ΑΤΕΙ ΘΕΣΣΑΛΟΝΙΚΗΣ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΔΙΚΤΥΑΚΟΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ

Σιδηράκη Ελένη

Επιβλέπων Καθηγητής

Π.Ράπτης

ΘΕΣΣΑΛΟΝΙΚΗ 2011

ΠΕΡΙΕΧΟΜΕΝΑ

ΠΕΡΙΛΗΨΗ

Η παρούσα πτυχιακή εργασία φιλοδοξεί να παρουσιάσει τις πτυχές του Δικτυακού προγραμματισμου.Καταρχήν,τι είναι δικτυακός προγραμματισμος – Ο Δικτυακος προγραμματισμος περιλαμβάνει τη συγγραφή προγραμμάτων που διευκολύνουν τις διαδικασίες για να επικοινωνούν μεταξύ τους μέσω δικτύου υπολογιστών.Η παρούσα πτυχιακή εργασία θα επικεντρωθει στη συγγραφη προγραμματων στη γλωσσα προγραμματισμου Python και προσδοκει να αναφερει και λιγα λογια για την γλωσσα προγραμμτισμου Java,οσον αφορα παντα το δικτυακο προγραμματισμο. Η Python είναι μια interpreted, object-oriented, υψηλού επιπέδου γλώσσα προγραμματισμού με dynamic semantics. Είναι υψηλού επιπέδου κτισμένη στις δομές δεδομένων,είναι ιδιαίτερα ελκυστική για την ταχεία ανάπτυξη εφαρμογών, καθώς και για χρήση ως scripting γλώσσα ή για να συνδέσετε υπάρχοντα στοιχεία μαζί. Είναι απλό και εύκολο να το μάθει κανεις την σύνταξη της Python,η οποία τονίζει αναγνωσιμότητα και κατά συνέπεια μειώνει το κόστος συντήρησης του προγράμματος.Η Python υποστηρίζει τις ενότητες και τα πακέτα, η οποία ενθαρρύνει program modularity και την επαναχρησιμοποίηση κώδικα. Ο διερμηνέας Python και η πλούσια βιβλιοθήκη της είναι διαθέσιμη σε μορφή δυαδικού κώδικα ή χωρίς χρέωση για όλες τις σημαντικές πλατφόρμες, και μπορεί να διανέμεται ελεύθερα.Σε αυτην την πτυχιακη θα γινει λογος για το πώς η Python χειριζετε διαφορα πρωτοκολλα κ αρχιτεκτονικες.Τα πρωτοκολλα για τα οποια θα γίνει λόγος είναι το TCP,UDP,DNS,Ipv4 και Ipv6.Επισης,θα αναφερθει η Client/Server αρχιτεκτονικη,η Unicast και Multicast μετάδοση δεδομένων και τέλος διαφορα προγραμματα για το πώς στελνουμε mail.Ολα τα προγραμματα εχουν πραγματοποιηθει σε περιβαλλον Unix και συγκεκριμενα Ubuntu.

ABSTRACT

This thesis seeks to present aspects of the Network Programming.First of all,what is network programming – Computer network programming involves writing computer programs that enable processes to communicate with each other across a computer network.This thesis will focus on writing programs in Python programming language and it expects to indicate a few words about the programming language Java, regarding everything to network programming. Python is an interpreted, object-oriented, high level programming language with dynamic semantics. It is a high-level built in data structures, combined with dynamic typing and dynamic binding, make it particularly attractive for rapid application development, and for use as a scripting language or to connect existing components together. It's simple and easy to learn the syntax of Python, which emphasizes readability and therefore reduces the costs of maintaining the program.Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and an extensive library is available in binary code, or free for all major platforms and can be delivered for free.In this thesis,will be discussed about many protocols and architectures how Python handle them.The protocols for which will be discussed is the TCP, UDP, DNS, Ipv4 and Ipv6.Also,it will be indicated the Client / Server architecture, Unicast and Multicast data transmission, and finally various programs on how sending mail. All programs have been written in Unix environment, specifically Ubuntu.

ΚΕΦΑΛΑΙΟ 1^{Ο}

ΕΙΣΑΓΩΓΗ

Στα τελευταία 10 χρόνια, ο δικτυακός προγραμματισμός έχει σταματήσει να είναι η επαρχία των λίγων ειδικών και να γίνει ένα βασικό μέρος της εργαλειοθήκης κάθε προγραμματιστή. Σήμερα, τα περισσότερα προγράμματα έχουν επίγνωση του δικτύου. Εκτός από τις κλασικές εφαρμογές όπως το ηλεκτρονικό ταχυδρομείο, προγράμματα περιήγησης στο Web, Telnet Clients, οι περισσότερες μεγάλες εφαρμογές έχουν κάποιο επίπεδο δικτύωσης. Σήμερα,υπάρχουν πολλές γλώσσες προγραμματισμού ,οι οποίες επιτρέπουν τον δικτυακό προγραμματισμό, κάποιες από αυτές είναι η Python,Java,Perl και C.

Στην εργασία αυτή,θα επικεντρωθεί το ενδιαφέρον στη γλώσσα Python. Έχοντας δοκιμάσει σχεδόν όλες τις τελευταίες εκδόσεις της Python κ όλα τα τελευταια περιβάλλοντα ubuntu ,κατέληξα να συγγράψω όλα τα προγράμματα σε περιβαλλον unix και συγκεκριμένα την δοκιμαστικη έκδοση ubuntu 11.04,έκδοση Python 2.7.Επίσης,υπάρχει η δυνατότητα εγκατάστασης της Python σε windows και Mac-Os περιβάλλοντα.Η Python είναι ενσωματωμένη σε όλα τα συστήματα unix,χωρις όμως πρόσθετα χαρακτηριστικά ,τα οποία χρειάστηκε να τα εγκαταστήσω στη συνεχεία. Η Python είναι μια εξαιρετικά ισχυρή δυναμική γλώσσα προγραμματισμού που χρησιμοποιείται σε ένα ευρύ φάσμα τομέων εφαρμογής. Η Python είναι συχνά σε σύγκριση με το Tcl, Perl, Ruby Scheme ή Java. Μερικά από βασικά χαρακτηριστικά του γνωρίσματα περιλαμβάνουν:

- πολύ σαφής, ευανάγνωστη σύνταξη
- ισχυρές δυνατότητες
- intuitive object orientation
- φυσική έκφραση του procedural code
- full modularity, υποστηρίζοντας ιεραρχικά πακέτα
- exception-based error handling
- πολύ υψηλό επίπεδο δυναμικών τύπων δεδομένων
- extensive standard libraries and third party modules for virtually every task
- οι επεκτάσεις και οι ενότητες είναι εύκολα γραμμέν σε C, C + + (ή Java για Jython, ή. NET γλώσσες για IronPython)
- embeddable within applications as a scripting interface

Οι οπαδοί της Python χρησιμοποιούν τη φράση "batteries included" για να περιγράψουν την πρότυπη βιβλιοθήκη, η οποία καλύπτει τα πάντα, από την ασύγχρονη επεξεργασία μέχρι αρχεία zip. Η ίδια η γλώσσα είναι ένα ευέλικτο εργοστάσιο παραγωγής ηλεκτρικού ρεύματος που μπορεί να χειριστεί σχεδόν κάθε

τομέα προβλήμματος. Κτίζεις το δικό σου web server σε τρεις γραμμές κώδικα. Δημιουργία ευέλικτων data-driven code χρησιμοποιώντας τις ισχυρές και δυναμικές ικανότητες ενδοσκόπησης της Python και άλλα επίσης προηγμένα χαρακτηριστικά, όπως οι meta-classes, duck typingκαι decorators.Η Python σας επιτρέπει να γράψετε τον κώδικα που χρειάζεστε, γρήγορα. Και, χάρη σε ένα ιδιαίτερα βελτιστοποιημένο byte compiler και βιβλιοθήκες υποστήριξης , ο Python κώδικας τρέχει κατι περισσότερο από γρήγορα για τις περισσότερες εφαρμογές. Η παραδοσιακή εφαρμογή CPython χρησιμοποιεί μία bytecode εικονική μηχανή,η PyPy υποστηρίζει just-in-time (JIT) σύνταξη σε κώδικα μηχανής. Επίσης, Jython και IronPython υποστηρίζουν JIT μεταγλώτιση στις αντίστοιχες υλοποιήσεις εικονικής μηχανής τους.

Η Python μπορεί να ενοποιηθεί με COM,. NET, καθώς και αντικείμενα CORBA. Για Java βιβλιοθήκες, χρησιμοποιείτε η Jython, μια υλοποίηση της Python για την Java Virtual Machine. For .NET, δοκιμάστε την IronPython , Microsoft's new implementation of Python for .NET, ή Python for .NET. Python υποστηρίζεται επίσης για το Internet Communications Engine (ICE) και πολλές άλλες τεχνολογίες ολοκλήρωσης.Εάν βρείτε κάτι που η Python δεν μπορεί να κάνει, ή αν χρειάζεστε το μεγιστο της απόδοσης του κώδικα χαμηλού επιπέδου, μπορείτε να γράψετε επεκτάσεις modules σε C ή C + +, ή wrap τον υπάρχοντα κώδικα με SWIG ή Boost.Python. Το να κάνεις Wrapped modules, εμφανίζονται στο πρόγραμμά σας ακριβώς όπως τη μητρική κώδικα Python. Το πόσο ολοκληρωμένη είναι αυτή η γλωσσά είναι αυτό που την κάνει εύκολη. Μπορείτε επίσης να κάνετε το αντίθετο, να ενσωματώσετε την Python στη δική σας εφαρμογή, παρέχοντας στους χρήστες σας μια γλώσσα που θα απολαύσουν τη χρήση.Η Python είναι διαθέσιμη για όλα τα σημαντικά λειτουργικά συστήματα: Windows, Linux / Unix, OS / 2, Mac, Amiga, μεταξύ άλλων. Υπάρχουν ακόμα και εκδόσεις που λειτουργούν με. NET, τη Java Virtual Machine, και τα Nokia Series 60 κινητά τηλέφωνα.

Η ομάδα συζήτησης της Python είναι γνωστή ευρέως ως μία από τις πιο φιλικές . Οι προγραμματιστές και η κοινότητα χρηστών διατηρεί ένα wiki, το οποίο φιλοξενεί διεθνή και τοπικά συνέδρια, τρέχει να επιταχύνει την ανάπτυξη, και συμβάλλει στο online αποθετήριο κώδικα.Η Python παρέχετε με πλήρη τεκμηρίωση, τόσο ενσωματωμένη στη γλώσσα αλλά και ως ξεχωριστές ιστοσελίδες. Υπάρχουν ηλεκτρονικά προγράμματα εκμάθησης,τα οποία στόχο εέχουν να διδάσκουν τόσο το έμπειρο προγραμματιστή όσο και το νεοφερμένο. Η Python είναι στο πλαίσιο μίας open source άδειας που την καθιστά ελευθέρως χρησιμοποιούμενη,όπως κ την διανομή τους, ακόμη και για εμπορική χρήση. Η άδεια Python χορηγείται από το Ίδρυμα Λογισμικού Python.Μπορείτε να την βρείτε στην παρακάτω διευθυνση:

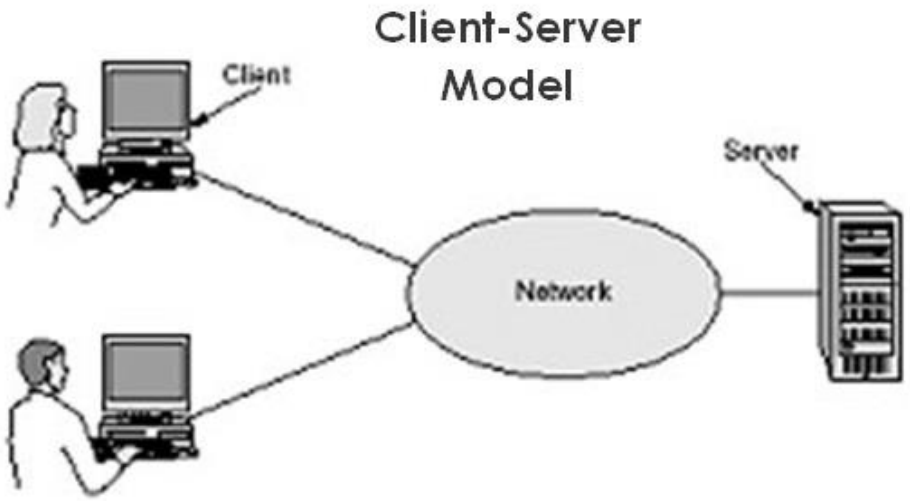
http://www.python.org/getit/ απλά διαλέγετε την εκδόση που θέλετε κ το λειτουργικό σύστημα γα το οποίο την χρειάζεστε.

ΚΕΦΑΛΑΙΟ 2Ο

Εισαγωγή στη Client/Server Αρχιτεκτονικη

Το client-server μοντέλο είναι ένα μοντέλο στον υπολογιστή που λειτουργεί ως κατανεμημένη εφαρμογή που χωρίζει τις εργασίες ή τον φόρτο εργασίας μεταξύ των παρόχων ενός πόρου ή μιας υπηρεσίας, που ονομάζεται διακομιστές, και από τους αιτούντες υπηρεσίας, που ονομάζεται πελάτες. Συχνά οι πελάτες και οι διακομιστές επικοινωνούν μέσω δικτύου ηλεκτρονικών υπολογιστών με ξεχωριστά hardware, αλλά και οι δύο πελάτης και διακομιστής μπορούν να διαμένουν στο ίδιο σύστημα. Ένα μηχάνημα server είναι ένας host που εκτελεί ένα ή περισσότερα προγράμματα διακομιστή,τα οποία που μοιράζονται τους πόρους τους με τους πελάτες. Ένας πελάτης δεν μοιράζει κανένα από τους πόρους της, αλλά ζητά περιεχόμενο κ λειτουργίες από το διακομιστή.Οι πελάτες για να επικοινωνήσουν με τους διακομιστές πρέπει να αρχίσουν μια session,για να είναι σε θέση οι διακομιστές να περιμένουν τα εισερχόμενα αιτήματα.

Το client-server μοντέλο περιγράφει τη σχέση των συνεργαζόμενων προγραμμάτων σε μια εφαρμογή. Το στοιχείο διακομιστή παρέχει μια λειτουργία ή μια υπηρεσία σε έναν ή πολλούς πελάτες, η οποία κινεί τις αιτήσεις για τέτοιες υπηρεσίες. Οι λειτουργίες, όπως η ανταλλαγή e-mail, πρόσβαση στο διαδίκτυο και πρόσβαση σε βάσεις δεδομένων, είναι χτισμένες πάνω στο μοντέλο client-server. Οι χρήστες έχουν πρόσβαση σε τραπεζικές υπηρεσίες,χρησιμοποιώντας τον υπολογιστή τους,κάνοντας χρήση ενός πελάτη web browser για να στείλετε ένα αίτημα για ένα web server σε μια τράπεζα. Το πρόγραμμα μπορεί με τη σειρά του προωθεί το αίτημα για στο δικο του πρόγραμμα-πελάτη της βάσης δεδομένων που στέλνει μια αίτηση σε ένα διακομιστή βάσης δεδομένων σε άλλο υπολογιστή της τράπεζας για να ανακτήσει τις πληροφορίες του λογαριασμού. Το πρόγραμμα με την σειρα του θα προωθήσει αυτήν την αίτηση στο δικό του πελάτη της βάσης δεδομένων,το οποίο θα στείλει μία αίτηση στο διακομιστή μια βάσης δεδομένων σε ένα άλλο υπολογιστή μίας άλλης τράπεζας για να ανακτήσει τις πληροφορίες του λογαριασμού.Το υπόλοιπο επιστρέφεται στη βάση δεδομένων πελάτη της τράπεζας , η οποία με τη σειρά της εξυπηρετεί το πρόγραμμα περιήγησης πελατη στο Web και εμφανίζει τα αποτελέσματα στον χρήστη. Το client-server μοντέλο έχει καταστεί μία από τις κεντρικές ιδέες των υπολογιστών του δικτύου. Πολλές επιχειρηματικές εφαρμογές που γράφονται σήμερα χρησιμοποιούν το client-server μοντέλο. Έτσι κάνουν κάποια κύρια applications πρωτόκολλα του Διαδικτύου, όπως HTTP, SMTP, Telnet, και DNS.



Εικόνα 1 Client/Server Αρχιτεκτονική

Τώρα,όσον αφορα την Python,ας αρχίσουμε με ένα πολύ απλό scipt.Έχουμε μία ταχυδρομική διεύθυνση,η οποία είναι

207 N. Defiance St
Archbold, OH

Και ενδιαφερόμαι να μάθω το γεωγραφικό πλάτος και μήκος αυτή της φυσικής διεύθυνσης. Είναι ακριβώς αύτο που μου παρέχει Google κ ονομάζετε "Maps API» .Είναι αυτό που μπορεί να εκτελέσει μια τέτοια μετατροπή.

Όταν θέλετε να χρησιμοποίησετε μια νέα υπηρεσία δικτύου , αξίζει πάντα τον κόπο να ξεκινήσετε από το γεγονός αν κάποιος έχει ήδη εφαρμόσει το πρωτόκολλο-στην περίπτωση αυτή, υπάρχει το Google Maps
πρωτόκολλο-με το οποίο το πρόγραμμά σας θα πρέπει να μιλήσει.

Ξεκινώντας την αναζήτηση με την ανάγνωση της τεκμηρίωσης της Βιβλιοθήκης Python Standard, ψάχνοντας για οτιδήποτε έχει να κάνει με το Google Maps:

http://docs.python.org/library/

Δεν υπάρχει κατι,το οποίο να αναφέρετε σε αυτήν την βιβλιοθήκη. Αλλά είναι σημαντικό για έναν Python προγραμματιστή να εξετάζει τον πίνακα περιεχομένων της Βιβλιοθήκης πρότυπο αρκετά συχνά, ακόμα και αν συνήθως δεν βρίσκετε αυτό που αναζητάτε, διότι κάθε ανάγνωση θα σας κάνει πιο εξοικειωμένους με τις υπηρεσίες που περιλαμβάνονται με την Python.

Από την στιγμή που η πρότυπη βιβλιοθήκη δεν έχει ένα πακέτο για να μας βοηθήσει, πρέπει να στρέψουμε το ενδιαφέρον μας Python Package Index, μια εξαιρετική πηγή

για την εύρεση όλων των ειδών γενικής χρήσης πακέτων της Python συνεισφορά από άλλους προγραμματιστές και οργανισμούς από όλο τον κόσμο. Μπορείτε, επίσης, φυσικά, να ρίξετε μια ματιά στην ιστοσελίδα
του προμηθευτή του οποίου την υπηρεσία θα χρησιμοποιήσετε για να δείτε εάν θα παρέχει μία βιβλιοθήκη Python για να έχετε πρόσβαση.
Ή μπορείς να κάνεις μια γενική αναζήτηση στο Google για "Python" συν το όνομα του δικτυακής υπηρεσίας που θέλετε να κάνετε χρήση, και να δούμε αν υπάρχει στα αποτελέσματα.
Στην περίπτωση αυτή, έψαξα την Python Package Index,, που υπάρχει σε αυτό το URL:

http://pypi.python.org/

Κάνοντας μια αναζήτηση για το Google Maps, βρέθηκε ένα πακέτο με το όνομα googlemaps και παρέχει ένα καθαρό περιβάλλον εργασίας στα χαρακτηριστικά του (αν και παρατηρήσετε από την περιγραφή του,ότι δεν είναι γραμμένο από τον προμηθευτή, αλλά από κάποιον εκτός της Google):

http://pypi.python.org/pypi/googlemaps/

Αρχικά,πληκτρολογούμε τις εντολές :

```
$ python -c 'import googlemaps'
Traceback (most recent call last):
File "<string>", line 1, in <module>
ImportError: No module named googlemaps
```

Όπως,μπορούμε να δουμε δεν υπάρχει το πακέτο εγκατεστημένο,για να γίνει εγκατάσταση,χρησιμοποιώ την pip εντολή,η οποία είναι μέσα στο virtualenv,αν δεν είναι εγκατεστημένο μπορείτε να το βρείτε κ να το κατεβάσετε από :

http://pypi.python.org/pypi/virtualenv

Τώρα πλέον,μπορουμε να τρέξουμε το πρώτο απλό πρόγραμμα search1.py

Προγραμμα 1.1 Να εμφανιστεί γεωγραφικό μήκος και πλάτος

```
#!/usr/bin/env python
from googlemaps import GoogleMaps
address = '207 N. Defiance St, Archbold, OH'
print GoogleMaps().address_to_latlng(address)
```

Τρέχοντας το παίρνουμε τα εξής αποτελέσματα:

```
$ python search1.py
(41.5228242, -84.3063479)
```

Το πρώτο μας πρόγραμμα χρησιμοποιούσε μια third-party Python βιβλιοθήκη,κατεβασμένη από το Python Package Index για να λύσει το προβλημμά

μας.Αν δεν υπήρχε η βιβλιοθήκη τι θα γινόταν?Καν αν επρεπε να φτιάξουμε ένα client για το Google's Maps API από μόνοι μας?Αυτο προσπάθησα να κάνω στο επόμενο πρόγραμμα search2.py.

Πρόγραμμα 1-2 Ανάκτηση ενός εγγράφου JSON από το Google Maps URL

```
#!/usr/bin/env python
import urllib, urllib2
try:
» import json
except ImportError: # for Python 2.5
» import simplejson as json
params = {'q': '207 N. Defiance St, Archbold, OH',
» » 'output': 'json', 'oe': 'utf8'}
url = 'http://maps.google.com/maps/geo?' + urllib.urlencode(params)
rawreply = urllib2.urlopen(url).read()
reply = json.loads(rawreply)
print reply['Placemark'][0]['Point']['coordinates'][:-1]
```

Τρέχοντας το παράπανω πρόγραμμα δίνει παρόμοια απάντηση με το προηγούμενο

```
$ python search2.py
[-84.3063479, 41.5228242]
```

Λοιπόν, η έξοδος δεν είναι ακριβώς η ιδία -μπορούμε να δούμε, για παράδειγμα, ότι το πρωτόκολλο JSON δεν κάνει διάκριση ανάμεσα σε μια πλειάδα και μια λίστα, και επίσης ότι η Google στέλνει πίσω το γεωγραφικό μήκος και πλάτος σε την αντίστροφη σειρά από αυτήν που η module googlemaps έστελνε. Όμως, είναι σαφές ότι αυτό το script πέτυχε να μοιάζει πολύ στο πρώτο.
Στο search2.py,έχουμε ενισχύσει τοκ κώδικα,αντί να χρησιμοποιήσουμε ένα οποιοδήποτε third-party πακέτο από όλα,καλούμε ρουτίνες από την Python's built-in Standard βιβλιοθήκη.Αυτός ο κώδικας λειτουργεί μόνο στην έκδοση Python 2.6 ή παραπάνω,εκτός και αν χρησιμοποιήσουμε την εντολή pip για να εγκαταστήσουμε το third-party simplejson πακέτο.

Λοιπόν, το δεύτερο πρόγραμμα μας δημιουργεί μια διεύθυνση URL και προσκομίζει το έγγραφο που αντιστοιχεί σε αυτό. Η λειτουργία ακούγεται αρκετά απλή, φυσικά ο web browser σας εργάζεται πολύ σκληρά για να φανεί αρκετά στοιχειώδη. Αλλά ο πραγματικός λόγος που μια διεύθυνση URL μπορεί να χρησιμοποιηθεί για να φέρω ένα έγγραφο, φυσικά, είναι ότι η διεύθυνση URL είναι αυτό που περιγράφει το πού θα βρω και πώς θα φέρω-συγκεκριμένο εγγράφο από τον παγκόσμιο ιστό. Το URL αποτελείται από το όνομα ενός πρωτοκόλλου, ακολουθούμενο από το όνομα του μηχανήματος, όπου υπάρχει το έγγραφο, τελειώνοντας με τη διαδρομή του εγγράφου στο μηχάνημα. Αυτό χαμηλότερου επιπέδου πρωτόκολλο που το URL χρησιμοποιεί , στην πραγματικότητα, είναι το πρωτόκολλο Hypertext Transfer Protocol, ή HTTP,η οποία είναι η βάση σχεδόν όλων των σύγχρονων επικοινωνιών web.

Πρόγραμμα 1.3 Κάνοντας μία Raw HTTP Connection to Google Maps

```
#!/usr/bin/env python
import httplib
try:
» import json
except ImportError: # for Python 2.5
» import simplejson as json
path = ('/maps/geo?q=207+N.+Defiance+St%2C+Archbold%2C+OH'
» » '&output=json&oe=utf8')
connection = httplib.HTTPConnection('maps.google.com')
connection.request('GET', path)
rawreply = connection.getresponse().read()

reply = json.loads(rawreply)
print reply['Placemark'][0]['Point']['coordinates'][:-1]
```

Στο πρόγραμμα αυτο,δεν υπάρχουν αναφορές βασισμένες πάνω στο URL-στην πραγματικότητα, καμία από της Python URLrelated βιβλιοθήκες της εισάγετε! Αντ 'αυτού, χειριζόμαστε άμεσα το πρωτόκολλο HTTP: ζητώντας να συνδεθεί με ένα συγκεκριμένο μηχάνημα, να εκδώσει μια αίτηση GET με μια διαδρομή που έχουμε κατασκευάσει, και
τελικά να διαβάσει την απάντηση απευθείας από τη σύνδεση HTTP. Αντί να παρέχετε το ερώτημα παραμέτρων μας,πρέπει να τους ενσωματώσουμε άμεσα, με το χέρι, στη διαδρομή που ζητάμε χρησιμοποιώντας ένα ερωτηματικό (?), ακολουθούμενο από το παραμέτρων στο όνομα μορφή = αξία και όλες οι χωρίζονται από & χαρακτήρες.

Παρόλα αυτά,το αποτέλεσμα είναι το ίδιο με αυτό που είδαμε προηγουμένως :

```
$ python search3.py
[-84.3063479, 41.5228242]
```

Συμπέρασμα

Κατ 'αρχάς, τώρα ίσως αποσαφηνίζετε ο όρος protocol stack: αυτό σημαίνει οικοδόμηση μιας υψηλού επιπέδου, σημασιολογικά εξελιγμένης συνομιλίας ,δηλαδή -«Θέλω τη γεωγραφική θέση της ταχυδρομικής διεύθυνσης "-η οποία είναι από τις πιο απλές κ υποτυπώδες συνομιλίες που απλά στέλνει κ λαμβάνει συμβολοσειρές κειμένου ανάμεσα σε δυο υπολογιστές χρησιμοποιώντας το υλικο του δικτύου.Το protocol stack που ανέφερα είναι τέσσερα πρωτόκολλα πάνω:

- Google Maps URLs επιστρέφουν JSON δεδομένα που περιέχουν συντεταγμένες.
- URLs έγγραφα μπορούν να ανακτηθούν χρησιμοποιώντας HTTP.
- HTTP χρησιμοποιούν sockets για να υποστηρίξουν εντολές εγγράφου πχ την GET
- Sockets γνωρίζουν μόνο πώς να στέλνουν κ να δέχονται κείμενο.

Κάθε επίπεδο του σωρού χρησιμοποιεί εργαλεία που παρέχονται από το χαμηλότερο επίπεδο και με τη σειρά του πρόσφερεί ικανοτητές για το παραπάνς επίπεδο.

Ένα δεύτερο σημείο που πρέπει να τονιστεί είναι πόσο ολοκληρωμένη υποστήριξη παρέχει η Python για κάθε ένα από τα επίπεδα δικτυού τα οποία έχουμε αναφέρει. Μόνο χρησιμοποιώντας ένα vendor-specific πρωτόκολλο, και ζητώντας απαιτήσεις που έχουν να κάνουν με τη μορφή η Google θα μπορεί να τις καταλάβει. Κάθε ένα από τα άλλα επίπεδα πρωτόκολλο που αντιμετωπίσαμε ήδη είχε την ισχυρή υποστήριξη στο εσωτερικό της Βιβλιοθήκης Python Standard. Είτε θέλουμε να βάλουμε ένα έγγραφο σε μια συγκεκριμένη διεύθυνση URL, ή να στείλετε και να λάβετε συμβολοσειρές σε ένα raw network socket, η Python ήταν έτοιμο με λειτουργίες και classes που θα μπορούσαμε να χρησιμοποιήσουμε για να ολοκληρώσουμε αυτό που θέουμε να κάνουμε.

ΚΕΦΑΛΑΙΟ 3^{Ο}

IPv4/IPv6

The Internet Protocol

Και οι δύο, η δικτύωση, η οποία εμφανίζεται όταν συνδέετε διάφορους υπολογιστές έτσι ώστε να μπορούν να επικοινωνούν, και η διαδικτύωση, η οποία συνδέει παρακείμενα δίκτυα μαζί για να σχηματίσουν ένα πολύ μεγαλύτερο σύστημα όπως το Internet, είναι ουσιαστικά μόνο περίτεχνα σχήματα για να καταστεί δυνατή η κατανομή των πόρων.

Όλα τα είδη των πραγμάτων σε έναν υπολογιστή, φυσικά, πρέπει να είναι κοινά: δίσκοι, μνήμη, και η CPU είναι όλα προσεκτικά φυλαγμένα από το λειτουργικό σύστημα έτσι ώστε τα επιμέρους προγράμματα που εκτελούνται στον υπολογιστή σας να μπορούν να έχουν πρόσβαση αυτούς τους πόρους χωρίς να εμποδίζουν κάποιο άλλο πρόγραμμα. Το δίκτυο είναι ένας ακόμη πόρος,το οποίο το λειτουργικό σύστημα πρέπει να προστατεύσει, έτσι ώστε τα προγράμματα να μπορούν να επικοινωνούν μεταξύ τους χωρίς παρεμβολές με άλλες συζητήσεις που τυχαίνει να συμβαίνουν στο ίδιο δίκτυο.

Οι φυσικές συσκευές δικτύωσης που χρησιμοποιεί ο υπολογιστής σας να επικοινωνήσει-όπως κάρτες Ethernet, wireless transmitters, και θύρες USB-είναι όλα σχεδιασμένα με ένα πολύπλοκη δυνατότητα να μοιράζονται ένα φυσικό μέσο μεταξύ πολλών διαφορετικών συσκευών που θέλουν να επικοινωνήσουν. Μια ντουζίνα Ethernet καρτών μπορεί να συνδεθεί στον ίδιο διανομέα, τριάντα ασύρματες κάρτες μπορεί να μοιράζονται το ίδιο radio
κανάλι,ένα DSL μόντεμ να χρησιμοποιεί frequency -domain πολυπλεξία.

Η θεμελιώδης μονάδα κατανομής μεταξύ των συσκευών δικτύου –το νόμισμα, αν θέλετε, στην οποία εμποπεύονται-είναι το "πακέτο". Ένα πακέτο είναι μια δυαδική συμβολοσειρά των οποίων το μήκος μπορεί να κυμαίνεται από μερικά bytes σε λίγα χιλιάδες bytes,το οποίο μεταδίδεται ως μια ενιαία μονάδα μεταξύ των συσκευών του δικτύου. Αν και υπάρχουν κάποα εξειδικευμένα δίκτυα σήμερα, ειδικά στα πεδία όπως οι τηλεπικοινωνίες, όπου κάθε byte κατεβαίνει μια γραμμή μεταφοράς μπορεί να είναι χωριστα δρομολογημένο σε διαφορετικό προορισμό, οι περισσότερες τεχνολογίες που χρησιμοποιούνται για την κατασκευή ψηφιακών δικτύων για όλους τους σύγχρονους υπολογιστές είναι με βάση την μεγαλύτερη μονάδα του πακέτου.

Ένα πακέτο έχει συχνά μόνο δύο ιδιότητες σε φυσικό επίπεδο: η δυαδική συμβολοσειρά που είναι τα στοιχεία που μεταφέρει, και μια διεύθυνση στην οποία πρόκειται να παραδοθεί. Η διεύθυνση είναι συνήθως ένα μοναδικό αναγνωριστικό που ονοματίζει μία από τις άλλες κάρτες δικτύου-και έτσι ο υπολογιστής πίσω από αυτη-που συνδεμένος στο ίδιο Ethernet τμήμα ή ασύρματο κανάλι όπως ο υπολογιστής που μεταφέρει το πακέτο.Η δουλεία της κάρτας δικτύου είναι να στέλνει

και να λαμβάνει τέτοια πακέτα χωρίς να χρειάζετε να γνωρίζει λεπτομέρειες του συστήματος για το πώς το δίκτυο λειτουργεί κάτω στο επίπεδο των καλωδίων και των τάσεων.Τελικά τι είναι το Internet Protocol? Το Internet Protocol είναι ένα σχήμα για την επιβολή ενός ενιαίου συστήματος διευθύνσεων σε όλους τους υπολογιστές που είναι συνδεδεμένη στο Internet σε ολόκληρο τον κόσμο, και για να καταστεί δυνατό για τα πακέτα να ταξιδέψουν από το ένα άκρο του το Διαδίκτυο για στο άλλο. Στην ιδανική περίπτωση, μια εφαρμογή όπως ο web browser σας θα πρέπει να είναι σε θέση να συνδέετε σε ένα host οπουδήποτε, χωρίς καν να ξέρει τις συσκευές δικτύου από τις οποίες κάθε πακέτο που διέρχετε στο ταξίδι του. Είναι πολύ σπάνιο για ένα πρόγραμμα σε Python να λειτουργεί σε τόσο χαμηλό επίπεδο ώστε να βλέπει το Πρωτόκολλο Internet το ίδιο σε δράση, αλλά σε πολλές περιπτώσεις, είναι χρήσιμο να γνωρίζουμε τουλάχιστον πώς λειτουργεί.

IPv4 Addresses

Το πρωτόκολλο διαδικτύου εκχωρεί μία 4-byte διεύθυνση σε κάθε υπολογιστή που είναι συνδεδεμένος με το δίκτυο.Τέτοιες διευθύνσεις γράφονται συνήθως ως τεσσάρων δεκαδικοί αριθμοί, που χωρίζονται από τελείες, οι οποίες αντιπροσωπεύουν κάθε μόνο byte της διεύθυνσης. Κάθε αριθμός μπορεί επομένως κυμαίνετε από 0 έως 255. Έτσι, μια διεύθυνση IP μοιάζει με αυτήν:

130.207.244.244

Επειδή μία αριθμητικά διευθύνση μπορεί να είναι δύσκολο για τους ανθρώπους να το θυμούνται, τα άτομα που χρησιμοποιούν το Διαδίκτυο βλέπουν γενικά hostnames και όχι διευθύνσεις IP. Ο χρήστης απλά πληκτρολογεί google.com και πίσω από αυτό υπάρχει μια διεύθυνση, η 74.125.67.103, στην οποία οι υπολογιστές τους μπορούν να διευθυσιδοτουν πακέτα για μετάδοση μέσω του Διαδικτυου. Στο getname.py μπορείτε να δείτε ένα πολύ απλό πρόγραμμα Python που ζητά από το λειτουργικό σύστημα Linux,Mac OS, Windows, ή με όποιο σύστημα το πρόγραμμα λειτουργεί -να επιλύσει το hostname google.com.Η συγκεκριμένη υπηρεσία δικτύου, η οποία ονομάζεται «Domain Name Service " και έρχετε να απαντήσει στο ερωτήμα hostname.

Προγραμμα 2.1 Μετατρέποντας ένα Hostname σε μία διευθυνση IP

```
#!/usr/bin/env python
#  getname.py
import socket
hostname = 'maps.google.com'
addr = socket.gethostbyname(hostname)
print 'The address of', hostname, 'is', addr
```

Για τώρα πρέπει να γνωρίζουμε δύο πράγματα:

- Πρώτον,το πρωτόκολλο Ιντερνετ χρησιμοποιεί μια 4-byte διευθυνση για να στείλει τα πακέτα στον προορισμό τους
- Οι πολύπλοκες λεπτομέρειες το πώς δηλαδή τα hostnames μετατρεπονται σε IP διευθυνσεις αντιμετωπίζονται συνήθως από το λειτουργικό σύστημα

Όπως και οι περισσότερες λεπτομέρειες της λειτουργίας του πρωτοκόλλου Internet, το λειτουργικό σας σύστημα κρύβει τις λεπτομέρειες τόσο από εσάς και Python κώδικα.
Στην πραγματικότητα, η διευθυνσηοδοτηση μπορεί να είναι λίγο πιο περίπλοκη αυτές τις μέρες από το απλό σχήμα που περιγράφηκε. Επειδή ο κόσμος έχει αρχίσει να εξαντλεί τις 4-byte διευθύνσεις IP, μία εκτεταμένη διεύθυνση συστήματος, με την ονομασία IPv6, αναπτύχθηκεκαι επιτρέπει 16-byte διευθύνσεις,με τις οποίες θα πρέπει να
εξυπηρετήσει τις ανάγκες της ανθρωπότητας για πολύ μεγάλο χρονικό διάστημα. Είναι γραμμένη με διαφορετικό τρόπο από την IP 4-byte διευθύνσεις, και μοιάζει με αυτό:
fe80::fcfd:4aff:fecf:ea4e

Αλλά όσο ο κώδικας επιτρέπει τις IP διευθύνσεις ή τα hostnames από το χρήστη και τα περνά απευθείας σε μλια διαδικτυακή βιβλιοθήκη για επεξεργασία,πιθανόν κανένας χρήστης δεν πρόκειται να καταλάβει την διαφορά ανάμεσσα στην IPv4 και IPv6.Το λειτουργικό σύστημα στο οποίο ο κώδικας Python τρέχει,γνωρίζει ποια έκδοση IP χρησιμοποιεί και θα πρέπει να το μεταφράσει αντίστοιχα.
Σε γενικές γραμμές, οι διευθύνσεις IP μπορεί να διαβαστεί από τα αριστερά προς τα δεξιά: η πρώτη ενός ή δύο bytes καθορίσετε μια
οργάνωση, και στη συνέχεια το επόμενο byte καθορίζει συχνά το συγκεκριμένο δευτερεύον δίκτυο στο οποίο το μηχάνημα-στόχο
κατοικεί. Το τελευταίο byte στενεύει κάτω τη διεύθυνση στο συγκεκριμένο μηχάνημα ή της υπηρεσίας. Υπάρχουν επίσης μερικές
ειδικές σειρές διεύθυνση IP που έχουν ιδιαίτερη σημασία:

- *127.*.*.**:IP διευθύνσεις που ξεκινούν με το byte 127 είναι μια ειδική διεύθυνση, που προορίζεται το εύρος που δείχνει από το τοπικό έως το μηχάνημα στο οποίο η εφαρμογή τρέχει.Όταν το πρόγραμμα περιήγησής σας, ή ο FTP client, ή το Python πρόγραμμα συνδέεται με μια διεύθυνση σε αυτό το περιοχή, είναι ζητήτε να μιλήσει σε κάποια άλλη υπηρεσία ή πρόγραμμα που εκτελείται στο ίδιο μηχάνημα. Οι περισσότερες μηχανές χρησιμοποιούν μόνο μία διεύθυνση από αυτην την περιοχή: η διεύθυνση IP 127.0.0.1 χρησιμοποιείται παγκοσμίως για να σημάνει "αυτό είναι το μηχάνημα που το πρόγραμμα εκτελειται" και μπορεί συχνά να είναι προσβάσιμη μέσω του localhost όνομα του κεντρικού υπολογιστή.
- *10.*.*.*, 172.16–31.*.*, 192.168.*.**:: Αυτές οι διευθύνσεις IP είναι προορισμένες ονομάζονται private subnets. Οι αρχές που διοικούν το Διαδίκτυο έχουν κάνει μια συμφωνία: ποτέ δεν θα δοθούν διευθύνσεις IP σε οποιαδήποτε από αυτές τις τρεις περιοχές σε εταιρείες που στήνουν servers ή προσφέρουν υπηρεσίες.Στο Διαδίκτυο, αυτές οι διευθύνσεις δεν έχουν κανένα νόημα,η διεύθυση αυτή δηλαδή δεν υπάρχει κάποιος κεντρικός υπολογιστής στον οποίο να αντιστοιχεί για να συνδεθείτε. Ως εκ τούτου, αυτές

οι διευθύνσεις είναι ελεύθερες για να το χρησιμοποιηθούν σε οποιοδήποτε από τα εσωτερικά δίκτυα ενός οργανισμού και είναι ελεύθεροι να εκχωρούν διευθύνσεις IP στο εσωτερικό του , αλλά των οποίων οι κεντρικοί υπολογιστές δεν χρειάζεται να προσπελάσιμοι από το διαδίκτυο.

Έτσι, τα λειτουργικά συστήματα που εφαρμόζουν το πρωτόκολλο του Internet επιτρέπουν στα προγράμματα να στέλνουν μηνύματα των οποίων οι διευθύνσεις προορισμοί IP-είναι ας πούμε, 8.8.4.4, για να παραδώσει κάθε πακέτο, το λειτουργικό σύστημα πρέπει να αποφασίσει πώς να το μεταδώσει χρησιμοποιώντας ένα από τα φυσικά δίκτυα με τα οποία η συσκευή είναι συνδεδεμένη. Αυτή η λήψη της απόφασης για το πού πρέπει να στείλει κάθε πακέτο, με βάση την διεύθυνση IP που έχει ο προορισμός-ονομάζεται δρομολόγηση.

- Αν η IP διεύθυνση είναι του τύπου 127.*.*.*,τότε το λειτουργίκο σύστημα γνωρίζει ότι το πακέτο προορίζετε για άλλη εφαρμογή που εκτελείται στο ίδιο μηχάνημα.
- Αν η IP διεύθυνση είναι στο ίδιο υποδίκτυο με το μηχάνημα,τότε ο προορισμός μπορεί να βρεθεί εύκολα από ένα απλό έλεγχο στο τοπικό Ethernet κομμάτι,ασύρματο κανάλι,ή οτιδήποτε στο τοπικό δίκτυο και στέλνει το πακέτο στη τοπικά συνδεδεμένο μηχάνημα.
- Αλλιώς,το μηχάνημα προωθεί το πακέτο σε ένα *gateway* μηχάνημα το οποίο συνδέει το τοπικό υποδίκτυο με το υπόλοιπο Ιντερνετ.Τότε αποφασίζει το *gateway* που θα στείλει το πακέτο μετά.

Φυσικά, η δρομολόγηση είναι απλή όσο αφορά τους σταθμούς που συναντάει στο δρόμο γιατί απλά έχει να αποφασίσει αν θα κρατήσει το πακέτο στο τοπικό δίκτυο ή θα πρέπει να το προωθήσει για να φτάσει στον προορισμό του.Το δύσκολο κομμάτι είναι στους κόμβους που συνδέουν ολόκληρες ηπείρους, οπότε πίνακες δρομολόγησης πρέπει να κατασκευάζονται, οι οποίοι πρέπει να ενημερώνονται συνεχώς, ώστε να γνωρίζουν ότι τα πακέτα που προορίζονται για το Google πρέπει να πάνε σε μια κατεύθυνση, αλλά τα πακέτα που απευθύνονται σε μια διεύθυνση IP Yahoo πρέπει να πάνε σε άλλη. Αλλά είναι πολύ σπάνιο για Python εφαρμογές να τρέχουν σε δρομολογητές κορμού Internet.

Μια τελευταία έννοια του πρωτοκόλλου Internet που αξίζει να αναφέρθει είναι ο κατακερματισμός πακέτων. Ενώ υποτίθεται ότι να είναι μια σκοτεινή λεπτομέρεια που είναι κρυμμένη με επιτυχία από το πρόγραμμά σας από τη στοίβα δικτύου του λειτουργικού συστήματος και έχει προκαλέσει αρκετά προβλήματα στην ιστορία του Διαδικτύου .Ο κατακερματισμός είναι απαραίτητος επειδή το Internet Protocol υποστηρίζει πολύ μεγάλα πακέτα-μπορούν να είναι πάνω από 64 KB σε μήκος, αλλά οι πραγματικές συσκευές δικτύου από το οποίο κατασκευάζονται τα δίκτυα IP συνήθως υποστηρίζουν πολύ μικρότερα μεγέθη πακέτων. Τα δίκτυα Ethernet, για παράδειγμα,υποστηρίζουν μόνο 1.500 byte πακέτα. Τα πακέτα, επιπλέον ,περιέχουν μία σημαία "don’t fragment" (DF) με την οποία ο αποστολέας μπορεί να επιλέξει ό, τι πρέπει να συμβεί εάν το πακέτο αποδειχτεί πολύ μικρό για να χωρέσει σε ένα από τα φυσικά δίκτυα που βρίσκεται μεταξύ των υπολογιστή προέλευσης και τον προορισμό:

- Αν η σημαία DF δεν είναι ορισμένη, τότε ο κατακερματισμός επιτρέπεται, και όταν το πακέτο φθάσει στο ανώτατο όριο του δικτύου πάνω στο οποίο δεν μπορούν να χωρέσει, η gateway μπορεί να το χωρίσει σε μικρότερα πακέτα και να τα μαρκάρει με νούμερα για ενωθούν στο άλλο άκρο.
- Αν η σημαία DF έχει οριστεί, τότε ο κατακερματισμός απαγορεύεται, και αν το πακέτο δεν μπορεί να χωρέσει,τότε απορρίπτεται και ένα μήνυμα σφάλματος στέλνετε πίσω σε ένα ειδικό πακέτο σηματοδότησης που ονομάζεται " Internet Control Message Protocol" (ICMP) –έτσι ώστε το μηχάνημα που στέλνει το πακέτο, να δοκιμάσει διαχωρισμό του μηνύματος σε μικρότερα κομμάτια και εκ νέου αποστολή.

Τα Python προγράμματά σας δεν θα έχουν συνήθως κανένα έλεγχο επί τη σημαία DF,Αντ 'αυτού, ορίζεται από το λειτουργικό συστήμα. Χονδρικά, η λογική που ακολουθούν τα συστήματα είναι συνήθως η εξής: αν έχετε μία UDP συνομιλίας που αποτελείται από μεμονωμένα datagrams,τα οποία υπάρχουν σε ολόκληρο το Διαδίκτυο,τότε το λειτουργικό σύστημα θα αφήσει DF unset έτσι ώστε κάθε datagram φτάνει στον προορισμό σε όσα κομμάτια μπορεί να χρειαστούν, αλλά σε μία συνομιλία TCP οπού η ροή των δεδομένών μπορεί να είναι εκατοντάδες ή χιλιάδες πακέτα, τότε το λειτουργικό σύστημα θα θέσει τη σημαία DF έτσι ώστε να μπορεί να επιλέξει ακριβώς το σωστό μέγεθος του πακέτου για να πραγματοποιείτε η ροή της συνομιλίας ομαλά, χωρίς τα πακέτα της να κατακερματίζονται καθ 'οδόν, γεγονός που καθιστά τη συνομιλία λιγότερο αποτελεσματική.

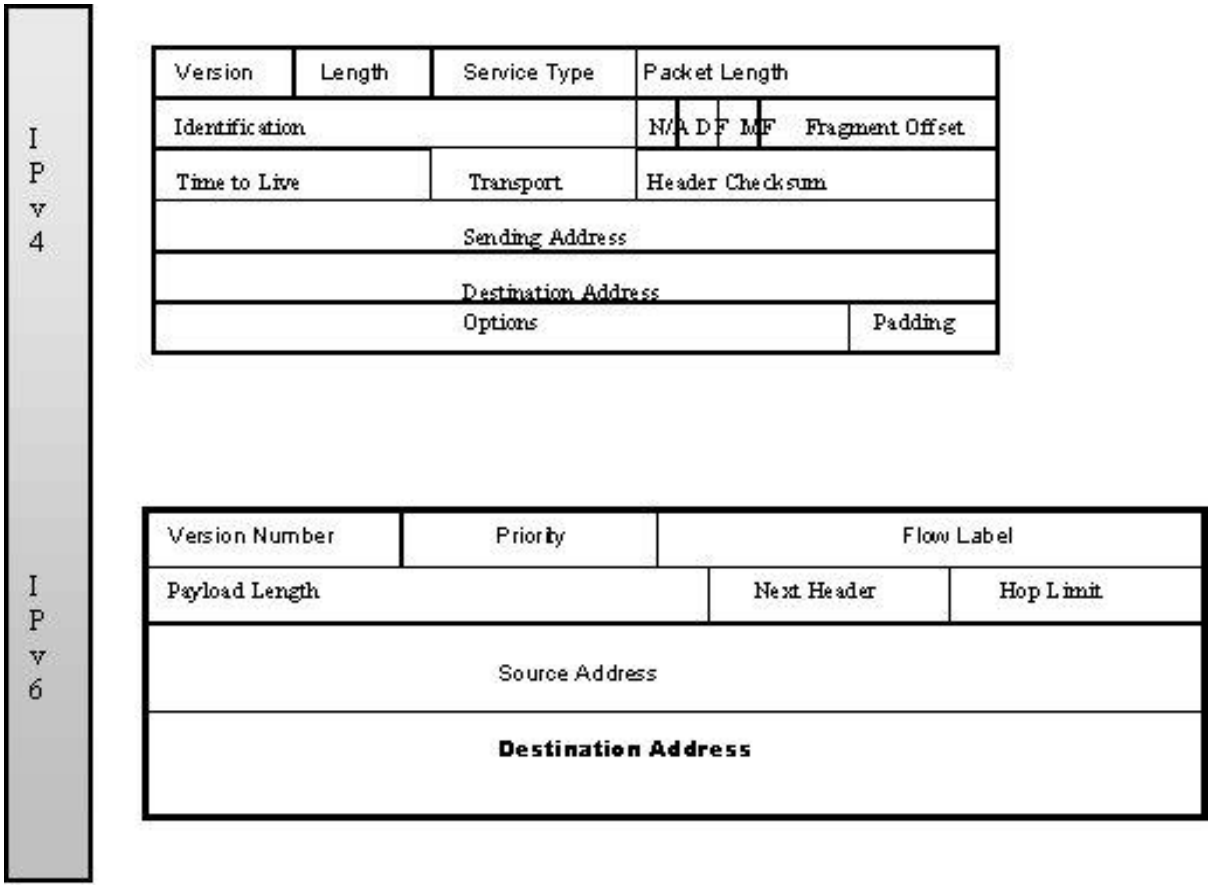
Το μεγαλύτερο πακέτο το οποίο ένα υποδίκτυο στο Internet μπορεί να δεχθεί αυτή ονομάζεται " maximum transmission unit» (MTU. Επιστροφή στη δεκαετία του 1990, οι πάροχοι υπηρεσιών Διαδικτύου (κυρίως τηλεφωνικές εταιρείες που προσφέρουν DSL συνδέσεις) άρχισαν να χρησιμοποιούν το PPPoE, ένα πρωτόκολλο που βάζει IP πακέτα μέσα σε μια κάψουλα που τους αφήνει χώρο για μόνο 1.492 bytes αντί του πλήρους 1.500 byte που συνήθως επιτρέπεται σε στην Ethernet. Πολλά sites στο Διαδίκτυο ήταν απροετοίμαστα για αυτό, επειδή χρησιμοποίησαν 1.500 byte πακέτα από προεπιλογή και είχε αποκλείσει κάθε ICMP πακέτο. Κατά συνέπεια, οι διακομιστές τους δεν θα μπορούσαν να λαμβάνουν τα ICMP λάθη για να μάθουν ότι είναι αρκετά μεγάλα κ έχουν την σημαία, “don’t fragment”.

Σήμερα το πρόβλημα αυτό σπάνια συναντάται, αλλά δείχνει πόσο ένα χαμηλό επίπεδου χαρακτηριστικό IP μπορεί να δημιουργήσει σε ένα χρήστης-ορατά συμπτώματα-και, ως εκ τούτου, γι 'αυτό είναι καλό για να κρατήσει όλα τα χαρακτηριστικά του IP στο μυαλό του κατά την γραφή και κατά τον εντοπισμού σφαλμάτων των προγραμμάτων δικτύου.

IPv6

Στις αρχές της δεκαετίας του 1990, το Internet Engineering Task Force ξεκίνησε μια προσπάθεια για την ανάπτυξη μιας διαδοχής του πρωτόκολλου IPv4. Ένα πρωταρχικό κίνητρο για αυτή την προσπάθεια ήταν η πραγματοποίηση 32-bit χώρου διευθύνσεων IP άρχιζε να εξαντλείται , με νέα υποδίκτυα και IP κόμβους να συνδέονται στο Διαδίκτυο (και την ανάθεση της μοναδική διεύθυνση IP), με ένα εκπληκτική ρυθμό. Για να ανταποκριθεί σε αυτή την ανάγκη για ένα μεγαλύτερο χώρο διευθύνσεων IP, ένα νέο IP πρωτόκολλο χρειάζονταν, έτσι το IPv6 αναπτύχθηκε. Οι σχεδιαστές του IPv6 εκμεταλλεύτηκαν επίσης την ευκαιρία να βελτιώσουν και άλλες πτυχές του IPv4 που υπήρχε πρόβλημα, με βάση τη εμπειρία που ήδη υπήρχε με το IPv4.Οι αλλαγές που πραγματοποιήθηκαν είναι:

- Διευρυμένες δυνατότητες διευθυνσιοδότησης Το IPv6 αυξάνει το μέγεθος της διεύθυνσης IP από 32 - 128 bits. Αυτό εξασφαλίζει ότι ο κόσμος δεν θα ξεμείνει από IP διευθύνσεις. Πλέον,κάθε κόκκο άμμου στον πλανήτη μπορεί να είναι IP-addressable. Εκτός από τις unicast και multicast διευθύνσεις , το IPv6 έχει αναπτύξει ένα νέο τύπο του διεύθυνσης , που ονομάζεται aoycast διεύθυνση, η οποία επιτρέπει σε ένα datagram να μπορεί να διανεμηθεί σε οποιονδήποτε από μια ομάδα hosts.
- A streamlined 40-byte header Όπως αναλύεται παρακάτω, μια σειρά από τομείς έχουν IPv4 δεν υπάρχουν ή καθίστανται προαιρετικά. Η προκύπτουσα 40-byte σταθερού μήκους header επιτρέπει ταχύτερη επεξεργασία των IP datagram. Μια νέα κωδικοποίηση των επιλογών επιτρέπει πιο ευέλικτες επιλογές επεξεργασίας.
- Flow labeling and priority Το IPv6 είναι ένας αφαιρετικός ορισμός της ροής. Τα RFC 1752 και RFC 2460 δηλώνουν ότι αυτό επιτρέπει «την επισήμανση των πακέτων που ανήκουν σε ιδιαίτερες ροές για τις ο αποστολέας ζητήσει ειδική μεταχείριση, όπως η μη προεπιλεγμένη ποιότητα της υπηρεσίας ή σε πραγματικό χρόνο εξυπηρέτηση. "Για παράδειγμα, μετάδοση ήχου και εικόνας θα μπορούσε να πιθανό να αντιμετωπίζεται ως μια ροή. Από την άλλη πλευρά, οι πιο παραδοσιακές εφαρμογές,όπως η μεταφορά αρχείων και e-mail, ίσως να μην αντιμετωπίζονται ως ροές. Είναι πιθανό ότι η κίνηση μεταφέρεται από ένα υψηλής προτεραιότητας χρήστη (για παράδειγμα, κάποιος πληρώνει για την καλύτερη εξυπηρέτηση για την κυκλοφορία τους) θα μπορούσε επίσης να θεωρηθεί ως μια ροή. Αυτό που είναι σαφές, ωστόσο, είναι ότι οι σχεδιαστές του IPv6 προβλέπουν την ενδεχόμενη ανάγκη να είναι σε θέση να διαφοροποιήσει μεταξύ ροών, ακόμη και αν η ακριβής έννοια της ροής δεν έχει ακόμη προσδιοριστεί. Ο IPv6 header έχει επίσης ένα 8-bit πεδίο κίνησης κλάσης . Αυτό το πεδίο, όπως το πεδίο TOS στην IPv4, μπορεί να χρησιμοποιηθεί για να δώσει προτεραιότητα σε ορισμένα datagrams μέσα σε μια ροή, ή μπορεί να είναι χρησιμοποιείται για να δώσει προτεραιότητα σε datagrams από ορισμένες εφαρμογές (για παράδειγμα, ICMP)πάνω από dalagrams από άλλες εφαρμογές (για παράδειγμα, ειδήσεις δικτύου).



Εικόνα 2 Διαφορές IPv4,IPv6

Αρχικά,με ένα πολύ εύκολα μπορούμε να δούμε αν το σύστημα μας υποστηρίζει IPv6,εκτελώντας την bolean εντολή has_ipv6 μέσα από ένα socket

```
>>> import socket
>>> socket.has_ipv6
True
```

Αλλά σημειώστε ότι αυτό δεν θα σας πει αν είναι μια πραγματική IPv6 διασύνδεση και αν έχει ρυθμιστεί και αν μπορεί επί του παρόντος να χρησιμοποιηθεί για να στείλει πακέτα κάπου, θα είναι καθαρά ένας ισχυρισμός σχετικά με το αν έχει την υποστήριξη του IPv6
έχουν συνταχθεί στο λειτουργικό σύστημα, όχι για το αν είναι σε χρήση!

Παρακάτω υπάρχουν 4 παραδείγματα προγράμματα που χρησιμοποιούν το TCP/IP πρώτοκολλο: ένας διακομιστής που απηχεί όλα τα δεδομένα που λαμβάνει πίσω (εξυπηρετώντας μόνο έναν πελάτη), και ένας πελάτης τα χρησιμοποιεί. Σημειώστε ότι ένας διακομιστής πρέπει να εκτελέσει την σειρά socket(), bind(), listen(), accept() (ενδεχομένως επαναλαμβάνοντας τηω accept () για να εξυπηρετεί περισσότερους από έναν πελάτες), ενώ ο πελάτης δεν χρειάζεται παρά την σειρά socket (), connect (). Επίσης σημειώστε ότι ο διακομιστής δεν send()/recv() στο socket που ακούει στο, αλλά για το νέο socket που επιστρέφεται από το accept ().

Τα πρώτα δύο παραδείγματα υποστηρίζουν μόνο IPv4:

```
# Echo server program
import socket

HOST = ''                 # Symbolic name meaning all available interfaces
PORT = 50007              # Arbitrary non-privileged port
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind((HOST, PORT))
s.listen(1)
conn, addr = s.accept()
print 'Connected by', addr
while 1:
    data = conn.recv(1024)
    if not data: break
    conn.send(data)
conn.close()
```

```
# Echo client program
import socket

HOST = 'daring.cwi.nl'    # The remote host
PORT = 50007              # The same port as used by the server
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((HOST, PORT))
s.send('Hello, world')
data = s.recv(1024)
s.close()
print 'Received', repr(data)
```

Τα επόμενα δύο παραδείγματα είναι πανομοιότυπα με τις δύο παραπάνω, αλλά υποστηρίζουν IPv4 και IPv6. Η πλευρά του διακομιστή θα ακούει στην πρώτη διαθέσιμη οικογένεια διευθύνσεων (θα πρέπει να ακούσουν και τις δύο αντ 'αυτού). Στα περισσότερα του IPv6 έτοιμα συστήματα, το IPv6 θα έχει προτεραιότητα και ο server μπορεί να μην δεχθεί την κυκλοφορία IPv4. Η πλευρά του client θα προσπαθήσει να συνδεθεί με όλες τις επιστρεφόμενες διευθύνσεις ως αποτέλεσμα την επίλυση ονομάτων, και στέλνει την κυκλοφορία με την πρώτη συνδεδεμένη με επιτυχία.

```
# Echo server program
import socket
import sys

HOST = None               # Symbolic name meaning all available interfaces
PORT = 50007              # Arbitrary non-privileged port
s = None
for res in socket.getaddrinfo(HOST, PORT, socket.AF_UNSPEC,
```

```
                              socket.SOCK_STREAM, 0, socket.AI_PASSIVE):
    af, socktype, proto, canonname, sa = res
    try:
        s = socket.socket(af, socktype, proto)
    except socket.error, msg:
        s = None
        continue
    try:
        s.bind(sa)
        s.listen(1)
    except socket.error, msg:
        s.close()
        s = None
        continue
    break
if s is None:
    print 'could not open socket'
    sys.exit(1)
conn, addr = s.accept()
print 'Connected by', addr
while 1:
    data = conn.recv(1024)
    if not data: break
    conn.send(data)
conn.close()
```

```
# Echo client program
import socket
import sys

HOST = 'daring.cwi.nl'    # The remote host
PORT = 50007              # The same port as used by the server
s = None
for res in socket.getaddrinfo(HOST, PORT, socket.AF_UNSPEC,
socket.SOCK_STREAM):
    af, socktype, proto, canonname, sa = res
    try:
        s = socket.socket(af, socktype, proto)
    except socket.error, msg:
        s = None
        continue
    try:
        s.connect(sa)
    except socket.error, msg:
        s.close()
        s = None
        continue
```

```
    break
if s is None:
    print 'could not open socket'
    sys.exit(1)
s.send('Hello, world')
data = s.recv(1024)
s.close()
print 'Received', repr(data)
```

ΚΕΦΑΛΑΙΟ 4Ο

UDP

Μέχρι τώρα,ότι έχουμε δει υποστηρίζει ότι όλες οι επικοινωνίες δικτύου είναι αυτές τις μέρες χτισμένες με τη μετάδοση των σύντομων μηνυμάτων που ονομάζονται πακέτα που συνήθως δεν ξεπερνούν τις μερικές χιλιάδες bytes.Τα πακέτα ταξιδεύουν με τον τρόπο τους σε όλο το δίκτυο ανεξάρτητα, ελεύθερα να λάβουν διαφορετικά μονοπάτια προς τον ίδιο προορισμό.Αυτό σημαίνει ότι τα πακέτα μπορεί να φτάσουν σε διαφορετική σειρά. Εάν οι συνθήκες του δικτύου είναι κακές , ή ένα πακέτο είναι απλά άτυχο, τότε θα μπορούσε εύκολα να μην φτάσουν ολα.

Υπάρχουν τρεις πιθανές προσεγγίσεις για την οικοδόμηση πάνω σε IP. Εδώ είναι, κατά φθίνουσα σειρά σε δημοτικότητα!

- Η συντριπτική πλειοψηφία των εφαρμογών σήμερα είναι χτισμένες πάνω στο TCP,το Transmission Control Protocol, το οποία προσφέρει διατεταγμένες και αξιόπιστες ροές δεδομένων μεταξύ IP εφαρμογές.
- Μερικά πρωτόκολλα, συνήθως με μικρά, αυτόνομα αιτήματα και απαντήσεις, και απλά clients που δεν θα ενοχληθεί αν κάποια σχετική αίτηση χαθεί και πρέπει να επαναληφθεί ,τότε να επιλέξτε UDP, δηλαδή User Datagram Protocol.
- Πολύ εξειδικευμένα πρωτόκολλα αποφεύγουν τις δύο αυτές τις επιλογές, και να επιλέγουν να δημιουργήσουν ένα εντελώς νέο πρωτόκολλο με βάση IP στο οποίο θα λειτουργούν παράλληλα με τα πρωτόκολλα TCP και UDP ως ένα εντελώς νέος τρόπος συζήτησης σε ένα δίκτυο IP.

Η τελευταία από αυτές τις τρεις επιλογές είναι πολύ σπάνια. Οι απλοί χρήστες του λειτουργικού συστήματος συνήθως δεν είναι επιτρέπεται να επικοινωνούν με το δίκτυο χωρίς να μην περάσουν από τα πρωτόκολλα TCP ή UDP, έτσι εξηγείτε το UDP όνομα: είναι ο τρόπος που κανονικοί "Χρήστες", σε αντίθεση με τους διαχειριστές του λειτουργικού συστήματος, μπορούν να στείλουν packet-based μηνύματα.

Addresses and Port Numbers

Το πρωτόκολλο IP που λαμβάνει συνήθως τη μορφή κώδικα τεσσάρων οκτάδων, όπως το 18.9.22.69-για κάθε μηχάνημα που είναι συνδεδεμένο με ένα δίκτυο IP. Στην πραγματικότητα, μια μηχανή με πολλές κάρτες δικτύου όταν συνδέετε με το δίκτυο θα έχει συνήθως μία διαφορετική διεύθυνση IP για κάθε κάρτα, έτσι ώστε οι άλλοι hosts να μπορούν να επιλέξουν το δίκτυο στο οποίο θα επικοινωνήσουμε με το μηχάνημα. Οι πολλαπλές διασυνδέσεις χρησιμοποιούνται επίσης για τη βελτίωση της redundancy και του εύρους ζώνης. Αλλά ακόμα και αν μια IP-συνδεδεμένο μηχάνημα έχει μόνο μία κάρτα δικτύου, τότε πάλι θα έχει τουλάχιστον ακόμα μια άλλη διεύθυνση του δικτύου: η διεύθυνση 127.0.0.1 είναι πώς οι μηχανές μπορούν

να συνδεθούν με τον εαυτό τους. Εξυπηρετεί ως ένα σταθερό όνομα που κάθε μηχανή έχει για τον εαυτό της.

Και αυτές οι IP διευθύνσεις παρέχουν τη δυνατότητα σε εκατομμύρια διαφορετικά μηχανήματα, χρησιμοποιώντας όλα τα είδη των διαφορετικών δικτύων υλικού, να περνούν πακέτα ο ένας στον άλλο πάνω από μια οπτική ίνα ενός δικτύου IP.

Αλλά με UDP και TCP κάνουμε τώρα ένα μεγάλο βήμα,δεν σκεφτόμαστε τις ανάγκες δρομολόγησης του δικτύου στο σύνολό του και θα αρχίσουν να μελετούν τις ανάγκες συγκεκριμένων εφαρμογών που εκτελούνται σε μία συγκεκριμένη μηχανή. Και το πρώτο πράγμα που παρατηρούμε είναι ότι έχουμε ένα μόνο σήμερα υπολογιστή στον οποίο τρέχουν δεκάδες
προγραμμάτων ανά πάσα στιγμή-και πολλοί από αυτά θέλουν να χρησιμοποιήσουν το δίκτυο την ίδια στιγμή! Μπορεί κάποιος να ελέγχει τα e-mail με τον Thunderbird, ενώ μια ιστοσελίδα κατεβαίνει στο Google Chrome, ή γίνεται εγκατάσταση ενός Python πακέτο με pip μέσω του δικτύου, καθώς ελέγχετε η κατάσταση ενός απομακρυσμένου server με SSH. Με κάποιο τρόπο, όλα αυτά οι διαφορετικές και ταυτόχρονες συνομιλίες πρέπει να λάβουν χώρα
χωρίς να παρεμβάλλονται μεταξύ τους.

Αυτό είναι ένα γενικό πρόβλημα και στα δίκτυα υπολογιστών και στην ηλεκτρομαγνητική θεωρία του σήματος. Είναι γνωστή ως η ανάγκη για πολύπλεξη: η ανάγκη για ένα μόνο κανάλι να μοιράζετε από πολλές διαφορετικές συζητήσεις. Ήταν γνωστό, ότι τα ραδιοκύματα μπορούν να διαχωριστούν το ένα από ένα άλλο, χρησιμοποιώντας διαφορετικές συχνότητες. Για να γίνει διάκριση μεταξύ των διαφόρων προορισμών σε ποιο UDP πακέτο θα πρέπει να διευθυνσιοδοτηθεί -όπου το μόνο που έχουμε για να συνεργαστούμε είναι τα αλφάβητα των συμβόλων, οι σχεδιαστές της IP επέλεξαν την rough-and-ready τεχνική σήμανσης για κάθε πακέτο UDP με ένα unsigned 16-bit αριθμό(ο οποίος είναι από 0 έως 65536), που προσδιορίζει μία πόρτα στην οποία μία εφαρμογή μπορεί να συνδεθεί και ακούει.

Για παράδειγμα,έχετε δημιουργήσει ένα DNS server σε ένα από τα μηχανήματα σας, με IP διεύθυνση 192.168.1.9. Για να επιτρέψετε σε άλλους υπολογιστές να βρουν την υπηρεσία, ο διακομιστής θα ζητήσει από το λειτουργικό συστήματος την άδεια να πάρει τον έλεγχο του πόρτας UDP με το αριθμό DNS 53.Υποθέτοντας ότι καμία διαδικασία δεν εκτελείται σε αυτή την πόρτα, ο διακομιστής DNS θα χορηγηθεί την εν λόγω πόρτα.

Στη συνέχεια,έχουμε ένα μηχάνημα-πελάτη με τη διεύθυνση IP 192.168.1.30 στο δίκτυό σας,δίνεται η IP διεύθυνση του νέου αυτού DNS server και θέλει να εκδώσει ένα ερώτημα. Θα δημιουργήσει ένα ερώτημα DNS στη μνήμη, και στη συνέχεια θα ζητήσει από το λειτουργικό σύστημα να στείλει το μπλοκ των δεδομένων, σαν ένα πακέτο UDP. Δεδομένου ότι θα χρειαστεί να υπάρχει κάποιος τρόπος για να αναγνωρίζει ο client όταν το πακέτο επιστρέφει, και αφού ο πελάτης δεν έχει ζητήσει ρητά μια πόρτα,το λειτουργικό σύστημα εκχωρεί μια τυχαία,πόρτα 44137 παράδειγμα.

Το πακέτο, επομένως ταχιδεύει το δρόμο του προς τη πόρτα 53 με ετικέτες που προσδιορίζουν την IP διεύθυνση και UDP αριθμό θύρας (εδώ διαχωρίζονται με άνω και κάτω τελεία).

192.168.1.30:44137

Και θα δώσει τον προορισμό του, όπως ακόλουθως :

192.168.1.9:53

Αυτή η διεύθυνση προορισμού, απλό κι αν φαίνεται, μόνο ο αριθμός του ηλεκτρονικού υπολογιστή, και ο αριθμός της πόρτας -είναι ότι μια IP στοίβα δίκτυου χρειάζετε για να οδηγήσει αυτό το πακέτο στον προορισμό του. Ο DNS διακομιστής θα λάβει το αίτημα από το λειτουργικό του σύστημα, μαζί με την αρχική IP και τον αριθμό της πόρτας.Από τη στιγμή που έχει διατυπώθεί μια απάντηση, ο διακομιστής DNS θα ζητήσει από το λειτουργικό σύστημα να του στείλει την απάντηση υπό τη μορφή ενός πακέτο UDP με την διεύθυνση IP και το UDP αριθμό πόρτας από την οποία το αίτημα αρχικά ήρθαν.

Το πακέτο απάντηση θα έχει την προέλευση και τον προορισμό αλλαγμένα το ένα με το άλλο απ'ότι ήταν στο αρχικό πακέτο, κατά την άφιξή του στο μηχάνημα πηγή, θα παραδοθεί στον πρόγραμμα-πελάτη που περιμένει.

Πως όμως οι πελάτες μαθαίνουν την διεύθυνση IP και την πόρτα που θα πρέπει να συνδεθούν?Υπάρχουν τρεις τρόποι:

- Convention: Πολλοί αριθμοί πορτών ορίστηκε έχουν οριστεί επίσημα,είναι γνωστές πόρτες για συγκεκριμένες υπηρεσίες, από την IANA, το Internet Assigned Numbers Authority.Γι 'αυτό ο DNS έτρεξε στο UDP πόρτα 53 στο ανωτέρω παράδειγμα.
- Automatic configuration: Συχνά είναι οι διευθύνσεις IP των κρίσιμων υπηρεσιών, όπως ο DNS είναι γνωστές όταν ένας υπολογιστής συνδέεται ένα δίκτυο, αν ένα πρωτόκολλο σαν το DHCP χρησιμοποιείται.Συνδυάζοντας αυτές τις διευθύνσεων IP με γνωστούς αριθμούς πορτών.τα προγράμματα,μπορούν να φτάσουν αυτές τις βασικές υπηρεσίες.
- Manual configuration: Για όλες τις καταστάσεις που δεν καλύπτονται από την δύο προηγούμενες περιπτώσεις, κάποια άλλο σχήμα θα πρέπει να παραδώσει μια διεύθυνση IP ή το αντίστοιχο hostname.

Κατά τη λήψη αποφάσεων σχετικά με τον καθορισμό των αριθμών των πορτών, όπως και 53 για το DNS, η IANA σκέφτηκε να τις ταξινομήσει σε τρεις κατηγορίες-και αυτό ισχύει τόσο για UDP και TCP αριθμούς πορτών:

- "Well-Known Ports" (0–1023) είναι για τα πρωτόκολλα οι πιο σημαντικές και ευρέως χρησιμοποιούμενες πόρτες. Σε πολλές Unix-like λειτουργικά συστήματα, κανονικά προγράμματα χρήστης δεν μπορεί να χρησιμοποιήσουν αυτές τις θύρες.
- "Registered Ports" (1024–49151) συνήθως δεν αντιμετωπίζονται ως κάτι ιδιαίτερο από το λειτουργικό συστήμα, κάθε χρήστης μπορεί να γράψει ένα πρόγραμμα που θα κάνει χρήση της πόρτας 5432 και προσποιείται ότι είναι μια Βάση δεδομένων PostgreSQL, για παράδειγμα, αλλά στον IANA καταχωρούνται για συγκεκριμένα πρωτόκολλα και η IANA συνιστά να

αποφεύγουν τη χρήση τους για κάθε άλλο αλλά για το πρωτόκολλο που τους έχει ανατεθεί.

- Οι πόρτες που απομένουν (49152–65535) είναι ελεύθερες για κάθε χρήση.Το λειτουργικο σύστημα αντλουν απόαυτές τις πόρτες όταν ένας client δεν ενδιαφέρετε για την πόρτα που έχει ανατεθεί.

Η Python εκθέτει τις κανονικές POSIX calls for raw UDP and TCP συνδέσεις ,αντί να προσπαθεί να εφεύρει κάποιο δικό του. οι κανονικές κλήσεις δικτύωσης POSIX λειτουργούν γύρω από μια κεντρική ιδέα που ονομάζεται sockets.

Οποίος έχει ξαναδουλέψει με POSIX,μάλλον θα πρέπει να παρακάψει το γεγονός ότι αντί να επαναλαμβάνουμε ένα όνομα αρχείο ξανά και ξανά, οι κλήσεις σας επιτρέπουν να χρησιμοποιήσετε το όνομα του αρχείου για να δημιουργήσετε ένα «file descriptor "που αντιπροσωπεύει μια σύνδεση στο αρχείο, και μέσω της οποίας μπορείτε να έχετε πρόσβαση στο αρχείο μέχρι να τελειώσετε με την επεξεργασία αυτή.Το πρωτόκολλο UDP χρησιμοποιεί sockets,δηλαδή τα sockets τι κάνουν?Όταν κάποιος ζητάει πρόσβαση σε μία γραμμή επικοινωνίας,όπως η πόρτα UDP,τότε δημιουργείτε ένα αφηρημένο αντικείμενο που ονομάζετε «socket»,έπειτα μπορεί να ζητήσει την πόρτα την οποία θέλει.Αν η σύνδεση είναι επιτυχής τότε το socket περιμένει στο αριθμό της πόρτας που έχει στη κατοχή του μέχρι την στιγμή που ο χρήστης θα “close” για να αποδεσμευσει τους πόρους.

Στην πραγματικότητα,sockets και file descriptors δεν είναι απλώς παρεμφερής έννοιες.Τα sockets στην πραγματικότητα είναι το file descriptors, που τυχαίνει να συνδεθεί με τις πηγές του δικτύου δεδομένων και όχι σε δεδομένα αποθηκευμένα σε ένα σύστημα αρχείων. Αυτό τους δίνει κάποιες ασυνήθιστες ικανότητες σε σχέση με τα κανονικά αρχεία. Αλλά το POSIX σας επιτρέπει επίσης να εκτελέσουν τις φυσιολογικές λειτουργίες αρχείων τους, όπως read () και write (), που σημαίνει ότι ένα πρόγραμμα που απλά θέλει να διαβάσει ή να γράψει απλά τα δεδομένα μπορεί να συμπεριφερθεί σε ένα socket σαν να ήταν ένα αρχείο χωρίς να γνωρίζουν τη διαφορά!
Πως μοιάζουν τα sockets στην πράξη? Το επόμενο προγραμμα δείχνει ένα απλό server και ένα client.Επίσης όλα τα είδη των δραστηριοτήτων που λαμβάνουν χώρα προέχρονται socket module στη Python Standard Library.

Πρόγραμμα 2-1. *UDP Server and Client on the Loopback Interface*

```
#!/usr/bin/env python
# UDP client and server on localhost
import socket, sys

s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
MAX = 65535
PORT = 1060
if sys.argv[1:] == ['server']:
```

```
» s.bind(('127.0.0.1', PORT))
» print 'Listening at', s.getsockname()
» while True:
» » data, address = s.recvfrom(MAX)
» » print 'The client at', address, 'says', repr(data)
» » s.sendto('Your data was %d bytes' % len(data), address)
elif sys.argv[1:] == ['client']:
» print 'Address before sending:', s.getsockname()
» s.sendto('This is my message', ('127.0.0.1', PORT))
» print 'Address after sending', s.getsockname()
» data, address = s.recvfrom(MAX) # overly promiscuous - see text!
» print 'The server', address, 'says', repr(data)
else:
» print >>sys.stderr, 'usage: udp_local.py server|client'
```

Ο καθένας είναι ικανός να τρέξει αυτό το πρόγραμμα,ακομά δεν έχει δίκτυο,επειδή server and client χρησιμοποιούν μόνο τη “localhost” IP διεύθυνση.Προσπαθώντας για το server πρώτα:

```
$ python udp_local.py server
Listening at ('127.0.0.1', 1060)
```

Μετά την εκτύπωση αυτή τη γραμμή της παραγωγής, ο server κολλάει και περιμένει για ένα εισερχόμενο μήνυμα. Στο κώδικα, μπορείτε να δείτε ότι χρειάστηκαν τρία βήματα για το server να σηκωθεί και να λειτουργήσει. Δημιούργηθηκε για πρώτη φορά ένα απλό socket με την υποδοχή socket (). Αυτό το νέο socket δεν έχει όνομα, δεν συνδέεται ακόμη με οτιδήποτε, και θα δημιουργηθεί μια εξαίρεση αν επιχειρήσετε οποιαδήποτε επικοινωνία μαζί του. αλλά το socket είναι, τουλάχιστον, ως ένδειξη ενός ιδιαίτερου τύπου: της οικογένειάς του είναι AF_INET, η οικογένεια του Διαδικτύου στα πρωτόκολλα, και είναι του τύπου SOCK_DGRAM datagram, το οποίο σημαίνει UDP. (Ο όρος "datagram" είναι ο επίσημος όρος για ένα μπλοκ σε επίπεδο εφαρμογής των διαβιβαζόμενων στοιχείων. Μερικοί άνθρωποι καλούν τα πακέτα UDP "Datagrams".
Στη συνέχεια, ο server χρησιμοποιεί την bind () εντολή για να ζητήσει μια διεύθυνση δικτύου UDP, την οποία μπορείτε να δείτε ότι είναι μια απλή πλειάδα που περιέχει μια διεύθυνση IP (το όνομα του συστήματος είναι επίσης αποδεκτό) και έναν αριθμό θύρας UDP.Σε αυτό το σημείο, η εξαίρεση θα δημιουργηθεί εάν ένα άλλο πρόγραμμα χρησιμοποιεί ήδη ότι η θύρα UDP και το server script δεν μπορούν να την αποκτήσουν.Προκαλώντας μία εξαίρεση:

```
$ python udp_local.py server
Traceback (most recent call last):
...
socket.error: [Errno 98] Address already in use
```

Υπάρχει περίπτωση να δημιουργηθεί και την πρώτη φορα μία εξαιρεση που τρέχω το server,αν η πόρτα 1060 χρησιμοποιούνταν ήδη στο σύστημα μου.

Μόλις το socket έχει δεσμευθεί με επιτυχία, ο server είναι έτοιμος να αρχίσει την παραλαβή των αιτήσεων! Εισέρχεται ένας βρόχος και τρέχει επανειλημμένα την recvfrom (), λέγοντας ότι η ρουτίνα θα λάβει μηνύματα μέχρι ένα μέγιστο μήκος MAX, το οποίο είναι ίσο με 65535 bytes-μια τιμή που συμβαίνει να είναι το μέγιστο μήκος το οποίο ένα πακέτο UDP μπορεί ενδεχομένως να έχει, έτσι ώστε να είμαστε πάντα σίγουροι ότι θα εμφανιστεί το πλήρες περιεχόμενο του κάθε πακέτου.Μέχρι να σταλεί ένα μήνυμα με έναν πελάτη, recvfrom () κλήση μας θα περιμένει για πάντα.

Ας δούμε τον κώδικα από την πλευρά του πελάτη.Ο κώδικας είναι ίδιος με αυτόν του προγράμματος 2-1,αρκεί στα arguments να περάσουμε την συμβολοσειρά client.

```
$ python udp_local.py client
Address before sending: ('0.0.0.0', 0)
Address after sending ('0.0.0.0', 33578)
The server ('127.0.0.1', 1060) says 'Your data was 18 bytes'
$ python udp_local.py client
Address before sending: ('0.0.0.0', 0)
Address after sending ('0.0.0.0', 56305)
The server ('127.0.0.1', 1060) says 'Your data was 18 bytes'
```

Πάνω από το παράθυρο εντολών του server, αυτό θα πρέπει να δειτε αναφέροντας κάθε σύνδεση που εξυπηρετεί:

```
The client at ('127.0.0.1', 41201) says, 'This is my message'
The client at ('127.0.0.1', 59490) says, 'This is my message'
```

Αν ο κώδικας πελάτη είναι ελαφρώς απλούστερος από εκείνη του διακομιστή - υπάρχουν μόνο δύο σημαντικές γραμμές κώδικα- εισάγει πολλές νέες έννοιες.

Πρώτον, ο πελάτης παίρνει το χρόνο για να προσπαθήσει μια getsockname () πριν οποιαδήποτε διεύθυνση ανατεθεί στο socket. Αυτό μας επιτρέπει να δούμε ότι τόσο η διεύθυνση IP και όσο και ο αριθμό θύρας ξεκινήσαν όλα μηδενικά,ένα νέο socket είναι μία κενή κατάσταση. Στη συνέχεια, ο πελάτης καλεί sendto () τόσο με ένα μήνυμα και μια διεύθυνση προορισμού.Αυτή η απλή κλήση είναι το μόνο που χρειάζεται είναι να στείλει ένα πακέτο που ταξιδεύει προς τον server! Αλλά, φυσικά, χρειαζόμαστε μια διεύθυνση IP τη διεύθυνση και τον αριθμό θύρας εαυτούς μας, στο τέλος πελάτη, αν θέλουμε να επικοινωνούν μεταξύ τους. Έτσι, το λειτουργικό σύστημα εκχωρεί αυτόματα μία, όπως μπορείτε να δείτε από την έξοδο της δεύτερης πρόσκλησης για getsockname (). Και, όπως είχε υποσχεθεί, οι αριθμοί των θυρών είναι κάθε πελάτη από το φάσμα της IANA για "εφήμερα" αριθμούς θυρών. Δεδομένου ότι ο πελάτης γνωρίζει ότι αναμένει μια απάντηση από τον server, ο ίδιος αποκαλεί απλά η υποδοχή του recv () μέθοδο, χωρίς να ενοχλεί με την recvfrom () έκδοση που επιστρέφει επίσης διεύθυνση. Όπως μπορείτε να δείτε από τον αποτελεσμα τους, τόσο ο πελάτης όσο και ο διακομιστής που βλέπουμε με επιτυχία τα μηνύματα ο ένας του άλλου του άλλου, κάθε φορά που ο πελάτης τρέχει, μια πλήρης μετ 'επιστροφής της αίτησης και απάντησης περνώντας ανάμεσα από τα δύο UDP sockets.

Unreliability

Επειδή ο client και server στην προηγούμενη ενότητα, και οι δύο τρέχουν στο ίδιο μηχάνημα και μιλώντας μέσω του loopback interface, τα οποία δεν είναι καν μια φυσική κάρτα δικτύου με την οποία θα μπορούσαν να βιώσουν μία signaling βλάβη και να χάσουν ένα πακέτο, αλλά απλώς μια εικονική σύνδεση πίσω στο ίδιο μηχάνημα στο βάθος στη στοίβα του δικτύου-δεν υπαρχεί κανένας τρόπος με τον οποίο τα πακέτα να μπορούσε να χαθούν, και έτσι δεν είδαμε στην πραγματικότητα τίποτα από την ταλαιπωρία του UDP. Πώς να αλλαχθεί ο κώδικας, έτσι ώστε κάποιο πακέτο να μπορούσε να χαθεί;
Ρίξτε μια ματιά στο Πρόγραμμαα 2-2. Σε αντίθεση με το προηγούμενο παράδειγμα, μπορείτε να εκτελέσετε αυτόν τον πελάτη και διακομιστή σε δύο διαφορετικά μηχανήματα στο Διαδίκτυο. Και αντί να απαντάει πάντα στις αιτήσεις των πελατών, αυτός ο διακομιστής
τυχαία επιλέγει να απαντήσει μόνο τα μισά από τα αιτήματα που έρχονται από τους πελάτες-τα οποία θα μας δείξουν πώς να χτίζετε η αξιοπιστία στον κώδικα των πελατών μας, χωρίς να περιμένει κάποιος για ώρες αν θα χαθεί κάποιο πακέτο.

Πρόγραμμα 2–2. *UDP Server and Client σε διαφορετικά μηχανήματα*

```
#!/usr/bin/env python
# udp_remote.py
# UDP client and server for talking over the network
import random, socket, sys
s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
MAX = 65535
PORT = 1060
if 2 <= len(sys.argv) <= 3 and sys.argv[1] == 'server':
» interface = sys.argv[2] if len(sys.argv) > 2 else ''
» s.bind((interface, PORT))
» print 'Listening at', s.getsockname()
» while True:
» » data, address = s.recvfrom(MAX)
» » if random.randint(0, 1):
» » » print 'The client at', address, 'says:', repr(data)
» » » s.sendto('Your data was %d bytes' % len(data), address)
» » else:
» » » print 'Pretending to drop packet from', address
elif len(sys.argv) == 3 and sys.argv[1] == 'client':
» hostname = sys.argv[2]
» s.connect((hostname, PORT))
» print 'Client socket name is', s.getsockname()
» delay = 0.1
» while True:
» » s.send('This is another message')
```

```
» » print 'Waiting up to', delay, 'seconds for a reply'
» » s.settimeout(delay)
» » try:
» » » data = s.recv(MAX)
» » except socket.timeout:
» » » delay *= 2 # wait even longer for the next request
» » » if delay > 2.0:
            raise RuntimeError('I think the server is down')
» » except:
» » » raise # a real error, so we let the user see it
» » else:
» » » break # we are done, and can stop looping
» print 'The server says', repr(data)
else:
» print >>sys.stderr, 'usage: udp_remote.py server [ <interface> ]'
» print >>sys.stderr, ' or: udp_remote.py client <host>'
» sys.exit(2)
```

Καθώς ο server στο προηγούμενο παράδειγμα είπε ότι το λειτουργικό σύστημα το οποίο ήθελε μόνο τα πακέτα που φτάνουν από άλλες διεργασίες στο ίδιο μηχάνημα μέσω της ιδιωτικής διεπαφής 127.0.0.1, αυτός ο server είναι είναι πιο γενναιόδωρη και φιλοξενεί πακέτα που φθάνουν στον server μέσω οποιουδήποτε δικτύου διασύνδεσης. Αυτός είναι ο λόγος που προσδιορίζει την διεύθυνση IP του διακομιστή '', που σημαίνει «οποιαδήποτε τοπική διεπαφή,"
που το Linux laptop μου που μεταφράζεται σε 0.0.0.0, όπως μπορούμε να δούμε από τη γραμμή που εκτυπώνει όταν αρχίζει:

```
$ python udp_remote.py server
Listening at ('0.0.0.0', 1060)
```

Όπως μπορείτε να δείτε, κάθε φορά που παραλαμβάνει την αίτηση, ο server χρησιμοποιεί την randint ()για να αποφασίσει αν το αίτημα αυτό θα πρέπει να απαντηθούν, έτσι ώστε να μην χρειάζεται να συνεχίσει να τρέχει στον πελάτη όλη την ημέρα περιμένοντας για ένα πραγματικό πακέτο πέσει. Όποια απόφαση παίρνει, τυπώνει ένα μήνυμα στην οθόνη έτσι ώστε να μπορεί να συμβαδίσει με τη δραστηριότητά του.

Λοιπόν,πως μπορούμε να γράψουμε ένα «πραγματικό» UDP client,έναν ο οποίος θα αντιμετωπίζει τα πακέτα τα οποία θα χάνονται?Πρώτα,αναξιοπιστία UDP σημαίνει ότι ότι ο πελάτης πρέπει να εκτελέσει τις αιτήσεις του μέσα σε έναν βρόχο, και ότι,στην πραγματικότητα, πρέπει να είναι κάπως αυθαίρετος κατά την λήψη απόφασεων,αποφασίοντας δηλαδή πότε περίμενε «παρα πολύ» για μια απαντηση και χρειάζετε να στειλει μία δεύτερη.Αυτη η δύσκολη επιλογή είναι απαραίτητη επείδη δεν υπαρχεί κανένας τρόπος γενικά για τον πελατή να διαχςρισεί τιν διαφορά σε τρεις διαφορετικες καταστάσεις:

- Η απάντηση παίρνει πολύ χρόνο για να επιστρέψει, αλλά θα φτάσει σύντομα.
- Η απάντηση ποτέ δεν θα φτάσει,γιατι η απάντηση ή το αίτημα έχει χαθεί.
- Ο server είναι down κ δεν απαντάει σε κανέναν.

Έτσι,ένας UDP client πρέπει να επιλέξει ένα πρόγραμμα στο οποίο θα στέλνονται τα αντίγραφα των αιτήσεων αν αυτά περιμένουν κάποια περίοδο του χρόνου χωρίς να πάρουν κάποια απάντηση.Φυσικά,αυτό μπορεί να είναι χρονοβορο για τον server επειδή το πρώτο αντίγραφο μπορεί να φτλανει κ το δευτέρο αντίγραφο της αιτήσης να προκαλέσει ένα δευτερο αντλιγραφο,αλλά σε κάποιο σημείο ο client πρέπει να αποφασίσει αν θα χαναστέλνει ή θα ρισκάρει και θα περιμένει για πάντα.

Όταν εκτελείτε το πρόγραμμα-πελάτη, δώστε το όνομα κεντρικού υπολογιστή του άλλου μηχανήματος στο οποίο τρέχετε το server script, όπως δείξαμε προηγουμένως:

```
$ python udp_remote.py client guinness
Client socket name is ('127.0.0.1', 45420)
Waiting up to 0.1 seconds for a reply
The server says 'Your data was 23 bytes'
```

Αν ένας server είναι down?Δυστηχώς,στο UDP δεν υπάρχει κάποιος τρόπος να διαχωρίσουμε ανάμεσα αν ο server είναι down ή ένα δίκτυο δεν είναι σε καλή κατάσταση και χάνει όλα τα πακέτα.Ετσι το καλυτέρο που έχει να κάνει ένας client είναι να τα παρατήσει αν έχει κάνει ήδη αρκετές προσπάθειες.Σταμάτησε τη διαδικασία server και προσπάθησε να τρέξετε τον client ξανά:

```
$ python udp_remote.py client guinness
Waiting up to 0.1 seconds for a reply
Waiting up to 0.2 seconds for a reply
Waiting up to 0.4 seconds for a reply
Waiting up to 0.8 seconds for a reply
Waiting up to 1.6 seconds for a reply
Traceback (most recent call last):
...
RuntimeError: I think the server is down
```

UDP Fragmentation

Μέχρι στιγμής το UDP σας επιτρέπει, ως χρήστης,να στείλετε raw πακέτα δικτύου με
στα οποία μόνο ένα μικρό κομμάτι των πληροφοριών (μία διεύθυνση IP και θύρα για τόσο τον αποστολέα και τον παραλήπτη) έχει πρόσθεθει. Αλλά ίσως να έχετε ήδη καταστεί ύποπτοι, γιατί έχουν τα ανωτέρω προγράμματα προτείνουν ότι ένα πακέτο UDP μπορεί να είναι μέχρι 64kB σε μέγεθος, ενώ ίσως ήδη γνωρίζετε ότι η Ethernet ή η ασύρματη κάρτα μπορεί να χειριστεί μόνο τα πακέτα των περίπου 1.500 bytes αντ 'αυτού.

Η πραγματική αλήθεια είναι ότι η IP στέλνει μικρά πακέτα UDP ως ενιαία πακέτα σχετικά με το wire, αλλά χωρίζει μεγαλύτερο UDP πακέτα σε πολλά μικρά πακέτα φυσικά. Αυτό σημαίνει ότι μεγάλα πακέτα είναι πιο πιθανό να μειωθούν, δεδομένου ότι εάν κάποιο από τα κομμάτια τους δεν κάνει το δρόμο προς τον προορισμό του,τότε ολόκληρο το πακέτο δεν μπορεί να συγκεντρωθεί και να παραδοθεί στο λειτουργικού συστήματος που περιμένει.

Αλλά εκτός από την υψηλή πιθανότητα αποτυχίας, η διαδικασία κατακερματισμού των μεγάλων πακέτων UDP, έτσι ώστε θα ταιριάζει στο σύρμα θα πρέπει να είναι αόρατη στην αίτησή σας. Υπάρχουν τρεις τρόποι, όμως, στην οποία μπορεί να είναι χρήσιμα:

- Εάν σκέφτεστε για την αποτελεσματικότητα, μπορεί να θέλετε να περιορίσει το πρωτόκολλο σε μικρά πακέτα, να είναι λιγότερο πιθανό να κάνει αναμετάδοση και να περιορίσει το πόσο διάστημα χρειάζεται η remote IP στοίβα να επανασυναρμολογήσει το πακέτο UDP σας και να το δώσει στην εφαρμογή που περιμένει.

- Εάν τα πακέτα ICMP είναι εσφαλμένα μπλοκαρισμένα από ένα τείχος προστασίας το οποίο θα επρέπε κανονικά να επέτρεπε το host σου να ανιχνεύσει αυτόματα το μέγεθος του MTU ανάμεσα σε εσάς και τον απομακρυσμένο υπολογιστή, τότε τα μεγαλύτερα πακέτα UDP μπορεί να εξαφανιστούν χωρίς να το καταλάβει κανείς.Το MTU είναι η "μέγιστη μονάδα μετάδοσης" ή "το μεγαλύτερο μέγεθος πακέτου" ,το οποίο όλες οι συσκευές δικτύου μεταξύ δύο hosts θα υποστηρίζουν.

- Εάν το πρωτόκολλο σας μπορεί να κάνει τις δικές του επιλογές σχετικά με το πώς χωρίζει τα δεδομένα ανάμεσα σε διαφορετικά πακέτα, και θέλετε να είστε σε θέση να ρυθμίζετε αυτόματα το μέγεθος με βάση το πραγματικό MTU μεταξύ δύο hosts, τότε ορισμένα λειτουργικά συστήματα σας επιτρέπουν να απενεργοποιήσετε τον κατακερματισμό και να λάβετε ένα σφάλμα, αν ένα πακέτο UDP είναι πολύ μεγάλο. Αυτό σας επιτρέπει να ανασυνταχθεί και να γίνει διαχωρισμός σε διάφορα πακέτα, αν αυτό είναι δυνατό.

Το Linx είναι ένα λετουργικό σύστημα,το οποίο υποστηρίζει την τελευταία επιλογή.Το επόμενο πρόγραμμα στέλνει ένα πολύ μεγάλο μήνυμα σε έναν από τους servers που έχουν σχεδιαστεί.

Πρόγραμμα 2-3.Στέλνοντας ένα πολύ μεγάλο UDP πακέτο

```
#!/usr/bin/env python
# big_sender.py
# Send a big UDP packet to our server.
import IN, socket, sys
s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
MAX = 65535
PORT = 1060
if len(sys.argv) != 2:
» print >>sys.stderr, 'usage: big_sender.py host'
» sys.exit(2)
hostname = sys.argv[1]
s.connect((hostname, PORT))
s.setsockopt(socket.IPPROTO_IP, IN.IP_MTU_DISCOVER, IN.IP_PMTUDISC_DO)
try:
```

```
» s.send('#' * 65000)
except socket.error:
» print 'The message did not make it'
» option = getattr(IN, 'IP_MTU', 14) # constant taken from <linux/in.h>
» print 'MTU:', s.getsockopt(socket.IPPROTO_IP, option)
else:
» print 'The big message was sent! Your network supports really big packets!'
```

Αν τρέξουμε αυτό το πρόγραμμα σε ένα διακομιστή οπουδήποτε στο δίκτυο του σπιτιού μου, τότε θα ανακαλύψουμε ότι το ασύρματο δίκτυο επιτρέπει φυσικά πακέτα τα οποία δεν θα είναι μεγαλύτερα από 1.500 byte συνήθως υποστηρίζονται από δίκτυα Ethernet τύπου:

```
$ python big_sender.py guinness
The message did not make it
MTU: 1500
```

Είναι ξεκάθαρο το γεγονός ότι η διασύνδεση loopback,το οποίο προφανώς θα μπορούσε να υποστηρίξη πακέτα τόσο μεγάλα όσο RAM μου, επιβάλλει επίσης μια MTU που απέχει πολύ από το μέγιστο πακέτο UDP μήκος:

```
$ python big_sender.py localhost
The message did not make it
MTU: 16436
```

Broadcast

Εάν το UDP έχει κάτι αξιοσημείωτο,αυτό είναι η ικανότητά του να υποστηρίξει broadcast: αντί για την αποστολή ενός πακέτου σε συγκεκριμένους άλλους hosts, μπορείτε να το επισημάνετε σε ολόκληρο υποδίκτυο στο οποίο το μηχάνημα είναι και έχουν το
φυσική κάρτα δικτύου μεταδίδει το πακέτο, έτσι ώστε όλοι οι συνημμένοι hosts το βλέπουν χωρίς να χρειάζεται να είναι αντιγράφετε ξεχωριστά σε κάθε έναν από αυτούς.

Τώρα, πρέπει να αναφερθεί ότι το broadcast θεωρείται passé αυτές τις μέρες, επειδή ένας
πιο εκλεπτυσμένη τεχνική που ονομάζεται «multicast» έχει αναπτυχθεί, το οποίο σας επιτρέπει τα σύγχρονα λειτουργικά συστήματα να αξιοποιήσουν καλύτερα τη νοημοσύνη που είναι χτισμένα πολλά δίκτυα και συσκευές διασύνδεσης δικτύου. Επίσης, το multicast μπορεί να συνεργαστεί με hosts που δεν είναι στο τοπικό υποδίκτυο, το οποίο είναι αυτό που κάνει το broadcast λιγότερο χρήσιμο για πολλές εφαρμογές! Αλλά εάν θέλετε έναν εύκολο τρόπο για να κρατήσει κάτι σαν παιχνίδια πελάτες ή αυτοματοποιημένα πίνακες ενημερωμένους στο τοπικό δίκτυο, και ο κάθε πελάτης να μπορεί να υπομένει περιστασιακό χάσιμο πακέτων,τότε το UDP Broadcast είναι μια εύκολη επιλογή.

Το πρόγραμμα 2-4 δείχνει ένα παράδειγμα ενός διακομιστή που μπορεί να λάβει πακέτα εκπομπής και έναν πελάτη που μπορεί να τα στείλει.

Πρόγραμμα 2–4. *UDP Broadcast*

```
#!/usr/bin/env python
# UDP client and server for broadcast messages on a local LAN
s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
s.setsockopt(socket.SOL_SOCKET, socket.SO_BROADCAST, 1)
MAX = 65535
PORT = 1060
if 2 <= len(sys.argv) <= 3 and sys.argv[1] == 'server':
» s.bind(('', PORT))
» print 'Listening for broadcasts at', s.getsockname()
» while True:
» » data, address = s.recvfrom(MAX)
» » print 'The client at %r says: %r' % (address, data)
elif len(sys.argv) == 3 and sys.argv[1] == 'client':
» network = sys.argv[2]
» s.sendto('Broadcast message!', (network, PORT))
else:
» print >>sys.stderr, 'usage: udp_broadcast.py server'
» print >>sys.stderr, ' or: udp_broadcast.py client <host>'
» sys.exit(2)
```

Όταν δοκιμάζετε αυτόν το διακομιστή και το πελάτη, το πρώτο πράγμα που πρέπει να παρατηρήσετε είναι ότι συμπεριφέρεται ακριβώς σαν ένας κανονικός client και server, αν μπορείτε απλά να χρησιμοποιήσετε τον πελάτη για να στείλετε πακέτα που απευθύνονται στη διεύθυνση IP ενός συγκεκριμένου server. Ενεργοποιώντας το broadcast για ένα UDP socket δεν απενεργοποιείτε ή να αλλάζει η δυνατότητα του να στέλνει και να λαμβάνει και διευθυνσιοδοτεί πακέτα.
Το ενδιαφέρον κομμάτι συμβαίνει όταν βλέπει κάποιος τις ρυθμίσεις για το τοπικό δίκτυο, και να χρησιμοποιεί την IP του "διεύθυνση Broadcast "ως προορισμό για τον client. Αφήστε πρώτα ένα ή δύο servers στο δίκτυό σας, χρησιμοποιώντας εντολές όπως παρακάτω:

```
$ python udp_broadcast.py server
Listening for broadcasts at ('0.0.0.0', 1060)
```

Στη συνέχεια, ενώ βρίσκονται σε εξέλιξη, χρησιμοποιήστε πρώτα τον client για να στείλετε μηνύματα σε κάθε server. Θα δείτε ότι μόνο ένας server παίρνει κάθε μήνυμα:

```
$ python udp_broadcast.py client 192.168.5.10
```

Αλλά όταν χρησιμοποιείτε τη διεύθυνση broadcast του τοπικού δικτύου, ξαφνικά θα δείτε ότι όλοι οι broadcasts servers θα πάρουν το πακέτο την ίδια στιγμή! (Αλλά κανένας συνήθης server θα δείτε).Στο τοπικό δίκτυο μου αυτή τη στιγμή, η εντολή ifconfig μου λέει ότι η διεύθυνση broadcast είναι η εξής:

```
$ python udp_broadcast.py client 192.168.5.255
```

Και, βεβαίως, και οι δύο διακομιστές αμέσως αναφέρουν ότι βλέπουν το μήνυμα! Σε περίπτωση που το λειτουργικό σύστημα καθιστά δύσκολο να προσδιοριστεί η διεύθυνση broadcast, και δεν σας πειράζει να κάνετε ένα broadcast από κάθε θύρα δικτύου του host σας, η Python σας επιτρέπει να χρησιμοποιήσετε το ειδικό hostname «<broadcast>» κατά την αποστολή με UDP socket. Να είστε προσεκτικοί με το όνομα,όταν περνώντας το στο πελάτη μας, δεδομένου ότι οι χαρακτήρες < και > έχουν ιδιαίτερη σημασία σε κάθε κανονικό POSIX shell:

$ python udp_broadcast.py client "<broadcast>"

ΚΕΦΑΛΑΙΟ 5^{Ο}

TCP

Το Transmission Control Protocol (TCP) είναι η κινητήριος δύναμη του Διαδικτύου. Πρώτα ορίστηκε το 1974, το πρωτόκολλα αυτό αφήνει τις εφαρμογές να στέλνει η μια στην άλλη streams of δεδομένων που, όταν φτάνουν όλα τελειωνει, εκτός αν η σύνδεση έχει πρόβλημα πάλι είναι εγγυημένο ότι θα φτάσει ανέπαφη, στη σειρά, και χωρίς επικαλύψεις.

Πρωτόκολλα που φέρουν έγγραφα και αρχεία σχεδόν πάντα οδηγούνται στην κορυφή του πρωτοκόλλου TCP, συμπεριλαμβανομένων των HTTP και όλα τους βασικούς τρόπους μετάδοσης ηλεκτρονικού ταχυδρομείου. Είναι επίσης η βάση της επιλογής για τα πρωτόκολλα που πραγματοποιούν μεγάλη συνομιλίες μεταξύ των ανθρώπων ή υπολογιστές, όπως SSH και πολλά δημοφιλή πρωτόκολλα συνομιλίας.

Πως το TCP λειτουργει

Τα πραγματικά δίκτυα είναι άστατος πράγματα που ρίχνουν μερικές φορές τα πακέτα που μεταδίδουν σε αυτά, κατά καιρούς δημιουργούν επιπλέον αντίγραφα ενός πακέτου αντ 'αυτού, και είναι επίσης γνωστό ότι μεταδίδουν τα πακέτα χωρίς σειρά. Με ένα πακέτο εγκατάστασης, όπως το UDP, θα πρέπει να ανησυχείτε για το αν ταμηνύματα έφτασαν. Αλλά με το TCP, τα πακέτα είναι κρυμμένα και η εφαργογή μπορεί πολύ απλά να μεταδώσει τα δεδομένα προς τον προορισμό τους , επίσης θα είναι αναμεταδωθούν μέχρι να φτάσουν τελικά.

Πώς το TCP παρέχει αξιόπιστη σύνδεση; Πρώτον, σε κάθε πακέτο δίνεται έναν αριθμό ακολουθίας, έτσι ώστε το σύστημα που τα λαμβάνει να μπορεί να βάλει μαζί με τη σωστή σειρά, και έτσι ώστε να μπορεί να παρατηρήσουν ποια πακέτα λείπουν από την ακολουθία και να ζητήσει
από αυτά να αναμεταδωθούν.

Αντί της χρησιμοποίησης διαδοχικών ακέραιων αριθμών (1,2, ...) με τον αριθμό του πακέτου, το TCP χρησιμοποιεί ένα μετρητή που μετρά τον αριθμό των bytes που μεταδίδονται. Έτσι, ένα 1024-byte πακέτο με αριθμό ακολουθίας των 7.200 θα πρέπει να ακολουθηθεί από ένα πακέτο με αριθμό ακολουθίας των 8.224. Αυτό σημαίνει ότι μια πολυάσχολη στοίβα δικτύου δεν χρειάζεται να
θυμάστε πώς πρέπει να σπάσει μια ροή δεδομένων σε πακέτα,αν ζητηθεί εκ νέου μετάδοση, μπορεί να σπάσει η ροή σε πακέτα με κάποιο τρόπο (που θα μπορούσε να αφήσει να χωρέσουν περισσότερα δεδομένα σε ένα πακέτο, αν περισσότερα bytes περιμένουν για μετάδοση), και ο δέκτης μπορεί να βάλει τα πακέτα μαζί.

Αντί η όλη διαδικασία να τρέχει πολύ αργά δηλαδη να πρεπει κάθε πακέτο να αναγνωριστεί πριν σταλεί το επόμενο, το TCP στέλνει πολλά πακέτα την ίδια στιγμη προτού να περιμένει μια απάντηση. Η ποσότητα των δεδομένων που ο αποστολέας είναι διαθέσιμος να στείλει σε κάθε δεδομένη στιγμή καλείται το μέγεθος του TCP "window".

Στην εφαρμογή του πρωτόκολλο TCP για την παραλαβή μπορεί να ρυθμίσει το μέγεθος του παραθύρου της διαβίβασης τέλος,και, επομένως, να κανει την συνδεση πιο αργή ή να επιβάλει την παύση της σύνδεσης. Αυτό ονομάζεται «ελέγχος ροής». Αυτό σας επιτρέπει να απαγορεύσετε τη μετάδοση επιπλέον πακέτων σε περιπτώσεις όπου ο buffer εισόδου του είναι πλήρης και θα πρέπει να απορρίψετε κάθε επιπλέον δεδομένα, εάν ήταν να φτάσει τώρα.

Τέλος, αν το πρωτόκολλο TCP βλέπει ότι τα πακέτα χάνονται, υποθέτει ότι στο δίκτυο γίνεται συμφόρηση και σταματά την αποστολή πακέτων κάθε δευτερόλεπτο. Αυτό μπορεί καταστροφικό σε ασύρματο δίκτυα και άλλα μέσα ενημέρωσης, όπου τα πακέτα μερικές φορές απλά χάνονται λόγω του θορύβου. Μπορεί επίσης να καταστρέψει συνδέσεις που λειτουργούν καλά μέχρι μία επανεκκίνηση του δρομολογητή και τα τελικά σημεία δεν μπορεί να μιλήσουν, ας πούμε, για 20 δευτερόλεπτο, από τη στιγμή που το δίκτυο επιστρέφει, οι δύο σταθμοί TCP θα έχουν διαπιστώσει ότι ο δίκτυο είναι αρκετά υπερφορτωμένο με κυκλοφορία, και για αρκετό καιρό αργότερα αρνούνται να στείλουν άλλα δεδομένα.

Πότε να χρησιμοποιήσουμε το TCP

Επειδή το TCP έχει πλέον γίνει προεπιλογή, όταν δύο προγράμματα πρέπει να επικοινωνήσουν,πρέπει να εξετάσουμε μερικές περιπτώσεις στις οποίες η συμπεριφορά της δεν είναι η βέλτιστη για ορισμένα είδη δεδομένων.

Πρώτον, το TCP είναι δύσχρηστο όταν οι clients θέλουν να στείλουν,μικρά αιτήματα σε ένα διακομιστή και δεν θα μιλήσουμε για περαιτέρω. Χρειάζονται τρία πακέτα για δύο hosts για να δημιουργήθει μία TCP σύνδεση-η περίφημη ακολουθία του SYN, SYN-ACK, και ACK -και στη συνέχεια άλλα τρία ή τέσσερα για να κλείσει την σύνδεση (είτε ένα γρήγορο FIN, το δίκτυο FIN-ACK, ACK, ή ένα ελαφρώς μεγαλύτερο ζεύγος από ξεχωριστά FIN και ACK πακέτα). Αυτά είναι τα έξι πακέτα για να υποβληθεί μία μόνο αίτηση! Οι σχεδιαστές του πρωτοκόλλου αυτές τις περιπτώσεις τις μετατρέπον γρήγορα σε UDP.

Ένα ερώτημα που τίθεται, είναι αν ένας client που θέλει να ανοίξει μια σύνδεση TCP και στη συνέχεια την χρησιμοποιήσει για αρκετά λεπτά ή ώρες για να κάνει πολλές χωριστές αιτήσεις στον ίδιο διακομιστή. Μόλις η σύνδεση τρέχει και το κόστος του handshake έχει καταβληθεί, σε κάθε αίτημα και απάντηση απαιτήτε μόνο ένα πακέτο σε κάθε κατεύθυνση.Αυτα οφείλονται όλα στα πλεονεκτήματα του πρωτόκολλου TCP για πληροφορίες για την εκ νέου διαβίβαση, exponential backing off,, και έλεγχο ροής.

Εκεί που το UDP διαπρέπει,είναι όταν μια long-term σχέση δεν ανακύπτουν μεταξύ
client και server, και ειδικά όταν υπάρχουν τόσοι πολλοί πελάτες όπου μια τυπική

εφαρμογή του πρωτόκολλο TCP θα εξαντλούσε τα port numbers αν έπρεπε να συμβαδίσει με μια ξεχωριστή ροή δεδομένων για κάθε ενεργό πελάτη.

Τι σημαίνει το TCP Sockets

Όπως συμβαίνει και με το UDP,το TCP χρησιμοποιεί port numbers για να διακρίνει διαφορετικές εφαρμογές που εκτελούνται κάτω από την ίδια διεύθυνση IP, και ακολουθεί ακριβώς τις ίδιες συμβάσεις σχετικά με τις γνωστές κ ephemeral port numbers.

Όπως είδαμε στο προηγούμενο κεφάλαιο, χρειάζεται μόνο ένα socket για να μιλήσει UDP: ένας server μπορεί να ανοίξει ένα datagram port και στη συνέχεια να λάβει πακέτα από χιλιάδες διαφορετικούς πελάτες.Καθώς είναι δυνατό να κάνεις connect () ένα datagram socket σε ένα συγκεκριμένο συνομιλητή, ώστε να μπορείτε πάντα να κάνετε send () σε μία
διεύθυνση και μόνο recv () πακέτα που αποστέλλονται πίσω από αυτή τη διεύθυνση, η ιδέα της σύνδεσης είναι μόνο για ευκολία. Το αποτέλεσμα του connect () είναι ακριβώς το ίδιο με μία εφαρμογή όταν απλά αποφασίζει να στέλνει sendto() σε μία μόνο διευθυνση.

Αλλά με ένα stateful stream πρωτόκολλο όπως το TCP, η connect () κλήση γίνεται η βασική πράξη που βασίζονται όλες οι άλλες αρθρώσεις επικοινωνίας του δικτύου. Είναι, στην πραγματικότητα, η στιγμή κατά τη λειτουργία του συστήματος σας η στοίβα δικτύου ξεκινά το handshake που μόλις περιγράψαμε -σε περίπτωση επιτυχίας, θα κάνει και τα δύο άκρα της ροής TCP έτοιμα για χρήση.

Και αυτό σημαίνει ότι το πρωτόκολλο TCP connect() μπορεί να αποτύχει.Ο απομακρυσμένος host δεν μπορεί να απαντήσει.μπορεί να αρνηθεί την σύνδεση,ή άλλα λάθη του πρωτόκολλου μπορεί να συμβούν, όπως την άμεση παραλαβή των RST ("reset")πακέτων. Επειδή ένα stream connection προϋποθέτει τη σύσταση μια μόνιμη σύνδεση μεταξύ δύο κεντρικών υπολογιστών,ο άλλος host πρέπει να ακούει και na είναι έτοιμος να αποδεχτεί τη σύνδεσή σας.

Από την "πλευρά του διακομιστή"-η οποία, για τους σκοπούς του παρόντος κεφαλαίου, είναι το συνομιλητή δεν κάνει connect () κλήση, αλλά λαμβάνει το πακέτου SYN το οποίο αρχικοποιεί-μία εισερχόμενη σύνδεση δημιουργεί ένα ακόμη πιο βαρυσήμαντο γεγονός, η δημιουργία ενός νέου socket! Αυτό οφείλεται στο γεγονός ότι η τυποποιημένη διεπαφή POSIX στο TCP που περιλαμβάνει στην πραγματικότητα δύο εντελώς διαφορετικά είδη sockets: "passive" ακούν sockets και "connected" στα ενεργά.

- Ένα passive socket κατέχει το "socket name", η διεύθυνση και ο αριθμός θύρας στα
  που ο server είναι έτοιμος να δεχτεί συνδέσεις. Δεν υπάρχουν δεδομένα που δεν μπορεί ποτέ να λάβει ή αποστέλλονται από αυτό το είδος του port, δεν αντιπροσωπεύει καμία πραγματική συνομιλία του δικτύου.Αντ 'αυτού, είναι

πως ο server προειδοποιεί το λειτουργικό σύστημα για την επιθυμία του να λάβει εισερχόμενες συνδέσεις στην πρώτη θέση.

- Ένα active ,connected socket είναι συνδεδεμένο σε ένα συγκεκριμένο απομακρυσμενή συνομιλία , ο οποίος έχει τη δική του διεύθυνση IP και τον port number. Μπορεί να χρησιμοποιηθεί μόνο για μιλάμε με τον εν λόγω εταίρο, και μπορεί να διαβάσει και να γράψει χωρίς να Ανησυχώ για το πώς τα δεδομένα που προκύπτουν θα πρέπει να χωριστούν σε πακέτα-σε πολλές περιπτώσεις, ένα connected socket μπορεί να περάσει σε ένα άλλο πρόγραμμα POSIX που αναμένει να διαβαστούν από ένα κανονικό αρχείο, και το πρόγραμμα ποτέ δεν θα γνωρίζει καν ότι μιλάμε για το δίκτυο!

Σημειώστε ότι ενώ ένα passive socket είναι μοναδικό από την interface address και το port number στην οποία να ακούει (έτσι ώστε κανένας άλλος δεν επιτρέπεται να τραβήξει την ίδια διεύθυνση και το port), μπορεί να υπάρχουν πολλά active socket που όλοι μοιράζονται το ίδιο τοπικό socket name . Είναι ένας web server με πολύ κυκλοφορια στον οποίο χιλιάδες πελάτες έχουν φτιάξει όλες τις συνδέσεις HTTP, για παράδειγμα, θα έχουν χίλια active sockets όλα προορισμένα για την δημόσια IP διεύθυνση στη θύρα 80. Τι κάνει ένα active socket μοναδικό, μάλλον, τα τέσσερα μέρη που το συντονίζουν:

(local_ip, local_port, remote_ip, remote_port)

Είναι αυτή η πλειάδα των τεσσάρων, με την οποία το λειτουργικό σύστημα ονομάζει κάθε ενεργή TCP σύνδεση και τα εισερχόμενα πακέτα TCP εξετάζονται για να διαπιστωθεί αν η πηγή τους και η διεύθυνση προορισμού τους συνδέονται με οποιαδήποτε από τα ενεργά socket που τρέχουν εκείνη τη στιγμή στο σύστημα.

‘Ενας απλός TCP Client and Server

Πρόγραμμα 3-1. *Simple TCP Server and Client*

```
#!/usr/bin/env python
#  tcp_sixteen.py
# Simple TCP client and server that send and receive 16 octets
import socket, sys
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
HOST = sys.argv.pop() if len(sys.argv) == 3 else '127.0.0.1'
PORT = 1060
def recv_all(sock, length):
» data = ''
» while len(data) < length:
» » more = sock.recv(length - len(data))
» » if not more:
» » » raise EOFError('socket closed %d bytes into a %d-byte message'
» » » % (len(data), length))
```

```
» » data += more
» return data
if sys.argv[1:] == ['server']:
» s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
» s.bind((HOST, PORT))
» s.listen(1)
» while True:
» » print 'Listening at', s.getsockname()
» » sc, sockname = s.accept()
» » print 'We have accepted a connection from', sockname
» » print 'Socket connects', sc.getsockname(), 'and', sc.getpeername()
» » message = recv_all(sc, 16)
» » print 'The incoming sixteen-octet message says', repr(message)
» » sc.sendall('Farewell, client')
» » sc.close()
» » print 'Reply sent, socket closed'
elif sys.argv[1:] == ['client']:
» s.connect((HOST, PORT))
» print 'Client has been assigned socket name', s.getsockname()
    » s.sendall('Hi there, server')

reply = recv_all(s, 16)
» print 'The server said', repr(reply)
» s.close()
else:
    » print >>sys.stderr, 'usage: tcp_local.py server|client [host]'
```

Στο Κεφάλαιο 2, προσεγγίσαμε το θέμα του bind () πολύ προσεκτικά, δεδομένου ότι η διεύθυνση που παρέχει ως argument της κάνει μια πολύ σημαντική επιλογή: να καθορίζει αν απομακρυσμένους υπολογιστές μπορούν να προσπαθήστε να συνδεθείτε στο διακομιστή μας, ή αν ο διακομιστής μας προστατεύεται από εξωτερικές συνδέσεις και να ενημερώνεστε μόνο από άλλα προγράμματα που τρέχουν στον ίδιο υπολογιστή. Έτσι, Κεφάλαιο 2 ξεκίνησε με ασφαλή καταχωρήσεις πρόγραμμα που χρησιμοποιείται μόνο διεπαφή του localhost του loopback, η οποία πάντα έχει την IP διεύθυνση 127.0.0.1, και στη συνέχεια προχώρησε σε πιο επικίνδυνες καταχωρήσεις που επιτρέπεται hosts οπουδήποτε στο Διαδίκτυο για να συνδεθείτε με τον κωδικό του δείγματος μας.

Εδώ, έχουμε συνδυασμό των δύο δυνατοτήτων σε μια ενιαία λίστα. Από προεπιλογή,αυτός o server code κάνει την ασφαλή επιλογή της σύνδεσης στο 127.0.0.1, αλλά θα μπορεί να παρέχει ένα όρισμα της γραμμής εντολών να συνδεθεί με μία από τις εξωτερικές IP μηχανές-ή θα μπορεί να παρέχει ακόμη ένα κενό string για να δείχνει ότι θα δέχεται συνδέσεις από οποιοδήποτε IP διευθύνση,σε οποιοδήποτε μηχανή μας

Η επιλογή του port number είναι επίσης το ίδιο με αυτό που κάναμε για την θύρα UDP μας στο κεφάλαιο 2 και,πάλι, η συμμετρία μεταξύ του TCP και UDP για το θέμα των port numbers είναι αρκετά κοντά ώστε να μπορεί να απλώς να εφαρμόσει το σκεπτικό που χρησιμοποιήσαμε εκεί για να καταλάβουμε γιατί η ίδια επιλογή έχει χρησιμοποιηθεί εδώ σε αυτό κεφάλαιο.

Έτσι, ποιες είναι οι διαφορές μεταξύ των προηγούμενων προσπαθειών μας με UDP, και αυτό το νέο πρόγραμμα-πελάτη και διακομιστή?

Ο client στην πραγματικότητα φαίνεται ο ίδιος. Δημιουργεί ένα socket,τρέχει connect () με τη διεύθυνση του διακομιστή με τον οποίο θέλει να επικοινωνήσει, και στη συνέχεια είναι ελεύθεροι να στέλνουν και να λαμβάνουν δεδομένα. Να σημειωθεί όμως ότι υπάρχουν αρκετές διαφορές.

Πρώτον, η TCP connect() κλήση-όπως συζητήθηκε πριν από λίγο-δεν είναι το bit of local socket configuration ότι είναι στην περίπτωση του UDP, όπου θέτει απλώς μια προεπιλεγμένη διεύθυνση που χρησιμοποιείται με οποιοδήποτε επακόλουθη send () κλήσεις, και δίνει ένα φίλτρο στα πακέτα που φθάνουν σε μας υποδοχή. Εδώ, connect () είναι μία πραγματική λειτουργία του δικτύου που ξεκινά την τριμερή χειραψία μεταξύ του πελάτη και εξυπηρετητή, έτσι ώστε να είναι έτοιμοι να επικοινωνήσουν. Αυτό σημαίνει ότι connect () μπορεί να αποτύχει, όπως μπορείτε να επιβεβαιώσετε εύκολα από την εκτέλεση αυτού του σεναρίου, όταν ο διακομιστής δεν λειτουργεί:

```
$ python tcp_sixteen.py client
Traceback (most recent call last):
File "tcp_sixteen.py", line 29, in <module>
s.connect((HOST, PORT))
File "<string>", line 1, in connect
socket.error: [Errno 111] Connection refused
```

Δεύτερον, θα δείτε ότι ο TCP client είναι σε ένα πολύ απλούστερο τρόπο από τον πελάτη UDP μας, γιατί δεν χρειάζεται να κάνει καμία πρόβλεψη για τα ελλείποντα στοιχεία. Λόγω των διαβεβαιώσεων που παρέχει το TCP, το μπορεί να κάνει send () δεδομένων χωρίς να ελέγχουν εάν ο απομακρυσμένος το λαμβάνει, και τρέχει recv () χωρίς να χρειάζεται να εξετάσει το ενδεχόμενο της εκ νέου εκπέμπει το αίτημά της. Ο πελάτης μπορεί να είναι βέβαιος ότι η στοίβα δικτύου
θα εκτελέσει όλες τις απαραίτητες αναμετάδοσης για τη λήψη δεδομένων του.

Τρίτον, υπάρχει μια κατεύθυνση στην οποία το πρόγραμμα αυτό είναι στην πραγματικότητα πιο περίπλοκη από ό, τι ο αντίστοιχος UDP κώδικας και αυτό μπορεί να σας εκπλήξει, γιατί με όλες τις εγγυήσεις της ακούγεται όπως το TCP ροές θα ήταν απλούστερο να προγραμματίσεις από ότι με UDP datagrams. Αλλά ακριβώς επειδή το TCP θεωρεί τα εξερχόμενων και εισερχόμενων δεδομένα σας να είναι, απλά,streams, χωρίς αρχή και τέλος, οφείλει να τα χωρίσει σε πακέτα.Και αυτό σημαίνει ότι send () και recv () σημαίνει διαφορετικά πράγματα από ό, τι σήμαινε πριν. Στην περίπτωση του UDP, απλά σήμαινε send this data in a packet" ή "receive a single data packet ", και έτσι κάθε datagram ήταν ατομικό: είτε να επιτύχει ή αποτύχει ως μια ολόκληρη μονάδα.

Σε επίπεδο εφαρμογής, ποτέ δεν βλέπουν τα UDP πακέτα που είναι μόνο half-sent ή half-received, μόνο πλήρως τα άθικτα datagrams παραδίδονται στην εφαρμογή.

Αλλά το TCP μπορεί να χωρίζει τα δεδομένα σε πολλά κομμάτια κατά τη μετάδοση και στη συνέχεια τα επανασυναρμολογήσετε σταδιακά στον τερματικό σταθμό. Αν και αυτό είναι απίθανο με τα μικρά δεκαέξι-οκτάδων μηνύματα στο Πρόραμμα 3-1,ο κώδικάς μας εξακολουθεί να πρέπει να προετοιμαστεί για την πιθανότητα. Ποιες είναι οι συνέπειες του πρωτοκόλλου TCP streams τόσο για send() και recv() κλήσεις;

Όταν κάνουμε ένα TCP send (), η στοίβα δικτύου του λειτουργικό σύστημα μας θα αντιμετωπίσει μία από τις τρεις καταστάσεις:

- Τα δεδομένα μπορεί να γίνει αμέσως αποδεκτά από το σύστημα, είτε επειδή η κάρτα δικτύου είναι ελεύθερη να το μεταδώσει, ή επειδή το σύστημα έχει χώρο για να αντιγράψετε τα δεδομένα σε έναν προσωρινό εξερχόμενη buffer έτσι ώστε το πρόγραμμά σας μπορεί να συνεχίσουν να τρέχει.Σε αυτές τις περιπτώσεις,η send () επιστρέφει αμέσως, και θα επιστρέψει το μήκος του data string,επειδή ολοκλίρο το string έχει μεταδωθεί.
- Μια άλλη πιθανότητα είναι ότι η κάρτα δικτύου είναι απασχολημένη και ότι ο εξερχόμενα buffer για sockets είναι πλήρης και το σύστημα δε μπορεί-ή δεν θα διαθέσει πλέον
  χώρο. Στην περίπτωση αυτή, η προεπιλεγμένη συμπεριφορά του send () είναι απλά να μπλοκάρει, σταματώντας το προγράμμας σας μέχρι τα δεδομένα μπορεί να γίνουν δεκτή.
- Υπάρχει μία τελευταία δυνατότητα: ότι οι εξερχόμενες buffers είναι σχεδόν πλήρης, και έτσι ένα μέρος των data που προσπαθείτε να στείλετε, μπορεί αμέσως να βρίσκονται στην ουρά,αλλά οι υπόλοιποι θα πρέπει να περιμένουν. Στην περίπτωση αυτή, send () και ολοκληρώνει αμέσως επιστρέφει τον αριθμό των bytes που έγιναν αποδεκτές από την αρχή του data string σας, αλλά αφήνει το υπόλοιπο των δεδομένων χωρίς επεξεργασία.

Λόγω αυτής της τελευταίας δυνατότητας, δεν μπορείτε απλά να καλέσετε send () σε ένα stream socket χωρίς να ελέγξετε την τιμή επιστροφής. Αντ 'αυτού, πρέπει να βάλετε μία send () κλήση μέσα σε έναν βρόχο, όπως αυτό, που-στην περίπτωση μίας μερικής μετάδοσης-συνεχίζει προσπαθώντας να στείλει τα υπόλοιπα δεδομένα έως ότου ολόκληρο το string να έχει σταλεί:

```
bytes_sent = 0
while bytes_sent < len(message):
» message_remaining = message[bytes_sent:]
» bytes_sent += s.send(message_remaining)
```

Ευτυχώς, η Python δεν μας υποχρεώνουν να το κάνουμε μόνοι μας αυτό κάθε φορά που έχουμε ένα μπλοκ δεδομένων για αποστολή: το Standard Library socket implementation παρέχει μία friendly Sendall () μέθοδο. Δυστυχώς, δεν υπάρχει ισοδύναμο που να προβλέπεται για την recv () κλήση, παρά το γεγονός ότι μπορεί να επιστρέψει μόνο μέρος των δεδομένων που είναι στο δρόμο από τον πελάτη. Εσωτερικά, η υλοποίηση του λειτουργικού συστήματος του recv () χρησιμοποιεί τη λογική πολύ κοντά με εκείνη που χρησιμοποιείται κατά την αποστολή:

- Εάν δεν υπάρχουν διαθέσιμα data, τότε η recv () μπλοκάρει και το πρόγραμμα σας σταματάει μέχρι τα δεδομένα να φτάσουν.
- Εάν πολλά στοιχεία είναι ήδη διαθέσιμα στον εισερχόμενο buffer, τότε σας δίνονται όσα είναι ο αριθμός των bytes που ζητήσατε recv ().
- Αλλά αν το buffer περιέχει ένα κομμάτι των δεδομένων, αλλά όχι τόσο όσο ζητάτε, τότε
  θα σας επιστραφούν αμέσα ότι συμβαίνει να είναι εκεί, ακόμα κι αν δεν είναι όσα έχετε ζητήσει.
- Αυτός είναι ο λόγος για τον οποίο μας recv () κλήση πρέπει να είναι μέσα σε ένα βρόχο: το λειτουργικό σύστημα δεν έχει τρόπο να γνωρίζει ότι
  αυτό το απλό client και server χρησιμοποιούν σταθερού πλάτους δεκαέξι οκτάδα μηνύματα, και έτσι το σύστημα δεν μπορεί να

Αυτός είναι ο λόγος για τον οποίο η recv () κλήση πρέπει να είναι μέσα σε ένα βρόχο: το λειτουργικό σύστημα δεν έχει τρόπο να γνωρίζει ότι αυτός ο απλός client και server χρησιμοποιούν σταθερού πλάτους δεκαέξι-οκτάδων μηνύματα, και έτσι το σύστημα δεν μπορεί να μαντέψει πότε τα εισερχόμενα δεδομένα θα μπορούν τελικά να προσθέθούν σε αυτό που το προγραμμά σας θεωρεί πλήρες μήνυμα.

Γιατί η πρότυπη βιβλιοθήκη της Python περιλαμβάνει Sendall () αλλά καμία ισοδύναμη για την recv () μέθοδος;Μάλλον επειδή σταθερού μήκους μηνύματα είναι τόσο σπάνιο αυτές τις μέρες. Τα περισσότερα πρωτόκολλα έχουν πολύ πιο περίπλοκοι κανόνες για το πώς μέρος της εισερχόμενης ροής οριοθετείται από μια απλή απόφαση ότι «ητο μήνυμα είναι πάντα 16 bytes.

Έτσι, στα περισσότερα προγράμματα, ο βρόχος που τρέχει η recv () είναι στην πραγματικότητα πολύ πιο περίπλοκη από ό, τι η άλλη στο Πρόγραμμα 3-1, επειδή αυτό πρέπει συχνα να διαβάσει ή να επεξεργάζεται τμήμα του μηνύματος, πριν να μπορεί να μαντέψει πόσα πολλά περισσότερα έρχονται. Για παράδειγμα, ένα αίτημα HTTP αποτελείται συχνά από κεφαλίδες, μια κενή γραμμή και στη συνέχεια , πολλά ακόμη bytes των δεδομένων που αναφέρονται στην κεφαλίδα Content-length. Δεν θα ξέρετε πόσες φορές να συνεχίζετε να τρέχετε recv () μέχρι να έχετε τουλάχιστον λάβει τις κεφαλίδες και στη συνέχεια αναλύεται να βρείτε το μήκος του περιεχομένου!

Deadlock

Ο όρος "deadlock" χρησιμοποιείται για όλα τα είδη των καταστάσεων στην επιστήμη των υπολογιστών, όπου τα δύο προγράμματα, την ανταλλαγή περιορισμένους πόρους, να καταλήξεις να περιμένει ο ένας στον άλλο για πάντα, λόγω της ανεπαρκούς σχεδιασμού. Αποδεικνύεται ότι
μπορεί να συμβεί αρκετά εύκολα, όταν χρησιμοποιείτε το πρωτόκολλο TCP.

Ανέφερα προηγουμένως ότι τυπικά το η στοίβα του TCP χρησιμοποιεί buffers, τόσο ώστε να έχουν κάπου να τοποθετήστε τα εισερχόμενα πακέτα δεδομένων μέχρι μία εφαρμογή είναι έτοιμη να τα διαβάσει, και έτσι ώστε να μπορούν να συλλέγουν τα εξερχόμενα στοιχεία μέχρι το υλικό δικτύου είναι έτοιμη να μεταδώσει μια εξερχόμενη πακέτο. Αυτοι οι buffers είναι συνήθως αρκετά περιορισμένοι σε μέγεθος, και το σύστημα δεν είναι γενικά πρόθυμη να αφήσει τα προγράμματα να συμπληρώσετε όλη την RAM με δεδομένα του δικτύου που δεν έχουν σταλεί. Μετά από όλα, αν το απομακρυσμένο άκρο δεν είναι ακόμη έτοιμο να επεξεργαστεί τα δεδομένα, δεν έχει νόημα να
δαπανά τους πόρους του συστήματος στο άκρο προσπαθεί να παράγει περισσότερο από αυτό.

Αυτός ο περιορισμός δεν θα σας προβληματίσει σε γενικές γραμμές, αν ακολουθήσετε το client-server μοντέλο φαίνεται στο Πρόγραμμα 3-1, όπου κάθε άκρο πάντα διαβάζει ολόκληρο το μήνυμα των συνεργατών της πριν αρχίσει την αποστολή των δεδομένων προς την άλλη κατεύθυνση. Αλλά μπορείτε να τρέξετε στο πρόβλημα πολύ γρήγορα αν σχεδιάζετε ένα πελάτη και εξυπηρετητή που αφήνουν πάρα πολλά δεδομένα, χωρίς να περιμένει κάποια ρύθμιση για αμέσως το διάβασμα.

Ρίξτε μια ματιά στο Πρόγραμμα 3-2 . Εδώ, ο συγγραφέας του διακομιστή έχει κάνει κάτι που είναι πραγματικά αρκετά ευφυής. Η δουλειά του είναι να μετατρέψει μια αυθαίρετη ποσότητα του κειμένου σε κεφαλαία. Αναγνωρίζοντας ότι οι αιτήσεις των πελατών της μπορεί να είναι αυθαίρετα πολλές , και ότι θα μπορούσε κανείς να ξεμείνουμε από μνήμη προσπαθώντασ να διαβάσει ένα ολόκληρο ρεύμα εισόδου πριν προσπαθήσετε να την επεξεργάζεται, ο διακομιστής διαβάζει και επεξεργάζεται μικρά μπλοκ των 1.024 bytes κάθε φορά.

Πρόγραμμα 3–2. *TCP Server and Client That Deadlock*

```
#!/usr/bin/env python
# Foundations of Python Network Programming - Chapter 3 - tcp_deadlock.py
# TCP client and server that leave too much data waiting
import socket, sys
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
HOST = '127.0.0.1'
PORT = 1060
if sys.argv[1:] == ['server']:
» s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
» s.bind((HOST, PORT))
» s.listen(1)
» while True:
» » print 'Listening at', s.getsockname()
» » sc, sockname = s.accept()
» » print 'Processing up to 1024 bytes at a time from', sockname
» » n = 0
» » while True:
```

```
            message = sc.recv(1024)
            if not message:
                break
            sc.sendall(message.upper()) # send it back uppercase
            n += len(message)

        print '\r%d bytes processed so far' % (n,),
        sys.stdout.flush()
    print
    sc.close()
    print 'Completed processing'
elif len(sys.argv) == 3 and sys.argv[1] == 'client' and sys.argv[2].isdigit():
    bytes = (int(sys.argv[2]) + 15) // 16 * 16 # round up to // 16
    message = 'capitalize this!' # 16-byte message to repeat over and over
    print 'Sending', bytes, 'bytes of data, in chunks of 16 bytes'
    s.connect((HOST, PORT))
    sent = 0
    while sent < bytes:
        s.sendall(message)
        sent += len(message)
        print '\r%d bytes sent' % (sent,),
        sys.stdout.flush()
    print
    s.shutdown(socket.SHUT_WR)
    print 'Receiving all the data the server sends back'
    received = 0
    while True:
        data = s.recv(42)
        if not received:
            print 'The first data received says', repr(data)
        received += len(data)
        if not data:
            break
        print '\r%d bytes received' % (received,),
    s.close()
else:
    print >>sys.stderr, 'usage: tcp_deadlock.py server | client <bytes>'
```

Μπορεί να χωριστεί το έργο κάτω τόσο εύκολα, γιατί είναι απλώς προσπαθεί να τρέξει την upper() string μέθοδο σε απλούς ASCII χαρακτήρες-μία λειτουργία που μπορεί να γίνει ξεχωριστά για κάθε μπλοκ εισόδου, χωρίς να ανησυχείτε για τα μπλοκ που ήρθαν πριν ή μετά. Τα πράγματα δεν θα είναι απλά για τον server αν προσπαθήσει να εκτελέσει μία πιο εξελιγμένη λειτουργία string όπως η title(), η οποία μετατρέπει ένα γράμμα σε κεφαλαίο στη μέση μιας λέξης, αν η λέξη έτυχε να χωριστεί στα όρια ενός μπλοκ. Για παράδειγμα, αν ένα μήνυμα έχεις χωριστεί σε 16-byte μπλοκ, τότε τα λάθη θα παρεισφρήσουν σε αυτό, όπως:

```
>>> message = 'the tragedy of macbeth'
>>> blocks = message[:16], message[16:]
```

```
>>> ''.join( b.upper() for b in blocks ) # works fine
'THE TRAGEDY OF MACBETH'
>>> ''.join( b.title() for b in blocks ) # whoops
'The Tragedy Of MAcbeth'
```

Η επεξεργασία κειμένου, καθώς διασπόνται σταθερού μήκους μπλοκ δεν θα εργαστεί επίσης για την κωδικοποίηση UTF-8 Unicode δεδομένα, δεδομένου ότι ένας multi-byte χαρακτήρας θα μπορούσε να χωρίζεται σε ένα όρια ανάμεσα σε δύο από τις binary μπλοκ.Και στις δύο περιπτώσεις, ο διακομιστής θα πρέπει να είναι πιο προσεκτικός και να φέρουν κάποια κατάσταση μεταξύ του ενός μπλοκ δεδομένων και του επόμενου.

Σε κάθε περίπτωση, ο χειρισμός εισόδου ενός μπλοκ σε μλια τέτοια περίοδο όπως αυτή είναι αρκετά έξυπνη για το διακομιστή, ακόμη και αν το 1024 -byte μπλοκ μεγέθους, που χρησιμοποιούνται εδώ για παράδειγμα, είναι πραγματικά μια πολύ μικρή τιμή για τους διακομιστές και τα δίκτυα του σήμερα.Με το χειρισμό των δεδομένων σε κομμάτια και την άμεση αποστολή των απαντήσεων, ο διακομιστής περιορίζει το ποσό των δεδομένων το οποίο πρέπει να κρατήσει στη μνήμη ανά πάσα στιγμή. Διακομιστές σχεδιάζονται έτσι για να μπορούν να χειριστούν εκατοντάδες πελάτες ταυτόχρονα, κάθε αποστολή stream συνολικού μεγέθους gigabyte, χωρίς να φορολογεί τη μνήμη των πόρων του μηχανήματος διακομιστή.

Και, για μικρές ροές δεδομένων, το client και server μοντέλο στο Πρόγραμμα 3-2 φαίνεται να δουλεύει μια χαρά. Αν ξεκινάτε το διακομιστή και στη συνέχεια να εκτελέσετε τον πελάτη με ένα όρισμα της γραμμής εντολών καθορίζοντας ένα μικρό αριθμό από bytes-λένε, ζητώντας του να στείλει 32 bytes δεδομένων (πιο απλά,οποιαδήποτε τιμή θα είναι πολλαπλάσιο του 16 bytes)-τότε θα πάρει πίσω το κείμενό της σε όλα τα κεφαλαία γράμματα:

```
$ python tcp_deadlock.py client 32
Sending 32 bytes of data, in chunks of 16 bytes
32 bytes sent
Receiving all the data the server sends back
The first data received says 'CAPITALIZE THIS!CAPITALIZE THIS!'
32 bytes received
```

Ο διακομιστής (που, παρεμπιπτόντως, πρέπει να εκτελείται στο ίδιο μηχάνημα-αυτό το script χρησιμοποιεί την localhost διεύθυνση IP για να κάνουν το παράδειγμα όσο το δυνατόν απλούστερο) θα αναφέρει ότι πράγματι σε επεξεργασία 32 bytes για λογαριασμό του πρόσφατου πελάτη:

```
$ python tcp_deadlock.py server
Processing up to 1024 bytes at a time from ('127.0.0.1', 46400)
32 bytes processed so far
Completed processing
Listening at ('127.0.0.1', 1060)
```

Έτσι, ο κώδικας αυτός λειτουργεί καλά για μικρές ποσότητες δεδομένων. Στην πραγματικότητα, θα μπορούσε επίσης να εργαστεί για τα μεγαλύτερα ποσότητες? Θα προσπαθήσει να τρέξω τον πελάτη με εκατοντάδες ή χιλιάδες bytes, και να δούμε αν εξακολουθεί να εργάζεται.

Το πρώτο παράδειγμα ανταλλαγής δεδομένων, σας δείχνει τη συμπεριφορά του recv () η οποία έχει περιγράφει προηγουμένως: ακόμη και αν ο διακομιστής ζήτησε 1.024 bytes να λάβει, recv (1024) ήταν στην ευχάριστη θέση να επιστρέψει μόνο 16 bytes, αν αυτή ήταν η ποσότητα των δεδομένων που έγιναν διαθέσιμα και δεν χρειάζονται περαιτέρω δεδομένα να φθάσουν ακόμη από τον πελάτη.

Αλλά αν δοκιμάσετε ένα αρκετά μεγάλο ποσό, τότε έρχεται η καταστροφή! Δοκιμάστε να χρησιμοποιήσετε τον πελάτη να στείλει ένα πολύ μεγάλο stream δεδομένων, ας πούμε, ένα ύψους ένα gigabyte:

$ python tcp_deadlock.py client 1073741824

Θα δείτε τόσο ο πελάτης όσο και ο διακομιστής ενημερώνουν τα παράθυρα τερματικού τους, καθώς εσυ ενημερώνεσαι με την ποσότητα των δεδομένων που έχουν διαβιβαστεί και ληφθεί. Οι αριθμοί θα σκαρφαλώνουν και σκαρφαλώνουν μέχρι, εντελώς ξαφνικά, και οι δύο συνδέσεις παγώσει. Στην πραγματικότητα, αν δείτε προσεκτικά, θα δείτε το διακομιστή να σταματάει πρώτα και στη συνέχεια ο πελάτης θα παραλύσει σύντομα

```
$ python tcp_deadlock.py server
Listening at ('127.0.0.1', 1060)
Processing up to 1024 bytes at a time from ('127.0.0.1', 46411)
602896 bytes processed so far
```

Και ο πελάτης έχει παγώσει περίπου στα 100.000 bytμπροστά γράφοντας η εξερχόμενη ροή του δεδομένων:

```
$ python tcp_deadlock.py client 1073741824
Sending 1073741824 bytes of data, in chunks of 16 bytes
734816 bytes sent
```

Γιατί και οι δύο πελάτης και διακομιστής έχουν ανακοπεί;

Η απάντηση είναι ότι buffer εξόδου του server και ο buffer εισόδου του πελάτη και οι δύο τελικά γεμίσει, και TCP έχει χρησιμοποιηθεί το παράθυρο του πρωτοκόλλου προσαρμογής για να επισημάνει το γεγονός αυτό και να σταματήσει το socket από την αποστολή περισσότερων δεδομένων που θα πρέπει να απορριφθεί και στη συνέχεια εκ νέου αποστολή.

Σκεφτείτε τι συμβαίνει σε κάθε μπλοκ δεδομένων των ταξιδιών. Ο πελάτης στέλνει με Sendall (). Στη συνέχεια, η διακομιστή δέχεται με recv (), το επεξεργάζεται, και στη συνέχεια μεταδίδει την capitalized έκδοση του πίσω από με Sendall() κλήση. Και μετά τι; Λοιπόν, τίποτα! Ο πελάτης δεν εκτελεί οποιαδήποτε recv () κλήσεις-ενώ εξακολουθεί να έχει τα δεδομένα να στείλει, έτσι ώστε όλο και περισσότερα

capitalized δεδομένα στηρίζει, μέχρι το λειτουργικό σύστημα να μην είναι πρόθυμο να δεχθεί οποιαδήποτε περισσότερο.

Αλλά πώς, τότε, οι πελάτες του δικτύου και servers επεξεργάζονται μεγάλες ποσότητες δεδομένων χωρίς να εισέρχονται αδιέξοδο; Υπάρχουν, στην πραγματικότητα, δύο πιθανές απαντήσεις: είτε να χρησιμοποιήσετε τις επιλογές socket για να απενεργοποιήσετε το blocking, έτσι ώστε οι κλήσεις όπως send () και recv () να επιστρέφουν αμέσως σε περίπτωση που διαπιστώσουν ότι δεν μπορεί να στείλει οποιοδήποτε δεδομένα ακόμα.

Ή, τα προγράμματα μπορούν να χρησιμοποιήσουν μία από τις πολλές τεχνικές για την επεξεργασία δεδομένων από περισσότερες εισροές σε έναν χρόνο,είτε με τη διάσπαση σε ξεχωριστά threads ή processes-είτε με ένα tasked με την αποστολή δεδομένων μέσα σε ένα socket,ίσως, και μια άλλο tasked με την ανάγνωση των δεδομένων πίσω-ή εκτελώντας κλήσεις του λειτουργικού συστήματος όπως select () ή poll() που να τους αφήσουμε να περιμένουν στο απασχολημένο εξερχόμενο και εισερχόμενο socket ταυτόχρονα, και ανταποκρίνονται σε όποιο είναι έτοιμο.

Τέλος, σημειώστε προσεκτικά ότι η ανωτέρω σενάριο δεν μπορεί ποτέ να συμβεί όταν χρησιμοποιούν το UDP! Αυτό επειδή το UDP δεν εκτελεί έλεγχο ροής. Εάν περισσότερα datagrams φτάνουν,αυτά μπορούν να επεξεργαστούν , τότε το UDP μπορεί να απορρίψει απλώς μερικά από αυτά, και το αφήνουμε μέχρι την εφαρμογή να ανακαλύψει ποια λείπουν.

Using TCP Streams like Files

Δεδομένου ότι το πρωτόκολλο TCP υποστηρίζει ροές δεδομένων, υποστιρίζει επίσης την
την ανάγνωση και τη γραφή κανονικών αρχείων ως θεμελιώδη δραστηριότητα. Python κάνει πολύ καλή δουλειά κρατώντας αυτές τις έννοιες χωριστά:τα file objects μπορεί να read() και write (),sockets μπορεί να send () και recv (), και κανένα αντικείμενο δεν μπορεί να κάνει και τα δύο. Αυτός είναι πραγματικά ένας εννοιολογικός διαχωρισμός από αυτή που επιτυγχάνεται με το υποκείμενο POSIX περιβάλλον, το οποίο επιτρέπει σε ένα C προγραμματιστής να κάνει κλήση της read () και write () σε ένα socket αδιακρίτως σαν να ήταν ένα normal file descriptor!

Αλλά μερικές φορές θα θέλετε να αντιμετωπίζει ένα socket σαν ένα κανονικό αρχείο αντικείμενο της Python-συχνά γιατί θέλετε να το περάσει στον κώδικα, όπως ότι από τα πολλά Python modules όπως pickle, JSON, και zlib,τα οποία μπορούν να διαβάζουν και να γράφουν δεδομένα απευθείας από ένα αρχείο. Για το σκοπό αυτό, η Python παρέχει μία makefile () μέθοδο για κάθε socket που επιστρέφει ένα αντικείμενο αρχείο της Python που ζητά recv () και send ():

```
>>> import socket
>>> s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
>>> hasattr(s, 'read')
False
>>> f = s.makefile()
>>> hasattr(f, 'read')
```

True

Sockets, σαν κανονικό αρχείο της Python, έχουν επίσης μια μέθοδο fileno () που σας επιτρέπει να ανακαλύψετε το file descriptor number σε περίπτωση που χρειαστεί να τα παράσχουν σε χαμηλόγ επίπεδοy κλήσεις.

ΚΕΦΑΛΑΙΟ 6^{Ο}

DNS

Έχοντας περάσει τα τελευταία δύο κεφάλαια στην μαθαίνοντας τα βασικά των UDP και TCP, αυτά τα δυο με τα οποά γίνονται μεγάλες μεταφορές δεδομένων σε δίκτυα IP, είναι καιρός για μας να συζητηθούν δύο μεγαλύτερα ζητήματα που πρέπει να θα πρέπει να αντιμετωπιστούν ανεξάρτητα από το ποια μεταφορά χρησιμοποιείτε. Σε αυτό το κεφάλαιο, θα συζητήσουμε το θέμα των διευθύνσεων δικτύου και θα περιγράφει η κατανομή με την επιτρέπετε τα ονόματα να επιλυθούν διευθύνσεις IP.

Hostnames and Domain Names

Προτού αναφερθώ σε αυτό το θέμα,θα πρέπει να διευκρινηστούν κάποιοι όροι που παίζουν σημαντικό ρόλο στη συζήτηση που θα ακολουθήσει:

- *Top-level domain (TLD):* Αυτές είναι μερικά εκατοντάδες string όπως com, net, org, gov και τα οποία, μαζί με τους κωδικούς χωρών όπως η Γερμανία και Ηνωμένο Βασίλειο, αποτελούν το σύνολο των πιθανών επιθημάτων για έγκυρα domain names. Συνήθως, κάθε TLD έχει το δικό του σύνολο των servers και τη δική του οργάνωση που είναι υπεύθυνη για τη χορήγηση ιδιοκτησίας σε τομείς κάτω από το TLD.
- *Domain name:* Αυτό είναι το όνομα που προσθέτει μια επιχείρηση ή ένας οργανισμός σαν κατάληξη στα site της και φιλοξενεί στο Διαδίκτυο, όπως python.org, imdb.com, ή bbc.co.uk. Κοστίζει συνήθως μια ετήσια αμοιβή για να αγοράσει κάποιος ένα domain name, αλλά κατέχοντας το σας δίνει το δικαίωμα να δημιουργήσετε όσες hostnames θέλετε.

Fully qualified domain name: Η FQDN ονομάζει μια ιστοσελίδα στο Διαδίκτυο ή φιλοξενεί
προσαρτόντας το πλήρες domain name της επιχείρησής του στο όνομα μίας συγκεκριμένης
μηχανής του εν λόγω οργανισμού. Παράδειγμα FQDN είναι τα gnu.org και asaph.rhodesmill.org. Εάν ένα domain name είναι "πλήρως αναγνωρισμένο» δεν εξαρτώνται στο να έχουν κάποιο συγκεκριμένο αριθμό συστατικών, μπορεί να έχει δύο, τρεις,τέσσερις ή περισσότερα ονόματα τα οποία διαχωρίζονται με τελεία.Αυτό που καθιστά ένα FQDN είναι ότι τελειώνει με ένα TLD και ως εκ τούτου, θα λειτουργήσει από οπουδήποτε. Μπορείτε να χρησιμοποιείτε συχνά ακριβώς το

όνομα athena, αν είστε συνδεδεμένοι σε ένα δίκτυο MIT, αλλά και από οπουδήποτε αλλού στην κόσμο, θα πρέπει να επωφελούνται πλήρως το όνομα και να καθορίσετε athena.mit.edu.

Hostname: Ο όρος αυτός, δυστυχώς, είναι ασαφής! Μερικές φορές αυτό σημαίνει ότι το όνομα που μια μηχανή μπορεί να εκτυπώνει όταν συνδεθεί με αυτό, όπως asaph ή athena. Αλλά μερικές φορές οι άνθρωποι αντί να εννοούν αυτό,εννοούν το FQDN όταν λένε "the hostname "

- Σε γενικές γραμμές, ένα FQDN μπορεί να χρησιμοποιηθεί για να προσδιορίσει ένα host από οπουδήποτε αλλού στο Διαδίκτυο.Bare hostnames, αντιθέτως, λειτουργούν ως σχετικά ονόματα μόνο αν είστε ήδη στο εσωτερικό του οργανισμού και χρησιμοποιώντας τα δικά τους nameservers τους, για την επίλυση ονομάτων στην επιφάνεια εργασίας, φορητό υπολογιστή σας, ή διακομιστή.Έτσι, η athena θα πρέπει να δουλεύει ως συντομογραφία για athena.mit.edu αν είστε πραγματικά στο campus του MIT, αλλά δεν θα λειτουργήσει εάν είστε οπουδήποτε αλλού στον κόσμο-εκτός και αν έχετε ρυθμίσει το σύστημά σας να δοκιμάζει πάντα το MIT hostnames πρώτα,το οποίο θα ήταν ασυνήθιστο, αλλά ίσως είναι για το προσωπικό τους ή κάτι.

Ένα σκίτσο πώς το DNS λειτουργεί

Το Σύστημα Ονομάτων Τομέα, το DNS είναι ένα σύστημα με το οποίο τα εκατομμύρια των hosts του Internet συνεργάζετε για να απαντήσει στο ερώτημα του ποια hostnames επιλύουν ποιες διευθύνσεις IP. Το DNS είναι πίσω από το γεγονός ότι μπορείτε να πληκτρολογήστε python.org στον browser σας αντί πάντα να χρειάζεται να θυμάστε 82.94.164.162 αν είναι με το IPv4, ή 2001:888:2000: d:: a2 εάν απολαμβάνετε ήδη το IPv6.

THE DNS PROTOCOL

Purpose: **Turn hostnames into IP addresses**
Standard: **RFC 1035 (1987) and subsequent**
Runs atop: **TCP/IP** and **UDP/IP**
Default port: **53**
Libraries: PyDNS, dnspython

Τα μηνύματα τα οποία οι υπολογιστές στέλνουν για να εκτελούν αυτή τη μορφή ανάλυσης του "DNS πρωτοκόλλου," η οποία λειτουργεί σε ένα ιεραρχικό τρόπο. Αν ο τοπικός υπολογιστή σας και nameserver δεν μπορεί να επιλύσει ένα hostname επειδή δεν είναι ούτε τοπικα στον οργανισμό σας ούτε έχετε δει πρόσφατα ώστε να

εξακολουθεί να είναι στην
cache ονομάτων, τότε το επόμενο βήμα είναι να ρωτήσουν ένα από τους παγκοσμίου κορυφαίους nameservers για να διευκρινίσουν ποιες μηχανές είναι υπεύθυνες για τον domain που χρειάζετε σχετικά. Μόλις οι IP διευθύνσεις τους διαπιστωθούν, μπορούν στη συνέχεια να ερωτηθούν για το ίδιο domain name .

Θα πρέπει πρώτα να κάνουμε ένα βήμα πίσω για μια στιγμή και να δούμε πώς αυτή η λειτουργία θέτετε σε κίνηση.

Για παράδειγμα, σκεφτείτε το domain name www.python.org. Αν ο web browser σας πρέπει να το γνωρίζουν αυτήν την διεύθυνση, τότε ο browser τρέχει μια κλήση όπως getaddrinfo ()για να ζητήσει από το λειτουργικό σύστημα να επιλύσει αυτό το όνομα. Το σύστημά σας θα γνωρίζει είτε ότι βρίσκεται σε λειτουργία ένας nameserver, ή ότι το δίκτυο στο οποίο είναι συνδεδεμένο παρέχει name service. Nameserver πληροφορίες αυτές τις μέρες συνήθως μαθαίνονται αυτόματα μέσω του DHCP, είτε σε εταιρικά γραφεία, στα σχολεία, στα ασύρματα δίκτυα, ή στις σπιτικές συνδέσεις DSL. Σε άλλες περιπτώσεις, ο DNS server θα διαμορφώσει τις IP διευθύνσεις με το χέρι όταν ένας διαχειριστής του συστήματος κάνει εγκατάσταση το μηχάνημά σας. Είτε έτσι είτε αλλιώς, οι διακομιστές DNS πρέπει να είναι τυπικά να καθορίζονται σαν διευθύνσεις IP, αφού προφανώς δεν μπορείτε να χρησιμοποιήσετε το ίδιο DNS για να τους βρείτε!

Μερικές φορές οι άνθρωποι είναι δυσαρεστημένοι με τη συμπεριφορά του DNS της υπηρεσίας ISP ή την απόδοσή του και να επιλέγουν να ρυθμίσετε τις παραμέτρους ενός third-party διακομιστή DNS της δικής τους επιλογής, όπως οι διακομιστές σε 8.8.8.8 και 8.8.4.4 διοικούνται από την Google. Και σε ορισμένες πιο σπάνιες περιπτώσεις, ο τοπικός DNS domain nameservers είναι γνωστός μέσα από κάποια άλλη ομάδα ονομάτων που είναι σε χρήση από τον υπολογιστή, όπως το WINS Windows name srrvice. Αλλά με τον ένα ή τον άλλο τρόπο, ένας DNS server πρέπει να προσδιορίζετε για την επίλυση ονομάτων για να συνεχίσει.

Ελέγχοντας τον DNS για το hostname δεν είναι στην πραγματικότητα το πρώτο πράγμα που ένα λειτουργικό σύστημα που κάνει συνήθως όταν πραγματοποιείτε μια κλήση, όπως getaddrinfo ()-στην πραγματικότητα,επείδη κάνοντας ένα ερώτημα DNS μπορεί να είναι χρονοβόρα,είναι συχνά η τελευταία επιλογή! Ανάλογα με τις hosts εγγραφές στο / etc / nsswitch.conf σας, αν είστε σε ένα
POSIX κουτί, ή αλλιώς ανάλογα με τις ρυθμίσεις των Windows πίνακα ελέγχου σας, θα μπορούσε να υπάρχει ένα ή περισσότερα άλλα μέρη όπου το λειτουργικό σύστημα ψάχνει πρώτα πριν την στροφή στο DNS. Στο Ubuntu φορητό υπολογιστή μου, για παράδειγμα, το / etc / hosts αρχείο ελέγχεται πρώτα για κάθε κάθε hostname lookup, τότε ένα εξειδικευμένο πρωτόκολλο που ονομάζεται multicast DNS χρησιμοποιείται, αν είναι δυνατόν και μόνο αν αυτό αποτύχει ή δεν είναι διαθέσιμος ο DNS να απαντήσει στο ερώτημα hostname.

Για να συνεχίσουμε το παράδειγμά μας, φανταστείτε ότι το www.python.org όνομα δεν έχει, στην πραγματικότητα πρόσφατα ερωτηθεί για να είναι σε οποιδήποτε τοπική μνήμη cache στο μηχάνημα όπου τρέχετε web browser σας.Στην περίπτωση αυτή, ο υπολογιστής θα εξετάσει το τοπικό DNS server και, συνήθως, στείλτε ένα πακέτο αίτημα DNS πάνω από το UDP.

Τώρα το ερώτημα είναι στα χέρια ενός πραγματικού διακομιστή DNS! Για το υπόλοιπο αυτής της συζήτησης, εμείς θα την αποκαλούμε "DNS server σας",αλλά, φυσικά, ο διακομιστής ανήκει πιθανότατα σε κάποιον άλλον, όπως τον εργοδότη σας ή τον ISP σας ή το Google!

Η πρώτη πράξη του DNS server σας θα είναι να ελέγξει τη δική του μνήμη cache του για πρόσφατα domain names για να δει εάν το www.python.org έχει ήδη ελεγχθεί

από κάποιο άλλο μηχάνημα που εξυπηρετούνται από το διακομιστή DNS στα τελευταία λεπτά ή ώρες. Αν η εγγραφή είναι τωρινή και δεν έχει ακόμη λήξει και ο ιδιοκτήτης του domain name μπορεί να επιλέξει χρονικό όριο λήξης του, διότι ορισμένες οργανώσεις θέλουν να αλλάζουν τις διευθύνσεις IP γρήγορα αν χρειαστεί, ενώ άλλοι είναι στην ευχάριστη θέση να έχουν παλιά διευθύνσεις IP στη παγκόσμια DNS cache-τότε μπορεί να επιστραφεί αμέσως. Αλλά ας φανταστούμε ότι είναι το πρωί και ότι είστε το πρώτο άτομο στο γραφείο σας που προσπαθεί να επικοινωνήσει με το www.python.org σήμερα, και έτσι ο διακομιστής DNS πρέπει να ξεκινήσει από το μηδέν.

Ο DNS server σας θα αρχίσει τώρα μια αναδρομική διαδικασία ρωτώντας για το www.python.org στην κορυφή της DNS server ιεραρχία του κόσμου: η «root-level" nameservers που γνωρίζουν όλα τα top-level domains (TLD) όπως τα . Com,. Org,. Net, και όλα τα domains των χωρών και γνωρίζουν τις ομάδες των servers που είναι υπεύθυνες για το καθένα. Nameserver λογισμικό έρχεται συνήθως με τις διευθύνσεις IP από αυτές τις υψηλού επιπέδου χτισμένες σε διακομιστές, για να λύσει το πρόβλημα κάνουν εκκίνηση για το πώς μπορείτε να βρείτε οποιοδήποτε domain nameservers προτού πράγματι συνδέονται με το σύστημα domain name! Με αυτήν την πρώτη UDP round-trip, ο διακομιστής DNS θα να μάθουν (αν δεν το ξέρατε ήδη από ένα άλλο πρόσφατο ερώτημα), το οποίο διακομιστές θα κρατήσουν το πλήρες ευρετήριο. org domain.

Τώρα, μια δεύτερη αίτηση DNS θα γίνει, αυτή τη φορά σε ένα από τους . org servers, οι οποίοι ζητούν ποιοι στη γη τρέχουν τον python.org domain. Μπορείτε να μάθετε ποιοι είναι οι top-level servers που γνωρίζουν για έναν domain τρέχοντας την εντολή whois της γραμμής εντολών του προγράμματος σε ένα σύστημα POSIX, ή να χρησιμοποιήσετε ένα από τα πολλέσ "whois" που είναι διαθέσιμες διαδικτυακά online:

Google it!

Whois python.org

Σε ένα από τα αποτέλεσμα πήρα την απάντηση:

REGISTRY WHOIS FOR PYTHON.ORG

Domain Name: **python.org**
Updated: 1 second ago - Refresh

Registrar: Network Solutions LLC (R63-LROR)
Status: CLIENT TRANSFER PROHIBITED

Expiration Date: 2016-03-28 05:00:00
Creation Date: 1995-03-27 05:00:00
Last Update Date: 2012-04-09 15:18:17

Name Servers:

ns1.p11.dynect.net
ns2.p11.dynect.net
ns3.p11.dynect.net
ns4.p11.dynect.net
See python.org DNS Records

Information Updated: Tue, 8 May 2012 14:18:17 UTC

Και αυτό παρέχει την απάντηση μας! Όπου κι αν βρίσκεστε στον κόσμο, ο DNS σας ζητήσει για οποιοδήποτε hostname εντός python.org πρέπει να περάσει σε ένα από τους DNS servers που αναφέρεται στο εν λόγω εγγραφή. Φυσικά,όταν DNS server σας κάνει αυτό το αίτημα, σε ένα κορυφαίο επίπεδο domain nameserver,αυτό δεν παίρνει στην πραγματικότητα μόνο δύο ονόματα, όπως αυτά που μόλις δόθηκαν,αντ 'αυτού, δίνονται επίσης οι IP διευθύνσεις τους, ώστε να μπορεί να επικοινωνήσει απ 'ευθείας χωρίς να υποστούν ένα νέο γύρο αναζήτησης DNS.

Ο DNS server σας έχει τώρα ολοκληρώσει την συνομιλία τόσο με τον root-level DNS server και με το top-level .org DNS server, και μπορούν να επικοινωνούν απευθείας με ns1.p11.dynect.net ns2.p11.dynect.net ns3.p11.dynect.net ns4.p11.dynect.netγια να ρωτήσει κάτι σχετικά με το python.org domain-και, στην πραγματικότητα, θα προσπαθήσει συνήθως ένα από αυτά και στη συνέχεια αν δεν είναι διαθεσιμό τότε θα προσπαθήσει με τους άλλους. Αυτό αυξάνει τις πιθανότητες να πάρει μια απάντηση, αλλά, φυσικά, μπορεί να
αυξάνει και το χρόνο που κάθεστε κοιτάζοντας web browser σας, μέχρι η σελίδα να εμφανιστεί!

Ανάλογα με το πώς python.org έχει ρυθμίσει τουσ nameservers του, ο διακομιστής DNS μπορεί να κάνει αίτηση για μόνο ένα ακόμη ερώτημα για να πάρετε απάντηση, ή μπορεί να χρειαστούν αρκετές, αν ο οργανισμός είναι μεγάλος με πολλά τμήματα και υπο-τμήματα που όλοι έχουν τους δικούς τους διακομιστές DNS στους οποίους τα αιτήματα πρέπει να είναι κατ 'εξουσιοδότηση. Στην περίπτωση αυτή, το ερώτημα www.python.org μπορούν να απαντηθεί άμεσα από τους τέσσερις διακομιστές μόνο ονομάζοντας και ο διακομιστής DNS μπορεί να επιστρέψει ένα πακέτο UDP στον browser σας λέγοντας το ποια διεύθυνση IP ανήκουν σε αυτό το hostname.

Σημειώστε ότι αυτή η διαδικασία απαιτούσε τέσσερις διαφορετικές network round-trips . Το μηχάνημά σας υπέβαλε μία αίτηση και πήρε απάντηση από το δικό σας DNS server, και προκειμένου να δοθεί απάντηση στο αίτημα, ο διακομιστής DNS είχε να κάνει μια αναδρομική ερώτηση που αποτελούνταν από τρεις διαφορετικές round-trips σε άλλους servers. Δεν υπάρχει αμφιβολία ότι ο web browser κάθεται εκει περιμένοντας όταν εισάγετε ένα domain name για πρώτη φορά!

Γιατί να ΜΗΝ χρησιμοποιήσουμε τον DNS

Η ανωτέρω εξήγηση ενός τυπικού ερωτήματος DNS,κατέστησε σαφές ότι το λειτουργικό σας σύστημα είναι κάνει πολλά για εμάς, όταν χρειάζεστε για ένα hostname.Για το λόγο αυτό,ότι,συνιστάτε αν δεν είναι απολύτως απαραίτητο να μιλήσετε με το DNS για κάποιο πολύ συγκεκριμένο λόγο, μπορείτε πάντα να βασίζεστε στη getaddrinfo () ή κάποιο άλλο σύστημα που υποστηρίζεται από μηχανισμό για την επίλυση hostnames.Δείτε τα οφέλη:

- Το DNS δεν είναι συχνά ο μόνος τρόπος που ένα σύστημα παίρνει το πληροφορίες ονόματοσ. Εάν η εφαρμογή σας τρέχει και προσπαθεί να χρησιμοποιεί DNS από μόνη της ως πρώτη επιλογή του για την επίλυση ένος domain name , τότε οι χρήστες θα παρατηρήσουν ότι ορισμένα ονόματα υπολογιστών τα οποία λειτουργούν οπουδήποτε αλλού στο σύστημά σας- στον browser τους, μέσα στο file share names -ξαφνικά δεν λειτουργούν όταν χρησιμοποιούν την αίτησή σας,επειδή δεν αναβάλουν τους μηχανισμούς, όπως WINS ή / etc / hosts όπως το ίδιο το λειτουργικό σύστημα κάνει.
- Η τοπική μηχανή έχει πιθανότατα ένα χώρο προσωρινής αποθήκευσης των διερωτημένων πρόσφατα για domain names τα οποία μπορεί ήδη να γνωρίζει για το host του οποίου την διεύθυνση IP χρειάζεστε. Αν προσπαθήσετε να μιλήσετε με το
  DNS για να απαντήσει στο ερώτημά σας, θα πρέπει να επαναλάβετε την διαδικασία που έχει ήδη γινει.
- Το σύστημα στο οποίο το python script τρέχει ήδη γνωρίζει για την τοπικό domain nameserver, είτε χάρη σε χειροκίνητη παρέμβαση από το διαχειριστή του συστήματος ή από ένα network configuration protocol, όπως DHCP στο γραφείο, στο σπίτι,. Για να ξεκινήσουμε το DNS μέσα από στο Python πρόγραμμά σας, θα πρέπει να γίνει γνωστό πώς να ρωτάτε το συγκεκριμένο λειτουργικό σύστημά σας για αυτές τις πληροφορίες.
- Αν δεν χρησιμοποιήσετε το τοπικό DNS server, τότε δεν θα είστε σε θέση να επωφεληθήτε από την δική του cache που θα εμποδίσει την εφαρμογή σας και άλλες εφαρμογές που τρέχουν στο ίδιο δίκτυο από τα επαναλαμβανόμενα αιτήματα σχετικά με ένα hostname το οποίο χρησιμοποιείται συχνά στην τοποθεσία σας.
- Από καιρό σε καιρό, γίνονται αναπροσαρμογές στις υποδομές DNS παγκοσμίως, και
  οι βιβλιοθήκες του συστήματος και τα daemons σταδιακά ενημερώνονται για να φιλοξενήσει αυτό. Εάν το πρόγραμμά σας κάνει raw DNS κλήσεις, τότε θα πρέπει να ακολουθήσει αυτές οι αλλαγές προσωπικά και να βεβαιωθείτε ότι ο κώδικας σας μένει up-to-date με τις πρόσφατες αλλαγές στο TLD server IP διευθύνσεις , που αφορούν συμβάσεις
  διεθνώς .

Τέλος, σημειώστε ότι η Python δεν είναι με τις εγκαταστάσεις DNS ενσωματωμένες στην πρότυπη βιβλιοθήκη.Αν πρόκειτε να μιλήσετε με DNS χρησιμοποιώντας Python, τότε πρέπει να επιλέξετε και να μάθετε μια third-party βιβλιοθήκη για να το κάνει.

Γιατί να χρησιμοποιήσουμε τον DNS

Υπάρχει, όμως, ένας νόμιμος λόγος να πραγματοποιήσετε μια κλήση DNS από την Python: επειδή είστε ένας mail server ,ή τουλάχιστον ένας πελάτης προσπαθεί να στείλει mail απευθείας στους παραλήπτες σας χωρίς να χρειάζεται να τρέξει ένα local mail relay, και θέλετε να αναζητήσετε τις εγγραφές MX που σχετίζονται με ένα domain έτσι ώστε να μπορείτε να βρείτε το σωστό mail server για τους φίλους στο @example.com.

Λοιπόν,για να ρίξουμε μία ματιά σε μία από τις third-party DNS βιβλιοθήκες για Python.Θα επικεντρωθώ στη περισσότερο δημοφιλή διανομή,pydns,η οποία κατάγετε από μία DNS module,η οποία γράφτηκε από τον Guido van Rossum Κάνει ένα DNS πακέτο στη διάθεσή σας για την εισαγωγή. Ανταγωνιστής της, η κατανομή dnspython, δημιουργεί ένα lower-case DNS πακέτο.

Σημειώστε ότι ούτε το project παρέχει κώδικα που ξέρει πώς να "ξεκινήσει από το μηδέν" και να αρχίσει ένα ερώτημα με μια αναζήτηση στη ρίζα των domain nameservers of Internet! Αντ 'αυτού, κάθε βιβλιοθήκη χρησιμοποιεί τη δικά της για να βρει τους domain nameservers των Windows ή POSIX λειτουργικό σύστημα που χρησιμοποιούν σήμερα, και στη συνέχεια ζητούν από εκείνους τους servers να κάνουν αναδρομικά ερωτήματα για λογαριασμό της.

Πρόγραμμα 4–3. *A Simple DNS Query Doing Its Own Recursion*

```
#!/usr/bin/env python
#  dns_basic.py
# Basic DNS query
import sys, DNS
if len(sys.argv) != 2:
» print >>sys.stderr, 'usage: dns_basic.py <hostname>'
» sys.exit(2)
DNS.DiscoverNameServers()
request = DNS.Request()
for qt in DNS.Type.A, DNS.Type.AAAA, DNS.Type.CNAME, DNS.Type.MX,
DNS.Type.NS:
» reply = request.req(name=sys.argv[1], qtype=qt)
» for answer in reply.answers:
» » print answer['name'], answer['classstr'], answer['typename'], \
» » » repr(answer['data'])
```

Τρέχοντας αυτό το πρόγραμμα για κάποιο site πχ google.com θα μάθουμε κάποια πράγματα για το DNS

Επιλύοντας Mail Domains

Η επίλυση ένα e-mail domain είναι μια πολύ θεμιτή χρήση τωνraw DNS στα περισσότερα προγράμματα Python. Οι κανόνες για αυτήν την επίλυση,ορίζονται στο RFC 5321. Είναι,εν συντομία, ότι εάν υπάρχουν εγγραφές MX, τότε θα πρέπει να προσπαθήσει να επικοινωνήσει με τους διακομιστές SMTP, και να επιστρέψει ένα σφάλμα στο χρήστη αν καμία από αυτές δεν θα δεχτεί το μήνυμα. Αν, αντίθετα, δεν υπάρχουν MX εγγραφές, αλλά ένα A(Ipv4) ή AAAA (Ipv6) παρέχετε για τον domain, τότε έχετε τη δυνατότητα να δοκιμάσετε μια SMTP σύνδεση σε αυτή τη διεύθυνση. Αν δεν υπάρχει εγγραφή, αλλά μία CNAME έχει καθοριστεί, τότε το domain name που αυτό παρέχει πρέπει να αναζητηθούν για MX ή A εγγραφές χρησιμοποιώντας τους ίδιους κανόνες.

Πρόγραμμα 4–4. *Resolving an E-mail Domain Name*

```
#!/usr/bin/env python
# dns_mx.py
# Looking up a mail domain - the part of an email address after the `@`
import sys, DNS
if len(sys.argv) != 2:
» print >>sys.stderr, 'usage: dns_basic.py <hostname>'
» sys.exit(2)
def resolve_hostname(hostname, indent=0):
» """Print an A or AAAA record for `hostname`; follow CNAMEs if necessary."""
» indent = indent + 4
» istr = ' ' * indent
» request = DNS.Request()
» reply = request.req(name=sys.argv[1], qtype=DNS.Type.A)
» if reply.answers:
» » for answer in reply.answers:
» » » print istr, 'Hostname', hostname, '= A', answer['data']
» » return
» reply = request.req(name=sys.argv[1], qtype=DNS.Type.AAAA)
» if reply.answers:
» » for answer in reply.answers:
» » » print istr, 'Hostname', hostname, '= AAAA', answer['data']
» » return
» reply = request.req(name=sys.argv[1], qtype=DNS.Type.CNAME)
» if reply.answers:

» cname = reply.answers[0]['data']
» » print istr, 'Hostname', hostname, 'is an alias for', cname
» » resolve_hostname(cname, indent)
» » return
» print istr, 'ERROR: no records for', hostname
def resolve_email_domain(domain):
» """Print mail server IP addresses for an email address @ `domain`."""
```

```
» request = DNS.Request()
» reply = request.req(name=sys.argv[1], qtype=DNS.Type.MX)
» if reply.answers:
» » print 'The domain %r has explicit MX records!' % (domain,)
» » print 'Try the servers in this order:'
» » datalist = [ answer['data'] for answer in reply.answers ]
» » datalist.sort() # lower-priority integers go first
» » for data in datalist:
» » » priority = data[0]
» » » hostname = data[1]
» » » print 'Priority:', priority, ' Hostname:', hostname
» » » resolve_hostname(hostname)
» else:
» » print 'Drat, this domain has no explicit MX records'
» » print 'We will have to try resolving it as an A, AAAA, or CNAME'

» » resolve_hostname(domain)
DNS.DiscoverNameServers()
resolve_email_domain(sys.argv[1])
```

Φυσικά, η εφαρμογή του resolve_hostname () που παρουσιάζεται εδώ είναι μάλλον εύθραυστη, δεδομένου ότι θα πρέπει να πραγματικά μια δυναμική προτίμηση μεταξύ των A και AAAA εγγραφών με βάση το εάν ο τρέχον host είναι συνδεδεμένος με ένα IPv4 ή σε ένα δίκτυο IPv6.

Μια πραγματική εφαρμογή mail server , αντί να εκτυπώνει τις mail server διευθύνσεις , θα προσπαθεί να παραδώσει mail σε αυτούς.Μπορούμε να δούμε ότι python.org αυτή τη στιγμή έχει όμως μόνο μία διεύθυνση ip mail sever:

```
$ python dns_mx.py python.org
The domain 'python.org' has explicit MX records!
Try the servers in this order:
Priority: 50 Hostname: mail.python.org
» Hostname mail.python.org = A 82.94.164.162
```

Επισής για να δούμε την IANA

```
$ python dns_mx.py iana.org
The domain 'iana.org' has explicit MX records!
Try the servers in this order:
Priority: 10 Hostname: pechora1.icann.org
» Hostname pechora1.icann.org = A 192.0.43.8

Priority: 10 Hostname: pechora2.icann.org
» Hostname pechora2.icann.org = A 192.0.43.8
...
Priority: 10 Hostname: pechora8.icann.org
» Hostname pechora8.icann.org = A 192.0.43.8
```

Τρέχοντας αυτό το script για πολλούς διαφορετικούς domains,μπορείτε να δείτε πως μεγάλοι και μικροί οργανισμοί διαχειρίζονται τα εισερχόμενα mails που είναι δρομολογημένα στην Ip διεύθυνση.

ΚΕΦΑΛΑΙΟ 7Ο

Sockets

Τα sockets χρησιμοποιούνται σχεδόν παντού, αλλά είναι μία από τα πιο σοβαρά παρεξηγημένες τεχνολογίες.

Sockets names

Έχει γίνει προφανές από τα προηγούμενα κεφάλαια,το γεγονός ότι τα sockets δεν μπορούν να ονομαστούν με ένα απλό πρωταρχικό Python τύπο σαν ένας αριθμός ή string. Αντ 'αυτού, τόσο τα πρωτόκολλα TCP και UDP χρησιμοποιούν ακέραια port numbers για να μοιραστούν μία IP διεύθυνση ενός ενιαίου μηχανήματος μεταξύ των πολλών διαφορετικών εφαρμογών που θα μπορούσαν να εκτελούνται εκεί,έτσι η διεύθυνση και το port number πρέπει να συνδυαστούν για να παράγουν ένα socket name, σαν αυτό:

('173.194.35.144',80)

Αν και μπορεί να καταλάβει κάποιος για τα sockets names διαβάζοντας τα τελευταίακεφάλαια,δηλαδή ότι το πρώτο αντικείμενο μπορεί να είναι είτε ένα hostname ή μια διεύθυνση IP με τελείες.Τα sockets names είναι σημαντικά σε διάφορα σημεία στη δημιουργία και στη χρήση των sockets.Παρακάτω υπάρχουν όλες οι σημαντικές μέθοδοι socket που απαιτούν από εσάς ένα socket name ως argument:

- mysocket.accept():Κάθε φορά που αυτό καλείται στο να ακούει ένα TCP stream socket που
  έχει εισερχόμενες συνδέσεις έτοιμο να παραδώσει στην εφαρμογή, επιστρέφει μια πλειάδα του οποίου το δεύτερο στοιχείο του οποίου είναι η απομακρυσμένη διεύθυνση που έχει συνδεθεί (το πρώτο στοιχείο στη πλειάδα είναι το net socket που συνδέεται με την απομακρυσμένη διεύθυνση).
- mysocket.bind(address): Εκχωρεί sto socket την τοπική διεύθυνση, έτσι ώστε τα εξερχόμενα πακέτα έχουν μια διεύθυνση από την οποία να προέρχονται, και έτσι ώστε κάθε εισερχόμενη συνδέση από άλλες μηχανές έχει ένα όνομα που μπορούν να χρησιμοποιήσουν για να συνδεθούν.
- mysocket.connect(address): Ορίζει ότι τα δεδομένα που αποστέλλονται μέσω αυτού του socket ότι θα κατευθύνονται προς τη συγκεκριμένη απομακρυσμένη διεύθυνση. Για π UDP socket, αυτό θέτει απλώς την προεπιλεγμένη διεύθυνση που χρησιμοποιείτε εάν ο καλών χρησιμοποιεί send () και όχι sendto (), για TCP sockets, αυτό στην πραγματικότητα

διαπραγματεύεται ένα νέο ρεύμα με ένα άλλο μηχάνημα με three-way handshake, και δημιουργεί μια εξαίρεση, εάν η διαπραγμάτευση αποτύχει.

- mysocket.getpeername():Επιστρέφει την απομακρυσμένη διεύθυνση στην οποία αυτό το socket είναι συνδεδεμένο.
- mysocket.getsockname():Επιστρέφει την διεύθυνση από αυτό το socket το τοπικό end point.
- mysocket.recvfrom(...)Για UDP sockets,αυτό επιστρέφει μία πλειάδα οπού ταιριάζει ένα string από τα επιστρεφόμενα δεδομένα με την διεύθυνση από πού στάλθηκαν.
- mysocket.sendto(data, address):Μία αποσυνδεμένη UDP port χρησιμοποιεί αυτήν την μέθοδο για να στείλει ένα data packet σε μία συγκεκριμένη απομακρυσμένη διεύθυνση.

Αυτές είναι οι σημαντικές λειτουργίες sockets που νοιάζονται για διευθύνσεις sockets.Σε γενικές γραμμές, οποιοδήποτε από τα ανωτέρω μεθόδους μπορεί να να λάβει ή να επιστρέψει οποιοδήποτε είδος διεύθυνσης που ακολουθείεννοώντας ότι αυτό θα δουλέχει άσχετα με το αν χρησιμοποιείτε IPv4, IPv6, ή ακόμα και μία από τις λιγότερο συνηθισμένες οικογένειες διευθύνσεων.

΄Ενα απλό παράδειγμα socket:

```
>>> import socket
>>> s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
>>> s.bind(('localhost', 1060))
```

Αρχικα, η address family παίρνει τη μεγαλύτερη απόφαση: ονομάζει με τι είδους δίκτυο που θέλετε να μιλήσετε, από τα πολλά είδη που ένα συγκεκριμένο μηχάνημα θα μπορούσε να υποστηρίξει.Σε αυτό την εργασία, θα χρησιμοποιηθεί η AF_INET τιμή για την address family, γιατί εξυπηρετεί καλύτερα την συντριπτική πλειοψηφία των προγραμμάτων Python, καθώς μπορεί να δουλέψει είτε με αυτούς που θα ασχοληθούν με το Linux είτε Mac OS, ή ακόμα και τα Windows.

Στη συνέχεια,μέτα την address family έρχετε ο τύπος του socket.Επιλέγει τη συγκεκριμένη τεχνική επικοινωνίας που θέλετε α χρησιμοποιήσετε στο δίκτυο που επιλέξατε. Αν και UDP και TCP είναι πολύ συγκεκριμένα στο AF_INET πρωτόκολλο της οικογένειας, οι σχεδιαστές διεπαφής sockets αποφάσισαν να δημιουργήσουν περισσότερες κοινές ονομασίες για την γενική ιδέα ενός packet-based socket, το οποίο ονομάζετε SOCK_DGRAM, και η γενική ιδέα μίας αξιόπιστης flow-controlled ροής δεδομένων, που όπως είδαμε είναι γνωστή ως SOCK_STREAM.

Το τρίτο πεδίο στην κλήση socket (), το πρωτόκολλο, χρησιμοποιείται σπάνια γιατί από τη στιγμή που έχετε ορίσει την address family και τον socket type,έχετε περιορίσει τα πιθανά πρωτόκολλα για μια σημαντική επιλογή.Για το λόγο αυτό, οι προγραμματιστές συνήθως αφήνουν αυτό απροσδιόριστο ή να παρέχουν το μηδέν για να επιλέγεται αυτόματα. Αν θέλετε μια ροή κάτω από την IP, το σύστημα ξέρει να

επιλέξει το TCP, Αν θέλετε
datagrams, τότε επιλέγει UDP. Αυτός είναι ο λόγος γιατί καμία από τα sockets() κλήσεις σε αυτό το βιβλίο δεν έχει ένα τρίτο επιχείρημα:στην πράξη είναι σχεδόν ποτέ δεν χρειάζετε.

Το τέταρτο και πέμπτο πεδίο είναι,η IP διεύθυνση και το UDP ή TCP port number. Αλλά θα πρέπει να αναφέρω ότι τα sockets names έχουν δύο components,το hostname και το port number! Αν είχατε επιλέξει AppleTalk ή ATM ή μέσω Bluetooth για την address family,τότε κάποια άλλη δομή δεδομένων θα απαιτούνταν από εσάς, αντί για μια πλειάδα με ένα string και έναν ακέραιο μέσα.

Δημιουργώντας ένα socket

Όταν κάποιος κάνει κλικ πάνω σε ένα σε ένα link για να ανοίχει μία σελίδα,ο browser κάνει τις ακόλουθες ενέργειες

```
#create an INET, STREAMing socket
s = socket.socket(
    socket.AF_INET, socket.SOCK_STREAM)
#now connect to the web server on port 80
# - the normal http port
s.connect(("www.google.com", 80))
```

Όταν το connect τελειώνει,το socket μπορεί να χρησιμοποιηθεί για να στείλει μία αίτηση για το κείμενο της σελίδας.Το ίδιο socket θα διαβάσει την απάντηση και μετά θα καταστραφεί.Client sockets χρησιμοποίουνται μόνο για μία ανταλλαγή.

Αυτό που συμβαίνει στο web server είναι λίγο πιο περίπλοκο.Πρώτα,ο web server δημιουργεί ένα server socket.

```
#create an INET, STREAMing socket
serversocket = socket.socket(
    socket.AF_INET, socket.SOCK_STREAM)
#bind the socket to a public host,
# and a well-known port
serversocket.bind((socket.gethostname(), 80))
#become a server socket
serversocket.listen(5)
```

Μερικά πράγματα που πρέπει να προσέξουμε: χρησιμοποιούμε το socket.gethostname() έτσι ώστε το socket να είναι ορατό στον έξω κόσμο.Αν χρησιμοποιούσαμε s.bind(('', 80)) ή s.bind(('localhost', 80)) ή s.bind(('127.0.0.1', 80))θα μπορούσαμε να έχουμε ένα server socket αλλά αυτό θα ήταν ορατό μόνο μέσα στην ίδια μηχανή.

Ένα δεύτερο που πρέπει να προσέξουμε είναι ότι low numbers port είναι κρατημένα για “well known” υπηρεσίες(HTTP,SNMP).

Τέλος το argument “listen” λέει στη βιβλιοθήκη socket ότι θέλουμε να εριμένουμε στην ουρά μέχρι και 5 αιτήσεις πριν αρνηθούμε εξωτερικές συνδέσεις. Εάν το υπόλοιπο του κώδικα είναι γραμμένο σωστά, αυτό θα είναι αρκετό.

```
while 1:
    #accept connections from outside
    (clientsocket, address) = serversocket.accept()
    #now do something with the clientsocket
    #in this case, we'll pretend this is a threaded server
    ct = client_thread(clientsocket)
    ct.run()
```

Αυτοί είναι γενικά 3 γενικοί τρόποι με τους οποίους αυτός ο βρόχος θα μπορούσε να δουλέψει-αποστέλλοντας ένα νήμα για να αντιμετωπίσει το clientsocket,δημιουργεί μία νέα διαδικασία για να αντιμετωπίσει το clientsocket,,ή αναδομεί αυτήν την εφαρμογή για να χρησιμοποιήσει non-blocking sockets και να πολυπλέξει ανάμεσα στο “server” socket και σε οποιοδήποτε clientsocket χρησιμοποιώντας το select.Αυτό που πρέπει να κατανοηθεί είναι ότι αυτά είναι αυτά που κάνει ένα “server” socket.Δεν στέλνει δεδομένα,δεν λαμβ’ανει δεδομένα,απλά παράγει “client” sockets.Κάθε clientsocket δημιουργήτε απαντώντας σε κάποιο άλλο “client” socket κάνοντας ένα connect() στο host και στο port που είμαστε πρόθυμοι.Μόλις δημιουργήσουμε αυτό το clientsocket επιστρέφουμε για να ακούσουμε για περισσότερες συνδέσεις.Οι δυο “clients” είναι ελεύθεροι να μιλήσουν,αυτοί χρησιμοποιούν μερικά δυναμικά allocated port τα οποία θα ανακυκλωθούν όταν η συζήτηση τελειώσει.

Χρησιμοποιώντας ένα Socket

Το πρώτο πράγμα που πρέπει να σημειωθεί, είναι ότι του web browser το “client” socket και ο Web server “client” είναι πανομοιότυποι. Δηλαδή, αυτό είναι μία "peer to peer" συνομιλία. Ή να το θέσω αλλιώς, ως σχεδιαστής, θα πρέπει να αποφασίσει ποιοι είναι οι κανόνες της εθιμοτυπίας είναι για μια συνομιλία. Κανονικά,το connect socket ξεκινά τη συζήτηση, στέλνοντας ένα αίτημα. Αλλά αυτή είναι μια σχεδιαστική απόφαση - δεν είναι ένας κανόνας των socket.

Τώρα υπάρχουν δύο set ρημάτων που θα χρησιμοποιηθούν για την επικοινωνία. Μπορείτε να χρησιμοποιήσετε το send και recv, ή μπορείτε να μεταμορφώσετε το client socket σας σε ένα file-like beast και να χρησιμοποιήσετε το read και write. Ο τελευταίος είναι ο τρόπος που παρουσιάζει η Java στα δικά της sockets. Θα πρέπει να χρησιμοποιήσετε flush on sockets.Αυτά είναι αρχεία buffer , και ένα κοινό λάθος είναι για να κάνεις write κάτι, μετά κάνεις read για μία απάντηση. Χωρίς το flush εκεί, μπορείτε να περιμένετε για πάντα για την απάντηση, διότι η αίτηση μπορεί ακόμα να είναι στο output buffer σας.

Όσον αφορα τα block sockets -send και recv λειτουργούν στους network buffers. Δεν χειρίζονται απαραίτητα όλα τα bytes που παραδίδουν (ή περιμένουμε από

αυτούς), διότι κύριος στόχος τους είναι η διαχείριση των network buffer . Σε γενικές γραμμές,αυτοί επιστρέφουν όταν οι σχετικοί network buffers έχουν γεμίσει (send) ή αδειάζει (recv). Θα σας πει πόσα bytes χειρίστηκαν. Είναι δική σας ευθύνη να τους καλέσετε και πάλι μέχρι το μήνυμά σας να έχει αντιμετωπιστεί πλήρως.

Όταν ένα recv επιστρέφει 0 byte, αυτό σημαίνει ότι η άλλη πλευρά έχει κλείσει (ή βρίσκεται στη διαδικασία του κλεισίματος) τη σύνδεση. Δεν θα δοθούν περισσότερα στοιχεία σχετικά με αυτό το θέμα.

Ένα πρωτόκολλο HTTP χρησιμοποιεί ένα socket για μία μόνο μεταφορά. Ο client στέλνει μια αίτηση, διαβάζει μια απάντηση. Αυτό είναι το. Το socket απορρίπτεται. Αυτό σημαίνει ότι ένας πελάτης μπορεί να εντοπίσει το τέλος της απάντησης από την παραλαβή 0 byte.

Αλλά αν σκοπεύετε να χρησιμοποιήσετε ξανά το socket σας για περαιτέρω μεταφορές, θα πρέπει να συνειδητοποιήσουμε ότι δεν υπάρχει "EOT" (Τέλος Μεταφοράς) σε ένα socket.Επαναλαμβάνω: αν ένα socket send or recv επιστρέφει μετά το χειρισμό 0 byte, η σύνδεση έχει διακοπεί. Εάν η σύνδεση δεν έχει σπάσει, μπορείτε να περιμένετε σε ένα recv για πάντα, επειδή το socket δεν θα σας πει ότι δεν υπάρχει τίποτα περισσότερο να διαβάστει (για τώρα). Τώρα, αν σκεφτεί κανείς ότι για λίγο, θα συνειδητοποιήσετε μια θεμελιώδη αλήθεια για τα sockets: Τα μηνύματα πρέπει να είναι είτε σταθερό μήκος (yuck), ή να οριοθετηθεί (shrug), ή δείχνουν το πόσο καιρό είναι (πολύ καλύτερα), ή τελειώνουν με το κλείσιμο της σύνδεσης. Η επιλογή είναι αποκλειστικά δική σας.

Υποθέτοντας ότι δεν θέλετε να τερματίσετε τη σύνδεση, η απλούστερη λύση είναι ένα σταθερό μήνυμα μήκος:

```
class mysocket:
    '''demonstration class only
      - coded for clarity, not efficiency
    '''

    def __init__(self, sock=None):
        if sock is None:
            self.sock = socket.socket(
                socket.AF_INET, socket.SOCK_STREAM)
        else:
            self.sock = sock

    def connect(self, host, port):
        self.sock.connect((host, port))

    def mysend(self, msg):
        totalsent = 0
        while totalsent < MSGLEN:
            sent = self.sock.send(msg[totalsent:])
            if sent == 0:
                raise RuntimeError("socket connection broken")
            totalsent = totalsent + sent
```

```
def myreceive(self):
    msg = ''
    while len(msg) < MSGLEN:
        chunk = self.sock.recv(MSGLEN-len(msg))
        if chunk == '':
            raise RuntimeError("socket connection broken")
        msg = msg + chunk
    return msg
```

Ο κώδικας που αποστέλλετε εδώ μπορεί να χρησιμοποιηθεί για σχεδόν οποιοδήποτε σύστημα ανταλλαγής μηνυμάτων - στην Python στέλνετε strings, και μπορείτε να χρησιμοποιήσετε την len () για να καθορίσει το μήκος της (έστω και αν έχει ενσωματωμένο \ 0 χαρακτήρες). Είναι κυρίως η λήψη κώδικα που γίνεται πιο περίπλοκη.

Μία ενίσχυση είναι να κάνεις το πρώτο χαρακτήρα του μηνύματος ένδειξη του τύπου μηνύματος, και ο τύπος καθορίζει το μήκος. Τώρα έχετε δύο recvs - το πρώτο για να πάρει (τουλάχιστον) τον πρώτο χαρακτήρα ώστε να μπορείτε να αναζητήσετε το μήκος, και το δεύτερο σε ένα βρόχο για να πάρει το υπόλοιπο. Αν αποφασίσετε να πάτε στην οριοθετημένη διαδρομή, θα λαμβάνετε σε κάποιο αυθαίρετο μέγεθος κομμάτι, (4096 ή 8192 είναι συχνά ένα καλό ταίριασμα για το μέγεθος των network buffer), σαρώνοντας ι αυτό που έχετε λάβει για οριοθέτη.

Ένα στοιχείο που πρέπει κάποιος να γνωρίζει: αν ένα conversational πρωτόκολλο σας επιτρέπει πολλαπλά μηνύματα που θα σταλούν χωρίς κάποιου είδους απάντηση, και περνάτε recv ένα αυθαίρετο κομμάτι μεγέθους, μπορείτε να καταλήξετε διαβάζοντας την έναρξη μιας ακόλουθο μήνυμα. Θα πρέπει να το κρατήσετε, μέχρι να χρειαστεί.

Disconnecting

Ένας τρόπος για να χρησιμοποιήστε την εντολή shutdown αποτελεσματικά είναι σε μία HTTP ανταλλαγή. Ο client στέλνει μια αίτηση και στη συνέχεια να κάνει ένα shutdown(1). Αυτό λέει ο διακομιστής "Ο client κάνει αποστολή, αλλά μπορεί να συνεχίσετε να λαμβάνει." Ο διακομιστής μπορεί να ανιχνεύσει "EOF" από την λήψη 0 byte. Μπορεί να υποθέσει ότι έχει την πλήρη αίτηση. Ο διακομιστής στέλνει μια απάντηση. Εάν η send ολοκληρωθεί με επιτυχία τότε, πράγματι, ο πελάτης λαμβάνει ακόμα.

Η Python παίρνει το αυτόματο shutdown ένα βήμα παραπέρα και λέει ότι όταν ένα socket garbage collected, θα το κάνει αυτόματα ένα close, αν είναι απαραίτητο. Αν το socket σας εξαφανίζεται χωρίς να κάνει close, το socket στο άλλο άκρο μπορεί να περιμένει επ 'αόριστον, σκεπτόμενος ότι είστε αργοί. Κλείστε τα socket σας όταν τελειώσετε.

Non-blocking Sockets

Αν έχετε καταλάβει τα τελευταία, ξέρετε ήδη τα περισσότερα από όσα χρειάζεται να γνωρίζετε σχετικά με τους μηχανισμούς των sockets. Θα εξακολουθούν να χρησιμοποιούν τις ίδιες κλήσεις, με τον ίδιο περίπου τρόπο. Είναι ακριβώς ότι, αν το κάνεις σωστά, η εφαρμογή σας θα είναι σχεδόν μέσα-έξω.

Στην Python, μπορείτε να χρησιμοποιήσετε socket.setblocking(0) ώστε να είναι non-blocking. Η κύρια διαφορά στο μηχανισμό είναι ότι οι send, recv, conntect και accept μπορούν να επιστρέψουν χωρίς να κάνουν τίποτα. Έχετε (φυσικά) μια σειρά από επιλογές. Μπορείτε να ελέγξετε τον κωδικό επιστροφής και οι κωδικοί σφαλμάτων.

Χρησιμοποιήστε το select.

Στην C ο κώδικας select είναι λίγο περίπλοκος.Στην Python,είναι αρκετά κοντά στην C έκδοση και αν κάποιος καταλάβει την select στην Python,τότε θα έχει λίγα προβλήμματα με την C.

```
ready_to_read, ready_to_write, in_error = \
            select.select(
              potential_readers,
              potential_writers,
              potential_errs,
              timeout)
```

Αυτό το κομμάτι κώδικα περνάει στην select τρεις λίστες:η πρώτη περιέχει όλα τα sockets που θα θέλατε να διαβάσετε,η δεύτερη όλα τα sockets που θέλατε να γράψετε κ η τελευταία εκείνη που θέλατε να ελέγξετε για λάθη.Θα πρέπει να σημειώθει ότι ένα socket μπορεί να πάρει παραπάνω από μία λίστα.Η κλήση select είναι blocking,αλλά μπορείτε να την δώσετε ένα timeout.

Στην επιστροφή,θα πάρετε τρεις λίστες.Αυτές έχουν τα sockets,τα οποία είναι readable,writable kai in error.Κάθε λίστα είναι ένα υποσύνολο από την αντίστοιχη λίστα που πέρασε.Αν βάλετε ένα socket σε περισσότερες από μία input λίστα,αυτό θα είναι σε μία output λίστα.

Αν έχετε ένα "server" socket,βάλτε το στη λίστα potential_readers. Αν βγαίνει στη readable λίστα , το accept θα δουλεύει. Αν έχετε δημιουργήσει ένα καινούργιο socket για connect με κάποιον άλλο,βάλτε το στη λίστα potential_writers. Αν εμφανίζεται στη writing λίστα , έχετε συνδεθεί!

Ένα πρόβλημα με το select: εάν κάπου σε αυτές τις input λίστες των sockets είναι ένα socket που δεν υπάρχη πια,το select θα αποτύχει.Τότε θα πρέπει να εκτελέσετε ένα βρόχο για κάθε socket σε όλες τις λίστες και να κάνετε ένα select([sock],[],[],0)μέχρι να βρείτε το “κακό” .Αυτό το χρονικό όριο του 0 σημαίνει ότι δεν θα πάρει πολύ, αλλά είναι άσχημο.

Στην πραγματικότητα,το select μπορεί να είναι βολικό, ακόμη και με blocking sockets. Είναι ένας τρόπος για να διαπιστωθεί αν θα μπλοκάρει - socket επιστρέφει ως readable όταν υπάρχει κάτι στα buffers. Ωστόσο, αυτό εξακολουθεί να μην βοηθάει με το πρόβλημα του καθορισμού αν η άλλη πλευρά έγινε, ή απλά απασχολημένος με κάτι άλλο.

Portability alert:Στο Unix,select δουλεύει και με sockets κ φακέλους.Μην το δοκιμάσετε στα Windows,η select δουλεύει μόνο με sockets

Stream/datagram Sockets

Ένα Stream socket είναι σαν ένα τηλεφώνημα - μία πλευρά κάνει την κλήση,η άλλη απαντάει, λέτε γεια ο ένας τον άλλο (SYN / ACK στο TCP), και στη συνέχεια να ανταλλάσσουν πληροφορίες. Μόλις τελειώσετε, μπορείτε να πω αντίο (FIN / ACK σε TCP). Εάν μία πλευρά δεν ακούει το αντίο, θα καλέσουν συνήθως τον άλλο ξανά από αυτό είναι ένα απροσδόκητο συμβάν. Συνήθως ο client θα επανασυνδεθεί στο server. Υπάρχει μια εγγύηση ότι τα δεδομένα δεν θα φτάσουν με διαφορετική σειρά από αυτή που το έστειλε, και υπάρχει εύλογη εξασφάλιση ότι τα δεδομένα δεν θα καταστραφούν.

Ένα datagram socket είναι όπως το να περνάσ μια σημείωση στην τάξη. Εξετάστε την περίπτωση που δεν είναι ακριβώς δίπλα το πρόσωπο που θέλετε να περάσετε τη σημείωση,το σημείωμα θα ταξιδέψει από άτομο σε άτομο. Μπορεί να μην φτάσει στον προορισμό του, και μπορεί να τροποποιηθεί. Εάν στέλνατε δύο σημειώματα στο ίδιο πρόσωπο, αυτά θα έφταναν σε μία σειρά που δεν θα θέλατε, δεδομένου ότι η διαδρομή που ακολουθούν οι σημειώσεις μέσα στην τάξη μπορεί να μην είναι η ίδια, ένα άτομο δεν μπορεί να περάσει μια σημείωση τόσο γρήγορα όσο ένα άλλο, κλπ. .

Ετσι, χρησιμοποιείτε ένα stream socket όταν οι πληροφορίες πρέπει να είναι ανέπαφες και είναι σημαντικές. Πρωτόκολλα μεταφοράς αρχείων είναι ένα καλό παράδειγμα εδώ. Δεν θέλετε να κατεβάσετε κάποιο αρχείο με τα περιεχόμενά του τοποθετημένα τυχαία και κατεστραμένα!

Θα μπορούσες να χρησιμοποιήσεις ένα datagram sockets όταν η σειρά είναι λιγότερο σημαντική από την έγκαιρη παράδοση (VoIP ή game protocol), όταν δεν θέλετε το υψηλό λειτουργικό κόστος ενός stream(αυτός είναι ο λόγος γιατι το UDP είναι ένα datagram protocol, έτσι ώστε να μπορεί οι servers να ανταποκριθούν σε πολλές αιτήσεις ταυτόχρονα πολύ γρήγορα), ή όταν δεν με ενδιαφέρει πάρα πολύ, αν τα δεδομένα φτάνει ποτέ στον προορισμό του.

Unix Domain Socket

Ένα Unix domain socket ή IPC socket(inter-process επικοινωνίας socket) είναι ένα καταληκτικό σημείο επικοινωνίας για την ανταλλαγή δεδομένων μεταξύ των διαδικασιών που εκτελούνται στο ίδιο host λειτουργικό σύστημα. Ενώ παρόμοια λειτουργικότητα με το named pipes, Unix domain sockets μπορεί να δημιουργηθεί ως byte streams ή ως datagram ακολουθίες, ενώ τα pipes είναι bytes stream μόνο. Διεργασίες χρησιμοποιώντας Unix domain sockets δεν χρειάζεται να μοιράζονται μια κοινή καταγωγή. Το API για Unix domain sockets είναι παρόμοια με αυτά ενός Internet socket, αλλά δεν χρησιμοποιεί ένα υποκείμενο πρωτοκόλλο δικτύου για την επικοινωνία.Η εγκατάσταση ενός Unix domain socket είναι ένα πρότυπο συστατικό του POSIX λειτουργικού συστήματος.

Τα Unix domain sockets χρησιμοποιούν το σύστημα αρχείων, όπως τις διευθύνσεις name space. Θα αναφέρονται από διαδικασίες όπως inodes στο σύστημα αρχείων. Αυτό επιτρέπει σε δύο διαδικασίες να ανοίξει το ίδιο socket, ώστε να επικοινωνούν μεταξύ τους. Ωστόσο, η επικοινωνία γίνεται εξ ολοκλήρου εντός του πυρήνα του λειτουργικού συστήματος.

Εκτός από την αποστολή των δεδομένων, οι διαδικασίες μπορούν να στείλουν file descriptors σε μια σύνδεση UNIX Domain Socket στο χρησιμοποιώντας tis sendmsg () και recvmsg () κλήσεις του συστήματος.

Network Exceptions

Τi λάθη μπορεί να προκύψουν δουλεύοντας με sockets; Αν και ο αριθμός των λαθών που μπορεί να λάβουν χώρα χρησιμοποιώντας μια σύνδεση δικτύου είναι αρκετά μεγάλος - για παράδειγμα, ο αριθμός των εξαιρέσεων με τις οποίες οι socket λειτουργίες μπορεί να χτυπήσουν τα προγράμματά σας είναι ευτυχώς πολύ λίγες . Οι εξαιρέσεις που αφορούν ειδικά socket πράξεις είναι :

- socket.gaierror: Η εξαίρεση αυτή δημιοιυργήτε όταν η getaddrinfo () δεν μπορούν να βρουν ένα όνομα ή υπηρεσία που να ρωτήσω σχετικά-Μπορεί να δημιουργήσει όχι μόνο όταν κάνετε μια κλήση για getaddrinfo (), αλλά αν σας προμηθεύσει
  ένα hostname αντί μιας διεύθυνσης IP σε μια κλήση, όπως bind () ή connetct() και η
  hostname lookup αποτυγχάνει:

```
>>> import socket
>>> s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
>>> s.connect(('nonexistent.hostname.foo.bar', 80))
```

```
Traceback (most recent call last):
  ...
gaierror: [Errno -5] No address associated with hostname
```

- socket.error: Αυτή είναι η κινητήριος δύναμη της socket module, και θα δημιουργηθεί για σχεδόν κάθε βλάβη που μπορεί να συμβεί σε οποιοδήποτε στάδιο σε ένα δίκτυο μεταφοράς.Ξεκινώντας με την Python 2.6, η εξαίρεση αυτή έγινε, αρκετά κατάλληλα, μια υποκλάση της γενικότερης IOError.
- socket.timeout: Η εξαίρεση αυτή θα δημιουργηθεί μόνο εάν εσείς, ή μια βιβλιοθήκη που χρησιμοποιείτε,αποφασίζει να θέσει ένα χρονικό όριο σε ένα socket και όχι να περιμένει πάντα για την send () ή recv () να ολοκληρωθεί. Αυτό δείχνει ότι το χρονικό όριο επιτεύχθηκε πριν η διαδικασία ολοκληρωθεί κανονικά.

Handling Exceptions

Υπάρχουν τέσσερις διαφορετικές προσεγγίσεις στο πως μπορούμε να χειριστούμε τις εξαιρέσεις.

Η πρώτη προσέγγιση είναι να μην χείριστεις καθόλου τις εξαιρέσεις που θα δημιουργηθούνε.Αυτο είναι εφικτό αν μόνο ένα άτομο χρησιμοποιεί το script ή μία ομάδα προγραμματιστών Python,οι οποίοι δεν θα μείνουν έκπληκτοι με το να δουν μία εξαίρεση.

Στην συνέχεια,αν κάποιος γράφει μία βιβλιοθήκη είναι γενικά περίπλοκο γιατι η βιβλιοθηκή έχει συνδέσεις με αρκετές άλλες υπηρεσίες και θα είναι δύσκολο για ένα προγραμματιστη να μαντέψει ποια τις διαδικασίες δικτύου που εκτέλειται κατέληξε σε ένα socket.error.

Αν κάποιος προσφέρει μία netcopy() μέθοδο,η οποία αντιγράφει ένα φάκελο από ένα απομακρυσμένο σε ένα άλλο,για παράδειγμα, ένα socket.error δεν βοηθάει τον καλών να μάθει εάν το error είναι στο μηχάνημα πηγής ή στο μηχάνημα προορισμού ή είναι ένα πρόβλημα με όλα μαζί!Σε αυτην την περίπτωση θα ήταν καλύτερο να ορίσεις τις δικές σου εξαιρέσεις,όπως SourceHostError kai DestHostError.Το αρχικό socket error,που μπορεί κάποιος να χρησιμοποιήσει κ αν θέλει να το επεκτείνει παραπάνω είναι:

```
try:
» host = sock.bind(address)
except socket.error as e:
» raise URLError(e)
```

Η Τρίτη προσέγγιση για τις εξαιρέσεις είναι να αντικαταστήσεις μία try…except πρόταση σε κάθε κλήση δικτύου,και να τυπώνεις μηνύματα λάθους στη θέση του.Αυτό είναι βολικό για μικρά προγραμμάτα,αλλα μπόρει να γίνει πολύ επαναλαμβανόμενο σε μεγάλα προγράμματα χώρις απαραίτητα να περιλαμβάνουν τις απαραίτητες πληροφορίες για οτν χρήστη.

Η τελευταία προσέγγιση έχει την ιδέα του να έχει χειριστές σφαλμάτων που έχουν πολλές γραμμές κώδικα,δηλαδή, μπορούν να αντικαταστήσουν διαδικασίες με χειριστές σφαλμάτων για παράδειγμα :

```
try:
» deliver_updated_keyfiles(…)
except (socket.error, socket.gaierror) as e:
» print >>sys.stderr, 'cannot deliver remote keyfiles: %s' % (e)
» exit(1)
```

Or, better yet, have pieces of code like this raise an error of your own devising:

```
except:
» FatalError('cannot send replies: %s' % (e))
```

ΚΕΦΑΛΑΙΟ 8^{Ο}

Email

Θα αναφέρω λίγα παραδείγματα το πώς χρησιμοποιούμε το email πακέτο για να διαβάσουμε,γράψουμε και να στείλουμε απλά emails.

```
# Import smtplib for the actual sending function
import smtplib

# Import the email modules we'll need
from email.mime.text import MIMEText

# Open a plain text file for reading.  For this example, assume that
# the text file contains only ASCII characters.
fp = open(textfile, 'rb')
# Create a text/plain message
msg = MIMEText(fp.read())
fp.close()

# me == the sender's email address
# you == the recipient's email address
msg['Subject'] = 'The contents of %s' % textfile
msg['From'] = me
msg['To'] = you

# Send the message via our own SMTP server, but don't include the
# envelope header.
s = smtplib.SMTP('localhost')
s.sendmail(me, [you], msg.as_string())
s.quit()
```

Και αναλύοντας τις RFC822 επικεφαλίδες μπορεί να γίνει εύκολα με τις parse(filename) or parsestr(message_as_string) μεθόδους της Parser() class:

```
# Import the email modules we'll need
from email.parser import Parser

#  If the e-mail headers are in a file, uncomment this line:
#headers = Parser().parse(open(messagefile, 'r'))
```

```
#  Or for parsing headers in a string, use:
headers = Parser().parsestr('From: <user@example.com>\n'
        'To: <someone_else@example.com>\n'
        'Subject: Test message\n'
        '\n'
        'Body would go here\n')

#  Now the header items can be accessed as a dictionary:
print 'To: %s' % headers['to']
print 'From: %s' % headers['from']
print 'Subject: %s' % headers['subject']
```

ΚΕΦΑΛΑΙΟ 9^{Ο}

Network programming with java

Θα ήταν πολύ ενδιαφέρον να παρουσιάσω το πώς η java εμπλέκετε με τα διαφόρα πρωτόκολλα,με τα sockets.Αρχικά,παρακάτω βρίσκετε κώδικας σε java για TCP Server/Client

TCP

TCPServer.java

```
import java.io.*;
import java.net.*;

class TCPServer
{
   public static void main(String argv[]) throws Exception
      {
         String clientSentence;
         String capitalizedSentence;
         ServerSocket welcomeSocket = new ServerSocket(6789);

         while(true)
         {
            Socket connectionSocket = welcomeSocket.accept();
            BufferedReader inFromClient =
               new BufferedReader(new
InputStreamReader(connectionSocket.getInputStream()));
            DataOutputStream outToClient = new
DataOutputStream(connectionSocket.getOutputStream());
            clientSentence = inFromClient.readLine();
            System.out.println("Received: " + clientSentence);
            capitalizedSentence = clientSentence.toUpperCase() + '\n';
            outToClient.writeBytes(capitalizedSentence);
         }
      }
}
```

και ο client:

TCPClient.java

```
import java.io.*;
import java.net.*;

class TCPClient
{
 public static void main(String argv[]) throws Exception
 {
   String sentence;
   String modifiedSentence;
   BufferedReader inFromUser = new BufferedReader( new
InputStreamReader(System.in));
   Socket clientSocket = new Socket("localhost", 6789);
   DataOutputStream outToServer = new
DataOutputStream(clientSocket.getOutputStream());
    BufferedReader inFromServer = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));
    sentence = inFromUser.readLine();
    outToServer.writeBytes(sentence + '\n');
    modifiedSentence = inFromServer.readLine();
    System.out.println("FROM SERVER: " + modifiedSentence);
    clientSocket.close();
 }
}
```

UDP

Όσον αφορα το UDP τώρα ακολουθούν δύο προγράμματα ένα για το server και ένα για το client.

UDPServer.java:

```
import java.io.*;
import java.net.*;

class UDPServer
```

```
{
  public static void main(String args[]) throws Exception
    {
      DatagramSocket serverSocket = new DatagramSocket(9876);
        byte[] receiveData = new byte[1024];
        byte[] sendData = new byte[1024];
        while(true)
          {
            DatagramPacket receivePacket = new DatagramPacket(receiveData,
receiveData.length);
            serverSocket.receive(receivePacket);
            String sentence = new String( receivePacket.getData());
            System.out.println("RECEIVED: " + sentence);
            InetAddress IPAddress = receivePacket.getAddress();
            int port = receivePacket.getPort();
            String capitalizedSentence = sentence.toUpperCase();
            sendData = capitalizedSentence.getBytes();
            DatagramPacket sendPacket =
            new DatagramPacket(sendData, sendData.length, IPAddress, port);
            serverSocket.send(sendPacket);
          }
    }
}
```

UDPClient.java:

```
import java.io.*;
import java.net.*;

class UDPClient
{
  public static void main(String args[]) throws Exception
  {
    BufferedReader inFromUser =
      new BufferedReader(new InputStreamReader(System.in));
    DatagramSocket clientSocket = new DatagramSocket();
    InetAddress IPAddress = InetAddress.getByName("localhost");
    byte[] sendData = new byte[1024];
    byte[] receiveData = new byte[1024];
    String sentence = inFromUser.readLine();
    sendData = sentence.getBytes();
    DatagramPacket sendPacket = new DatagramPacket(sendData,
sendData.length, IPAddress, 9876);
    clientSocket.send(sendPacket);
    DatagramPacket receivePacket = new DatagramPacket(receiveData,
receiveData.length);
    clientSocket.receive(receivePacket);
    String modifiedSentence = new String(receivePacket.getData());
    System.out.println("FROM SERVER:" + modifiedSentence);
    clientSocket.close();
```

```
  }
}
```

Sockets

Το παρακάτω παράδειγμα δείχνει πως ένα πρόγραμμα μπορεί να κάνει μία σύνδεση σε ένα server πρόγραμμα χρησιμοποιώντας την Socket class,και έπειτα πως ο client στέλνει και λαμβάνει δεδομένα από το server μέσω ενός socket.

```
import java.io.*;
import java.net.*;

public class EchoClient {
    public static void main(String[] args) throws IOException {

        Socket echoSocket = null;
        PrintWriter out = null;
        BufferedReader in = null;

        try {
            echoSocket = new Socket("taranis", 7);
            out = new PrintWriter(echoSocket.getOutputStream(), true);
            in = new BufferedReader(new InputStreamReader(
                                    echoSocket.getInputStream()));
        } catch (UnknownHostException e) {
            System.err.println("Don't know about host: taranis.");
            System.exit(1);
        } catch (IOException e) {
            System.err.println("Couldn't get I/O for "
                               + "the connection to: taranis.");
            System.exit(1);
        }

            BufferedReader stdIn = new BufferedReader(
                               new InputStreamReader(System.in));
            String userInput;

            while ((userInput = stdIn.readLine()) != null) {
                out.println(userInput);
                System.out.println("echo: " + in.readLine());
            }

            out.close();
            in.close();
            stdIn.close();
            echoSocket.close();
    }
}
```

ΒΙΒΛΙΟΓΡΑΦΙΑ

- Foundations of Python Network Programming 2nd Edition
- Computer Networking A Top-Down Approach 5th edition
- Python.org
- Πηγές από το google.com