κόμβων και χρησιμοποιώντας μόνο τοπικό συντονισμό μεταξύ κόμβων με κοντινά nodeIds.
Στην άλλη τυπική εφαρμογή του Pastry, στο **SCRIBE** [16] σύστημα δημοσίευσης / εγγραφής, μια λίστα συνδρομητών αποθηκεύεται σε έναν κόμβο με nodeId αριθμητικά πλησιέστερα στο topicId του θέματος, όπου topicId είναι ο κατακερματισμός του ονόματος του θέματος. Αυτός ο κόμβος αποτελεί σημείο συνάντησης για εκδότες και συνδρομητές. Οι συνδρομητές στέλνουν ένα μήνυμα στο δίκτυο Pastry χρησιμοποιώντας το topicId σαν κλειδί και η εγγραφή καταγράφεται σε κάθε κόμβο κατά μήκος της διαδρομής. Ένας εκδότης στέλνει δεδομένα στο σημείο-κόμβο συνάντησης μέσω του Pastry χρησιμοποιώντας πάλι το topicId σαν κλειδί. Το σημείο συνάντησης προωθεί τα δεδομένα κατά μήκος του multicast δέντρου που δημιουργείται από τις αντίστροφες διαδρομές από το σημείο-κόμβο συνάντησης προς όλους τους συνδρομητές.

Σχεδιασμός του Pastry
Το σύστημα Pastry είναι ένα αυτο-διοργανούμενο υπερκείμενο δίκτυο κόμβων όπου κάθε κόμβος δρομολογεί αιτήσεις πελατών και αλληλεπιδρά με τοπικά στιγμιότυπα μίας ή περισσοτέρων εφαρμογών. Οποιοσδήποτε υπολογιστής είναι συνδεδεμένος στο Διαδίκτυο και τρέχει λογισμικό κόμβου Pastry μπορεί να δράσει σαν κόμβος Pastry και υπόκειται μόνο στις πολιτικές ασφαλείας της εφαρμογής.
Σε κάθε κόμβο στο peer-to-peer υπερκείμενο δίκτυο Pastry ανατίθεται ένα αναγνωριστικό των 128 bits (*nodeId*). Το nodeId χρησιμοποιείται για να προσδιορίσει τη θέση του κόμβου σε έναν κυκλικό χώρο των nodeIds που κυμαίνεται από το 0 έως το 2^{128-1}. Το nodeId ανατίθεται τυχαία όταν ένας κόμβος προσχωρεί στο σύστημα. Τα nodeIds παράγονται έτσι ώστε το προκύπτον σύνολο των nodeIds να είναι ομοιόμορφα κατανεμημένο στο χώρο των nodeIds. Επί παραδείγματι, τα nodeIds θα μπορούσαν να παράγονται υπολογίζοντας τον κρυπτογραφημένο κατακερματισμό του δημοσίου κλειδιού ενός κόμβου ή της IP διεύθυνσής του. Σαν αποτέλεσμα της τυχαίας ανάθεσης των nodeIds, με πολύ μεγάλη πιθανότητα, οι κόμβοι με διπλανά nodeIds διαφοροποιούνται σε γεωγραφική θέση, σε ιδιοκτήτη και σε δικαιοδοσία.
Εάν έχουμε ένα δίκτυο με N κόμβους, το Pastry μπορεί να δρομολογήσει στον αριθμητικά πλησιέστερο κόμβο σε δεδομένο κλειδί σε λιγότερα από $\lceil \log_{2^b} N \rceil$ βήματα υπό κανονικές συνθήκες (b είναι μια παράμετρος ρύθμισης με τυπική τιμή 4). Παρόλες τις ταυτόχρονες αποτυχίες κόμβων η τελική παράδοση είναι εγγυημένη, εάν δεν «πέσουν» ταυτόχρονα $\lfloor |L| / 2 \rfloor$ κόμβοι με διπλανά nodeIds ($|L|$ είναι μια παράμετρος ρύθμισης με τυπική τιμή 16 ή 32).
Για να εξυπηρετηθεί η δρομολόγηση, τα nodeIds και τα κλειδιά θεωρούνται ακολουθίες ψηφίων με βάση 2^b. Το Pastry δρομολογεί μηνύματα στον κόμβο του οποίου το nodeId είναι αριθμητικά κοντύτερα στο δεδομένο κλειδί. Αυτό επιτυγχάνεται ως εξής: σε κάθε βήμα δρομολόγησης, ένας κόμβος προωθεί κανονικά το μήνυμα σε έναν κόμβο του οποίου το nodeId μοιράζεται με το κλειδί ένα πρόθεμα το οποίο είναι τουλάχιστον ένα ψηφίο (ή b bits) μακρύτερο από το πρόθεμα που

μοιράζεται το κλειδί με τον τοπικό κόμβο. Εάν δεν υπάρχει τέτοιος κόμβος, το μήνυμα προωθείται στον κόμβο του οποίου το nodeId μοιράζεται ένα πρόθεμα με το κλειδί τόσο μακρύ όσο και ο τοπικός κόμβος, αλλά είναι αριθμητικά κοντύτερα στο κλειδί από ότι το nodeId του τοπικού κόμβου. Για να υποστηριχθεί αυτή η διαδικασία δρομολόγησης, κάθε κόμβος διατηρεί κάποια πληροφορία δρομολόγησης

Πληροφορία ανά κόμβο στο Pastry

Κάθε κόμβος Pastry διατηρεί έναν πίνακα δρομολόγησης (*routing table*), ένα σύνολο γειτονιάς (*neighborhood set*) και ένα σύνολο φύλλων (*leaf set*).

- Ο πίνακας δρομολόγησης R ενός κόμβου οργανώνεται σε $\lceil \log_{2^b} N \rceil$ γραμμές με $2^b - 1$ εγγραφές η κάθε μία. Κάθε μία από τις $2^b - 1$ εγγραφές στη γραμμή n του πίνακα δρομολόγησης αναφέρεται στον κόμβο με nodeId που μοιράζεται τα πρώτα n ψηφία με το nodeId του τοπικού κόμβου, αλλά το n+1 –ιοστό ψηφίο του έχει μία από τις $2^b - 1$ πιθανές τιμές (εκτός από το n+1 –ιοστό ψηφίο του τοπικού κόμβου).

  Κάθε εγγραφή στον πίνακα δρομολόγησης περιέχει την IP διεύθυνση ενός κόμβου του οποίου το nodeId έχει το κατάλληλο πρόθεμα. Στην πράξη, επιλέγεται ένας κόμβος που είναι κοντά στον τοπικό κόμβο σύμφωνα με ένα μέτρο εγγύτητας. Αυτή η επιλογή παρέχει καλές ιδιότητες τοπικότητας. Εάν κανένας κόμβος δεν είναι γνωστός με το κατάλληλο nodeId, τότε η είσοδος του πίνακα δρομολόγησης παραμένει άδεια. Η ομοιόμορφη κατανομή των nodeIds εξασφαλίζει έναν κανονικό πληθυσμό μέσα στον χώρο των nodeIds. Έτσι, κατά μέσο όρο μόνο $\lceil \log_{2^b} N \rceil$ γραμμές είναι κατειλημμένες στον πίνακα δρομολόγησης.

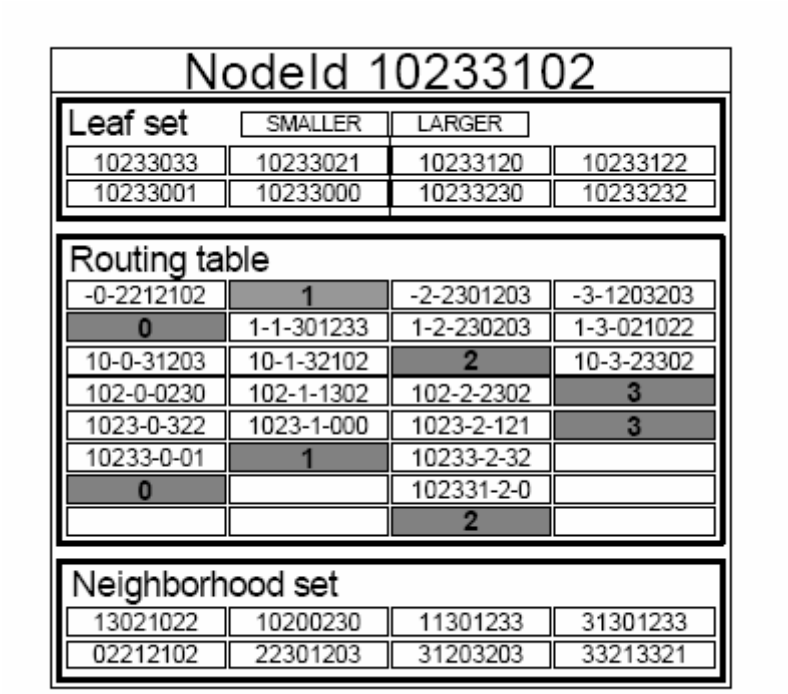
  Η επιλογή του b περιλαμβάνει επιλογή ανάμεσα στο μέγεθος του κατειλημμένου ποσοστού του πίνακα δρομολόγησης (περίπου $\lceil \log_{2^b} N \rceil \times (2^b - 1)$ εγγραφές) και στο μέγιστο αριθμό των βημάτων που απαιτούνται για τη δρομολόγηση ανάμεσα σε δύο οποιουσδήποτε κόμβους ($\lceil \log_{2^b} N \rceil$). Με b = 4 και 10^6 κόμβους, ο πίνακας δρομολόγησης περιέχει περίπου 75 εγγραφές και ο αναμενόμενος αριθμός των βημάτων για δρομολόγηση είναι 4, ενώ με 10^9 κόμβους ο πίνακας δρομολόγησης περιέχει περίπου 105 εγγραφές και ο αναμενόμενος αριθμός των βημάτων για δρομολόγηση είναι 7.
- Το σύνολο γειτονιάς M περιέχει τα nodeIds και τις IP διευθύνσεις των |M| κόμβων που είναι κοντύτερα (σύμφωνα με το μέτρο εγγύτητας) στον τοπικό κόμβο. Το σύνολο γειτονιάς δε χρησιμοποιείται κανονικά στα μηνύματα δρομολόγησης. Είναι χρήσιμο για τη διατήρηση των ιδιοτήτων τοπικότητας.
- Το σύνολο φύλλων L είναι το σύνολο των κόμβων με τα |L| / 2 αριθμητικά αμέσως μεγαλύτερα nodeIds, και τα |L| / 2 αριθμητικά αμέσως μικρότερα nodeIds σε σχέση πάντα με το nodeId του τοπικού κόμβου. Το σύνολο φύλλων χρησιμοποιείται κατά τη δρομολόγηση μηνυμάτων. Τυπικές τιμές του |L| και του |M| είναι 2^b ή 2×2^b.

Στην παρακάτω εικόνα φαίνεται η πληροφορία ενός υποθετικού κόμβου Pastry με nodeId 10233102 (βάση 4) σε ένα σύστημα που χρησιμοποιεί nodeIds των 16 bits και b = 2.

Δρομολόγηση στο Pastry

Όταν ένας κόμβος έχει ένα μήνυμα για αποστολή, πρώτα ελέγχει να δει εάν το κλειδί πέφτει στην περιοχή των nodeIds που καλύπτεται από το σύνολο φύλλων του. Εάν αυτό ισχύει, το μήνυμα προωθείται αμέσως στον κόμβο-προορισμό που θα είναι ο

κόμβος στο σύνολο φύλλων με nodeId πλησιέστερα στο κλειδί (πιθανότατα ο τοπικός κόμβος).
Εάν το κλειδί δεν ανήκει στην παραπάνω περιοχή, τότε χρησιμοποιείται ο πίνακας δρομολόγησης και το μήνυμα προωθείται σε έναν κόμβο που μοιράζεται κοινό πρόθεμα με το κλειδί τουλάχιστον κατά ένα ψηφίο περισσότερο. Σε ορισμένες



NodeId 10233102

Leaf set

SMALLER		LARGER	
10233033	10233021	10233120	10233122
10233001	10233000	10233230	10233232

Routing table

-0-2212102	1	-2-2301203	-3-1203203
0	1-1-301233	1-2-230203	1-3-021022
10-0-31203	10-1-32102	2	10-3-23302
102-0-0230	102-1-1302	102-2-2302	3
1023-0-322	1023-1-000	1023-2-121	3
10233-0-01	1	10233-2-32	
0		102331-2-0	
		2	

Neighborhood set

13021022	10200230	11301233	31301233
02212102	22301203	31203203	33213321

Εικόνα 14: Η πληροφορία ενός υποθετικού κόμβου Pastry με nodeId 10233102 (βάση 4) σε ένα σύστημα που χρησιμοποιεί nodeIds των 16 bits και b = 2. Η πρώτη γραμμή του πίνακα δρομολόγησης είναι η γραμμή 0. Το γκρι κελί σε κάθε γραμμή του πίνακα δρομολόγησης δείχνει το αντίστοιχο ψηφίο του nodeId του τοπικού κόμβου. Τα nodeIds σε κάθε εγγραφή είναι χωρισμένα έτσι ώστε να φαίνεται το κοινό πρόθεμα με το 10233102 – επόμενο ψηφίο – υπόλοιπο nodeId. Έχουν παραλειφθεί οι IP διευθύνσεις των κόμβων. [42]

περιπτώσεις είναι δυνατό η κατάλληλη εγγραφή στον πίνακα δρομολόγησης να είναι άδεια ή ο κόμβος που υποδεικνύει να μην είναι προσιτός. Τότε, το μήνυμα προωθείται σε έναν κόμβο που μοιράζεται ένα πρόθεμα με το κλειδί ίδιου μήκους με το κοινό πρόθεμα του τοπικού κόμβου με το κλειδί, και που είναι αριθμητικά κοντύτερα στο κλειδί από ότι ο τοπικός κόμβος. Ένας τέτοιος κόμβος πρέπει να υπάρχει στο σύνολο φύλλων, εάν το μήνυμα δεν έχει ήδη φτάσει στον τελικό κόμβο. Και, εάν εξαιρέσουμε την περίπτωση $\lfloor |L| /2 \rfloor$ διπλανοί κόμβοι στο σύνολο φύλλων να έχουν «πέσει» ταυτόχρονα, τουλάχιστον ένας από αυτούς τους κόμβους πρέπει να είναι ζωντανός.
Αυτή η απλή διαδικασία δρομολόγησης πάντα συγκλίνει, γιατί κάθε βήμα οδηγεί το μήνυμα σε έναν κόμβο που είτε μοιράζεται ένα μακρύτερο πρόθεμα με το κλειδί από ότι ο τοπικός κόμβος είτε μοιράζεται ένα πρόθεμα με το ίδιο μήκος αλλά αριθμητικά κοντύτερα στο κλειδί από ότι ο τοπικός κόμβος. Ο αναμενόμενος αριθμός βημάτων δρομολόγησης είναι $\lceil \log_{2^b} N \rceil$ με την προϋπόθεση ότι έχουμε ακριβείς πίνακες δρομολόγησης και καμία πρόσφατη αποτυχία κόμβου.

Το API του Pastry

Αναφέρουμε επιγραμματικά τις κύριες λειτουργίες του Pastry API σε απλοποιημένη μορφή:

- *nodeId = pastryInit (Credentials, Application)*

  Επιτρέπει στον τοπικό κόμβο να προσχωρήσει σε ένα υπάρχων δίκτυο Pastry (ή να ξεκινήσει ένα καινούριο) και να αρχικοποιήσει όλη την απαραίτητη πληροφορία. Επιστρέφει το nodeId του τοπικού κόμβου. Τα *Credentials*, τα οποία εξαρτώνται από την εφαρμογή, περιέχουν την απαραίτητη πληροφορία για την πιστοποίηση της αυθεντικότητας του τοπικού κόμβου. Το όρισμα *Application* είναι ένας χειριστής του αντικειμένου της εφαρμογής, ο οποίος παρέχει στο Pastry τις διαδικασίες που πρέπει να εκκινεί, όταν συμβαίνουν συγκεκριμένα γεγονότα (π.χ. άφιξη μηνύματος).

- *route (msg, key)*

  Δρομολογεί το δεδομένο *msg* στον κόμβο με nodeId αριθμητικά κοντύτερα -σε σχέση με όλους τους ζωντανούς κόμβους του δικτύου Pastry - στο κλειδί *key*.

Οι εφαρμογές που βρίσκονται ένα στρώμα πάνω από το Pastry πρέπει να εξάγουν τις ακόλουθες λειτουργίες:

- *deliver (msg, key)*

  Καλείται από το Pastry, όταν λαμβάνεται ένα μήνυμα και το nodeId του τοπικού κόμβου είναι το πλησιέστερο στο κλειδί.

- *forward (msg, key, nextId)*

  Καλείται από το Pastry, μόλις πριν ένα μήνυμα προωθηθεί στον κόμβο με nodeId = *nextId*. Θέτοντας το *nextId* NULL το μήνυμα τερματίζει την πορεία του στον τοπικό κόμβο.

- *newLeafs(leafSet)*

  Καλείται από το Pastry όποτε υπάρχει αλλαγή στο σύνολο φύλλων του τοπικού κόμβου. Αυτό παρέχει στην εφαρμογή τη δυνατότητα να προσαρμόζει σταθερές συγκεκριμένες για κάθε εφαρμογή στο σύνολο των φύλλων.

Αυτο-διοργάνωση και προσαρμογή

Άφιξη κόμβου Όταν εισάγεται ένας νέος κόμβος στο δίκτυο, πρέπει να αρχικοποιήσει τους πίνακές του και μετά να ενημερώσει τους άλλους κόμβους για την παρουσία του. Υποθέτουμε ότι ο καινούριος κόμβος X γνωρίζει έναν κοντινό -σύμφωνα με το μέτρο εγγύτητας- κόμβο A, που ανήκει ήδη στο δίκτυο Pastry (ένας τέτοιος κόμβος μπορεί να εντοπιστεί αυτόματα με χρήση IP multicast «επεκτεινόμενου δακτυλίου» ή να δοθεί απευθείας από το διαχειριστή του συστήματος μέσω εξωτερικών καναλιών).

Ο X ζητά από τον A να δρομολογήσει ένα ειδικό μήνυμα *"join"* με κλειδί ίσο με X. Το μήνυμα θα παραδοθεί στον κόμβο Z με nodeId πλησιέστερα στο X. Αυτή η αίτηση για *"join"* έχει σαν αποτέλεσμα να στείλουν οι A, Z, και όλοι οι κόμβοι από τους οποίους πέρασε το *"join"*, τους πίνακές τους στον X. Ο νέος κόμβος X ελέγχει όλη αυτήν την πληροφορία, ζητά ενδεχομένως πληροφορία από επιπλέον κόμβους και έτσι δομεί τους δικούς του πίνακες. Τέλος, ο X ενημερώνει τους κόμβους που πρέπει να γνωρίζουν την άφιξή του. Αυτή η διαδικασία εξασφαλίζει ότι ο X αρχικοποιεί την πληροφορία του με σωστές τιμές και ότι η πληροφορία όλων των επηρεαζόμενων κόμβων ανανεώνεται.

Αναχώρηση κόμβου Οι κόμβοι στο δίκτυο Pastry μπορεί να πέσουν ή να αναχωρήσουν χωρίς προειδοποίηση. Ένας κόμβος θεωρείται εκτός δικτύου, όταν οι άμεσοι γείτονές του στο χώρο των nodeIds δεν μπορούν πλέον να επικοινωνήσουν μαζί του. Για να αντικατασταθεί ένας νεκρός κόμβος στο σύνολο φύλλων, ο γείτονας του νεκρού κόμβου στο χώρο των nodeIds έρχεται σε επαφή με το ζωντανό κόμβο με

το μεγαλύτερο δείκτη από την πλευρά του νεκρού κόμβου και του ζητά τον πίνακα φύλλων του. Το σύνολο που λαμβάνει επικαλύπτεται μερικώς από το σύνολο φύλλων του τοπικού κόμβου και περιέχει κόμβους με κοντινά Ids που δε βρίσκονται στο σύνολο φύλλων του τοπικού κόμβου. Ανάμεσα σε αυτούς τους κόμβους επιλέγεται ο κατάλληλος για να εισαχθεί στο σύνολο φύλλων έχοντας πρώτα εξακριβωθεί ότι είναι ζωντανός. Αυτή η διαδικασία εγγυάται ότι κάθε νωχελικός κόμβος διορθώνει το σύνολο φύλλων του αρκεί να μην έχουν πέσει ταυτόχρονα $\lfloor |L| /2 \rfloor$ κόμβοι με γειτονικά nodeIds. Λόγω της διαφοροποίησης (σε θέση, ιδιοκτησία κ.λπ.) των κόμβων με γειτονικά nodeIds μια τέτοια αποτυχία είναι πολύ απίθανη.
Η αποτυχία ενός κόμβου που εμφανίζεται στον πίνακα δρομολόγησης ενός δεύτερου κόμβου ανιχνεύεται, όταν ο δεύτερος κόμβος επιχειρεί να επικοινωνήσει με τον πρώτο και δε λαμβάνει απάντηση. Αυτό το γεγονός δεν καθυστερεί συνήθως τη δρομολόγηση ενός μηνύματος, αφού το μήνυμα μπορεί να προωθηθεί σε άλλο κόμβο. Ωστόσο, πρέπει να βρεθεί μια εγγραφή αντικατάστασης, για να διατηρηθεί η ακεραιότητα του πίνακα δρομολόγησης. Ο δεύτερος κόμβος ζητά αντίστοιχες εγγραφές από άλλους κόμβους της ίδιας σειράς μέχρι να βρει έναν κόμβο με το κατάλληλο πρόθεμα.
Το σύνολο γειτόνων δε χρησιμοποιείται κανονικά στη δρομολόγηση μηνυμάτων, ωστόσο είναι σημαντικό να παραμένει έγκυρο και ενημερωμένο, καθώς παίζει σημαντικό ρόλο στην ανταλλαγή πληροφοριών με κοντινούς κόμβους. Για αυτό το λόγο κάθε κόμβος επιχειρεί να επικοινωνεί περιοδικά με κάθε κόμβο του συνόλου γειτονιάς, για να διαπιστώσει εάν είναι ζωντανός. Εάν κάποιο μέλος δεν ανταποκρίνεται, ο κόμβος ζητά από άλλα μέλη του συνόλου τους πίνακες γειτόνων τους, ελέγχει την απόσταση από κάθε νέο κόμβο που ανακαλύπτει και ενημερώνει τον πίνακά του ανάλογα.

Τοπικότητα
Το Pastry φροντίζει η διαδρομή που θα επιλεχθεί για ένα μήνυμα να είναι η «καλύτερη» δυνατή με χρήση ενός μέτρου εγγύτητας. Η αντίληψη του Pastry για δικτυακή εγγύτητα βασίζεται σε ένα κλιμακούμενο σε σχέση με το μέγεθος του δικτύου μέτρο εγγύτητας, όπως ο αριθμός των IP βημάτων δρομολόγησης ή η γεωγραφική απόσταση. Θεωρούμε ότι κάθε εφαρμογή παρέχει μια συνάρτηση σε κάθε κόμβο Pastry, για να μπορεί να καθορίζει την «απόστασή» του από έναν κόμβο με δεδομένη IP διεύθυνση. Μια εφαρμογή αναμένεται να υλοποιεί αυτή τη συνάρτηση ανάλογα με την επιλογή του μέτρου εγγύτητας, χρησιμοποιώντας υπηρεσίες δικτύου όπως το traceroute, και να χρησιμοποιεί την κατάλληλη προσωρινή αποθήκευση και τεχνικές για την ελαχιστοποίηση των επικεφαλίδων.
Σε σχέση με τον πίνακα δρομολόγησης επιδιώκεται όλες οι εγγραφές να αναφέρονται σε έναν κόμβο κοντινό, σύμφωνα με το μέτρο εγγύτητας, στον τοπικό κόμβο σε σύγκριση με όλους τους άλλους ζωντανούς κόμβους με κατάλληλο πρόθεμα για την εγγραφή. Συμπερασματικά, σε κάθε βήμα δρομολόγησης, ένα μήνυμα προωθείται σε έναν σχετικά κοντινό κόμβο με nodeId που μοιράζεται ένα μεγαλύτερο κοινό πρόθεμα ή είναι αριθμητικά πλησιέστερα στο κλειδί από ότι ο τοπικός κόμβος. Έτσι, κάθε βήμα μετακινεί το μήνυμα πλησιέστερα στον προορισμό του στο χώρο των nodeIds, ενώ ταξιδεύει την ελάχιστη δυνατή απόσταση στο χώρο. Η παραπάνω διαδικασία, την οποία δε θα περιγράψουμε περαιτέρω, προφανώς δεν εγγυάται τη συντομότερη διαδρομή από την πηγή στον επιλεγμένο προορισμό, αλλά δίνει αρκετά καλές διαδρομές.
Τέλος, κάποιες εφαρμογές peer-to-peer που χρησιμοποιούν το Pastry, όπως το PAST, αντιγράφουν πληροφορία στους k κόμβους Pastry με τα αριθμητικά πλησιέστερα

nodeIds στο κλειδί. Έτσι, εξασφαλίζεται υψηλή διαθεσιμότητα παρά τις αποτυχίες. Το Pastry δρομολογεί τα μηνύματά του περιμένοντας ένας τουλάχιστον κόμβος από τους k πλησιέστερους στο κλειδί κόμβους να λάβει το αντίστοιχο για αυτόν μήνυμα. Οι ιδιότητες τοπικότητας του Pastry προσθέτουν το εξής πλεονέκτημα στην παραπάνω δρομολόγηση: κατά μήκος της διαδρομής από έναν πελάτη στον αριθμητικά πλησιέστερο κόμβο, το μήνυμα πρώτα θα φτάσει στον κόμβο που είναι πιο «κοντά» (σύμφωνα με το μέτρο εγγύτητας) στον πελάτη ανάμεσα στους k κόμβους. Έτσι, μειώνεται και η καθυστέρηση για τον πελάτη και ο φόρτος του δικτύου.

Συμπερασματικά
Το Pastry είναι ένα γενικευμένο peer-to-peer σύστημα εντοπισμού περιεχομένου και δρομολόγησης. Χρησιμοποιείται σαν δομικό στοιχείο για το χτίσιμο μιας ποικιλίας peer-to-peer εφαρμογών Διαδικτύου. Δρομολογεί σε οποιονδήποτε κόμβο στο υπερκείμενο δίκτυο σε λιγότερα από O (log N) βήματα, και διατηρεί πίνακες δρομολόγησης με μόνο O (log N) εισόδους. Λαμβάνει υπόψη τοπικότητα κατά τη δρομολόγηση των μηνυμάτων. Ένα προσομοιωμένο δίκτυο των 100.000 κόμβων επιβεβαιώνει ότι είναι αποδοτικό, παρουσιάζει καλή κλιμάκωση ως προς το μέγεθος, είναι πλήρως αυτο-διοργανούμενο, προσαρμόζεται καλά σε αποτυχίες κόμβων και επωφελείται αρκετά από τις ιδιότητες τοπικότητας. [42]

3.2.2.6 Bamboo

Το Bamboo είναι ένας κατανεμημένος πίνακας κατακερματισμού (DHT) που υλοποιήθηκε τον Ιανουάριο του 2003. Αποτελείται από πάνω από 10.000 γραμμές κώδικα Java, εκ των οποίων οι περίπου 3.300 υλοποιούν το στρώμα δρομολόγησης που μελετάται στη συνέχεια. Η ανάπτυξη του Bamboo στόχευε κυρίως σε ένα σύστημα ανθεκτικό σε μεγάλα επίπεδα από "churn". *Churn* είναι η συνεχής διαδικασία εισαγωγής και αποχώρησης κόμβων από το δίκτυο. Αυτό υλοποιείται με βελτιστοποίηση της χρήσης του εύρους δικτύου, το οποίο αυξομειώνεται λογαριθμικά ανάλογα με τον αριθμό των κόμβων στο σύστημα. Το κύριο πλεονέκτημα του Bamboo σε σύγκριση με άλλα συστήματα peer-to-peer, όπως το Chord και το Pastry, είναι ότι δεν καταρρέει σε υψηλά ποσοστά *churn*.
Το Bamboo βασίζεται στο Pastry, αλλά τροποποιεί ελαφρά τα πρωτόκολλα και τους αλγορίθμους του. Ωστόσο, το Bamboo αντιμετωπίζει καλύτερα μεγάλες ή συνεχείς αλλαγές μελών ειδικά σε περιβάλλοντα περιορισμένου εύρους δικτύου. Κατά τα άλλα χρησιμοποιεί ίδιους μηχανισμούς με το Pastry (βλ. παράγραφο 3.2.2.5).

Γεωμετρία δικτύου
Η γεωμετρία του Bamboo είναι παρόμοια με αυτήν του Pastry. Σε κάθε κόμβο του Bamboo ανατίθεται ένα αριθμητικό αναγνωριστικό από την περιοχή [0,2160) (160 bits), το οποίο προέρχεται είτε από τον SHA-1 κατακερματισμό της IP διεύθυνσης και της πόρτας στην οποία ο κόμβος λαμβάνει τα πακέτα ή από τον SHA-1 κατακερματισμό ενός δημόσιου κλειδιού. Κατά αυτόν τον τρόπο τα αναγνωριστικά διανέμονται ομοιόμορφα σε όλο το χώρο των αναγνωριστικών. Κάθε κόμβος στο δίκτυο Bamboo διατηρεί ένα *σύνολο φύλλων* L - το σύνολο των 2k κόμβων που προηγούνται και που ακολουθούν αμέσως στο κυκλικό διάστημα αναγνωριστικών (συνηθισμένο μέγεθος πίνακα 16, άρα k=8). Με Li , όπου $-k \leq i \leq k$, συμβολίζονται τα μέλη του L και Lo είναι ο ίδιος ο κόμβος.

Εκτός από το σύνολο φύλλων, κάθε κόμβος Bamboo διατηρεί έναν *πίνακα δρομολόγησης*. Θεωρώντας κάθε αναγνωριστικό ως ακολουθία ψηφίων βάσης 2^b και συμβολίζοντας την εγγραφή του πίνακα δρομολόγησης στη σειρά l και τη στήλη i με Rl[i], ένας κόμβος επιλέγει τους γείτονές του έτσι ώστε η εγγραφή Rl[i] να είναι ένας κόμβος, του οποίου το αναγνωριστικό ταιριάζει με το δικό του ακριβώς στα πρώτα l ψηφία και του οποίου το (l + 1)-ιοστό ψηφίο είναι i. Όπως και το Pastry, το Bamboo προσπαθεί να επιλέξει τον κόμβο που είναι πλησιέστερα σε αυτόν σε καθυστέρηση δικτύου από όλους τους κόμβους που μπορούν να συμπληρώσουν την είσοδο στον πίνακα δρομολόγησης.

Δρομολόγηση

Αλγοριθμικά, η δρομολόγηση στο Bamboo εξελίσσεται ως εξής: για τη δρομολόγηση ενός μηνύματος με κλειδί D, ένας κόμβος πρώτα ελέγχει εάν το D ανήκει στο σύνολο φύλλων του, και σε αυτή την περίπτωση, το προωθεί στο αριθμητικά πλησιέστερο μέλος αυτού του συνόλου. Εάν αυτό το μέλος είναι ο τοπικός κόμβος, η δρομολόγηση ολοκληρώνεται. Εάν το D δεν εμπίπτει στο σύνολο φύλλων, ο τοπικός κόμβος υπολογίζει το μήκος l του μακρύτερα ταυτιζόμενου προθέματος μεταξύ του D και του αναγνωριστικού του. Έστω ότι D[i] είναι το i-ιοστό ψηφίο του D. Εάν η εγγραφή Rl[D[l]] δεν είναι κενή, το μήνυμα προωθείται σε αυτόν τον κόμβο. Εάν καμία από τις παραπάνω συνθήκες δεν ισχύει, το μήνυμα προωθείται στο μέλος του συνόλου φύλλων του τοπικού κόμβου που είναι αριθμητικά πλησιέστερα στο D.

Η παραπάνω δρομολόγηση είναι αναδρομική (ένα μήνυμα προωθείται από μια σειρά κόμβων στο δρόμο προς τον προορισμό του), αλλά είναι φυσικά δυνατό η δρομολόγηση να γίνει επαναληπτικά. Τότε, ο αρχικός κόμβος εκτελεί μια σειρά αναζητήσεων στους ενδιάμεσους κόμβους πριν στείλει το μήνυμα απευθείας στον προορισμό του. Εντούτοις, η επιλογή της αναδρομικής δρομολόγησης στο Bamboo αποδεικνύεται σημαντική για το χειρισμό του *churn*. Σε ένα δίκτυο με N κόμβους ένα μήνυμα μπορεί να δρομολογηθεί σε οποιονδήποτε κόμβο σε λιγότερα από $\lceil \log_{2^b} N \rceil$ βήματα κατά μέσο όρο, όπου *b* μία ρυθμιζόμενη παράμετρος που επηρεάζει το μέγεθος του πίνακα δρομολόγησης, και είναι συνήθως 4. Το σύνολο φύλλων επιτρέπει την πρόοδο της προώθησης (με αντάλλαγμα ενδεχομένως μεγαλύτερες διαδρομές) στην περίπτωση που ο πίνακας δρομολόγησης είναι ελλιπής [29].

Επιπλέον, το σύνολο φύλλων προσθέτει αρκετή στατική ανθεκτικότητα στη γεωμετρία Bamboo. Έχει αποδειχτεί ότι με ένα σύνολο φύλλων 16 κόμβων ακόμα και εάν «πέσει» ένα τυχαίο 30% των συνδέσεων, υπάρχουν ακόμα διαδρομές μεταξύ όλων των ζευγαριών κόμβων. Τέτοια στατική ανθεκτικότητα είναι σαφώς χρήσιμη στο χειρισμό των αποτυχιών γενικότερα και του *churn* ειδικότερα, και για αυτό επιλέχθηκε η γεωμετρία του Pastry για χρήση στο Bamboo.

Στρώμα επικοινωνίας

Οι κόμβοι Bamboo επικοινωνούν χρησιμοποιώντας πρωτόκολλο UDP. Αρχικά, επιλέχθηκε το UDP για να αποφευχθούν οι δυσκολίες που είχαν εμφανιστεί με τον περιορισμένο αριθμό δεικτών σε αρχεία (file descriptors) τρέχοντας πολλά εικονικά στιγμιότυπα DHT πάνω σε TCP στην ίδια φυσική μηχανή . Μία τέτοια εικονικότητα είναι αρκετά συνηθισμένη για τη δοκιμή μεγάλων δικτύων σε περιορισμένους φυσικούς πόρους και είναι αρκετά χρήσιμη για την εύρεση λαθών. Συνεπώς, το TCP είναι ακατάλληλο για την υλοποίηση ενός DHT. Αντ' αυτού, αυτό που απαιτείται είναι μία βασισμένη στα μηνύματα, αναξιόπιστη, άτακτη, αλλά ελεγχόμενη υπό συμφόρηση επικοινωνία. Ακολουθώντας το ύφος του TCP, το στρώμα επικοινωνίας του Bamboo χρησιμοποιεί το χρόνο μεταξύ της αποστολής ενός μηνύματος και της

παραλαβής της αντίστοιχης επιβεβαίωσης (acknowledgement – ACK), για να διατηρήσει έναν εκθετικά σταθμισμένο μέσο χρόνο ταξιδιού μετ'επιστροφής (Round Trip Time - RTT). Αυτές οι τιμές τίθενται στη διάθεση των υψηλότερων στρωμάτων του συστήματος. Υπολογίζονται οι χρόνοι λήξης ταξιδιού μετ'επιστροφής (Round-Trip Timeouts - RTOs), για να αποφασιστεί πότε θα αναμεταδοθεί ένα πακέτο, και με κάθε λήξη ο χρόνος RTO υποχωρεί εκθετικά. Διατηρείται επίσης ένα παράθυρο συμφόρησης κατά τρόπο παρόμοιο με αυτόν του αλγορίθμου αργής έναρξης του TCP, το οποίο ενημερώνει την εφαρμογή όποτε λαμβάνεται ACK για ένα πακέτο.

Bamboo και *churn*

Το Bamboo παρουσιάζει ευρωστία κάτω από συνθήκες *churn* μέσω του συνδυασμού τριών κύριων χαρακτηριστικών:

- στατική ανθεκτικότητα στις αποτυχίες
- έγκαιρη και ακριβής ανίχνευση αποτυχίας
- μηχανισμοί αποκατάστασης που αφορούν τη συμφόρηση

Στατική ανθεκτικότητα σημαίνει ότι το Bamboo μπορεί να συνεχίσει να εκτελεί αναζητήσεις παρά τις αποτυχίες κόμβων, δρομολογώντας τα μηνύματα γύρω από αυτούς ακόμη και προτού αρχίσει η αποκατάσταση. Εντούτοις, για να γίνει αυτό, είναι απαραίτητο το σύστημα να διακρίνει με ακρίβεια τους κόμβους με υψηλό φόρτο ή εκείνους που βρίσκονται στις διαδρομές δικτύου που υποφέρουν από συμφόρηση. Η ανικανότητα να παρατηρηθούν οι αποτυχίες οδηγεί γρήγορα σε υπερβολικές καθυστερήσεις αναζητήσεων, ενώ η πολύ σύντομη υπόθεση για μία αποτυχία οδηγεί σε κατάρρευση λόγω συμφόρησης. Ένας συνδυασμός ενεργού εξέτασης και αναδρομικής δρομολόγησης επιτρέπει στο Bamboo να κάνει αποτελεσματικά αυτή τη διάκριση. Τέλος, το Bamboo ενσωματώνει τους νέους κόμβους και συνέρχεται από την αποτυχία παλιών με ένα τρόπο καθοδηγούμενο από τη συμφόρηση. Η δυναμική αποκατάσταση - όπου ένας DHT προσπαθεί να αντιδράσει αμέσως στις αλλαγές των μελών - μονάχα προσθέτει επιπλέον πίεση σε ένα ήδη πιεσμένο δίκτυο. Για να αποφύγει την κατάρρευση λόγω συμφόρησης, το Bamboo χρησιμοποιεί περιοδικούς αλγορίθμους για την αυτόματη (ανάλογα με τη συμφόρηση) κλιμάκωση των περιόδων διατήρησης.

Κατ' αρχάς, οι κόμβοι Bamboo αποκτούν και διατηρούν την πληροφορία γειτόνων με διάφορες συνεχείς και ανάλογα με τη συμφόρηση διαδικασίες βελτιστοποίησης. Αυτή η διαδικασία διαιρείται στις εξής ενέργειες: 1) παράγοντας ορθότητα μέσω της διατήρησης των συνόλων φύλλων, 2) παράγοντας αποδοτικότητα μέσω της συμπλήρωσης του πίνακα δρομολόγησης, 3) συντονίζοντας τις καταχωρήσεις στον πίνακα δρομολόγησης σύμφωνα με την εγγύτητα. Σε όλες τις περιπτώσεις, ο ρυθμός με τον οποίο στέλνονται αυτά τα μηνύματα κλιμακώνεται και αποκλιμακώνεται ανάλογα με τη συμφόρηση ή το φορτίο. Δεύτερον, οι κόμβοι Bamboo εξετάζουν ενεργά τους γείτονές τους, για να υπολογίσουν καλές τιμές χρόνου λήξης, και με τη χρησιμοποίηση της αναδρομικής δρομολόγησης αποφεύγουν να στέλνουν μηνύματα στους κόμβους για τους οποίους δεν έχουν τέτοιους χρόνους λήξης.

Συμπερασματικά

Η ανθεκτικότητα του Bamboo σε συνθήκες *churn* είναι το συνδυασμένο αποτέλεσμα διάφορων παραγόντων. Κατ' αρχάς, η στατική ανθεκτικότητα της υβριδικής γεωμετρίας του επιτρέπει στο δίκτυο να συνεχίσει μετά από μια αποτυχία έως ότου ξεκινήσει η αποκατάσταση. Αυτή η στατική ανθεκτικότητα είναι ένα απαραίτητο πρώτο βήμα για το χειρισμό των αποτυχιών γενικά και του *churn* ειδικότερα. Δεύτερον, η αποκατάσταση εκτελείται κατά τρόπο περιοδικό: η ίδια η διαδικασία

ανάκαμψης δε φορτώνει περαιτέρω το δίκτυο, αποφεύγοντας ένα πιθανό θετικό σύστημα ανατροφοδότησης πληροφοριών στο οποίο η συμφόρηση ερμηνεύεται ως περαιτέρω αποτυχία κόμβων. Τρίτον, η χρήση της αναδρομικής δρομολόγησης από το Bamboo επιτρέπει την ενεργό εξέταση των συνδέσεων γειτόνων, η οποία επιτρέπει στη συνέχεια τον υπολογισμό των αποτελεσματικών χρόνων λήξης απάντησης για τις αιτήσεις που στέλνονται πάνω από αυτές τις συνδέσεις. Χωρίς τόσο ακριβείς πληροφορίες για τους χρόνους λήξης, μια πιθανή αποτυχία κόμβων είναι δύσκολο να διακριθεί από συμφόρηση δικτύου ή επεξεργαστή, και η δυνατότητα του DHT να προσαρμοστεί εμποδίζεται. Επιπλέον, στα δίκτυα με μεγάλη κίνηση, αναμένεται αυτή η ανθεκτικότητα στη συμφόρηση να φανεί σημαντική στο χειρισμό χαμηλότερων επιπέδων *churn*. [43],[44]

3.2.3 Κοινό API για δομημένα peer-to-peer Υπερκείμενα Δίκτυα

Τα δομημένα peer-to-peer Υπερκείμενα Δίκτυα έχουν γίνει ιδιαίτερα δημοφιλή ως πλατφόρμα κατασκευής ανθεκτικών κατανεμημένων συστημάτων μεγάλης κλίμακας. Τα δομημένα υπερκείμενα στρώματα έχουν μια συγκεκριμένη χαρακτηριστική δομή γράφου, η οποία επιτρέπει τον εντοπισμό αντικειμένων ανταλλάσσοντας **O (logN)** μηνύματα σε ένα υπερκείμενο δίκτυο N κόμβων. Υπηρεσίες που επωφελούνται από την ιδιαίτερη δομή τους είναι οι κατανεμημένοι πίνακες κατακερματισμού (Distributed Hash Tables – DHTs), το κλιμακούμενο σε μέγεθος group multicast / anycast (CAST) και ο αποκεντρωμένος εντοπισμός αντικειμένου (DOLR). Πάνω σε αυτές τις υπηρεσίες μπορούν να χτιστούν με τη σειρά τους κλιμακούμενες σε μέγεθος, ανθεκτικές και κατανεμημένες εφαρμογές.
Προς το παρόν, κάθε πρωτόκολλο υπερκείμενου δικτύου εξάγει ένα διαφορετικό API και ένα σύνολο υπηρεσιών με σχετικά διαφορετικές σημασιολογίες. Αυτό έχει ως αποτέλεσμα οι σχεδιαστές εφαρμογών να χρειάζεται να κατανοούν κάθε πρωτόκολλο και τις υπηρεσίες που προσφέρει, ώστε να επιλέγουν κάθε φορά το κατάλληλο για την εφαρμογή τους. Επιπλέον, οι εφαρμογές δομούνται πάνω σε ένα συγκεκριμένο πρωτόκολλο και λόγω των διαφοροποιήσεων στις υπηρεσίες που προσφέρουν τα διάφορα πρωτόκολλα δυσχεραίνεται η αντικειμενική αξιολόγησή τους. Για αυτούς τους λόγους, επιδιώκεται να οριστούν οι στοιχειώδεις αφαιρετικές έννοιες των δομημένων υπερκείμενων στρωμάτων και τα APIs για ένα κοινό σύνολο υπηρεσιών που όλα αυτά θα προσφέρουν.

Κοινές έννοιες σε όλα τα δομημένα Υπερκείμενα Δίκτυα
Κόμβος: ένα στιγμιότυπο ενός συμμετέχοντος στο υπερκείμενο δίκτυο. Ένας ή περισσότεροι κόμβοι μπορούν να φιλοξενούνται σε έναν φυσικό IP υπολογιστή.
NodeId: το αναγνωριστικό κάθε κόμβου, το οποίο ανατίθεται τυχαία και ομοιόμορφα από ένα μεγάλο χώρο αναγνωριστικών.
Χώρος αναγνωριστικών:

- Tapestry και Chord: κυκλικός χώρος αναγνωριστικών των n-bit ακεραίων modulo 2^n , n = 160
- Pastry: ομοίως, με n = 128
- CAN: d-διάστατος καρτεσιανός χώρος αναγνωριστικών με nodeIds των 128 bits, που ορίζουν σημεία στο χώρο

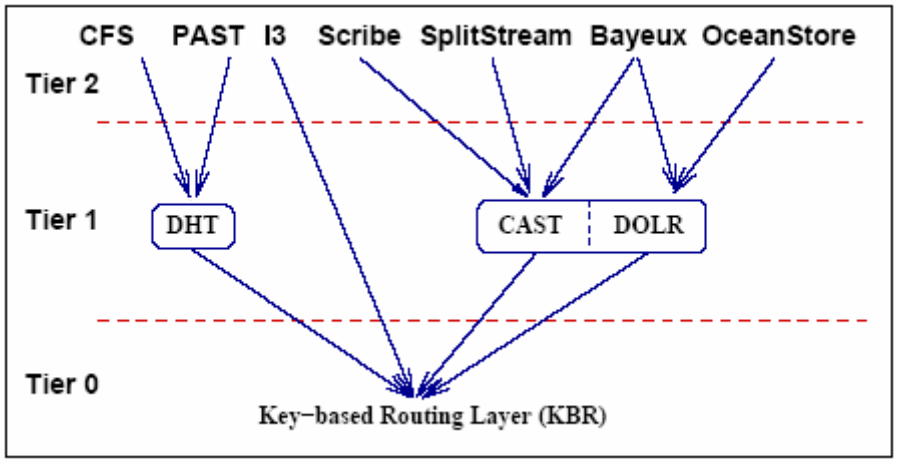
Κλειδί: μοναδικό αναγνωριστικό που ανατίθεται σε κάθε αντικείμενο μιας συγκεκριμένης εφαρμογής και επιλέγεται από τον παραπάνω χώρο αναγνωριστικών.
Ρίζα: ένας μοναδικός ζωντανός κόμβος, στον οποίο αντιστοιχίζεται ένα κλειδί από το υπερκείμενο δίκτυο.

Πίνακας δρομολόγησης: ένας πίνακας με τα nodeIds και τις IP διευθύνσεις των κόμβων με τους οποίους συνδέεται ο τοπικός κόμβος στο υπερκείμενο δίκτυο. Με χρήση αυτού του πίνακα προωθούνται τα μηνύματα στον κόμβο με το κοντινότερο nodeId στο κλειδί του αντικειμένου, το οποίο σχετίζεται με το μήνυμα. Είναι απαραίτητος, για να φτάσει το μήνυμα στη ρίζα.

Κάθε σύστημα χρησιμοποιεί μια συνάρτηση που αντιστοιχεί κλειδιά σε κόμβους.

- Chord: κάθε κλειδί αντιστοιχίζεται στο ζωντανό κόμβο με το κοντινότερο nodeId σε αυτό με τη φορά του ρολογιού.
- Pastry: κάθε κλειδί αντιστοιχίζεται στο ζωντανό κόμβο με το κοντινότερο nodeId σε αυτό.
- Tapestry: κάθε κλειδί αντιστοιχίζεται στο ζωντανό κόμβο με το nodeId που ταυτίζεται μακρύτερα σε πρόθεμα με το κλειδί.
- CAN: οι γειτονικοί κόμβοι στο χώρο διευθύνσεων συμφωνούν σε κατάτμηση του χώρου γύρω από τα nodeIds τους. Κάθε κλειδί αντιστοιχίζεται στο ζωντανό κόμβο που είναι υπεύθυνος για το χώρο που περιέχει το κλειδί.

Αφαιρέσεις
Όλα τα υπάρχοντα συστήματα παρέχουν επίπεδα αφαίρεσης πάνω στα βασικώς δομημένα υπερκείμενα δίκτυα. Παραδείγματα τέτοιων επιπέδων αφαίρεσης είναι.οι **DHTs** (Distributed Hash Tables), το **DOLR** (Decentralized Object Location and Routing), και το **CAST** (group anycast /multicast). Στην Εικόνα 15 φαίνεται πώς αυτές οι αφαιρέσεις σχετίζονται μεταξύ τους.
Το στρώμα 0 παρέχει την κοινή για όλα τα συστήματα δρομολόγηση με βάση το κλειδί (Key-Based Routing - KBR). Στο στρώμα 1 έχουμε ένα υψηλότερο επίπεδο αφαίρεσης (DHT/CAST/DOLR). Στο στρώμα 2 ανήκουν οι περισσότερες εφαρμογές και υπηρεσίες υψηλού επιπέδου, οι οποίες χρησιμοποιούν μία ή περισσότερες αφαιρέσεις από το προηγούμενο επίπεδο. Κάποια συστήματα στρώματος 2, όπως το I3 [46], χρησιμοποιούν κατευθείαν τη δρομολόγηση με βάση το κλειδί (KBR).



Εικόνα 15: Βασικές αφαιρέσεις και APIs. Τα interfaces του 1ου στρώματος είναι: DHT, DOLR, CAST. [45]

Η αφαίρεση DHT παρέχει την ίδια λειτουργικότητα με έναν παραδοσιακό πίνακα κατακερματισμού. Με άλλα λόγια, υλοποιείται η αποθήκευση των αντιστοιχήσεων κλειδί-τιμή (key-value). Αυτό το επίπεδο υλοποιεί τις δύο απλές λειτουργίες της

αποθήκευσης και επαναφοράς, ενώ η τιμή είναι πάντα αποθηκευμένη στο ζωντανό κόμβο στον οποίο αντιστοιχίζεται το κλειδί από το στρώμα KBR. Οι τιμές μπορεί να είναι αντικείμενα οποιουδήποτε τύπου.
Η αφαίρεση DOLR παρέχει μια υπηρεσία κατανεμημένου / αποκεντρωμένου καταλόγου. Κάθε αντίγραφο αντικειμένου έχει ένα *objectID* και μπορεί να τοποθετηθεί οπουδήποτε μέσα στο σύστημα. Οι εφαρμογές ανακοινώνουν την παρουσία των αντικειμένων *δημοσιεύοντας* την τοποθεσία τους. Ένα μήνυμα πελάτη που απευθύνεται σε ένα συγκεκριμένο objectID παραδίδεται σε έναν κοντινό κόμβο με το ίδιο objectID. Ο υποκείμενος κατανεμημένος κατάλογος μπορεί να υλοποιηθεί με το σχηματισμό δέντρων για κάθε objectID. Ωστόσο αυτή δεν είναι η μόνη δυνατή υλοποίηση. Εύλογο ερώτημα είναι γιατί το DOLR δεν υλοποιείται πάνω από ένα DHT αποθηκεύοντας ως τιμές δείκτες σε δεδομένα. Ωστόσο, κάτι τέτοιο δεν είναι δυνατό, γιατί το DOLR δρομολογεί μηνύματα στον κοντινότερο διαθέσιμο κόμβο, ενώ το στρώμα DHT δεν υποστηρίζει συνήθως τέτοια ιδιότητα τοπικότητας. Βασικό κομμάτι αυτής της διαδικασίας είναι η διατήρηση του κατανεμημένου καταλόγου παρόλες τις αλλαγές στους υποκείμενους κόμβους και συνδέσεις.
Η αφαίρεση CAST παρέχει κλιμακούμενη σε μέγεθος επικοινωνία και συνεργασία ομάδων. Υπερκείμενοι κόμβοι μπορούν να προσχωρούν ή να αφήνουν μια ομάδα, να στέλνουν multicast μηνύματα στην ομάδα ή να στέλνουν anycast μηνύματα σε κάποιο μέλος της ομάδας. Επειδή η ομάδα αναπαρίσταται σαν δέντρο, η διαχείριση μελών είναι κατανεμημένη. Έτσι, το CAST μπορεί να υποστηρίξει μεγάλες και εξαιρετικά δυναμικές ομάδες. Επιπλέον, εάν το υπερκείμενο στρώμα στην υπηρεσία KBR έχει ιδιότητες τοπικότητας, τότε το multicast είναι ιδιαίτερα αποδοτικό και τα anycast μηνύματα παραδίδονται στο μέλος που βρίσκεται πλησιέστερα στον κόμβο-πηγή.
Οι αφαιρέσεις DOLR και CAST είναι στενά σχετιζόμενες. Και οι δύο διατηρούν ένα σύνολο από κόμβους δεσπαρμένους στο δίκτυο και λαμβάνουν υπόψη την τοπικότητα των κόμβων, για να φτιάξουν το δέντρο που οδηγεί από κάθε κόμβο στην κοινή ρίζα που σχετίζεται με το σύνολο των κόμβων. Ωστόσο, η αφαίρεση DOLR προσανατολίζεται στον εντοπισμό αντικειμένου, ενώ η αφαίρεση CAST στοχεύει στην επικοινωνία ανάμεσα στα μέλη μιας ομάδας. Συνεπώς, οι υλοποιήσεις τους συνδυάζουν διαφορετικές τακτικές με βάση έναν κοινό μηχανισμό. Από την άλλη πλευρά, η αφαίρεση DHT παρέχει μια ευρεία υπηρεσία αποθήκευσης ζευγών (κλειδί-τιμή).
Στόχος είναι ο ορισμός των APIs για τα DHT, DOLR και CAST interfaces. Έχει ήδη προταθεί ένα API για τη δρομολόγηση με βάση το κλειδί (KBR API), για το οποίο δίνονται περισσότερα στοιχεία αμέσως παρακάτω

Το API για τη δρομολόγηση με βάση το κλειδί

- ❖ Τύποι δεδομένων

Το *key* είναι ένα 160-bit String. Ο *nodehandle* ενθυλακώνει τη διεύθυνση μεταφοράς και το nodeId ενός κόμβου στο σύστημα. Το nodeId είναι τύπου *key*, ενώ η διεύθυνση μεταφοράς μπορεί να είναι, επί παραδείγματι, ένα ζεύγος IP διεύθυνσης-πόρτας. Τα μηνύματα (*msg*) περιέχουν πληροφορία εφαρμογής αυθαιρέτου μήκους.

- ❖ συμβολισμοί ανεξαρτήτου γλώσσας για την περιγραφή του API

→ p : παράμετρος read-only
↔ p : παράμετρος read-write
T [ ] p : ταξινομημένο σύνολο p αντικειμένων τύπου T

- ❖ Δρομολόγηση μηνυμάτων

void route (key → K, msg → M, nodehandle → hint)

αυτή η συνάρτηση προωθεί ένα μήνυμα *M* προς τη ρίζα του κλειδιού *K*. Το προαιρετικό όρισμα *hint* προσδιορίζει τον κόμβο που θα έπρεπε να είναι το πρώτο βήμα κατά τη δρομολόγηση του μηνύματος. Ένα «καλό» *hint* μπορεί να καταλήξει στην παράδοση του μηνύματος σε ένα βήμα, ενώ ένα «κακό» *hint* προσθέτει το πολύ ένα βήμα στη δρομολόγηση. Ή το κλειδί *K* ή το *hint* μπορεί να είναι NULL, αλλά ποτέ και τα δύο συγχρόνως. Η παράδοση του μηνύματος είναι μια υπηρεσία καλύτερης προσπάθειας, καθώς το μήνυμα μπορεί να χαθεί, να αντιγραφεί, να «φθαρεί» ή να καθυστερήσει κατά ένα απροσδιόριστο χρονικό διάστημα.

Η λειτουργία *route* παραδίδει ένα μήνυμα στη ρίζα του κλειδιού. Οι εφαρμογές επεξεργάζονται τα μηνύματα εκτελώντας κώδικα στα upcalls που επιφέρονται από το σύστημα δρομολόγησης KBR στους κόμβους που μεσολαβούν από την πηγή του μηνύματος έως τη ρίζα. Για να είναι δυνατές και αποδοτικές οι οδηγούμενες από τα γεγονότα υλοποιήσεις, οι χειριστές των upcalls δεν πρέπει να μπλοκάρουν ούτε να αναλαμβάνουν μακριούς υπολογισμούς.

void forward (key ↔ K, msg ↔ M, nodehandle ↔ nextHopNode)

αυτό το upcall επιφέρεται σε κάθε κόμβο που προωθεί το μήνυμα *M*, συμπεριλαμβανομένου της πηγής και του κόμβου-ρίζας του κλειδιού. Το upcall ενημερώνει την εφαρμογή ότι το μήνυμα *M* με κλειδί *K* είναι έτοιμο προς προώθηση στον *nextHopNode*. Η εφαρμογή μπορεί να τροποποιήσει τις *M*, *K* και *nextHopNode* παραμέτρους ή να τερματίσει το μήνυμα θέτοντας το *nextHopNode* σε NULL. Έτσι, παρακάμπτει και την default συμπεριφορά δρομολόγησης.

void deliver (key → K, msg → M)

αυτή η συνάρτηση καλείται στον κόμβο που είναι η ρίζα του κλειδιού *K* κατά την άφιξη του μηνύματος *M*. Χρησιμεύει στην ανάπτυξη των εφαρμογών.

Πρόσβαση στην πληροφορία δρομολόγησης

Το API επιτρέπει στις εφαρμογές να έχουν πρόσβαση στην πληροφορία δρομολόγησης ενός κόμβου μέσω των παρακάτω κλήσεων. Όλες αυτές οι λειτουργίες είναι αυστηρά τοπικές και δεν εμπλέκουν καθόλου άλλους κόμβους. Μία εφαρμογή μπορεί, επί παραδείγματι, να ζητήσει κομμάτι της πληροφορίας δρομολόγησης, για να μάθει ποιοι κόμβοι μπορούν να χρησιμοποιηθούν από το upcall της προώθησης ως επόμενα βήματα.

Κάποιες λειτουργίες επιστρέφουν πληροφορία για την **r-ρίζα (r-root)** ενός κλειδιού. Η r-ρίζα είναι μία γενίκευση της ρίζας ενός κλειδιού. Ένας κόμβος είναι η r-ρίζα για ένα κλειδί, εάν αυτός ο κόμβος γίνεται ρίζα του κλειδιού, όταν όλες οι i-ρίζες με i < r αποτυγχάνουν.

- *nodehandle[] local_lookup (key → K, int → num, boolean → safe)*

αυτή η κλήση παράγει μια λίστα από κόμβους που μπορούν να χρησιμοποιηθούν σαν επόμενα βήματα στη διαδρομή προς το κλειδί *K*, έτσι ώστε η διαδρομή να ικανοποιεί τελικά τον περιορισμό του υπερκείμενου πρωτοκόλλου όσο αφορά τον αριθμό των βημάτων.

Εάν η τιμή του *safe* είναι "true", το αναμενόμενο ποσοστό των «ελαττωματικών» κόμβων στη λίστα θα είναι σίγουρα το πολύ όσο το ποσοστό των «ελαττωματικών» κόμβων στο υπερκείμενο δίκτυο. Στην περίπτωση που η τιμή του *safe* είναι "false", η λίστα των κόμβων μπορεί να επιλεχθεί έτσι ώστε να

βελτιστοποιείται η απόδοση, αλλά συγχρόνως να αυξάνεται το ποσοστό των «ελαττωματικών» κόμβων. Αυτή η επιλογή επιτρέπει στις εφαρμογές να υλοποιούν δρομολόγηση σε υπερκείμενα δίκτυα με μη αξιόπιστους κόμβους. Οι υλοποιήσεις που τερματίζουν σε περίπτωση αποτυχίας μπορούν να αγνοούν την παράμετρο *safe*. Το ποσοστό των «ελαττωματικών» κόμβων στην επιστρεφόμενη λίστα μπορεί να είναι σχετικά μεγάλο, εάν η παράμετρος *safe* δεν είναι "true", γιατί ενδεχομένως κακόβουλοι κόμβοι να έχουν προκαλέσει στον τοπικό κόμβο τη δόμηση ενός πίνακα δρομολόγησης που κατευθύνει προς κακόβουλους κόμβους [47].

- *nodehandle [] neighborSet (int → num)*

αυτή η συνάρτηση παράγει μια μη διατεταγμένη λίστα από nodehandles που είναι γείτονες του τοπικού κόμβου στον ID χώρο. Επιστρέφονται μέχρι *num* nodehandles.

- *nodehandle [] replicaSet (key → k, int → max_rank)*

αυτή η συνάρτηση επιστρέφει ένα διατεταγμένο σύνολο από nodehandles, στους οποίους μπορούν να αποθηκευτούν αντίγραφα του αντικειμένου με κλειδί *k*. Οι κόμβοι που επιστρέφονται έχουν μέχρι και *max_rank* τάξη. Εάν το *max_rank* υπερβαίνει το μέγιστο αριθμό αντιγράφων (replica set size) της υλοποίησης, τότε επιστρέφονται nodehandles ίσοι σε αριθμό με το μέγιστο αριθμό αντιγράφων. Μερικά πρωτόκολλα (Tapestry, CAN) υποστηρίζουν μόνο *max_rank* τιμής 1. Στα υπόλοιπα πρωτόκολλα οι επιστρεφόμενοι κόμβοι μπορούν να χρησιμοποιηθούν για την αντιγραφή δεδομένων, αφού αποτελούν τους κόμβους που θα γίνουν ρίζες για το κλειδί k, όταν «πέσει» ο τοπικός κόμβος.

- *update(nodehandle → n, bool → joined)*

αυτό το upcall επιφέρεται, για να ενημερώσει την εφαρμογή ότι ο κόμβος *n* έχει είτε προσχωρήσει είτε εγκαταλείψει το σύνολο γειτόνων του τοπικού κόμβου.

- *boolean range (nodehandle→ N, rank →r, key ↔ lkey, key ←rkey)*

αυτή η λειτουργία παρέχει πληροφορίες για τις περιοχές κλειδιών, για τις οποίες είναι r-ρίζα ο κόμβος *N*. Επιστρέφει "false", εάν η περιοχή των κλειδιών δεν μπορεί να καθοριστεί, διαφορετικά επιστρέφει "true". Είναι λάθος να ζητήσει κανείς την περιοχή ενός κόμβου που δεν ανήκει στο σύνολο των γειτόνων, ενώ κάποιες υλοποιήσεις μπορεί να επιστρέψουν λάθος, εάν το *r* είναι μεγαλύτερο του μηδενός. Το ζεύγος *[lkey,rkey]* δίνει μία περιοχή από τιμές κλειδιών, η οποία περιλαμβάνεται στη ζητούμενη περιοχή.
Μερικά πρωτόκολλα μπορεί να έχουν πολλαπλές, ανεξάρτητες περιοχές κλειδιών, για τις οποίες είναι υπεύθυνος ένας δεδομένος κόμβος. Η παράμετρος *lkey* προσδιορίζει ποια περιοχή να επιστραφεί. Εάν ο κόμβος *N* είναι υπεύθυνος για το κλειδί *lkey*, τότε η επιστρεφόμενη περιοχή θα είναι αυτή που περιέχει το *lkey*. Διαφορετικά, επιστρέφεται η περιοχή που είναι πλησιέστερα με τη φορά του ρολογιού στο *lkey* και για την οποία είναι υπεύθυνος ο κόμβος *N*.

Χρήση του API δρομολόγησης με βάση το κλειδί

- **DHT**

Ένας DHT παρέχει δύο λειτουργίες: 1) *put (key, value)* και 2) *value = get (key).*
Μια απλή υλοποίηση του *put* δρομολογεί ένα μήνυμα PUT με περιεχόμενο το *value* και τον nodehandle S του τοπικού κόμβου. Αυτό γίνεται χρησιμοποιώντας τη *route*

(key, [PUT, value, S], NULL). Η ρίζα του κλειδιού, R, αποθηκεύει το ζεύγος (key, value) στον τοπικό αποθηκευτικό χώρο, μόλις λάβει το μήνυμα. Εάν το *value* είναι μεγάλο σε μέγεθος, μπορεί να απαντήσει η ρίζα στο PUT επιστρέφοντας τον nodehandle της, R, και ύστερα ο S να στείλει το *value* σε ένα βήμα με *route (key, [PUT, value], R)*.
Η λειτουργία *get* μπορεί να υλοποιηθεί με ένα μήνυμα GET του τύπου *route (key, [GET, S], NULL)*. Τότε η ρίζα του κλειδιού επιστρέφει το *value* και τον nodehandle της σε ένα βήμα με *route (NULL, [value, R], S)*. Εάν ο τοπικός κόμβος θυμάται τον R από προηγούμενο GET για αυτό το κλειδί, μπορεί να δώσει τον nodehandle R σαν *hint*.

- **CAST**

Η επικοινωνία ομάδων αποτελεί σημαντικό δομικό στοιχείο σε πολλές κατανεμημένες εφαρμογές. Παρακάτω περιγράφεται μια προσέγγιση υλοποίησης της αφαίρεσης CAST. Ένα κλειδί αντιστοιχίζεται σε μια ομάδα και η ρίζα του κλειδιού γίνεται η ρίζα του multicast δέντρου της ομάδας. Οι κόμβοι προσχωρούν στην ομάδα δρομολογώντας ένα μήνυμα SUBSCRIBE που περιέχει τον nodehandle τους στο κλειδί της ομάδας.
Όταν καλείται ένα upcall προώθησης σε έναν κόμβο, ο κόμβος ελέγχει εάν είναι μέλος της ομάδας. Εάν είναι, τερματίζεται η δρομολόγηση του μηνύματος SUBSCRIBE. Διαφορετικά, εισάγει τον nodehandle του στο μήνυμα και το προωθεί προς τη ρίζα της ομάδας. Έτσι, εγγράφεται εμμέσως στην ομάδα. Σε κάθε περίπτωση, προσθέτει τον nodehandle του κόμβου που κάνει "join" στη λίστα των παιδιών του multicast δέντρου της ομάδας.
Κάθε υπερκείμενος κόμβος μπορεί να κάνει multicast ένα μήνυμα στην ομάδα δρομολογώντας ένα μήνυμα MCAST με το κλειδί της ομάδας. Λαμβάνοντας αυτό το μήνυμα η ρίζα του κλειδιού θα το προωθήσει στα παιδιά της κ.ο.κ. Για να στείλει ένας κόμβος μήνυμα anycast αρκεί να δρομολογήσει ένα μήνυμα ANYCAST με το κλειδί της ομάδας. Ο πρώτος κόμβος στη διαδρομή που ανήκει στην ομάδα προωθεί το μήνυμα σε ένα από τα παιδιά του και όχι προς τη ρίζα. Το μήνυμα συνεχίζει την πορεία του κατεβαίνοντας το δέντρο μέχρι να παραδοθεί στην εφαρμογή. Εάν η υποκείμενη KBR υποστηρίζει τοπικότητα, τότε ο anycast παραλήπτης είναι ένα μέλος της ομάδας κοντά στον anycast αποστολέα.

- **DOLR**

Το στρώμα DOLR δίνει τη δυνατότητα σε εφαρμογές να ελέγχουν την τοποθέτηση αντικειμένων στο υπερκείμενο δίκτυο. Οι λειτουργίες που παρέχει είναι οι ακόλουθες τρεις: *publish(objectId)*, *unpublish(objectId)* και *sendToObj(msg, objectId, [n])*.
Η λειτουργία *publish* ανακοινώνει τη διαθεσιμότητα ενός αντικειμένου υπό το όνομα *objectId* στον κόμβο που επιχειρεί αυτή τη λειτουργία. Ένα απλό *publish* μπορεί να γίνει με κλήση της *route(objectId, [PUBLISH, objectId, S], NULL)*, όπου *S* είναι το όνομα του κόμβου-πηγή. Σε κάθε βήμα ένας χειριστής των upcalls της εφαρμογής αποθηκεύει τοπικά την αντιστοίχηση *objectId – S*. Κάποια άλλη πιο εμπνευσμένη έκδοση του *publish* θα μπορούσε να βάζει δείκτες προς δευτερεύοντα μονοπάτια προς τη ρίζα. Με τη λειτουργία *unpublish* ακολουθείται η ίδια διαδρομή και αφαιρούνται οι αντιστοιχήσεις (*route(objectId, [UNPUBLISH, objectId, S], NULL)*).
Η λειτουργία *sendToObj* παραδίδει ένα μήνυμα σε *n* κοντινά αντίγραφα ενός αντικειμένου. Ξεκινά με τη δρομολόγηση ενός μηνύματος τύπου *route(objectId, [n, msg], NULL)* και σε κάθε βήμα ο χειριστής των upcalls ψάχνει τοπικά για αντιστοιχήσεις στο δεδομένο *objectId* και στέλνει απευθείας ένα αντίγραφο του μηνύματος στις *n* κοντινότερες θέσεις. Εάν βρεθούν λιγότεροι από *n* δείκτες, ο

χειριστής ελαττώνει την τιμή του *n* στον αριθμό των ευρεθέντων δεικτών και προωθεί το παραπάνω αρχικό μήνυμα.
Ακολουθεί ένας πίνακας που συνοψίζει τις λειτουργίες των τριών αφαιρέσεων επιπέδου1.

DHT	DOLR	CAST
put (key, data)	*publish (objectId)*	*join(groupId)*
remove (key)	*unpublish (objectId)*	*leave(groupId)*
value = get (key)	*sendToObj (msg, objectId, [n])*	*multicast(msg, groupId)* *anycast(msg, groupId)*

Εικόνα 16: Interfaces επιπέδου 1. [45]

Υλοποίηση του API σε ήδη υπάρχοντα δομημένα υπερκείμενα πρωτόκολλα
CAN
Η λειτουργία *route* υποστηρίζεται από ήδη υπάρχουσες λειτουργίες και η λειτουργικότητα του *hint* μπορεί εύκολα να προστεθεί. Η κλήση *range* επιστρέφει την περιοχή που σχετίζεται με τον τοπικό κόμβο, η οποία στο CAN μπορεί να αναπαρασταθεί από ένα δυαδικό πρόθεμα. Η *local_lookup* είναι τοπική αναζήτηση στον πίνακα δρομολόγησης και προς το παρόν αγνοεί την τιμή του *safe*. Η λειτουργία *update* καλείται κάθε φορά που ένας κόμβος χωρίζει την περιοχή του στο χώρο των IDs ή όταν ενώνει την περιοχή του με αυτή ενός γείτονα.
Chord
Η δρομολόγηση υλοποιείται επαναληπτικά στο Chord. Σε κάθε βήμα ο τοπικός κόμβος προκαλεί ένα RPC στον επόμενο κόμβο της διαδρομής. Αυτό το RPC επιφέρει το κατάλληλο upcall (*route* ή *deliver*) και επιστρέφει τον κόμβο επόμενου βήματος. Εάν δοθεί *hint*, χρησιμοποιείται ως πρώτο βήμα κατά την αναζήτηση αντί να επιλέγεται ένας κόμβος από τον πίνακα δρομολόγησης. Η *local_lookup* επιστρέφει τους κοντινότερους *num* διαδόχους (successors) κόμβους του τοπικού κόμβου που βρίσκονται στον πίνακά του. Οι κλήσεις *neighborSet* και *replicaSet* επιστρέφουν τη λίστα των διαδόχων του κόμβου. Η *neighborSet* επιστρέφει επιπλέον τον προηγούμενο κόμβο. Η κλήση *range* μπορεί να υλοποιηθεί ζητώντας τη λίστα των διαδόχων. Με δεδομένο τον n-ιοστό κόμβο θα επιστραφεί το διάστημα [succ[n].ID, succ[n+1].ID]. Εξαίρεση σε αυτόν τον κανόνα αποτελεί ο προηγούμενος, το διάστημα του οποίου δεν μπορεί να καθοριστεί.
Pastry
Η λειτουργία *route* μπορεί εύκολα να υλοποιηθεί πάνω στη δρομολόγηση του Pastry. Το όρισμα *hint*, αν υπάρχει, εκτοπίζει / αντικαθιστά την αναζήτηση στον πίνακα δρομολόγησης. Η λειτουργία *range* υλοποιείται με βάση συγκρίσεις των nodeIds των μελών του συνόλου φύλλων. Η *local_lookup* μεταφράζεται σε μία απλή αναζήτηση του πίνακα δρομολόγησης και εάν η τιμή του *safe* είναι "true", η αναζήτηση θα διεξαχθεί στον περιορισμένο πίνακα δρομολόγησης του Pastry. Η λειτουργία *update* καλείται από μια αλλαγή στο σύνολο φύλλων του Pastry.
Tapestry
Η λειτουργία *route* είναι πανομοιότυπη με την κλήση *TapestryRouteMsg* του Tapestry API. Οι πίνακες δρομολόγησης του Tapestry βελτιστοποιούν την απόδοση και διατηρούν ένα μικρό (γενικά τρεις) σύνολο κόμβων, οι οποίοι είναι οι κοντινότεροι που έχουν το ίδιο πρόθεμα. Η κλήση *local_lookup* επιστρέφει τους κόμβους επόμενου βήματος, αφού έχει γίνει βελτιστοποίηση για την επιλογή τους. Προς το παρόν δεν υποστηρίζεται *safe* τρόπος δρομολόγησης. Η λειτουργία *range*

επιστρέφει ένα σύνολο από περιοχές, μία για κάθε συνδυασμό από επίπεδα στα οποία ο κόμβος μπορεί να δρομολογήσει. Η λειτουργία *update* καλείται, όταν ένας κόμβος λαμβάνει ένα μήνυμα acknowledged multicast για την εισαγωγή ενός κόμβου, ή όταν λαμβάνει μία αίτηση μετακίνησης αντικειμένου λόγω διαγραφής κόμβου.

Συμπερασματικά

Η κατάληξη σε ένα κοινό API δρομολόγησης με βάση το κλειδί (KBR API) περιπλέχτηκε από τη στενή σύνδεση μεταξύ των εφαρμογών και των συστημάτων αναζήτησης πάνω στα οποία αναπτύχθηκαν. Τα τρέχοντα σχήματα αντιγραφής τμημάτων αρχείων, ειδικά το σύνολο γειτόνων του Chord και του Pastry, είναι στενά συνδεδεμένα στον τρόπο με τον οποίο τα κλειδιά αντιστοιχίζονται στους κόμβους. Η υποστήριξη αποδοτικής αντιγραφής δεδομένων ανεξαρτήτως συστήματος αναζήτησης απαιτεί τις κλήσεις *range* και *replicaSet* που επιτρέπουν σε έναν κόμβο να αποφασίσει πού να αντιγράψει τα κλειδιά. Η κοινή πρακτική της προσωρινής αποθήκευσης των τμημάτων αρχείων κατά μήκος των πιθανών μονοπατιών αναζήτησης απαιτεί επίσης πρόσθετη ευελιξία στο API. Συγκεκριμένα απαιτείται ο μηχανισμός upcall που επιτρέπει στις διαδικασίες της εφαρμογής να εκτελούν κατά τη διάρκεια της αναζήτησης.

Το KBR API που μελετήθηκε προορίζεται να αποτελέσει ουδέτερη γλώσσα, έτσι ώστε να είναι δυνατή η μέγιστη πιθανή ευελιξία στην υλοποίηση των συστημάτων αναζήτησης. Χωρίς τον ακριβή προσδιορισμό του API σε μια γλώσσα, οι προγραμματιστές εφαρμογών δε θα είναι σε θέση να αλλάξουν εύκολα το σύστημα το οποίο χρησιμοποιούν. Αντ' αυτού, το API κατευθύνει τους προγραμματιστές να δομήσουν τις εφαρμογές τους κατά τέτοιο τρόπο ώστε να μπορούν να μεταφραστούν από το ένα σύστημα σε κάποιο άλλο με ελάχιστη προσπάθεια. [45]

Κεφάλαιο 4: Γενική αρχιτεκτονική προτεινόμενης υποδομής

Η υποδομή, η οποία προτείνεται σε αυτήν τη διπλωματική εργασία για την αντιμετώπιση των επιθέσεων DDoS, χρησιμοποιεί τα δίκτυα peer-to-peer για το σχηματισμό ενός Υπερκείμενου Δικτύου κόμβων που ανήκουν σε διάφορες διαχειριστικές περιοχές (domains). Οι κόμβοι αυτοί οργανώνονται αυτόματα και αλληλεπιδρούν κατά την εκδήλωση μίας επίθεσης DDoS, έτσι ώστε να παρέχουν από κοινού υπηρεσίες αναγνώρισης διαδρομής και συντονισμένης αντίδρασης ενάντια στην επίθεση.
Για να συμμετέχει ένας κόμβος στην παραπάνω συνεργατική υποδομή αρκεί να είναι μέλος του δικτύου peer-to-peer βάσει του οποίου υλοποιείται το παραπάνω υπερκείμενο δίκτυο και να συνδέεται με ένα Σύστημα Ανίχνευσης Επίθεσης (Intrusion Detection System – IDS), το οποίο παράγει μηνύματα IDMEF (Intrusion Detection Message Exchange Format) σε τοπικό επίπεδο (δηλαδή για τη συγκεκριμένη δικτυακή περιοχή). Η επικοινωνία του με την εφαρμογή peer-to-peer που εκτελείται στον τοπικό κόμβο γίνεται με το άνοιγμα ενός socket, ενώ οι πληροφορίες που στέλνει ακολουθούν την αυστηρά ορισμένη μορφή ενός μηνύματος IDMEF.
Η αρχιτεκτονική ενεργοποιείται με τη λήψη ενός μηνύματος IDMEF τύπου *Alert* για επίθεση DDoS από τον τοπικό κόμβο. Το IDS προειδοποιεί ότι πιθανότατα να εξελίσσεται επίθεση DDoS, η οποία πλήττει και τον τοπικό κόμβο. Στη συνέχεια, ο τοπικός κόμβος επεξεργάζεται την πληροφορία που του παρέχεται από το μήνυμα IDMEF για την επίθεση DDoS και δρομολογεί σύμφωνα με τα αποτελέσματα της επεξεργασίας το μήνυμα IDMEF στον κατάλληλο κόμβο του δικτύου peer-to-peer. Η επεξεργασία του μηνύματος IDMEF αναμένεται να δώσει ίδιο κόμβο προορισμού, όταν δύο ή παραπάνω κόμβοι πλήττονται από την ίδια επίθεση και το IDS τους δύναται να την ανιχνεύσει. Αυτό έχει ως αποτέλεσμα τη συγκέντρωση δεδομένων για την επίθεση DDoS σε συγκεκριμένο κόμβο του δικτύου peer-to-peer. Στη συνέχεια, ένας από τους αρχικούς κόμβους ανίχνευσης μπορεί να κάνει αίτηση προκειμένου να αποσταλούν σε αυτόν όλα τα δεδομένα που υπάρχουν για την επίθεση DDoS, την οποία το IDS του έχει υποδείξει ότι δέχεται. Επόμενο βήμα είναι να επεξεργαστεί τα ληφθέντα δεδομένα, για να καταλήξει σε συμπεράσματα για αυτήν όπως είναι το δέντρο της επίθεσης. Σε αυτό το σημείο ο κόμβος έχει διεξάγει το *traceback* της επίθεσης και είναι έτοιμος να λάβει δράση για την αντιμετώπισή της.
Στη συνέχεια αναλύεται βήμα προς βήμα η λειτουργία της προτεινόμενης υποδομής μέχρι την αναγνώριση της διαδρομής, ενώ στο Κεφάλαιο 6 προτείνονται ορισμένες τακτικές αντίδρασης. Το κομμάτι αυτό δεν υλοποιήθηκε ως μέρος αυτής της εργασίας.

4.1 Μηνύματα IDMEF

Η τυποποίηση IDMEF (Intrusion Detection Message Exchange Format) ανήκει στην κατηγορία των Internet-Drafts. Τα Internet-Drafts είναι αρχεία draft της IETF με μέγιστη διάρκεια ισχύος 6 μήνες, τα οποία μπορούν να ανανεωθούν, να αντικατασταθούν ή να απαρχαιωθούν από άλλα αρχεία οποιαδήποτε στιγμή. Κοινώς αποτελούν εξελισσόμενη δουλειά.
Στόχος της τυποποίησης IDMEF είναι ο ορισμός τύπων δεδομένων και διαδικασιών ανταλλαγής για το μοίρασμα χρήσιμης πληροφορίας ανάμεσα σε συστήματα ανίχνευσης και αντίδρασης σε επιθέσεις, και στα συστήματα διαχείρισης που αλληλεπιδρούν μαζί τους. Ένα αυτοματοποιημένο IDS μπορεί να χρησιμοποιήσει την

τυποποίηση IDMEF, για να αναφέρει προειδοποιήσεις για περιστατικά ασφαλείας. Η ανάπτυξη της τυποποίησης IDMEF καθιστά δυνατή τη διαλειτουργικότητα ανάμεσα σε συστήματα εμπορικά, ανοιχτού λογισμικού και ερευνητικά, και επιτρέπει στους χρήστες να τα συνδυάσουν σύμφωνα με τα πλεονεκτήματα και τα μειονεκτήματά τους, για να δημιουργήσουν τη βέλτιστη υλοποίηση. Προφανώς, τα μηνύματα IDMEF ακολουθούν τη διαδρομή που ενώνει το IDS και με το διαχειριστή στον οποίο αποστέλλονται οι συναγερμοί. Στη δική μας περίπτωση το ρόλο του διαχειριστή παίζει το υπερκείμενο δίκτυο peer-to-peer.
Η υλοποίηση IDMEF ουσιαστικά ορίζει ένα μοντέλο δεδομένων για την αναπαράσταση της πληροφορίας που εξάγεται από τα IDSs. Πρόκειται δηλαδή για μία αντικειμενοστραφή αναπαράσταση των δεδομένων περιστατικού που κατευθύνονται προς τους διαχειριστές ανίχνευσης εισβολής από τους αναλυτές ανίχνευσης εισβολής. Αυτή η αντικειμενοστραφής προσέγγιση προσθέτει την πολλή σημαντική ιδιότητα της ευελιξίας, έτσι ώστε να μπορούν να αναπαρασταθούν και απλοί και πολύπλοκοι συναγερμοί (ανάλογα με τις δυνατότητες του IDS) υπό ένα κοινό format.
Ο σχεδιασμός του μοντέλου δεδομένων οδηγείται από το περιεχόμενο. Αυτό σημαίνει ότι όποια καινούρια αντικείμενα εισάγονται έχουν ως σκοπό την πρόσθεση περιεχομένου και όχι την επισήμανση διαφορών ανάμεσα σε συναγερμούς. Αυτή η επιλογή είναι συνειδητή, καθώς η διαδικασία της κατηγοριοποίησης των τρωτών σημείων των υπολογιστών είναι εξαιρετικά δύσκολη και υποκειμενική.
Τέλος, το μοντέλο δεδομένων πρέπει να είναι σαφές. Με άλλα λόγια, ενώ επιτρέπεται στα IDSs να είναι λιγότερο ή περισσότερο ακριβή, δεν επιτρέπεται να παράγουν σε δύο διαφορετικούς συναγερμούς που αναφέρονται στο ίδιο γεγονός πληροφορίες που είναι αντιφατικές μεταξύ τους. Έτσι, για να μη μειωθεί η διαλειτουργικότητα των μηνυμάτων IDMEF, οι επιπλέον χρήσιμες πληροφορίες για ένα γεγονός πρέπει να τοποθετούνται στα πεδία που ανήκουν και όχι σε επιπλέον πεδία επέκτασης.

Η κωδικοποίηση του μοντέλου δεδομένων είναι σε γλώσσα XML (Extensible Markup Language). Η ευελιξία της XML την καθιστά μία πολύ καλή επιλογή για μία ποικιλία εφαρμογών συμπεριλαμβανομένου της υλοποίησης IDMEF. Μερικοί ακόμα λόγοι που συνηγορούν υπέρ αυτής της επιλογής είναι:

- Η XML επιτρέπει την ανάπτυξη μίας γλώσσας ειδικά για την περιγραφή συναγερμών ανίχνευσης εισβολής. Επιπλέον, ορίζει έναν καθορισμένο τρόπο περαιτέρω επέκτασης της γλώσσας.
- Τα εργαλεία λογισμικού για την επεξεργασία αρχείων XML είναι ευρέως διαθέσιμα είτε σε εμπορική μορφή είτε σε μορφή ανοιχτού λογισμικού. Όλα αυτά τα εργαλεία και τα APIs για το *parsing* και το *validation* των αρχείων XML διατίθενται σε πολλές γλώσσες (Java, C, C++, Tcl, Perl, Python, GNU Emacs Lisp). Έτσι, η IDMEF θα μπορεί να υιοθετηθεί γρήγορα και εύκολα.
- Η XML ικανοποιεί την απαίτηση της IDMEF τα μηνύματα να υποστηρίζουν πλήρη διεθνοποίηση και προσαρμογή σε τοπικά δίκτυα. Η τυποποίηση XML υποστηρίζει κωδικοποιήσεις UTF-8, UTF-16 και Unicode με αποτέλεσμα όλες οι εφαρμογές XML να είναι συμβατές με τις παραπάνω κυριαρχούσες κωδικοποιήσεις χαρακτήρων.
- Η XML ικανοποιεί την απαίτηση της IDMEF τα μηνύματα να υποστηρίζουν φιλτράρισμα και συγχώνευση. Αυτό το αναλαμβάνει η XSL (eXtensible Stylesheet Language) η οποία επιτρέπει σε μηνύματα XML να συνδυάζονται, να αναδιοργανώνονται και να αφαιρούνται.

Στη συνέχεια παρουσιάζουμε και αναλύουμε τη σημασία των πεδίων σε ένα ενδεικτικό μήνυμα IDMEF τύπου Alert (υπάρχει και ο τύπος *Heartbeat*, ο οποίος όμως δε χρησιμοποιείται στην περίπτωσή μας) που αφορά μία επίθεση DDoS. Προφανώς, αποτελεί μία ειδική περίπτωση μηνύματος IDMEF.

```
<?xml version="1.0" encoding="UTF-8"?>
<idmef:IDMEF-Message version="1.0"
                    xmlns:idmef="http://iana.org/idmef">

  <idmef:Alert messageid="abc123456789">
    <idmef:Analyzer analyzerid="bc-corr-01">
    </idmef:Analyzer>

    <idmef:CreateTime ntpstamp="0xbc72423b.0x00000000">
    2000-03-09T15:31:07Z
    </idmef:CreateTime>

   <idmef:Classification text="DDoS">
   </idmef:Classification>

   <idmef:Source ident="A">
   </idmef:Source>

  <idmef:Target ident="T">
    <idmef:Node>
      <idmef:Address category="ipv4-addr">
        <idmef:address>147.102.13.10</idmef:address>
        <idmef:netmask>255.255.255.255</idmef:netmask>
       </idmef:Address>
    </idmef:Node>
    <idmef:Service ident="TCP">
        <idmef:port>25
        </idmef:port>
    </idmef:Service>
  </idmef:Target>

  <idmef:AdditionalData type="string", meaning="Next Hop Domain(s)">
  B,C,D
  </idmef:AdditionalData>
  <idmef:AdditionalData type="string", meaning="Attack Packet Type(s)">
  SYN
  </idmef:AdditionalData>

 </idmef:Alert>
</idmef:IDMEF-Message>
```

Όπως αναφέρθηκε και προηγουμένως, ένα **IDMEF-Message** μπορεί να είναι τύπου Alert ή τύπου Heartbeat. Το χαρακτηριστικό *version* επισημαίνει την έκδοση του specification του IDMEF-Message, την οποία ακολουθεί το μήνυμα.
Ένα Alert IDMEF-Message στέλνεται κάθε φορά που ο αναλυτής ανιχνεύει ένα γεγονός, για το οποίο έχει ρυθμιστεί να ψάχνει. Ανάλογα με τον αναλυτή μπορεί να αντιστοιχεί σε ένα ή περισσότερα γεγονότα. Οι συναγερμοί λαμβάνουν χώρα ασύγχρονα σε απάντηση εξωτερικών γεγονότων. Το χαρακτηριστικό *messageid* είναι προαιρετικό και δίνει ένα μοναδικό αναγνωριστικό στο συναγερμό. Τα πεδία ενός Alert IDMEF-Message είναι τα ακόλουθα:

- Analyzer: πληροφορίες που προσδιορίζουν τον αναλυτή που παρήγαγε αυτό το μήνυμα. Το χαρακτηριστικό *analyzerid* δίνει πληροφορίες για το μηχάνημα στο οποίο βρίσκεται ο αναλυτής.
- CreateTime: η χρονική στιγμή που δημιουργήθηκε το μήνυμα. Το χαρακτηριστικό *ntpstamp* πρέπει να αναπαριστά την ίδια ημερομηνία και ώρα με το περιεχόμενο του πεδίου.
- Classification: το «όνομα» του συναγερμού. Στη δική μας εφαρμογή η οποία αφορά μόνο τις επιθέσεις DDoS το αντίστοιχο «όνομα» είναι "DDoS" (χαρακτηριστικό *text*).
- Source: η/οι πιθανή/πιθανές πηγή/πηγές του/των γεγονότος/ γεγονότων που οδήγησαν στο συναγερμό. Στη δική μας περίπτωση πρόκειται για το δικτυακό χώρο από τον οποίο προέρχεται η ύποπτη ροή πακέτων (χαρακτηριστικό *ident*).
- Target: ο/οι πιθανός/πιθανοί στόχος/στόχοι του/των γεγονότος/ γεγονότων που οδήγησαν στο συναγερμό. Στη δική μας περίπτωση πρόκειται για το δίκτυο θύμα της επίθεσης DDoS.
- AdditionalData: επιπλέον χρήσιμες πληροφορίες του αναλυτή που δεν ταιριάζουν σε κανένα από τα παραπάνω πεδία. Στη δική μας εφαρμογή τέτοια πληροφορία αποτελούν οι δικτυακοί χώροι επόμενου βήματος και ο/οι τύπος/τύποι των πακέτων της επίθεσης.

Χρειάζεται επιπλέον να παρατηρήσουμε ότι το πεδίο *Target* έχει δύο πεδία - παιδιά. Το πεδίο *Node* χρησιμοποιείται για να προσδιορίσει μηχανήματα ή άλλες δικτυακές συσκευές. Στην ενδεικτική περίπτωση δίνεται η IP διεύθυνση του κόμβου – στόχου (πεδίο *Address*).Το πεδίο *Netmask* χρειάζεται γενικά για την περίπτωση όπου παραπάνω από μία IP διεύθυνση αποτελεί στόχος. Το πεδίο *Service* περιγράφει υπηρεσίες δικτύου σε πηγές ή στόχους. Όταν το πεδίο προσδιορίζει το στόχο, αναφέρεται στην υπηρεσία του στόχου προς την οποία κατευθύνεται η ύποπτη δραστηριότητα Στην ενδεικτική περίπτωση πρόκειται για την υπηρεσία TCP (χαρακτηριστικό *ident*) στην πόρτα 25 (πεδίο *Port*). [48]

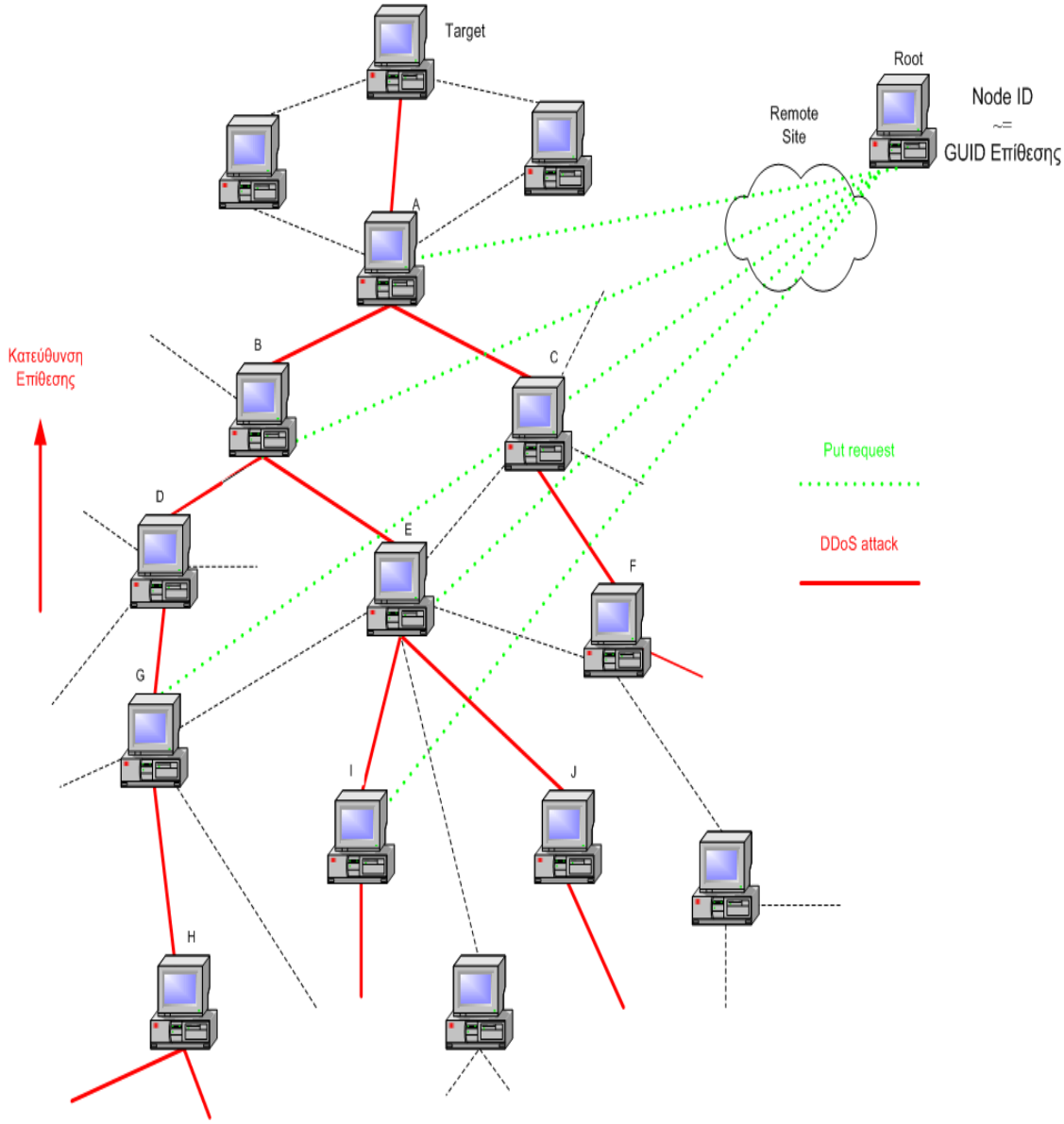
4.2 Δρομολόγηση προς τον κόμβο – ρίζα

Λόγω της κατανεμημένης φύσης των επιθέσεων DDoS δεν μπορεί μία και μόνο αναφορά από το IDS με στοιχεία για εξελισσόμενη επίθεση DDoS να φανεί χρήσιμη σε έναν κόμβο που ανήκει στη διαδρομή της επίθεσης και έχει ανιχνεύσει την ύποπτη κίνηση. Καταρχάς, αυτός ο κόμβος χρειάζεται περισσότερα στοιχεία, για να διαπιστώσει εάν όντως εξελίσσεται επίθεση DDoS. Ύστερα, για να αναλάβει δράση ενάντια στην επίθεση, χρειάζεται μία εικόνα της διαδρομής της επίθεσης.
Αυτή η εικόνα θα μπορούσε να σχηματιστεί από τις αναφορές – συναγερμούς των κόμβων που βρίσκονται κοντά στο στόχο της επίθεσης και αλληλεπιδρούν με IDSs ικανά να ανιχνεύσουν την επίθεση. Έτσι, το ζητούμενο πλέον είναι η συγκέντρωση

των αναφορών που αφορούν μία συγκεκριμένη επίθεση DDoS σε ένα σημείο. Η συνεργατική υποδομή που προτείνεται σε αυτή τη διπλωματική εργασία αυτοματοποιεί το διαχωρισμό των αναφορών που αναφέρονται σε μία συγκεκριμένη επίθεση DDoS από τις υπόλοιπες, και την επιλογή του σημείου συγκέντρωσης των πληροφοριών για το περιστατικό.
Η κεντρική ιδέα είναι η εξής: δύο ή περισσότερα μηνύματα IDMEF τύπου Alert αναφέρονται με μεγάλη πιθανότητα στην ίδια επίθεση DDoS, εάν έχουν κοινό/κοινούς στόχο/στόχους (χαρακτηριστικό *ident* σε πεδίο *Target*), κοινή υπηρεσία – στόχο (χαρακτηριστικό *ident* σε πεδίο *Service* του στόχου), κοινή πόρτα για την παραπάνω υπηρεσία (πεδίο *Port* του παραπάνω πεδίου *Service*) και κοινό/κοινούς τύπο/τύπους πακέτων επίθεσης (πεδίο *AdditionalData* με χαρακτηριστικό *meaning* = "Attack Packet Type(s)"). Κατακερματίζοντας (hashing) τις παραπάνω τιμές μπορούμε να πάρουμε ένα μοναδικό αναγνωριστικό για μία συγκεκριμένη επίθεση DDoS. Αυτή η τεχνική ανήκει στα δίκτυα peer-to-peer. Τα αντικείμενα που μοιράζονται σε ένα δίκτυο peer-to-peer αντιστοιχίζονται σε ένα μοναδικό αναγνωριστικό που προκύπτει από τον κατακερματισμό των περιεχομένων τους, των ονομάτων τους κ.α.
Με την παραγωγή ενός μοναδικού αναγνωριστικού για μία συγκεκριμένη επίθεση DDoS μπορούμε πλέον αυτοματοποιημένα να διαχωρίσουμε τις αναφορές που αναφέρονται σε μία συγκεκριμένη επίθεση DDoS από τις υπόλοιπες. Αυτό που απομένει είναι η δρομολόγησή τους σε έναν κόμβο – σημείο συγκέντρωσης. Είναι επιθυμητό αυτός ο κόμβος:

- να μην ανήκει στο μονοπάτι της επίθεσης, για να μην επηρεάζεται από αυτήν.
- να είναι διαφορετικός για κάθε ανεξάρτητη επίθεση DDoS που εμφανίζεται.
- να είναι γνωστός σε κάθε κόμβο που θέλει να του στείλει αναφορά από IDS χωρίς να χρειάζεται να χρησιμοποιηθεί κεντρική υπηρεσία (έτσι ώστε να μην αποτελεί ανάγκη η διαθεσιμότητά της).

Οι παραπάνω απαιτήσεις ικανοποιούνται, εάν κόμβος συγκέντρωσης των αναφορών για μία συγκεκριμένη επίθεση DDoS είναι ο κόμβος-ρίζα του μοναδικού αναγνωριστικού που παράγεται από τα χαρακτηριστικά που διαφοροποιούν τις επιθέσεις DDoS μεταξύ τους (στόχος, υπηρεσία, πόρτα υπηρεσίας, τύπος πακέτων). Υπενθυμίζουμε ότι κόμβος-ρίζα ενός αντικειμένου στα δίκτυα peer-to-peer είναι ο κόμβος με αναγνωριστικό (nodeID) πλησιέστερα (σύμφωνα με το μέτρο εγγύτητας που χρησιμοποιεί το δίκτυο peer-to-peer και σε σύγκριση με όλους τους ζωντανούς κόμβους που ανήκουν στο δίκτυο peer-to-peer) στο αναγνωριστικό (GUID) του αντικειμένου. Αυτή η τεχνική ανήκει επίσης στα δίκτυα peer-to-peer και προσομοιάζει μία από τις βασικότερες λειτουργίες των δικτύων peer-to-peer, τη δημοσίευση αντικειμένου. Έτσι, κάθε κόμβος που λαμβάνει μήνυμα συναγερμού από το IDS του για επίθεση DDoS επεξεργάζεται το μήνυμα, για να παράγει το αναγνωριστικό της επίθεσης και στη συνέχεια στέλνει **αίτηση PUT** με περιεχόμενο το μήνυμα IDMEF μέσω του peer-to-peer υπερκείμενου δικτύου στον κόμβο-ρίζα του αναγνωριστικού της επίθεσης. Τη δρομολόγηση του μηνύματος την αναλαμβάνει πλήρως το δίκτυο peer-to-peer. Προφανώς, το μόνο που χρειάζεται ο κόμβος για να τη χρησιμοποιήσει είναι να συμμετέχει στο δίκτυο peer-to-peer. Κατά αυτόν τον τρόπο, όλα τα μηνύματα IDMEF τύπου Alert που αναφέρονται στην ίδια επίθεση DDoS θα συγκεντρώνονται σε έναν συγκεκριμένο κόμβο.
Στην Εικόνα 17 φαίνεται η διαδικασία που μόλις περιγράφτηκε.

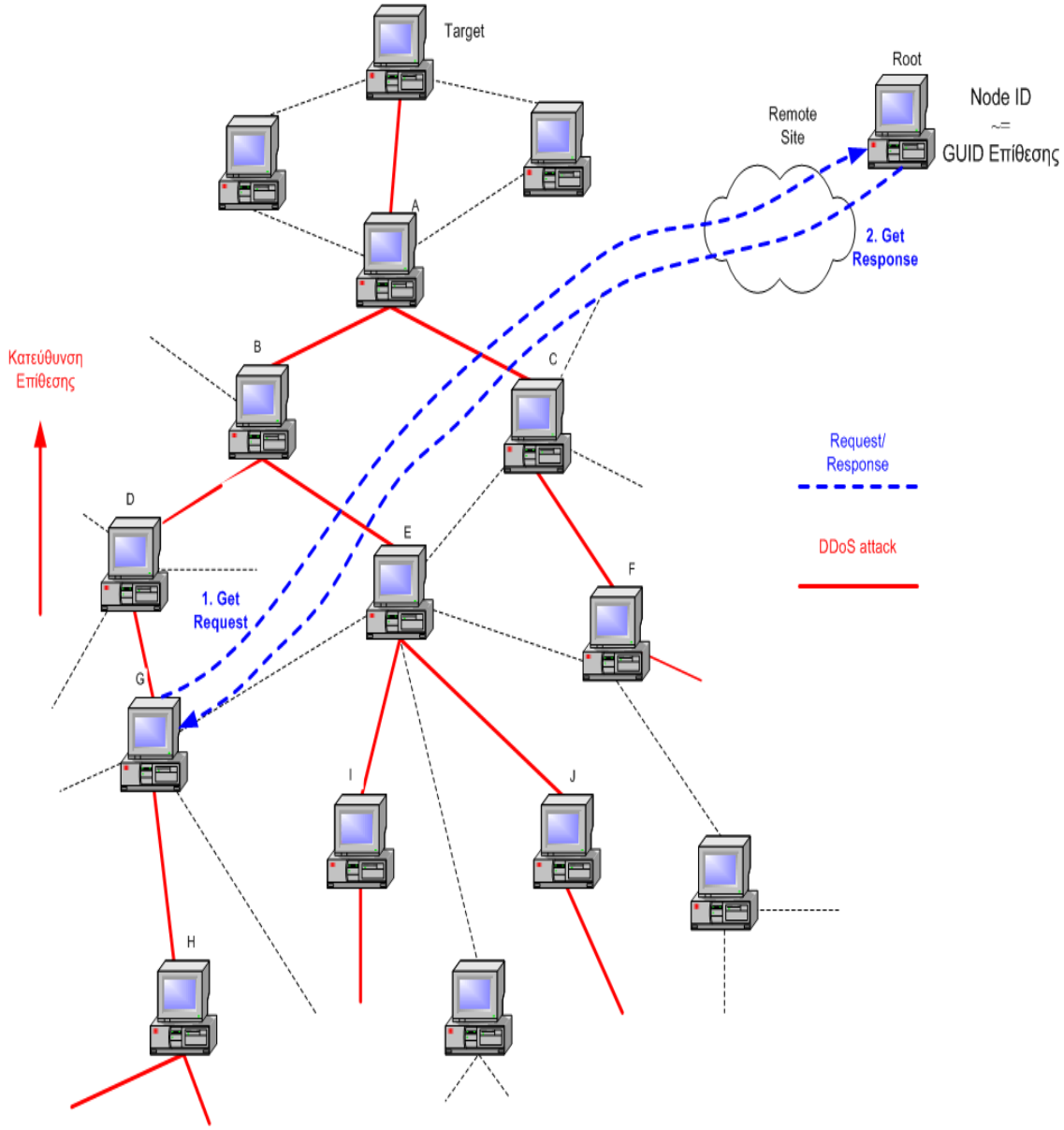


Εικόνα 17: Δρομολόγηση αιτήσεων PUT προς τον κόμβο - σημείο συγκέντρωσης της πληροφορίας για την επίθεση DDoS.

4.3 Αίτηση για την υπάρχουσα πληροφορία πάνω σε μία επίθεση

Ο κόμβος – σημείο συγκέντρωσης για μία επίθεση DDoS λειτουργεί μόνο σαν αποθηκευτικός χώρος για τα μηνύματα IDMEF. Ένας κόμβος μπορεί να ανακτήσει όλη αυτήν την πληροφορία, μόνο εάν με δική του πρωτοβουλία τη ζητήσει από τον αντίστοιχο κόμβο – ρίζα. Ένας κόμβος μπορεί να δρομολογήσει μία **αίτηση GET** με παράμετρο το αναγνωριστικό της επίθεσης DDoS (το οποίο ταυτίζεται με τον προορισμό της αίτησης) που τον ενδιαφέρει και να λάβει **απάντηση GET** με περιεχόμενο όλα τα μηνύματα IDMEF που αντιστοιχούν σε αυτό το αναγνωριστικό. Μία τέτοια προσέγγιση παρουσιάζει πλεονεκτήματα σε σύγκριση με την εναλλακτική να επεξεργάζονται τα δεδομένα και να υπολογίζεται το μονοπάτι της επίθεσης από τον κόμβο – σημείο συγκέντρωσης. Ωστόσο, αυτά τα πλεονεκτήματα τα παρουσιάζουμε σε επόμενο κεφάλαιο.

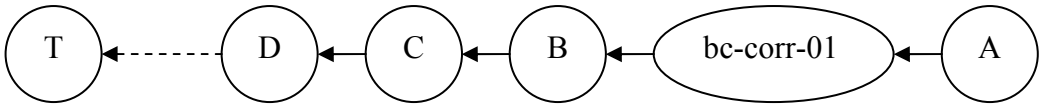
Υποθέτουμε ότι, για να προβεί ένας κόμβος σε μία τέτοια ενέργεια, έχει ήδη ανιχνεύσει την επίθεση DDoS για την οποία ζητά πληροφορίες. Η υπόθεση αυτή είναι πολύ λογική, αφού ένας κόμβος χρειάζεται αυτά τα δεδομένα μόνο εάν έχει πρόθεση να αντιδράσει και αυτή η πρόθεση αναμένεται να εκδηλωθεί μόνο από κόμβους που επηρεάζονται εν γνώσει τους από την επίθεση. Έτσι ένας τέτοιος κόμβος έχει ήδη λάβει μήνυμα Alert από το IDS του για τη δεδομένη επίθεση και έχει παράγει (και αποθηκεύσει) το αναγνωριστικό της.
Τα κριτήρια για να ενεργοποιήσει ένας κόμβος με μία αίτηση GET τους αντιδραστικούς μηχανισμούς του υπερκείμενου δικτύου εξαρτώνται καθαρά από τη διαχειριστική περιοχή στην οποία ανήκει ο κόμβος και στις πολιτικές ασφαλείας της. Μία τέτοια απόφαση μπορεί να παίρνεται αυτοματοποιημένα από το IDS, εφόσον αυτό έχει ρυθμιστεί για κάτι τέτοιο ή να ανήκει στη δικαιοδοσία του διαχειριστή δικτύου. Σε κάθε περίπτωση δεν εμποδίζεται η λειτουργία της συνεργατικής υποδομής.Η παρακάτω εικόνα παρουσιάζει τη διαδρομή μίας αίτησης GET και μίας απάντησης GET.



Εικόνα 18: Δρομολόγηση αίτησης GET και λήψη απάντησης GET.

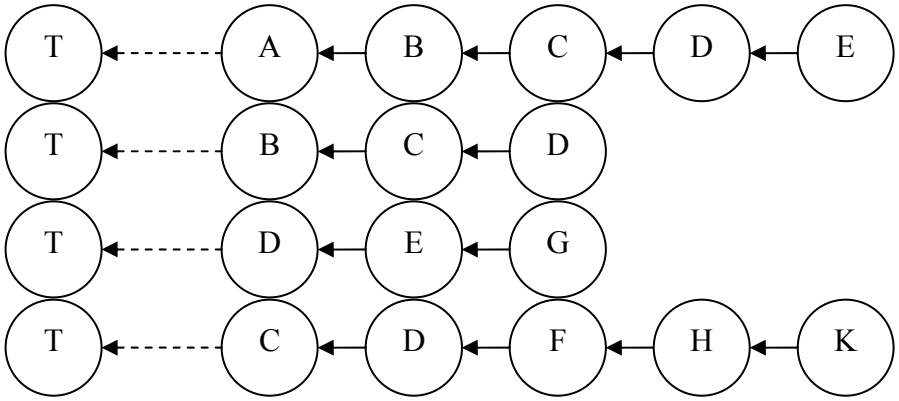
4.4 Επεξεργασία της πληροφορίας και αναγνώριση της διαδρομής

Το τελευταίο υλοποιημένο βήμα της υποδομής είναι η επεξεργασία της πληροφορίας που περιέχει η απάντηση GET από τον κόμβο που έκανε αίτηση GET. Τα μηνύματα IDMEF που λαμβάνει, εκτός από το κοινό περιεχόμενο των πεδίων που καθορίζουν το αναγνωριστικό της επίθεσης, μεταφέρουν την εξής επιπλέον χρήσιμη πληροφορία που αφορά τη διαδρομή της κίνησης: 1) το δίκτυο στο οποίο βρίσκεται ο αναλυτής, 2) το δίκτυο από το οποίο προέρχεται η κίνηση της επίθεσης, 3) τις δικτυακές περιοχές επόμενου βήματος για τη ροή της ύποπτης κίνησης. Το δίκτυο στο οποίο κατευθύνεται η κίνηση (δίκτυο-στόχος) είναι κοινό για τη λίστα των μηνυμάτων IDMEF που λαμβάνονται μετά από αίτηση GET.
Συνδυάζοντας τα παραπάνω δίκτυα σε σειρά σύμφωνα με τη φορά της κίνησης παίρνουμε έναν κλάδο (μία ροή) της κίνησης της επίθεσης DDoS. Επί παραδείγματι, από το μήνυμα IDMEF της παραγράφου 4.1 παίρνουμε τον ακόλουθο κλάδο.

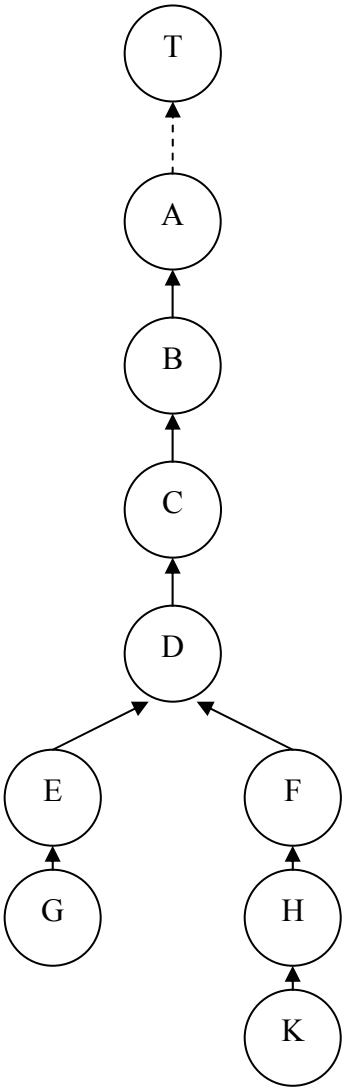


Εικόνα 19: Κλάδος της επίθεσης DDoS όπως προκύπτει από το μήνυμα IDMEF της παραγράφου 4.1. Η διακεκομμένη γραμμή επισημαίνει ότι ανάμεσα στο δίκτυο D και στο δίκτυο-στόχο T μπορεί να παρεμβάλλονται άλλες δικτυακές περιοχές.

Συνεπώς, από κάθε μήνυμα IDMEF εξάγεται και ένας κλάδος του δέντρου της επίθεσης. Ένας απλός αλγόριθμος (βλ. παράγραφο 5.7) που «τρέχει» στον τοπικό κόμβο συνδυάζει τους παραπάνω κλάδους συγχωνεύοντας τους κοινούς κόμβους και επιστρέφει το δέντρο της επίθεσης DDoS. Ρίζα αυτού του δέντρου είναι προφανώς ο στόχος της επίθεσης. Στην Εικόνα 20 δίνονται τέσσερις κλάδοι και το προκύπτον δέντρο.
Με την ολοκλήρωση αυτού του βήματος έχει γίνει η αναγνώριση της διαδρομής της επίθεσης κοντά στο στόχο της. Η γνώση της διαδρομής της επίθεσης αποτελεί απαραίτητη – και για αυτό το λόγο πολύτιμη – πληροφορία, για να υπάρξει αποτελεσματική αντίδραση. Μηχανισμοί αντίδρασης προς υλοποίηση προτείνονται στο Κεφάλαιο 6.



(a)



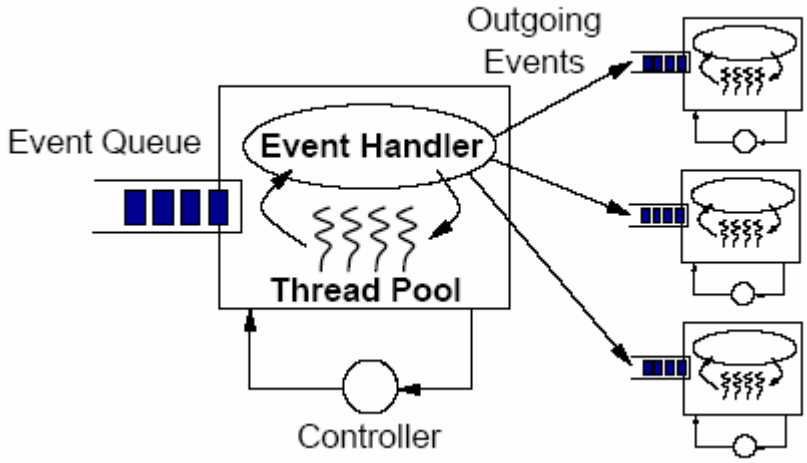
(b)

Εικόνα 20: (a) Τέσσερις κλάδοι που προκύπτουν από την πληροφορία που μεταφέρουν τέσσερα μηνύματα IDMEF, τα οποία αφορούν την ίδια επίθεση DDoS. (b) Το δέντρο που προκύπτει από την ενοποίηση των παραπάνω κλάδων.

Κεφάλαιο 5: Η εφαρμογή

5.1 Bamboo και SEDA

Το Bamboo είναι γραμμένο σύμφωνα με μία οδηγούμενη-από-γεγονότα (event-driven) μονονηματική (single-threaded) προγραμματιστική τεχνική. Για την ακρίβεια κληρονομεί τη δομή της αρχιτεκτονικής SEDA (Staged Event-Driven Architecture). Η αρχιτεκτονική SEDA έχει σχεδιαστεί, για να υποστηρίζει μαζικές ταυτόχρονες αιτήσεις και να απλουστεύσει τη δόμηση υπηρεσιών Διαδικτύου που απαιτούν υψηλό συγχρονισμό. Συνδυάζει στοιχεία του πολυνηματικού (multi-threaded) προγραμματισμού και του οδηγούμενου-από-γεγονότα προγραμματισμού. Στο SEDA οι εφαρμογές αποτελούνται από ένα δίκτυο *σταδίων* (stages) που οδηγούνται από γεγονότα και συνδέονται μεταξύ τους με *ουρές* (queues). Συγκεκριμένα, το στάδιο αποτελεί τη βασική μονάδα επεξεργασίας στο SEDA και είναι ένα ανεξάρτητο επιμέρους κομμάτι της εφαρμογής (βλ. Εικόνα 21). Αυτή η αρχιτεκτονική επιτρέπει στις εφαρμογές να μπορούν να αντεπεξέλθουν στο φόρτο εμποδίζοντας την υπεραπασχόληση των πόρων, όταν η ζήτηση ξεπερνά την ικανότητα εξυπηρέτησης. Το SEDA χρησιμοποιεί ένα σύνολο από δυναμικούς ελεγκτές πόρων (dynamic resource controllers), έτσι ώστε τα στάδια να παραμένουν στην κανονική περιοχή λειτουργίας τους ανεξάρτητα από τις διακυμάνσεις στο φόρτο. [37]



Εικόνα 21: Ένα στάδιο SEDA αποτελείται από μία εισερχόμενη ουρά γεγονότων (event queue), μία πισίνα νημάτων (thread pool) και έναν χειριστή γεγονότων (event handler) που παρέχεται από την εφαρμογή. Τη λειτουργία του σταδίου διαχειρίζεται ο ελεγκτής (controller), ο οποίος προσαρμόζει δυναμικά την ανάθεση των πόρων και τη δρομολόγηση. [37]

Σύμφωνα με την αρχιτεκτονική SEDA κάθε επιμέρους κομμάτι μιας εφαρμογής Bamboo αποτελεί ένα στάδιο. Η επικοινωνία επιτυγχάνεται με το πέρασμα γεγονότων σε κάθε στάδιο, ενώ η εφαρμογή συνολικά είναι μονονηματική. Ένα στάδιο εγγράφεται στο Bamboo και μπορεί επίσης να εγγραφεί στα γεγονότα που θέλει να λαμβάνει και να στέλνει μηνύματα σε άλλα στάδια και κόμβους. Τα στάδια της εφαρμογής μας αναλύονται στην παράγραφο 5.3.

5.2 Το API του Bamboo

Το API του Bamboo περιλαμβάνει δέκα κλήσεις. Αναφέρουμε επιγραμματικά τη λειτουργικότητα που έχει κάθε μία από αυτές δίνοντας περισσότερο έμφαση στις δύο τελευταίες, οι οποίες στηρίζουν την εφαρμογή μας.

- BambooLeafSetChanged: ενημερώνει για αλλαγή στο σύνολο φύλλων του τοπικού κόμβου.
- BambooNeighborInfo: παρέχει πληροφορίες για το σύνολο γειτόνων του τοπικού κόμβου.
- BambooReverseRoutingTableChanged: ενημερώνει για αλλαγές στον αντίστροφο πίνακα δρομολόγησης του τοπικού κόμβου.
- BambooRoutingTableChanged: ενημερώνει για αλλαγές στον πίνακα δρομολόγησης του τοπικού κόμβου
- BambooRouteContinue: συνεχίζει τη διαδικασία δρομολόγησης μετά από ένα upcall.
- BambooRouterAppRegReq: ζητά την εγγραφή μίας εφαρμογής στο σύστημα.
- BambooRouterAppRegResp: απαντά ως προς την επιτυχία της παραπάνω αίτησης.
- BambooRouteUpcall: καλείται στους ενδιάμεσους κόμβους της διαδρομής δρομολόγησης.
- BambooRouteInit (app_id, dest, intermediate_upcall, iter, payload): εκκινεί δρομολόγηση προς τον *dest*. Το μήνυμα προς αποστολή βρίσκεται στο *payload*. Ο κόμβος που είναι υπεύθυνος για το *dest* πρέπει να έχει εγγραφεί στην εφαρμογή με αναγνωριστικό *app_id* , για να λάβει το μήνυμα. Εάν η παράμετρος *intermediate_upcall* είναι true, θα στέλνεται στη συγκεκριμένη εφαρμογή σε κάθε ενδιάμεσο κόμβο της διαδρομής ένα γεγονός BambooRouteUpcall. Για να συνεχιστεί η διαδικασία της δρομολόγησης, πρέπει ο εκάστοτε ενδιάμεσος κόμβος να στέλνει ένα γεγονός BambooRouteContinue.
- BambooRouteDeliver (app_id, dest, est_rtt_ms, immediate_src, payload, src, tries, wait_ms): στέλνεται όταν η διαδικασία δρομολόγησης φτάνει τον κόμβο που είναι υπεύθυνος για το *dest*. [49]

5.3 Αρχείο ρύθμισης – MyNode.cfg

Το αρχείο ρύθμισης (configuration file) περιέχει τις τιμές των παραμέτρων που χρειάζονται τα στάδια της εφαρμογής κατά την εκτέλεσή τους. Το αρχείο ρύθμισης είναι τύπου XML και πολύ εύκολο στην κατανόηση. Το εργαλείο SandStorm για το parsing του αρχείου ρύθμισης κληρονομείται από το OceanStore [20]. Παρακάτω φαίνεται το αρχείο ρύθμισης της εφαρμογής μας.
Στην αρχή ορίζονται τα *global* ορίσματα. Κάθε στάδιο έχει πρόσβαση σε αυτά τα ορίσματα. Μοναδικό global όρισμα του MyNode.cfg είναι το *node_id*, το οποίο είναι της μορφής hostname:portnumber. Αυτό το ζεύγος (IP διεύθυνση, πόρτα) ορίζει ουσιαστικά πού θα «τρέχει» ο εικονικός κόμβος Bamboo.
Στη συνέχεια, έχουμε το "stage configuration". Τα στάδια τα οποία χρειαζόμαστε, για να υλοποιήσουμε το υπερκείμενο δίκτυο που περιγράφεται στο Κεφάλαιο 4 είναι: Network, Router, DataManager, StorageManager, Dht, WebInterface, ApplicationStage. Οι μεταβλητές ${hport}, ${dbfile}, ${mport}, ${sport} παίρνουν τιμές κατά την εκτέλεση του Perl script που δίνεται στο τέλος του Κεφαλαίου (βλ. παράγραφο5.8).

MyNode.cfg

```
<sandstorm>
  <global>
    <initargs>
      node_id 147.102.13.24: ${hport}
    </initargs>
  </global>
  <stages>
    <Network>
      class bamboo.network.Network
      <initargs>
      </initargs>
    </Network>

    <Router>
      class bamboo.router.Router
      <initargs>
        gateway_count 1
        gateway_0     147.102.13.24:3630
      </initargs>
    </Router>

    <DataManager>
      class bamboo.dmgr.DataManager
      <initargs>
        debug_level          0
        merkle_tree_expansion  2
      </initargs>
    </DataManager>

    <StorageManager>
      class bamboo.db.StorageManager
      <initargs>
        homedir       ${dbfile}
      </initargs>
    </StorageManager>

    <Dht>
      class bamboo.dht.Dht
      <initargs>
        debug_level 1
      </initargs>
    </Dht>

    <WebInterface>
      class bamboo.www.WebInterface
      <initargs>
        storage_manager_stage StorageManager
      </initargs>
```

```
    </WebInterface>

    <LastStage>
      class bamboo.api.LastStage
      <initargs>
        debug_level 1
        port_from_IDS ${mport}
        port_for_GUID ${sport}
      </initargs>
    </LastStage>
  </stages>
</sandstorm>
```

Τα στάδια *Network* και *Router* χρειάζονται για την αποστολή μηνυμάτων πάνω από το σύστημα peer-to-peer Bamboo. Για να προσχωρήσει ένας κόμβος στο σύστημα peer-to-peer Bamboo πρέπει να ορίσουμε τουλάχιστον έναν (όρισμα *gateway_count*) *gateway* κόμβο. Αυτός θα είναι ο πρώτος κόμβος του συστήματος.
Το στάδιο *DataManager* διαχειρίζεται τα δεδομένα που είναι αποθηκευμένα στους κόμβους του Bamboo. Γενικά, το όρισμα *debug_level* ενεργοποιεί τα μηνύματα debugging, όταν είναι μεγαλύτερο του 0 και τα απενεργοποιεί, όταν είναι 0. Το όρισμα *merkle_tree_expansion* καθορίζει την έκταση του δέντρου Merkle των κλειδιών που αποθηκεύονται στον κόμβο. Ένα δέντρο Merkle είναι ένα δέντρο όπου η τιμή που συνδέεται με έναν κόμβο είναι μία μίας κατεύθυνσης συνάρτηση των τιμών των παιδιών του κόμβου [50] και είναι η δομή που χρησιμοποιείται για την αποθήκευση των κλειδιών σε έναν κόμβο του Bamboo. Υπενθυμίζουμε ότι το σύστημα peer-to-peer Bamboo είναι ένας κατανεμημένος πίνακας κατακερματισμού (DHT) που αποθηκεύει ζεύγη (κλειδί,τιμή). Κάθε κόμβος στο δέντρο Merkle έχει το πολύ $2^{expansion}$ παιδιά.
Το στάδιο *StorageManager* υλοποιεί ένα ασύγχρονο interface στη βάση δεδομένων BerkeleyDB, η οποία αποθηκεύει τα δεδομένα του κόμβου. Στον κατάλογο *homedir* βρίσκουμε τα logs της BerkeleyDB.
Το στάδιο *Dht* υλοποιεί το στρώμα DHT για το Bamboo.
Το στάδιο *WebInterface* υλοποιεί ένα web interface στην πληροφορία δρομολόγησης.
Ο ορισμός ενός καινούριου σταδίου ξεκινά με ένα tag που περιέχει το όνομα του σταδίου ("ApplicationStage") ακολουθούμενο από την κλάση, έτσι ώστε ο "stage loader" να ξέρει ποια κλάση να φορτώσει. Για το στάδιο *ApplicationStage* έχουμε τρία ορίσματα:

- debug_level
- port_from_IDS: η πόρτα επικοινωνίας με το IDS
- port_for_GUID: η πόρτα επικοινωνίας για αίτηση GET που ενεργοποιείται από εξωτερικό παράγοντα

5.4 Κλάση ApplicationStage

5.4.1 Δομή

Ένα καινούριο στάδιο σε μία εφαρμογή Bamboo πρέπει να υλοποιεί το interface seda.sandStorm.api.EventHandlingIF, έτσι ώστε να συμμετέχει στο χειρισμό των γεγονότων. Επιπλέον, ένα καινούριο στάδιο μπορεί να κληρονομήσει από την κλάση bamboo.util.StandardStage. Κάθε στάδιο μπορεί να χωριστεί σε τρία κομμάτια. Το

κάθε ένα από αυτά αναφέρεται σε διαφορετική χρονική στιγμή της διάρκεια ζωής ενός σταδίου Bamboo.

- ApplicationStage ( ) (constructor)
- init (ConfigDataIF config)
- handleEvent (QueueElementIF elem)

Ο *Constructor* χρησιμοποιείται για την αρχικοποίηση των δεδομένων του σταδίου (όπως σε κάθε πρόγραμμα Java). Στο Bamboo χρειάζεται επιπλέον κάθε στάδιο να εγγράφεται στα γεγονότα από τα οποία θέλει να ακούει. Έτσι, μέσα στον constructor συμπληρώνονται οι δομές πινάκων (*TypeTable[], event_types[]*) που παρέχονται από το StandardStage με τα γεγονότα που χρειάζεται να ακούει το στάδιο της εφαρμογής. Η εγγραφή στα γεγονότα γίνεται στη συνάρτηση *init( )*. Αυτή η συνάρτηση καλείται κατά την εκκίνηση της εκτέλεσης του κόμβου και παίρνει τις παραμέτρους ρύθμισης του σταδίου από το αρχείο ρύθμισης (cfg). Τέλος, η *handleEvent( )* καλείται από το Bamboo κάθε φορά που φτάνει ένα γεγονός, για το οποίο έχει εγγραφεί το στάδιο και το οποίο πρέπει να χειριστεί. Αυτός είναι ο πυρήνας του σταδίου.

5.4.2 Διασχίζοντας τον κώδικα

Η κλάση ApplicationStage κληρονομεί από την κλάση StandardStage ένα logging interface και ένα πλήθος χρήσιμων συναρτήσεων για το χειρισμό του σταδίου και την υλοποίηση της εφαρμογής.
Κατά τη δρομολόγηση μηνυμάτων στο Bamboo, και γενικά στα περισσότερα υπερκείμενα δίκτυα, χρειάζεται να προσδιορίζουμε σε ποια εφαρμογή ανήκουν. Αυτή τη λειτουργικότητα έχει το *app_id* (application ID), το οποίο χρησιμοποιείται στο Bamboo όπως ο TCP/UDP αριθμός πόρτας στα δίκτυα IP. Για να λάβουμε ένα μοναδικό αναγνωριστικό για κάθε εφαρμογή κατακερματίζουμε το όνομα της κλάσης που υλοποιεί την εφαρμογή, δηλαδή το στάδιο εφαρμογής.
Κατά την αρχικοποίηση του σταδίου εφαρμογής παίρνουμε τα δεδομένα από το αρχείο ρύθμισης. Το parsing των δεδομένων γίνεται από το seda.sandStorm.
Πολύ σημαντικό στοιχείο για ένα στάδιο εφαρμογής είναι ο πίνακας *event_types[]*. Σε αυτόν τον πίνακα αποθηκεύουμε τις κλάσεις των γεγονότων που θέλουμε να «ακούμε». Μόνο αυτά τα γεγονότα θα προωθούνται στη συνάρτηση *handleEvent( )*.

Η εφαρμογή μας θέλει να «ακούει» στους ακόλουθους τύπους γεγονότων:

- StagesInitializedSignal: λαμβάνεται, όταν όλα τα στάδια του κόμβου έχουν αρχικοποιηθεί. Είναι το σήμα ότι μπορούμε πλέον να χρησιμοποιήσουμε το σύστημα Bamboo.
- BambooRouterAppRegResp (βλ. παράγραφο 5.2)
- BambooRouteDeliver (βλ. παράγραφο 5.2)
- PutResp: η απάντηση ως προς την επιτυχία της αίτησης PutReq (βλ. παρακάτω).
- GetResp: η απάντηση ως προς την επιτυχία της αίτησης GetReq (βλ. παρακάτω).
- GetAlarm: το σήμα για την εκκίνηση αναζήτησης πληροφοριών (αίτηση GET) για την τελευταία επίθεση DDoS που ανιχνεύτηκε από το IDS.
- MsgFromIDS: λαμβάνεται, όταν το IDS έχει ανιχνεύσει επίθεση DDoS και ο τοπικός κόμβος έχει λάβει το μήνυμα IDMEF, το οποίο περιγράφει την επίθεση, από ένα δεδομένο socket.

- MsgForGet: λαμβάνεται, όταν κάποιος εξωτερικός παράγοντας επιθυμεί να ξεκινήσει ο τοπικός κόμβος αναζήτηση πληροφοριών για την επίθεση DDoS με αναγνωριστικό αυτό που αποστέλλεται μέσω δεδομένου socket.
- InMsg: δημιουργείται, όταν λαμβάνεται γεγονός BambooRouteDeliver από τον τοπικό κόμβο, για να εξακριβωθεί τι είδους πληροφορία μεταφέρει (αίτηση PUT, αίτηση GET ή απάντηση GET).

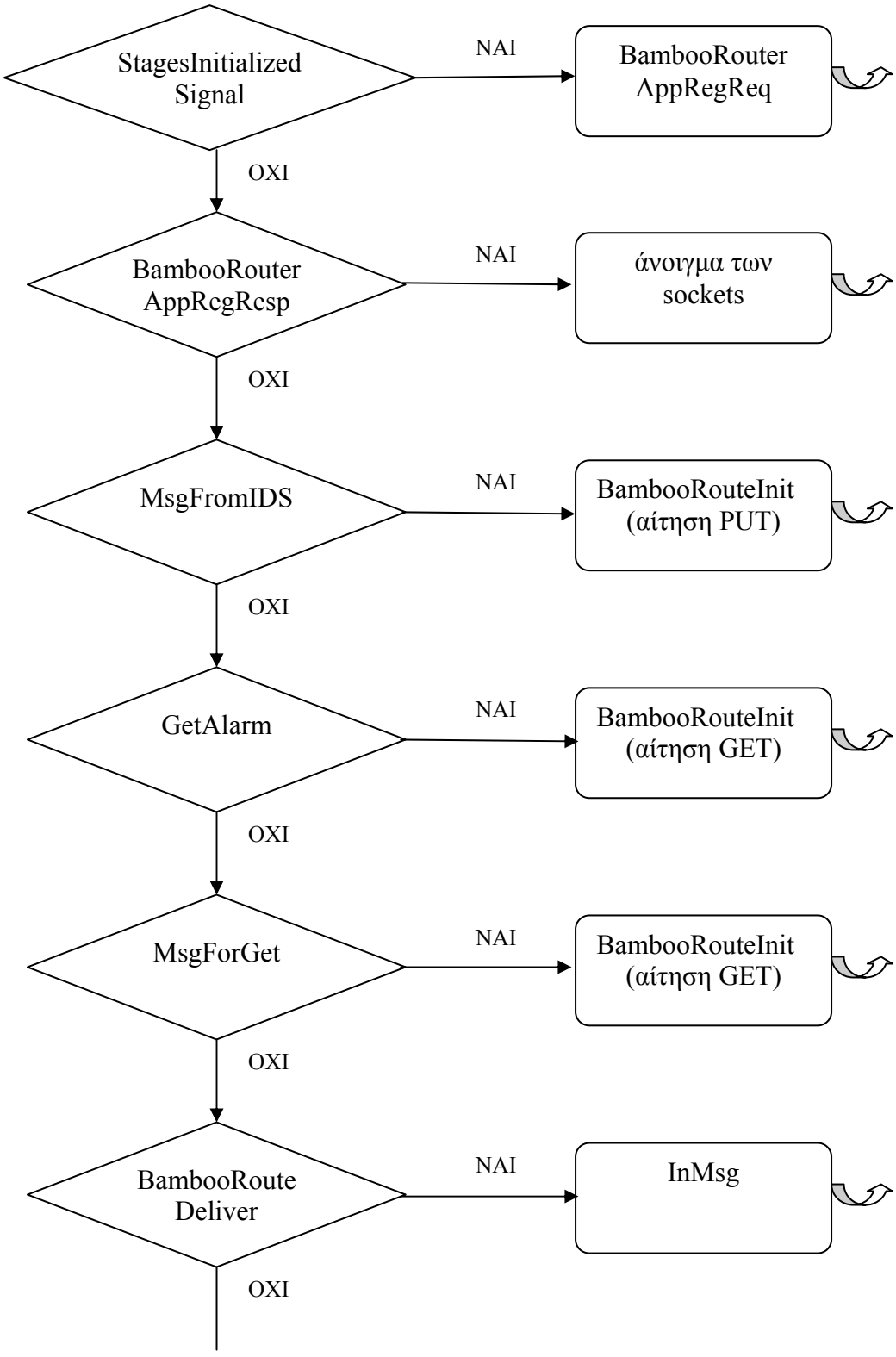
Τέλος, η handleEvent (QueueElementIF elem) καλείται κάθε φορά που εμφανίζεται στον τοπικό κόμβο γεγονός για το οποίο έχει εγγραφεί. Αυτή η συνάρτηση χειρίζεται κάθε γεγονός ανάλογα με τον τύπο του. Έτσι, ανάλογα με τον τύπο του γεγονότος που λαμβάνει η handleEvent( ) κάνει τα εξής:

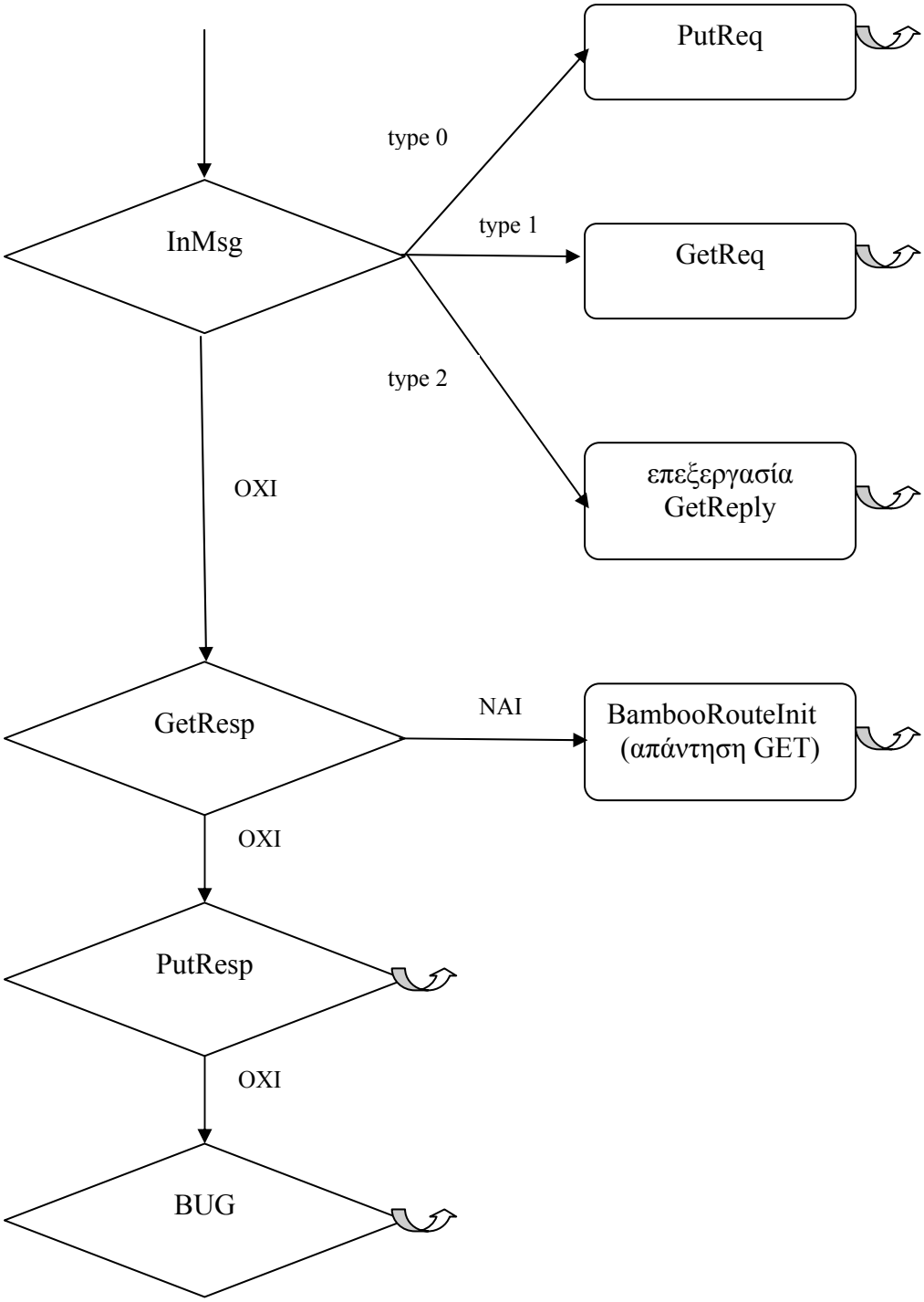
- StagesInitializedSignal: ζητά εγγραφή της εφαρμογής (του αναγνωριστικού της, application ID) από το Bamboo, για να λαμβάνει τα γεγονότα που απευθύνονται στη συγκεκριμένη εφαρμογή.
- BambooRouterAppRegResp: ξεκινά η εκτέλεση δύο νημάτων. Το πρώτο ανοίγει ένα socket που «ακούει» για νέο μήνυμα IDMEF από το IDS, ενώ το δεύτερο ανοίγει ένα socket που «ακούει» σε αίτηση αναζήτησης πληροφοριών για συγκεκριμένη επίθεση DDoS.
- BambooRouteDeliver: λαμβάνει το περιεχόμενο, την πηγή και τον προορισμό του γεγονότος (τιμές των *payload*, *dest*, *src*) και στέλνει γεγονός τύπου InMsg.
- GetResp: δημιουργεί ένα XML από τα δεδομένα (μηνύματα IDMEF) που έχουν ανακτηθεί και το στέλνει σαν *payload* στον peer που έστειλε την αίτηση GET.
- GetAlarm: στέλνει αίτηση GET για συγκεκριμένη επίθεση DDoS
- MsgFromIDS: στέλνει αίτηση GET για συγκεκριμένη επίθεση DDoS
- InMsg: ανάλογα με τον τύπο του InMsg, 1) βάζει δεδομένα (μήνυμα IDMEF) στον DHT με αίτηση PutReq, 2) ανακτά δεδομένα (μηνύματα IDMEF για συγκεκριμένη επίθεση DDoS) με αίτηση GetReq, 3) παράγει το δέντρο της επίθεσης από τη λίστα των μηνυμάτων IDMEF για συγκεκριμένη επίθεση DDoS.

Για την καλύτερη κατανόηση του κώδικα επισημαίνουμε ακόμα ότι: 1) *sink* είναι η ουρά γεγονότων ενός σταδίου, 2) καλώντας *dispatch (message)* το μήνυμα περνάει στον λεγόμενο *classifier*, ο οποίος αποφασίζει ποιο στάδιο – στον τοπικό κόμβο ή σε οποιονδήποτε άλλο κόμβο του συστήματος – πρέπει να λάβει το μήνυμα, 3) τα GUIDs είναι αντικείμενα τύπου *BigInteger*. Στην Εικόνα 21 φαίνεται το διάγραμμα ροής της συνάρτησης handleEvent( ), η οποία αποτελεί τον πυρήνα της εφαρμογής.

5.5 Νέοι τύποι γεγονότων

Ένας νέος τύπος γεγονότος πρέπει να υλοποιεί το interface *QueueElementIF*. Αυτό το interface του seda.sandstorm.api αναπαριστά ένα γεγονός που εισέρχεται και εξέρχεται από μία ουρά γεγονότων. Είναι ένα άδειο interface, του οποίου στιγμιότυπα πρέπει να υλοποιούν οι εφαρμογές, για να αναπαραστήσουν διαφορετικά γεγονότα. Στην υλοποίηση της εφαρμογής μας χρειάστηκαν οι επόμενοι τέσσερις νέοι τύποι γεγονότων.


StagesInitialized Signal
NAI
BambooRouter AppRegReq
OXI
BambooRouter AppRegResp
NAI
άνοιγμα των sockets
OXI
MsgFromIDS
NAI
BambooRouteInit (αίτηση PUT)
OXI
GetAlarm
NAI
BambooRouteInit (αίτηση GET)
OXI
MsgForGet
NAI
BambooRouteInit (αίτηση GET)
OXI
BambooRoute Deliver
NAI
InMsg
OXI



Εικόνα 21: Διάγραμμα ροής της λειτουργίας της handleEvent( ). Πηγή για το διάγραμμα ροής είναι η ουρά γεγονότων του σταδίου ApplicationStage. Το μπλοκ βέλος επισημαίνει το τέλος του χειρισμού του γεγονότος και την επιστροφή στην αρχή του διαγράμματος ροής για νέο γεγονός.

- Κλάση MsgFromIDS

Ο constructor της κλάσης παίρνει το stream των δεδομένων από το socket που έχει οριστεί για την επικοινωνία του τοπικού κόμβου με το IDS και φτιάχνει το αρχείο XML τύπου IDMEF. Επιπλέον, είναι υπεύθυνη για το parsing του XML. Η συνάρτηση getGUID( ) επιστρέφει το αναγνωριστικό της επίθεσης DDoS που αναφέρεται από το μήνυμα IDMEF κατακερματίζοντας τις τιμές των κατάλληλων πεδίων (βλ. παράγραφος 4.2). Το αρχείο XML ταξιδεύει μέσα στο δίκτυο υπό μορφή αντικειμένου Payload, αφού μετατραπεί πρώτα σε *String*.

- Κλάση GetAlarm

Η παραπάνω κλάση είναι άδεια και χρησιμεύει μόνο σαν σήμα που στέλνεται μετά το πέρασμα ενός χρονικού διαστήματος από την τελευταία αίτηση PUT, για να σταλεί μία αίτηση GET με στόχο το τελευταίο αναγνωριστικό (GUID) επίθεσης.

- Κλάση MsgForGet

Ο constructor της κλάσης παίρνει το stream των δεδομένων από το socket που έχει οριστεί για την επικοινωνία του τοπικού κόμβου με κάποιον εξωτερικό παράγοντα, ο οποίος έχει το δικαίωμα σύμφωνα με την πολιτική της διαχειριστικής περιοχής να εκκινήσει αίτηση GET, και δημιουργεί ένα στιγμιότυπο *BigInteger*. Αυτός ο αριθμός αποτελεί το αναγνωριστικό (GUID), για το οποίο ζητείται να σταλεί αίτηση GET.

- Κλάση InMsg

Ο constructor της κλάσης παίρνει το *payload* από ένα γεγονός τύπου BambooRouteDeliver και υλοποιεί τη συνάρτηση, η οποία αποφαίνεται για το εάν πρόκειται για αίτηση PUT, GET ή απάντηση GET(getType ( )).

5.6 Κλάση Payload

Η κλάση Payload υλοποιεί το interface *ostore.util.QuickSerializable*. Το περιεχόμενο των μηνυμάτων που θέλουμε να ταξιδέψουν μέσα στο υπερκείμενο δίκτυο είναι αντικείμενα της κλάσης Payload. Το περιεχόμενο πρέπει να διατάσσεται σειριακά (υλοποίηση συνάρτησης serialize()) πριν την αποστολή και να αναδιατάσσεται στη λήψη.

5.7 Κλάση GetReply

Η κλάση GetReply είναι παιδί της κλάσης InMsg και όπως φαίνεται από το όνομά της, αναφέρεται στην περίπτωση που το εισερχόμενο μήνυμα αποτελεί απάντηση σε αίτηση GET. Σε αυτήν την περίπτωση το περιεχόμενο του μηνύματος είναι ένα αρχείο XML με μία λίστα από στοιχεία (elements) "idmef:IDMEF-Message". Η συνάρτηση getList( ) εκτελεί το parsing του XML και επιστρέφει έναν κλάδο της επίθεσης DDoS από κάθε στοιχείο "idmef:IDMEF-Message". Ο κλάδος υλοποιείται με συνδεδεμένη λίστα από αντικείμενα *Node*. Η συνάρτηση getTree( ) φτιάχνει το δέντρο (αντικείμενο *Tree*) της επίθεσης συγχωνεύοντας κάθε έναν κλάδο με το ήδη υπάρχον δέντρο. Αρχικά το δέντρο έχει μόνο ρίζα (το στόχο της επίθεσης). Η συγχώνευση ενός κλάδου στο δέντρο είναι εργασία της συνάρτησης merge( ). Η merge( ) εκτελεί έναν λογικό αλγόριθμο, τον οποίο παραθέτουμε στη συνέχεια υπό μορφή ψευδοκώδικα. Αρχικά κόμβος^{A} είναι η ρίζα του δέντρου και κόμβος^{B} είναι η κεφαλή της λίστας.

```
εάν υπάρχει ο  κόμβος^B στο δέντρο κάτω από τον κόμβο^A {
        κόμβος^A ← κόμβος του δέντρου που ταυτίζεται με κόμβο^B
}
όσο υπάρχει κόμβος^B {
        εάν δεν υπάρχει το παιδί του κόμβου^A
                κρέμασε τη λίστα από τον κόμβο^B (και έπειτα) κάτω από τον κόμβο^A
        αλλιώς {
                εάν το παιδί του κόμβου^A είναι ίδιο με τον κόμβο^B {
                        κόμβος^A ← παιδί του κόμβου^A
                        κόμβος^B ← επόμενος του κόμβου^B στη λίστα
                }
                αλλιώς
                        παιδί του κόμβου^A ← επόμενο παιδί του κόμβου^A
        }
}
```

5.8 Ενεργοποίηση πολλών κόμβων

Εκτελώντας την εντολή **./bin/run-java bamboo.lss.DustDevil MyNode.cfg** ξεκινά ένας κόμβος. Συγκεκριμένα, η κλάση DustDevil χρησιμοποιεί το sandStorm για το parsing του αρχείου ρύθμισης MyNode.cfg και ξεκινούν τα στάδια που περιλαμβάνονται σε αυτό. Ωστόσο, για να ελέγξουμε τη σωστή λειτουργία της υποδομής χρειαζόμαστε ένα δίκτυο Bamboo τουλάχιστον 10 κόμβων. Για αυτό το λόγο καταφύγαμε στη συγγραφή ενός Perl script, το οποίο ξεκινά όσους κόμβους επιθυμούμε και δίνει διαφορετικές τιμές στις χρησιμοποιούμενες από κάθε εικονικό κόμβο πόρτες.

runnodes.pl

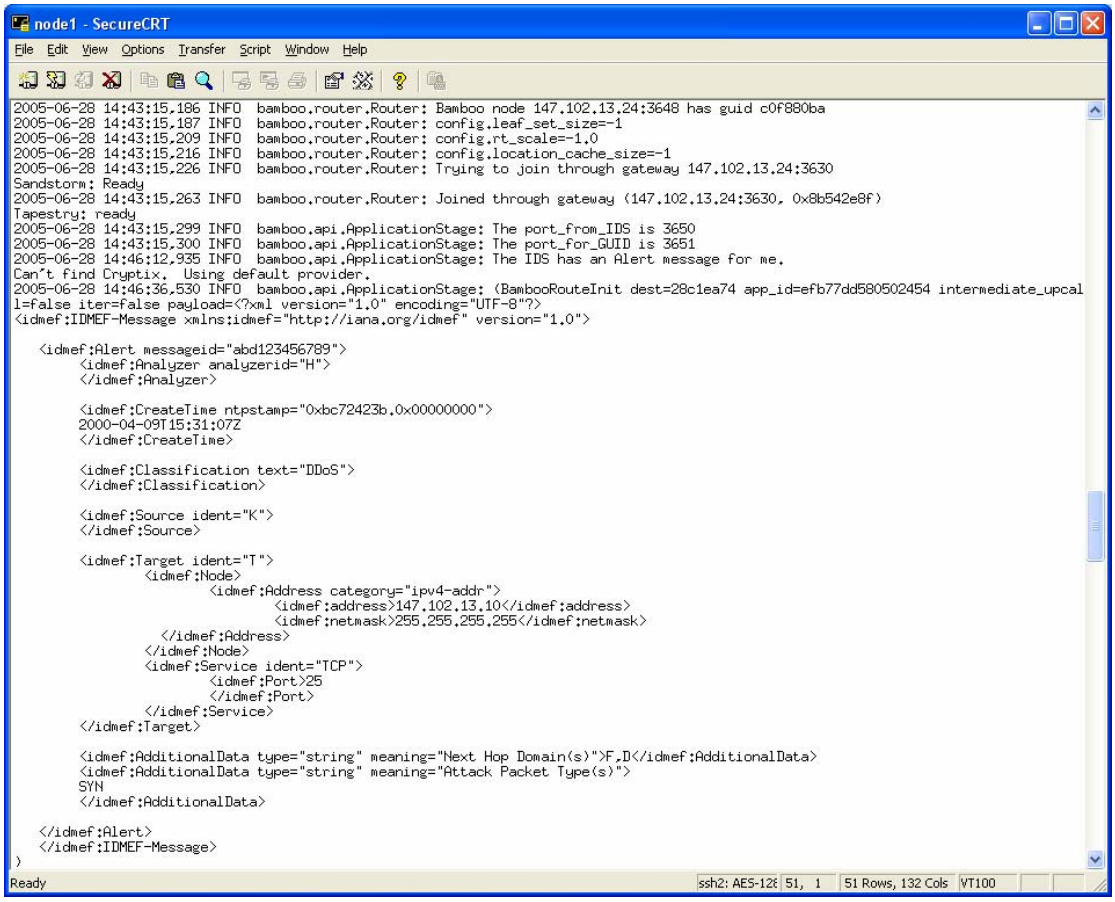
```
#!/usr/bin/perl -w

$i = 0;

sub fork_process {
   my $log = shift( @_ );

   my $pid = fork();
   if( $pid > 0 ) {
        return $pid;

   } elsif( defined $pid ) {
        # child; run the named process..

        open STDOUT, ">$log" or die( "can't redirect STDOUT");
        open STDERR, ">&STDOUT" or die( "can't redirect STDERR");

        # now exec the process as specified.
        exec (@_);
   }

   # should never reach this point.
```

```
   &die( "fork failed" );
}

while ( $i < ($ARGV[0]*4) )
{
        $hport = 3630 + $i;
        $mport = 3632 + $i;
        $sport = 3633 + $i;
        $dbfile = "/home/kiki/db-blocks/sm-blocks-".$i;
        $outfile = "/home/kiki/file".$i;


        open (CFG_IN, "/home/kiki/mycfg.cfg") or die "Could not open mycfg.cfg";
        open (CFG_OUT, ">/home/kiki/my-$i.cfg") or die "Could not open file";

        my %variable_map = ("hport"  => $hport,
                "dbfile" => $dbfile,
                "mport"  => $mport,
                        "sport" => $sport);

        while (<CFG_IN>) {
        if (m/\$\{([^}]+)\}/) {
                        my $value = $variable_map{$1};
                if (defined $value) {
                s/\$\{([^}]+)\}/$value/;
                        }
        }
        print CFG_OUT $_;
        }

        close (CFG_IN);
        close (CFG_OUT);

        my $pid = &fork_process ($outfile,"/home/kiki/bamboo/bin/run-java",
"bamboo.lss.DustDevil", "/home/kiki/my-$i.cfg");

        $i = $i + 4;
}
```

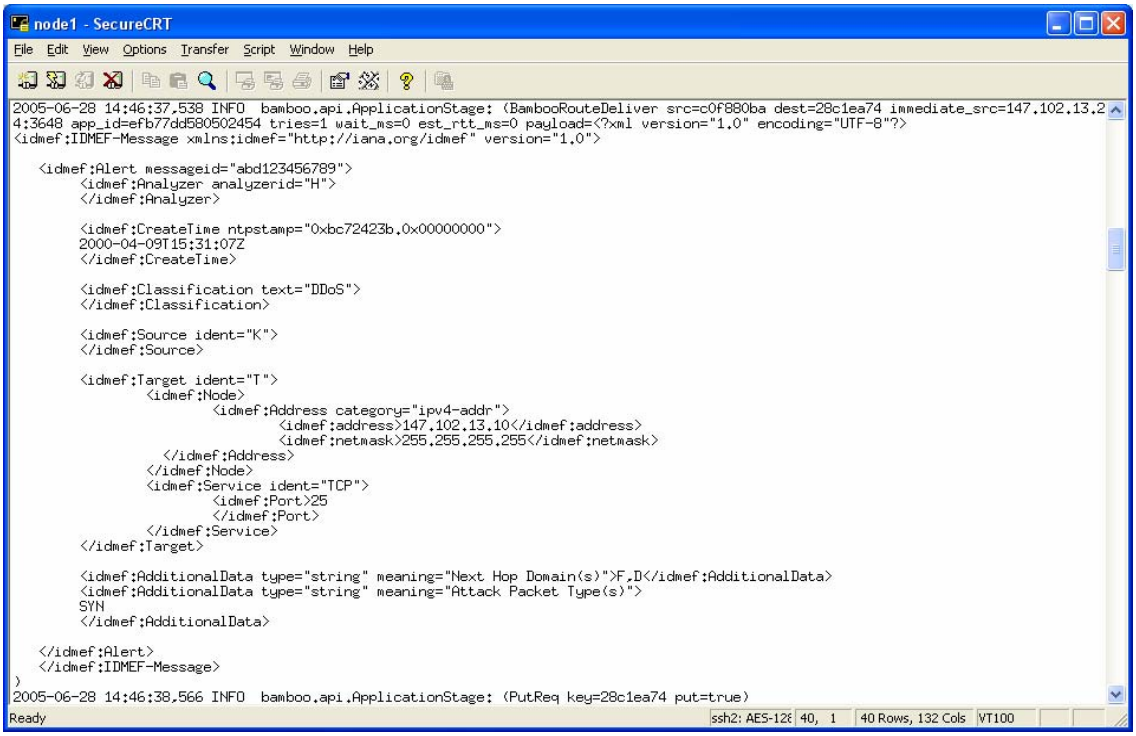
5.9 Παράδειγμα ενδεικτικής περίπτωσης λειτουργίας

Σε αυτό το σημείο παραθέτουμε screenshots μίας τυπικής εκτέλεσης της εφαρμογής μας:

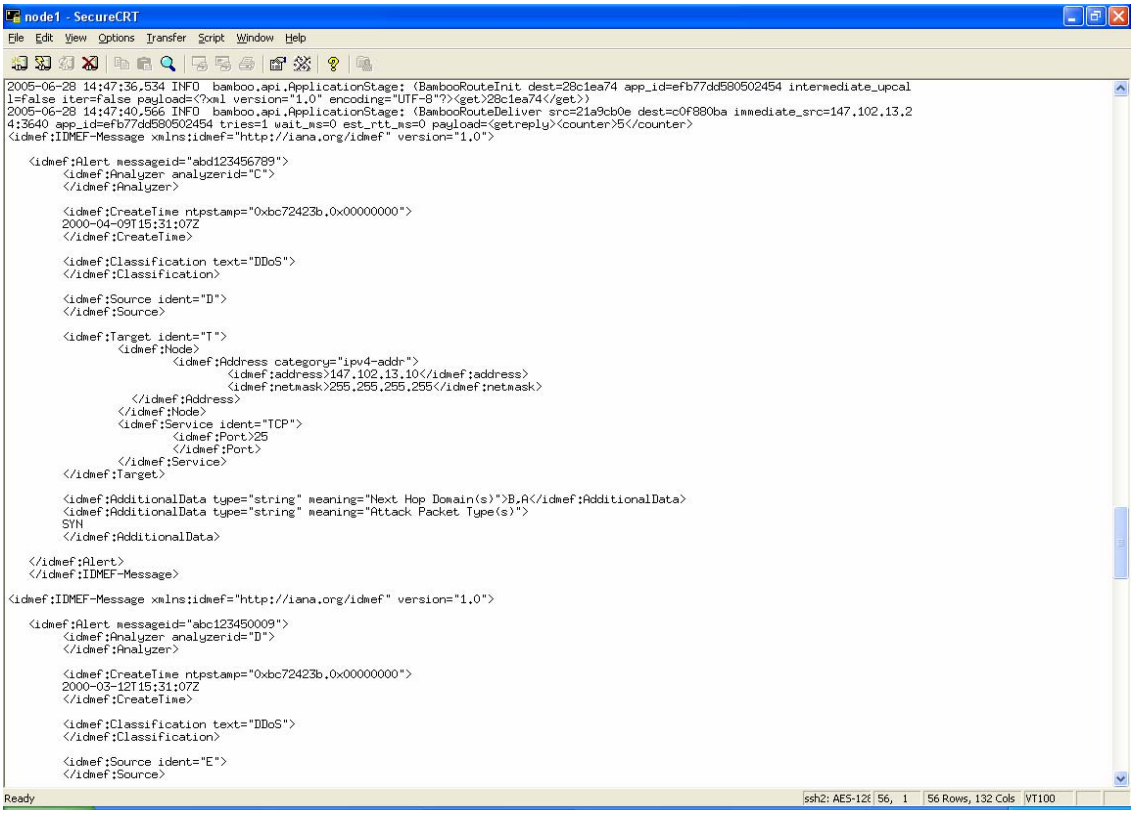
- Αίτηση PUT από κόμβο της συνεργατικής υποδομής που εμπλέκεται σε επίθεση DDoS



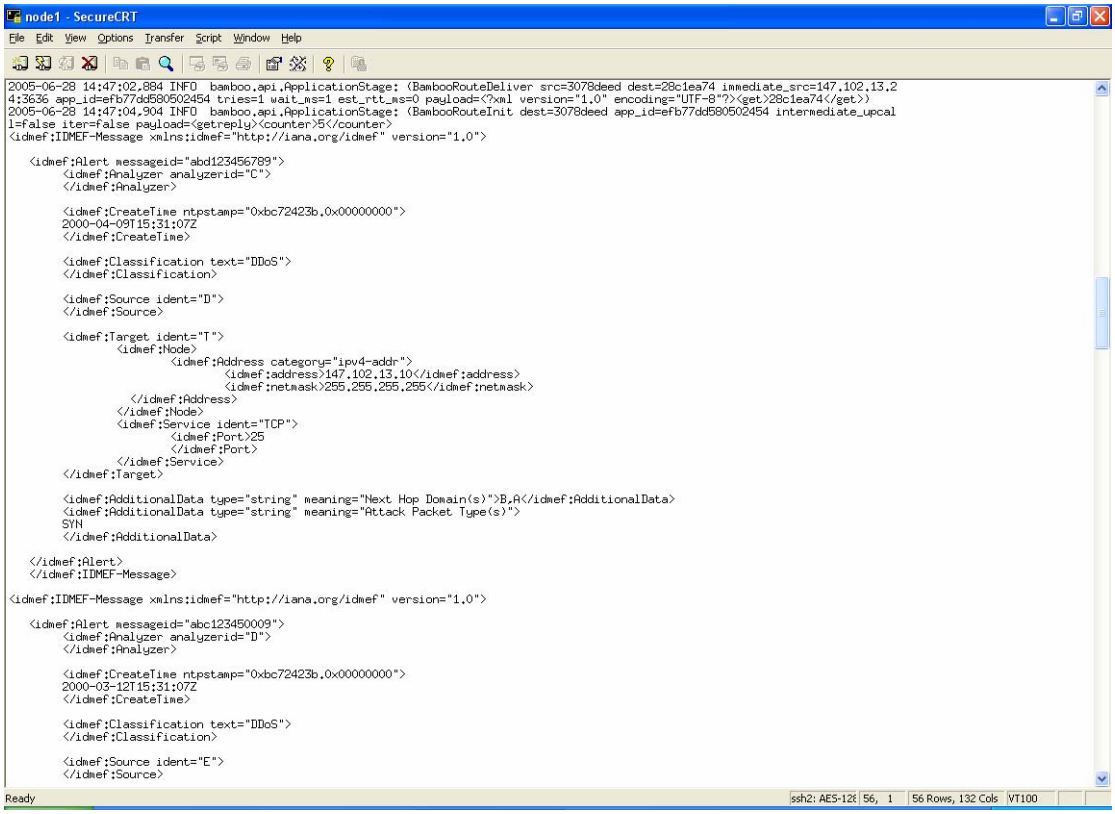
- Λήψη της παραπάνω αίτησης PUT από τον κόμβο – σημείο συγκέντρωσης για τη συγκεκριμένη επίθεση της συνεργατικής υποδομής



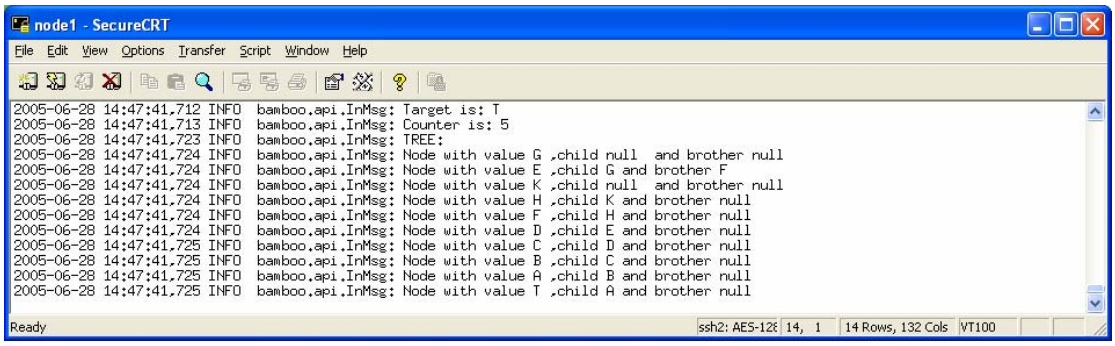
- Αίτηση GET από κόμβο της συνεργατικής υποδομής που εμπλέκεται σε επίθεση DDoS και μέρος της απάντησης GET



- Λήψη της αίτησης GET από τον κόμβο – σημείο συγκέντρωσης για τη συγκεκριμένη επίθεση της συνεργατικής υποδομής



- Προκύπτον δέντρο από την απάντηση GET



Κεφάλαιο 6: Συμπεράσματα

6.1 Συνέπεια του συστήματος

6.1.1 Συνέπεια του δικτύου peer-to-peer

Κάθε δίκτυο peer-to-peer διαθέτει μηχανισμούς αυτόματης οργάνωσης, έτσι ώστε το σύστημα να παραμένει συνεπές παρά τις αναχωρήσεις και τις αφίξεις κόμβων. Η συνέπεια περιλαμβάνει ορθή πληροφορία ανά κόμβο και διατήρηση των δεδομένων που μοιράζονται από τους κόμβους του υπερκείμενου δικτύου. Αυτό το χαρακτηριστικό των δικτύων peer-to-peer στηρίζει τις υπερκείμενες σε αυτά εφαρμογές και δικαιολογεί την επιλογή τους ως υποκείμενα στρώματα σε συστήματα με δυναμική μορφή.
Συγκεκριμένα για την εφαρμογή μας διατήρηση των δεδομένων σημαίνει ότι, ακόμα και εάν ο κόμβος συγκέντρωσης των δεδομένων για μία επίθεση DDoS «πέσει», τα δεδομένα δε θα χαθούν, αλλά θα αποθηκεύονται στο νέο κόμβο συγκέντρωσης, που θα είναι ο κόμβος με nodeID πλησιέστερα στο GUID της επίθεσης για τη δεδομένη χρονική στιγμή. Επιπλέον, για να αυξηθεί ο βαθμός εφεδρείας του συστήματος, μπορούν τα δεδομένα για μία επίθεση DDoS να αποθηκεύονται στις k-ρίζες του αναγνωριστικού της επίθεσης (βλ. παράγραφο 3.2.3).

6.1.2 Συνέπεια της αρχιτεκτονικής της προτεινόμενης υποδομής

Όπως είδαμε και στο Κεφάλαιο 4, στο προτεινόμενο υπερκείμενο δίκτυο ο συντονισμός μεταξύ των κόμβων βασίζεται στην ανταλλαγή τριών ειδών μηνυμάτων.

6.1.2.1 *Αίτηση PUT*
Η αποστολή μίας αίτησης PUT από ένα δίκτυο που συμμετέχει στην υπερκείμενη υποδομή εξαρτάται από τη λειτουργία των IDSs κατά μήκος της διαδρομής της επίθεσης. Δεν αναμένεται όλα τα IDSs, συνεπώς και όλες οι διαχειριστικές περιοχές, να αντιληφθούν την επίθεση και να αντιδράσουν. Ωστόσο, αυτό δεν αποτελεί πρόβλημα για τη λειτουργία της υποδομής αλλά πλεονέκτημά της. Η συγκέντρωση και η επεξεργασία των αιτήσεων PUT (απάντηση σε GET) μπορεί να υποδείξει ποια δίκτυα βρίσκονται πάνω στο μονοπάτι της επίθεσης και δεν έχουν στείλει αίτηση PUT.
Ένα άλλο ζήτημα που προκύπτει σχετικά με τις αιτήσεις PUT είναι το τι γίνεται με μία αίτηση PUT που είναι αποτέλεσμα ενός "false-positive" συναγερμού του IDS. Σε αυτήν την περίπτωση δε συγκεντρώνεται άλλη πληροφορία στον κόμβο – σημείο συγκέντρωσης και εάν ο ίδιος κόμβος, που έστειλε τη «λάθος» αίτηση PUT, στείλει αίτηση GET, θα λάβει πίσω μονάχα ένα μήνυμα IDMEF (αυτό που έστειλε ο ίδιος) και έτσι θα αντιληφθεί ότι το IDS του έκανε λάθος εκτίμηση. Συμπερασματικά, δεν κλονίζεται η σταθερότητα του συστήματος.
Επιπλέον, εάν μία αίτηση PUT δεν καταφέρει να φτάσει στον προορισμό της λόγω συμφόρησης ή για άλλο απροσδιόριστο λόγο, το σύστημα θα παραμείνει συνεπές, γιατί εάν πράγματι εξελίσσεται επίθεση DDoS, θα έχουν κινητοποιηθεί κάποιοι από τους υπόλοιπους κόμβους του υπερκείμενου δικτύου που επηρεάζονται από την επίθεση και είναι εξαιρετικά απίθανο να μη φτάσει έστω και μία από τις αιτήσεις PUT στον κόμβο συγκέντρωσης. Είναι επίσης εξαιρετικά απίθανο ο κόμβος συγκέντρωσης να επηρεάζεται από την επίθεση DDoS, καθώς ο τρόπος που καθίσταται υπεύθυνος ένας κόμβος peer-to-peer για τη συγκέντρωση των δεδομένων

αναφορικά με μία επίθεση DDoS εγγυάται τη γεωγραφική διαφοροποίηση των εμπλεκόμενων κόμβων σε μία επίθεση DDoS με τον κόμβο συγκέντρωσης. Αυτό το γεγονός είναι άμεση απόρροια των μηχανισμών ονοματοδοσίας των δικτύων peer-to-peer και συγκαταλέγεται στα σημαντικότερα πλεονεκτήματα της προτεινόμενης υποδομής. Τέλος, ακόμη και αν ο κόμβος συγκέντρωσης επηρεάζεται από την επίθεση DDoS, με χρήση k κόμβων συγκέντρωσης (βλ. παράγραφο 6.1.1) η συνεργατική υποδομή μπορεί να συνεχίσει την απρόσκοπτη λειτουργία της, γιατί τουλάχιστον ένας από αυτούς θα βρίσκεται σε απομακρυσμένη περιοχή σε σχέση με την επίθεση και θα μπορεί να λειτουργήσει κανονικά.

6.1.2.2 *Αίτηση GET*

Οι περισσότερες από τις παρατηρήσεις που κάναμε για τις αιτήσεις PUT ισχύουν και για τις αιτήσεις GET, καθώς αυτοί οι δύο τύποι μηνυμάτων ακολουθούν τις ίδιες διαδρομές. Η αποστολή μίας αίτησης GET που δεν αντιστοιχεί σε επίθεση που εξελίσσεται ή η απώλεια μίας αίτησης GET δεν επηρεάζει τη σωστή λειτουργία του συστήματος. Ωστόσο, πρέπει να επισημάνουμε ότι αναμένεται οι αιτήσεις GET να είναι λιγότερες από τις αιτήσεις PUT, γιατί από τους κόμβους/δίκτυα, οι οποίοι αντιλαμβάνονται ύποπτη κίνηση που αντιστοιχεί σε επίθεση DDoS, ένα μικρό ποσοστό θα αποφασίσει να αντιδράσει. Αυτοί θα είναι οι κόμβοι/δίκτυα που «υποφέρουν» περισσότερο, άρα βρίσκονται πλησιέστερα στο στόχο της επίθεσης. Επίσης, εάν από όλες τις αιτήσεις GET, παραδοθεί μόνο μία στον κόμβο συγκέντρωσης, και πάλι μπορούν να ενεργοποιηθούν οι μηχανισμοί αντίδρασης της υποδομής. Με άλλα λόγια, υπάρχει μεγαλύτερη ανεκτικότητα για απώλεια σε αιτήσεις GET, καθώς αυτές δε μεταφέρουν πληροφορία. Έτσι, ναι μεν αποστέλλονται λιγότερες αιτήσεις GET, αλλά είναι αναγκαίες και λιγότερες, για να συνεχίσει η υποδομή κανονικά τη λειτουργία της.

6.1.2.3 *Απάντηση GET*

Η απάντηση GET μεταφέρει τη μεγαλύτερη ποσότητα πληροφορίας και κατευθύνεται προς κόμβους/δίκτυα που πλήττονται από επίθεση DDoS. Συνεπώς, η παράδοση ενός τέτοιου τύπου μηνύματος διατρέχει το μεγαλύτερο κίνδυνο αποτυχίας. Συγχρόνως, είναι σημαντικό η απάντηση GET να παραδοθεί σε τουλάχιστον έναν από τους κόμβους που έστειλε αίτηση GET. Διαφορετικά, δεν πρόκειται να υπάρξει αντίδραση και αυτό ισοδυναμεί με αχρήστευση της υποδομής.

Για να αποφευχθεί η παραπάνω περίπτωση, μπορούν να γίνουν οι εξής βελτιώσεις στην υλοποίηση:

1) όταν ένας κόμβος στέλνει αίτηση GET, να τίθεται σε λειτουργία ένας χρονιστής, ο οποίος να καταργείται, όταν λαμβάνεται η απάντηση GET. Εάν ο χρονιστής λήξει και δεν έχει ληφθεί απάντηση, τότε θα ξαναστέλνεται αίτηση GET.

2) όταν ένας κόμβος λάβει απάντηση GET, να στέλνει επιβεβαίωση στον κόμβο συγκέντρωσης δεδομένων για την επίθεση, έτσι ώστε να γνωρίζει ο τελευταίος, εάν έχει φτάσει η απάντηση GET. Εάν δε λάβει ούτε επιβεβαίωση ούτε δεύτερη αίτηση GET από τον κόμβο – αποστολέα του GET (όπως προβλέπεται από την 1)), τότε προκύπτει το συμπέρασμα ότι ο κόμβος που έστειλε την αίτηση GET είναι «νεκρός». Σε αυτήν την περίπτωση μπορεί ο κόμβος συγκέντρωσης να στείλει την απάντηση GET σε κάποιον από τους υπόλοιπους κόμβους που έχουν κάνει PUT για την ίδια επίθεση DDoS. Αυτό δε θα είναι αναγκαίο, εάν κάποιος από αυτούς τους κόμβους έχει ήδη στείλει αίτηση GET και επιβεβαίωση για λήψη της απάντησης. Το μειονέκτημα ενός τέτοιου μηχανισμού είναι ότι προσθέτει εργασία στον κόμβο συγκέντρωσης, πράγμα αρνητικό, όπως αναλύεται παρακάτω.

3) εναλλακτικά, μπορεί ο κόμβος συγκέντρωσης να στέλνει πάντα απάντηση GET στον κόμβο που έστειλε αίτηση GET και σε έναν ακόμη κόμβο που έχει κάνει PUT για την ίδια επίθεση DDoS. Σε αυτήν την περίπτωση δεν εξασφαλίζεται η ενεργοποίηση της υποδομής, καθότι ο κόμβος που λαμβάνει απάντηση GET χωρίς να έχει στείλει αίτηση GET μπορεί να αρνηθεί να αντιδράσει.

6.2 Ασφάλεια του συστήματος

Όπως είδαμε, η προτεινόμενη υποδομή παραμένει σε γενικές γραμμές σταθερή και ανεπηρέαστη από τις αλλαγές στο σύνολο των κόμβων που συμμετέχουν στο υπερκείμενο δίκτυο και από απώλειες σε πακέτα. Μένει να δούμε πώς αντιδρά όταν κάποιοι από τους εμπλεκόμενους κόμβους έχουν παραβιαστεί.
Εάν ένας κακόβουλος κόμβος στείλει μία αίτηση PUT με ψευδή δεδομένα, δεν μπορεί να δημιουργηθεί πρόβλημα, καθώς θα πρόκειται για μία αίτηση PUT που δεν μπορεί να συνδυαστεί με καμία άλλη. Ακόμη και αν σταλούν πολλές αιτήσεις PUT για κάποια επίθεση DDoS που δε λαμβάνει χώρα στην πραγματικότητα, η λειτουργία της υποδομής δεν επηρεάζεται, αφού δεν επηρεάζονται τα δεδομένα για τις επιθέσεις DDoS που πράγματι εξελίσσονται. Μένει να δούμε τι γίνεται, εάν ένας ή περισσότεροι κακόβουλοι κόμβοι στείλουν αιτήσεις PUT με παραπλανητικά δεδομένα σχετικά με επιθέσεις DDoS, οι οποίες πράγματι πλήττουν το δίκτυο. Σε αυτήν την περίπτωση τα δεδομένα για τις επιθέσεις αυτές θα δώσουν μετά την επεξεργασία τους πληροφορία που δεν ανταποκρίνεται στην πραγματικότητα, αλλά δε θα διαγράφουν την πραγματική πληροφορία. Συνεπώς, δεν διακινδυνεύεται η αντίδραση της συνεργατικής υποδομής.
Ένας κόμβος που σκοπεύει να ενεργοποιήσει τους μηχανισμούς αντίδρασης της υποδομής μπορεί ωστόσο να προστατευτεί από τέτοιου είδους κακόβουλες κινήσεις ελέγχοντας τα δεδομένα που λαμβάνει από μία απάντηση GET. Μπορεί, επί παραδείγματι, να εκτελεί traceroute στους προκύπτοντες κλάδους, για να επαληθεύσει την ύπαρξή τους. Επιπλέον, μπορεί να ελέγχει εάν οι αποστολείς των μηνυμάτων IDMEF βρίσκονται στη γειτονιά του θύματος της επίθεσης, καθώς, όπως έχουμε ήδη αναφέρει, μηνύματα Alert από τα IDSs λαμβάνουν με πολύ μεγαλύτερη πιθανότητα οι κόμβοι/δίκτυα που βρίσκονται πάνω στο μονοπάτι της επίθεσης και είναι κοντά στο στόχο. Ωστόσο, ο τελευταίος αυτός έλεγχος μπορεί να παρακαμφθεί από τους κακόβουλους αποστολείς των αιτήσεων PUT, εάν οι τελευταίοι συγκαλύπτουν με IP spoofing την ταυτότητά τους και θέσουν στο πεδίο πηγής της αίτησης PUT έναν κόμβο/δίκτυο που να ανήκει στη γειτονιά του θύματος της επίθεσης. Προς αντιμετώπιση αυτής της περίπτωσης μπορεί ο κόμβος – παραλήπτης απάντησης σε GET να επαληθεύει την αποστολή των μηνυμάτων IDMEF από τις πηγές τους με μία απλή ερώτηση.
Η λειτουργία της συνεργατικής υποδομής διακυβεύεται και στην περίπτωση που ο κόμβος συγκέντρωσης των δεδομένων για μια επίθεση είναι ελεγχόμενος από επιτιθέμενο και είτε δεν απαντά στις αιτήσεις GET είτε στέλνει παραπλανητικές απαντήσεις. Αυτή η περίπτωση αντιμετωπίζεται εύκολα, εάν δεν ορίζεται μονάχα ένας κόμβος συγκέντρωσης αλλά k. Αυτοί θα είναι, όπως έχουμε ήδη αναφέρει, οι k ρίζες του αναγνωριστικού της επίθεσης. Κατά αυτόν τον τρόπο το σύστημα γίνεται πιο ασφαλές και παρουσιάζει και υψηλότερο βαθμό εφεδρείας. Πρέπει, ωστόσο, να σημειώσουμε ότι όλα τα παραπάνω ενδεχόμενα μπορούν να γίνουν σχετικά απίθανα, εάν απαιτείται πιστοποίηση για τη συμμετοχή ενός κόμβου στο υπερκείμενο δίκτυο peer-to-peer.

6.3 Πλεονεκτήματα προτεινόμενης υποδομής

Η προτεινόμενη υποδομή παρουσιάζει ένα πλήθος πλεονεκτημάτων, εκ των οποίων κάποια απορρέουν από τη χαρακτηριστική δομή των δικτύων peer-to-peer και κάποια από την κεντρική ιδέα της αρχιτεκτονικής της. Έτσι, η συνεργατική υποδομή που προτείνουμε για την αντιμετώπιση των επιθέσεων DDoS κληρονομεί πολύτιμες ιδιότητες των δικτύων peer-to-peer, όπως είναι:

- η καλή κλιμάκωση με αποτέλεσμα τη δυνατότητα εφαρμογής σε δυναμικά υπερκείμενα δίκτυα. Η προτεινόμενη υποδομή μπορεί να λειτουργήσει με έναν μικρό αριθμό μελών, αλλά και με έναν πολύ μεγάλο αριθμό μελών. Βέβαια με έναν μικρό αριθμό μελών η λειτουργία της υποδομής δεν έχει τόσο ουσιαστικό αποτέλεσμα. Ωστόσο παραμένει σταθερή.
- η αυτόματη οργάνωση και η προσαρμογή στις αλλαγές του συνόλου των μελών. Οι συμμετέχοντες δεν πλήττουν την ευρωστία της υποδομής με την αποχώρησή τους, ενώ η επέκτασή της είναι εξίσου εύκολη.
- η συνέπεια ως προς το περιεχόμενο που αποθηκεύει ο DHT και μοιράζονται τα μέλη της εφαρμογής (βλ. παράγραφο 6.1).
- η ανθεκτικότητα σε λάθη και η απρόσκοπτη λειτουργία παρά την αποτυχία συνδέσεων ή κόμβων (peers), καθώς οι υπηρεσίες που παρέχονται είναι αποκεντρωμένες. Η κατανεμημένη φύση των δικτύων peer-to-peer ευνοεί τις εφαρμογές που απαιτούν αντοχή στη συμφόρηση και στα λάθη που αυτή επιφέρει.

Πλεονεκτήματα, τα οποία σχετίζονται με την κεντρική ιδέα της αρχιτεκτονικής είναι τα εξής:

- Ο κόμβος συγκέντρωσης που δρα σαν εξυπηρετητής για κάθε διαφορετική επίθεση DDoS είναι ένας κόμβος με **τυχαία** χαρακτηριστικά (θέση, διαχειριστική περιοχή κ.α.), ο οποίος αντικαθίσταται αυτόματα από κάποιον άλλο, εάν δεν ανταποκρίνεται, και ο οποίος επιπλέον δεν επηρεάζεται – με μεγάλη πιθανότητα – από την επίθεση.
- Η εργασία που απαιτείται να εκτελεστεί από τον κόμβο συγκέντρωσης δεν απαιτεί την κατανάλωση πολλών πόρων. Έτσι, με τέτοιο χαμηλό κόστος, είναι ρεαλιστικό το σενάριο ένας κόμβος που δεν εμπλέκεται σε επίθεση DDoS να είναι πρόθυμος να βοηθήσει στην αντιμετώπισή της. Εξάλλου, εάν βρεθεί αυτός σε αυτή τη θέση, θα επωφεληθεί από τη λειτουργία της συνεργατικής υποδομής. Αυτή η λογική (**μοντέλο PULL**) δημιουργεί τις προϋποθέσεις για ευρεία εφαρμογή της υποδομής. Στα πλαίσια αυτού του ιδίου σκεπτικού, είναι επιθυμητό η οποιαδήποτε επέκταση της λειτουργικότητας του υπερκείμενου δικτύου να προσθέτει υπολογιστικό φόρτο στους άμεσα επηρεαζόμενους κόμβους και όχι στον κόμβο συγκέντρωσης.
- Τα κριτήρια για να παρθεί η απόφαση της αποστολής μίας αίτησης PUT ή μίας αίτησης GET μπορούν να διαφοροποιούνται πλήρως ανάλογα με τη διαχειριστική περιοχή και τις τακτικές ασφαλείας της χωρίς να επηρεάζεται η αποτελεσματικότητα της υποδομής. Τροποποιώντας το parsing του εγγράφου XML που περιέχει τα δεδομένα της επίθεσης μπορούν να συμμετέχουν στο σύστημα και κόμβοι που επικοινωνούν με IDSs που δεν παράγουν μηνύματα IDMEF. Αυτή η ελευθερία δημιουργεί επιπλέον προϋποθέσεις για εκτεταμένη αποδοχή της υποδομής.

6.4 Μειονεκτήματα προτεινόμενης υποδομής

Ορισμένα μειονεκτήματα της υποδομής αναφέρθηκαν εξετάζοντας τη συνέπεια και την ασφάλεια του συστήματος. Είδαμε, όμως, ότι είναι γενικά αντιμετωπίσιμες καταστάσεις. Υπενθυμίζουμε συγκεκριμένα τα εξής:

- Εάν τύχει ο κόμβος συγκέντρωσης για μία δεδομένη επίθεση DDoS να εμπλέκεται στην ίδια την επίθεση, τότε τίθεται σε κίνδυνο η λειτουργία του συστήματος όσον αφορά τη συγκεκριμένη επίθεση. Χρησιμοποιώντας, όμως, περισσότερους του ενός κόμβους συγκέντρωσης μπορεί να παρακαμφθεί το πρόβλημα.
- Όταν ανάμεσα στους κόμβους που στέλνουν αιτήσεις PUT ή αιτήσεις GET σχετικά με μία επίθεση DDoS υπάρχουν αρκετοί κακόβουλοι κόμβοι, οι οποίοι επιθυμούν να αποπροσανατολίσουν την υποδομή με ψευδή δεδομένα, τότε προστίθεται υπολογιστική εργασία, αλλά δεν εμποδίζονται οι μηχανισμοί αντίδρασης.

Το σημαντικότερο μειονέκτημα του συστήματος είναι ότι τουλάχιστον η μέχρι τώρα υλοποίηση λειτουργεί αποτελεσματικά μόνο για τις επιθέσεις DDoS που αποτελούνται από ροές κίνησης με κοινά και σταθερά χαρακτηριστικά. Αυτό σημαίνει ότι μία επίθεση DDoS με μεταβαλλόμενα στο χρόνο χαρακτηριστικά – επί παραδείγματι, μεταβαλλόμενη υπηρεσία-στόχος – δεν μπορεί να ανιχνευθεί, ενώ μία επίθεση DDoS που απλά κατευθύνει διάφορες (με διαφορετικά χαρακτηριστικά) ροές πακέτων προς το στόχο, πάλι δεν μπορεί να ανιχνευθεί και επομένως να αντιμετωπιστεί.

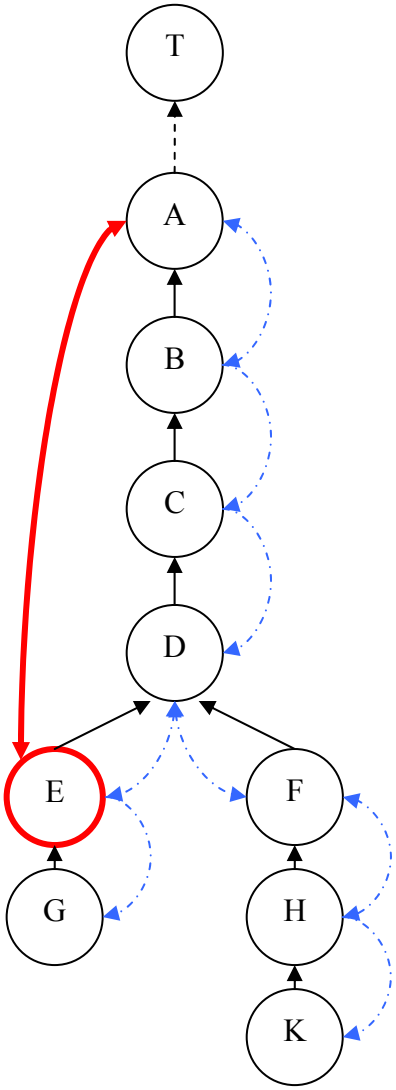
6.5 Προεκτάσεις στην προτεινόμενη υποδομή

Σε αυτό το σημείο προτείνουμε ένα σχέδιο δράσης ενάντια στην επίθεση, το οποίο θα ενεργοποιείται μόλις ένας κόμβος φτιάξει το δέντρο της επίθεσης, αφού έχει επεξεργαστεί τα δεδομένα από μία απάντηση GET. Η αντίδραση θα αποτελείται από τα εξής βήματα:

1) Ο αρχικός κόμβος στέλνει μήνυμα στα παιδιά πρώτου επιπέδου του δέντρου περιλαμβάνοντας τα χαρακτηριστικά της επίθεσης και τον κλάδο της επίθεσης στον οποίο ανήκουν, και περιμένει επιβεβαίωση συμμετοχής στη συγκεκριμένη επίθεση.
2) Ο κόμβος–παιδί του πρώτου επιπέδου, ο οποίος επιβεβαιώνει τη συμμετοχή του στην επίθεση, στέλνει ίδιο μήνυμα με το παραπάνω στα παιδιά του και περιμένει επιβεβαίωση συμμετοχής. Παράλληλα, ελέγχει εάν υπάρχει εισερχόμενη κίνηση με τα ίδια χαρακτηριστικά από άλλο δίκτυο και εάν αυτό ισχύει, στέλνει μήνυμα ειδοποίησης στην πηγή – ή στις πηγές - της ύποπτης κίνησης περιλαμβάνοντας τα χαρακτηριστικά της επίθεσης και τον κλάδο, στον οποίο ενδεχομένως να προστεθούν, και περιμένει επιβεβαίωση από αυτήν ότι πράγματι αποτελεί ενδιάμεσο βήμα στην κίνηση της ύποπτης ροής. Κατά αυτόν τον τρόπο επεκτείνεται και επαληθεύεται το δέντρο. Με τον ίδιο τρόπο ενεργούν και οι κόμβοι–παιδιά του κόμβου πρώτου επιπέδου. Η παραπάνω διαδικασία σταματά, όταν ζητηθεί επιβεβαίωση συμμετοχής από τους κόμβους–φύλλα του δέντρου.
3) Εάν ένας κόμβος–παιδί στο δέντρο δεν ανταποκριθεί καθόλου, τότε ο κόμβος –πατέρας συνεχίζει τη διαδικασία στέλνοντας τα σχετικά μηνύματα (βλ. 2)) στους κόμβου –εγγόνια του.

4) Οι κόμβοι–φύλλα και οι δικτυακές περιοχές που προστίθενται στο δέντρο μπορούν να συνεχίσουν τη διαδικασία της αναγνώρισης διαδρομής ψάχνοντας για τις πηγές της εισερχόμενης κακόβουλης κίνησης ή μπορούν να επιλέξουν να «κόψουν» την ύποπτη κίνηση με αποτέλεσμα να ανακουφιστεί το σύστημα.

Επισημαίνουμε ότι, όταν ένας κόμβος δεν επιβεβαιώσει τη συμμετοχή του (με αρνητική επιβεβαίωση) στην επίθεση, απλά διαγράφεται από το δέντρο, ενώ κάθε κόμβος που επιβεβαιώνει τη συμμετοχή του στην επίθεση και συνεχίζει την παραπάνω διαδικασία, αποθηκεύει τις επιβεβαιώσεις που λαμβάνει. Συνεπώς, η πληροφορία του δέντρου διανέμεται στο σύστημα. Ακολουθεί ένα σχήμα που συνοψίζει το παραπάνω σχέδιο δράσης ενάντια στην επίθεση.



Εικόνα 22: Επικοινωνία μεταξύ των εμπλεκόμενων κόμβων στην επίθεση DDoS με στόχο την αντίδραση. Ο αρχικός κόμβος είναι ο κόμβος Ε και ο κόμβος-παιδί πρώτου επιπέδου είναι ο κόμβος Α. Το φαρδύ βέλος αναφέρεται στα μηνύματα του βήματος 1), ενώ τα διακεκομμένα, καμπύλα βέλη στα μηνύματα του βήματος 2).

Τέλος, η υπάρχουσα υποδομή μπορεί να επεκταθεί, ώστε να ανιχνεύει επιθέσεις με λιγότερα κοινά χαρακτηριστικά. Αυτό μπορεί να γίνει χρησιμοποιώντας πολλά επίπεδα κατακερματισμού και πολλούς κόμβους συγκέντρωσης - έναν για κάθε επίπεδο. Το πρώτο επίπεδο κατακερματισμού θα παράγει το αναγνωριστικό της επίθεσης από τα χαρακτηριστικά που μελετήθηκαν στην παράγραφο 4.1 (πεδία στο μήνυμα IDMEF). Ο κόμβος συγκέντρωσης του πρώτου επιπέδου θα αποθηκεύει την πιο συγκεκριμένη πληροφορία για την επίθεση DDoS. Ο ίδιος αυτός κόμβος συγκέντρωσης μπορεί να παράγει ένα νέο αναγνωριστικό (αναγνωριστικό επόμενου επιπέδου) για την επίθεση κατακερματίζοντας ένα πεδίο λιγότερο (π.χ. το πεδίο *Port*) και να στέλνει αίτηση PUT στον κόμβο συγκέντρωσης επόμενου επιπέδου. Η πληροφορία που θα συγκεντρώνεται στον κόμβο συγκέντρωσης επόμενου επιπέδου θα περιλαμβάνει την προηγούμενη πληροφορία, αλλά θα είναι ευρύτερη από αυτήν, εάν ο στόχος δέχεται επίθεση DDoS σε πολλές διαφορετικές πόρτες. Συγκρίνοντας τα δέντρα που προκύπτουν μπορεί να παρθεί η απόφαση για το τι συμβαίνει. Η παραπάνω διαδικασία μπορεί να συνεχιστεί με περισσότερα επόμενα επίπεδα και να προκύψει μία περισσότερο ολοκληρωμένη εικόνα για την επίθεση.

Βιβλιογραφία

[1] J. Mirkovic, M. Robinson, P. Reiher, and G. Kuenning. Alliance formation for ddos defense. In Proc. New Security Paradigms Workshop, ACM SIGSAC, Aug. 2003.

[2] G. Koutepas, B. Maglaris, "Detection and Reaction to Denial of Service Attacks", Cyprus Security Society "2nd Conference on Information Security: From Theory to Practice", Nikosia, Cyprus, October 2002.

[3] G. Koutepas, F. Stamatelopoulos, and B. Maglaris, "Distributed Management Architecture for Cooperative Detection and Reaction to DDoS Attacks", Special Issue on Security and Management, Journal of Network and Systems Management, Vol. 12, No. 1, March 2004.

[4] R. Stone. CenterTrack: An IP Overlay Network for Tracking DoS floods, 9th USENIX Security Symposium, Denver Col., USA, August 2000.

[5] J. Ioannidis and S. Bellovin, Pushback: Router-Based Defense Against DDoS Attacks, NDSS, February 2002.

[6] Panoptis. http://panoptis.sourceforge.net/

[7] ICQ. http://www.icq.com

[8] Gnutella. http://gnutella.wego.com

[9] Napster. http://www.napster.com.

[10] Scour. http://www.scourdesign.com

[11] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong, “Freenet: A distributed anonymous information storage and retrieval system,” in *Proc. Int. Workshop Design Issues in Anonymity and Unobservability*, Berkeley, CA, 2001.

[12] Ohaha. http://www.ohaha.com

[13] Jungle Monkey. http://www.junglemonkey.net/

[14] B.Wilcox-O’Hearn, “Experiences deploying a large-scale emergent network,” in *Proc. IPTPS*, Cambridge, MA, Mar. 2002.

[15] S. Ratnasamy, M. Handley, R. Karp, and S. Schenker, “Application-level multicast using content-addressable networks,” in *Proc. NGC*, London, U.K., Nov. 2001.

[16] A. Rowstron, A.-M. Kermarrec, P. Druschel, and M. Castro, “SCRIBE: The design of a large-scale event notification infrastructure,” in *Proc. NGC*, London, U.K., Nov. 2001.

[17] S. Q. Zhuang, B. Y. Zhao, A. D. Joseph, R. H. Katz, and J. D. Kubiatowicz, "Bayeux: An architecture for scalable and fault-tolerant wide-area data dissemination," in *Proc. NOSSDAV*, Port Jefferson, NY, June 2001.

[18] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica, "Wide-area cooperative storage with CFS," in *Proc. SOSP*, Banff, Canada, Oct. 2001.

[19] S. Hand and T. Roscoe, "Mnemosyne: Peer-to-peer steganographic storage," in *Proc. IPTPS*, Cambridge, CA, Mar. 2002.

[20] S. Rhea, P. Eaton, D. Geels, H. Weatherspoon, B. Zhao, and J. Kubiatowicz, "Pond: The OceanStore prototype," in *Proc. FAST*, San Francisco, CA, Apr. 2003.

[21] A. Rowstron and P. Druschel, "Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility," in *Proc. SOSP*, Banff, Canada, Oct. 2001.

[22] A. Keromytis, V. Misra, and D. Rubenstein, "SOS: Secure overlay services" in *Proc. SIGCOMM*, Pittsburgh, PA, Aug. 2002.

[23] I. Stoica, D. Adkins, S. Zhuang, S. Shenker, and S. Surana, "Internet indirection infrastructure," in *Proc. SIGCOMM*, Pittsburgh, PA, Aug. 2002.

[24] F. Zhou, L. Zhuang, B.Y. Zhao, L. Huang, A. D. Joseph, and J. D.Kubiatowicz, "Approximate object location and spam filtering on peer-to-peer systems," in *Proc. Middleware*, Rio de Janeiro, Brazil, June 2003.

[25] W. Bolosky, J. Douceur, D. Ely, and M. Theimer. Feasibility of a Serverless Distributed File System Deployed on an existing set of Desktop PCs. In *Proceedings of SIGMETRICS 2000*, Santa Clara, CA, June 2000.

[26] A. D. R. Marc Waldman and L. F. Cranor. Publius: A Robust, Tamper-evident, Censorship-resistant, Web Publishing System. In *Proceedings of the 9th USENIX Security Symposium*, August 2000.

[27] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, Scott Shenker. A Scalable Content-Addressable Network. In Proc. ACM SIGCOMM 2001, August 2001.

[28] Stefan Saroiu, P. Krishna Gummadi and Steven D. Gribble. A Measurement Study of Peer-to- Peer File Sharing Systems. Technical Report UW-CSE-01-06-02, University of Washington, Department of Computer Science and Engineering, July 2001.

[29] Petar Maymounkov, David Mazieres. Kademlia: A Peer-to-peer Information System Based on the XOR Metric. In Proc. IPTPS, Cambridge, MA, Mar 2002.

[30] Frank Dabek, Emma Brunskill, M. Frans Kaashoek, David Karger, Robert Morris, Ion Stoica, Hari Balakrishnan. MIT Laboratory for Computer Science:

Building Peer-to-Peer Systems with Chord, a Distributed Lookup Service. http://pdos.lcs.mit.edu/chord

[31] Kevin Fu, M. Frans Kaashoek, and David Mazi`eres. Fast and secure distributed read-only file system. In *Proceedings of the 4th USENIX Symposium on Operating Systems Design and Implementation (OSDI 2000)*, San Diego, California, October 2000.

[32] David Chaum. Untraceable electronic mail, return addresses and digital pseudonyms. *Communications of the A.C.M.*, 24(2):84–88, 1981.

[33] B. Y. Zhao, A. D. Joseph, and J. Kubiatowicz, "Locality-aware mechanisms for large-scale networks," presented at the Int. Workshop on Future Directions in Distributed Computing, Bertinoro, Italy, June 2002.

[34] M. J. B. Robshaw, "MD2, MD4, MD5, SHA and other hash functions," RSA Labs., vol. 4.0, Tech. Rep. TR-101, 1995.

[35] Y. Rekhter and T. Li. (1993) An architecture for IP address allocation with CIDR. [Online]. Available: http://www.isi.edu/in-notes/rfc1518.txt

[36] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph, "Tapestry: An infrastructure for fault-tolerant wide-area location and routing," Univ. California, Berkeley, CA, Tech. Rep. CSD-01-1141, Apr. 2001.

[37] M. Welsh, D. Culler, and E. Brewer, "SEDA: An architecture for wellconditioned, scalable internet services," in *Proc. SOSP*, Banff, Canada, Oct. 2001.

[38] B. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Commun. ACM*, vol. 13, no. 7, pp. 422–426, July 1970.

[39] J. R. Douceur, "The Sybil attack," in *Proc. IPTPS*, Cambridge, MA, Mar. 2002.

[40] S. Ratnasamy,M. Handley, R. Karp, and S. Schenker, "Application-level multicast using content-addressable networks," in Proc. NGC, London, U.K., Nov. 2001, pp. 14–29.

[41] P. Druschel and A. Rowstron. PAST: A large-scale, persistent peer-to-peer storage utility. In *Proc. HotOS VIII*, Schloss Elmau, Germany, May 2001.

[42] A. Rowstron and P. Druschel, "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems," in Proc. Middleware, Heidelberg, Germany, Nov. 2001.

[43] S. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz. Handling churn in a DHT. Technical Report UCB//CSD-03-1299, University of California, Berkeley, December 2003.

[44] Marcel Dischinger. A flexible and scalable peer-to-peer multicast application using Bamboo, June, 27th 2004.

[45] Frank Dabek, Ben Zhao, Peter Druschel, John Kubiatowicz, Ion Stoica. Towards a Common API for Structured Peer-to-Peer Overlays, *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS '03)*.

[46] Stoica, I., Adkins, D., Zhuang, S., Shenker, S., and Surana, S. Internet indirection infrastructure. In Proceedings of SIGCOMM (August 2002), ACM.

[47] Castro, M., Druschel, P., Ganesh, A., Rowstron, A., and Wallach, D. S. Secure routing for structured peer-to-peer overlay networks. In *Proceedings of OSDI* (December 2002).

[48] http://www.ietf.org/internet-drafts/draft-ietf-idwg-idmef-xml-14.txt

[49] http://www.bamboo-dht.org

[50] Markus Jakobsson, Tom Leighton, Silvio Micali, and Michael Szydlo. Fractal Merkle Tree Representation and Traversal. RSA Laboratories, Bedford, MIT Laboratory for Computer Science, Cambridge, Akamai Technologies, Cambridge.

[51] http://research.microsoft.com/~antr/Pastry/

[52] http://www.cs.berkeley.edu/~ravenben/tapestry/

[53] http://oceanstore.org/publications/papers/abstracts/iptps03-api.html

ΠΑΡΑΡΤΗΜΑ

ApplicationStage.java

```
package bamboo.api;

import java.math.BigInteger;
import java.util.LinkedList;
import java.util.Iterator;
import java.io.*;
import java.net.ServerSocket;
import java.net.Socket;
import java.lang.Thread;
import java.nio.ByteBuffer;

import org.jdom.*;

import bamboo.api.*;
import bamboo.dht.*;
import bamboo.util.StandardStage;
import bamboo.util.GuidTools;

import ostore.util.InputBuffer;
import ostore.util.OutputBuffer;
import ostore.util.QuickSerializable;
import ostore.util.TypeTable;

import seda.sandStorm.api.ConfigDataIF;
import seda.sandStorm.api.EventHandlerException;
import seda.sandStorm.api.QueueElementIF;
import seda.sandStorm.api.StagesInitializedSignal;

public class ApplicationStage extends StandardStage {

    protected static final long app_id =
    bamboo.router.Router.app_id(ApplicationStage.class);

    protected int portIDS;
    protected int portGET;
    protected boolean initialized = false;
    protected LinkedList wait_q = new LinkedList();
    protected LinkedList msg_q = new LinkedList();

    protected static class GetAlarm implements QueueElementIF{}

    protected class ListenThread1 extends Thread {
        protected int port;

        public ListenThread1(int port) {
            this.port = port;
```

```
            setDaemon(true);
        }

        public void run() {
            try {
                ServerSocket listener = new ServerSocket(port);
                while (true) {
                    Socket fromIDS = listener.accept();
                    logger.info("The IDS has an Alert message for me.");
                    InputStream in = fromIDS.getInputStream();
                    BufferedReader bin = new BufferedReader(new
InputStreamReader(in));
                    classifier.dispatch_later(new MsgFromIDS(bin), 1000);
                    fromIDS.close();
                }
            }
            catch (IOException e) {
                logger.info("IOException!");
            }
        }
    }

      protected class ListenThread2 extends Thread {
        protected int port;

        public ListenThread2(int port) {
            this.port = port;
            setDaemon(true);
        }

        public void run() {
            try {
                ServerSocket listener = new ServerSocket(port);
                while (true) {
                    Socket GUIDget = listener.accept();
                    logger.info("I have been asked to send a GET request.");
                    InputStream in = GUIDget.getInputStream();
                    BufferedReader bin = new BufferedReader(new
InputStreamReader(in));
                    classifier.dispatch_later(new MsgForGet(bin), 1000);
                    GUIDget.close();
                }
            }
            catch (IOException e) {
                logger.info("IOException!");
            }
        }
    }

    public ApplicationStage() throws Exception {
```

```
        super();

        //register the new types -> .class
        ostore.util.TypeTable.register_type(bamboo.api.Payload.class);
        ostore.util.TypeTable.register_type(bamboo.api.MsgFromIDS.class);
        ostore.util.TypeTable.register_type(bamboo.api.MsgForGet.class);
        ostore.util.TypeTable.register_type(bamboo.api.InMsg.class);

        event_types = new Class [] {
                StagesInitializedSignal.class,
                GetAlarm.class,
                MsgFromIDS.class,
                MsgForGet.class,
                InMsg.class,
                BambooRouteDeliver.class,
                BambooRouterAppRegResp.class,
                Dht.GetResp.class,
                Dht.PutResp.class
                };
    }

    public void init (ConfigDataIF config) throws Exception {
                super.init(config);

                //edw pairnw tis prm apo config file
                 portIDS = Integer.parseInt (config_get_string
                (config,"port_from_IDS"));
                 portGET = Integer.parseInt (config_get_string
                (config,"port_for_GUID"));
    }

    protected BigInteger get_src,attackGUID;
    protected String msg,inmsg;

    public void handleEvent(QueueElementIF elem) {
                logger.debug("Got event" + elem);

                if (!initialized){
                        if (elem instanceof StagesInitializedSignal){
                                dispatch(new
BambooRouterAppRegReq(app_id,false,false,false,my_sink));
                         }
                        else if (elem instanceof BambooRouterAppRegResp){
                                initialized = true;
                                logger.info("The port_from_IDS is "+ portIDS);
                                logger.info("The port_for_GUID is "+ portGET);
                                Thread _listenThread1 = new ListenThread1(portIDS);
                                _listenThread1.start();
                                Thread _listenThread2 = new ListenThread2(portGET);
                                _listenThread2.start();
```

```
                        while (!wait_q.isEmpty())
                        handleEvent((QueueElementIF) wait_q.removeFirst());
                    }
                    else wait_q.addLast(elem);
                }
                else {
                    if (elem instanceof MsgFromIDS) {
                        MsgFromIDS _msgFromIDS = (MsgFromIDS) elem;
                        msg_q.addLast(_msgFromIDS);
                        BambooRouteInit init = new BambooRouteInit(
                            _msgFromIDS.getGUID(), app_id, false, false, new
Payload(_msgFromIDS.toString()));
                        logger.info(init.toString());
                        attackGUID = _msgFromIDS.getGUID();
                        classifier.dispatch_later(new GetAlarm(),60000);
                        classifier.dispatch_later(init,1000);
                    }
                    else if (elem instanceof GetAlarm) {
                        BambooRouteInit init = new BambooRouteInit(
                            attackGUID, app_id, false, false, new Payload(
                            "<?xml version=\"1.0\" encoding=\"UTF-8\"?><get>"+
                     bamboo.util.GuidTools.guid_to_string(attackGUID)+"</get>"));
                            logger.info(init.toString());
                            classifier.dispatch_later(init,1000);
                    }
                    else if (elem instanceof MsgForGet) {
                        MsgForGet _msgForGet = (MsgForGet) elem;
                        BambooRouteInit init = new BambooRouteInit(
                            _msgForGet.getGUID(), app_id, false, false, new
Payload(_msgForGet.toString()));
                        logger.info(init.toString());
                        classifier.dispatch_later(init,1000);
                     }
                     else if (elem instanceof BambooRouteDeliver) {
                        BambooRouteDeliver deliver = (BambooRouteDeliver) elem;
                        logger.info(deliver.toString());
                        Payload pay = (Payload) deliver.payload;
                            classifier.dispatch_later(new InMsg(pay.message,
deliver.dest, deliver.src), 1000);
                     }
                     else if (elem instanceof InMsg) {
                        InMsg _inMsg = (InMsg) elem;
                        switch (_inMsg.getType()) {
                            case 0 :
                            //put
                            inmsg = _inMsg.toString();
                            byte[] data_bytes = inmsg.getBytes();
                            ByteBuffer data = ByteBuffer.wrap (data_bytes);
                            Dht.PutReq preq = new Dht.PutReq   (_inMsg.
getGUID(), data, true, my_sink, null);
```

```
                        logger.info(preq.toString());
                        classifier.dispatch_later (preq, 1000);
                        break;
                        case 1 :
                        //get
                        get_src = _inMsg.getSrc();
                        msg = "</counter>";
                        Dht.GetReq greq = new Dht.GetReq (_inMsg.
getGUID(), my_sink, null);
                        classifier.dispatch_later(greq, 1000);
                        break;
                        case 2 :
                        //process GET reply
                        GetReply greply;
                        greply = new GetReply(_inMsg.msg,
_inMsg.targetGUID, _inMsg.src);
                        greply.getTree();
                        break;
                        default:
                        logger.info("GetType oute 0, oute 1, oute 2!");
                        break;
                    }
                }
                else if (elem instanceof Dht.GetResp) {
                        int index;
                        String idmef;

                        Dht.GetResp resp = (Dht.GetResp) elem;
                        Iterator i = resp.values.iterator();
                        while (i.hasNext()){
                                ByteBuffer data = (ByteBuffer) i.next();
                                String ndata = new String(data.array(),
                                        data.arrayOffset() + data.position(),
                                        data.limit() - data.position());
                                index = ndata.indexOf('>');
                                idmef = ndata.substring(index+1,ndata.length());
                                msg += idmef;
                        }
                         Payload payload = new Payload
("<getreply><counter>"+resp.values.size()+msg+"</getreply>");
                         BambooRouteInit in = new BambooRouteInit(get_src,
app_id, false, false, payload);
                         logger.info(in.toString());
                         classifier.dispatch_later (in,1000);
                    }
                    else if (elem instanceof Dht.PutResp) {
                    }
                    else {
                       BUG ("Event "+ elem +" unknown.");
                }
```

```
                }
        }
}
```

MsgFromIDS.java

```
package bamboo.api;

import java.io.*;
import java.util.*;
import java.math.BigInteger;

import org.jdom.*;
import org.jdom.input.SAXBuilder;
import org.jdom.output.XMLOutputter;´

import org.apache.log4j.Logger;

import bamboo.api.*;
import bamboo.util.GuidTools;
import ostore.util.InputBuffer;
import ostore.util.OutputBuffer;
import ostore.util.QuickSerializable;
import ostore.util.SHA1Hash;
import ostore.util.SecureHash;
import seda.sandStorm.api.QueueElementIF;


    public class MsgFromIDS implements QueueElementIF, QuickSerializable {
        private Document doc;
        private Namespace nms;
        static  Logger logger = Logger.getLogger (MsgFromIDS.class);

        public MsgFromIDS (BufferedReader b){
                    SAXBuilder builder = new SAXBuilder();
                     try {
                         //logger.info("Creating Document");
                         doc = builder.build(b);
                         //logger.info("Document created");
                         nms = doc.getRootElement().getNamespace();
                     }
                     catch (JDOMException e1) {}
                     catch (IOException e2) {}
        }

        public MsgFromIDS (InputBuffer b) {
        }

        public void serialize(OutputBuffer b) {
            b.add(this.toString());
```

```
        }

        private String getTargIdent() {
            return doc.getRootElement().getChild("Alert", nms). getChild
("Target", nms). getAttribute("ident").getValue();
        }

        private String getServIdent() {
             return doc.getRootElement(). getChild("Alert",nms). getChild
("Target",nms).getChild("Service",nms).getAttribute("ident").getValue();
        }

        private String getPort() {
            return doc.getRootElement().getChild ("Alert",nms).getChild
("Target",nms).getChild("Service",nms).getChild("Port",nms).getText();
        }

        private String getPackType() {
            return ((Element) doc.getRootElement(). getChild("Alert",nms).
getChildren("AdditionalData",nms).get(1)).getText();
        }

        public BigInteger getGUID() {
            SecureHash guid;

            String my_fields = getTargIdent()+ getServIdent()+ getPort()+
            getPackType();
            guid = new SHA1Hash (my_fields);
            return GuidTools.secure_hash_to_big_integer(guid);
        }

            public String toString() {
            XMLOutputter outputter = new XMLOutputter();
            return outputter.outputString(doc);
        }
}
```

MsgForGet.java

```
package bamboo.api;

import java.io.*;
import java.util.*;
import java.math.BigInteger;
import java.lang.Math;

import org.apache.log4j.Logger;

import bamboo.api.*;
import bamboo.util.GuidTools;
```

```
import ostore.util.InputBuffer;
import ostore.util.OutputBuffer;
import ostore.util.QuickSerializable;
import seda.sandStorm.api.QueueElementIF;



    public class MsgForGet implements QueueElementIF, QuickSerializable {
        private BigInteger GUID;
        private String str;
        static  Logger logger = Logger.getLogger (MsgForGet.class);

        public MsgForGet (BufferedReader b){
            BigInteger big;
            try {
                    GUID = new BigInteger(b.readLine(),16);
                     logger.info("Searching info for attack with GUID:    "+
                    bamboo.util.GuidTools.guid_to_string(GUID));
                    str = "<?xml version=\"1.0\" encoding=\"UTF-8\"?><get>"
                    +bamboo.util.GuidTools.guid_to_string(GUID)+"</get>";
            }catch (IOException e) {}
        }

        public MsgForGet (InputBuffer b) {
        }

        public void serialize(OutputBuffer b) {
            b.add(str);
        }

        public BigInteger getGUID() {
            return GUID;
        }

        public String toString() {
            return str;
        }
}
```

Payload.java

```
package bamboo.api;

import java.math.BigInteger;
import java.util.LinkedList;

import bamboo.api.*;
import ostore.util.InputBuffer;
import ostore.util.OutputBuffer;
import ostore.util.QuickSerializable;
```

```
        public  class Payload implements QuickSerializable {
            public String message;

            public Payload(String m) {
                message = m;
            }

            public Payload(InputBuffer b) {
                message = b.nextString();
            }

            public void serialize(OutputBuffer b) {
                b.add(message);
            }

            public String toString() {
            return message;
            }
    }
```

InMsg.java

```
package bamboo.api;

import java.io.*;
import java.util.*;
import java.math.BigInteger;

import org.jdom.*;
import org.jdom.input.SAXBuilder;
import org.jdom.output.XMLOutputter;

import org.apache.log4j.Logger;

import bamboo.api.*;
import ostore.util.InputBuffer;
import ostore.util.OutputBuffer;
import ostore.util.QuickSerializable;
import seda.sandStorm.api.QueueElementIF;


      public class InMsg implements QueueElementIF, QuickSerializable {
            protected String msg;
            protected BigInteger targetGUID,src;
            protected Document doc;
            static  Logger logger = Logger.getLogger (InMsg.class);

            public InMsg (String m, BigInteger d, BigInteger s){
                msg = m;
```

```
        targetGUID = d;
        src = s;

        SAXBuilder builder = new SAXBuilder();
                try {
                    Reader r = new StringReader(msg);
                    doc = builder.build(r);
                }
                catch (JDOMException e1) {}
                catch (IOException e2) {}
    }

    public InMsg (InputBuffer b) {
    }

    public void serialize(OutputBuffer b) {
            b.add(this.toString());
    }

    public int getType() {
        int i ;
        String s = doc.getRootElement().getName();

        if (s.equals("getreply")) {
                return i=2;
        }
        else if (s.equals("get")) {
                return i=1;
        }
        else return i=0;
    }

    public BigInteger getGUID() {
        return targetGUID;
    }

    public BigInteger getSrc() {
        return src;
    }

     public String toString() {
        XMLOutputter outputter = new XMLOutputter();
        return outputter.outputString(doc);
    }

}
```

GetReply.java

```
package bamboo.api;

import java.io.*;
import java.util.*;
import java.math.BigInteger;

import org.jdom.*;

public class GetReply extends InMsg {
	private Namespace nms;

	//NODE
	protected class Node {
		private String value;
		private Node child;
		private Node brother;

		public Node(String s) {
			value = s;
			child = null;
			brother = null;
		}

		public String toString() {
			String str = "Node with value "+value+" ,child ";
			if (child == null) {
				str +="null ";
			}
			else {
				str += child.value;
			}
			if (brother == null) {
				str +=" and brother null";
			}
			else {
				str += " and brother "+brother.value;
			}
			return str;
		}
	}

	//TREE
	protected class Tree {
		private Node head;

		public Tree(String trg) {
			head = new Node(trg);
		}
```

```
public Tree(Node hd) {
        head = hd;
}

public void merge(LinkedList b) {
        Node a = head;
        Iterator i = b.iterator();
        String str = (String)i.next();

        if (a.child == null){
                a.child= new Node (str);
                a = a.child;
                while (i.hasNext()) {
                        a.child = new Node ((String)i.next());
                        a = a.child;
                }
        }
        else {
                if (search(str,a.child)!= null){
                        a = search(str,a.child);
                        str = (String) i.next();
                }
                else {
                }

                int flag = 1;
                while (i.hasNext()) {
                        if (flag == 1){
                                if (a.child != null){
                                if (str.equals((String)a.child.value)) {
                                        a = a.child;
                                        str=(String)i.next();
                                        if (!i.hasNext()) {
                                                hang(str,a);
                                                return;
                                        }
                                }
                                else {
                                    if (a.child.brother == null) {
                                        a.child.brother = new Node (str);
                                        a = a.child.brother;
                                        while (i.hasNext()) {

                a.child = new Node((String)i.next());
                                        a = a.child;
                                        }
                                        return;
                                    }
                                   else {
```

```
                        if (str.equals((String)a.child.brother.value)) {
                                a = a.child.brother;
                                str=(String)i.next();
                                if (!i.hasNext()) {
                                        hang(str,a);
                                        return;
                                }
                        }
                        else {
                                a = a.child.brother;
                                flag = 2;
                        }
                }
        }
        else {
                a.child = new Node (str);
                a = a.child;
                while (i.hasNext()) {
                        a.child = new Node ((String)i.next());
                        a = a.child;
                }
                return;
        }
}
else {  //flag=2
        if (a.brother != null){
                if (str.equals((String)a.brother.value)){
                        a = a.brother;
                        str=(String)i.next();
                        flag = 1;
                        if (!i.hasNext()) {
                                hang(str,a);
                                return;
                        }
                }
                else {
                        if (a.brother.brother == null) {
                                a.brother.brother = new Node (str);
                                a = a.brother.brother;
                                while (i.hasNext()) {
                                        a.child = new Node ((String)i.next());
                                        a = a.child;
                                }
                                return;
                        }
                        else {
                                if (str.equals((String)a.brother.brother.value)) {
                                        a = a.brother.brother;
```

```
                                        str=(String)i.next();
                                        flag = 1;
                                        if (!i.hasNext()) {
                                                hang(str,a);
                                                return;
                                        }
                                }
                                else {
                                        a = a.brother.brother;
                                }
                        }
                }
                else {
                        a.brother = new Node(str);
                        a = a.brother;
                        while (i.hasNext()) {
                                a.child = new Node ((String)i.next());
                                a = a.child;
                        }
                        return;
                }
        }
    }
  }
}
                private void output(Node a) {
                        if (a != null) {
                                output(a.child);
                                logger.info(a.toString());
                                output(a.brother);
                        }
                }

                private Node search(String str,Node a) {
                        if (a != null) {
                                if (str.equals((String)a.value)) {
                                        return a;
                                }
                                else {
                                        if (search(str,a.child) == null){
                                                if (search(str,a.brother) == null){
                                                        return null;
                                                }
                                                else {
                                                        return search(str,a.brother);
                                                }
                                        }
                                        else {
```

```
                                        return search(str,a.child);
                                }
                        }
                }
                else {
                        return null;
                }
        }

        private void hang(String str,Node a) {
                        if (a.child == null) {
                        a.child = new Node (str);
                        return;
                        }
                        else {
                                if (str.equals((String)a.child.value)) {
                                        return;
                                }
                                else {
                                        a = a.child;
                                        while (a.brother != null){
                                        if (str.equals((String)a.brother.value)) {
                                                return;
                                        }
                                        else {
                                                a = a.brother;
                                        }
                                        }
                                        a.brother = new Node (str);
                                        return;
                                }
                        }
        }

}

public  GetReply (String m, BigInteger d, BigInteger s) {
        super (m, d, s);
        nms = ((Element) doc.getRootElement(). getChildren() .get(1)).
        getNamespace();
}

private String getTarget() {
        return doc.getRootElement().getChild("IDMEF-Message",nms)
        .getChild("Alert",nms).getChild ("Target",nms).
        getAttribute("ident").getValue();
}

private int getCounter() {
```

```
        return Integer.parseInt (doc.getRootElement(). getChild("counter").
        getText());
}

private LinkedList getList(Element el) {
        String hop,value;
        LinkedList branch = new LinkedList();
        int index1,index2;
        value = el.getChild ("Alert",nms).getChild ("AdditionalData",nms).
getText();

        if (value.equals("")){}
        else {
                index1 = value.lastIndexOf(',');
                if (index1 == -1){
                        branch.add(value);
                }
                else {
                        hop = value.substring(index1+1,value.length());
                        branch.add(hop);
                        index2 = value.lastIndexOf(',',index1-1);
                        while (index2 != -1) {
                                hop = value.substring(index2+1,index1);
                                branch.add(hop);
                                index1 = index2;
                                index2 = value.lastIndexOf(',',index2-1);
                        }
                        branch.add(value.substring(0,index1));
                }
        }
        value = el.getChild("Alert",nms) .getChild("Analyzer",nms)
.getAttribute("analyzerid").getValue();
        branch.add(value);
                value = el.getChild("Alert",nms).getChild("Source",nms)
 .getAttribute("ident").getValue();
        branch.add(value);
        return branch;
}

public void getTree() {
        String trg;
        int counter;

        trg=getTarget();
        logger.info("Target is: "+trg);
        counter=getCounter();
        logger.info("Counter is: "+counter);
        Tree _tree = new Tree(trg);
        List MsgList = doc.getRootElement().getChildren("IDMEF-
Message",nms);
```

```
                for (int i = 0; counter!=0 ; i++, counter--) {
                        Element msg = (Element) MsgList.get(i);
                        _tree.merge(getList(msg));
                        if (counter==1){
                        logger.info("TREE: ");
                        _tree.output(_tree.head);
                        }
                }
        }

}
```