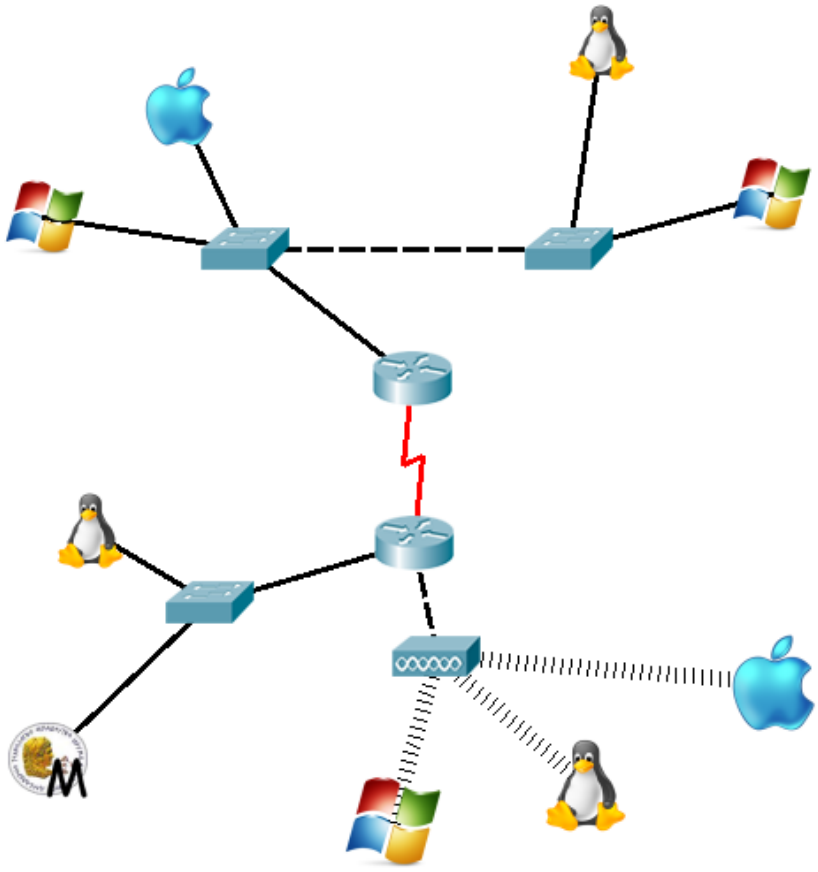
Πτυχιακή Εργασία:

Δημιουργία εφαρμογής για απομακρυσμένη διαχείριση των υπολογιστών των εργαστηρίων



Εισηγητής: Χαρχαλάκης Στέφανος

Επιβλέπων: Ψαρράς Νικόλαος

Κουλούρης Αθανάσιος-Σπυρίδων

ΑΜ: 05/2968

Μάιος 2012

Πρόλογος

Ένα μόνιμο θέμα με τους υπολογιστές των εργαστηρίων του Τμήματος είναι η διαχείρισή τους και συγκεκριμένα η επίβλεψη της κατάστασής τους και το θέμα του shutdown.

Η πτυχιακή αυτή εργασία βελτιώνει αυτό το θέμα δημιουργώντας μια client-server εφαρμογή η οποία θα διαχειρίζεται τους υπολογιστές των εργαστηρίων του Τμήματος.

Η πτυχιακή έγινε σε δύο στάδια: Αρχικά έγινε μελέτη της υπάρχουσας κατάστασης και απόπειρα χρήσης έτοιμων εφαρμογών (ssh κλπ) για απόκτηση εμπειρίας. Στη συνέχεια αναπτύχτηκε open source εφαρμογή σε Python. Ο client της εφαρμογής που αναπτύχτηκε είναι cross-platform και λειτουργεί τόσο σε Windows, όσο και σε Linux.

Περιεχόμενα

1. Εισαγωγή

Υπάρχουν διάφοροι τρόποι για να κλείσουμε τους υπολογιστές ενός δικτύου, οι οποίοι αναλύονται παρακάτω.

1.1 Shutdown μέσω μιας Remote Desktop εφαρμογής

Αν σε ένα IP δίκτυο είναι γνώστες οι IP διευθύνσεις των υπολογιστών που βρίσκονται σε λειτουργία, τότε ένας διαχειριστής μπορεί να συνδεθεί ξεχωριστά με κάθε έναν από αυτούς μέσω μιας εφαρμογής Remote Desktop (πχ. TightVNC) και να εκτελέσει Shutdown.

Ένας τρόπος για να μάθουμε τις διευθύνσεις των υπολογιστών που βρίσκονται σε λειτουργία είναι η χρήση του nmap:

```
athspk@athspk-VBox:~$ nmap -sP -n 192.168.56.0/24

Starting Nmap 5.21 ( http://nmap.org ) at 2011-12-05 05:31 EET
Nmap scan report for 192.168.56.200
Host is up (0.00041s latency).
Nmap scan report for 192.168.56.232
Host is up (0.0021s latency).
Nmap scan report for 192.168.56.233
Host is up (0.00031s latency).
```

Αυτός ο τρόπος είναι πολύ χρονοβόρος. Έστω και αν εξαιρέσουμε το χρόνο που απαιτείται για να τρέξει το nmap, η καθυστέρηση που προκύπτει αθροιστικά από τη σύνδεση σε κάθε έναν υπολογιστή, καθιστά αυτή τη μέθοδο αναποτελεσματική.

1.2 Shutdown μέσω ssh

Με την προϋπόθεση ότι οι διευθύνσεις των υπολογιστών είναι γνωστές και ότι σε όλους τους υπολογιστές τρέχει ένας ssh server, μπορούμε να εκτελέσουμε απομακρυσμένο shutdown.

Έστω ότι το αρχείο hosts.txt περιέχει διευθύνσεις, usernames και passwords των υπολογιστών που βρίσκονται σε λειτουργία στο δίκτυο.

```
athspk 192.168.56.232 password111

kubuntuvbox 192.168.56.233 password222
```

Μπορούμε να εκτελέσουμε απομακρυσμένο shutdown τρέχοντας το παρακάτω bash script:

```
cmdLin="sudo -S shutdown -h now"
cmdWin="shutdown -s -f"

while read line
do
   set $line
   ssh $1@$2 $cmdWin <&-
   if [ $? -ne 0 ]; then
      ssh $1@$2 "echo $3 | $cmdLin" <&-
   fi
done < hosts.txt
```

Εφόσον δεν είναι εκ των προτέρων γνωστό το λειτουργικό σύστημα που τρέχει ο κάθε υπολογιστής, το παραπάνω script διαβάζει την κάθε γραμμή του αρχείου hosts.txt και στη συνέχεια, δοκιμάζει να εκτελέσει την εντολή για το Windows. Αν η εντολή αποτύχει και επιστρέψει με exit status διάφορο του μηδενός, δοκιμάζεται η εντολή για το Linux.

Αυτός ο τρόπος έχει έναν περιορισμό. Αν οι διευθύνσεις των υπολογιστών ανατίθενται δυναμικά μέσω DHCP, δεν μπορούμε να γνωρίζουμε εκ των προτέρων τα usernames και passwords που αντιστοιχούν σε κάθε διεύθυνση, και ο μόνος τρόπος για να αντιμετωπιστεί αυτός ο περιορισμός είναι η χρήση username και password κοινού σε όλους τους υπολογιστές.

Πέραν αυτού, ένας άλλος λόγος που καθιστά αυτή τη μέθοδο αναποτελεσματική είναι το σχετικά υψηλό overhead που υπεισέρχεται αθροιστικά για κάθε μια TCP σύνδεση του ssh.

1.3 Shutdown μέσω της εφαρμογής της πτυχιακής εργασίας

Η εφαρμογή που υλοποιήθηκε στα πλαίσια της παρούσας πτυχιακής εργασίας απαλείφει τους περιορισμούς των εναλλακτικών μεθόδων που αναλύθηκαν προηγουμένως. Καθιστά δυνατό το απομακρυσμένο shutdown των υπολογιστών ενός IP δικτύου και υπερτερεί έναντι των προηγουμένων μεθόδων στα εξής σημεία:

- Λειτουργεί σε περιβάλλον δυναμικής διευθυνσιοδότησης (DHCP).
- Λειτουργεί ανεξαρτήτως IP version (IPv4 / IPv6).
- Είναι μια cross-platform εφαρμογή.
- Δεν απαιτείται προεργασία (nmap) για να συγκεντρωθούν οι διευθύνσεις των Η/Υ.
- Λειτουργεί χρησιμοποιώντας UDP, για να μην επιβαρύνει το δίκτυο με περιττό traffic.
- Λειτουργεί σε δίκτυα με πολύ χαμηλή τιμή MTU λόγω του μικρού μήκους των μηνυμάτων που ανταλλάσει.

2. Σχεδίαση

2.1 Γενικό πλάνο εφαρμογής

Η εφαρμογή αποτελείται από 2 μέρη, το Manager και τον Agent, και λειτουργεί ανεξαρτήτως λειτουργικού συστήματος και IP version. Σε ένα IP δίκτυο μπορεί να λειτουργούν πολλοί Managers και Agents. Οι Agents κάνουν JOIN σε 1 ή 2 προσυμφωνημένα Multicast Groups, ένα για το IPv4 και ένα για το IPv6 (αν υποστηρίζεται), που χρησιμοποιούνται για την ανακάλυψη των Agents από τους Managers. Ένας Manager ανακαλύπτει και καταγράφει τους Agents, και στη συνέχεια μπορεί να στείλει μήνυμα Restart ή Shutdown σε έναν ή περισσότερους.

Τα δύο αυτά μέρη υλοποιούν το πρωτόκολλο που σχεδιάστηκε στα πλαίσια της παρούσας πτυχιακής ώστε να επιτευχθεί η εξερεύνηση του δικτύου για την ανίχνευση των agents και την αποστολή εντολών τερματισμού και επανεκκίνησης σε αυτούς. Το πρωτόκολλο υλοποιείται στο επίπεδο εφαρμογής και για επικοινωνία με το δίκτυο κάνει χρήση του μηχανισμού υποδοχών (sockets). Για την αμφίδρομη ανταλλαγή unicast μηνυμάτων μεταξύ manager και agents χρησιμοποιούνται UDP sockets. Η διαδικασία του discovery γίνεται με multicast.

2.2 Γλώσσα Προγραμματισμού

Για την υλοποίηση της εφαρμογής χρησιμοποιήθηκε η γλώσσα Python η οποία είναι μια interpreted και dynamically typed γλώσσα προγραμματισμού. Η ενσωματωμένη βιβλιοθήκη της python που περιλαμβάνει πολλά έτοιμα εργαλεία, καθώς και το λιτό της συντακτικό, βοηθούν στην αύξηση της παραγωγικότητας.

Ο κώδικας ενός προγράμματος σε python:

- έχει λιγότερα dependencies εφόσον πολλά υποπρογράμματα που χρησιμοποιεί μια τυπική εφαρμογή περιλαμβάνονται στην ενσωματωμένη βιβλιοθήκη, βοηθώντας έτσι και στη λήψη αποφάσεων κατά τη φάση του σχεδιασμού.

- έχει μικρότερο μέγεθος και γι αυτό το λόγο είναι πιο ευανάγνωστος και ευκολότερος στη συντήρηση.

2.3 User Interface

Για το σχεδιασμό και την υλοποίηση του γραφικού περιβάλλοντος της εφαρμογής χρησιμοποιήθηκε η βιβλιοθήκη PyQt που αποτελεί τα bindings του QT Framework για την Python.

Η χρήση της QT καθιστά δυνατή την υλοποίηση cross-platform εφαρμογών, προγραμματίζοντας με βάση τα γεγονότα (Event-driven Programming). Η υλοποίηση του Event-driven programming στην QT γίνεται μέσω του μοντέλου Signal – Slot. Όταν ένα γεγονός εκπέμψει ένα signal, μπορούμε να το διαχειριστούμε μέσω μιας συνάρτησης την οποία έχουμε ορίσει ως slot, συνδέοντας (connect) το signal με το slot. Στο παράδειγμα που ακολουθεί, φαίνεται η σύνδεση του signal `aboutToQuit` με το slot `cleanUp`.

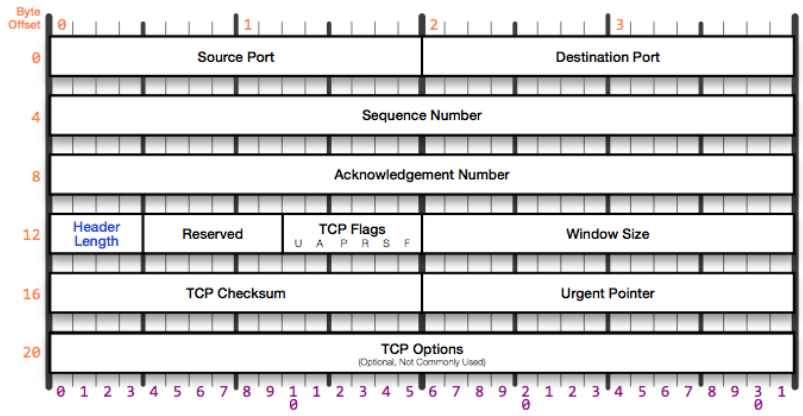
```
@pyqtSlot()
def cleanUp():
    print("will execute on close")

app.aboutToQuit.connect(cleanUp)
```

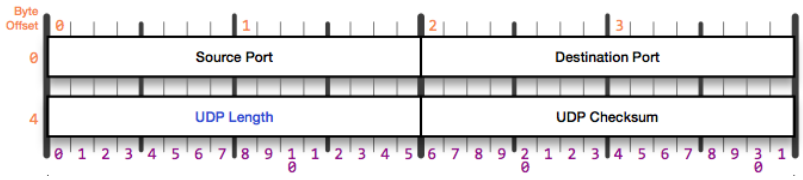
2.4 Πρωτόκολλο Επιπέδου Μεταφοράς

Στην εφαρμογή της παρούσας πτυχιακής εργασίας δεν διακινείται μεγάλος όγκος δεδομένων (πχ. μεταφορά αρχείων) και καμία λειτουργία δεν απαιτεί σύνδεση (TCP). Για αυτό το λόγο προτιμήθηκε το πρωτόκολλο UDP για την ανταλλαγή μηνυμάτων.

Το πρωτόκολλο UDP είναι ασυνδεσμικό και όπως φαίνεται στις εικόνες 2.1 και 2.2 πολύ ελαφρύ σε σύγκριση με το TCP διότι δεν εφαρμόζει τους μηχανισμούς αξιόπιστης επικοινωνίας που υπάρχουν στο δεύτερο, καθιστώντας έτσι το UDP αρκετά πιο γρήγορο. Όσον αφορά τους μηχανισμούς αξιοπιστίας, υλοποιούνται στο επίπεδο Εφαρμογής όπου απαιτείται.



Εικ. 2.1: TCP Header



Εικ. 2.2: UDP Header

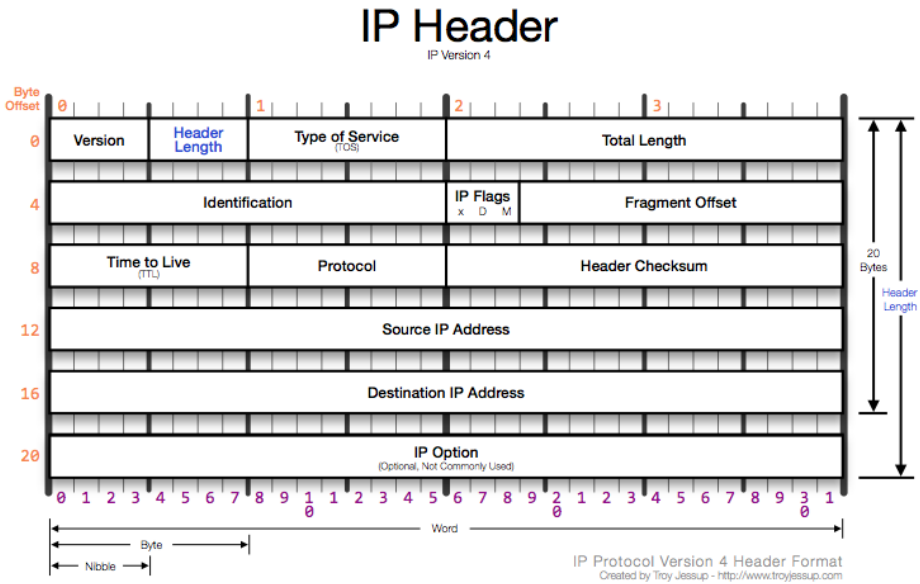
Για την αποστολή UDP datagrams χρειάζεται να δημιουργηθεί ένα UDP socket και να οριστεί η διεύθυνση στην οποία θα σταλούν τα δεδομένα καθώς και η θύρα υποδοχής (port number) ώστε η παράδοση στο άλλο άκρο να γίνει στη σωστή εφαρμογή. Η python απλοποιεί αυτή τη διαδικασία χρησιμοποιώντας μόνο 2 γραμμές κώδικα όπως φαίνεται στο παρακάτω code snippet.

```
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.sendto("hello".encode(), ("192.168.1.2", 12345))
```

2.5 Πρωτόκολλο Επιπέδου Δικτύου

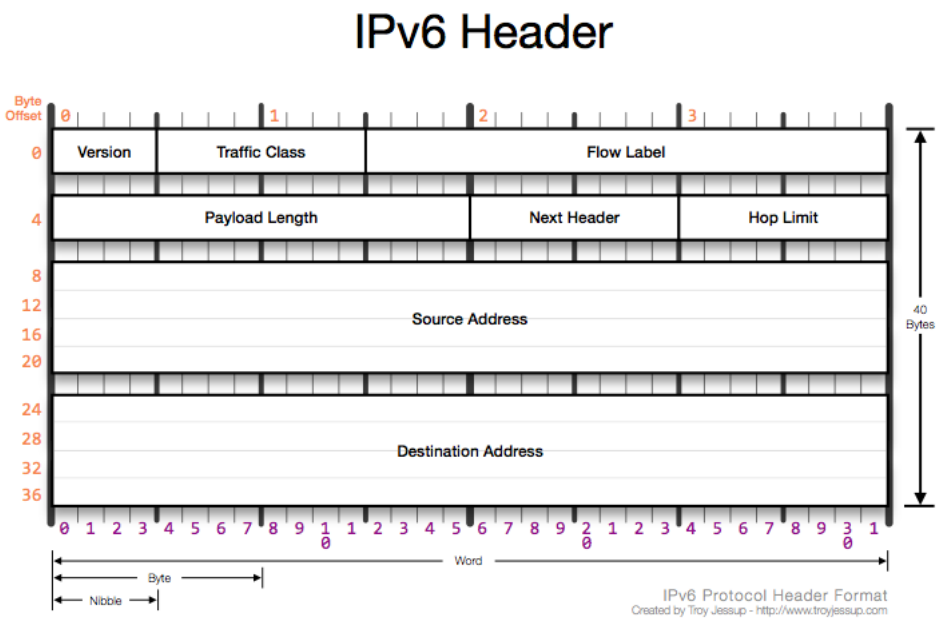
Η εφαρμογή της πτυχιακής εργασίας προορίζεται για χρήση σε IP δίκτυα και χρησιμοποιεί τα IPv4 και IPv6 ως πρωτόκολλα επιπέδου δικτύου για την επικοινωνία μεταξύ Manager – Agents.

Το IPv4 χρησιμοποιεί διευθύνσεις μήκους 32 bit για τον προσδιορισμό των πόρων του δικτύου, και η λογική αναπαράσταση αυτών των 32 bits σε ανθρωπίνως αναγνώσιμη μορφή είναι σε 4 ακεραίους από `0` έως `255` χωρισμένους με τελείες (πχ. `176.92.186.143`). Χρησιμοποιώντας αυτό το πρωτόκολλο μπορούν να αναπαρασταθούν 2^{32} = 4.294.967.296 διευθύνσεις. Το Header ενός IPv4 πακέτου φαίνεται στο παρακάτω σχήμα (Εικ. 2.3).



Εικ. 2.3: IPv4 Header

Το IPV6 δημιουργήθηκε για να αντιμετωπίσει το πρόβλημα εξάντλησης των διαθέσιμων IPv4 διευθύνσεων. Χρησιμοποιεί διευθύνσεις μήκους 128 bit αυξάνοντας το πλήθος των διευθύνσεων σε 3.4×10^{38} κατά προσέγγιση. Η λογική αναπαράσταση αυτών των διευθύνσεων είναι σε 8 το πολύ τμήματα ακεραίων από `0000` έως `ffff` στο δεκαεξαδικό σύστημα αρίθμησης, χωρισμένα με άνω-κάτω τελεία (πχ. `ff15:7079:7468:6f6e:6465:6d6f:6d63:6173`). Τα μηδενικά τμήματα μπορούν να παραληφθούν το πολύ μια φορά, για παράδειγμα η διεύθυνση `ff15:0000:0000:0000:0000:0000:0063:6173` μπορεί να γραφεί συνεπτυγμένα ως `ff15::63:6173`. Η δομή του Header ενός IPv6 πακέτου (Εικ. 2.4) είναι πιο απλή καθώς έχουν αφαιρεθεί ορισμένα πεδία με σκοπό την αποτελεσματικότερη επεξεργασία από τους δρομολογητές. (Deering, Cisco, Hinden, & Nokia, 1998, p. 2).

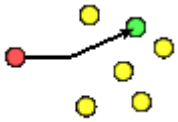


Εικ. 2.4: IPv6 Header

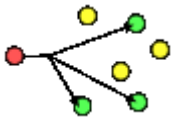
2.6 Μοντέλα προώθησης πακέτων

Στην εφαρμογή που αναπτύχτηκε, χρησιμοποιήθηκαν τα μοντέλα μονοδιανομής(unicast) και πολυδιανομής(multicast). Για την εξερεύνηση του δικτύου και την καταγραφή των agents (DISCOVERY), ο manager εκπέμπει ένα multicast μήνυμα σε τακτά χρονικά διαστήματα, ενώ για όλες τις άλλες ανταλλαγές μηνυμάτων χρησιμοποιείται unicast.

Όπως φαίνεται στις εικόνες 2.5 και 2.6, στο μοντέλο της μονοδιανομής(unicast) κάθε πακέτο προωθείται σε έναν συγκεκριμένο παραλήπτη, ενώ στο μοντέλο της πολυδιανομής(multicast) ένα αντίγραφο του πακέτου προωθείται σε όλα τα μέλη ενός ρητά προσδιορισμένου υποσυνόλου των κόμβων οι οποίοι έχουν επιλέξει να συμμετάσχουν στην ομάδα πολυδιανομής(multicast group).



Εικ. 2.5: Unicast



Εικ. 2.6: Multicast

Όσον αφορά το multicast, από την πλευρά του αποστολέα, πρέπει να οριστεί ως διεύθυνση παραλήπτη μια IPv4 διεύθυνση από το εύρος 224.0.0.0/4, ή μια IPv6 διεύθυνση από το εύρος ff00::/8. Οι διευθύνσεις σε αυτά τα εύρη έχουν ειδική σημασία και ονομάζονται Multicast Groups. Το UDP socket του αποστολέα δεν χρειάζεται να είναι μέλος του multicast group για να στείλει δεδομένα σε αυτό. Από την πλευρά του παραλήπτη, το UDP socket πρέπει να δεσμεύσει (bind) ένα port number ενός interface και στη συνέχεια να γίνει μέλος (join) του προσυμφωνημένου multicast group. Παρακάτω φαίνεται ο τρόπος με τον οποίο ένα socket κάνει join στο Multicast Group 239.1.2.3 στην Python.

```
mreq = struct.pack("!4sL", socket.inet_aton("239.1.2.3"),
socket.INADDR_ANY)
self.setsockopt(socket.IPPROTO_IP, socket.IP_ADD_MEMBERSHIP, mreq)
```

2.7 Ασφάλεια

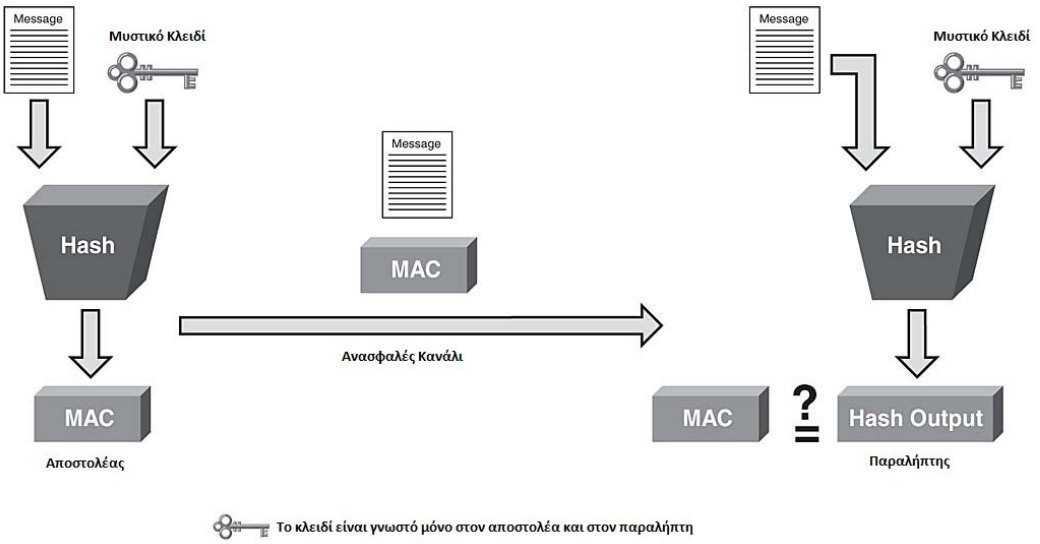
Για την αντιμετώπιση πιθανών απειλών κατά της ακεραιότητας και της αυθεντικότητας των μηνυμάτων που ανταλλάσσονται, αποφασίστηκε η χρήση Hash-based Message Authentication Code (HMAC).

Κάθε μήνυμα περνά μέσα από μια συνάρτηση κατακερματισμού (hash function) και σε συνδυασμό με ένα μυστικό κλειδί υπολογίζεται η hash value. Το μήκος του κλειδιού δεν πρέπει να είναι λιγότερο από το μήκος του hash value που παράγει η συνάρτηση, διαφορετικά υποβαθμίζεται η ασφάλεια που παρέχει αυτή η μέθοδος. (Krawczyk, Bellare, & Canetti, 1997, p. 4). Στον πίνακα 2.1 φαίνονται τα μήκη των hash values που παράγουν ορισμένες hash functions.

Πίνακας 2.1: Μήκη εξόδων διαφόρων hash functions.
("Cryptographic hash function", 2012)

Algorithm	Output size (bits)
MD4	128
MD5	128
SHA-0	160
SHA-1	160
SHA-256/224	256/224
SHA-512/384	512/384

Η εικόνα 2.5 απεικονίζει τον τρόπο με τον οποίο λειτουργεί ένα σύστημα που χρησιμοποιεί HMAC για την ανταλλαγή μηνυμάτων μεταξύ των μελών του. Ο αποστολέας και ο παραλήπτης συμμετέχουν στην επικοινωνία κατέχοντας το προσυμφωνημένο μυστικό κλειδί. Ο αποστολέας υπολογίζει το hash value του μηνύματος σε συνδυασμό με το κλειδί και στη συνέχεια στέλνει το μήνυμα και το hash value. Ο παραλήπτης λαμβάνει το μήνυμα και χρησιμοποιεί το ίδιο κλειδί για να υπολογίσει εκ νέου το hash value. Αν το hash value συμπίπτει με αυτό που έλαβε, συμπεραίνει πρώτον ότι το μήνυμα που έλαβε δεν αλλοιώθηκε και δεύτερον ότι το μήνυμα ελήφθη από εξουσιοδοτημένη πηγή.



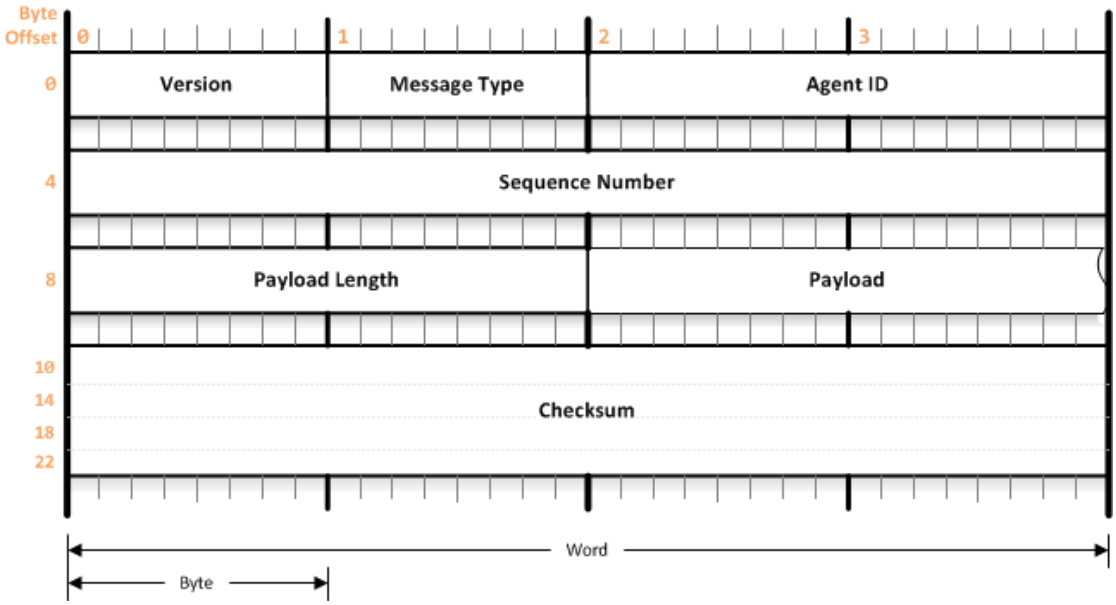
Εικ. 2.5: Απεικόνιση τρόπου λειτουργίας HMAC

Στο παράδειγμα που ακολουθεί, φαίνεται η διαδικασία υπολογισμού του HMAC με χρήση MD5 ως hash function.

```
msg = "message".encode()
key = "SecretKey"
hmac_md5 = hmac.new(bytes(key,"utf-8"), msg, hashlib.md5).digest()
```

3. Πρωτόκολλο

Το πρωτόκολλο που σχεδιάστηκε και υλοποιήθηκε στα πλαίσια της παρούσας πτυχιακής εργασίας αποτελείται από 2 βασικά μέρη. Το πρώτο μέρος αφορά τη διαδικασία ανακάλυψης ενεργών agents στο δίκτυο και λαμβάνει χώρα καθ' όλη τη διάρκεια λειτουργίας της εφαρμογής. Το δεύτερο μέρος αφορά την αποστολή μηνυμάτων τερματισμού και επανεκκίνησης στους επιλεγμένους από το χρήστη agents και πραγματοποιείται μετά από απαίτηση του χρήστη της εφαρμογής. Στην εικόνα 3.1 φαίνεται η μορφή των μηνυμάτων στο επίπεδο εφαρμογής, ενώ στις επόμενες ενότητες του κεφαλαίου ακολουθεί επεξήγηση των πεδίων του πρωτοκόλλου εφαρμογής και των τύπων μηνυμάτων που ανταλλάσσονται.



Εικ. 3.0.1: Application Layer PDU

3.1 Πεδία πρωτοκόλλου

Τα μηνύματα του πρωτοκόλλου της εφαρμογής αποτελούνται από 7 πεδία: Version, Message Type, Agent ID, Sequence Number, Payload Length, Payload και Checksum. Το ελάχιστο μήκος ενός μηνύματος είναι 26 bytes όταν δεν

υπάρχει Payload. Στον πίνακα 3.1 φαίνεται το μέγεθος κάθε πεδίου του πρωτοκόλλου της εφαρμογής.

Πίνακας 3.1: Μεγέθη πεδίων πρωτοκόλλου

Πεδίο	Μήκος
Version	1 byte
Message Type	1 byte
Agent ID	2 bytes
Sequence Number	4 bytes
Payload Length	2 bytes
Payload	Μεταβλητό
Checksum	16 bytes

Version

Το πεδίο Version έχει μέγεθος 1 byte και περιέχει ένα μη προσημασμένο ακέραιο αριθμό από 0 έως 255 που υποδηλώνει την έκδοση του πρωτοκόλλου της εφαρμογής. Ο σκοπός της ύπαρξης αυτού του πεδίου είναι η αποφυγή προβλημάτων συμβατότητας που μπορεί να προκύψουν από πιθανές μελλοντικές εκδόσεις του πρωτοκόλλου.

Όταν ο παραλήπτης λαμβάνει ένα μήνυμα, ελέγχεται πρώτα η έκδοση του πρωτοκόλλου. Αν η έκδοση του πρωτοκόλλου δεν υποστηρίζεται από την έκδοση της εφαρμογής, το μήνυμα απορρίπτεται και το γεγονός καταγράφεται στο αρχείο συμβάντων (Log file).

Message Type

Το πεδίο Message Type έχει μέγεθος 1 byte και περιέχει ένα μη προσημασμένο ακέραιο αριθμό από 0 έως 255 που υποδηλώνει τον τύπο του μηνύματος. Ο

σκοπός της ύπαρξης αυτού του πεδίου είναι να γνωρίζει η εφαρμογή τον τρόπο με τον οποίο θα επεξεργαστεί το κάθε μήνυμα.

Όταν ο παραλήπτης λαμβάνει ένα μήνυμα, ελέγχεται η τιμή αυτού του πεδίου. Αν η έκδοση της εφαρμογής υποστηρίζει τον τύπο μηνύματος που αναφέρεται στο πεδίο, τότε το μήνυμα προωθείται εσωτερικά για επεξεργασία στην αντίστοιχη συνάρτηση, διαφορετικά το μήνυμα απορρίπτεται και το γεγονός καταγράφεται στο αρχείο συμβάντων.

Agent ID

Το πεδίο Agent ID έχει μέγεθος 2 bytes και περιέχει ένα μη προσημασμένο ακέραιο αριθμό από 0 έως 65535 που υποδηλώνει τον κωδικό ενός agent. Ο σκοπός της ύπαρξης αυτού του πεδίου είναι να αναγνωρίζεται μοναδικά ο κάθε agent.

Το ζεύγος (IP address, port) είναι αρκετό για να αναγνωριστεί μοναδικά ένας agent, το γεγονός όμως ότι η εφαρμογή λειτούργει παράλληλα σε IPv4 και IPv6 καθιστά ανέφικτο για το manager να αναγνωρίσει μοναδικά έναν agent με αυτό τον τρόπο. Για παράδειγμα, τα ζεύγη (192.168.56.234 , 50000) και (fe80::abcd:1234 , 50001) ανήκουν μεν στον ίδιο agent, αλλά αν δεν υπήρχε το Agent ID τότε στον manager θα φαινόταν ως δυο διαφορετικοί.

Στην παρούσα φάση της υλοποίησης της εφαρμογής οι agents έχουν ID που ανήκει στο εύρος [1025,65535]. Σε αυτή τη φάση δεν υπάρχει λόγος να έχουν και οι managers μοναδικό κωδικό, παρόλα αυτά το εύρος [1,1024] έχει δεσμευτεί για πιθανή μελλοντική χρήση ως Manager ID. Το πεδίο έχει την τιμή 0 όταν το μήνυμα είναι multicast και απευθύνεται σε όλους όσους ανήκουν στο multicast group, η όταν διαπιστωθεί collision στο ID.

Όταν ο παραλήπτης λαμβάνει ένα μήνυμα, ελέγχεται η τιμή αυτού του πεδίου. Αν η τιμή δεν είναι 0 (Multicast) ή δεν συμπίπτει με το ID του παραλήπτη, τότε το μήνυμα απορρίπτεται.

Sequence Number

Το πεδίο Sequence Number έχει μέγεθος 4 bytes και περιέχει ένα μη προσημασμένο ακέραιο αριθμό από 0 έως 4.294.967.295 που υποδηλώνει τον αύξων αριθμό του μηνύματος. Ο σκοπός της ύπαρξης αυτού του πεδίου είναι να εξασφαλίζεται η χρονική ακολουθία της επικοινωνίας και να αναγνωρίζονται τα διπλότυπα πακέτα, καθώς το UDP δεν υλοποιεί μηχανισμούς αξιοπιστίας.

Κάθε παραλήπτης αποθηκεύει το Sequence Number του τελευταίου μηνύματος που έλαβε από μια συγκεκριμένη πηγή. Έτσι, όταν λαμβάνει ένα μήνυμα ελέγχεται η τιμή αυτού του πεδίου και αν είναι προγενέστερη της αποθηκευμένης το μήνυμα απορρίπτεται.

Payload Length

Το πεδίο Payload Length έχει μέγεθος 2 bytes και περιέχει ένα μη προσημασμένο ακέραιο αριθμό από 0 έως 65535 που υποδηλώνει το μέγεθος του πεδίου Payload. Ο σκοπός της ύπαρξης αυτού του πεδίου είναι να γνωρίζει η εφαρμογή κατά τη φάση τεμαχισμού του μηνύματος τα όρια των δεδομένων που μεταφέρει, εφόσον το πεδίο Payload έχει μεταβλητό μέγεθος.

Payload

Το πεδίο Payload έχει μεταβλητό μέγεθος καθώς μεταφέρει διαφορετικά δεδομένα για κάθε τύπο μηνύματος. Για παράδειγμα, το μήνυμα `DISCOVERY_ASK` δεν περιέχει καθόλου δεδομένα, ενώ το μήνυμα `DISCOVERY_REPLY` περιέχει δεδομένα σχετικά με την έκδοση της εφαρμογής και το είδος του λειτουργικού συστήματος στο οποίο λειτούργει ο agent.

Checksum

Το πεδίο Checksum έχει μέγεθος 16 bytes εφόσον χρησιμοποιείται η συνάρτηση κατακερματισμού `HMAC-MD5` για τον υπολογισμό του. Αν απαιτείται μεγαλύτερη

ασφάλεια, ο HMAC μπορεί να συνδυαστεί και με άλλες συναρτήσεις (SHA1, SHA256, κτλ.) αυξάνοντας όμως παράλληλα και το μέγεθος του πεδίου. Ο σκοπός της ύπαρξης αυτού του πεδίου είναι να διαπιστώσει ο παραλήπτης πρώτον εάν το μήνυμα που ελήφθη αλλοιώθηκε και δεύτερον εάν το μήνυμα ελήφθη από εξουσιοδοτημένη πηγή.

Ο αποστολέας υπολογίζει τη hash value του μηνύματος σε συνδυασμό με ένα προσυμφωνημένο μυστικό κλειδί και στη συνέχεια στέλνει το μήνυμα το οποίο περιέχει τη hash value στο πεδίο Checksum. Ο παραλήπτης λαμβάνει το μήνυμα και χρησιμοποιεί το ίδιο κλειδί για να υπολογίσει εκ νέου τη hash value. Αν η hash value δεν είναι ίδιες το μήνυμα απορρίπτεται.

3.2 Τύποι μηνυμάτων

Στον πίνακα 3.2 φαίνονται οι τύποι μηνυμάτων του πρωτοκόλλου. Από αυτά, δεν υλοποιήθηκαν τα DISCOVERY_PATH_MTU_TRY, DISCOVERY_PATH_MTU_ACK, και DISCOVERY_MANAGER_ID_EXISTS (βλ. Κεφ. 5 Προτάσεις-Συμπεράσματα). Τα μηνύματα που υλοποιούνται στο πρωτόκολλο αναλύονται παρακάτω.

Πίνακας 3.2: Τύποι μηνυμάτων εφαρμογής

Message Type	Value
DISCOVERY_PATH_MTU_TRY	11
DISCOVERY_PATH_MTU_ACK	12
DISCOVERY_ASK	21
DISCOVERY_REPLY	22
DISCOVERY_AGENT_ID_EXISTS	23
DISCOVERY_MANAGER_ID_EXISTS	24
DISCOVERY_REPLY_USER	25
EXEC_OP_RESTART	31
EXEC_OP_SHUTDOWN	32

DISCOVERY_ASK

Είναι το πρώτο μήνυμα που στέλνει ο manager όταν τεθεί σε λειτουργία, και συνεχίζει να το στέλνει σε τακτά χρονικά διαστήματα καθ' όλη τη διάρκεια της λειτουργιάς του. Για την ακρίβεια στέλνεται διπλό, ένα σε IPv4 και ένα σε IPv6. Αποστέλλεται ως multicast και λαμβάνεται από όλους όσους ανήκουν στο multicast group. Ο manager στέλνοντας αυτό το μήνυμα γνωστοποιεί την παρουσία του στο δίκτυο και απαιτεί απαντήσεις από όλους τους agents που το λαμβάνουν.

DISCOVERY_REPLY

Το μήνυμα `DISCOVERY_REPLY` έχει χαρακτήρα επιβεβαίωσης και είναι η απάντηση του agent σε κάθε `DISCOVERY_ASK` που λαμβάνει. Επιπλέον, παρέχει δεδομένα σχετικά με το ID του, την έκδοση της εφαρμογής, το είδος του λειτουργικού συστήματος στο οποίο λειτούργει ο agent, και 2 τυχαίους αριθμούς από 0 έως 255 που βοηθούν στην αναγνώριση των collisions στα Agent IDs.

DISCOVERY_AGENT_ID_EXISTS

Όταν ένας agent τεθεί σε λειτουργία επιλέγει τυχαία ένα ID στο εύρος [1025,65535] το οποίο θα επιχειρήσει να χρησιμοποιήσει κατά την επικοινωνία του με τον manager. Αν ο manager έχει προηγουμένως λάβει απάντηση από άλλον agent με το ίδιο ID, τότε στέλνει μήνυμα `DISCOVERY_AGENT_ID_EXISTS` σε όποιον προσπαθήσει να επικοινωνήσει με το ήδη καταγεγραμμένο Agent ID, έτσι ώστε να επιλέξει ένα άλλο.

DISCOVERY_REPLY_USER

Το μήνυμα `DISCOVERY_REPLY_USER` περιέχει το όνομα χρήστη που έθεσε σε λειτουργία τον agent. Αυτό το μήνυμα έχει καθαρά πληροφοριακό χαρακτήρα και αν και ανήκει στην κατηγορία Discovery, δεν συμβάλλει με κανένα τρόπο στη λειτουργία εξερεύνησης του δικτυού.

EXEC_OP_RESTART

Το μήνυμα `EXEC_OP_RESTART` στέλνεται μετά από απαίτηση του χρήστη σε ένα συγκεκριμένο agent για να εκτελέσει επανεκκίνηση του υπολογιστή.

EXEC_OP_SHUTDOWN

Το μήνυμα `EXEC_OP_SHUTDOWN` στέλνεται μετά από απαίτηση του χρήστη σε ένα συγκεκριμένο agent για να τερματίσει τη λειτουργία του υπολογιστή.

4. Λειτουργία

Σε αυτό το κεφάλαιο περιγράφεται αναλυτικά η λειτουργία όλων των συστατικών μερών που αποτελούν την εφαρμογή, καθώς επίσης και οι τρόποι αντιμετώπισης διαφόρων προβλημάτων που μπορούν να προκύψουν κατά τη λειτουργία.

4.1 Γενικά

Η εφαρμογή αποτελείται από 2 μέρη, το Manager και τον Agent, και λειτουργεί ανεξαρτήτως λειτουργικού συστήματος και IP version. Σε ένα IP δίκτυο μπορεί να λειτουργούν πολλοί Managers και Agents. Οι Agents κάνουν JOIN σε 1 ή 2 προσυμφωνημένα Multicast Groups, ένα για το IPv4 και ένα για το IPv6 (αν υποστηρίζεται), που χρησιμοποιούνται για την ανακάλυψη των Agents από τους Managers. Ένας Manager ανακαλύπτει και καταγράφει τους Agents, και στη συνέχεια μπορεί να στείλει μήνυμα Restart ή Shutdown σε έναν ή περισσότερους.

Σε υπολογιστές με παλιό λειτουργικό σύστημα ενδέχεται να μην υπάρχει by default υποστήριξη για το IPv6, αυτό όμως δεν επηρεάζει καθόλου τη λειτουργία της εφαρμογής. Αν παρόλα αυτά κριθεί απαραίτητη για οποιοδήποτε λόγο η παράλληλη λειτουργία και στα δυο IP versions, θα πρέπει να γίνει εγκατάσταση του IPv6 στο λειτουργικό σύστημα από το χρήστη. Στο Windows XP αυτό γίνεται από το περιβάλλον `netsh` όπως φαίνεται παρακάτω. ("Netsh commands for Interface IPv6", 2005).

```
C:\>netsh interface ipv6 install
Ok.
```

4.2 Απαιτήσεις Συστήματος

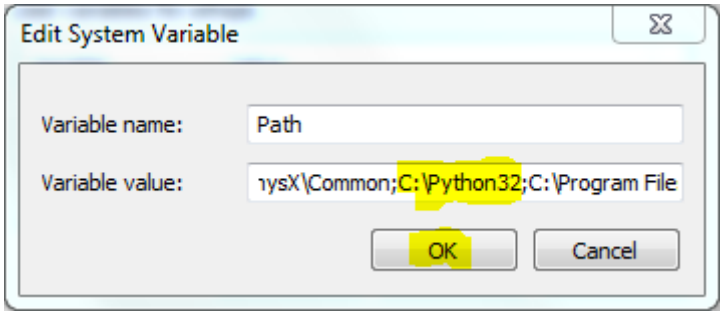
Για τη λειτουργία της εφαρμογής απαιτείται:

- να είναι εγκατεστημένη η Python, έκδοση 3.2.x ή μεταγενέστερη. (http://python.org/download/).
- να είναι εγκατεστημένη η PyQt, έκδοση 4.8.x ή μεταγενέστερη. (http://www.riverbankcomputing.co.uk/software/pyqt/download).
- οι Routers του δικτύου και τα firewalls των υπολογιστών να επιτρέπουν την unicast και multicast κίνηση UDP datagrams σε IPv4 ή/και IPv6.
- Να υποστηρίζεται στο Linux η αποστολή μηνυμάτων στο D-Bus (`dbus-send`).

Παρατηρήσεις:

Μετά την εγκατάσταση της Python στο Windows, το path για το python.exe δεν προστίθεται αυτόματα στο system path. Μπορεί όμως να προστεθεί από το χρήστη είτε μέσω command line με την εντολή `PATH=%PATH%;C:\Python32` ή από το Control Panel (Control Panel → System → Environment Variables), όπως φαίνεται παρακάτω.

```
C:\>echo %PATH%
C:\WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\System32\Wbem
C:\>PATH=%PATH%;C:\Python32
C:\>echo %PATH%
C:\WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\System32\Wbem;C:\Python32
```



Εικ. 4.1: Προσθήκη path για την Python.

4.3 Αρχικοποίηση (Initialization)

Σε αυτή την ενότητα αναλύονται τα στάδια που προηγούνται και πρέπει να ολοκληρωθούν ώστε να μπορέσει να ξεκινήσει η κυρίως λειτουργία της εφαρμογής.

4.3.1 Logger

Αρχικοποιείται πρώτα ο logger της εφαρμογής, έτσι ώστε να μπορούν να καταγράφουν τα οποιαδήποτε σφάλματα προκύψουν στη συνέχεια. Ο logger συνδέεται με 2 handlers, ένα StreamHandler για την εκτύπωση μηνυμάτων στο stdout, και ένα RotatingFileHandler για την καταγραφή μηνυμάτων στο αρχείο συμβάντων.

Τα log files αποθηκεύονται στον υποφάκελο “logs” της εφαρμογής. Αν το μέγεθος του αρχείου φτάσει ένα προκαθορισμένο μέγεθος (maxBytes), τότε το τρέχον αρχείο κλείνει, μετονομάζεται προθέτοντας έναν αριθμό στο τέλος (πχ. `logfile.txt.1`) και δημιουργείται νέο αρχείο. Πάντα όμως τηρείται το μέγιστο πλήθος που έχει οριστεί (backupCount).

Ο logger που χρησιμοποιήθηκε ανήκει στην ενσωματωμένη βιβλιοθήκη της Python, είναι Thread-safe και αρχικοποιείται με απλό τρόπο όπως φαίνεται παρακάτω.

```
import logging
log = logging.getLogger("appName")
```

4.3.2 Ανάγνωση παραμέτρων

Μετά την αρχικοποίηση του logger σειρά έχει η ανάγνωση των παραμέτρων της εφαρμογής, οι οποίες θα χρησιμοποιηθούν για τις περεταίρω λειτουργία.

Για τη διασφάλιση της ακεραιότητας λειτουργίας του συστήματος, οι παράμετροι βρίσκονται σε 2 σημεία, hardcoded στον κώδικα της εφαρμογής και σε αρχείο ρυθμίσεων (`settings.ini`). Η πρωτεύουσα πηγή ανάγνωσης των παραμέτρων είναι το αρχείο ρυθμίσεων, ενώ οι hardcoded παράμετροι λειτουργούν ως failsafe. Τα περιεχόμενα του αρχείου ρυθμίσεων είναι οργανωμένα σε sections όπως φαίνεται παρακάτω.

```
[LOGGING]
Backup_files=2
KiB_per_file=512

[SECURITY]
HMAC-MD5_key=Anything_above_16_characters

[MULTICAST]
Group_v4=239.6.6.6
Port_v4=50000
TTL_v4=32
Group_v6=ff15:7079:7468:6f6e:6465:6d6f:6d63:6173
Port_v6=50001
TTL_v6=32

[DISCOVERY]
Interval=3000
Inactivity_Counter=2
```

Μετά την ανάγνωση των περιεχομένων του αρχείου ρυθμίσεων, οι παράμετροι ελέγχονται ως προς την ορθότητά τους. Ο έλεγχος ορθότητας γίνεται αρχικά ως προς τον τύπο της παραμέτρου και στη συνέχεια ως προς την τιμή. Πρόκειται για μια υποτυπώδη τεχνική αμυντικού προγραμματισμού για τη διασφάλιση της σωστής λειτουργίας της εφαρμογής σε σημεία όπου ο χρήστης μπορεί να εισάγει λάθος τιμές. Οι κανόνες εγκυρότητας είναι οι εξής (πίνακας 4.1):

Πίνακας 4.1: Κανόνες εγκυρότητας παραμέτρων

Παράμετρος	Κανόνας Εγκυρότητας
Backup_files	> 0
KiB_per_file	> 0
HMAC-MD5_key	Πλήθος χαρακτήρων >= 16
Group_v4	Να είναι έγκυρη IPv4 διεύθυνση στο εύρος 224.0.0.0/4
Port_v4	1024 <= port number <= 65535
TTL_v4	0 < TTL <= 255
Group_v6	Να είναι έγκυρη IPv6 διεύθυνση στο εύρος ff00::/8
Port_v6	1024 <= port number <= 65535
TTL_v6	0 < TTL <= 255
Interval	> 1000 ms
Inactivity_Counter	> 0

Αν η τιμή κάποιας παραμέτρου δεν ικανοποιεί τον κανόνα εγκυρότητάς της, τότε χρησιμοποιείται η αντίστοιχη default τιμή. Οι default τιμές χρησιμοποιούνται και στην περίπτωση που το αρχείο δεν υπάρχει ή δεν είναι δυνατή η ανάγνωσή του.

4.3.3 UDP Server

Ο UDP Server αρχικοποιείται με τις παραμέτρους που δοθήκαν στην προηγούμενη φάση και είναι ένα thread που διαχειρίζεται 2 UDP Sockets. Η αρχικοποίηση του thread γίνεται με τον παρακάτω τρόπο:

```
def initThread():
    asyncore.loop()

srvThread = threading.Thread(target=initThread)
srvThread.name = "UDPSrvThread"
srvThread.start()
```

Η εφαρμογή προσπαθεί πρώτα να δημιουργήσει ένα UDP socket για το IPv6 και στη συνέχεια ένα για το IPv4. Αν η δημιουργία του socket για το IPv4

αποτύχει, τότε το γεγονός καταγράφεται στο αρχείο συμβάντων και η εφαρμογή τερματίζεται. Γενικά είναι επιθυμητό να λειτουργούν και τα δυο sockets γιατί εξασφαλίζεται με αυτό τον τρόπο καλύτερη «ορατότητα». Για παράδειγμα, ένας agent που λειτουργεί μόνο σε IPv6 δεν θα είναι ορατός σε ένα manager που λειτουργεί σε IPv4. Σχετικά με το IPv6, παρουσιαστήκαν ορισμένα προβλήματα σχετικά με την επιβολή της cross-platform απαίτησης της πτυχιακής εργασίας, τα όποια όμως αντιμετωπίστηκαν (βλ. Κεφ. Προτάσεις - Συμπεράσματα).

Σε όλη τη διάρκεια λειτουργίας της εφαρμογής, το thread του UDP server παρακολουθεί τα sockets και αν υπάρχουν δεδομένα για ανάγνωση τα προωθεί εσωτερικά στην εφαρμογή για επεξεργασία, ενώ αν υπάρχουν δεδομένα για αποστολή, αναλαμβάνει να τα προωθήσει στα sockets.

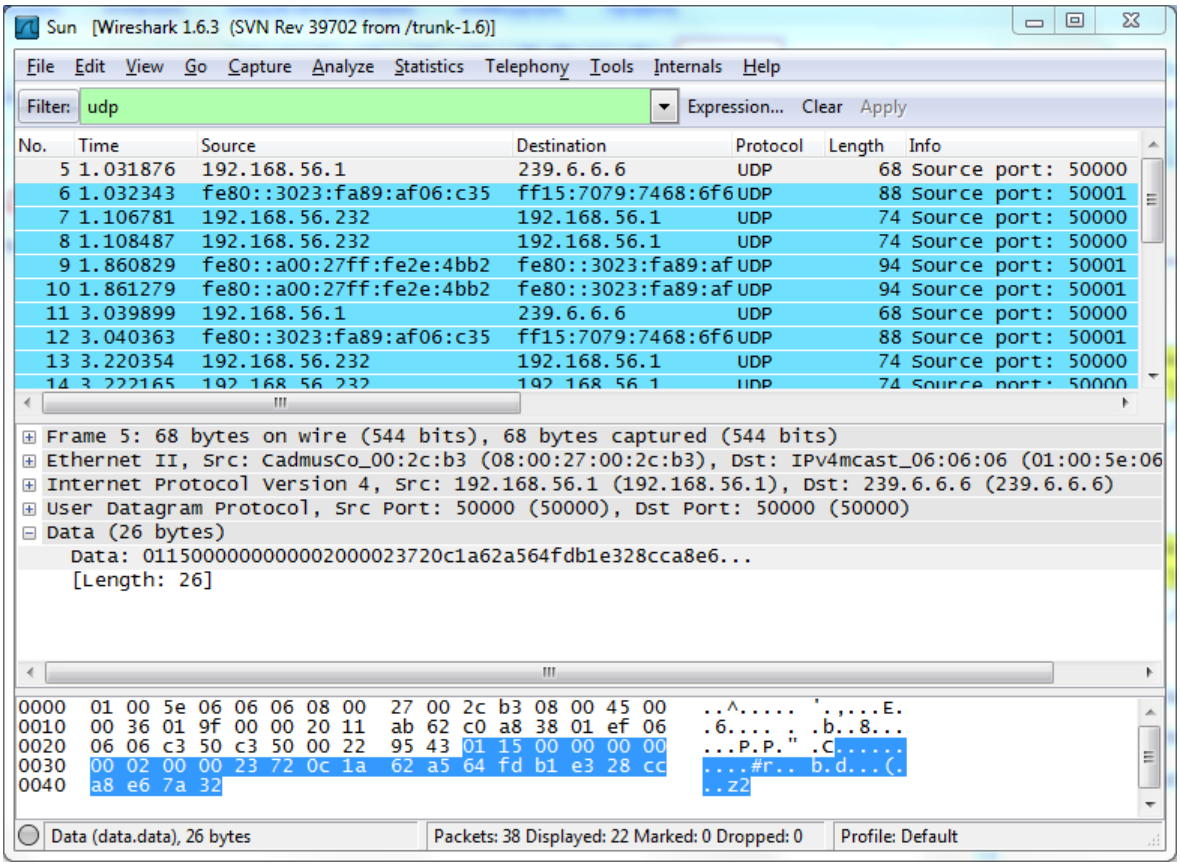
4.4 Discovery

Η διαδικασία εξερεύνησης του δικτύου για την ανακάλυψη των agents αποτελεί βασικό συστατικό της εφαρμογής. Σε ένα δίκτυο όπου οι διευθύνσεις των υπολογιστών ανατίθενται δυναμικά μέσω DHCP, δεν θα υπήρχε τρόπος να γνωρίζει ο manager τις διευθύνσεις των agents, εκτός αν ο manager είχε στατική διεύθυνση και οι agents έστελναν αιτήσεις στην προσυμφωνημένη διεύθυνση. Αυτό όμως θα περιόριζε το manager να λειτουργεί μόνο σε συγκεκριμένο υπολογιστή.

Σε όλη τη διάρκεια λειτουργίας του manager, εκπέμπεται ένα multicast μήνυμα `DISCOVERY_ASK` στο multicast group σε τακτά χρονικά διαστήματα. Το μήνυμα αυτό φαίνεται παρακάτω στον πίνακα 4.2, ενώ η εικόνα 4.2 δείχνει την ανάλυση του μηνύματος στο Wireshark.

Πίνακας 4.2: DISCOVERY_ASK

Version	Message Type	Agent ID	Seq No	Payload Length	Checksum
1	21	0	1	0	1234abcd…



Εικ. 4.2: DISCOVERY_ASK μέσα από το Wireshark

Οι agents που λαμβάνουν αυτό το μήνυμα απαντούν αρχικά με ένα μήνυμα `DISCOVERY_REPLY` και ύστερα με ένα `DISCOVERY_REPLY_USER`.

Οι agents κατά την έναρξη της λειτουργίας τους επιλέγουν ένα τυχαίο Agent ID (βλ. Κεφ. 3.1). Αυτή γίνεται για τη μείωση της κίνησης στο δίκτυο, γιατί αλλιώς ο manager θα έπρεπε να αποδώσει ID ξεχωριστά για κάθε agent. Εκτός από το ID, οι agents επιλέγουν άλλους 2 τυχαίους αριθμούς από 0 έως 255, οι οποίοι χρησιμοποιούνται από το manager για την ανίχνευση και επίλυση συγκρούσεων σε περίπτωση που δυο ή περισσότεροι agents επιλέξουν το ίδιο ID.

Έστω ότι λειτουργούν οι παρακάτω 2 agents στο δίκτυο τη στιγμή που εστάλη το `DISCOVERY_ASK`. (Πίνακας 4.3).

Πίνακας 4.3: Στοιχεία των agents κατά το Discovery.

Username	Agent ID	failsafe 1	failsafe 2	OS
User1	1234	24	233	Windows
User2	5678	123	181	Linux

Ο τύπος του λειτουργικού συστήματος δεν αποστέλλεται ως string αλλά ως μονοψήφιος μη προσημασμένος ακέραιος. Συγκεκριμένα, οι κωδικοί 1,2,3 αντιστοιχούν στα λειτουργικά συστήματα Windows, Linux και Macintosh. Έχοντας υπ' όψη τα παραπάνω, τα μηνύματα που θα λάβει ο manager από αυτούς τους 2 agents θα είναι τα έξης (πίνακες 4.4-4.7):

Πίνακας 4.4: DISCOVERY_REPLY

Version	Message Type	Agent ID	Seq No	Payload Length	Payload	Checksum
1	22	1234	1	6	101124233	1234abcd…

Πίνακας 4.5: DISCOVERY_REPLY_USER

Version	Message Type	Agent ID	Seq No	Payload Length	Payload	Checksum
1	25	1234	2	5	User1	3456cdef…

Πίνακας 1.6: DISCOVERY_REPLY

Version	Message Type	Agent ID	Seq No	Payload Length	Payload	Checksum
1	22	5678	1	6	1012123181	abcd4321…

Πίνακας 4.7: DISCOVERY_REPLY_USER

Version	Message Type	Agent ID	Seq No	Payload Length	Payload	Checksum
1	25	5678	2	5	User2	1a2b3c4d…

Για λόγους απλοποίησης του παραδείγματος δεν φαίνονται τα μηνύματα για το IPv6. Στον παρακάτω πίνακα φαίνονται τα στοιχεία που έχει συγκεντρώσει ο manager μετά τη λήψη των replies.

Πίνακας 4.8: Δεδομένα που προκύπτουν κατά τη φάση του DISCOVERY

Agent ID	User name	Fsafe 1	Fsafe 2	Ver.	OS code	Time Out	Latest SeqNo	IPv4 address	IPv4 port	IPv6 address	IPv6 port
1234	User1	24	233	1.0.1	1	2	2	10.0.0.1	50000	fe80::dec0:1234	50001
5678	User2	123	181	1.0.1	2	2	2	10.0.0.2	50000	fe80::2ec1:5678	50001

4.4.1 Interval

Η διαδικασία του DISCOVERY επαναλαμβάνεται σε τακτά χρονικά διαστήματα που ορίζονται από την παράμετρο `Interval` στο αρχείο ρυθμίσεων. Αυτό γίνεται για να ενημερώνεται η δομή δεδομένων (πινάκας 4.8) που διατηρεί ο manager, ώστε να προστίθενται νέοι agents και να αφαιρούνται οι ανενεργοί. Όπως φαίνεται στον πίνακα 4.1 (Κανόνες εγκυρότητας παραμέτρων), το `Interval` του Discovery παίρνει τιμές άνω των 1000 ms. Αυτό το χρονικό διάστημα, όπως εξηγείται και στην ενότητα 4.4.3, δεσμεύεται για το random delay των απαντήσεων των agents. Παρακάτω φαίνεται το τμήμα κώδικα που είναι υπεύθυνο για την εκκίνηση κάθε κύκλου Discovery.

```
self.discoveryTimer = QTimer()
self.discoveryTimer.timeout.connect(self.triggerDiscovery)
self.discoveryTimer.setInterval(DiscoveryProtocol.INTERVAL)
self.discoveryTimer.start()

def triggerDiscovery(self):
    self.decrementAllTimeouts()
    self.removeInactiveAgents()
    self.send_msg_discovery_ask()
```

Η default τιμή του `Interval` είναι 5000 ms. Ο χρήστης έχει τη δυνατότητα να αλλάξει αυτή την τιμή, αλλά θα πρέπει να ληφθούν υπ' όψη τα εξής:

- Όταν το `Interval` είναι μικρό, το σύστημα θα έχει γρηγορότερη απόκριση, καθώς οι νέοι agents αλλά και οι ανενεργοί θα γίνονται αντιληπτοί γρηγορότερα, αυξάνοντας όμως παράλληλα και την κίνηση στο δίκτυο.
- Όταν αντιθέτως το `Interval` είναι μεγάλο, οι αλλαγές των καταστάσεων των agents θα χρειάζονται περισσότερο χρόνο για να γίνουν αντιληπτές, αλλά μειώνεται παράλληλα η κίνηση στο δίκτυο.

4.4.2 Timeout

Η αφαίρεση των ανενεργών agents γίνεται με την βοήθεια της τιμής του πεδίου timeout (πίνακας 4.8), και της παραμέτρου `Inactivity_Counter` (πίνακας 4.1) του αρχείου ρυθμίσεων.

Όταν κατά το Discovery προστίθεται ένας νέος agent το πεδίο timeout παίρνει την τιμή του `Inactivity_Counter`, του οποίου η default τιμή είναι 2. Κατά την εκκίνηση κάθε νέου κύκλου Discovery η τιμή του πεδίου timeout μειώνεται κατά 1, ενώ με κάθε εισερχόμενο μήνυμα `DISCOVERY_REPLY` το timeout παίρνει ξανά την τιμή του `Inactivity_Counter`. Αν κατά τη μείωση του timeout το πεδίο έχει αρνητική τιμή, τότε ο συγκεκριμένος agent προστίθεται στην λίστα των ανενεργών και διαγράφεται από τον manager. Στον πίνακα 4.9 φαίνονται οι αλλαγές του πεδίου timeout που κάνει ο manager κατά τη φάση του Discovery. Για την απλούστευση του παραδείγματος θεωρούμε ότι η επικοινωνία γίνεται μόνο σε ένα IP version, επίσης παραλείπονται τα μηνύματα `DISCOVERY_REPLY_USER`.

Πίνακας 4.9: Οι αλλαγές του πεδίου timeout κατά το Discovery

Κατεύθυνση	Μήνυμα	Timeout	Παρατηρήσεις
Sent	`DISCOVERY_ASK`	N/A	Ο manager στέλνει το πρώτο ASK
Received	`DISCOVERY_REPLY`	2	Ο agent απάντησε (timeout=2)
Sent	`DISCOVERY_ASK`	1	Νέος κύκλος Discovery (timeout - 1)
Received	`DISCOVERY_REPLY`	2	Ο agent απάντησε (timeout=2)
Sent	`DISCOVERY_ASK`	1	Νέος κύκλος Discovery (timeout - 1)
Sent	`DISCOVERY_ASK`	0	Νέος κύκλος Discovery (timeout - 1)
Sent	`DISCOVERY_ASK`	-1	Ο agent αφαιρείται από τη λίστα

Στο παραπάνω παράδειγμα, ο manager ξεκινά το Discovery για να ανακαλύψει τους agents. Έρχεται ένα `DISCOVERY_REPLY` από έναν agent ο οποίος προστίθεται στη λίστα με timeout=2. Στη συνέχεια ξεκινά ο νέος κύκλος Discovery και το timeout μειώνεται κατά 1 (timeout=1). Ο agent απαντά ξανά και το timeout αρχικοποιείται (timeout=2). Ο επόμενος κύκλος Discovery ξεκινά, το timeout μειώνεται κατά 1 (timeout=1), αλλά αυτή τη φορά ο agent δεν απαντά. Ομοίως, ξεκινά ο επόμενος κύκλος Discovery, το timeout μειώνεται κατά 1

(timeout=0) και είναι η δεύτερη φορά που δεν λαμβάνεται απάντηση από τον agent. Στην αρχή του επόμενου κύκλου Discovery, το timeout μειώνεται κατά 1 (timeout=-1), ο agent θεωρείται πλέον ανενεργός και διαγράφεται από τη λίστα του manager.

4.4.3 Random Delay

Το random delay είναι η σκόπιμη τυχαία καθυστέρηση στις απαντήσεις των agents πριν από κάθε `DISCOVERY_REPLY`. Οι agents επιλέγουν έναν τυχαίο μη προσημασμένο ακέραιο στο εύρος [0,1000] που αντιπροσωπεύει το χρονικό διάστημα σε milliseconds που θα περιμένουν μέχρι να απαντήσουν. Αυτή η λειτουργία υλοποιήθηκε με σκοπό την αποφυγή καταστάσεων συμφόρησης στον manager, αφού σε διαφορετική περίπτωση όλες οι απαντήσεις θα ερχόταν σχεδόν ταυτόχρονα. Η συμφόρηση δεν θα ήταν αντιληπτή με μικρό πλήθος agents, θα αποτελούσε όμως πρόβλημα με την αύξηση του πλήθους.

Όσον αφορά τον κανόνα εγκυρότητας του πεδίου `Interval` (>1000 ms), ο περιορισμός τέθηκε με σκοπό τη διασφάλιση της λήψης των απαντήσεων πριν από την έναρξη κάθε νέου κύκλου Discovery. Αν δεν υπήρχε αυτός ο περιορισμός, θα υπήρχαν περιπτώσεις με δίπλες απαντήσεις από κάποιους agents, γεγονός το οποίο θα εμπόδιζε την ομαλή λειτουργία του manager.

4.4.4 ID collisions

Το πεδίο Agent ID περιέχει ένα μη προσημασμένο ακέραιο από 1025 έως 65535 που χρησιμοποιείται για να αναγνωρίζεται μοναδικά ο κάθε agent. Αυτός ο αριθμός επιλέγεται τυχαία κατά την έναρξη της λειτουργίας του agent. Υπάρχει όμως η πιθανότητα 2 ή περισσότεροι agents να επιλέξουν το ίδιο ID. Σε αυτή την ενότητα αναλύεται ο τρόπος ανίχνευσης και επίλυσης αυτών των συγκρούσεων

4.4.4.1 True Positive

Έστω ότι λειτουργούν στο δίκτυο 2 agent με τα παρακάτω στοιχεία (πίνακας 4.10):

Πίνακας 4.10: Παράδειγμα ID collision (True Positive)

Agent ID	Fsafe 1	Fsafe 2	IPv4 address	IPv4 port
1234	24	233	10.0.0.1	50000
1234	123	181	10.0.0.2	50000

Ο πρώτος agent απαντά πρώτος με ένα `DISCOVERY_REPLY` με τα στοιχεία του και ο manager τον αποθηκεύει στη δομή δεδομένων του. Στη συνέχεια απαντά και ο δεύτερος agent, και σε αυτό το σημείο ο manager ανιχνεύει ID collision, εφόσον ο δεύτερος απάντησε με ένα ήδη υπάρχον ID αλλά από διαφορετική διεύθυνση από την καταγεγραμμένη. Τότε το `DISCOVERY_REPLY` του δεύτερου agent απορρίπτεται και του απαντά με ένα μήνυμα `DISCOVERY_AGENT_ID_EXISTS`. Ο agent αφού λάβει το μήνυμα αυτό, επιλέγει ξανά τυχαίο ID και failsafes και απαντά αμέσως με ένα `DISCOVERY_REPLY`, χωρίς να περιμένει τον επόμενο κύκλο Discovery.

4.4.4.2 False Positive

Έστω ότι λειτούργει στο δίκτυο 1 agent με τα παρακάτω στοιχεία (πίνακας 4.11):

Πίνακας 4.11: Παράδειγμα ID collision (False Positive)

Agent ID	Fsafe 1	Fsafe 2	IPv4 address	IPv4 port	IPv6 address	IPv6 port
1234	24	233	10.0.0.1	50000	fe80::dec0:1234	50001

Κατά τη διαδικασία του Discovery ο agent στέλνει 2 `DISCOVERY_REPLY` (IPv4/IPv6) στα οποία το πεδίο Agent ID είναι ίδιο. Ο manager λαμβάνει το IPv4 μήνυμα και αποθηκεύει το νέο agent με ID=1234 με την IPv4 address του. Στη συνέχεια λαμβάνει το IPv6 μήνυμα, παρατηρεί ότι το ID υπάρχει ήδη σε άλλη διεύθυνση, και ξεκινά τη διαδικασία για το collision resolution. Σε αυτό το σημείο, ο

manager πρέπει να διαπιστώσει το αν πρόκειται για εσφαλμένο ID από διαφορετικό agent ή για έγκυρο ID από την εναλλακτική IP address του ίδιου agent. Παρατηρεί ότι τα πεδία Failsafe1 και Failsafe2 που έχει ήδη καταγεγραμμένα από το πρώτο μήνυμα είναι ιδία με αυτά του δευτέρου μηνύματος, οπότε συμπεραίνει ότι πρόκειται για έγκυρο ID.

4.4.4.3 False Negative

Έστω ότι λειτουργούν στο δίκτυο 2 agent με τα παρακάτω στοιχεία (πίνακας 4.12):

Πίνακας 4.12: Παράδειγμα ID collision (False Negative)

Agent ID	Fsafe 1	Fsafe 2	IPv4 address	IPv4 port	IPv6 address	IPv6 port
1234	24	233	10.0.0.1	50000	fe80::dec0:1234	50001
1234	24	233	10.0.0.2	50000	fe80::2ec1:5678	50001

Σε αυτή την περίπτωση η εφαρμογή συμπεριφέρεται διαφορετικά σε κάθε κύκλο Discovery ανάλογα με τη σειρά με την οποία λαμβάνονται τα `DISCOVERY_REPLY` από τον manager. Εφόσον τα πεδία Failsafe έχουν ίδιες τιμές, ο manager δεν μπορεί να ξεχωρίσει τους 2 agents. Η πιθανότητα όμως να συμβεί αυτό είναι πολύ μικρή όπως φαίνεται στη σχέση 4.1.

$$P(false\ neg.) = \frac{1}{64510 \cdot 256 \cdot 256} = \frac{1}{4227727360} \cong 2{,}365 \cdot 10^{-10} \qquad (4.1)$$

4.5 Shutdown

Ο τερματισμός και η επανεκκίνηση επιτυγχάνονται με την αποστολή των μηνυμάτων `EXEC_OP_SHUTDOWN` και `EXEC_OP_RESTART` αντίστοιχα. Στο επόμενο σχήμα φαίνεται η δομή και το περιεχόμενο ενός μηνύματος τερματισμού.

Πίνακας 4.13: EXEC_OP_SHUTDOWN

Version	Message Type	Agent ID	Seq No	Payload Length	Checksum
1	32	1234	5	0	abcd1234…

Ο agent που λαμβάνει αυτό το μήνυμα, τερματίζει τη λειτουργία του εκτελώντας την αντίστοιχη εντολή ανάλογα με το λειτουργικό σύστημα στο οποίο βρίσκεται. Οι εντολές τερματισμού είναι οι εξής:

Για το Windows

```
shutdown -s -f -t 1
```

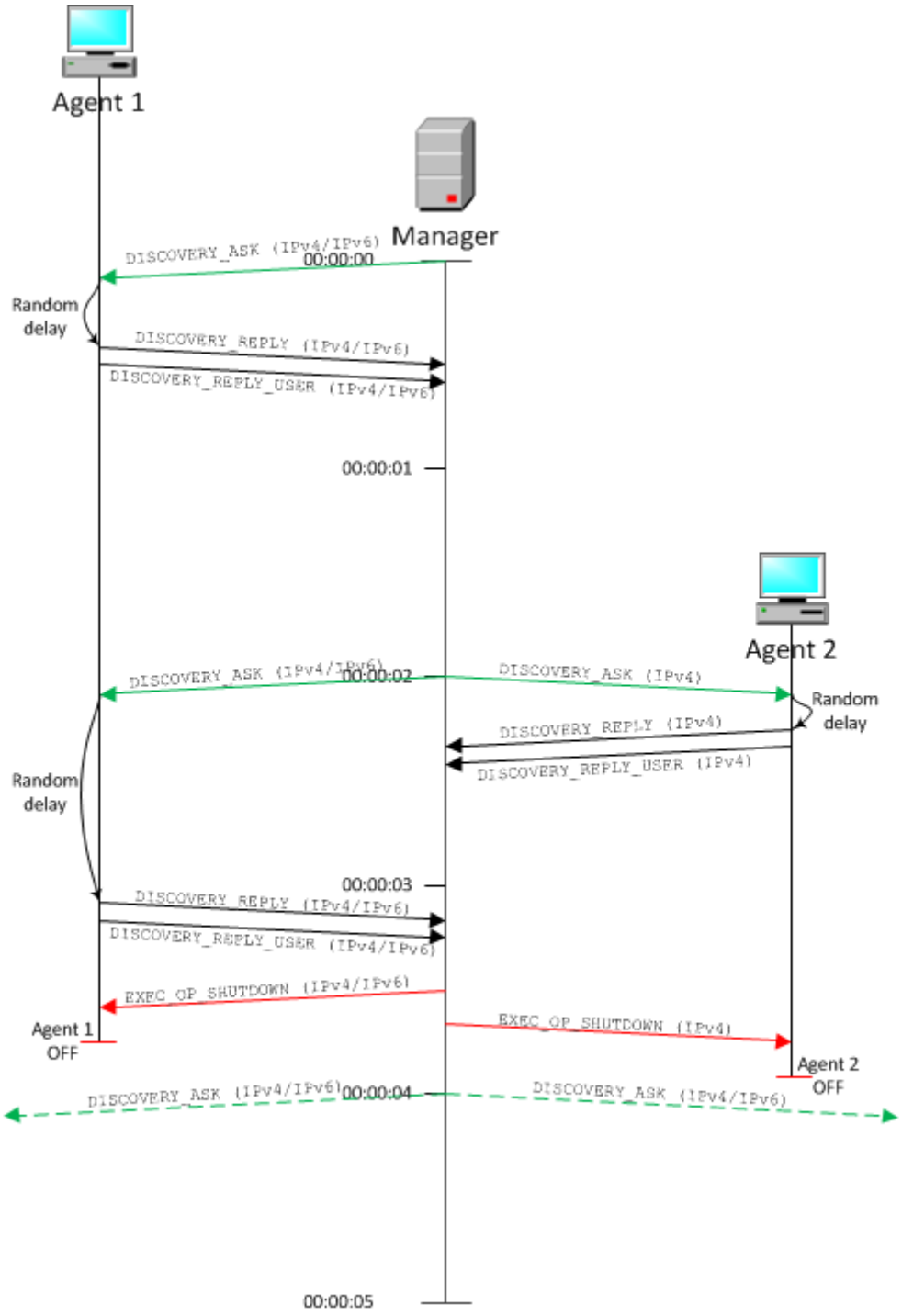
και για το Linux

```
dbus-send --system --print-reply --dest=org.freedesktop.ConsoleKit
/org/freedesktop/ConsoleKit/Manager
org.freedesktop.ConsoleKit.Manager.Stop
```

Για το Linux προτιμήθηκε η χρήση της εντολής `dbus-send` για την αποστολή του σχετικού μηνύματος στο D-Bus αντί της εντολής `shutdown`, διότι η δεύτερη απαιτεί αυξημένα δικαιώματα για την εκτέλεση. Εφόσον η εφαρμογή δεν υποστηρίζει κρυπτογράφηση, θα ήταν ανασφαλής η αποστολή ενός password για την εκτέλεση της εντολής με `sudo`. Επίσης, η τροποποίηση του /etc/sudoers θα προσέθετε επιπλέον πολυπλοκότητα στην εγκατάσταση των agents στους υπολογιστές.

4.6 Τυπικό σενάριο λειτουργίας

Η εικόνα 4.3 δείχνει 5 δευτερόλεπτα από τη λειτουργία ενός συστήματος με 2 agents και έναν manager όταν δεν προκύπτουν προβλήματα.



Εικ. 4.3: Timeline ανταλλαγής μηνυμάτων (Τυπική λειτουργία)

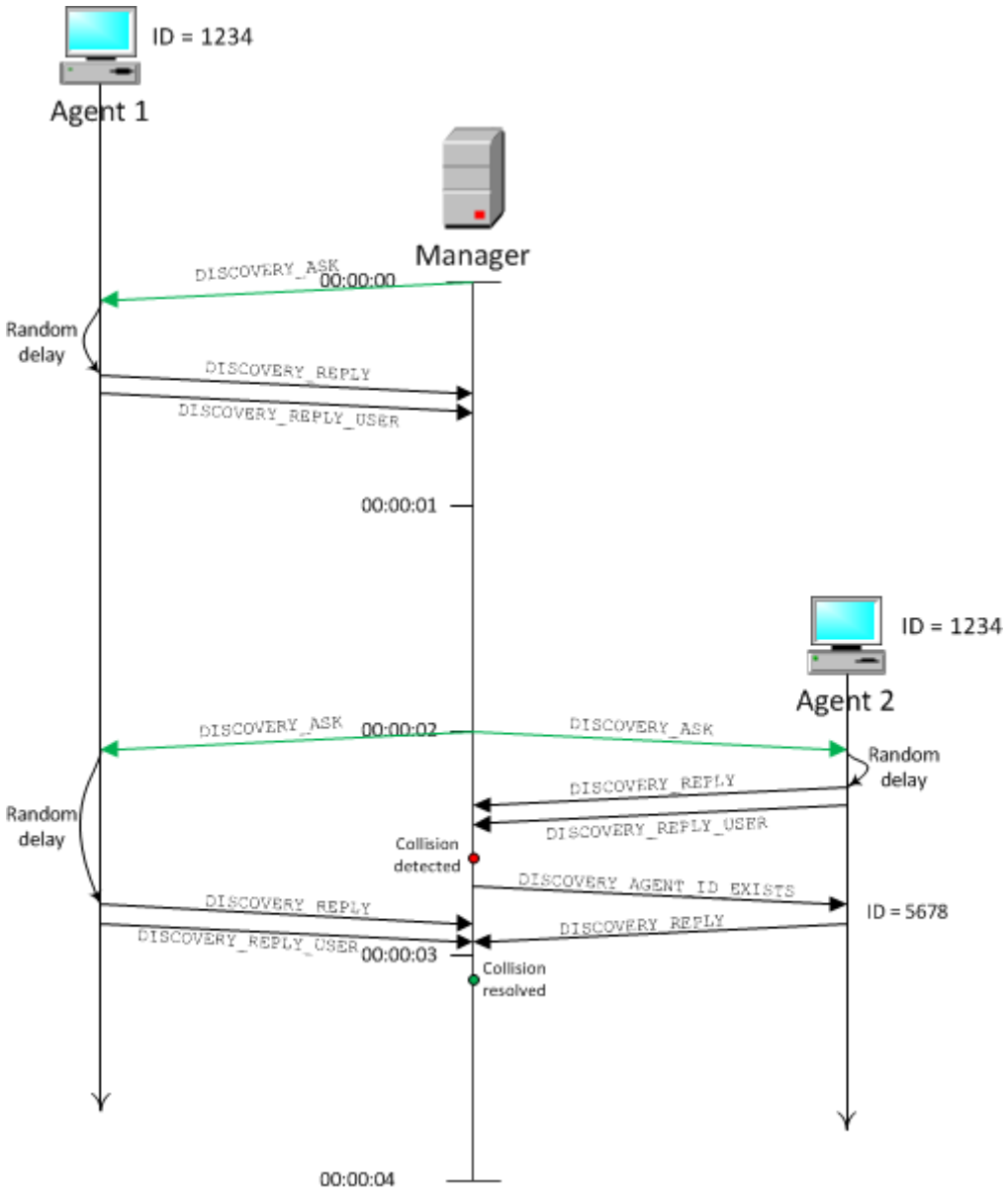
Δεν παίζει ρόλο η σειρά με την οποία πρέπει να τεθούν σε λειτουργία οι agents και ο manager. Στο συγκεκριμένο παράδειγμα, λειτουργεί ήδη ο Agent 1 όταν ανοίγει ο manager με `Interval=2000` και `Inactivity_Counter=1`, και στη συνέχεια τίθεται σε λειτουργία και ένας δεύτερος agent.

Αρχικά ο Agent 1 παρακολουθεί το δίκτυο για μηνύματα που τον αφορούν αλλά εφόσον δεν υπάρχει κάποιος manager, δεν επικοινωνεί με κανέναν. Στη συνέχεια τίθεται σε λειτουργία ο manager και στέλνει multicast το πρώτο `DISCOVERY_ASK` σε IPv4 και IPv6. Ο Agent 1 απαντά και στα 2 εφόσον τα υποστηρίζει. Διαλέγει μια τυχαία καθυστέρηση από 0 έως 1000 ms και στέλνει πρώτα ένα ζεύγος `DISCOVERY_REPLY-DISCOVERY_REPLY_USER` με IPv4 και άλλο ένα με IPv6. Ο manager λαμβάνει αυτά τα μηνύματα και καταγράφει τα στοιχεία του Agent 1. Κάπου μετά το πρώτο δευτερόλεπτο τίθεται σε λειτουργία και ο Agent 2 αλλά ο manager δεν το γνωρίζει ακόμα αυτό, αφού ο δεύτερος «έχασε» τον προηγούμενο κύκλο Discovery. Στον επόμενο κύκλο (t=2s), ο manager μειώνει το timeout του Agent 1 και στέλνει ξανά ένα multicast ζεύγος `DISCOVERY_ASK`. Ο Agent 1 απαντά ξανά με τον ίδιο τρόπο και το timeout του αρχικοποιείται. Ο Agent 2 υποστηρίζει μόνο IPv4 οπότε αφού διαλέξει μια τυχαία καθυστέρηση, απαντά μόνο με ένα ζεύγος `DISCOVERY_REPLY-DISCOVERY_REPLY_USER` με IPv4. Ο manager λαμβάνει αυτά τα μηνύματα και καταγράφει τα στοιχεία του Agent 2. Λίγο πριν τον επόμενο κύκλο Discovery (t=3,5s), ο manager στέλνει μια εντολή τερματισμού στον Agent 1. Αυτό γίνεται με την unicast αποστολή σε αυτόν ενός ζεύγους μηνυμάτων `EXEC_OP_SHUTDOWN` και με τα δυο IP versions. Όταν ο Agent 1 λάβει το πρώτο μήνυμα τερματισμού, εκτελεί την εντολή του manager και τίθεται εκτός λειτουργίας. Τη χρονική στιγμή (t=3,7s) ο manager στέλνει μια εντολή τερματισμού και στον Agent 2, όπως και στην προηγούμενη περίπτωση, αλλά αυτή τη φορά μόνο με IPv4 αφού μόνο αυτό υποστηρίζεται στο συγκεκριμένο agent. Ο Agent 2 λαμβάνει το μήνυμα και τερματίζει επίσης τη λειτουργία του. Οι agents δε στέλνουν επιβεβαιώσεις για τη λήψη αυτών των μηνυμάτων. Αντί αυτών, χρησιμοποιείται ο επόμενος κύκλος Discovery ως θετική η αρνητική επιβεβαίωση.

4.7 Σενάριο λειτουργίας με ID collision

Η εικόνα 4.4 δείχνει 4 δευτερόλεπτα από τη λειτουργία ενός συστήματος με 2 agents και έναν manager, όταν συμβεί ένα Agent ID collision. Όπως και στο

προηγούμενο παράδειγμα, δεν παίζει ρόλο η σειρά με την οποία πρέπει να τεθούν σε λειτουργία οι agents και ο manager.



Εικ. 4.4: Timeline ανταλλαγής μηνυμάτων (ID collision)

Αρχικά ο Agent 1 παρακολουθεί το δίκτυο για μηνύματα που τον αφορούν αλλά εφόσον δεν υπάρχει κάποιος manager, δεν επικοινωνεί με κανέναν. Στη συνέχεια τίθεται σε λειτουργία ο manager και στέλνει multicast το πρώτο `DISCOVERY_ASK` σε. Για την απλοποίηση του παραδείγματος θεωρούμε ότι η

επικοινωνία γίνεται μόνο με ένα IP version. Ο Agent 1 διαλέγει μια τυχαία καθυστέρηση από 0 έως 1000 ms και στέλνει ένα ζεύγος `DISCOVERY_REPLY`-`DISCOVERY_REPLY_USER` με ID=1234. Ο manager λαμβάνει αυτά τα μηνύματα και καταγράφει τα στοιχεία του Agent 1. Κάπου μετά το πρώτο δευτερόλεπτο (t=1,7s) τίθεται σε λειτουργία και ο Agent 2 αλλά ο manager δεν το γνωρίζει ακόμα αυτό, αφού ο δεύτερος «έχασε» τον προηγούμενο κύκλο Discovery. Στον επόμενο κύκλο (t=2s), ο manager στέλνει ξανά ένα multicast `DISCOVERY_ASK`. Ο Agent 1 απαντά ξανά με το καταγεγραμμένο του ID. Στη συνέχεια απαντά και ο Agent 2, και σε αυτό το σημείο ο manager ανιχνεύει ID collision, εφόσον ο δεύτερος απάντησε με ένα ήδη υπάρχον ID αλλά από διαφορετική διεύθυνση από την καταγεγραμμένη. Ο manager ελέγχει τα πεδία Failsafe για να διαπιστώσει αν πρόκειται για εσφαλμένο ID από διαφορετικό agent ή για έγκυρο ID από την εναλλακτική IP address του ίδιου agent. Τα πεδία δεν ταιριάζουν άρα είναι πραγματικό collision. Τότε το `DISCOVERY_REPLY` του Agent 2 απορρίπτεται και του απαντά με ένα μήνυμα `DISCOVERY_AGENT_ID_EXISTS`. Ο agent αφού λάβει το μήνυμα αυτό, επιλέγει ξανά τυχαίο ID (5678) και failsafes και απαντά αμέσως με ένα `DISCOVERY_REPLY`, χωρίς να περιμένει τον επόμενο κύκλο Discovery. Τέλος, ο manager λαμβάνει τη νέα απάντηση και η σύγκρουση επιλύεται.

Η δομή και το περιεχόμενο των μηνυμάτων που ανταλλάσει ο Manager με τον Agent 2 φαίνονται παρακάτω.

Πίνακας 4.14: DISCOVERY_ASK

Version	Message Type	Agent ID	Seq No	Payload Length	Checksum
1	21	0	2	0	1234abcd…

Πίνακας 4.15: DISCOVERY_REPLY

Version	Message Type	Agent ID	Seq No	Payload Length	Payload	Checksum
1	22	1234	1	6	101124233	abcd1234…

Πίνακας 4.16: DISCOVERY_REPLY_USER

Version	Message Type	Agent ID	Seq No	Payload Length	Payload	Checksum
1	25	1234	2	5	User2	3456cdef…

Πίνακας 4.17: DISCOVERY_AGENT_ID_EXISTS

Version	Message Type	Agent ID	Seq No	Payload Length	Checksum
1	24	0	3	0	1a2b3c4d…

Πίνακας 4.18: DISCOVERY_REPLY

Version	Message Type	Agent ID	Seq No	Payload Length	Payload	Checksum
1	22	5678	3	6	101133222	abcd1234…

5. Προτάσεις – Συμπεράσματα

Η εφαρμογή που υλοποιήθηκε στα πλαίσια της πτυχιακής εργασίας πετυχαίνει τους στόχους που τεθήκαν. Μπορεί να λειτουργήσει αξιόπιστα σε ένα IP δίκτυο όπου οι διευθύνσεις ανατίθενται δυναμικά μέσω DHCP, υποστηρίζει παράλληλη λειτουργία σε IPv4 και IPv6, και είναι cross-platform. Χρησιμοποιώντας UDP μηνύματα για την επικοινωνία, η επιβάρυνση που προσθέτει στην κίνηση του δικτύου είναι χαμηλή. Οι παράμετροι λειτουργιάς της εφαρμογής μπορούν να τροποποιηθούν εύκολα μέσω του αντίστοιχου User Interface, ενώ μέσω HMAC υλοποιείται authentication μηνυμάτων χαμηλής μεν ποιότητας (λόγω md5), κρατώντας όμως το συνολικό μήκος των μηνυμάτων σε χαμηλά επίπεδα.

Ένα σημαντικό πρόβλημα που αντιμετωπίστηκε κατά την υλοποίηση της εφαρμογής ήταν η cross-platform υποστήριξη του πρωτοκόλλου IPv6. Αυτό το πρόβλημα έχει τις ρίζες του στη διαφορετική υλοποίηση των βιβλιοθηκών στα λειτουργικά συστήματα Windows και Linux, όπως φαίνεται στο παράδειγμα που ακολουθεί.

```
athspk@LubuntuVBox:~$ python
Python 3.2.2 (default, Sep  5 2011, 21:17:14) [GCC 4.6.1] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import socket
>>> socket.IPPROTO_IPV6
41
>>>
>>> socket.inet_pton(socket.AF_INET6, "fe80::abcd:1234")
b'\xfe\x80\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\xab\xcd\x124'


C:\>python
Python 3.2.2 (default, Sep  4 2011, 09:07:29) [MSC v.1500 64 bit (AMD64)]
on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import socket
>>> socket.IPPROTO_IPV6
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'module' object has no attribute 'IPPROTO_IPV6'
>>>
>>> socket.inet_pton(socket.AF_INET6, "fe80::abcd:1234")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'module' object has no attribute 'inet_pton'
```

Η σταθερά `socket.IPPROTO_IPV6` έχει τιμή 41 στο Linux, ενώ στο Windows δεν ορίζεται, παρομοίως η συνάρτηση `socket.inet_pton(address_family, ip_string)`, στο Linux μετατρέπει το string μιας διεύθυνσης σε bytes ενώ στο Windows δεν υλοποιείται. Για την αντιμετώπιση αυτού του προβλήματος και την παραμετροποίηση των IPv6 sockets, η σταθερά `socket.IPPROTO_IPV6` αντικαταστάθηκε με την hardcoded τιμή 41, ενώ η συνάρτηση `inet_pton` υλοποιήθηκε στο module ipParser.py.

Όσον αφορά τη πιθανή μελλοντική βελτίωση της εφαρμογής, η ανακάλυψη του PATH MTU θα μπορούσε να αποτελέσει σημαντικό μέρος της εφαρμογής. Στο πεδίο Message Type του πρωτοκόλλου της εφαρμογής η δεκάδα 11-20 έχει δεσμευτεί για αυτό το σκοπό. Το IP fragmentation δεν αποτελεί πρόβλημα για το πρωτόκολλο TCP. Στο UDP όμως που δεν υλοποιούνται μηχανισμοί αξιοπιστίας, το fragmentation μπορεί να δημιουργήσει προβλήματα κατά τη μετάδοση μεγάλου μεγέθους datagrams. Μια μέθοδος για την εύρεση του PATH MTU είναι η αποστολή IP πακέτων με τον ορισμό Don't Fragment στο πεδίο IP Flags και η λήψη απαντήσεων ICMP Fragmentation Needed. Αυτή η μέθοδος όμως δεν καλύπτει την περίπτωση όπου οι routers μπλοκάρουν τις απαντήσεις του ICMP.

Τα μηνύματα που ανταλλάσει η εφαρμογή έχουν μικρό μέγεθος και δεν επηρεάζονται από το fragmentation. Όπως ορίζει το RFC 791, η μικρότερη τιμή που μπορεί να πάρει το MTU είναι 68 bytes, ενώ το τυπικό μέγεθος ενός IP Header είναι 20 bytes. (Postel, 1981, p. 13,25). Όσον αφορά το πρωτόκολλο UDP όπως ορίζεται στο RFC 768, το μέγεθος ενός Header είναι 8 bytes. (Postel, 1980, p. 1). Με βάση τα παραπάνω, το μέγιστο μέγεθος που μιας Application Layer PDU που μπορεί να περάσει από το interface ενός router χωρίς fragmentation είναι 40 bytes (Σχέση 5.1).

$$\begin{aligned} APDU\_Length\_NoFrag_{max} &= \\ &= MTU_{min} - IP\_Header\_Length - UDP\_Header\_Length \\ &= 68 - 20 - 8 = 40 \end{aligned} \qquad (5.1)$$

Η εφαρμογή δεν λαμβάνει υπ' όψη την περίπτωση του IP fragmentation και δεν υλοποιεί κάποιο μηχανισμό για την αντιμετώπισή του, διότι το μέγιστο μέγεθος μιας τυπικής Application Layer PDU (APDU) δεν ξεπερνά το όριο των 40 bytes που αναφέρθηκε προηγουμένως. Το ελάχιστο μέγεθος μιας APDU είναι 26 bytes όταν δεν υπάρχει Payload. Στην περίπτωση του μηνύματος `DISCOVERY_REPLY` το Payload είναι 6 bytes, αυξάνοντας το μέγεθος της APDU στα 26+6=32 bytes. Η μόνη περίπτωση όπου μπορεί να ξεπεραστεί το όριο των 40 bytes είναι κατά την μετάδοση του μηνύματος `DISCOVERY_REPLY_USER`, όπου το μέγεθος του Payload είναι ίσο με το πλήθος των χαρακτήρων του username που μεταδίδεται. Σε αυτή την περίπτωση, το μέγιστο πλήθος χαρακτήρων που μπορεί να έχει ένα username ώστε να μεταδοθεί χωρίς να ξεπεραστεί το όριο των 40 bytes είναι 40-26=14 χαρακτήρες. Το μέγιστο μήκος ενός username είναι 20 χαρακτήρες στο Windows και 32 στο Linux. Αυτό σημαίνει ότι στη χειρότερη περίπτωση (Linux), χρησιμοποιώντας το μέγιστο μέγεθος Payload (32 bytes), το όριο θα ξεπεραστεί κατά 32-14=18 bytes. Με βάση τα παραπάνω συμπεραίνεται ότι όταν το μήκος των μηνυμάτων της εφαρμογής έχει τη μεγίστη τιμή 58 bytes (Σχέση 5.2), τότε η ελάχιστη τιμή MTU που απαιτείται για την ομαλή λειτουργία της εφαρμογής είναι 86 bytes (Σχέση 5.3).

$$APDU\_Length_{max} = APDU\_Length_{min} + Payload_{max} = 26 + 32 = 58 \qquad (5.2)$$

$$MTU_{req.} = IP\_Header\_Length + UDP\_Header\_Length + APDU\_Length_{max} = 20 + 8 + 58 = 86 \qquad (5.3)$$

6. Δομή Κώδικα

Σε αυτό το κεφάλαιο περιγράφεται η δομή του κώδικα της εφαρμογής όσον αφορά τα modules από τα όποια αποτελείται και τις βασικότερες συναρτήσεις τους. Τα modules είναι αρχεία κειμένου που έχουν κατάληξη .py και περιέχουν πηγαίο κώδικα. Τα modules είναι ίδια στον manager και στον agent, εκτός από ορισμένες διαφοροποιήσεις οι οποίες περιγράφονται παρακάτω.

6.1 appLogger.py

Αυτό το Module χειρίζεται τον τρόπο καταγραφής συμβάντων (Logging) της εφαρμογής. Ανάλογα με τη φάση της εφαρμογής (debug,release), η καταγραφή ενός συμβάντος μπορεί να γίνει στο μόνο στο log file κατά τη φάση του release, η παράλληλα και την οθόνη του υπολογιστή κατά τη φάση ανάπτυξης ή αποσφαλμάτωσης (debug).

6.2 configurationParser.py

Αυτό το Module χειρίζεται τη ανάγνωση και συντακτική ανάλυση (parsing) των παραμέτρων του αρχείου ρυθμίσεων. Η διαφορά αυτού του module μεταξύ manager και agent είναι ότι στον agent δεν ορίζονται οι παράμετροι του section `DISCOVERY`.

`doParse(settingsFile):`

Διαβάζει το αρχείο ρυθμίσεων χρησιμοποιώντας τον configparser της Python και χρησιμοποιεί το dictionary με τις default τιμές για να ελεγχθεί η εγκυρότητα του τύπου της κάθε παραμέτρου. Εάν μια μεταβλητή είναι διαφορετικού τύπου από την default, τότε χρησιμοποιείται η default τιμή. Εάν για

παράδειγμα δοθεί εσφαλμένα ένα string ενώ αναμένεται ένας ακέραιος, τότε χρησιμοποιείται η default τιμή, όπως φαίνεται παρακάτω.

```
defaults["LOGGING"]["Backup_files"]=5
parser["LOGGING"]["Backup_files"]="hello" (INVALID TYPE)
newSettings["LOGGING"]["Backup_files"]=5.
```

`applyRulesToSettings():`

Διαβάζει το dictionary `newSettings` και ελέγχει αν οι τιμές των ορισμάτων υπακούουν σε ορισμένους κανόνες εγκυρότητας. Εάν η τιμή ενός ορίσματος παραβιάζει κάποιο κανόνα τότε χρησιμοποιείται η default τιμή για το συγκεκριμένο όρισμα, όπως φαίνεται παρακάτω.

```
newSettings["LOGGING"]["Backup_files"] = -1 (INVALID VALUE)
newSettings["LOGGING"]["Backup_files"] = 5  (default used).
```

`getSettingsFromFile(settingsFile):`

Είναι μια wrapper συνάρτηση που καλεί τις δυο προηγούμενες ώστε να επιστρέψει ένα dictionary που περιέχει έγκυρα ορίσματα.

6.3 discovery.py

Αυτό το Module υλοποιεί το πρωτόκολλο επιπέδου Εφαρμογής, δηλαδή τον τρόπο επικοινωνίας μεταξύ manager – agents.

6.3.1 Manager

Στον manager ορίζεται μια δομή δεδομένων τύπου dictionary που ονομάζεται `agents` και περιέχει τα στοιχεία των agents που ανακαλύπτονται κατά τη διαδικασία του Discovery. Ως κλειδί του agents ορίζεται το εκάστοτε agentID και η τιμή αυτού του κλειδιού είναι ένα άλλο dictionary που περιέχει τα στοιχεία ενός

agent. Στον πίνακα 6.1 φαίνεται ένα παράδειγμα περιεχομένων του `agents`, όπου κατά το Discovery έχουν ανακαλυφθεί 2 διαφορετικοί agents.

Πίνακας 6.1: Παράδειγμα περιεχομένων του dictionary agents.

Key	Value	
1234	Key	Value
	osCode	1
	latestSeqNumber	16
	version	“1.0.1”
	timeOut	1
	peerSockV4	('192.168.56.231', 50000)
	peerSockV6	('abcd::1010',50001)
	failSafe1	12
	failSafe2	213
5678	Key	Value
	osCode	2
	latestSeqNumber	27
	version	“1.0.1”
	timeOut	1
	peerSockV4	('192.168.56.232', 50000)
	peerSockV6	('fec0::1234',50001)
	failSafe1	121
	failSafe2	166

Στη συνέχεια περιγράφονται οι βασικότερες συναρτήσεις που περιέχει αυτό το module.

- `__init__(self):`

Αρχικοποιεί τον πίνακα των agents και εκκινεί τη διαδικασία του Discovery.

- `addAgent(self, agentID, timeStamp, peerSock, version, osCode, failSafe1, failSafe2):`

Προσθέτει ένα νέο agent στο dictionary με τις τιμές των παραμέτρων αυτής της συνάρτησης.

- `updateAgentPeerSock(self, agentID, peerSock):`

Προσθέτει το εναλλακτικό socket για έναν ήδη υπάρχων agent. Αν για παράδειγμα υπάρχει ήδη ένας agent με καταγεγραμμένο IPv4 socket, αυτή η συνάρτηση ενημερώνει το dictionary με το IPv6 socket του ίδιου agent.

- `updateAgentIncrementTimeout(self, timeStamp, agentID):`

Ενημερώνει το πεδίο `latestSeqNumber` στο dictionary για ένα συγκεκριμένο agent.

- `decrementAllTimeouts(self):`

Μειώνει την τιμή του πεδίου `timeOut` κατά 1 όλων των agents. Αν η τιμή μετά τη μείωση είναι μικρότερη του μηδενός, τότε ο συγκεκριμένος agent θεωρείται ανενεργός και προστίθεται σε μια λίστα.

- `removeInactiveAgents(self):`

Διαγράφει από το dictionary `agents` αυτούς οι οποίοι προστεθήκαν στη λίστα των ανενεργών κατά την εκτέλεση της προηγούμενης συνάρτησης.

- `resolveIDcollisions(self, agentID, timeStamp, peerSock, osCode, failSafe1, failSafe2):`

Αναλαμβάνει την επίλυση πιθανών συγκρούσεων όσον αφορά το agentID.

- `handle_discovery_reply(self, agentID, timeStamp, payloadLength, payload, peerSock):`

Χειρίζεται τα εισερχόμενα μηνύματα DISCOVERY_REPLY.

- `handle_discovery_reply_user(self, agentID, timeStamp, payloadLength, payload, peerSock):`

Χειρίζεται τα εισερχόμενα μηνύματα DISCOVERY_REPLY_USER.

- `send_msg_discovery_ask(self):`

Συνθέτει ένα μήνυμα DISCOVERY_ASK και το προωθεί για αποστολή.

- `send_msg_discovery_agent_id_exists(self, peerSock):`

Συνθέτει ένα μήνυμα DISCOVERY_AGENT_ID_EXISTS και το προωθεί για αποστολή.

- `send_msg_exec_op_restart(self, agentID, peerSock):`

Συνθέτει ένα μήνυμα EXEC_OP_RESTART και το προωθεί για αποστολή.

- `send_msg_exec_op_shutdown(self, agentID, peerSock):`

Συνθέτει ένα μήνυμα DISCOVERY_ASK και το προωθεί για αποστολή.

- `triggerDiscovery(self):`

Είναι ένα Slot που εκτελείται σε συγκεκριμένα χρονικά διαστήματα και σηματοδοτεί την έναρξη κάθε κύκλου Discovery.

- `packAPDU(self, msgType, agentID, payloadList=[]):`

Επιστρέφει τη δυαδική μορφή ενός μηνύματος ώστε να είναι έτοιμο προς αποστολή.

- `unpackAPDU(self, aPDU, peerSock):`

Εξάγει πληροφορίες από ένα εισερχόμενο μήνυμα σε δυαδική μορφή και το προωθεί εσωτερικά για περεταίρω επεξεργασία.

- `callByType(self, msgType, agentID, timeStamp, payloadLength, payload, peerSock):`

Διαβάζει την τιμή του πεδίου MSG_TYPE και προωθεί το μήνυμα στην κατάλληλη συνάρτηση.

- `handleRead(self, aPDU, peerSock):`

Χειρίζεται την ανάγνωση των εισερχόμενων μηνυμάτων.

- `handleSend(self):`

Χειρίζεται την αποστολή μηνυμάτων.

6.3.2 Agent

Το ίδιο module στον Agent έχει αρκετές συναρτήσεις που υλοποιούνται με τον ίδιο τρόπο όπως και στο Manager. Οι συναρτήσεις που είναι διαφορετικές περιγράφονται παρακάτω.

- `__init__(self):`

Αρχικοποιεί τη μεταβλητή `agentID` με ένα τυχαίο ακέραιο στο διάστημα [1025,65535], και τις μεταβλητές `failSafe1` και `failSafe2` με ένα τυχαίο ακέραιο στο διάστημα [0,255].

- `connectTimerIdToFunc(self, id):`

Συνδέει το timeout Signal ενός συγκεκριμένου QTimer με μια Slot συνάρτηση.

- `timed_send_msg_discovery_reply(self, id):`

Είναι μια Slot συνάρτηση που καλείται όταν πυροδοτηθεί το timeOut signal ενός συγκεκριμένου QTimer που αντιπροσωπεύει το Random Delay πριν την αποστολή ενός μηνύματος DISCOVERY_REPLY.

- `timed_send_msg_discovery_reply_user(self, id):`

Ομοίως, για την αποστολή ενός μηνύματος DISCOVERY_REPLY_USER.

- `getCurrentUser(self):`

Επιστρέφει το username του χρήστη που εκτέλεσε την εφαρμογή του Agent.

- `handle_discovery_ask(self, agentID, timeStamp, payloadLength, payload, peerSock):`

Χειρίζεται τα εισερχόμενα μηνύματα DISCOVERY_ASK.

- `handle_discovery_agent_id_exists(self, agentID, timeStamp, payloadLength, payload, peerSock):`

Χειρίζεται τα εισερχόμενα μηνύματα DISCOVERY_AGENT_ID_EXISTS.

- `handle_msg_exec_op_restart(self, agentID, timeStamp, payloadLength, payload, peerSock):`

Εκτελεί επανεκκίνηση του συστήματος.

- `handle_msg_exec_op_shutdown(self, agentID, timeStamp, payloadLength, payload, peerSock):`

Τερματίζει τη λειτουργία του συστήματος

- `send_msg_discovery_reply(self, peerSock):`

Συνθέτει ένα μήνυμα DISCOVERY_REPLY και το προωθεί για αποστολή.

- `send_msg_discovery_reply_user(self, peerSock):`

Συνθέτει ένα μήνυμα DISCOVERY_REPLY_USER και το προωθεί για αποστολή.

6.4 globalConst.py

Αυτό το module περιέχει σταθερές που ρυθμίζουν τη λειτουργία του πρωτοκόλλου αλλά και της εφαρμογής γενικότερα. Οι σημαντικότερες σταθερές περιγράφονται παρακάτω.

- `LOG_PATH = "logs/"`

Το path του φακέλου στον οποίο αποθηκεύονται τα log files. Ο default φάκελος είναι ο υποφάκελος logs από όπου εκτελείται η εφαρμογή.

- `MAX_BYTES = 512*1024`

Το μέγιστο μέγεθος ενός log file σε bytes. Όταν ένα log file φτάσει το μέγιστο μέγεθος, τότε το αρχείο κλείνει, μετονομάζεται σε logfile.txt.n, όπου n ένας αύξων αριθμός από 1 έως `BACKUP_COUNT` (βλ. παρακάτω), και η καταγραφή συνεχίζεται σε νέο αρχείο. Το default μέγεθος είναι 512 KiB.

- `BACKUP_COUNT = 2`

Καθορίζει το μέγιστο πλήθος εφεδρικών Log files. Το default είναι 2 αρχεία.

- `DEFAULT_CONF_FILE = "settings.ini"`

Το path του αρχείου ρυθμίσεων. Το default είναι το αρχείο settings.ini στο ίδιο path με το εκτελέσιμο αρχείο της εφαρμογής.

- `MD5_KEY = "Anything_above_16_characters"`

Το κλειδί που θα εισάγει ο HMAC στη συνάρτηση MD5 για την παραγωγή του hash, το οποίο, συμφώνα με το RFC 2104 θα πρέπει να έχει μήκος τουλάχιστον 16 bytes (Krawczyk, Bellare, & Canetti, 1997, p. 4).

- `MD5_LENGTH = 16`

Το μήκος του hash output (16 bytes) της συνάρτησης MD5.

- `AGENT_MIN_ID = 1025, AGENT_MAX_ID = 65535`

Το ελάχιστο(1024) και μέγιστο(65535) ID ενός agent παρέχει τη δυνατότητα ταυτόχρονης λειτουργίας 64510 agents.

Οι παρακάτω σταθερές αφορούν γενικές ρυθμίσεις του Multicast, όπως η διεύθυνση, το Port number και το TTL (Hop Limit για το IPv6):

- `MCAST_GRP = '239.6.6.6'`
- `MCAST_PORT = 50000`
- `MCAST_TTL = 32`
- `MCAST_GRP_V6 = 'ff15:7079:7468:6f6e:6465:6d6f:6d63:6173'`
- `MCAST_PORT_V6 = 50001`
- `MCAST_TTL_V6 = 32`

Οι παρακάτω σταθερές αφορούν τα μήκη των πεδίων του πρωτοκόλλου σε bytes (βλ. Κεφ. 3.1: Πεδία πρωτοκόλλου):

- `LENGTH_VERSION = 1`

- `LENGTH_MSG_TYPE = 1`
- `LENGTH_AGENT_ID = 2`
- `LENGTH_SEQ_NUM = 4`
- `LENGTH_PAYLOAD_LENGTH = 2`
- `LENGTH_CHECKSUM = MD5_LENGTH`

Οι παρακάτω σταθερές αφορούν την τιμή του πεδίου MSG_TYPE για κάθε ένα τύπο μηνύματος που μεταδίδεται από την εφαρμογή (βλ. Κεφ. 3.2: Τύποι μηνυμάτων):

- `DISCOVERY_ASK = 21`
- `DISCOVERY_REPLY = 22`
- `DISCOVERY_AGENT_ID_EXISTS = 23`
- `DISCOVERY_REPLY_USER = 25`
- `EXEC_OP_RESTART = 31`
- `EXEC_OP_SHUTDOWN = 32`

- `VERSION = 1`

Καθορίζει την έκδοση του πρωτοκόλλου στο αντίστοιχο πεδίο. Στην παρούσα φάση της ανάπτυξης της εφαρμογής το πρωτόκολλο βρίσκεται στην έκδοση 1.

- `INTERVAL = 5000`

Καθορίζει την περίοδο εκπομπής των multicast μηνυμάτων `DISCOVERY_ASK` σε milliseconds (βλ. Κεφ. 4.4.1: Interval). Αυτό το μήνυμα εκπέμπεται κάθε 5 δευτερόλεπτα by default.

- `INACTIVITY_COUNTER = 2`

Καθορίζει το πλήθος των φορών που επιτρέπεται σε έναν Agent να μην απαντήσει σε ένα `DISCOVERY_ASK`, πριν θεωρηθεί ανενεργός (βλ. Κεφ. 4.4.2: Timeout). Η default τιμή είναι 2 φορές.

- `MAX_ARTIFICIAL_DELAY = 1000`

Καθορίζει τη μεγίστη τεχνητή καθυστέρηση σε milliseconds πριν την απάντηση ενός agent σε ένα `DISCOVERY_ASK` (βλ. Κεφ. 4.4.3: Random Delay). Η default τιμή είναι 1000 milliseconds.

Οι παρακάτω σταθερές αφορούν τον τύπο του λειτουργικού συστήματος στο οποίο εκτελείται ο agent.

- `OS_CODE_WIN = 1`
- `OS_CODE_LINUX = 2`
- `OS_CODE_MAC = 3`

6.5 img_rc.py

Αυτό το module παράγεται από το αντίστοιχο .qrc αρχείο (Resource file) του QtDesigner με την εκτέλεση της παρακάτω εντολής, και περιέχει σε δυαδική μορφή τα εικονίδια που χρησιμοποιούνται στην εφαρμογή.

```
$ pyrcc4 -py3 -o img_rc.py img.qrc
```

6.6 ipParser.py

Αυτό το module περιέχει 16 συναρτήσεις που αφορούν τον έλεγχο εγκυρότητας μιας IP διεύθυνσης, καθώς και τη μετατροπή της σε δυαδική μορφή. Περιέχει επίσης τη σταθερά `socket.IPPROTO_IPV6=41` η οποία δεν ορίζεται στο λειτουργικό σύστημα Windows. Οι σημαντικότερες από αυτές τις συναρτήσεις περιγράφονται παρακάτω.

- `isValid_v6(inputAddress):`

Ελέγχει την εγκυρότητα μιας IPv6 διεύθυνσης.

- `isValid_v4(inputAddress):`

Ομοίως για τις IPv4 διευθύνσεις.

- `getExpandedAddressString(inputAddress):`

Επιστρέφει την ανεπτυγμένη μορφή μιας IPv6 διεύθυνσης. Για παράδειγμα, η συνεπτυγμένη διεύθυνση `fec0::21:abcd` επιστρέφεται ως `fec0:0000:0000:0000:0000:0000:0021:abcd`.

- `getBytesFromAddressV6(inputAddress):`

Επιστρέφει ένα αντικείμενο τύπου bytes που περιέχει τη δυαδική μορφή μιας δοθείσας IPv6 διεύθυνσης.

- `getBytesFromAddressV4(inputAddress):`

Ομοίως για τις IPv4 διευθύνσεις.

- `inet_pton(addressFamily, address):`

Επιστρέφει ένα αντικείμενο τύπου bytes που περιέχει τη δυαδική μορφή μιας δοθείσας διεύθυνσης (address) ενός πρωτοκόλλου (addressFamily). Η συνάρτηση υποστηρίζει μόνο τα Address Families `socket.AF_INET` του πρωτοκόλλου IPv4 και `socket.AF_INET6` του IPv6.

6.7 netUtils.py

Αυτό το module περιέχει βοηθητικές συναρτήσεις γενικού σκοπού οι οποίες χρησιμοποιούνται από διάφορα άλλα μέρη της εφαρμογής. Οι σημαντικότερες από αυτές περιγράφονται παρακάτω.

- `getCurrentOS():`

Ανιχνεύει τον τύπο του λειτουργικού συστήματος στο οποίο εκτελείται η εφαρμογή και επιστρέφει το όνομά του.

- `getCurrentOSCode(currentOS):`

Επιστρέφει έναν ακέραιο αριθμό από 1 έως 3 που αντιστοιχεί στον τύπο του λειτουργικού συστήματος της παραμέτρου currentOS. (Βλ. Κεφ. 6.4: globalConst.py)

- `intToList(intInput):`

Επιστρέφει μια λίστα που περιέχει τα ψηφία ενός ακεραίου αριθμού (intInput).

- `BytesToListOfInt(bytesInput):`

Μετατρέπει ένα αντικείμενο bytes σε μια λίστα ακεραίων αριθμών.

- `getMessageDigestWithFixedKey(msg):`

Επιστρέφει το hmac-md5 hash ενός μηνύματος (msg) που δίνεται σε δυαδική μορφή, χρησιμοποιώντας ένα συγκεκριμένο κλειδί.

6.8 rsManager.py, rsAgent.py

Αυτά τα modules είναι το σημείο έναρξης της εφαρμογής για τον Manager και τον Agent αντίστοιχα, και αναλαμβάνει την έναρξη του main event loop της QT. Η μόνη συνάρτηση που ορίζεται σε αυτά τα modules είναι η `cleanUp`() η οποία είναι ένα Slot που καλείται από την πυροδότηση του Signal `aboutToQuit` όταν πρόκειται να τερματίσει η εφαρμογή.

- `cleanUp():`

Τερματίζει το thread των UDP servers καθώς επίσης και το logging όταν η εφαρμογή πρόκειται να κλείσει.

6.9 runUI*.py

Όλα τα modules της μορφής `runUI*.py` αποτελούν wrappers για τα αντίστοιχα `UI*.py` modules που έχουν παραχθεί από το εργαλείο `pyuic4` και αφορούν τη λειτουργία του γραφικού περιβάλλοντος της εφαρμογής. Για παραδειγμα το molude `runUIAbout.py` είναι wrapper για το `UIAbout.py`.

Όσον αφορά το γραφικό περιβάλλον, τα σημαντικότερα Modules είναι το `runUIMain.py` που αναλαμβάνει την απεικόνιση του κεντρικού παραθύρου του manager, και το `runUIAgent.py` αντιστοίχως για τον agent. Στη συνέχεια περιγράφονται οι σημαντικότερες συναρτήσεις αυτών των Modules.

- runUIMain.py

Αυτό το module περιέχει την κλάση `WindowUIMain` στην οποία ορίζονται 5 Signals τα οποία συνδέονται με τις αντίστοιχες Slot συναρτήσεις όπως περιγράφεται στη συνέχεια.

```
class WindowUIMain(QMainWindow):
    resizeTrigger = pyqtSignal()
    updatePeerSock = pyqtSignal(str,str,str)
    updateUserName = pyqtSignal(str,str)
    addAgent = pyqtSignal(str,str,str,str,str,str,str)
    removeAgent = pyqtSignal(str)
```

`__init__(self):`

Αρχικοποιεί το αντικείμενο της κλάσης καλώντας πρώτα τον constructor της υπερκλάσης `QMainWindow` και στη συνέχεια συνδέει τα Signals με τα Slots.

`gatherSelectedAgentIDs(self):`

Επιστρέφει μια λίστα που περιέχει τα IDs των agents που έχει επιλέξει ο χρήστης στο GUI.

`triggerRestart(self):`

Εκκινεί τη διαδικασία επανεκκίνησης για τους επιλεγμένους agents των οποίων τα IDs συλλέγονται από την παραπάνω συνάρτηση.

`triggerShutdown(self):`

Ομοίως για τη διαδικασία του shutdown.

`resizeTableRowsCols(self):`

Είναι ένα Slot το οποίο εκτελείται από το Signal `resizeTrigger` κάθε φόρα που προστίθεται ένας agent, και ορίζει το πλάτος των γραμμών και στηλών του πίνακα των agents ανάλογα με τα περιεχόμενα των κελιών.

`updateAgentUsername(self, agentID, username):`

Είναι ένα Slot το οποίο εκτελείται από την πυροδότηση του Signal `updateUserName` όταν γνωστοποιείται στο Manager το username ενός agent, και ενημερώνει το GUI με αυτό το Username.

`updateAgentPeerSock(self, agentID, address, port):`

Είναι ένα Slot το οποίο εκτελείται από την πυροδότηση του Signal `updatePeerSock` όταν ο Manager καταγράφει την εναλλακτική διεύθυνση και port number για έναν ήδη γνωστό agent, και ενημερώνει το GUI κατάλληλα.

`addAgent(self, agentID, addressV4, portV4, addressV6, portV6, currentOS, version):`

Είναι ένα Slot το οποίο εκτελείται από την πυροδότηση του Signal `addAgent` όταν ο Manager καταγράφει ένα νέο agent, και τον προσθέτει στο GUI στον πίνακα των agents.

`removeAgent(self, agentID):`

Είναι ένα Slot το οποίο εκτελείται από την πυροδότηση του Signal `removeAgent` όταν ο Manager διαγράφει έναν agent, και ενημερώνει το GUI κατάλληλα.

- runUIAgent.py

Αυτό το module περιέχει την κλάση `DialogUIAgent` στην οποία ορίζονται 3 Signals τα οποία συνδέονται με τις αντίστοιχες Slot συναρτήσεις όπως περιγράφεται στη συνέχεια.

```
class DialogUIAgent(QDialog):
    timerStart = pyqtSignal(int)
    createTimer = pyqtSignal(int,int,int,tuple)
    removeTimer = pyqtSignal(int)
```

`__init__(self):`

Αρχικοποιεί το αντικείμενο της κλάσης καλώντας πρώτα τον constructor της υπερκλάσης `QDialog` και στη συνέχεια συνδέει τα Signals με τα Slots. Επίσης αρχικοποιεί ένα dictionary με ονομα `timers` στο οποίο θα αποθηκεύονται αντικείμενα QTimer.

`startMyTimer(self, id):`

Εκκινεί έναν QTimer με συγκεκριμένο id που είναι αποθηκευμένος στο dictionary `timers`.

`createTimer(self, id, timeOut, msgType, peerSock):`

Δημιουργεί έναν QTimer που αντιπροσωπεύει το Random Delay της αποστολής ενός μηνύματος, και τον αποθηκεύει στο dictionary `timers`.

`removeTimer(self, id):`

Διαγράφει ένα συγκεκριμένο QTimer από το dictionary `timers`.

6.10 SrvThread.py

Σε αυτό το Module ορίζονται οι κλάσεις `UDP4Server` και `UDP6Server` οι οποίες είναι υποκλάσεις της `asyncore.dispatcher` και χειρίζονται την επικοινωνία της εφαρμογής χρησιμοποιώντας αντίστοιχα το IPv4 και IPv6 ως πρωτόκολλο

Δικτύου. Στον constructor της κάθε κλάσης δημιουργείται αρχικά ένα UDP socket και στη συνέχεια το socket κάνει join στο προκαθορισμένο Multicast Group. Για την αποστολή Multicast δεδομένων δεν χρειάζεται να γίνει join σε κάποιο group, αυτό χρειάζεται μόνο για τη λήψη.

6.11 UI*.py

Όλα τα modules της μορφής `UI*.py` παράγονται από τα αντίστοιχα αρχεία `UI*.ui` με τη χρήση του εργαλείου `pyuic4`.

```
$ pyuic4 -o UIAbout.py UIAbout.ui
```

Τα *.ui αρχεία παράγονται από το QtDesigner και δεν είναι τίποτε άλλο από απλά XML αρχεία που περιέχουν τη δομημένη περιγραφή ενός γραφικού περιβάλλοντος, όπως αυτό σχεδιάστηκε μέσα από το QtDesigner. Το εργαλείο `pyuic4` αναλαμβάνει την παραγωγή κώδικα Python από αυτά τα αρχεία.

7. Αναφορές

Cryptographic hash function. (n.d.). *Wikipedia*. Retrieved January 12, 2012, from http://en.wikipedia.org/wiki/Cryptographic_hash_function

Deering, S., Cisco, Hinden, R., & Nokia (1998, December). *Internet Protocol, Version 6 (IPv6) Specification*. RFC 2460 (Standard). Updated by RFC 5095, 5722, 5871, 6437, obsoletes RFC 1883.

Krawczyk, H., Bellare, M., & Canetti, R. (1997). *HMAC: Keyed-Hashing for Message Authentication*. RFC 2104 (Informational). Updated by RFC 6151.

Postel, J. (Ed.). (1981). *Internet Protocol*. RFC 791 (Standard). Updated by RFC 1349.

Postel, J. (1980). *User Datagram Protocol*. RFC 768 (Standard).

Netsh commands for Interface IPv6. (2005, January 21). *Microsoft TechNet*. Retrieved January 28, 2012, from http://technet.microsoft.com/en-us/library/cc740203%28WS.10%29.aspx#BKMK_16

8. Βιβλιογραφία

Cheng, P., & Glenn, R. (1997). *Test Cases for HMAC-MD5 and HMAC-SHA-1*. RFC 2202 (Informational).

Comer, D. E. (2000). *Internetworking with TCP/IP: Principles, Protocols, and Architectures*. (4th ed.). Prentice-Hall.

Crawford, M. (1998). *Transmission of IPv6 Packets over Ethernet Networks*. RFC 2464 (Proposed Standard). Updated by RFC 6085, obsoletes RFC 1972.

Handley, M., Floyd, S., Whetten, B., Kermode, R., Vicisano, L., & Luby, M. (2000). *The Reliable Multicast Design Space for Bulk Data Transfer*. RFC 2887 (Informational).

Kawamura, S., & Kawashima, M. (2010). *A Recommendation for IPv6 Address Text Representation*. RFC 5952 (Proposed Standard). Updates RFC 4291.

Krawczyk, H., Bellare, M., & Canetti, R. (1997). *HMAC: Keyed-Hashing for Message Authentication*. RFC 2104 (Informational). Updated by RFC 6151.

Lutz, M. (2009). *Learning Python*. (4th ed.). O'Reilly.

Meyer, D. (1998). *Administratively Scoped IP Multicast*. RFC 2365 (Best Current Practice).

OpenSSH. (2011, September 11). *OpenBSD Reference Manual*. Retrieved December 4, 2011, from http://www.openbsd.org/cgi-bin/man.cgi?query=ssh

Postel, J. (1980). *User Datagram Protocol*. RFC 768 (Standard).

Quinn, B., & Almeroth, K. (2001). *IP Multicast Applications: Challenges and Solutions*. RFC 3170 (Informational).

Shutdown. (n.d.). *Microsoft Technet*. Retrieved December 4, 2011, from http://technet.microsoft.com/en-us/library/bb491003.aspx

Summerfield, M. (2008). *Rapid GUI Programming with Python and Qt: The Definitive Guide to PyQt Programming*. Prentice-Hall.

Turner, S., & Chen, L. (2011). *Updated Security Considerations for the MD5 Message-Digest and the HMAC-MD5 Algorithms*. RFC 6151 (Informational). Updates RFCs 1321, 2104.

Whetten, B., Vicisano, L., Kermode, R., Handley, M., Floyd, S., & Luby, M. (2001). *Reliable Multicast Transport Building Blocks for One-to-Many Bulk-Data Transfer*. RFC 3048 (Informational).

9. Οδηγός Χρήσης Λογισμικού

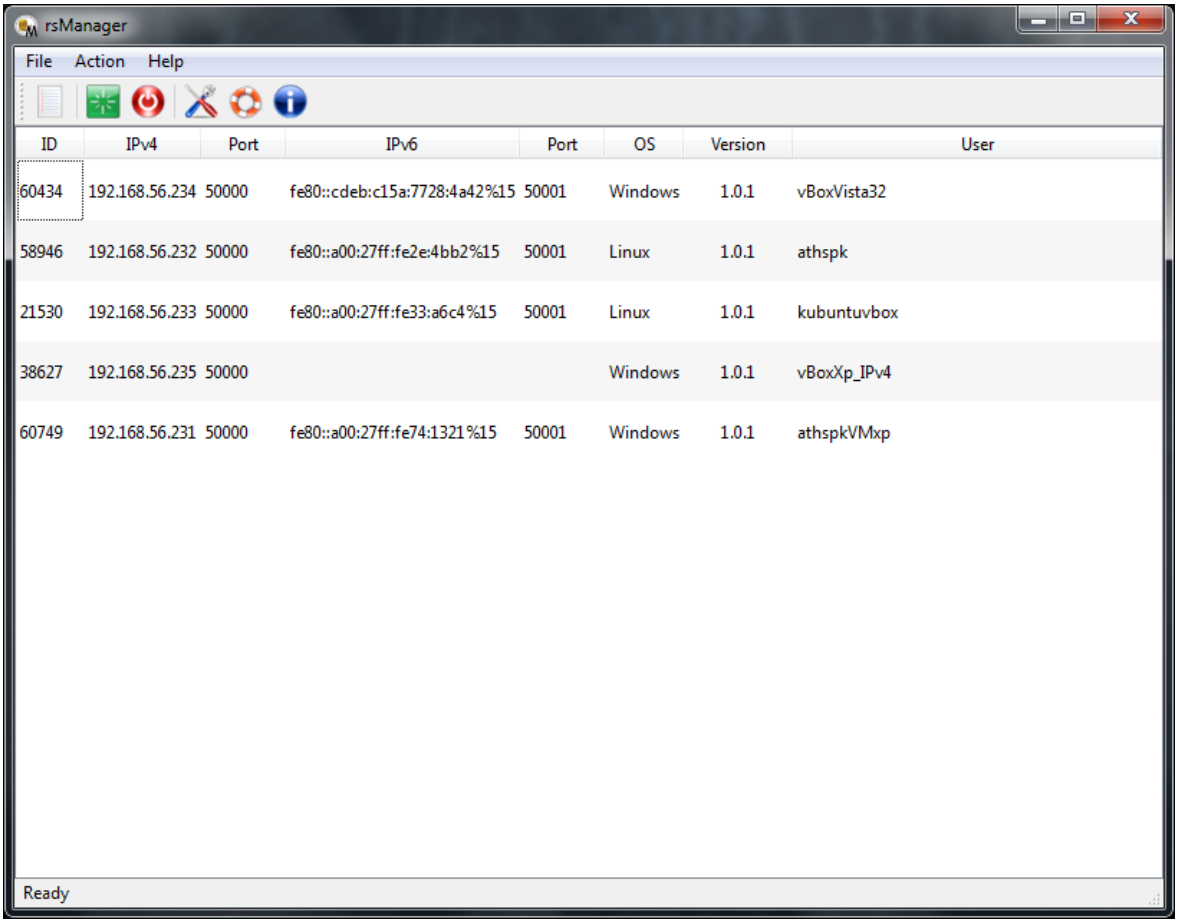
Για τη λειτουργία της εφαρμογής απαιτείται η ύπαρξη:

- Python, έκδοση 3.2.x ή μεταγενέστερη
- PyQt, έκδοση 4.8.x ή μεταγενέστερη

9.1 Manager

Τρέχοντας τον manager με την εντολή `python3 rsManager.py`, εμφανίζεται το αρχικό interface (Εικ. 9.1).

9.1.1 Κεντρικό παράθυρο



Εικ. 9.1: Κεντρικό παράθυρο του manager

Είναι το κυρίως παράθυρο της εφαρμογής όπου απεικονίζονται όλοι οι Agents. Για κάθε Agent εμφανίζεται κατά σειρά, το ID του, η διεύθυνση και το port για το IPv4, η διεύθυνση και το port για το IPv6 (αν υποστηρίζεται), το λειτουργικό σύστημα στο οποίο τρέχει ο Agent, ο αριθμός έκδοσής του, και το όνομα χρήστη στο συγκεκριμένο υπολογιστή.

Επιλέγοντας έναν ή περισσότερους Agents, μπορούμε να εκτελέσουμε Restart ή Shutdown από το μενού “Action” ή από το toolbar. Συνολικά, οι ενέργειες που μπορούμε να εκτελέσουμε είναι οι εξής:


Εμφανίζει το πιο πρόσφατο αρχείο καταγραφής συμβάντων


Εκτελεί Restart στους επιλεγμένους Agents

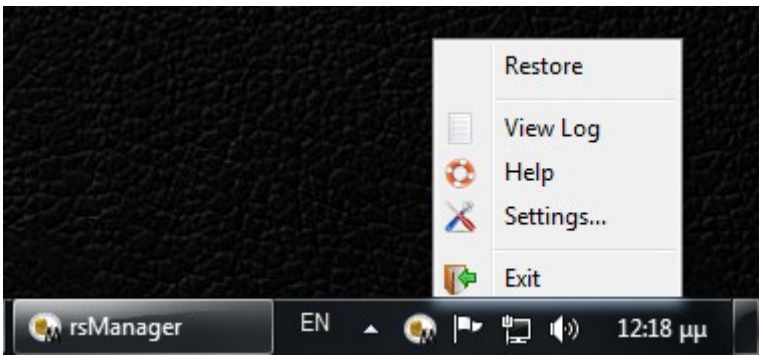

Εκτελεί Shutdown στους επιλεγμένους Agents


Εμφανίζει το παράθυρο ρυθμίσεων

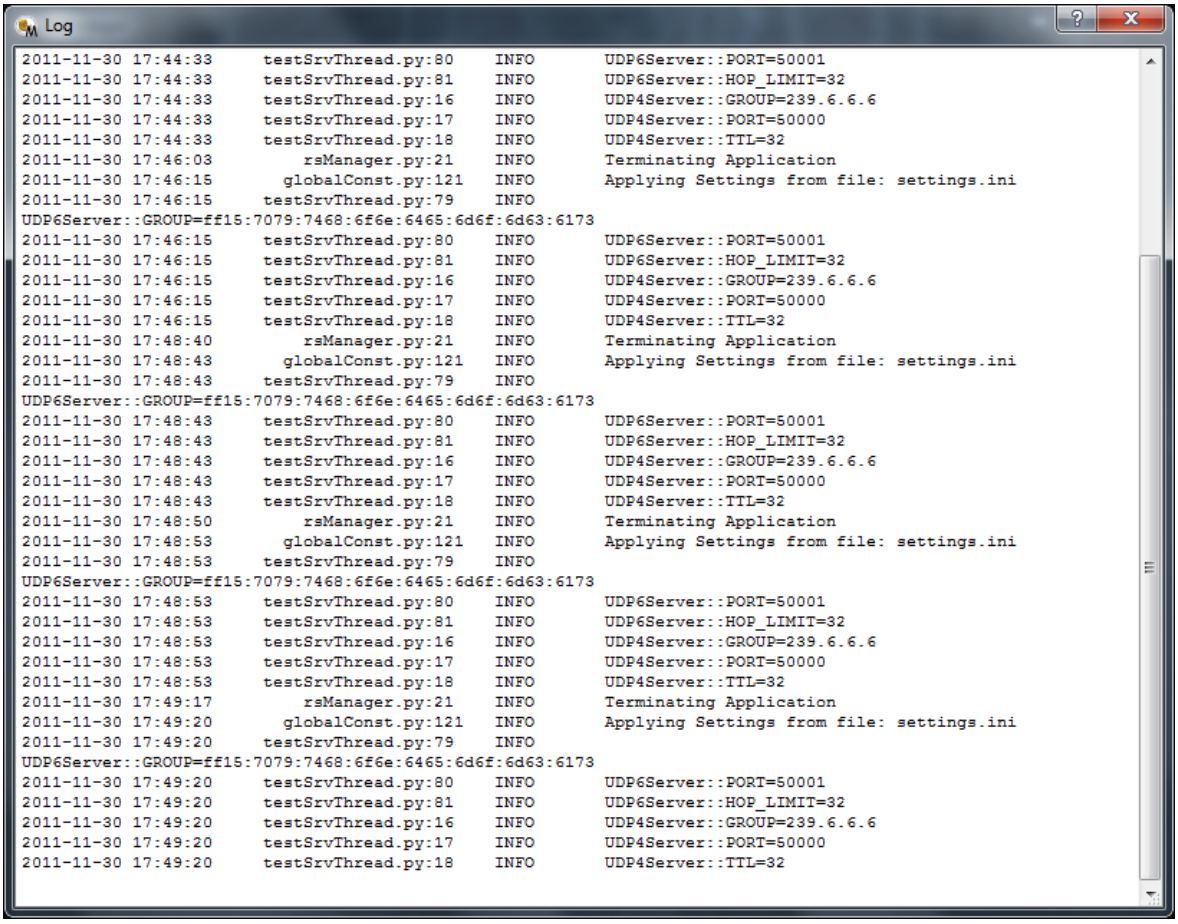

Εμφανίζει το αρχείο Βοηθείας


Εμφανίζει γενικές πληροφορίες για την εφαρμογή

Οι παραπάνω λειτουργίες, εκτός των Restart και Shutdown, μπορούν επίσης να εκτελεστούν και από το εικονίδιο της εφαρμογής στο system tray όπως φαίνεται στην παρακάτω εικόνα.



9.1.2 Παράθυρο Συμβάντων (Log)

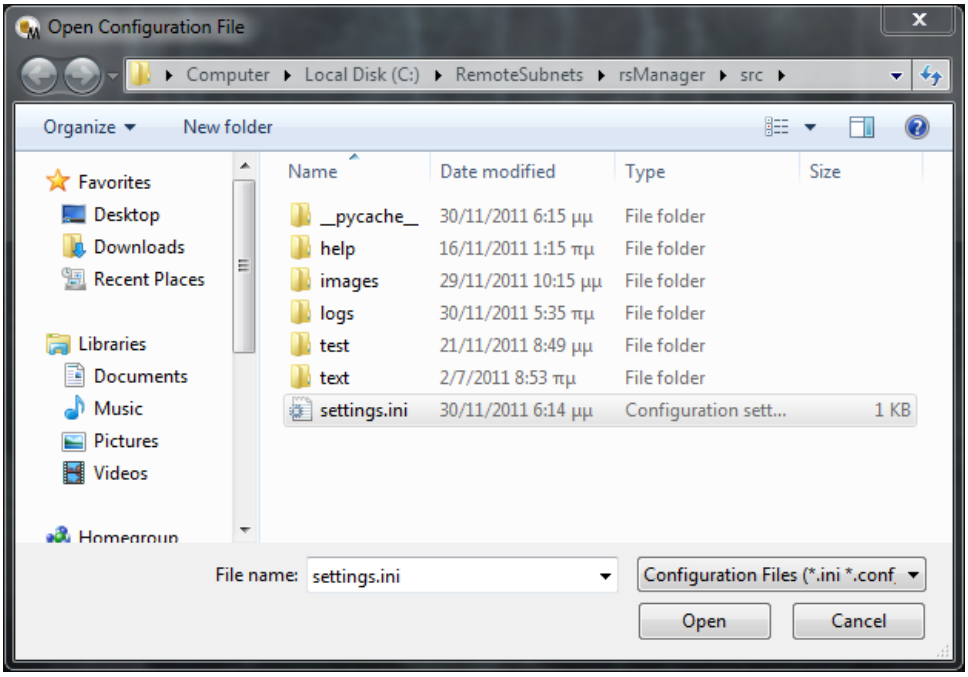


Εικ. 9.2: Παράθυρο Συμβάντων (Log)

Σε αυτό το παράθυρο εμφανίζεται το πιο πρόσφατο αρχείο συμβάντων. Κάθε γραμμή περιλαμβάνει:

- Ημερομηνία και ώρα συμβάντος
- Όνομα του script και αριθμός γραμμής όπου παρουσιάστηκε το συμβάν
- Επίπεδο κρισιμότητας συμβάντος: `{DEBUG|INFO|WARNING|ERROR|CRITICAL}`
- Μήνυμα συμβάντος

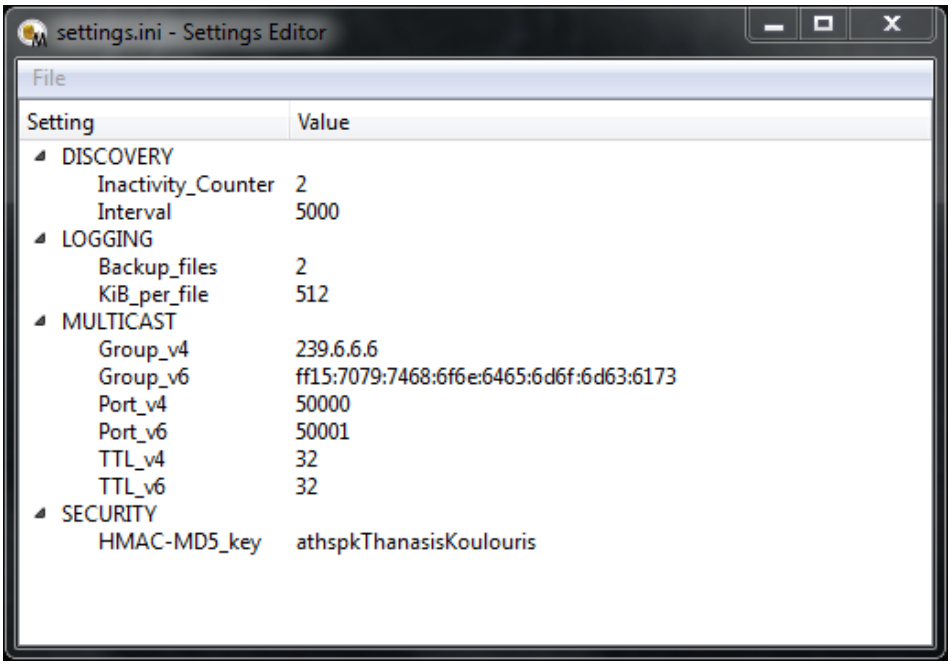
9.1.3 Παράθυρο Ρυθμίσεων (Settings)



Εικ. 9.3: Επιλογή αρχείου ρυθμίσεων

Αρχικά εμφανίζεται ένα παράθυρο διαλόγου (εικόνα 9.3) από όπου μπορούμε να επιλέξουμε το αρχείο στο οποίο θα γίνει η καταγραφή των ρυθμίσεων. Το default αρχείο είναι το “settings.ini” το οποίο είναι ένα απλό αρχείο κειμένου όπου οι παράμετροι είναι οργανωμένες σε Sections.

Στη συνέχεια ανοίγει το παράθυρο ρυθμίσεων (εικόνα 9.4) από το οποίο μπορούμε να ορίσουμε τιμές για διάφορες παραμέτρους που χρησιμοποιούνται στην εφαρμογή. Οι τιμές γράφονται στο αρχείο “settings.ini” και οι αλλαγές εφαρμόζονται την επόμενη φορά που θα εκτελεστεί η εφαρμογή. Αν το αρχείο δεν υπάρχει ή περιέχει λάθος τιμές, τότε εφαρμόζονται οι default τιμές όπου απαιτείται. Οι κανόνες εγκυρότητας των παραμέτρων που δίνονται από το χρήστη φαίνονται στον Πίνακα 4.1.

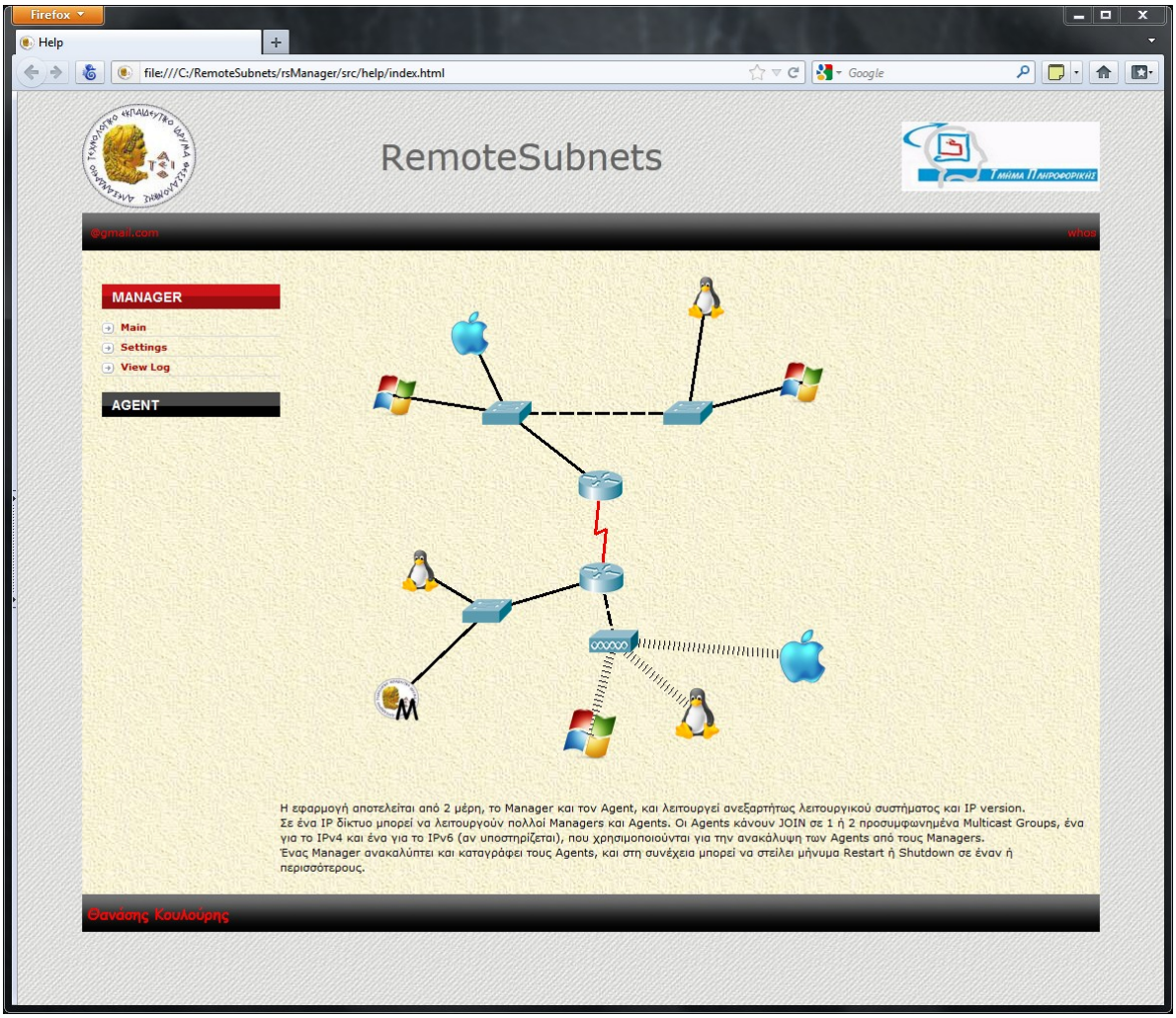


Εικ. 9.4: Παράθυρο Ρυθμίσεων manager

Πίνακας 9.1: Πεδία αρχείου ρυθμίσεων manager

Πεδίο	Περιγραφή
`Inactivity_Counter`	Πόσες φόρες επιτρέπεται σε έναν Agent να μην απαντήσει, πριν θεωρηθεί ανενεργός.
`Interval`	Κάθε πόσα milliseconds θα στέλνει ο Manager αίτημα ανακάλυψης (DISCOVERY_ASK).
`Backup_files`	Μέγιστο πλήθος log files.
`KiB_per_file`	Το μέγιστο μέγεθος του κάθε log file. Αν το μέγεθος φτάσει αυτή την τιμή, τότε το τρέχον αρχείο κλείνει και δημιουργείται νέο αρχείο log. Πάντα όμως τηρείται το μέγιστο πλήθος που έχει οριστεί.
`Group_v4`	Η διεύθυνση του Multicast Group για το IPv4
`Group_v6`	Η διεύθυνση του Multicast Group για το IPv6
`Port_v4`	Το Port του UDP server για το IPv4
`Port_v6`	Το Port του UDP server για το IPv6
`TTL_v4`	Η τιμή του πεδίου TTL του IPv4
`TTL_v6`	Η τιμή του πεδίου Hop Limit του IPv6
`HMAC-MD5_key`	Το κλειδί που χρησιμοποιείται για τον υπολογισμό του MD5 hash. Πρέπει να είναι ίδιο και στα δυο άκρα (Manager-Agents) και να έχει μέγεθος μεγαλύτερο ή ίσο με 16 χαρακτήρες. (Krawczyk, Bellare, & Canetti, 1997, p. 4).

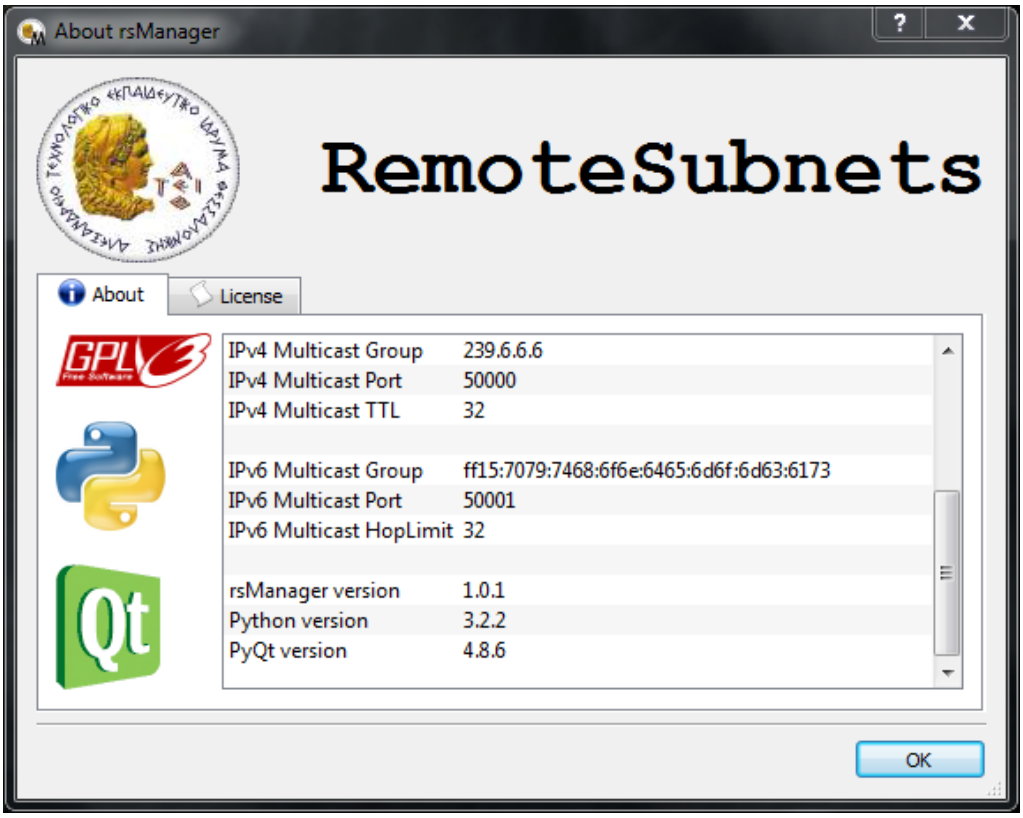
9.1.4 Παράθυρο Βοηθείας (Help)



Εικ. 9.5: Παράθυρο Βοηθείας

Επιλέγοντας Help, ανοίγει ο default web browser με τον οδηγό χρήσης της εφαρμογής.

9.1.5 Παράθυρο Πληροφοριών (About)



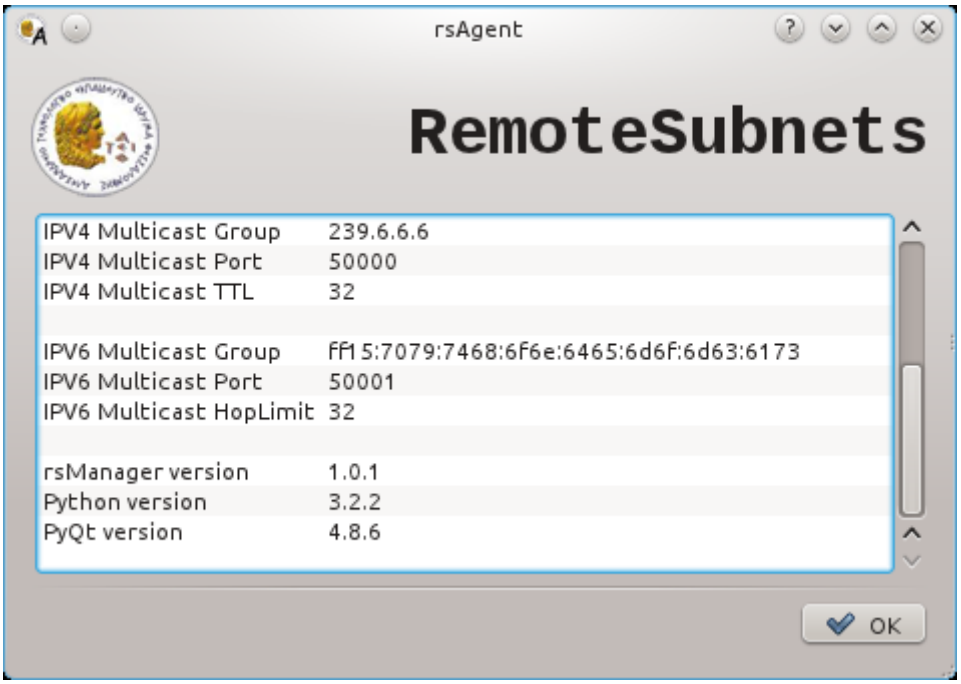
Εικ. 9.6: Παράθυρο Πληροφοριών

Σε αυτό το παράθυρο εμφανίζονται γενικές πληροφορίες σχετικά με τις παραμέτρους λειτουργίας της εφαρμογής, και η άδεια χρήσης GPLv3 όπως ορίζεται στις προδιαγραφές της πτυχιακής εργασίας.

9.2 Agent

Τρέχοντας τον manager με την εντολή `python3 rsAgent.py`, εμφανίζεται το αρχικό interface (Εικ. 9.7). Ο agent έχει πολύ απλό interface, καθώς ο ρόλος του είναι κυρίως παθητικός.

9.2.1 Κεντρικό Παράθυρο

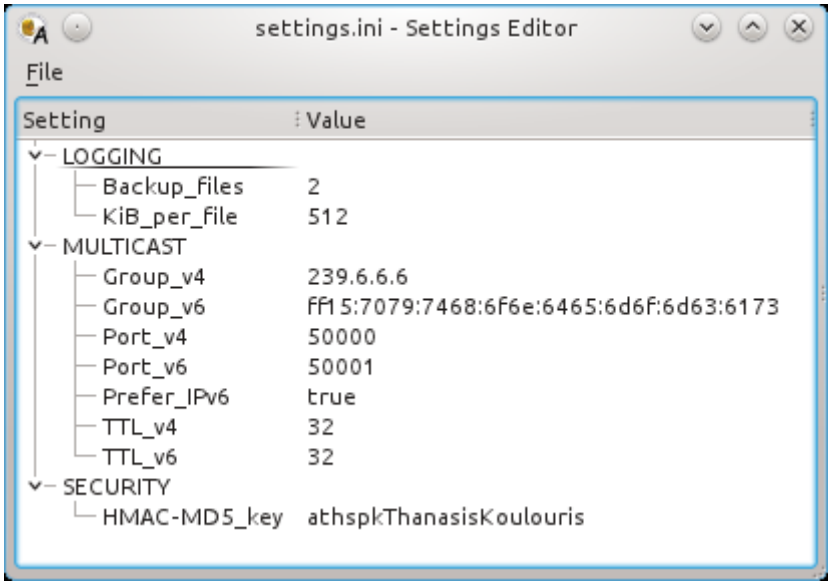


Εικ. 9.7: Κεντρικό Παράθυρο agent

Σε αυτό το παράθυρο εμφανίζονται γενικές πληροφορίες σχετικά με τις παραμέτρους λειτουργίας της εφαρμογής.

9.2.2 Παράθυρο Ρυθμίσεων (Settings)

Λειτουργεί με τον ίδιο τρόπο όπως περιγράφηκε στο κεφάλαιο 9.1.3 για τον manager, με τη διαφορά ότι δεν υπάρχουν ρυθμίσεις για το Discovery.



Εικ. 9.8: Παράθυρο Ρυθμίσεων agent