
# ns-3 Tutorial

***Release ns-3.22***

**ns-3 project**

May 18, 2016

ΠΕΡΙΕΧΟΜΕΝΑ

Αυτός είναι ο οδηγός του ns-3. Η βασική τεκμηρίωση για το project του ns-3 είναι διαθέσιμη σε πέντε μορφές:

- Στο Doxygen του ns-3: Τεκμηρίωση των δημόσιων APIs του προσομοιωτή
- Στον Οδηγό *(το παρόν έγγραφο)*, στο Εγχειρίδιο, και στη Βιβλιοθήκη Μοντέλων για την τελευταία έκδοση και το δέντρο ανάπτυξης του ns-3
- Στο wiki του ns-3

Το έγγραφο αυτό είναι γραμμένο σε reStructuredText για το Sphinx και διατηρείται στον κατάλογο `doc/tutorial` του πηγαίου κώδικα του ns-3.

ΚΕΦΑΛΑΙΟ

ΕΝΑ

# ΕΙΣΑΓΩΓΗ

Ο *ns-3* προσομοιωτής είναι ένας προσομοιωτής δικτύου διακριτών-γεγονότων με στόχο την έρευνα και την εκπαιδευτική χρήση. Το πρόγραμμα |ns-3|, ξεκίνησε το 2006, είναι ένα ανοιχτού κώδικα πρόγραμμα ανάπτυξης *ns-3*.

Ο σκοπός αυτού του οδηγού είναι να εισαγάγει νέους χρήστες *ns-3* στο σύστημα με ένα δομημένο τρόπο. Μερικές φορές είναι δύσκολο για τους νέους χρήστες να μαζέψουν τις απαραίτητες πληροφορίες από λεπτομερή εγχειρίδια και να τις μετατρέψουν στην εργασία προσομοίωσης. Σε αυτό το εγχειρίδιο, θα χτίσουμε αρκετά παραδείγματα προσομοιώσεων, εισαγωγής και εξήγησης των βασικών εννοιών και χαρακτηριστικών.

Καθώς το εγχειρίδιο συνεχίζει, θα εισάγουμε την πλήρη έκδοση του *ns-3* και παρέχονται υποδείξεις για τον πηγαίο κώδικα για όσους ενδιαφέρονται να ψάξουν βαθύτερα μέσα στη λειτουργία του συστήματος.

Μερικά βασικά σημεία αξίζουν να σημειωθούν κατά την έναρξη:

- Ο *ns-3* είναι ανοιχτός-κώδικας, και το πρόγραμμα προσπαθεί να διατηρήσει ένα ανοιχτό περιβάλλον για τους ερευνητές ώστε να συμβάλλουν και να μοιράζονται το λογισμικό τους.
- Ο *ns-3* δεν είναι επέκταση του ns-2; Ο *ns-3* είναι ένας νέος προσομοιωτής. Οι δύο εξομοιωτές είναι γραμμένοι σε C++ αλλά ο *ns-3* είναι ένας νέος προσομοιωτής που δεν υποστηρίζει τις APIs του *ns-2*. Μερικά μοντέλα έχουν ήδη μεταφερθεί από τον *ns-2* στον *ns-3*. Το πρόγραμμα θα συνεχίσει να διατηρεί τον *ns-2* καθώς ο *ns-3* θα χτίζεται, και θα μελετήσει μηχανισμούς μετάβασης και ολοκλήρωσης.

## 1.1 Σχετικά με τον *ns-3*

Ο *ns-3* έχει αναπτυχθεί για να παρέχει μια ανοιχτή, επεκτάσιμη πλατφόρμα προσομοίωσης δικτύου, για την δικτύωση της έρευνας και της εκπαίδευσης. Συνοπτικά, ο *ns-3* παρέχει μοντέλα για το πώς τα πακέτα δεδομένων των δικτύων δουλεύουν και εκτελούνται, και παρέχει μια μηχανή προσομοίωσης για τους χρήστες να διεξάγουν πειράματα προσομοίωσης. Μερικοί από τους λόγους για να χρησιμοποιήσετε τον *ns-3* περιλαμβάνουν την πραγματοποίηση μελετών που είναι πιο δύσκολο ή αδύνατο να διενεργηθεί με πραγματικά συστήματα, για να μελετήσουμε τη συμπεριφορά του συστήματος σε ένα ιδιαίτερα ελεγχόμενο, αναπαραγόμενο περιβάλλον, και να μάθουν για το πώς τα δίκτυα δουλεύουν. Οι χρήστες θα παρατηρήσουν ότι το διαθέσιμο πρότυπο που παρατίθεται στο *ns-3* εστιάζει στην μοντελοποίηση πώς τα πρωτόκολλα του Διαδικτύου και των δικτύων δουλεύουν, αλλά ο *ns-3* δεν περιορίζεται σε συστήματα Διαδικτύου - Οι διάφοροι χρήστες που χρησιμοποιούν *ns-3* για τη μοντελοποίηση των συστημάτων που δεν βασίζονται στο Διαδίκτυο.

Υπάρχουν πολλά εργαλεία προσομοίωσης για μελέτες προσομοίωσης του δικτύου. Παρακάτω είναι μερικά χαρακτηριστικά γνωρίσματα του *ns-3* σε αντίθεση με άλλα εργαλεία.

- Ο *ns-3* έχει σχεδιαστεί ως ένα σύνολο βιβλιοθηκών που μπορούν να συνδυαστούν μεταξύ τους και επίσης με άλλες εξωτερικές βιβλιοθήκες λογισμικού. Ενώ ορισμένες πλατφόρμες προσομοίωσης παρέχουν στους χρήστες με ένα ενιαίο, ολοκληρωμένο γραφικό περιβάλλον χρήστη στις οποίες είναι όλες οι εργασίες που πραγματοποιούνται, ο *ns-3* είναι περισσότερο σπονδυλωτός στο θέμα αυτό. Αρκετά εξωτερικά animators και ανάλυση δεδομένων και τα εργαλεία οπτικοποίησης μπορούν να χρησιμοποιηθούν με τον *ns-3*. Ωστόσο,

οι χρήστες θα πρέπει να περιμένουμε για να εργαστούν στη γραμμή εντολών και με C++ και/ή Python εργαλεία ανάπτυξης λογισμικού.

- Ο *ns-3* χρησιμοποιείται κυρίως σε συστήματα Linux, αν και υπάρχει υποστήριξη για το FreeBSD, Cygwin (για Windows), και η υποστήριξη του Windows Visual Studio είναι στη διαδικασία της προετοιμασίας.
- Ο | ns-3 | δεν είναι επίσημα προϊόν λογισμικού κάποιας εταιρείας. Υποστήριξη για τον | ns-3 | γίνεται με βάση λίστα με την καλύτερη δυνατή προσπάθεια για τους *ns-3* χρήστες.

## 1.2 Για τους *ns-2* χρήστες

Για όσους είναι εξοικειωμένοι με *ns-2* (ένα δημοφιλές εργαλείο που προηγήθηκε του *ns-3*), η πιο ορατή αλλαγή κατά τη μετακίνηση προς *ns-3* είναι η επιλογή γλώσσας του scripting. Προγράμματα σε *ns-2* είναι γραμμένα σε OTcl και τα αποτελέσματα των προσομοιώσεων μπορούν να απεικονιστούν χρησιμοποιώντας το Network Animator nam. Δεν είναι δυνατόν να εκτελέσετε μια προσομοίωση σε *ns-2* μόνο από την C++ (δηλαδή, ως ένα πρόγραμμα main () χωρίς OTcl). Επιπλέον, ορισμένα συστατικά του *ns-2* είναι γραμμένα σε C++ και τα άλλα στην OTcl. Στην *ns-3*, ο προσομοιωτής είναι γραμμένος εξολοκλήρου σε C++, με επιλογή σε Python bindings. Σενάρια προσομοίωσης μπορούν να γραφούν σε C++ ή Python. Νέα animators και visualizers είναι διαθέσιμα και σε εξέλιξη. Από την στιγμή που ο *ns-3* παράγει pcap packet trace files, άλλα εργαλεία μπορούν επίσης να χρησιμοποιηθούν για να αναλύσουν τα ίχνη. Σε αυτό τον οδηγό, αρχικά θα επικεντρωθούμε στην σε scripting απευθείας σε C++ και την ερμηνεία των αποτελεσμάτων μέσω αρχείων παρακολούθησης.

Από την άλλη έχουν και ομοιότητες καθώς (και τα δύο, για παράδειγμα βασίζονται σε C++, και ορισμένος κώδικας από τον *ns-2* έχει ήδη μεταφερθεί στον *ns-3*). Θα προσπαθήσουμε να τονίσουμε τις διαφορές μεταξύ του *ns-2* και του *ns-3*, καθώς προχωράμε αυτό τον οδηγό.

Μία ερώτηση που συχνά ακούμε είναι «Πρέπει ακόμα να χρησιμοποιώ τον *ns-2* ή να μετακινηθώ στον *ns-3*;» Κατά την γνώμη του συγγραφέα, αν ο χρήστης κατά κάποιο τρόπο δεν ανήκει στον *ns-2* (είτε με βάση την υπάρχουσα προσωπική άνεση και γνώση του *ns-2*, είτε βασίζεται σε ένα συγκεκριμένο μοντέλο προσομοίωσης που είναι διαθέσιμο μόνο στον *ns-2*), ένας χρήστης θα είναι πιο παραγωγικός στον *ns-3* για τους ακόλουθους λόγους:

- Ο *ns-3* διατηρείται ενεργός με μία ενεργό, ενημερωτική λίστα χρηστών, ενώ ο *ns-2* διατηρείται λιγότερο καθώς δεν έχει δεί σημαντική εξέλιξη στον κεντρικό κώδικα του για πάνω από μια δεκαετία.
- Ο *ns-3* παρέχει λειτουργίες που δεν είναι διαθέσιμες στον *ns-2*, όπως ένα περιβάλλον εκτέλεσης κώδικα εφαρμογής (επιτρέποντας στους χρήστες να τρέχουν τον πραγματικό κώδικα της εφαρμογής στον προσομοιωτή).
- Ο *ns-3* παρέχει ένα χαμηλότερο επίπεδο βάσης της αφαίρεσης σε σύγκριση με *ns-2*, επιτρέποντάς τον να προσαρμοστεί καλύτερα με το πώς πραγματικά τα συστήματα τοποθετούνται μαζί. Κάποιοι περιορισμοί που βρέθηκαν στον *ns-2* (όπως η σωστή υποστήριξη πολλαπλών τύπων διεπαφών στους κόμβους) έχουν διορθωθεί στον *ns-3*.

Ο *ns-2* έχει ένα πιο διαφοροποιημένο σύνολο που συνέβαλαν στις ενότητες από ό,τι κάνει ο *ns-3*, λόγω της μακράς ιστορίας του. Ωστόσο, ο *ns-3* έχει πιο λεπτομερή μοντέλα σε διάφορες δημοφιλείς περιοχές της έρευνας (συμπεριλαμβανομένων εξελιγμένα μοντέλα LTE και WiFi), και η υποστήριξη της εφαρμογής του κώδικα αναγνωρίζει ένα πολύ ευρύ φάσμα μοντέλων υψηλής πιστότητας. Οι χρήστες μπορούν να εκπλαγούν όταν μάθουν ότι ολόκληρη η στοίβα δικτύου του Linux μπορεί να περιοριστεί σε ένα *ns-3* κόμβο, χρησιμοποιώντας την άμεση εκτέλεση κώδικα (Direct Code Execution - DCE) πλαίσιο. Τα μοντέλα του *ns-2* μπορούν μερικές φορές να μεταφερθούν και στον *ns-3*, συγκεκριμένα όταν έχουν υλοποιηθεί σε C++.

Σε περίπτωση αμφιβολίας, μια καλή συμβουλή θα ήταν να δούμε τους δύο προσομοιωτές (καθώς και άλλους προσομοιωτές), και κυρίως τα διαθέσιμα μοντέλα για την έρευνά σας, αλλά να έχετε κατά νου ότι η εμπειρία σας μπορεί να είναι καλύτερη χρησιμοποιώντας το εργαλείο που είανι ενεργά αναπτυσσόμενο και διατηρείται (*ns-3*).

## 1.3 Συνεισφορά

Ο *ns-3* είναι ένας ερευνητικός και εκπαιδευτικός προσομοιωτής, από και για την ερευνητική κοινότητα. Θα βασίζεται στις τρέχουσες εισφορές της κοινότητας για την ανάπτυξη νέων μοντέλων, διόρθωση ή διατήρηση των υπαρχόντων, και το μοίρασμα των αποτελεσμάτων. Υπάρχουν λίγες πολιτικές που ελπίζουμε ότι θα ενθαρρύνει τους ανθρώπους να συμβάλλουν στον *ns-3* όπως έχουν συμβάλλει για τον *ns-2*:

- Αδειοδότηση ανοιχτού κώδικα με βάση τη συμβατότητα του GNU GPLv2
- wiki
- Σελίδα Κώδικα Συνεισφοράς, παρόμοια με τη δημοφιλή σελίδα του *ns-2* Κώδικα Συνεισφοράς σελίδα
- Άνοιξε bug tracker

Αντιλαμβανόμαστε ότι, αν διαβάζετε αυτό το έγγραφο, συμβάλλοντας πίσω στο έργο είναι πιθανόν να μην είναι η κύρια ανησυχία σας σε αυτό το σημείο, αλλά θέλουμε να γνωρίζετε ότι η συνεισφορά είναι στο πνεύμα του έργου και ότι ακόμη και η πράξη της εγκατάλειψής μας μια σημείωση για την πρώιμη εμπειρία σας με *ns-3* (π.χ. «αυτό το τμήμα του οδηγού δεν ήταν σαφές ...”), reports σχετικά με το έγγραφο που εργάζεστε, κλπ. θα ήταν εκτιμήσιμο.

## 1.4 Οδηγός Οργάνωσης

Ο οδηγός υποθέτει ότι οι νέοι χρήστες αρχικά θα ακολουθήσουν μία απο τις παρακάτω ιστοσελίδες:

- Προσπαθήστε να κατεβάσετε και να χτίσετε ένα αντίγραφο,
- Προσπαθήστε να τρέξετε μερικά δείγματα-προγράμματα,
- Κοιτάξτε στην έξοδο προσομοίωσης, και να προσπαθήστε να το ρυθμίσετε.

Ως αποτέλεσμα, έχουμε προσπαθήσει να οργανώσουμε τον οδηγό σύμφωνα με τα παραπάνω με ευρείες ακολουθίες γεγονότων.

ΚΕΦΑΛΑΙΟ

# ΔΥΟ

# ΠΟΡΟΙ

## 2.1 Στο Διαδίκτυο

Υπάρχουν διάφοροι σημαντικοί πόροι τους οποίους πρέπει να γνωρίζει κάθε χρήστης του *ns-3*. Ο κύριος ιστότοπος βρίσκεται στη διεύθυνση http://www.nsnam.org και παρέχει πρόσβαση σε βασικές πληροφορίες σχετικά με το σύστημα του *ns-3*. Λεπτομερής τεκμηρίωση είναι διαθέσιμη μέσω του κύριου ιστότοπου στη διεύθυνση http://www.nsnam.org/documentation/. Από εκεί μπορείτε επίσης να βρείτε έγγραφα που σχετίζονται με την αρχιτεκτονική του συστήματος.

Υπάρχει και ένα Wiki που λειτουργεί συμπληρωματικά ως προς τον κύριο ιστότοπο του *ns-3*, το οποίο θα βρείτε στη διεύθυνση http://www.nsnam.org/wiki/. Εκεί θα βρείτε και FAQ (Frequently Asked Questions, δηλαδή μια λίστα με συχνές ερωτήσεις) για χρήστες και προγραμματιστές, καθώς επίσης και οδηγούς αντιμετώπισης προβλημάτων, κώδικα που έχουν συνεισφέρει άλλοι, papers, κτλ.

Ο πηγαίος κώδικας βρίσκεται και μπορείτε να τον δείτε μέσω της διεύθυνσης http://code.nsnam.org/. Εκεί θα βρείτε και το τρέχον δέντρο ανάπτυξης στο αποθετήριο με όνομα `ns-3-dev`. Εκεί θα βρείτε επίσης παλιές εκδόσεις και πειραματικά αποθετήρια των βασικών προγραμματιστών του *ns-3*.

## 2.2 Mercurial

Τα πολύπλοκα συστήματα λογισμικού απαιτούν κάποιο τρόπο διαχείρισης της οργάνωσης και των αλλαγών στον βασικό τους κώδικα και στην τεκμηριώση. Υπάρχουν πολλοί τρόποι για να πραγματοποιηθεί αυτό το δύσκολο έργο, και μπορεί να έχετε ακούσει μερικά από τα συστήματα που χρησιμοποιούνται αυτή τη στιγμή για να γίνει αυτό. Το Concurrent Version System (CVS) είναι μάλλον το πιο γνωστό.

Το project του *ns-3* χρησιμοποιεί το Mercurial ως σύστημα διαχείρισης του πηγαίου κώδικά του. Παρόλο που δε χρειάζεται να γνωρίζετε πολλά σχετικά με το Mercurial ώστε να ολοκληρώσετε τον παρόντα οδηγό, εμείς θα σας προτείναμε να εξοικειωθείτε με το Mercurial και να το χρησιμοποιείτε ώστε να προσπελαύνετε τον πηγαίο κώδικα. Ο ιστότοπος του Mercurial βρίσκεται στη διεύθυνση http://www.selenic.com/mercurial/, απ' όπου μπορείτε να κατεβάσετε διάφορες εκδόσεις αυτού του Software Configuration Management (SCM) συστήματος, είτε σε εκτελέσιμη (δυαδική ή αγγλιστί binary) μορφή είτε ως πηγαίο κώδικα. Ο Selenic (ο προγραμματιστής του Mercurial) παρέχει επίσης έναν οδηγό στη διεύθυνση http://www.selenic.com/mercurial/wiki/index.cgi/Tutorial/, και έναν πιο σύντομο οδηγό στη διεύθυνση http://www.selenic.com/mercurial/wiki/index.cgi/QuickStart/.

Μπορείτε να βρείτε επίσης σημαντικές πληροφορίες σχετικά με τη χρήση του Mercurial και του *ns-3* στον κύριο ιστότοπο του *ns-3*.

## 2.3 Waf

Μόλις κατεβάσετε τον πηγαίο κώδικα στο σύστημά σας, θα χρειαστεί να μεταγλωττίσετε αυτόν τον πηγαίο κώδικα για να παράξετε χρήσιμα προγράμματα. Όπως και στην περίπτωση της διαχείρισης του πηγαίου κώδικα,

υπάρχουν πολλά διαθέσιμα εργαλεία για να εκτελεστεί αυτή η λειτουργία. Πιθανώς το πιο γνωστό από αυτά τα εργαλεία να είναι το `make`. Εκτός του ότι είναι το πιο γνωστό, το `make` είναι μάλλον και το πιο δύσχρηστο για ένα μεγάλο και πολύ παραμετροποιήσιμο σύστημα. Λόγω αυτού, έχουν αναπτυχθεί πολλά εναλλακτικά εργαλεία/συστήματα. Πρόσφατα, μάλιστα, τέτοια συστήματα αναπτύχθηκαν με χρήση της γλώσσας Python.

Το σύστημα build (κατασκευής) Waf χρησιμοποιείται για το project του *ns-3*. Είναι ένα σύστημα από τη νέα γενιά συστημάτων build που βασίζονται στην Python. Δε θα χρειαστεί να γνωρίζετε κάτι από Python για να κάνετε build το υπάρχον *ns-3* σύστημα.

Για όσους ενδιαφέρονται για τις βαθύτερες λεπτομέρειες του Waf, ο κύριος ιστότοπός του μπορεί να βρεθεί στη διεύθυνση http://code.google.com/p/waf/.

## 2.4 Περιβάλλον Ανάπτυξης

Όπως αναφέρθηκε και παραπάνω, η συγγραφή στον *ns-3* γίνεται σε C++ ή Python. Το μεγαλύτερο μέρος του API του *ns-3* είναι διαθέσιμο σε Python, αλλά σε κάθε περίπτωση τα μοντέλα είναι γραμμένα σε C++. Η έμπρακτη γνώση της C++ και εννοιών αντικειμενοστραφούς προγραμματισμού θεωρείται δεδομένη για το παρόν έγγραφο. Θα αφιερώνουμε λίγο χρόνο για κάνουμε μια ανασκόπηση κάποιων από τις πιο προχωρημένες έννοιες ή πιθανότατα κάποιων άγνωστων χαρακτηριστικών της γλώσσας, ιδιωμάτων και σχεδιαστικών προτύπων όταν αυτά θα προκύπτουν. Παρόλ' αυτά, δε θέλουμε αυτός ο οδηγός να καταλήξει να είναι οδηγός της C++, οπότε περιμένουμε από εσάς να είστε σε θέση να χρησιμοποιείτε τη γλώσσα σε ένα βασικό επίπεδο. Υπάρχει ένας απίστευτος αριθμός από πηγές πληροφοριών σχετικά με τη C++ που είναι διαθέσιμες στο διαδίκτυο ή σε έντυπη μορφή.

Αν ξεκινάτε τώρα με τη C++, ίσως θελήσετε να βρείτε έναν οδηγό ή ένα βιβλίο με «γρήγορες συμβουλές» ή έναν ιστότοπο, και να εξασκηθείτε τουλάχιστον στα βασικά χαρακτηριστικά της γλώσσας, πριν να προχωρήσετε. Για παράδειγμα, μπορείτε να δείτε αυτόν τον οδηγό.

Το σύστημα του *ns-3* χρησιμοποιεί διάφορα μέρη από την «εργαλειοθήκη» του GNU για προγραμματισμό. Μια εργαλειοθήκη λογισμικού είναι ένα σύνολο από προγραμματιστικά εργαλεία που είναι διαθέσιμα σε ένα δεδομένο περιβάλλον. Για μια γρήγορη επισκόπηση του τι περιλαμβάνει η εργαλειοθήκη του GNU δείτε στη διεύθυνση http://en.wikipedia.org/wiki/GNU_toolchain. Ο *ns-3* χρησιμοποιεί τα gcc, GNU binutils, και gdb. Ωστόσο, δε χρησιμοποιούμε τα εργαλεία κατασκευής συστήματος του GNU, ούτε το make ούτε το autotools. Εμείς χρησιμοποιούμε το Waf για αυτές τις λειτουργίες.

Τυπικά, ένας προγραμματιστής του *ns-3* θα εργαστεί σε κάποιο περιβάλλον Linux ή παρόμοιο με Linux. Για εκείνους που εργάζονται σε Windows, υπάρχουν περιβάλλοντα που προσομοιώνουν το περιβάλλον του Linux σε διάφορες βαθμίδες. Το project του *ns-3* έχει υποστηρίξει στο παρελθόν (αλλά δεν υποστηρίζει πλέον) τον προγραμματισμό στο περιβάλλον του Cygwin για αυτούς τους χρήστες. Δείτε στη διεύθυνση http://www.cygwin.com/ για λεπτομέρειες σχετικά με τη λήψη του, και επισκεφθείτε το wiki του *ns-3* για περισσότερες πληροφορίες πάνω στο Cygwin και στον *ns-3*. Το MinGW δεν υποστηρίζεται επίσημα αυτή τη στιγμή. Μια άλλη εναλλακτική για το Cygwin είναι η εγκατάσταση ενός περιβάλλοντος εικονικών μηχανών, όπως είναι ο εξυπηρετητής VMware και η εγκατάσταση μια εικονικής μηχανής Linux.

## 2.5 Προγραμματισμός Sockets

Στα παραδείγματα που παρουσιάζονται σε αυτόν τον οδηγό θα θεωρήσουμε ότι έχετε κάποια βασική ευχέρεια με το Berkeley Sockets API. Αν δε γνωρίζετε κάτι σχετικά με τα sockets, θα συνιστούσαμε να ανατρέξετε στο API και σε κάποια τυπικά παραδείγματα χρήσης. Για μια καλή επισκόπηση του προγραμματισμού για TCP/IP sockets συνιστούμε το TCP/IP Sockets in C, Donahoo and Calvert.

Υπάρχει ένας αντίστοιχος ιστότοπος που περιλαμβάνει τον πηγαίο κώδικα από τα παραδείγματα στο βιβλίο, τον οποίο μπορείτε να βρείτε εδώ: http://cs.baylor.edu/~donahoo/practical/CSockets/.

Εάν καταλάβετε τα πρώτα τέσσερα κεφάλαια αυτού του βιβλίου (ή για όσους δεν έχουν πρόσβαση σε κάποιο αντίτυπο του βιβλίο, εάν κατανοήσετε τους πελάτες και εξυπηρετητές echo που αναφέρονται στον παραπάνω

ιστότοπο), τότε θα είστε σε καλή θέση ώστε να καταλάβετε και τον οδηγό. Υπάρχει ένα παρόμοιο βιβλίο πάνω στα Multicast Sockets, το Multicast Sockets των Makofske και Almeroth, το οποίο καλύπτει υλικό που ίσως πρέπει να κατανοήσετε, εάν θέλετε να δείτε και τα παραδείγματα αναφορικά με τα multicast μέσα στη διανομή.

ΚΕΦΑΛΑΙΟ

# ΤΡΙΑ

# ΞΕΚΙΝΩΝΤΑΣ

Η ενότητα αυτή έχει στόχο να πάρει ένα χρήστη σε μια κατάσταση εργασίας ξεκινώντας με μια μηχανή που μπορεί να μην είχε ποτέ εγκαταστήσει τον *ns-3*. Καλύπτει υποστηριζόμενες πλατφόρμες, προϋποθέσεις, τους τρόπους να αποκτήσετε τον *ns-3*, τους τρόπους για να οικοδομήσετε τον *ns-3*, και τρόπους για να ελέγξετε την κατασκευή σας και να τρέξετε απλά προγράμματα.

## 3.1 Επισκόπηση

Ο *ns-3* είναι χτισμένος ως ένα σύστημα βιβλιοθηκών λογισμικού που λειτουργούν μαζί. Τα προγράμματα χρηστών μπορούν να είναι γραμμένα τα οποία συνδέουν με (ή τις εισάγουν από) αυτές τις βιβλιοθήκες. Τα προγράμματα χρηστών γράφονται είτε σε C++ ή Python γλώσσες προγραμματισμού.

Ο *ns-3* διανέμεται ως πηγαίος κώδικας, που σημαίνει ότι ο στόχος του συστήματος πρέπει να έχει ένα περιβάλλον ανάπτυξης λογισμικού για την κατασκευή αρχικά των βιβλιοθηκών, μετά χτίζετε το πρόγραμμα του χρήστη. Ο *ns-3* θα μπορούσε αρχικά να διανέμεται ως προ-κατασκευασμένες βιβλιοθήκες για επιλεγμένα συστήματα, και στο μέλλον μπορεί να διανεμηθεί με αυτόν τον τρόπο, αλλά προς το παρόν, πολλοί χρήστες πραγματικά κάνουν τη δουλειά τους με την επεξεργασία του *ns-3* όπως είναι, έτσι έχοντας τον πηγαίο κώδικα γύρω από την ανοικοδόμηση οι βιβλιοθήκες είναι χρήσιμες. Αν κάποιος θα ήθελε να αναλάβει την δουλειά προ-χτίζοντας βιβλιοθήκες και πακέτα για λειτουργικά συστήματα, παρακαλούμε επικοινωνήστε με τους NS-προγραμματιστές στην ενημερωτική λίστα.

Στη συνέχεια, θα δούμε δύο τρόπους για τη λήψη και την οικοδόμηση του *ns-3*. Το πρώτο είναι να κατεβάσετε και να οικοδομήσετε μια επίσημη έκδοση από την κύρια ιστοσελίδα. Το δεύτερο είναι να φέρετε και να οικοδομήσετε την ανάπτυξη αντιγράφων του *ns-3*. Θα δούμε δύο παραδείγματα καθώς τα εργαλεία που περιέχονται είναι λίγο διαφορετικά.

## 3.2 Κατεβάζοντας τον *ns-3*

Το σύστημα του *ns-3* στο σύνολό του είναι ένα αρκετά πολύπλοκο σύστημα και έχει έναν αριθμό εξαρτήσεων από άλλες συνιστώσες. Μαζί με τα συστήματα που πιθανότατα θα ασχολείστε κάθε μέρα (η εργαλειοθήκη GNU, Mercurial, έναν επεξεργαστή κειμένου - editor) θα χρειαστεί να εξασφαλίσετε ότι είναι παρών στο σύστημά σας πριν προχωρήσετε μια σειρά από πρόσθετες βιβλιοθήκες. Ο *ns-3* παρέχει μία wiki σελίδα που περιλαμβάνει σελίδες με πολλές χρήσιμες συμβουλές και υποδείξεις. Μια τέτοια σελίδα είναι η σελίδα «Εγκατάσταση», http://www.nsnam.org/wiki/Installation.

Η ενότητα “Προϋποθέσεις” αυτής της σελίδας του wiki εξηγεί ποια πακέτα απαιτούνται για την υποστήριξη κοινών επιλογών *ns-3*, και επίσης παρέχει τις εντολές που χρησιμοποιούνται για την εγκατάσταση των κοινών παραλλαγών του Linux. Οι χρήστες του Cygwin θα πρέπει να χρησιμοποιήσουν το πρόγραμμα εγκατάστασης Cygwin (αν είστε χρήστης Cygwin, το χρησιμοποιήσατε για να εγκαταστήστε Cygwin).

Μπορεί να θέλετε να εκμεταλλευτείτε αυτή την ευκαιρία για να εξερευνήσετε τον *ns-3* στο wiki λίγο δεδομένου ότι υπάρχει πραγματικά μια πληθώρα πληροφοριών εκεί.

Από αυτό το σημείο προς τα εμπρός, πρόκειται να υποθέσουμε ότι ο αναγνώστης εργάζεται σε Linux ή σε ένα περιβάλλον εξομοίωσης Linux (Linux, Cygwin, κ.λπ.) και έχει εγκατεστημένη την GNU εργαλειοθήκη και έχει επαληθεύσει μαζί με τις προϋποθέσεις που αναφέρονται παραπάνω. Επίσης, πρόκειται να υποθέσουμε ότι έχετε το Mercurial και το Waf εγκατεστημένο και τρέχουν στο κυρίως σύστημα.

Ο *ns-3* κώδικας είναι διαθέσιμος σε Mercurial αποθετήρια στον διακομιστή http://code.nsnam.org. Μπορείτε επίσης να κατεβάσετε μία tarball(συμπιεσμένη) έκδοση στο http://www.nsnam.org/release/, ή μπορείτε να εργαστείτε με τα αρχεία καταγραφής(αποθετήρια) χρησιμοποιώντας Mercurial. Σας προτείνουμε να χρησιμοποιείτε το Mercurial, εκτός αν υπάρχει ένας καλός λόγος για να μην τον χρησιμοποιήσετε. Δείτε το τέλος αυτής της ενότητας για οδηγίες σχετικά με το πώς να πάρετε μία συμπιεσμένη έκδοση.

Ο απλούστερος τρόπος για να ξεκινήσετε τη χρήση αποθετήρια του Mercurial είναι να χρησιμοποιήσετε το " ns-3-allinone" περιβάλλον. Πρόκειται για μια σειρά από σενάρια που διαχειρίζεται το κατέβασμα και την κατασκευή των διαφόρων υποσυστημάτων του *ns-3* για σένα. Συνιστούμε να ξεκινήσετε την εργασία *ns-3* σε αυτό το περιβάλλον.

Μια πρακτική είναι να δημιουργήσετε ένα κατάλογο με το όνομα `workspace` στην αρχή κάποιου καταλόγου βάσει του οποίου μπορεί κανείς να κρατήσει τα τοπικά Mercurial αποθετήρια. Κάθε όνομα καταλόγου θα κάνει, αλλά υποθέτουμε ότι το `workspace` χρησιμοποιείται εδώ (σημειώστε: `repos` μπορεί επίσης να χρησιμοποιηθεί σε κάποια τεκμηρίωση ως ένα όνομα καταλόγου παράδειγμα).

### 3.2.1 Κατεβάζοντας τον *ns-3* χρησιμοποιώντας Tarball

Ένα tarball είναι μία συγκεκριμένη μορφή λογισμικού αρχείο, όπου πολλαπλά αρχεία ομαδοποιούνται και το αρχείο ενδεχομένως να συμπιέζεται. Οι εκδόσεις λογισμικού του *ns-3* παρέχονται μέσω ενός tarball - συμπιεσμένο αρχείο. Η διαδικασία για τη λήψη του *ns-3* μέσω tarball είναι απλή: απλά πρέπει να επιλέξετε μία έκδοση, να το κατεβάσετε και να το αποσυμπιέσετε αυτό.

Ας υποθέσουμε ότι εσείς, ως χρήστης, επιθυμείτε να δημιουργήσετε τον *ns-3* σε έναν τοπικό κατάλογο με την ονομασία `workspace`. Εάν έχετε υιοθετήσει την προσέγγιση του καταλόγου `workspace`, μπορείτε να πάρετε ένα αντίγραφο της έκδοσης, πληκτρολογώντας τα εξής στο κέλυφος του Linux σας (αντικαθιστάτε τους κατάλληλους αριθμούς έκδοσης, φυσικά):

```
$ cd
$ mkdir workspace
$ cd workspace
$ wget http://www.nsnam.org/release/ns-allinone-3.22.tar.bz2
$ tar xjf ns-allinone-3.22.tar.bz2
```

Εάν αλλάξετε μέσα στον κατάλογο `ns-allinone-3.22` θα πρέπει να δείτε έναν αριθμό αρχείων:

```
$ ls
bake      constants.py   ns-3.22               README
build.py  netanim-3.105  pybindgen-0.16.0.886  util.py
```

Τώρα είστε έτοιμοι να οικοδομήσουμε τη διανομή της βάσης του *ns-3*.

### 3.2.2 Κατεβάζοντας τον *ns-3* χρησιμοποιώντας Bake

Ο Bake είναι ένα εργαλείο για κατανεμημένη ολοκλήρωση και οικοδόμηση, που αναπτύχθηκε για το έργο του *ns-3*. Ο Bake μπορεί να χρησιμοποιηθεί για να φέρει αναπτυγμένες εκδόσεις στο λογισμικό του *ns-3*, και να κατεβάσετε και να οικοδομήσετε επεκτάσεις στη διανομή της βάσης του *ns-3*, όπως το περιβάλλον Εκτέλεσης Άμεση Κώδικα(Direct Code Execution environment), Δίκτυο Προσομοίωσης λίκνο(Network Simulation Cradle), την ικανότητα να δημιουργήσετε νέες συνδέσεις Python, και άλλα.

Σε πρόσφατες εκδόσεις *ns-3*, ο Bake έχει συμπεριληφθεί στο tarball της έκδοσης. Το αρχείο των ρυθμίσεων που περιλαμβάνονται στην επίσημη έκδοση θα επιτρέψει σε κάποιον να κατεβάσει οποιοδήποτε λογισμικό που ίσχυε κατά τη στιγμή της έκδοσης. Δηλαδή, για παράδειγμα, η έκδοση του Bake που διανέμεται με την έκδοση `ns-3.22` μπορεί να χρησιμοποιηθεί για να φέρει τα συστατικά για αυτήν την έκδοση *ns-3* ή και παλαιότερη, αλλά δεν μπορεί να χρησιμοποιηθεί για να φέρει τα συστατικά για νεότερες εκδόσεις (εκτός εάν το αρχείο `bakeconf.xml` είναι ενημερωμένο).

Μπορείτε επίσης να πάρετε το πιο πρόσφατο αντίγραφο του `bake` πληκτρολογώντας το παρακάτω στο κέλυφος του Linux σας (αν έχετε εγκαταστήσει το Mercurial)

```
$ cd
$ mkdir workspace
$ cd workspace
$ hg clone http://code.nsnam.org/bake
```

Καθώς η hg (Mercurial) εντολή εκτελείται, θα πρέπει να δείτε κάτι να εμφανίζεται σαν το ακόλουθο,:

```
...
destination directory: bake
requesting all changes
adding changesets
adding manifests
adding file changes
added 339 changesets with 796 changes to 63 files
updating to branch default
45 files updated, 0 files merged, 0 files removed, 0 files unresolved
```

Αφού ολοκληρωθεί η εντολή κλώνος, θα πρέπει να έχετε έναν κατάλογο που ονομάζεται `bake`, τα περιεχόμενα της οποίας θα πρέπει να δούμε κάτι σαν το παρακάτω

```
$ ls
bake                  bakeconf.xml  doc       generate-binary.py  TODO
bake.py               examples      test
```

Σημειώστε ότι πραγματικά κατεβάσατε μερικά σενάρια Python και μία ενότητα Python που ονομάζεται `bake`. Το επόμενο βήμα είναι να χρησιμοποιηθούν αυτά τα σενάρια για να κατεβάσετε και να οικοδομήσετε την κατανομή του *ns-3* της επιλογής σας.

Υπάρχουν μερικές ρυθμίσεις ακόμα:

1. `ns-3.22`: η ενότητα που αντιστοιχεί στην έκδοση, θα κατεβάσει συστατικά παρόμοια με την έκδοση tarball.
2. `ns-3-dev`: μια παρόμοια ενότητα, αλλά χρησιμοποιώντας τον κώδικα-δένδρο ανάπτυξης
3. `ns-allinone-3.22`: η ενότητα που περιλαμβάνει άλλα προαιρετικά χαρακτηριστικά όπως την δρομολόγηση του κλικ, τον openflow *ns-3*, και Προσομοίωση Δικτύων λίκνο(Network Simulation Cradle)
4. `ns-3-allinone`: παρόμοια με την επίσημη έκδοση του allinone, αλλά για την ανάπτυξη κώδικα.

Το τρέχον στιγμιότυπο ανάπτυξης (ακυκλοφόρητο) του *ns-3* μπορεί να βρεθεί στο http://code.nsnam.org/ns-3-dev/. Οι προγραμματιστές προσπαθούν να κρατήσουν αυτά τα αποθετήρια με συνέπεια, εργάζοντας κομμάτια αλλά είναι σε μια περιοχή ανάπτυξης με ακυκλοφόρητο κώδικα προσωρινά, οπότε μπορεί να θέλετε να εξετάσετε την διαμονή σας με την επίσημη έκδοση, εφόσον δεν χρειάζεστε νεοεισαχθέν χαρακτηριστικά.

Μπορείτε να βρείτε την τελευταία έκδοση του κώδικα είτε με την επιθεώρηση της λίστας του χώρου αποθήκευσης ή πηγαίνοντας στην ιστοσελίδα "ns-3 Releases" και κάνοντας κλίκ στον σύνδεσμο της τελευταίας έκδοσης. Θα προχωρήσουμε σε αυτόν τον οδηγό με παράδειγμα τον `ns-3.22`.

Πάμε να χρησιμοποιήσουμε το εργαλείο bake για να χωρίσουμε τα διάφορα κομμάτια του *ns-3* που θα χρησιμοποιείτε. Κατ 'αρχάς, θα πούμε μία λέξη για το τρέξιμο του bake.

Ο bake λειτουργεί κατεβάζοντας πακέτα πηγαίου κώδικα σε έναν κατάλογο πηγή, και εγκαθηστώντας βιβλιοθήκες σε έναν κατάλογο κατασκευής. Ο bake μπορεί να τρέξει με την παραπομπή του δυαδικού, αλλά αν κάποιος

επιλέξει να τρέξει τον bake από το εξωτερικό του καταλόγου απο το οποίο έγινε λήψη, είναι συμβουλή να τοποθετήσετε το bake στη διαδρομή(path) που ξέρετε, όπως ακολουθεί (Linux κέλυφος bash παράδειγμα). Πρώτον, να αλλάξετε μέσα στον κατάλογο 'bake', και στη συνέχεια ορίστε τις ακόλουθες μεταβλητές περιβάλλοντος:

```
$ export BAKE_HOME=`pwd`
$ export PATH=$PATH:$BAKE_HOME:$BAKE_HOME/build/bin
$ export PYTHONPATH=$PYTHONPATH:$BAKE_HOME:$BAKE_HOME/build/lib
```

Αυτό θα θέσει το πρόγραμμα bake.py στη διαδρομή του κελύφους, και θα επιτρέψει άλλα προγράμματα να τα βρείτε εκτελέσιμα και τις βιβλιοθήκες που έχουν δημιουργηθεί από το bake. Παρά το γεγονός ότι αρκετές περιπτώσεις το bake δεν απαιτείται η χρήση ρύθμιση της διαδρομής και PYTHONPATH όπως παραπάνω, η πλήρης έκδοση του ns-3-allinone (με τα προαιρετικά πακέτα) συνήθως χρειάζεται.

Μπείτε στο κατάλογο εργασίας και πληκτρολογήστε τα ακόλουθα στο κέλυφος

```
$ ./bake.py configure -e ns-3.22
```

Στη συνέχεια, εμείς θα ζητήσουμε από το bake να ελέγξει αν έχουμε αρκετά εργαλεία για να κατεβάσουμε διάφορα συστατικά. Τύπος

```
$ ./bake.py check
```

Θα πρέπει να δείτε κάτι όπως παρακάτω,

```
> Python - OK
> GNU C++ compiler - OK
> Mercurial - OK
> CVS - OK
> GIT - OK
> Bazaar - OK
> Tar tool - OK
> Unzip tool - OK
> Unrar tool - is missing
> 7z  data compression utility - OK
> XZ data compression utility - OK
> Make - OK
> cMake - OK
> patch tool - OK
> autoreconf tool - OK

> Path searched for tools: /usr/lib64/qt-3.3/bin /usr/lib64/ccache
/usr/local/bin /bin /usr/bin /usr/local/sbin /usr/sbin /sbin
/home/tomh/bin bin
```

Ειδικότερα, λήψη εργαλείων όπως το Mercurial, CVS, GIT και Bazaar είναι οι κυριότερες ανησυχίες μας σε αυτό το σημείο, διότι μας επιτρέπουν να φέρουν τον κώδικα. Παρακαλώ εγκαταστήστε τα εργαλεία που λείπουν σε αυτό το στάδιο, με το συνήθη τρόπο για το σύστημά σας (αν είστε σε θέση), ή επικοινωνήστε με τον διαχειριστή του συστήματός σας, όπως απαιτείται για την εγκατάσταση αυτών των εργαλείων.

Μετά, προσπαθήστε να κατεβάσετε το λογισμικό

```
$ ./bake.py download
```

θα πρέπει να δώσει κάτι, όπως

```
>> Searching for system dependency pygoocanvas - OK
>> Searching for system dependency python-dev - OK
>> Searching for system dependency pygraphviz - OK
>> Downloading pybindgen-0.16.0.886 - OK
>> Searching for system dependency g++ - OK
>> Searching for system dependency qt4 - OK
```

```
>> Downloading netanim-3.105 - OK
>> Downloading ns-3.22 - OK
```

Απο τα ανωτέρω προκύπτει ότι οι τρείς πηγές έχουν ληφθεί. Ελέγξτε τώρα τον κατάλογο `source` και πληκτρολογείστε `ls`, πρέπει να φανεί

```
$ ls
netanim-3.105  ns-3.22  pybindgen-0.16.0.886
```

Είστε έτοιμη για την κατασκευή της διανομής του *ns-3*.

## 3.3 Χτίζοντας τον *ns-3*

### 3.3.1 Χτίζοντας με `build.py`

Δουλεύοντας απο μία έκδοση tarball, η πρώτη φορά που θα κατασκευάσετε την εργασία *ns-3* μπορείτε να δημιουργήσετε χρησιμοποιώντας ένα εύχρηστο πρόγραμμα που θα βρείτε στον κατάλογο `allinone`. Αυτό το πρόγραμμα ονομάζεται `build.py`. Αυτό το πρόγραμμα θα πάρει την ρυθμισμένη εργασία για εσάς στο πιο χρήσιμο τρόπο. Ωστόσο, παρακαλούμε να σημειώσετε ότι πιο προηγμένες ρυθμίσεις και εργασίες με τον *ns-3* τυπικά περιλαμβάνει τη χρήση του φυσικού συστήματος κατασκευής *ns-3*, Waf, στο οποίο θα εισαχθούμε αργότερα στον οδηγό αυτό.

Αν έχετε κατεβάσει χρησιμοποιώντας ένα tarball θα πρέπει να έχετε έναν κατάλογο που ονομάζεται `ns-allinone-3.22` κάτω από τον κατάλογο `~/workspace`. Πληκτρολογήστε την ακόλουθη εντολή

```
$ ./build.py --enable-examples --enable-tests
```

Επειδή εργαζόμαστε με παραδείγματα και δοκιμές σε αυτόν τον οδηγό, και επειδή δεν έχουν κατασκευαστεί από προεπιλογή στον *ns-3*, τα ορίσματα για build.py λέει να τα κατασκευάσουμε για εμάς. Το πρόγραμμα, επίσης, αποτυγχάνει την οικοδόμηση όλων των διαθέσιμων ενοτήτων. Αργότερα, μπορείτε να χτίσετε τον *ns-3* χωρίς παραδείγματα και δοκιμές, ή την εξάλειψη των ενοτήτων που δεν είναι απαραίτητα για την εργασία σας, εάν το επιθυμείτε.

Θα δείτε πολλά μηνύματα εξόδου τυπικού compiler να εμφανίζονται όσο το σενάριο κατασκευής χτίζει τα διάφορα κομμάτια που κατεβάσατε. Ενδεχομένως να δείτε το παρακάτω

```
Waf: Leaving directory `/path/to/workspace/ns-allinone-3.22/ns-3.22/build'
'build' finished successfully (6m25.032s)

Modules built:
antenna                  aodv                     applications
bridge                   buildings                config-store
core                     csma                     csma-layout
dsdv                     dsr                      energy
fd-net-device            flow-monitor             internet
lr-wpan                  lte                      mesh
mobility                 mpi                      netanim (no Python)
network                  nix-vector-routing       olsr
point-to-point           point-to-point-layout    propagation
sixlowpan                spectrum                 stats
tap-bridge               test (no Python)         topology-read
uan                      virtual-net-device       wave
wifi                     wimax

Modules not built (see ns-3 tutorial for explanation):
brite                    click                    openflow
```

```
visualizer

Leaving directory `./ns-3.22'
```

Όσον αφορά το τμήμα σχετικά με τις ενότητες δεν χτίστηκε

```
Modules not built (see ns-3 tutorial for explanation):
brite                     click                     openflow
visualizer
```

Αυτό σημαίνει απλά ότι κάποιες ενότητες του *ns-3* που έχουν εξαρτήσεις σε εξωτερικές βιβλιοθήκες μπορεί να μην έχουν κατασκευαστεί, ή ότι η διαμόρφωση ζήτησε συγκεκριμένα να μην τις κατασκευάσει. Αυτό δεν σημαίνει ότι ο προσομοιωτής δεν έχτισε με επιτυχία ή ότι θα παρέχει λανθασμένα αποτελέσματα για τις ενότητες που αναφέρονται καθώς χτίζονται.

### 3.3.2 Χτίζοντας με Bake

Εάν χρησιμοποιείτε bake παραπάνω για να φέρετε τον πηγαίο κώδικα από τα αποθετήρια εργασιών, μπορείτε να συνεχίσετε να το χρησιμοποιήσετε για να οικοδομήσετε τον *ns-3*. Πληκτρολογήστε:

```
$ ./bake.py build
```

και θα πρέπει να δείτε

```
>> Building pybindgen-0.16.0.886 - OK
>> Building netanim-3.105 - OK
>> Building ns-3.22 - OK
```

*Συμβουλή: Μπορείτε επίσης να εκτελέσετε δύο βήματα, να κατεβάσετε και να οικοδομήσετε καλώντας 'bake.py deploy'.*

Αν συμβαίνει να υπάρχει μια αποτυχία, παρακαλώ ρίξτε μια ματιά στα ακόλουθα, μπορεί να δώσει μια υπόδειξη ως προς τι λείπει:

```
$ ./bake.py show
```

Αυτό θα εμφανίσει τις διάφορες εξαρτήσεις των πακέτων που προσπαθούμε να οικοδομήσουμε.

### 3.3.3 Χτίζοντας με Waf

Μέχρι αυτό το σημείο, έχουμε χρησιμοποιήσει είτε το σενάριο *build.py*, ή το εργαλείο *bake*, για να ξεκινήσετε την οικοδόμηση του *ns-3*. Τα εργαλεία αυτά είναι χρήσιμα για την ανάπτυξη του *ns-3* και την υποστήριξη βιβλιοθηκών, και καλούν μέσω του καταλόγου του *ns-3* το εργαλείο Waf να κάνει την πραγματική οικοδόμηση. Οι περισσότεροι χρήστες κάνουν την μετάβαση γρήγορα χρησιμοποιώντας άμεσα τον Waf για να διαμορφώσουν και να οικοδομήσουμε τον *ns-3*. Έτσι, για να προχωρήσει, παρακαλούμε να αλλάξετε τον κατάλογο εργασίας σας με τον κατάλογο του *ns-3* που έχετε αρχικά κατασκευάσει.

Δεν είναι απολύτως απαραίτητο σε αυτό το σημείο, αλλά θα είναι χρήσιμο να κάνουμε μια μικρή παράκαμψη και να δούμε πώς να κάνετε αλλαγές στη διαμόρφωση της εργασίας. Ίσως η πιο χρήσιμη αλλαγή ρυθμίσεων που μπορείτε να κάνετε θα είναι να οικοδομήσετε τη βελτιστοποιημένη έκδοση του κώδικα. Από προεπιλογή έχετε ρυθμίσει την εργασία σας να χτίσει την έκδοση εντοπισμού σφαλμάτων. Ας πούμε ότι θα φτιάξουμε μία βελτιστοποιημένη κατασκευή για την εργασία. Για να εξηγήσουμε στο Waf ότι θα πρέπει να βελτιστοποιηθούν οι εκδόσεις που περιλαμβάνουν τα παραδείγματα και τις δοκιμές, θα πρέπει να εκτελέσετε τις ακόλουθες εντολές

```
$ ./waf clean
$ ./waf --build-profile=optimized --enable-examples --enable-tests configure
```

Αυτό τρέχει τον Waf έξω από τον τοπικό κατάλογο (το οποίο παρέχεται ως διευκόλυνση για εσάς). Η πρώτη εντολή για να καθαρίσετε την προηγούμενη κατασκευή δεν είναι απολύτως αναγκαία, αλλά είναι καλή πρακτική (αλλά δείτε παρακάτω Προφίλ Κατασκευών), θα καταργήσει τις προηγούμενες κατασκευασμένες βιβλιοθήκες και τα αρχεία αντικειμένων που βρέθηκαν στον κατάλογο `build/`. Όταν το έργο έχει αναδιαμορφωθεί και το σύστημα κατασκευής ελέγχει για διάφορες εξαρτήσεις, θα πρέπει να δείτε κάτι που μοιάζει με το παρακάτω

```
Setting top to                           : .
Setting out to                           : build
Checking for 'gcc' (c compiler)          : /usr/bin/gcc
Checking for cc version                  : 4.2.1
Checking for 'g++' (c++ compiler)        : /usr/bin/g++
Checking boost includes                  : 1_46_1
Checking boost libs                      : ok
Checking for boost linkage               : ok
Checking for click location              : not found
Checking for program pkg-config          : /sw/bin/pkg-config
Checking for 'gtk+-2.0' >= 2.12          : yes
Checking for 'libxml-2.0' >= 2.7         : yes
Checking for type uint128_t              : not found
Checking for type __uint128_t            : yes
Checking high precision implementation   : 128-bit integer (default)
Checking for header stdint.h             : yes
Checking for header inttypes.h           : yes
Checking for header sys/inttypes.h       : not found
Checking for header sys/types.h          : yes
Checking for header sys/stat.h           : yes
Checking for header dirent.h             : yes
Checking for header stdlib.h             : yes
Checking for header signal.h             : yes
Checking for header pthread.h            : yes
Checking for header stdint.h             : yes
Checking for header inttypes.h           : yes
Checking for header sys/inttypes.h       : not found
Checking for library rt                  : not found
Checking for header netpacket/packet.h   : not found
Checking for header sys/ioctl.h          : yes
Checking for header net/if.h             : not found
Checking for header net/ethernet.h       : yes
Checking for header linux/if_tun.h       : not found
Checking for header netpacket/packet.h   : not found
Checking for NSC location                : not found
Checking for 'mpic++'                    : yes
Checking for 'sqlite3'                   : yes
Checking for header linux/if_tun.h       : not found
Checking for program sudo                : /usr/bin/sudo
Checking for program valgrind            : /sw/bin/valgrind
Checking for 'gsl'                       : yes
Checking for compilation flag -Wno-error=deprecated-d... support : ok
Checking for compilation flag -Wno-error=deprecated-d... support : ok
Checking for compilation flag -fstrict-aliasing... support       : ok
Checking for compilation flag -fstrict-aliasing... support       : ok
Checking for compilation flag -Wstrict-aliasing... support       : ok
Checking for compilation flag -Wstrict-aliasing... support       : ok
Checking for program doxygen                                     : /usr/local/bin/doxygen
---- Summary of optional NS-3 features:
Build profile                 : debug
Build directory               : build
Python Bindings               : enabled
```

```
BRITE Integration                  : not enabled (BRITE not enabled (see option --with-brite))
NS-3 Click Integration             : not enabled (nsclick not enabled (see option --with-nsclick))
GtkConfigStore                     : enabled
XmlIo                              : enabled
Threading Primitives               : enabled
Real Time Simulator                : enabled (librt is not available)
Emulated Net Device                : enabled (<netpacket/packet.h> include not detected)
File descriptor NetDevice          : enabled
Tap FdNetDevice                    : not enabled (needs linux/if_tun.h)
Emulation FdNetDevice              : not enabled (needs netpacket/packet.h)
PlanetLab FdNetDevice              : not enabled (PlanetLab operating system not detected (see option --fo
Network Simulation Cradle          : not enabled (NSC not found (see option --with-nsc))
MPI Support                        : enabled
NS-3 OpenFlow Integration          : not enabled (Required boost libraries not found, missing: system, sig
SQlite stats data output           : enabled
Tap Bridge                         : not enabled (<linux/if_tun.h> include not detected)
PyViz visualizer                   : enabled
Use sudo to set suid bit           : not enabled (option --enable-sudo not selected)
Build tests                        : enabled
Build examples                     : enabled
GNU Scientific Library (GSL)       : enabled
'configure' finished successfully (1.944s)
```

Σημειώστε το τελευταίο μέρος της παραπάνω εξόδου. Μερικές επιλογές στον *ns-3* δεν είναι ενεργοποιημένες από προεπιλογή ή απαιτούν υποστήριξη από το υποκείμενο σύστημα για να λειτουργήσει σωστά. Για παράδειγμα, για να ενεργοποιήσετε τον XmlTo, η βιβλιοθήκη LibXML-2.0 πρέπει να βρεθεί στο σύστημα. Αν δεν βρεθεί αυτή η βιβλιοθήκη, το αντίστοιχο χαρακτηριστικό του *ns-3* δεν θα έπρεπε να ενεργοποιηθεί και ένα μήνυμα θα εμφανιστεί. Σημειώστε, επίσης, ότι υπάρχει ένα χαρακτηριστικό για να χρησιμοποιήσετε το πρόγραμμα `sudo` να ρυθμίσετε το suid κομμάτι ορισμένων προγραμμάτων. Αυτό δεν είναι ενεργοποιημένο από προεπιλογή και έτσι αυτό το χαρακτηριστικό αναφέρεται ως "όχι ενεργοποιημένο."("not enabled.")

Τώρα συνεχίστε και επιστρέψετε στην κατασκευή εντοπισμού σφαλμάτων που περιλαμβάνει τα παραδείγματα και δοκιμές.

```
$ ./waf clean
$ ./waf --build-profile=debug --enable-examples --enable-tests configure
```

Το σύστημα κατασκευής είναι τώρα ρυθμισμένο και μπορείτε να χτίσετε τις debug εκδόσεις των προγραμμάτων *ns-3* απλά πληκτρολογώντας:

```
$ ./waf
```

Εντάξει, συγγνώμη, σας έκανα να φτιάξετε το μέρος του συστήματος *ns-3* δύο φορές, αλλά τώρα ξέρετε πώς να αλλάξετε τη διαμόρφωση και την κατασκευή για βελτιστοποιημένο κώδικα.

Το σενάριο build.py συζητήθηκε παραπάνω, υποστηρίζει επίσης τα `--enable-examples` και `enable-tests` ορίσματα, αλλά σε γενικές γραμμές, δεν υποστηρίζει άμεσα άλλες WAF επιλογές, για παράδειγμα, αυτό δεν θα λειτουργήσει:

```
$ ./build.py --disable-python
```

θα οδηγήσει σε

```
build.py: error: no such option: --disable-python
```

Ωστόσο, ο ειδικός φορέας `--` μπορεί να χρησιμοποιηθεί για να περάσουν επιπλέον επιλογές μέσω του WAF, έτσι ώστε αντί των ανωτέρω, τα ακόλουθα θα λειτουργήσουν:

```
$ ./build.py -- --disable-python
```

δεδομένου ότι δημιουργεί τη βασική εντολή `./waf configure --disable-python`.

Εδώ είναι λίγο περισσότερες εισαγωγικές συμβουλές για τον Waf.

### Ρύθμιση(Διαμόρφωση) εναντίον Κατασκευής

Μερικές εντολές του Waf έχουν νόημα μόνο κατά τη διάρκεια της φάσης της παραμετροποίησης και κάποιες εντολές ισχύουν κατά τη φάση της κατασκευής. Για παράδειγμα, αν θέλετε να χρησιμοποιήσετε τις λειτουργίες της εξομοίωσης του *ns-3*, ίσως πρέπει να ενεργοποιήσετε αυτήν την ρύθμιση του κομματιού suid χρησιμοποιώντας την εντολή sudo, όπως περιγράφεται παραπάνω. Αυτό αποδεικνύεται ότι είναι μια εντολή διαμόρφωσης χρόνου, και έτσι θα μπορείτε να αναμορφώσετε χρησιμοποιώντας την ακόλουθη εντολή, που περιλαμβάνει επίσης τα παραδείγματα και δοκιμές.

```
$ ./waf configure --enable-sudo --enable-examples --enable-tests
```

Αν το κάνετε αυτό, το Waf θα έχει εκκινήσει το sudo για να αλλάξει τα προγράμματα (socket creator) του κώδικα εξομοίωσης να εκτελούνται ως root.

Υπάρχουν πολλές άλλες ρυθμίσεις- και χρόνο-κατασκευής που διατίθεται σε Waf. Για να διερευνήσετε αυτές τις επιλογές, πληκτρολογήστε

```
$ ./waf --help
```

Θα χρησιμοποιήσουμε κάποιες από τις εντολές δοκιμών που σχετίζονται με την επόμενη ενότητα.

### Προφίλ Κατασκευών

Μόλις είδαμε πως μπορούμε να ρυθμίσουμε τον Waf για `debug` ή `optimized` κατασκευές

```
$ ./waf --build-profile=debug
```

Υπάρχει επίσης ένα ενδιάμεσο προφίλ κατασκευής, `release`. " -d" είναι ένα συνώνυμο για " –build-profile".

Από προεπιλογή ο Waf βάζει τα αντικείμενα κατασκευής στον κατάλογο `build`. Μπορείτε να καθορίσετε ένα διαφορετικό κατάλογο εξόδου με την επιλογή `--out`, π.χ.

```
$ ./waf configure --out=foo
```

Συνδυάζοντας αυτό με τα προφίλ κατασκευών σας επιτρέπει να πραγματοποιήσετε εναλλαγή μεταξύ των διαφόρων επιλογών μεταγλώττισης σε ένα καθαρό τρόπο

```
$ ./waf configure --build-profile=debug --out=build/debug
$ ./waf build
...
$ ./waf configure --build-profile=optimized --out=build/optimized
$ ./waf build
...
```

Αυτό σας επιτρέπει να εργάζεστε με πολλαπλές κατασκευές και όχι πάντα αντικαθιστώντας την τελευταία έκδοση. Όταν αλλάζετε, ο Waf θα μεταγλωττίζει μόνο ό,τι έχει, αντί να κάνει μεταγλώττιση πάλι.

Όταν κάνετε εναλλαγή προφίλ κατασκευής όπως αυτό, θα πρέπει να είστε προσεκτικοί για να δώσει τις ίδιες παραμέτρους διαμόρφωσης κάθε φορά. Μπορεί να είναι βολικό να καθορίσει κάποιες μεταβλητές περιβάλλοντος για να σας βοηθήσει να αποφύγετε τα λάθη

```
$ export NS3CONFIG="--enable-examples --enable-tests"
$ export NS3DEBUG="--build-profile=debug --out=build/debug"
$ export NS3OPT=="--build-profile=optimized --out=build/optimized"
```

```
$ ./waf configure $NS3CONFIG $NS3DEBUG
$ ./waf build
...
$ ./waf configure $NS3CONFIG $NS3OPT
$ ./waf build
```

### Μεταγλωττιστές

Στα παραπάνω παραδείγματα, ο Waf χρησιμοποιεί τον μεταγλωττιστή GCC C++, `g++`, για την οικοδομή του *ns-3*. Ωστόσο, είναι δυνατόν να αλλάξει τον μεταγλωττιστή του C++ που χρησιμοποιείται από τον Waf με την μεταβλητή περιβάλλοντος `CXX`. Για παράδειγμα, για να χρησιμοποιήσετε τον μεταγλωττιστή Clang C++, `clang++`,

```
$ CXX="clang++" ./waf configure
$ ./waf build
```

Κάποιος μπορεί επίσης να εγκαταστήσει τον Waf κάνοντας κατανεμημένη συλλογή με `distcc` με παρόμοιο τρόπο

```
$ CXX="distcc g++" ./waf configure
$ ./waf build
```

Περισσότερες πληροφορίες για το `distcc` και διανέμονται σύνταξη μπορείτε να βρήτε σε αυτή την σελίδα του έργου σύμφωνα με το τμήμα του οδηγού αυτού.

### Εγκατάσταση

Ο Waf μπορεί να χρησιμοποιηθεί για την εγκατάσταση βιβλιοθηκών σε διάφορα σημεία του συστήματος. Η προεπιλεγμένη θέση όπου οι βιβλιοθήκες και τα εκτελέσιμα είναι χτισμένα είναι ο κατάλογος `build`, και επειδή ο Waf γνωρίζει τη θέση των βιβλιοθηκών αυτών και τα εκτελέσιμα, δεν είναι απαραίτητο να εγκαταστήσετε τις βιβλιοθήκες και αλλού.

Εάν οι χρήστες επιλέγουν να εγκαταστήσουν πράγματα έξω από το κατάλογο κατασκευής, οι χρήστες μπορούν να εκδώσουν την εντολή `./waf install`. Από προεπιλογή, το πρόθεμα για την εγκατάσταση είναι `/usr/local`, έτσι `./waf install` θα εγκαταστήσει προγράμματα σε `/usr/local/bin`, βιβλιοθήκες σε `/usr/local/lib`, και τους τίτλους σε `/usr/local/include`. Τα προνόμια υπερχρήστη συνήθως απαιτούνται για την εγκατάσταση στο προεπιλεγμένο πρόθεμα, οπότε η τυπική εντολή θα είναι `sudo ./waf install`. Όταν τα προγράμματα που εκτελούνται με τον Waf, ο Waf πρώτα θα προτιμά να χρησιμοποιεί κοινές βιβλιοθήκες στον κατάλογο κατασκευής, μετά θα κοιτάξουμε για τις βιβλιοθήκες στη ρύθμιση διαδρομή βιβλιοθήκης στο τοπικό περιβάλλον. Έτσι, κατά την εγκατάσταση των βιβλιοθηκών για το σύστημα, είναι καλή πρακτική να ελέγξετε ότι οι προβλεπόμενες βιβλιοθήκες έχουν χρησιμοποιηθεί.

Οι χρήστες μπορούν να επιλέξουν να εγκαταστήσουν σε ένα διαφορετικό πρόθεμα με το πέρασμα την επιλογή `--prefix` σε συγκεκριμένο χρόνο, όπως:

```
./waf configure --prefix=/opt/local
```

Αν αργότερα, μετά την κατασκευή ο χρήστης εκδίδει την εντολή `./waf install`, το πρόθεμα `/opt/local` θα χρησιμοποιηθεί.

Η εντολή `./waf clean` πρέπει να χρησιμοποιείται πριν από την αναμόρφωση του έργου, εάν το Waf θα χρησιμοποιηθεί για να εγκαταστήσει πράγματα σε ένα διαφορετικό πρόθεμα.

Εν ολίγοις, δεν είναι απαραίτητο να καλέσετε `./waf install` για να χρησιμοποιήσετε τον *ns-3*. Οι περισσότεροι χρήστες δεν θα χρειαστούν αυτήν την εντολή αφού ο Waf θα πάρει τις σημερινές βιβλιοθήκες από τον κατάλογο `build`, αλλά μερικοί χρήστες μπορεί να το βρούν χρήσιμο εάν η χρήση τους περιλαμβάνει εργασία με προγράμματα εκτός από τον κατάλογο του *ns-3*.

### Ένας Waf

Υπάρχει μόνο ένα σενάριο Waf, στο ανώτατο επίπεδο του δέντρου πηγαίου κώδικα *ns-3*. Καθώς εργάζεστε, μπορείτε να βρείτε τον εαυτό σας να ξοδέψει πολύ χρόνο σε `scratch/`, ή βαθιά στο `src /...`, και να χρειαστεί να επικαλεστείτε τον Waf. Θα μπορούσατε απλά να θυμάστε πού είστε, και να επικαλέσετε τον Waf όπως αυτό

```
$ ../../../waf ...
```

αλλά αυτό είναι κουραστικό και επιρρεπή σε λάθη, και υπάρχουν καλύτερες λύσεις.

Εάν έχετε το πλήρη αποθετήριο *ns-3* αυτό το μικρό διαμάντι είναι μια αρχή

```
$ cd $(hg root) && ./waf ...
```

Ακόμα καλύτερα είναι να το ορίσετε ως συνάρτηση κέλυφος

```
$ function waff { cd $(hg root) && ./waf $* ; }

$ waff build
```

Εάν έχετε μόνο το tarball αρχείο, μια μεταβλητή περιβάλλοντος μπορεί να βοηθήσει

```
$ export NS3DIR="$PWD"
$ function waff { cd $NS3DIR && ./waf $* ; }

$ cd scratch
$ waff build
```

Θα μπορούσε να είναι δελεαστικό σε μια μονάδα καταλόγου για να προσθέσετε ένα ασήμαντο σενάριο `waf` κατά μήκος των γραμμών του `exec ../../waf`. Παρακαλώ μην το κάνετε. Είναι σύγχυση για τους νεοφερμένους, και όταν γίνει ανεπαρκώς οδηγεί σε λεπτά σφάλματα κατασκευής. Οι παραπάνω λύσεις είναι ο τρόπος να πάει καλά.

## 3.4 Κάνοντας τέστ στον *ns-3*

Μπορείτε να εκτελέσετε τη μονάδα των δοκιμών της διανομής του *ns-3* εκτελώντας το σενάριο `./test.py -c core`

```
$ ./test.py -c core
```

Οι δοκιμές αυτές έχουν παράλληλη πορεία με τον Waf. Θα πρέπει τελικά να δείτε μια έκθεση λέγοντας ότι

```
92 of 92 tests passed (92 passed, 0 failed, 0 crashed, 0 valgrind errors)
```

Αυτό είναι σημαντικό μήνυμα.

Θα δείτε επίσης την περίληψη εξόδου από τον Waf και τη δοκιμή του δρομέα εκτέλεσης κάθε δοκιμής, η οποία θα εξετάσει πραγματικά κάτι σαν

```
Waf: Entering directory `/path/to/workspace/ns-3-allinone/ns-3-dev/build'
Waf: Leaving directory `/path/to/workspace/ns-3-allinone/ns-3-dev/build'
'build' finished successfully (1.799s)

Modules built:
aodv                      applications              bridge
click                     config-store              core
csma                      csma-layout               dsdv
emu                       energy                    flow-monitor
internet                  lte                       mesh
```

```
mobility                  mpi                       netanim
network                   nix-vector-routing        ns3tcp
ns3wifi                   olsr                      openflow
point-to-point            point-to-point-layout     propagation
spectrum                  stats                     tap-bridge
template                  test                      tools
topology-read             uan                       virtual-net-device
visualizer                wifi                      wimax

PASS: TestSuite ns3-wifi-interference
PASS: TestSuite histogram

...

PASS: TestSuite object
PASS: TestSuite random-number-generators
92 of 92 tests passed (92 passed, 0 failed, 0 crashed, 0 valgrind errors)
```

Αυτή η εντολή τυπικά τρέχει από τους χρήστες για την γρήγορη επαλήθευση ότι μια διανομή *ns-3* έχει χτιστεί σωστά. (Σημειώστε τη σειρά των `PASS: ...` μπορεί να διαφέρουν, το οποίο είναι εντάξει. Αυτό που είναι σημαντικό είναι ότι η συνοπτική γραμμή στο τέλος ότι όλες οι δοκιμές πέρασαν, καμία δεν απέτυχε ή συνετρίβη.)

## 3.5 Τρέχοντας ένα Σενάριο

Εμείς συνήθως τρέχουμε σενάρια υπό τον έλεγχο του Waf. Αυτό επιτρέπει στο σύστημα κατασκευής να εξασφαλίσει ότι οι κοινές διαδρομές βιβλιοθήκης είναι σωστά ρυθμισμένες και ότι οι βιβλιοθήκες είναι διαθέσιμες κατά το χρόνο εκτέλεσης. Για να εκτελέσετε ένα πρόγραμμα, απλά χρησιμοποιήστε την επιλογή `--run` του Waf. Ας τρέξουμε τον *ns-3* αντίστοιχο του πανταχού πρόγραμματος Hello World, πληκτρολογώντας τα εξής

```
$ ./waf --run hello-simulator
```

Ο Waf ελέγχει πρώτα για να βεβαιωθεί ότι το πρόγραμμα έχει χτιστεί σωστά και εκτελεί μια συγκέντρωση, εάν απαιτείται. Ο Waf εκτελεί τότε το πρόγραμμα, το οποίο παράγει την ακόλουθη έξοδο.

```
Hello Simulator
```

Συγχαρητήρια! Τώρα είστε χρήστης του ns-3!

**Τι μπορώ να κάνω αν δεν βλέπω την έξοδο;**

Αν δείτε τα μηνύματα του Waf υποδεικνύοντας ότι η κατασκευή ολοκληρώθηκε με επιτυχία, αλλά δεν βλέπετε την έξοδο “Hello Simulator”, οι πιθανότητες είναι ότι έχετε αλλάξει την λειτουργία κατασκευής σας στο `optimized` στο Χτίζοντας με Waf τμήμα, αλλά έχετε ξεχάσει την αλλαγή πίσω στη λειτουργία `debug`. Όλοι οι έξοδοι της κονσόλας που χρησιμοποιούνται σε αυτό τον οδηγό, χρησιμοποιούν ένα ειδικό συστατικό καταγραφής *ns-3* που είναι χρήσιμο για την εκτύπωση μηνυμάτων του χρήστη στην κονσόλα. Η έξοδος από το συστατικό αυτό είναι απενεργοποιημένη αυτόματα κατά τη μεταγλώττιση του βελτιστοποιημένου κώδικα – αυτό είναι “optimized out.” Εάν δεν μπορείτε να δείτε την έξοδο “Hello Simulator», πληκτρολογήστε τα ακόλουθα

```
$ ./waf configure --build-profile=debug --enable-examples --enable-tests
```

για να πούμε στον Waf να χτίσει τις εκδόσεις debug απο τα προγράμματα του *ns-3* που περιλαμβάνει τα παραδείγματα και τις δοκιμές(τεστ). Θα πρέπει ακόμα να οικοδομήσουμε την πραγματική debug έκδοση του κώδικα πληκτρολογώντας

```
$ ./waf
```

Τώρα, αν εκτελέσετε το πρόγραμμα `hello-simulator`, θα πρέπει να δείτε την αναμενόμενη εξαγωγή.

### 3.5.1 Τα Ορίσματα του Προγράμματος

Για να τροφοδοτήσει τα ορίσματα της γραμμής εντολών για ένα πρόγραμμα *ns-3* χρησιμοποιήστε αυτό το μοτίβο

```
$ ./waf --run <ns3-program> --command-template="%s <args>"
```

Αντικαταστήστε το όνομα του προγράμματος σας για `<ns3-program>`, και τα ορίσματα για `<args>`. Το όρισμα `--command-template` στο Waf είναι βασικά μια συνταγή για την κατασκευή της πραγματικής γραμμής εντολών Waf που θα πρέπει να χρησιμοποιήσετε για να εκτελέσει το πρόγραμμα. Ο Waf ελέγχει ότι η κατασκευή έχει ολοκληρωθεί, θέτει τους κοινούς διαδρόμους(paths), μετά επικαλείται το εκτελέσιμο χρησιμοποιώντας το παρεχόμενο πρότυπο της γραμμής εντολών, εισάγοντας το όνομα του προγράμματος για την θέση `%s`. (Ομολογώ ότι αυτό είναι λίγο περίεργο, αλλά αυτός είναι ο τρόπος. Patches ευπρόσδεκτα!)

Ένα άλλο ιδιαίτερα χρήσιμο παράδειγμα είναι να εκτελέσετε μια σουίτα δοκιμής από μόνη της. Ας υποθέσουμε ότι υπάρχει ένα τέστ `mytest` σουίτα (δεν υπάρχει). Πάνω, χρησιμοποιήσαμε το σενάριο `. / test.py` για να εκτελέσει μια αρμαθιά από δοκιμές(τέστ) παράλληλα, κατ 'επανάληψη επίκληση του πραγματικού προγράμματος δοκιμών, `test-runner`. Για να επικαλεστείτε το `test-runner` άμεσα για μία μόνο δοκιμή

```
$ ./waf --run test-runner --command-template="%s --suite=mytest --verbose"
```

Αυτό περνά τα ορίσματα για το `test-runner`. Από τη στιγμή που το `mytest` δεν υπάρχει, ένα μήνυμα σφάλματος θα δημιουργηθεί. Για να εκτυπώσετε τις διαθέσιμες επιλογές `test-runner`

```
$ ./waf --run test-runner --command-template="%s --help"
```

### 3.5.2 Εντοπισμός Σφαλμάτων

Για να εκτελέσετε τα *ns-3* προγράμματα υπό τον έλεγχο μιας άλλης κοινής ωφέλειας, όπως ένα πρόγραμμα εντοπισμού σφαλμάτων (*π.χ.* `gdb`) ή ελεγκτή μνήμης (*π.χ.* `valgrind`), μπορείτε να χρησιμοποιήσετε μια παρόμοια μορφή `--command-template = "..."`.

Για παράδειγμα, για να εκτελέσετε το *ns-3* πρόγραμμα `hello-simulator` με τα ορίσματα `<args>` κάτω από τον εντοπισμό σφαλμάτων `gdb`

```
$ ./waf --run=hello-simulator --command-template="gdb %s --args <args>"
```

Παρατηρήστε ότι το όνομα του προγράμματος *ns-3* πηγαίνει με το όρισμα `--run`, και το βοηθητικό πρόγραμμα ελέγχου (εδώ `gdb`) είναι το πρώτο συμβολικό στο όρισμα `--commmand-template`. Το `--args` λέει στο `gdb` ότι το υπόλοιπο της γραμμής εντολών ανήκει στην «κατώτερο» πρόγραμμα. (Ορισμένοι `gdb` δεν καταλαβαίνουν το χαρακτηριστικό `--args`. Σε αυτήν την περίπτωση, παραλείψτε τους ορισμούς του προγράμματος από την `--command-template`, και να χρησιμοποιήσετε το `gdb` εντολή `set args`.)

Μπορούμε να συνδυάσουμε αυτή τη συνταγή και το προηγούμενο για να εκτελέσετε μια δοκιμή σύμφωνα με το πρόγραμμα εντοπισμού σφαλμάτων

```
$ ./waf --run test-runner --command-template="gdb %s --args --suite=mytest --verbose"
```

### 3.5.3 Κατάλογος Εργασίας

Ο Waf χρειάζεται να τρέχει από την τοποθεσία του στην κορυφή του δέντρου του *ns-3*. Αυτό γίνεται ο κατάλογος εργασίας όπου θα γραφτούν τα αρχεία εξόδου. Τι γίνεται όμως αν θέλετε να διατηρήσετε αυτές τις ouf στο δέντρο *ns-3* πηγαίου κώδικα; Χρησιμοποιήστε το " –cwd" επιχείρημα

```
$ ./waf --cwd=...
```

Μπορεί να είναι πιο βολικό να ξεκινήσετε με τον κατάλογο εργασίας σας όπου θέλετε τα αρχεία εξόδου, οπότε μπορεί να βοηθήσει το παρακάτω

```
$ function waff {
    CWD="$PWD"
    cd $NS3DIR >/dev/null
    ./waf --cwd="$CWD" $*
    cd - >/dev/null
  }
```

Αυτό το στολίδι της προηγούμενης έκδοσης αποθηκεύει τον τρέχοντα κατάλογο εργασίας, `cd` στον κατάλογο Waf, και μετά δίνει εντολή στο Waf να αλλάξει τον κατάλογο εργασίας *πίσω* στον αποθηκευμένο τρέχοντα κατάλογο εργασίας πριν από την εκτέλεση του προγράμματος.

ΚΕΦΑΛΑΙΟ

# ΤΕΣΣΕΡΑ

# ΕΝΝΟΙΟΛΟΓΙΚΗ ΕΠΙΣΚΟΠΗΣΗ

Το πρώτο πράγμα που χρειάζεται να κάνουμε πριν αρχίσουμε ουσιαστικά να κοιτάμε ή να γράφουμε κώδικα για τον *ns-3* είναι να εξηγήσουμε μερικές κεντρικές ιδέες και αφαιρετικές έννοιες του συστήματος. Αρκετές από αυτές μπορεί να φανούν πολύ προφανείς σε κάποιους, αλλά εμείς θα σας συνιστούσαμε να αφιερώσετε κάποιο χρόνο για το διάβασμα αυτού του μέρους, ώστε να διασφαλίσετε πως ξεκινάτε πάνω σε στέρεη βάση.

## 4.1 Αφαιρέσεις-Κλειδιά

Σε αυτό το τμήμα, θα εξετάσουμε κάποιους όρους που χρησιμοποιούνται συχνά στα δίκτυα, αλλά έχουν ένα συγκεκριμένο νόημα στον *ns-3*.

### 4.1.1 Κόμβος

Στη διαδικτυακή ορολογία, μια υπολογιστική συσκευή που συνδέεται σε ένα δίκτυο ονομάζεται *ξενιστής* (host) ή μερικές φορές και *τερματικό σύστημα* (end system). Επειδή ο *ns-3* είναι ένας προσομοιωτής *δικτύων*, και όχι ειδικά ένας προσομοιωτής του *Διαδικτύου*, σκόπιμα δεν χρησιμοποιούμε τον όρο ξενιστής, καθώς σχετίζεται άμεσα με το Διαδίκτυο και τα πρωτόκολλά του. Αντ' αυτού, χρησιμοποιούμε έναν πιο γενικό όρο που επίσης χρησιμοποιείται από άλλους προσομοιωτές και προέρχεται από τη Θεωρία Γράφων — τον όρο *κόμβος*.

Στον *ns-3* η βασική αφαίρεση της υπολογιστικής συσκευής αποκαλείται κόμβος. Αυτή η αφαίρεση αναπαριστάται στη C++ από την κλάση `Node`. Η κλάση `Node` παρέχει μεθόδους για τη διαχείριση των αναπαραστάσεων των υπολογιστικών συσκευών στις προσομοιώσεις.

Μπορείτε να σκεφτείτε ένα `Node` ως έναν υπολογιστή στον οποίο θα προσθέσετε κάποια λειτουργικότητα. Κάποιος θα μπορούσε να προσθέσει πράγματα όπως εφαρμογές, στοίβες πρωτοκόλλων και κάρτες περιφερειακών με τους σχετικούς οδηγούς τους, ώστε να επιτρέψει στον υπολογιστή να πραγματοποιήσει χρήσιμες εργασίες. Το ίδιο βασικό μοντέλο χρησιμοποιούμε κι εμείς στον *ns-3*.

### 4.1.2 Εφαρμογή

Τυπικά, το λογισμικό των υπολογιστών χωρίζεται σε δύο ευρείες κατηγορίες. Το *Λογισμικό Συστήματος* (System Software) οργανώνει τους διάφορους πόρους του υπολογιστή, όπως τη μνήμη, τους κύκλους του επεξεργαστή, τους δίσκους, το δίκτυο, κτλ., σύμφωνα με κάποιο υπολογιστικό μοντέλο. Το λογισμικό συστήματος συνήθως δε χρησιμοποιεί αυτούς τους πόρους για να ολοκληρώσει εργασίες οι οποίες ωφελούν άμεσα τον χρήστη. Ένας χρήστης τυπικά θα έτρεχε μια *εφαρμογή* που δεσμεύει και χρησιμοποιεί τους πόρους που ελέγχονται από το λογισμικό του συστήματος για να επιτύχει κάποιον στόχο.

Συχνά, η διαχωριστική γραμμή μεταξύ λογισμικού συστήματος και εφαρμογών χαράσσεται στην αλλαγή του επιπέδου δικαιωμάτων που λαμβάνει χώρα στις παγίδες (traps) του λειτουργικού συστήματος. Στον *ns-3* δεν υπάρχει ουσιαστικά η έννοια του λειτουργικού συστήματος και ειδικότερα καμία έννοια που να αφορά επίπεδα δικαιωμάτων ή κλήσεις συστήματος. Έχουμε, ωστόσο, την ιδέα της εφαρμογής. Όπως οι εφαρμογές λογισμικού

τρέχουν σε υπολογιστές ώστε να πραγματοποιήσουν εργασίες στον «πραγματικό κόσμο», οι εφαρμογές του *ns-3* τρέχουν σε `Nodes` του *ns-3* ώστε να καθοδηγήσουν τις προσομοιώσεις στον προσομοιωμένο κόσμο.

Στον *ns-3* η βασική αφαίρεση για κάποιο πρόγραμμα του χρήστη που προκαλεί κάποια δραστηριότητα, που πρέπει να προσομοιωθεί, είναι η εφαρμογή. Αυτή η αφαίρεση αναπαριστάται στη C++ από την κλάση `Application`. Η κλάση `Application` παρέχει μεθόδους για τη διαχείριση των αναπαραστάσεων της δικής μας έκδοσης των εφαρμογών επιπέδου χρήστη (user-level applications) στις προσομοιώσεις. Οι προγραμματιστές αναμένεται να εξειδικεύσουν (specialize) την κλάση `Application` στο πνεύμα του αντικειμενοστραφούς προγραμματισμού ώστε να δημιουργήσουν νέες εφαρμογές. Σε αυτόν τον οδηγό, θα χρησιμοποιήσουμε εξειδικεύσεις της κλάσης `Application`, οι οποίες καλούνται `UdpEchoClientApplication` και `UdpEchoServerApplication`. Όπως θα περιμένατε, αυτές οι εφαρμογές συνθέτουν ένα σύνολο εφαρμογών πελάτη-εξυπηρετητή που χρησιμοποιείται για να παράξει και να αναμεταδώσει προσομοιωμένα πακέτα δικτύου.

### 4.1.3 Κανάλι

Στον πραγματικό κόσμο, μπορούμε να συνδέσουμε έναν υπολογιστή σε ένα δίκτυο. Συχνά τα μέσα από τα οποία περνάνε τα δεδομένα σε αυτά τα δίκτυα ονομάζονται *κανάλια*. Όταν συνδέετε το καλώδιο Ethernet στην υποδοχή στον τοίχο, συνδέετε τον υπολογιστή σας σε ένα επικοινωνιακό κανάλι Ethernet. Στον προσομοιωμένο κόσμο του *ns-3*, κάποιος μπορεί να συνδέσει ένα `Node` σε ένα αντικείμενο που αναπαριστά ένα επικοινωνιακό κανάλι. Εδώ η βασική αφαίρεση του επικοινωνιακού υποδικτύου ονομάζεται κανάλι και αναπαριστάται στη C++ από την κλάση `Channel`.

Η κλάση `Channel` παρέχει μεθόδους για τη διαχείριση αντικειμένων του επικοινωνιακού υποδικτύου και για τη σύνδεση κόμβων σε αυτά. Η `Channel` μπορεί επίσης να εξειδικευθεί από τους προγραμματιστές, με την έννοια του αντικειμενοστραφούς προγραμματισμού. Μια εξειδίκευση της κλάσης `Channel` μπορεί να μοντελοποιεί κάτι τόσο απλό όσο ένα καλώδιο. Το εξειδικευμένο `κανάλι` μπορεί επίσης να μοντελοποιεί πράγματα τόσο περίπλοκα όσο έναν μεγάλο Ethernet μεταγωγέα (switch), ή έναν τρισδιάστατο χώρο γεμάτο με εμπόδια στην περίπτωση των ασύρματων δικτύων.

Θα χρησιμοποιήσουμε εξειδικευμένες εκδόσεις της `Channel` που καλούνται `CsmaChannel`, `PointToPointChannel` και `WifiChannel` σε αυτόν τον οδηγό. Η `CsmaChannel`, για παράδειγμα, μοντελοποιεί μια έκδοση ενός επικοινωνιακού υποδικτύου που υλοποιεί ένα *carrier sense multiple access* (CSMA) επικοινωνιακό μέσο. Αυτό μας παρέχει λειτουργικότητα όμοια με του Ethernet.

### 4.1.4 Δικτυακή Συσκευή

Συνήθως ήταν δεδομένο πως εάν ήθελες να συνδέσεις έναν υπολογιστή σε ένα δίκτυο, έπρεπε να αγοράσεις ένα συγκεκριμένο είδος δικτυακού καλωδίου και εξοπλισμό που ονομαζόταν (σύμφωνα με την ορολογία των Η/Υ) *περιφερειακή κάρτα*, η οποία έπρεπε να εγκατασταθεί στον υπολογιστή. Αν η περιφερειακή κάρτα υλοποιούσε κάποια δικτυακή λειτουργία, ονομαζόταν Κάρτα Διεπαφής Δικτύου (Network Interface Card) ή *NIC*. Σήμερα οι περισσότεροι υπολογιστές έχουν ενσωματωμένο εξοπλισμό δικτυακής διεπαφής και οι χρήστες δεν βλέπουν τα συστατικά του μέρη.

Μια κάρτα NIC δε θα λειτουργήσει χωρίς τον οδηγό λογισμικού που θα χειρίζεται το υλικό. Στο Unix (ή Linux), ένα μέρος του περιφερειακού υλικού κατηγοριοποιείται ως *συσκευή*. Οι συσκευές ελέγχονται μέσω *οδηγών συσκευών*, και οι δικτυακές συσκευές (NICs) ελέγχονται μέσω *οδηγών δικτυακών συσκευών*, συλλογικά γνωστές ως *συσκευές δικτύου*. Στο Unix και στο Linux γίνεται αναφορά σε αυτές τις συσκευές δικτύου με ονόματα όπως *eth0*.

Στον *ns-3* η αφαίρεση της *συσκευής δικτύου* καλύπτει τόσο τον οδηγό λογισμικού όσο και το προσομοιωμένο υλικό. Μια συσκευή δικτύου «εγκαθίσταται» σε ένα `Node` προκειμένου να επιτρέψει στο `Node` να επικοινωνεί με άλλα `Nodes` στην προσομοίωση μέσω `Channels`. Όπως και σε έναν αληθινό υπολογιστή, ένα `Node` μπορεί να είναι συνδεδεμένο σε περισσότερα από ένα `Channels` μέσω πολλαπλών `NetDevices`.

Η αφαίρεση της συσκευής δικτύου αναπαρίσταται στη C++ από την κλάση `NetDevice`. Η κλάση `NetDevice` παρέχει μεθόδους για τη διαχείριση συνδέσεων σε αντικείμενα `Node` και `Channel` και μπορεί να εξειδικευ-

θεί από τους προγραμματιστές σύμφωνα με την έννοια του αντικειμενοστραφούς προγραμματισμού. Εμείς θα χρησιμοποιήσουμε τις διάφορες εξειδικευμένες εκδόσεις της `NetDevice` που καλούνται `CsmaNetDevice`, `PointToPointNetDevice`, και `WifiNetDevice` σε αυτόν τον οδηγό. Όπως μια Ethernet NIC είναι σχεδιασμένη ώστε να λειτουργεί μαζί με ένα δίκτυο Ethernet, η `CsmaNetDevice` έχει σχεδιαστεί ώστε να δουλεύει με ένα `CsmaChannel`, η `PointToPointNetDevice` ώστε να λειτουργεί με ένα `PointToPointChannel` και η `WifiNetNevice` ώστε να λειτουργεί με ένα `WifiChannel`.

### 4.1.5 Βοηθοί Τοπολογίας

Σε ένα πραγματικό δίκτυο, θα βρείτε υπολογιστές-ξενιστές (host) με επιπρόσθετες (ή ενσωματωμένες) NIC. Στον *ns-3* θα λέγαμε ότι θα βρείτε `Nodes` με συνδεδεμένες `NetDevices`. Σε ένα μεγάλο προσομοιωμένο δίκτυο θα χρειαστεί να καθορίσετε πολλές συνδέσεις ανάμεσα σε `Nodes`, `NetDevices` και `Channels`.

Καθώς η σύνδεση `NetDevices` σε `Nodes`, η σύνδεση `NetDevices` σε `Channels`, η ανάθεση IP διευθύνσεων, κτλ., είναι τόσο κοινές ενέργειες στον *ns-3*, εμείς παρέχουμε κάτι το οποίο ονομάζουμε *βοηθούς τοπολογίας*, ώστε να το κάνουμε αυτό όσο το δυνατόν πιο εύκολο. Για παράδειγμα, μπορεί να χρειαστούν πολλές ξεχωριστές κεντρικές εργασίες του *ns-3* για τη δημιουργία μιας `NetDevice`, την προσθήκη μιας διεύθυνσης MAC, την εγκατάσταση αυτής της συσκευής δικτύου σε ένα `Node`, τη ρύθμιση της στοίβας πρωτοκόλλου του κόμβου, και έπειτα τη σύνδεση της `NetDevice` σε ένα `Channel`. Ακόμα περισσότερες εργασίες θα απαιτούνταν για τη σύνδεση πολλαπλών συσκευών πάνω σε κανάλια πολλών κατευθύνσεων και έπειτα η σύνδεση των ανεξάρτητων δικτύων σε διαδίκτυα. Εμείς παρέχουμε τα αντικείμενα των βοηθών τοπολογίας, που συνδυάζουν αυτές τις πολλές ξεχωριστές διαδικασίες σε ένα εύχρηστο μοντέλο για τη δική σας ευκολία.

## 4.2 Ένα Πρώτο Σενάριο στον ns-3

Αν έχετε κατεβάσει το σύστημα όπως σας προτάθηκε παραπάνω, θα έχετε μια έκδοση του *ns-3* σε έναν κατάλογο που ονομάζεται `repos` μέσα στον home κατάλογό σας. Μεταβείτε στον κατάλογο αυτής της έκδοσης, και θα πρέπει να βρείτε μια δομή καταλόγου παρόμοια με την ακόλουθη:

```
AUTHORS       examples       scratch        utils      waf.bat*
bindings      LICENSE        src            utils.py   waf-tools
build         ns3            test.py*       utils.pyc  wscript
CHANGES.html  README         testpy-output  VERSION    wutils.py
doc           RELEASE_NOTES  testpy.supp    waf*       wutils.pyc
```

Μεταβείτε στον κατάλογο `examples/tutorial`. Θα πρέπει να δείτε ένα αρχείο με όνομα `first.cc` που βρίσκεται εκεί. Αυτό είναι ένα σενάριο που θα δημιουργήσει έναν απλό από-σημείο-σε-σημείο (point-to-point) σύνδεσμο ανάμεσα σε δύο κόμβους και θα μεταδώσει ένα μόνο πακέτο ανάμεσα στους κόμβους. Ας ρίξουμε μια ματιά στο σενάριο αυτό γραμμή προς γραμμή, οπότε προχωρήστε και ανοίξτε το `first.cc` στον αγαπημένο σας κειμενογράφο.

### 4.2.1 Στερεότυπο

Η πρώτη γραμμή στο αρχείο είναι μια γραμμή κατάστασης για τον emacs. Αυτή λέει στον emacs τις συμβάσεις διαμόρφωσης (στυλ κώδικα) που χρησιμοποιούμε στον πηγαίο μας κώδικα.

```
/* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
```

Το θέμα αυτό είναι πάντα κάπως αμφιλεγόμενο, οπότε ας το θέσουμε εκτός συζήτησης αμέσως. Το project του *ns-3*, όπως και τα περισσότερα μεγάλα project, έχει υιοθετήσει ένα στυλ κώδικα το οποίο πρέπει να ακολουθείται σε όλο τον κώδικα που προέρχεται από συνεισφορά. Εάν θέλετε να συνεισφέρετε τον κώδικά σας στο project, θα πρέπει εκ των πραγμάτων να προσαρμοστείτε στο πρότυπο κώδικα του *ns-3*, όπως περιγράφεται στο αρχείο `doc/codingstd.txt` ή όπως φαίνεται στην ιστοσελίδα του project εδώ.

Θα σας συνιστούσαμε, λοιπόν, να συνηθίσετε την όψη και την αίσθηση του κώδικα του *ns-3* και να υιοθετήσετε το πρότυπο αυτό όποτε δουλεύετε με τον κώδικά μας. Όλοι στην ομάδα προγραμματιστών και όσοι συνεισφέρουν το έχουν κάνει αυτό με ποικίλες δόσεις γκρίνιας. Η γραμμή δήλωσης κατάστασης (mode) του emacs παραπάνω καθιστά πιο εύκολη τη σωστή διαμόρφωση, εάν χρησιμοποιείτε τον επεξεργαστή emacs.

Ο προσομοιωτής *ns-3* υπάγεται στην Γενική Άδεια Δημόσιας Χρήσης GNU GPL. Θα δείτε την κατάλληλη νομική ορολογία του GNU στην κορυφή κάθε αρχείου στη διανομή του *ns-3*. Συχνά θα δείτε και μια ειδοποίηση πνευματικής ιδιοκτησίας για ένα από τα ινστιτούτα που συμμετέχουν στο project του *ns-3* πάνω από το κείμενο της GPL και έναν συγγραφέα από κάτω.

```
/*
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License version 2 as
 * published by the Free Software Foundation;
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307 USA
 */
```

### 4.2.2 Συμπερίληψη Ενοτήτων (Module Includes)

Ο κώδικας ξεκινάει κανονικά με έναν αριθμό από δηλώσεις συμπερίληψης (include).

```
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/internet-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/applications-module.h"
```

Προκειμένου να βοηθήσουμε τους χρήστες σεναρίων υψηλού-επιπέδου να διαχειριστούν το μεγάλο αριθμό από αρχεία include που βρίσκονται στο σύστημα, ομαδοποιούμε τα include σε σχετικά μεγάλες ενότητες. Παρέχουμε ένα και μόνο αρχείο include το οποίο θα φορτώνει αναδρομικά όλα τα αρχεία include που χρησιμοποιούνται σε κάθε ενότητα. Αντί να αναγκαστείτε να ψάξετε για το τι επικεφαλίδα (header) χρειάζεστε, και πιθανόν να πρέπει να ρυθμίσετε σωστά έναν αριθμό εξαρτήσεων, σας δίνουμε τη δυνατότητα να φορτώσετε ένα σύνολο από αρχεία με υψηλό βαθμό λεπτομέρειας. Δεν είναι και η πιο αποτελεσματική προσέγγιση αλλά σίγουρα κάνει τη συγγραφή σεναρίων πολύ πιο εύκολη.

Κάθε ένα από τα αρχεία include του *ns-3* βρίσκεται σε έναν κατάλογο που ονομάζεται `ns3` (μέσα στον κατάλογο build) κατά τη διάρκεια της διαδικασίας του build, ώστε να αποφευχθούν συγκρούσεις ονομάτων μεταξύ αρχείων include. Το αρχείο `ns3/core-module.h` αντιστοιχεί στην ενότητα του ns-3 που θα βρείτε στον κατάλογο `src/core` στην κατεβασμένη έκδοσή σας. Αν δείτε τη λίστα των αρχείων σε αυτόν τον κατάλογο θα συναντήσετε έναν μεγάλο αριθμό από αρχεία επικεφαλίδας. Όταν πραγματοποιήσετε το build, το Waf θα τοποθετήσει τα καθολικά αρχεία επικεφαλίδας σε έναν κατάλογο με όνομα `ns3` μέσα στον κατάλληλο κατάλογο `build/debug` ή `build/optimized`, σύμφωνα με τις ρυθμίσεις σας. Το Waf θα δημιουργήσει επίσης αυτόματα ένα αρχείο συμπερίληψης ως ενότητα (module include file) για τη φόρτωση όλων των καθολικών αρχείων επικεφαλίδας.

Φυσικά, από τη στιγμή που ακολουθείτε αυτόν τον οδηγό με θρησκευτικη ευλάβεια, θα έχετε ήδη εκτελέσει την εντολή

```
$ ./waf -d debug --enable-examples --enable-tests configure
```

προκειμένου να ρυθμίσετε το project ώστε να πραγματοποιήσει τις κατασκευές αποσφαλμάτωσης (debug builds) που περιέχουν παραδείγματα και τεστ. Θα έχετε εκτελέσει επίσης και την εντολή

```
$ ./waf
```

ώστε να κατασκευάσετε (build) το project. Οπότε τώρα εάν κοιτάξετε στον κατάλογο `../../build/debug/ns3` θα βρείτε τα τέσσερα αρχεία συμπερίληψης ως ενότητες που παρουσιάστηκαν παραπάνω. Μπορείτε να ρίξετε μια ματιά στα περιεχόμενα αυτών των αρχείων και να ανακαλύψετε ότι όντως περιλαμβάνουν όλα τα αρχεία καθολικής συμπερίληψης στις αντίστοιχες ενότητες.

### 4.2.3 Χώρος Ονομάτων του ns-3

Η επόμενη γραμμή στο σενάριο `first.cc` είναι μια δήλωση χώρου ονομάτων.

```
using namespace ns3;
```

Το project του *ns-3* είναι υλοποιημένο σε έναν χώρο ονομάτων της C++ που ονομάζεται `ns3`. Αυτός ομαδοποιεί όλες τις δηλώσεις που σχετίζονται με τον *ns-3* σε έκταση εκτός του καθολικού χώρου ονομάτων, κάτι το οποίο ελπίζουμε ότι θα βοηθήσει στην ενοποίηση με άλλον κώδικα. Η δήλωση `using` της C++ εισάγει τον χώρο ονομάτων του *ns-3* μέσα στην τρέχουσα (καθολική) περιοχή δηλώσεων. Είναι ένας κομψός τρόπος για να πούμε ότι μετά από αυτή τη δήλωση, δε θα χρειαστεί να πληκτρολογήσετε τον τελεστή ανάλυσης εμβέλειας `ns3::` (scope resolution operator) πριν από κάθε γραμμή κώδικα του *ns-3* προκειμένου να τον χρησιμοποιήσετε. Αν δεν είστε σχετικοί με τους χώρους ονομάτων, παρακαλούμε συμβουλευτείτε σχεδόν οποιονδήποτε οδηγό της C++ και συγκρίνετε τον χώρο ονομάτων `ns3` και τη χρήση του εδώ με περιπτώσεις του χώρου ονομάτων `std` και τις δηλώσεις `using namespace std;`, που θα συναντήσετε συχνά σε συζητήσεις που αφορούν την `cout` και τις ροές δεδομένων (streams).

### 4.2.4 Καταγραφή (Logging)

Η επόμενη γραμμή του σεναρίου είναι η ακόλουθη,

```
NS_LOG_COMPONENT_DEFINE ("FirstScriptExample");
```

Θα αξιοποιήσουμε την προκειμένη δήλωση ως μια βολική ευκαιρία για να μιλήσουμε σχετικά με το σύστημα τεκμηρίωσής μας Doxygen. Αν κοιτάξετε στον ιστότοπο του project, ns-3 project, θα βρείτε έναν σύνδεσμο προς την τεκμηρίωση ("Documentation") στην μπάρα πλοήγησης. Εάν επιλέξετε αυτόν τον σύνδεσμο, θα οδηγηθείτε στη σελίδα τεκμηρίωσής μας. Υπάρχει ένας σύνδεσμος προς την τελευταία έκδοση ("Latest Release") ο οποίος θα σας οδηγήσει στην τεκμηρίωση για την τελευταία σταθερή έκδοση του *ns-3*. Εάν επιλέξετε τον σύνδεσμο "API Documentation", θα οδηγηθείτε στη σελίδα τεκμηρίωσης του API του *ns-3*.

Κατά μήκος της αριστερής πλευράς, θα βρείτε μια γραφική αναπαράσταση της δομής της τεκμηρίωσης. Ένα καλό μέρος για να ξεκινήσετε είναι το «βιβλίο» `NS-3 Modules` (Ενότητες του NS-3) στο δέντρο πλοήγησης του *ns-3*. Αν πατήσετε και επεκτείνετε τα `Modules` θα δείτε μια λίστα με τεκμηριώσεις ενοτήτων του *ns-3*. Η ιδέα της ενότητας εδώ συνδέεται απευθείας με τα αρχεία συμπερίληψης ως ενότητες που αναφέραμε παραπάνω. Το υποσύστημα καταγραφής του *ns-3* αναφέρεται στον τομέα `C++ Constructs Used by All Modules`, οπότε προχωρήστε και επεκτείνετε αυτόν τον κόμβο τεκμηρίωσης. Τώρα, επεκτείνετε το βιβλίο `Debugging` και έπειτα επιλέξτε τη σελίδα `Logging`.

Σε αυτό το σημείο θα πρέπει να βλέπετε την τεκμηρίωση Doxygen για την ενότητα της καταγραφής. Στη λίστα των `#define` στο πάνω μέρος της σελίδας θα δείτε την καταχώρηση για το `NS_LOG_COMPONENT_DEFINE`. Προτού μεταβείτε εκεί, θα ήταν μάλλον καλό να δείτε την λεπτομερή περιγραφή ("Detailed Description") της ενότητας καταγραφής για να αποκτήσετε μια αίσθηση της όλης λειτουργίας. Μπορείτε είτε να κατεβείτε προς τα κάτω, είτε να επιλέξετε το σύνδεσμο "More..." κάτω από το συνεργατικό διάγραμμα για να το κάνετε αυτό.

Μόλις έχετε αποκτήσει μια γενική ιδέα του τι συμβαίνει, συνεχίστε και ρίξτε μια ματιά στη συγκεκριμένη τεκμηρίωση του `NS_LOG_COMPONENT_DEFINE`. Δε θα γράψουμε για δεύτερη φορά την τεκμηρίωση εδώ, αλλά

για να συνοψίσουμε, αυτή η γραμμή δηλώνει ένα στοιχείο καταγραφής (logging component) που ονομάζεται `FirstScriptExample` και σας επιτρέπει να ενεργοποιήσετε και να απενεργοποιήσετε την καταγραφή μηνυμάτων κονσόλας μέσω αναφοράς στο όνομα.

### 4.2.5 Η Μέθοδος Main

Οι επόμενες γραμμές του σεναρίου που θα συναντήσετε είναι,

```
int
main (int argc, char *argv[])
{
```

Αυτή είναι απλά η δήλωση της main μεθόδου του προγράμματός (σεναρίου) σας. Όπως και σε οποιοδήποτε C++ πρόγραμμα, θα χρειαστεί να ορίσετε μια main μέθοδο η οποία θα είναι η πρώτη μέθοδος που θα εκτελεστεί. Δεν υπάρχει τίποτε το ιδιαίτερο εδώ. Το *ns-3* σενάριό σας είναι απλά ένα C++ πρόγραμμα.

Η επόμενη γραμμή θέτει την ακρίβεια της ώρας σε ένα νανοδευτερόλεπτο (1 ns), το οποίο τυχαίνει να είναι και η προκαθορισμένη τιμή:

```
Time::SetResolution (Time::NS);
```

Η ακρίβεια είναι η μικρότερη τιμή του χρόνου που μπορεί να αναπαρασταθεί (καθώς επίσης και η μικρότερη αναπαραστήσιμη διαφορά μεταξύ δύο τιμών χρόνου). Μπορείτε να αλλάξετε την ακρίβεια μόνο μία φορά. Ο μηχανισμός που μας επιτρέπει αυτή την ευελιξία απαιτεί κάπως πολύ μνήμη, οπότε μόλις η ακρίβεια έχει καθοριστεί ρητά, απελευθερώνουμε τη μνήμη, εμποδίζοντας περαιτέρω ανανεώσεις. (Εάν δεν θέσετε ρητά την ακρίβεια, θα τεθεί εξ ορισμού στο ένα νανοδευτερόλεπτο, και η μνήμη θα απελευθερωθεί όταν ξεκινήσει η προσομοίωση.)

Οι δύο επόμενες γραμμές του σεναρίου χρησιμοποιούνται για την ενεργοποίηση δύο στοιχείων καταγραφής που είναι ενσωματωμένα στις εφαρμογές του echo πελάτη και του echo εξυπηρετητή:

```
LogComponentEnable("UdpEchoClientApplication", LOG_LEVEL_INFO);
LogComponentEnable("UdpEchoServerApplication", LOG_LEVEL_INFO);
```

Αν έχετε διαβάσει την τεκμηρίωση για τα στοιχεία της καταγραφής, θα έχετε δει ότι υπάρχει ένας αριθμός επιπέδων λεπτομέρειας κατά την καταγραφή, που μπορείτε να ενεργοποιήσετε σε κάθε τέτοιο στοιχείο. Αυτές οι δύο γραμμές κώδικα επιτρέπουν την καταγραφή αποσφαλμάτωσης στο επίπεδο INFO για τους πελάτες και εξυπηρετητές echo. Αυτό θα έχει σαν αποτέλεσμα το να εκτυπώνει η εφαρμογή μηνύματα καθώς στέλνονται και λαμβάνονται τα πακέτα κατά τη διάρκεια της προσομοίωσης.

Τώρα θα πάμε κατευθείαν στη διαδικασία δημιουργίας μιας τοπολογίας και την εκτέλεση μιας προσομοίωσης. Θα χρησιμοποιήσουμε τα αντικείμενα βοηθών τοπολογίας για να κάνουμε αυτή την εργασία όσο πιο εύκολη γίνεται.

### 4.2.6 Βοηθοί Τοπολογίας

#### NodeContainer

Οι επόμενες δύο γραμμές κώδικα στο σενάριό μας θα δημιουργήσουν στην ουσία τα αντικείμενα `Node` του *ns-3*, τα οποία θα αναπαριστούν τους υπολογιστές στην προσομοίωση.

```
NodeContainer nodes;
nodes.Create (2);
```

Ας βρούμε την τεκμηρίωση για την κλάση `NodeContainer` πριν συνεχίσουμε. Ένας άλλος τρόπος για να φτάσουμε στην τεκμηρίωση για μια δεδομένη κλάση είναι μέσω της καρτέλας `Classes` στις σελίδες του Doxygen. Εάν έχετε ακόμα πρόχειρο το Doxygen, απλά ανεβείτε στο πάνω μέρος της σελίδας και επιλέξτε την καρτέλα `Classes`. Θα πρέπει να δείτε να εμφανίζεται ένα νέο σύνολο από καρτέλες, μία από τις οποίες θα είναι και η

`Class List`. Κάτω από αυτήν την καρτέλα θα δείτε μία λίστα με όλες τις κλάσεις του *ns-3*. Κατεβείτε προς τα κάτω, ψάχνοντας για την `ns3::NodeContainer`. Μόλις βρείτε την κλάση, προχωρήστε και επιλέξτε την ώστε να πάτε στην τεκμηρίωση της κλάσης αυτής.

Μπορεί να θυμάστε ότι μία από τις αφαιρέσεις-κλειδιά μας είναι ο `κόμβος`. Αυτός αντιπροσωπεύει έναν υπολογιστή στον οποίο εμείς πρόκειται να προσθέσουμε πράγματα, όπως στοίβες πρωτοκόλλου, εφαρμογές και περιφερειακές κάρτες. Ο βοηθός τοπολογίας `NodeContainer` παρέχει έναν βολικό τρόπο για τη δημιουργία, τη διαχείριση και την πρόσβαση σε οποιοδήποτε `Node` αντικείμενο δημιουργούμε, ώστε να εκτελέσουμε την προσομοίωση. Η πρώτη γραμμή παραπάνω απλά δηλώνει ένα NodeContainer, τον οποίο αποκαλούμε `nodes`. Η δεύτερη γραμμή καλεί τη μέθοδο `Create` πάνω στο αντικείμενο `nodes` και ζητάει από τον container να δημιουργήσει δύο κόμβους. Όπως περιγράφθηκε και στο Doxygen, ο container καλεί το κατάλληλο σύστημα του *ns-3* για να δημιουργήσει δύο αντικείμενα `Node` και αποθηκεύει τους δείκτες προς αυτά τα αντικείμενα εσωτερικά.

Οι κόμβοι δεν κάνουν τίποτα έτσι όπως είναι στο σενάριο. Το επόμενο βήμα για την κατασκευή μιας τοπολογίας είναι να συνδέσουμε τους δύο κόμβους μεταξύ τους σε ένα δίκτυο. Η απλούστερη μορφή δικτύου που υποστηρίζουμε είναι ένας απλός σύνδεσμος σημείου-προς-σημείο ανάμεσα σε δύο κόμβους. Θα κατασκευάσουμε εδώ έναν από αυτούς τους συνδέσμους.

### PointToPointHelper

Κατασκευάζουμε ένα σύνδεσμο σημείου-προς-σημείο, και, με ένα πρότυπο που θα σας γίνει αρκετά οικείο, χρησιμοποιούμε ένα αντικείμενο βοηθού τοπολογίας ώστε να κάνουμε τη δουλειά χαμηλού επιπέδου που απαιτείται για να στήσουμε το σύνδεσμο. Θυμηθείτε ότι δύο από τις κύριες αφαιρέσεις μας είναι η `NetDevice` και το `Channel`. Στον πραγματικό κόσμο, αυτοί οι όροι αντιστοιχούν περίπου στις περιφερειακές κάρτες και στα καλώδια δικτύου. Τυπικά αυτά τα δύο πράγματα είναι στενά συνδεδεμένα μεταξύ τους και δεν θα πρέπει να περιμένετε ότι συλλειτουργούν π.χ. συσκευές Ethernet και ασύρματα κανάλια. Οι βοηθοί τοπολογίας μας ακολουθούν αυτό το αυστηρό ταίριασμα και κατά συνέπεια θα χρησιμοποιήσετε έναν `PointToPointHelper` για να ρυθμίσετε και να συνδέσετε τα αντικείμενα του *ns-3* `PointToPointNetDevice` και `PointToPointChannel` σε αυτό το σενάριο.

Οι επόμενες τρεις γραμμές στο σενάριο είναι,

```
PointToPointHelper pointToPoint;
pointToPoint.SetDeviceAttribute ("DataRate", StringValue ("5Mbps"));
pointToPoint.SetChannelAttribute ("Delay", StringValue ("2ms"));
```

Η πρώτη γραμμή,

```
PointToPointHelper pointToPoint;
```

δημιουργεί ένα αντικείμενο `PointToPointHelper` στη στοίβα. Από την οπτική υψηλού επιπέδου η επόμενη γραμμή,

```
pointToPoint.SetDeviceAttribute ("DataRate", StringValue ("5Mbps"));
```

λέει στο αντικείμενο `PointToPointHelper` να χρησιμοποιήσει την τιμή "5Mbps" (πέντε μεγα-μπιτ ανά δευτερόλεπτο) ως "DataRate" (ρυθμό δεδομένων) όταν δημιουργήσει ένα αντικείμενο `PointToPointNetDevice`.

Από μια πιο λεπτομερή οπτική, η ακολουθία χαρακτήρων "DataRate" αντιστοιχεί σε αυτό που αποκαλούμε `Attribute` (χαρακτηριστικό) της `PointToPointNetDevice`. Εάν κοιτάξετε στο Doxygen την κλάση `ns3::PointToPointNetDevice` και βρείτε την τεκμηρίωση για τη μέθοδο `GetTypeId`, θα δείτε μια λίστα από χαρακτηριστικά που ορίζονται για αυτή τη συσκευή. Ανάμεσα σε αυτά είναι και το "DataRate" `Attribute`. Τα περισσότερα αντικείμενα του *ns-3*, που είναι ορατά από τους χρήστες, έχουν παρόμοιες λίστες από χαρακτηριστικά. Χρησιμοποιούμε αυτόν το μηχανισμό για να ρυθμίσουμε εύκολα τις προσομοιώσεις χωρίς να επαναμεταγλωττίζουμε, όπως θα δείτε στο ακόλουθο τμήμα.

Αντίστοιχα με το "DataRate", στην `PointToPointNetDevice` θα βρείτε ένα "Delay" (καθυστέρηση) `Attribute` συσχετισμένο με το `PointToPointChannel`. Η τελευταία γραμμή,

```
pointToPoint.SetChannelAttribute ("Delay", StringValue ("2ms"));
```

λέει στον PointToPointHelper να χρησιμοποιήσει την τιμή “2ms” (δύο χιλιοστά του δευτερολέπτου) ως την τιμή της καθυστέρησης μετάδοσης κάθε καναλιού σημείου-προς-σημείο που θα δημιουργήσει στη συνέχεια.

### NetDeviceContainer

Σε αυτό το σημείο του σεναρίου, έχουμε ένα NodeContainer που περιέχει δύο κόμβους. Έχουμε έναν PointToPointHelper που έχει καθοριστεί και είναι έτοιμο να φτιάξει PointToPointNetDevices και να τις ενώσει με αντικείμενα της PointToPointChannel μεταξύ τους. Με τον ίδιο τρόπο που χρησιμοποιήσαμε το αντικείμενο βοηθού τοπολογίας NodeContainer για να δημιουργήσουμε τους κόμβους της προσομοίωσής μας, θα ζητήσουμε από τον PointToPointHelper να κάνει τη δουλειά που περιλαμβάνει τη δημιουργία, τη ρύθμιση και την εγκατάσταση των συσκευών για εμάς. Θα χρειαστεί να έχουμε μια λίστα με όλα τα αντικείμενα τύπου NetDevice που θα δημιουργηθούν, οπότε χρησιμοποιούμε ένα NetDeviceContainer για να τα αποθηκεύσουμε, όπως χρησιμοποιήσαμε και ένα NodeContainer για να κρατήσουμε τους κόμβους που δημιουργήσαμε. Οι δύο ακόλουθες γραμμές κώδικα,

```
NetDeviceContainer devices;
devices = pointToPoint.Install (nodes);
```

θα τελειώσουν τη ρύθμιση των συσκευών και του καναλιού. Η πρώτη γραμμή δηλώνει τον container συσκευών που αναφέρθηκε παραπάνω, και η δεύτερη κάνει τη βαριά δουλειά. Η μέθοδος Install του PointToPointHelper δέχεται ένα NodeContainer ως παράμετρο. Εσωτερικά, δημιουργείται ένας NetDeviceContainer. Για κάθε κόμβο του NodeContainer (πρέπει να είναι ακριβώς δύο για έναν σύνδεσμο σημείου-προς-σημείο) μια PointToPointNetDevice δημιουργείται και αποθηκεύεται στον container συσκευών. Ένα PointToPointChannel δημιουργείται και οι δύο PointToPointNetDevices συνδέονται σε αυτό. Όταν δημιουργούνται αντικείμενα από τον PointToPointHelper, τα Attributes που έχουν τεθεί νωρίτερα στον βοηθό χρησιμοποιούνται για να αρχικοποιήσουν τα αντίστοιχα Attributes στα αντικείμενα που δημιουργήθηκαν.

Μετά την εκτέλεση της κλήσης pointToPoint.Install (nodes), θα έχουμε δύο κόμβους, σε κάθε έναν από αυτούς μια εγκατεστημένη δικτυακή συσκευή σημείου-προς-σημείο και ένα μονό κανάλι σημείου-προς-σημείο ανάμεσά τους. Και οι δύο συσκευές θα ρυθμιστούν ώστε να μεταδίδουν δεδομένα με ρυθμό 5 megabit ανά δευτερόλεπτο πάνω από το κανάλι, που έχει καθυστέρηση μετάδοσης δύο χιλιοστά του δευτερολέπτου.

### InternetStackHelper

Τώρα έχουμε ρυθμισμένους κόμβους και συσκευές, αλλά δεν έχουμε καμία στοίβα πρωτοκόλλου εγκατεστημένη στους κόμβους. Οι επόμενες δύο γραμμές κώδικα θα το φροντίσουν αυτό.

```
InternetStackHelper stack;
stack.Install (nodes);
```

Ο InternetStackHelper είναι ένας βοηθός τοπολογίας, που είναι για τις διαδικτυακες στοίβες ότι είναι ο PointToPointHelper για τις δικτυακές συσκευές σημείου-προς-σημείο. Η μέθοδος Install δέχεται ένα NodeContainer ως παράμετρο. Όταν εκτελεστεί, θα εγκαταστήσει μια Διαδικτυακή Στοίβα (TCP, UDP, IP, κτλ.) σε κάθε έναν από τους κόμβους μέσα στον container κόμβων.

### Ipv4AddressHelper

Έπειτα χρειάζεται να συσχετίσουμε τις συσκευές στους κόμβους μας με διευθύνσεις IP. Εμείς παρέχουμε έναν βοηθό τοπολογίας που διαχειρίζεται το διαμοιρασμό των διευθύνσεων IP. Το μόνο API που είναι ορατό στους χρήστες είναι ο καθορισμός της βασικής διεύθυνσης IP (base IP address) και της μάσκας δικτύου (network mask),

που θα χρησιμοποιηθούν όταν πραγματοποιηθεί ο ουσιαστικός διαμοιρασμός διευθύνσεων (ο οποίος γίνεται σε ένα κατώτερο επίπεδο μέσα στο βοηθό).

Οι επόμενες δύο γραμμές κώδικα στο σενάριο που έχουμε ως παράδειγμα, το `first.cc`,

```
Ipv4AddressHelper address;
address.SetBase ("10.1.1.0", "255.255.255.0");
```

δηλώνουν ένα αντικείμενο βοηθού διευθύνσεων, και του λένε ότι πρέπει να ξεκινήσει την διανομή IP διευθύνσεων από το δίκτυο 10.1.1.0 χρησιμοποιώντας τη μάσκα 255.255.255.0 για να καθορίσουν τα bit που θα διανεμηθούν. Εξ ορισμού οι διευθύνσεις που δεσμεύονται θα αρχίσουν από το ένα και θα αυξάνονται μονότονα, οπότε η πρώτη διεύθυνση που θα δοθεί από τη βάση θα είναι η 10.1.1.1, ακολουθούμενη από την 10.1.1.2, κτλ. Το σύστημα χαμηλού επιπέδου του *ns-3* θυμάται στην ουσία όλες τις διευθύνσεις IP που έχουν κατανεμηθεί και θα παράξει ένα fatal error (μοιραίο λάθος) εάν προκαλέσετε κατά λάθος τη δημιουργία της ίδιας διεύθυνσης δύο φορές (σφάλμα που είναι πολύ δύσκολο προς αποσφαλμάτωση, παρεμπιπτόντως).

Η επόμενη γραμμή κώδικα,

```
Ipv4InterfaceContainer interfaces = address.Assign (devices);
```

πραγματοποιεί την ουσιαστική ανάθεση διευθύνσεων. Στον *ns-3* κάνουμε τη συσχέτιση μεταξύ μιας IP διεύθυνσης και μιας συσκευής χρησιμοποιώντας ένα αντικείμενο `Ipv4Interface`. Όπως κάποιες φορές χρειαζόμαστε μια λίστα δικτυακών συσκευών που έχουν δημιουργηθεί από έναν βοηθό για μελλοντική αναφορά, έτσι κάποιες φορές χρειαζόμαστε και μια λίστα από αντικείμενα `Ipv4Interface`. Ο `Ipv4InterfaceContainer` παρέχει αυτή τη λειτουργία.

Τώρα έχουμε φτιάξει ένα δίκτυο σημείου-προς-σημείο, με εγκατεστημένες στοίβες και διευθύνσεις IP ανατεθημένες. Αυτό που χρειαζόμαστε σε αυτό το σημείο είναι εφαρμογές οι οποίες θα δημιουργήσουν κίνηση στο δίκτυό μας.

### 4.2.7 Εφαρμογές

Άλλη μια από τις κεντρικές αφαιρέσεις του συστήματος του ns-3 είναι η `Application`. Σε αυτό το σενάριο χρησιμοποιούμε δύο εξειδικεύσεις (specializations) της κεντρικής κλάσης του *ns-3* `Application`, που καλούνται `UdpEchoServerApplication` και `UdpEchoClientApplication`. Όπως έχουμε κάνει και στις προηγούμενες επεξηγήσεις μας, χρησιμοποιούμε βοηθούς αντικειμένων για να μας βοηθήσουν με τη ρύθμιση και τη διαχείριση των βασικών αντικειμένων. Εδώ, χρησιμοποιούμε αντικείμενα των `UdpEchoServerHelper` και `UdpEchoClientHelper` ώστε να κάνουμε τη ζωή μας πιο εύκολη.

#### UdpEchoServerHelper

Οι ακόλουθες γραμμές κώδικα στο σενάριο του παραδείγματός μας, `first.cc`, χρησιμοποιούνται για να θέσουν μια εφαρμογή UDP echo εξυπηρετητή σε έναν από τους κόμβους που έχουμε δημιουργήσει προηγούμενως.

```
UdpEchoServerHelper echoServer (9);

ApplicationContainer serverApps = echoServer.Install (nodes.Get (1));
serverApps.Start (Seconds (1.0));
serverApps.Stop (Seconds (10.0));
```

Η πρώτη γραμμή κώδικα στο παραπάνω απόσπασμα δηλώνει τον `UdpEchoServerHelper`. Ως συνήθως, αυτή δεν είναι η ίδια η εφαρμογή, αλλά ένα αντικείμενο που χρησιμοποιείται για να μας βοηθήσει να δημιουργήσουμε τις πραγματικές εφαρμογές. Μια από τις συμβάσεις μας είναι να τοποθετούμε τα *απαιτούμενα* `Attributes` στο δημιουργό (constructor) του βοηθού. Σε αυτή την περίπτωση, ο βοηθός δε μπορεί να κάνει τίποτα χρήσιμο εάν δεν του δοθεί ένας αριθμός port που να γνωρίζει ήδη ο πελάτης. Αντί να επιλεχθεί απλά ένας αριθμός και να ελπίζουμε ότι όλα θα δουλέψουν, απαιτούμε να υπάρχει ένας αριθμός port ως παράμετρος για τον δημιουργό.

Ο δημιουργός, με τη σειρά του, απλά κάνει μια ανάθεση τιμής `SetAttribute` με τη δοθείσα τιμή. Εάν θέλετε, μπορείτε να θέσετε το `Attribute` "Port" σε μια άλλη τιμή αργότερα χρησιμοποιώντας τη `SetAttribute`.

Παρόμοια με πολλά άλλα αντικείμενα βοηθών, το αντικείμενο `UdpEchoServerHelper` έχει μια μέθοδο `Install`. Είναι η εκτέλεση αυτής της μεθόδου που στην ουσία κάνει τη βασική εφαρμογή του echo εξυπηρετητή να δημιουργηθεί και να ενσωματωθεί σε έναν κόμβο. Με κάπως ενδιαφέροντα τρόπο, η μέθοδος `Install` δέχεται ένα `NodeContainter` ως παράμετρο, όπως και οι άλλες μέθοδοι `Install` που έχουμε δει. Είναι ουσιαστικά αυτό που μεταβιβάζεται στη μέθοδο παρόλο που δε φαίνεται έτσι σε αυτή την περίπτωση. Υπάρχει μια *υπόρρητη μετατροπή* (implicit conversion) της C++ που λαμβάνει χώρα εδώ και η οποία παίρνει το αποτέλεσμα της `nodes.Get (1)` (που επιστρέφει έναν έξυπνο δείκτη σε ένα αντικείμενο κόμβου — `Ptr<Node>`) και το χρησιμοποιεί σε έναν δημιουργό για έναν ανώνυμο `NodeContainer`, ο οποίος ύστερα μεταβιβάζεται στην `Install`. Εάν βρεθείτε ποτέ στο σημείο να ψάχνετε για μια συγκεκριμένη υπογραφή μεθόδου (method signature) σε κώδικα C++ που να μεταγλωττίζει και να τρέχει ομαλά, ανατρέξτε σε αυτών των ειδών τις υπόρρητες μετατροπές.

Τώρα βλέπουμε ότι η `echoServer.Install` πρόκειται να εγκαταστήσει μία `UdpEchoServerApplication` στον κόμβο που βρίσκεται στην πρώτη θέση του `NodeContainer` που χρησιμοποιήσαμε για να διαχειριστούμε τους κόμβους μας. Η `Install` θα επιστρέψει έναν container που θα κρατάει τους δείκτες για όλες τις εφαρμογές (μία σε αυτή την περίπτωση, καθώς δώσαμε ως όρισμα ένα `NodeContainer` που περιλαμβάνει έναν κόμβο) που έχουν δημιουργηθεί από τον βοηθό.

Οι εφαρμογές απαιτούν χρόνο για να «ξεκινήσουν» να δημιουργούν κίνηση και μπορεί να χρειαστούν προαιρετικά κάποιον χρόνο για να «σταματήσουν». Εμείς παρέχουμε και τα δύο. Αυτοί οι χρόνοι καθορίζονται με τη χρήση των μεθόδων `Start` και `Stop` του `ApplicationContainer`. Αυτές οι μέθοδοι δέχονται παραμέτρους τύπου `Time`. Σε αυτή την περίπτωση, χρησιμοποιούμε μια *ρητή* ακολουθία μετατροπών της C++ ώστε να πάρουμε τη μεταβλητή διπλής ακρίβειας (double) της C++ 1.0 και να τη μετατρέψουμε σε ένα αντικείμενο `Time` του *ns-3* χρησιμοποιώντας μια ρητή μετατροπή σε `Seconds`. Έχετε υπ' όψιν σας ότι οι κανόνες μετατροπής μπορούν να ελεγθούν από το συγγραφέα του μοντέλου, και ότι η C++ έχει τους δικούς της κανόνες, οπότε δεν μπορείτε πάντα απλά να υποθέτετε ότι οι παράμετροι θα μετατραπούν επιτυχώς για χάρη σας. Οι δύο γραμμές,

```
serverApps.Start (Seconds (1.0));
serverApps.Stop (Seconds (10.0));
```

θα κάνουν την εφαρμογή του echo εξυπηρετητή να ξεκινήσει (`Start`, να ενεργοποιήσει τον εαυτό της) στο πρώτο δευτερόλεπτο της προσομοίωσης και να σταματήσει (`Stop`, να απενεργοποιήσει τον εαυτό της) στο δέκατο δευτερόλεπτο της προσομοίωσης. Δεδομένου του γεγονότος ότι έχουμε δηλώσει ότι ένα γεγονός της προσομοίωσης (το γεγονός απενεργοποίησης της εφαρμογής) θα συμβεί στο δέκατο δευτερόλεπτο, η προσομοίωση θα διαρκέσει *τουλάχιστον* δέκα δευτερόλεπτα.

### UdpEchoClientHelper

Η εφαρμογή echo πελάτη καθορίζεται με μία μέθοδο η οποία είναι κατ' ουσίαν παρόμοια με αυτή για τον εξυπηρετητή. Υπάρχει μια βασική `UdpEchoClientApplication` που είναι διαχειρίσιμη από έναν `UdpEchoClientHelper`.

```
UdpEchoClientHelper echoClient (interfaces.GetAddress (1), 9);
echoClient.SetAttribute ("MaxPackets", UintegerValue (1));
echoClient.SetAttribute ("Interval", TimeValue (Seconds (1.0)));
echoClient.SetAttribute ("PacketSize", UintegerValue (1024));

ApplicationContainer clientApps = echoClient.Install (nodes.Get (0));
clientApps.Start (Seconds (2.0));
clientApps.Stop (Seconds (10.0));
```

Για τον echo πελάτη, χρειάζεται να θέσουμε πέντε διαφορετικά `Attributes`. Τα πρώτα δύο `Attributes` τίθενται κατά τη διάρκεια της δημιουργίας του `UdpEchoClientHelper`. Περνάμε τις παραμέτρους που χρησιμοποιούνται (εσωτερικά στον βοηθό) για να θέσουμε τα `Attributes` "RemoteAddress" και "RemotePort" σύμ-

φωνα με τη σύμβασή μας για προσδιορισμό των απαιτούμενων παραμέτρων `Attributes` στους δημιουργούς του helper.

Θυμηθείτε ότι χρησιμοποιήσαμε έναν `Ipv4InterfaceContainer` για να καταγράψουμε τις διευθύνσεις IP που αναθέσαμε στις συσκευές μας. Η διεπαφή στη θέση μηδέν στον container των `interfaces` θα ανταποκρίνεται στην IP διεύθυνση του κόμβου μηδέν στον container των `nodes`. Η διεπαφή στη θέση ένα στον container των `interfaces` αντιστοιχεί στην IP διεύθυνση του κόμβου ένα στον container των `nodes`. Έτσι, στην πρώτη γραμμή του κώδικα (παραπάνω), δημιουργούμε τον βοηθό και του λέμε να θέσει την απομακρυσμένη (remote) διεύθυνση του πελάτη έτσι ώστε να είναι η IP διεύθυνση που έχει ανατεθεί στον κόμβο στον οποίο βρίσκεται ο εξυπηρετητής. Του λέμε επίσης να κανονίσει να στέλνει τα πακέτα στο port εννέα.

Το `Attribute` "MaxPackets" λέει στον πελάτη το μέγιστο νούμερο των πακέτων που του επιτρέπουμε να στείλει κατά τη διάρκεια της προσομοίωσης. Το `Attribute` "Interval" λέει στον πελάτη πόσο πρέπει να περιμένει μεταξύ των πακέτων, και το `Attribute` "PacketSize" λέει στον πελάτη πόσο μεγάλο πρέπει να είναι το ωφέλιμο φορτίο σε κάθε πακέτο του. Με αυτόν τον συγκεκριμένο συνδυασμό από `Attributes`, λέμε στον πελάτη να στείλει ένα πακέτο μεγέθους 1024 byte.

Όπως και στην περίπτωση του echo εξυπηρετητή, λέμε στον echo πελάτη να ξεκινήσει (`Start`) και να σταματήσει (`Stop`) αλλά εδώ ενεργοποιούμε τον πελάτη ένα δευτερόλεπτο αφότου ο εξυπηρετητής έχει ενεργοποιηθεί (στο δεύτερο δευτερόλεπτο της προσομοίωσης).

### 4.2.8 Simulator

Αυτό που χρειαζόμαστε σε αυτό το σημείο είναι να εκτελέσουμε όντως την προσομοίωση. Αυτό γίνεται με τη χρήση της καθολικής συνάρτησης `Simulator::Run`.

```
Simulator::Run ();
```

Όταν καλέσαμε πριν τις μεθόδους,

```
serverApps.Start (Seconds (1.0));
serverApps.Stop (Seconds (10.0));
...
clientApps.Start (Seconds (2.0));
clientApps.Stop (Seconds (10.0));
```

προγραμματίσαμε όντως τα γεγονότα στον προσομοιωτή να γίνουν στο 1.0 δευτερόλεπτο, στα 2.0 δευτερόλεπτα και δύο γεγονότα να γίνουν στα 10.0 δευτερόλεπτα. Όταν καλείται η `Simulator::Run`, το σύστημα θα αρχίσει να κοιτάζει στη λίστα των προγραμματισμένων γεγονότων και να τα εκτελεί. Αρχικά θα εκτελέσει το γεγονός στο 1.0 δευτερόλεπτο, το οποίο θα ενεργοποιήσει την εφαρμογή του echo εξυπηρετητή (αυτό το γεγονός μπορεί, με τη σειρά του, να προγραμματίζει και άλλα γεγονότα). Έπειτα θα εκτελέσει το γεγονός που είναι προγραμματισμένο για t=2.0 δευτερόλεπτα, το οποίο θα ξεκινήσει την εφαρμογή του echo πελάτη. Ξανά, αυτό το γεγονός μπορεί να προγραμματίσει πολλά ακόμα γεγονότα. Η υλοποίηση εκκίνησης γεγονότων στην εφαρμογή του echo πελάτη θα ξεκινήσει τη φάση της μεταφοράς δεδομένων στην προσομοίωση στέλνοντας ένα πακέτο προς τον εξυπηρετητή.

Η πράξη της αποστολής του πακέτου προς τον εξυπηρετητή θα πυροδοτήσει μια αλυσίδα γεγονότων που θα προγραμματιστούν αυτόματα στο παρασκήνιο, και τα οποία θα εκτελέσουν τη λειτουργία του packet echo σύμφωνα με τις διάφορες χρονικές παραμέτρους που έχουμε θέσει εμείς στο σενάριο.

Τελικά, από τη στιγμή που στέλνουμε μόνο ένα πακέτο (θυμηθείτε ότι το `Attribute` `MaxPackets` τέθηκε στην τιμή ένα), η αλυσίδα των γεγονότων που θα πυροδοτηθούν από αυτή και μόνο την αίτηση echo στον πελάτη (client echo request) θα εξαλειφθεί και η προσομοίωση θα καταστεί αδρανής. Μόλις γίνει αυτό, τα εναπομείναντα γεγονότα θα είναι τα γεγονότα `Stop` για τον εξυπηρετητή και τον πελάτη. Μόλις αυτά τα γεγονότα εκτελούνται δεν υπάρχουν πλέον άλλα γεγονότα προς επεξεργασία και η `Simulator::Run` επιστρέφει τον έλεγχο. Η προσομοίωση τότε ολοκληρώνεται.

Αυτό που απομένει είναι η εκκαθάριση. Αυτή γίνεται με κλήση της καθολικής συνάρτησης `Simulator::Destroy`. Καθώς εκτελούνταν οι συναρτήσεις του βοηθού (ή κώδικας χαμηλού επιπέδου

του *ns-3*), το κανόνισαν έτσι ώστε να εισαχθούν hooks στον προσομοιωτή που να καταστρέφουν όλα τα αντικείμενα που δημιουργήθηκαν. Δε χρειάστηκε εσείς οι ίδιοι να καταγράψετε κανένα από αυτά τα αντικείμενα — αυτό που είχατε να κάνετε ήταν να καλέσετε την `Simulator::Destroy` και να βγείτε. Το σύστημα του *ns-3* θα αναλάβει το δύσκολο μέρος για εσάς. Οι υπόλοιπες γραμμές του πρώτου μας σεναρίου *ns-3*, `first.cc`, κάνουν ακριβώς αυτό:

```
  Simulator::Destroy ();
  return 0;
}
```

### Πότε θα σταματήσει ο προσομοιωτής;

Ο *ns-3* είναι ένας προσομοιωτής Διακριτών Γεγονότων (Discrete Event ή DE). Σε έναν τέτοιο προσομοιωτή, κάθε γεγονός συσχετίζεται με τον χρόνο εκτέλεσής του, και η προσομοίωση συνεχίζει εκτελώντας γεγονότα κατά τη χρονική σειρά του χρόνου προσομοίωσης. Τα γεγονότα μπορούν να προκαλέσουν τον προγραμματισμό μελλοντικών γεγονότων (για παράδειγμα, ένα χρονόμετρο μπορεί να επαναπρογραμματίσει τον εαυτό του ώστε να εκλείψει στο επόμενο διάστημα).

Τα αρχικά γεγονότα συνήθως πυροδοτούνται από το κάθε αντικείμενο, π.χ. το IPv6 θα προγραμματίσει ενημερώσεις δρομολογητών (router advertisements), παρακλήσεις σε γείτονες (neighbor solicitations), κτλ., μια εφαρμογή θα προγραμματίσει το γεγονός αποστολής του πρώτου πακέτου, κτλ.

Όταν ένα γεγονός βρίσκεται υπό επεξεργασία, μπορεί να παράξει μηδεν, ένα ή περισσότερα γεγονότα. Καθώς η προσομοίωση εκτελείται, «καταναλώνονται» γεγονότα, αλλά περισσότερα γεγονότα μπορεί (ή μπορεί και όχι) να δημιουργηθούν. Η προσομοίωση θα σταματήσει αυτόματα όταν δεν υπάρχουν άλλα γεγονότα στην ουρά των γεγονότων, ή όταν βρεθεί ένα ειδικό γεγονός παύσης (Stop). Το γεγονός παύσης δημιουργείται μέσω της συνάρτησης `Simulator::Stop (stopTime);`.

Υπάρχει μια χαρακτηριστική περίπτωση όπου η `Simulator::Stop` είναι απολύτως απαραίτητη για να σταματήσει η προσομοίωση: όταν υπάρχει ένα αυτο-συντηρούμενο γεγονός (self-sustaining event). Αυτο-συντηρούμενα (ή αναδρομικά) γεγονότα είναι γεγονότα που επαναπρογραμματίζουν συνέχεια τον εαυτό τους. Κατά συνέπεια, θα διατηρούν για πάντα την ουρά γεγονότων γεμάτη.

Υπάρχουν πολλά πρωτόκολλα και ενότητες που περιλαμβάνουν αναδρομικά γεγονότα, π.χ.:

- FlowMonitor - περιοδικός έλεγχος για χαμένα πακέτα
- RIPng - περιοδική εκπομπή για ανανέωση πινάκων δρομολόγησης
- κτλ.

Σε αυτές τις περιπτώσεις, η `Simulator::Stop` είναι απαραίτητη για να σταματήσει επιτυχώς η προσομοίωση. Επιπρόσθετα, όταν ο *ns-3* είναι σε κατάσταση προσομοίωσης, ο `RealtimeSimulator` (προσομοιωτής πραγματικού χρόνου) χρησιμοποιείται για να διατηρήσει το ρολόι της προσομοίωσης σε αντιστοιχία με το ρολόι της μηχανής, και η `Simulator::Stop` είναι αναγκαία για να σταματήσει τη διαδικασία.

Πολλά από τα προγράμματα προσομοίωσης σε αυτόν τον οδηγό δεν καλούν ρητώς τη `Simulator::Stop`, καθώς η ουρά των γεγονότων θα αδειάσει αυτόματα από γεγονότα. Ωστόσο, αυτά τα προγράμματα θα επιτρέψουν μια κλήση της `Simulator::Stop`. Για παράδειγμα, η ακόλουθη συμπληρωματική δήλωση στο πρόγραμμα του πρώτου παραδείγματος θα προγραμματίσει ένα ρητό σταμάτημα στα 11 δευτερόλεπτα:

```
+  Simulator::Stop (Seconds (11.0));
   Simulator::Run ();
   Simulator::Destroy ();
   return 0;
 }
```

Το παραπάνω δεν θα αλλάξει ουσιαστικά τη συμπεριφορά αυτού του προγράμματος, καθώς η συγκεκριμένη προσομοίωση τελειώνει εκ των πραγμάτων μετά από 10 δευτερόλεπτα. Αλλά εάν αλλάξατε το χρόνο σταματήματος στην παραπάνω δήλωση από 11 δευτερόλεπτα σε 1 δευτερόλεπτο, θα παρατηρούσατε ότι η προσομοίωση

σταματάει πριν τυπωθεί κάποια έξοδος στην οθόνη (καθώς η έξοδος προκύπτει περίπου στο δεύτερο δευτερόλεπτο του χρόνου προσομοίωσης).

Είναι σημαντικό να καλείτε τη `Simulator::Stop` *πριν* την κλήση της `Simulator::Run`. Διαφορετικά, η `Simulator::Run` μπορεί να μην επιστρέψει ποτέ τον έλεγχο στο κεντρικό πρόγραμμα ώστε αυτό να εκτελέσει το σταμάτημα!

### 4.2.9 Κάνοντας Build το Σενάριό σας

Έχουμε κάνει το build των απλών σεναρίων σας πολύ εύκλο. Αυτό που έχετε να κάνετε είναι απλά να μεταφέρετε το σενάριό σας στον κατάλογο scratch και θα κάνει αυτόματα build εάν εκτελέσετε το Waf. Ας το δοκιμάσουμε. Αντιγράψτε το `examples/tutorial/first.cc` στον κατάλογο `scratch` αφότου μεταβείτε στον κατάλογο του υψηλότερου επιπέδου.

```
$ cd ../..
$ cp examples/tutorial/first.cc scratch/myfirst.cc
```

Τώρα κάντε build το σενάριο του πρώτου παραδείγματός σας χρησιμοποιώντας το Waf:

```
$ ./waf
```

Θα πρέπει να δείτε μηνύματα που να αναφέρουν ότι το παράδειγμά σας `myfirst` έγινε build επιτυχώς.

```
Waf: Entering directory `/home/craigdo/repos/ns-3-allinone/ns-3-dev/build'
[614/708] cxx: scratch/myfirst.cc -> build/debug/scratch/myfirst_3.o
[706/708] cxx_link: build/debug/scratch/myfirst_3.o -> build/debug/scratch/myfirst
Waf: Leaving directory `/home/craigdo/repos/ns-3-allinone/ns-3-dev/build'
'build' finished successfully (2.357s)
```

Μπορείτε τώρα να τρέξετε το παράδειγμα (σημειώστε πως εάν θέλετε να κάνετε build το πρόγραμμά σας στον κατάλογο scratch θα πρέπει να το τρέξετε εκτός του καταλόγου scratch):

```
$ ./waf --run scratch/myfirst
```

Θα πρέπει να δείτε κάποια έξοδο:

```
Waf: Entering directory `/home/craigdo/repos/ns-3-allinone/ns-3-dev/build'
Waf: Leaving directory `/home/craigdo/repos/ns-3-allinone/ns-3-dev/build'
'build' finished successfully (0.418s)
Sent 1024 bytes to 10.1.1.2
Received 1024 bytes from 10.1.1.1
Received 1024 bytes from 10.1.1.2
```

Εδώ βλέπετε ότι το σύστημα κατασκευής (build) ελέγχει, ώστε να διασφαλίσει πως το αρχείο έχει γίνει build, και έπειτα το τρέχει. Παρατηρείτε ότι το στοιχείο καταγραφής στον echo πελάτη καταδεικνύει ότι έχει σταλεί ένα πακέτο μεγέθους 1024 byte προς τον echo εξυπηρετητή στη διεύθυνση 10.1.1.2. Βλέπετε επίσης πως το στοιχείο καταγραφής στον echo εξυπηρετητή λέει ότι έχει λάβει τα 1024 byte από τη διεύθυνση 10.1.1.1. Ο echo εξυπηρετητής μεταδίδει σιωπηλά το πακέτο και βλέπετε ότι ο echo πελάτης καταγράφει πως έλαβε το πακέτο του πίσω από τον εξυπηρετητή.

## 4.3 Ο Πηγαίος Κώδικας του ns-3

Τώρα που έχετε χρησιμοποιήσει κάποιους από τους βοηθούς του *ns-3*, μπορεί να θέλετε να ρίξετε μια ματιά σε κάποιο μέρος από τον πηγαίο κώδικα που υλοποιεί αυτή τη λειτουργία. Ο πιο πρόσφατος κώδικας μπορεί να βρεθεί στον δικτυακό μας εξυπηρετητή στον ακόλουθο σύνδεσμο: http://code.nsnam.org/ns-3-dev. Εκεί, θα δείτε μια σελίδα περίληψης του Mercurial για το δέντρο ανάπτυξης του *ns-3*.

Στην κορυφή της σελίδας, θα δείτε έναν αριθμό από συνδέσμους,

```
summary | shortlog | changelog | graph | tags | files
```

Προχωρήστε και επιλέξτε το σύνδεσμο `files`. Το υψηλότερο επίπεδο των περισσότερων αποθετηρίων μας θα φαίνεται κάπως έτσι:

```
drwxr-xr-x                                [up]
drwxr-xr-x                                bindings python  files
drwxr-xr-x                                doc              files
drwxr-xr-x                                examples         files
drwxr-xr-x                                ns3              files
drwxr-xr-x                                scratch          files
drwxr-xr-x                                src              files
drwxr-xr-x                                utils            files
-rw-r--r-- 2009-07-01 12:47 +0200 560     .hgignore        file | revisions | annotate
-rw-r--r-- 2009-07-01 12:47 +0200 1886    .hgtags          file | revisions | annotate
-rw-r--r-- 2009-07-01 12:47 +0200 1276    AUTHORS          file | revisions | annotate
-rw-r--r-- 2009-07-01 12:47 +0200 30961   CHANGES.html      file | revisions | annotate
-rw-r--r-- 2009-07-01 12:47 +0200 17987   LICENSE          file | revisions | annotate
-rw-r--r-- 2009-07-01 12:47 +0200 3742    README           file | revisions | annotate
-rw-r--r-- 2009-07-01 12:47 +0200 16171   RELEASE_NOTES    file | revisions | annotate
-rw-r--r-- 2009-07-01 12:47 +0200 6       VERSION          file | revisions | annotate
-rwxr-xr-x 2009-07-01 12:47 +0200 88110   waf              file | revisions | annotate
-rwxr-xr-x 2009-07-01 12:47 +0200 28      waf.bat          file | revisions | annotate
-rw-r--r-- 2009-07-01 12:47 +0200 35395   wscript          file | revisions | annotate
-rw-r--r-- 2009-07-01 12:47 +0200 7673    wutils.py        file | revisions | annotate
```

Τα σενάρια παραδειγμάτων μας είναι στον κατάλογο `examples`. Εάν κάνετε κλικ στα `examples` θα δείτε μια λίστα από υποκαταλόγους. Ένα από τα αρχεία στον υποκατάλογο `tutorial` είναι το `first.cc`. Εάν κάνετε κλικ στο `first.cc` θα βρείτε τον κώδικα τον οποίο μόλις εξετάσατε.

Ο πηγαίος κώδικας είναι κυρίως στον κατάλογο `src`. Μπορείτε να δείτε τον πηγαίο κώδικα είτε κάνοντας κλικ στο όνομα του καταλόγου, είτε κάνοντας κλικ στο σύνδεσμο `files` στα δεξιά του ονόματος του καταλόγου. Εάν κάνετε κλικ στον κατάλογο `src`, θα μεταφερθείτε στην λίστα των υποκαταλόγων του `src`. Εάν τότε κάνετε κλικ στον υποκατάλογο `core`, θα βρείτε μία λίστα από αρχεία. Το πρώτο αρχείο που θα βρείτε (κατά τη διάρκεια της συγγραφής του παρόντος οδηγού) είναι το `abort.h`. Εάν κάνετε κλικ στο σύνδεσμο `abort.h`, θα μεταβείτε στο πηγαίο αρχείο του `abort.h`, το οποίο περιέχει χρήσιμες μακροεντολές για την έξοδο από σενάρια σε περίπτωση που ανιχνευθούν αφύσικες συνθήκες.

Ο πηγαίος κώδικας για τους βοηθούς που χρησιμοποιήσαμε σε αυτό το κεφάλαιο μπορεί να βρεθεί στον κατάλογο `src/applications/helper`. Μπορείτε ελεύθερα να ψάξετε στο δέντρο καταλόγων για να αποκτήσετε μια αίσθηση του τι βρίσκεται εκεί και του στυλ των προγραμμάτων του *ns-3*.

ΚΕΦΑΛΑΙΟ

# ΠΕΝΤΕ

# ΜΙΚΡΟΡΥΘΜΙΣΕΙΣ

## 5.1 Χρησιμοποιώντας την Ενότητα Καταγραφής

Έχουμε ήδη δει σύντομα την ενότητα καταγραφής του *ns-3* κατά τη διάρκεια του σεναρίου προσομοίωσης `first.cc`. Εδώ θα εξετάσουμε εις βάθος το σύστημα καταγραφής και τις περιπτώσεις χρήσης τις οποίες παρέχει.

### 5.1.1 Σύνοψη Καταγραφής

Είναι κοινή πρακτική τα μεγάλα συστήματα, όπως το *ns-3*, να παρέχουν τη δυνατότητα καταγραφής μηνυμάτων. Σε μερικές περιπτώσεις καταγράφονται μόνο τα μηνύματα λάθους στο "operator console" (που τυπικά είναι το `stderr` στα συστήματα Unix). Σε άλλα συστήματα, καταγράφονται μηνύματα προειδοποίησης καθώς και άλλα μηνύματα που παρέχουν περισσότερη πληροφορία. Τέλος σε μερικές περιπτώσεις, η διαδικασία καταγραφής χρησιμοποιείται για να παράγει μηνύματα αποσφαλμάτωσης που όταν παρουσιαστούν θολώνουν την έξοδο.

Κατά το *ns-3*, όλα τα παραπάνω επίπεδα καταγραφής είναι χρήσιμα και για το λόγο αυτό παρέχεται η δυνατότητα καταγραφής μηνυμάτων σε πολλαπλά επίπεδα. Η διαδικασία καταγραφής μπορεί να απενεργοποιηθεί πλήρως, να ενεργοποιηθεί ανά συστατικό στοιχείο (component), ή να ενεργοποιηθεί συνολικά στο σύστημα. Επίσης δίνεται η δυνατότητα επιλογής του εύρους των μηνυμάτων σε επίπεδα. Η ενότητα καταγραφής του *ns-3* παρέχει έναν απλό και εύκολο τρόπο για εξαγωγή χρήσιμων πληροφοριών από μια προσομοίωση.

Πρέπει να γίνει σαφές ότι παρέχεται ένας μηχανισμός γενικού σκοπού — ιχνηλασίας — για την εξαγωγή δεδομένων από τα μοντέλα του χρήστη, ο οποίος θα έπρεπε να προτιμάται για την έξοδο της εξομοίωσης (δείτε τον τομέα "Χρησιμοποιώντας το Σύστημα Ιχνηλασίας" για περισσότερες πληροφορίες). Ο μηχανισμός της καταγραφής αντίστοιχα θα πρέπει να προτιμάται για πληροφορίες αποσφαλμάτωσης, για μηνύματα προειδοποίησης και λάθους, καθώς και για περιπτώσεις που χρειάζεται να πάρουμε ένα γρήγορο μήνυμα από ένα μοντέλο ή σενάριο προσομοίωσης.

Στο σύστημα έχουν οριστεί προς το παρόν επτά επίπεδα μηνυμάτων καταγραφής αυξανόμενου εύρους μηνυμάτων.

- LOG_ERROR — Καταγραφή μηνυμάτων λάθους (σχετική μακροεντολή: NS_LOG_ERROR);
- LOG_WARN — Καταγραφή μηνυμάτων προειδοποίησης (σχετική μακροεντολή: NS_LOG_WARN);
- LOG_DEBUG — Καταγραφή σχετικά σπανίων, προσαρμοσμένων μηνυμάτων αποσφαλμάτωσης (σχετική μακροεντολή: NS_LOG_DEBUG);
- LOG_INFO — Καταγραφή πληροφοριακών μηνυμάτων για την πρόοδο του προγράμματος (σχετική μακροεντολή: NS_LOG_INFO);
- LOG_FUNCTION — Καταγραφή ενός μηνύματος περιγραφής για κάθε συνάρτηση που καλείται (δύο σχετικές μακροεντολές: NS_LOG_FUNCTION, που χρησιμοποιείται για member functions, και NS_LOG_FUNCTION_NOARGS, που χρησιμοποιείται για static functions);

- LOG_LOGIC – Καταγραφή μηνυμάτων που περιγράφουν τη λογική ροή μέσα σε μια συνάρτηση (σχετική μακροεντολή: NS_LOG_LOGIC);
- LOG_ALL — Καταγραφή όλων των παραπάνω (καμία σχετική μακροεντολή).

Για κάθε επίπεδο, εκτός από το LOG_TYPE υπάρχει και το LOG_LEVEL_TYPE το οποίο αν χρησιμοποιηθεί, ενεργοποιεί την καταγραφή όλων των επιπέδων που βρίσκονται πάνω από αυτό. (Κατά συνέπεια, τα LOG_ERROR και LOG_LEVEL_ERROR είναι ισοδύναμα μεταξύ τους όπως επίσης και τα LOG_ALL και and LOG_LEVEL_ALL.) Για παράδειγμα, ενεργοποιώντας το LOG_INFO θα παραχθούν μόνο μηνύματα από την μακροεντολή NS_LOG_INFO, ενώ ενεργοποιώντας το LOG_LEVEL_INFO θα παραχθούν επίσης και μηνύματα από τις μακροεντολές NS_LOG_DEBUG, NS_LOG_WARN και NS_LOG_ERROR.

Επίσης παρέχεται μια μακροεντολή καταγραφής χωρίς περιορισμούς, ανεξάρτητη από τα επίπεδα καταγραφής ή την επιλογή συστατικού στοιχείου.

- NS_LOG_UNCOND – Καταγραφή του μηνύματος χωρίς περιορισμούς (κανένα σχετικό επίπεδο καταγραφής).

Το κάθε επίπεδο μπορεί να ζητηθεί μεμονωμένο ή σε συνδυασμό με τα άλλα. Η καταγραφή αντίστοιχα μπορεί να ρυθμιστεί μέσω μιας μεταβλητής του περιβάλλοντος φλοιού (NS_LOG) ή μέσω καταγραφής της κλήση συνάρτησης συστήματος. Όπως έχουμε ήδη δει νωρίτερα σε αυτό τον οδηγό, το σύστημα καταγραφής έχει τεκμηρίωση Doxygen και ίσως είναι χρήσιμο να μελετήσετε την τεκμηρίωση της Ενότητας Καταγραφής.

Αφού έχετε διαβάσει την τεκμηρίωση σε μεγάλο βαθμό, είναι ώρα να χρησιμοποιήσουμε αυτή τη γνώση για να εξάγουμε μερικές χρήσιμες πληροφορίες από το παράδειγμα `scratch/myfirst.cc` που έχετε ήδη δημιουργήσει.

### 5.1.2 Ενεργοποίηση Καταγραφής

Μπορούμε να χρησιμοποιήσουμε τη μεταβλητή συστήματος NS_LOG για να ενεργοποιήσουμε επιπλέον λειτουργίες καταγραφής, αλλά ας τρέξουμε αρχικά το παρακάτω σενάριο

```
$ ./waf --run scratch/myfirst
```

Θα πρέπει να βλέπετε τώρα τη γνώριμη έξοδο του πρώτου παραδείγματος από τα προγράμματα του *ns-3*.

```
$ Waf: Entering directory `/home/craigdo/repos/ns-3-allinone/ns-3-dev/build'
Waf: Leaving directory `/home/craigdo/repos/ns-3-allinone/ns-3-dev/build'
'build' finished successfully (0.413s)
Sent 1024 bytes to 10.1.1.2
Received 1024 bytes from 10.1.1.1
Received 1024 bytes from 10.1.1.2
```

Τα μηνύματα “Sent” και “Received” που βλέπετε παραπάνω είναι στην πραγματικότητα μηνύματα καταγραφής από τα `UdpEchoClientApplication` και `UdpEchoServerApplication`. Μπορούμε, για παράδειγμα, να ζητήσουμε από την εφαρμογή του χρήστη να τυπώσει περισσότερες πληροφορίες θέτοντας το επίπεδο καταγραφής της μέσω της μεταβλητής περιβάλλοντος NS_LOG.

Στη συνέχεια θεωρούμε ότι ο χρήστης χρησιμοποιεί έναν φλοιό που χρησιμοποιεί τη σύνταξη “VARIABLE=value” όπως ο sh. Αν χρησιμοποιείτε ένα φλοιό τύπου csh, τότε πρέπει να μετασχηματίσετε τα παρακάτω παραδείγματα σε σύνταξη “setenv VARIABLE value”.

Αυτή τη στιγμή, η εφαρμογή UDP echo client ανταποκρίνεται στην παρακάτω γραμμή κώδικα του `scratch/myfirst.cc`,

```
LogComponentEnable("UdpEchoClientApplication", LOG_LEVEL_INFO);
```

Αυτή η γραμμή κώδικα ενεργοποιεί το επίπεδο καταγραφής `LOG_LEVEL_INFO`. Όταν περάσουμε κάποια παράμετρο επιπέδου καταγραφής, στην ουσία ενεργοποιούμε το συγκεκριμένο επίπεδο και όλα τα χαμηλότερά του. Στο συγκεκριμένο παράδειγμα, ενεργοποιούμε τα `NS_LOG_INFO`, `NS_LOG_DEBUG`, `NS_LOG_WARN` και

NS_LOG_ERROR. Μπορούμε να αυξήσουμε το επίπεδο καταγραφής και να πάρουμε περισσότερες πληροφορίες χωρίς να χρειαστεί να αλλάξουμε το σενάριο και να επαναμεταγλωτίσσουμε, αν θέσουμε τη μεταβλητή περιβάλλοντος NS_LOG ως εξής:

```
$ export NS_LOG=UdpEchoClientApplication=level_all
```

Αυτό θα θέσει τη μεταβλητή περιβάλλοντος NS_LOG στο αλφαριθμητικό,

```
UdpEchoClientApplication=level_all
```

Το αριστερό σκέλος της ανάθεσης είναι το όνομα του στοιχείου καταγραφής που θέλουμε να θέσουμε, ενώ το δεξί σκέλος είναι το όρισμα που θέλουμε να χρησιμοποιήσουμε. Στην περίπτωσή μας θα ενεργοποιήσουμε όλα τα επίπεδα αποσφαλμάτωσης για τη συγκεκριμένη εφαρμογή. Αν τρέξετε το σενάριο θέτοντας την NS_LOG με αυτόν τον τρόπο, το σύστημα καταγραφής του *ns-3* θα δει την αλλαγή και θα πρέπει να δείτε την παρακάτω έξοδο:

```
Waf: Entering directory `/home/craigdo/repos/ns-3-allinone/ns-3-dev/build'
Waf: Leaving directory `/home/craigdo/repos/ns-3-allinone/ns-3-dev/build'
'build' finished successfully (0.404s)
UdpEchoClientApplication:UdpEchoClient()
UdpEchoClientApplication:SetDataSize(1024)
UdpEchoClientApplication:StartApplication()
UdpEchoClientApplication:ScheduleTransmit()
UdpEchoClientApplication:Send()
Sent 1024 bytes to 10.1.1.2
Received 1024 bytes from 10.1.1.1
UdpEchoClientApplication:HandleRead(0x6241e0, 0x624a20)
Received 1024 bytes from 10.1.1.2
UdpEchoClientApplication:StopApplication()
UdpEchoClientApplication:DoDispose()
UdpEchoClientApplication:~UdpEchoClient()
```

Η πρόσθετη πληροφορία αποσφαλμάτωσης που παρέχεται από την εφαρμογή βρίσκεται στο επίπεδο NS_LOG_FUNCTION. Αυτή εμφανίζεται κάθε φορά που καλείται μια συνάρτηση της εφαρμογής. Γενικά η χρήση της NS_LOG_FUNCTION(this) ενδείκνυται σε member functions, ενώ η NS_LOG_FUNCTION_NOARGS() σε static functions. Σημειώστε όμως ότι στο σύστημα *ns-3*, δεν υπάρχει η απαίτηση τα μοντέλα να υποστηρίζουν κάποια συγκεκριμένη λειτουργία καταγραφής. Η απόφαση για το εύρος της πληροφορίας που καταγράφεται, επαφίεται στον προγραμματιστή του μοντέλου. Σε περίπτωση εφαρμογής αντανάκλασης, ένα μεγάλο μέρος της εξόδου καταγραφής είναι διαθέσιμο.

Μπορείτε να δείτε μια καταγραφή των κλήσεων σε συναρτήσεις που έγιναν στην εφαρμογή. Αν κοιτάξετε προσεκτικά, θα παρατηρήσετε μια μονή στήλη μεταξύ του αλφαριθμητικού UdpEchoClientApplication και του ονόματος της μεθόδου, αντί του τελεστή `::` της C++ που θα περιμένατε. Αυτό γίνεται εσκεμμένα.

Το όνομα δεν είναι στην πραγματικότητα το όνομα μιας κλάσης, αλλά το όνομα του στοιχείου καταγραφής. Όταν υπάρχει αντιστοίχηση 1-προς-1 μεταξύ του αρχείου πηγής και της κλάσης, το όνομα θα είναι γενικά ίδιο με της κλάσης. Η μονή στήλη χρησιμοποιείται αντί της διπλής για να διαχωρίσει το στοιχείο καταγραφής από το όνομα της κλάσης.

Σε μερικές περιπτώσεις είναι δύσκολο να προσδιορίσεις ποια μέθοδος ακριβώς παράγει ένα μήνυμα καταγραφής. Αν κοιτάξετε το παραπάνω κείμενο, ίσως αναρωτιέστε από που προέρχεται το αλφαριθμητικό "`Received 1024 bytes from 10.1.1.2`". Μπορείτε να λύσετε την απορία σας μέσω του επιπέδου `prefix_func` στη μεταβλητή συστήματος NS_LOG. Δοκιμάστε το παρακάτω,

```
$ export 'NS_LOG=UdpEchoClientApplication=level_all|prefix_func'
```

Σημειώστε ότι τα εισαγωγικά χρειάζονται, αφού η κάθετη στήλη χρησιμοποιείται στο Unix για να υποδείξει τον τελεστή Ή.

Τώρα αν τρέξετε το σενάριο, θα παρατηρήσετε ότι το σύστημα καταγραφής προσθέτει σε κάθε μήνυμα ένα πρόθεμα με το όνομα του στοιχείου.

```
Waf: Entering directory `/home/craigdo/repos/ns-3-allinone/ns-3-dev/build'
Waf: Leaving directory `/home/craigdo/repos/ns-3-allinone/ns-3-dev/build'
'build' finished successfully (0.417s)
UdpEchoClientApplication:UdpEchoClient()
UdpEchoClientApplication:SetDataSize(1024)
UdpEchoClientApplication:StartApplication()
UdpEchoClientApplication:ScheduleTransmit()
UdpEchoClientApplication:Send()
UdpEchoClientApplication:Send(): Sent 1024 bytes to 10.1.1.2
Received 1024 bytes from 10.1.1.1
UdpEchoClientApplication:HandleRead(0x6241e0, 0x624a20)
UdpEchoClientApplication:HandleRead(): Received 1024 bytes from 10.1.1.2
UdpEchoClientApplication:StopApplication()
UdpEchoClientApplication:DoDispose()
UdpEchoClientApplication:~UdpEchoClient()
```

Μπορείτε να ταυτοποιήσετε τώρα όλα τα μηνύματα που προέρχονται από την εφαρμογή πελάτη UDP echo. Το μήνυμα “Received 1024 bytes from 10.1.1.2” φαίνεται τώρα ξεκάθαρα ότι προέρχεται από την εφαρμογή του πελάτη. Αντίστοιχα το άλλο μήνυμα προέρχεται από την εφαρμογή του εξυπηρετητή UDP echo. Μπορούμε να ενεργοποιήσουμε το στοιχείο αυτό, προσθέτοντας μια λίστα στοιχείων στη μεταβλητή περιβάλλοντος NS_LOG.

```
$ export 'NS_LOG=UdpEchoClientApplication=level_all|prefix_func:
              UdpEchoServerApplication=level_all|prefix_func'
```

Προσοχή: Θα χρειαστεί να αφαιρέσετε τη νέα γραμμή μετά το : στο παραπάνω κείμενο του παραδείγματος, το οποίο υπάρχει απλά για λόγους μορφοποίησης.

Αν τρέξετε το σενάριο τώρα, θα παρατηρήσετε όλα τα μηνύματα τόσο από την εφαρμογή του πελάτη όσο και του εξυπηρετητή. Αυτό μπορεί να αποδειχτεί πολύ χρήσιμο σε περιπτώσεις προβλημάτων αποσφαλμάτωσης.

```
Waf: Entering directory `/home/craigdo/repos/ns-3-allinone/ns-3-dev/build'
Waf: Leaving directory `/home/craigdo/repos/ns-3-allinone/ns-3-dev/build'
'build' finished successfully (0.406s)
UdpEchoServerApplication:UdpEchoServer()
UdpEchoClientApplication:UdpEchoClient()
UdpEchoClientApplication:SetDataSize(1024)
UdpEchoServerApplication:StartApplication()
UdpEchoClientApplication:StartApplication()
UdpEchoClientApplication:ScheduleTransmit()
UdpEchoClientApplication:Send()
UdpEchoClientApplication:Send(): Sent 1024 bytes to 10.1.1.2
UdpEchoServerApplication:HandleRead(): Received 1024 bytes from 10.1.1.1
UdpEchoServerApplication:HandleRead(): Echoing packet
UdpEchoClientApplication:HandleRead(0x624920, 0x625160)
UdpEchoClientApplication:HandleRead(): Received 1024 bytes from 10.1.1.2
UdpEchoServerApplication:StopApplication()
UdpEchoClientApplication:StopApplication()
UdpEchoClientApplication:DoDispose()
UdpEchoServerApplication:DoDispose()
UdpEchoClientApplication:~UdpEchoClient()
UdpEchoServerApplication:~UdpEchoServer()
```

Σε κάποιες περιπτώσεις είναι επίσης χρήσιμο να μπορούμε να δούμε τον χρόνο εξομοίωσης κατά τον οποίο παράχθηκε ένα μήνυμα καταγραφής. Μπορείτε να το κάνετε αυτό με τον τελεστή Ή στο ψηφίο prefix_time.

```
$ export 'NS_LOG=UdpEchoClientApplication=level_all|prefix_func|prefix_time:
              UdpEchoServerApplication=level_all|prefix_func|prefix_time'
```

Και εδώ, όπως προηγουμένως, πρέπει να αφαιρέσετε τη νέα γραμμή. Αν τρέξετε το σενάριο θα δείτε την παρακάτω έξοδο:

```
Waf: Entering directory `/home/craigdo/repos/ns-3-allinone/ns-3-dev/build'
Waf: Leaving directory `/home/craigdo/repos/ns-3-allinone/ns-3-dev/build'
'build' finished successfully (0.418s)
0s UdpEchoServerApplication:UdpEchoServer()
0s UdpEchoClientApplication:UdpEchoClient()
0s UdpEchoClientApplication:SetDataSize(1024)
1s UdpEchoServerApplication:StartApplication()
2s UdpEchoClientApplication:StartApplication()
2s UdpEchoClientApplication:ScheduleTransmit()
2s UdpEchoClientApplication:Send()
2s UdpEchoClientApplication:Send(): Sent 1024 bytes to 10.1.1.2
2.00369s UdpEchoServerApplication:HandleRead(): Received 1024 bytes from 10.1.1.1
2.00369s UdpEchoServerApplication:HandleRead(): Echoing packet
2.00737s UdpEchoClientApplication:HandleRead(0x624290, 0x624ad0)
2.00737s UdpEchoClientApplication:HandleRead(): Received 1024 bytes from 10.1.1.2
10s UdpEchoServerApplication:StopApplication()
10s UdpEchoClientApplication:StopApplication()
UdpEchoClientApplication:DoDispose()
UdpEchoServerApplication:DoDispose()
UdpEchoClientApplication:~UdpEchoClient()
UdpEchoServerApplication:~UdpEchoServer()
```

Βλέπετε πως έγινε κλήση στο δημιουργό της UdpEchoServer τη χρονική στιγμή 0. Αυτό στην πραγματικότητα συμβαίνει πριν ξεκινήσει η εξομοίωση, αλλά ο χρόνος που εμφανίζεται είναι το 0. Το ίδιο συμβαίνει και για το δημιουργό της UdpEchoClient.

Θυμηθείτε ότι στο σενάριο `scratch/first.cc` η ενεργοποίηση την εφαρμογής του εξυπηρετητή γίνεται το πρώτο δευτερόλεπτο της εξομοίωσης. Μπορείτε να δείτε ότι η μέθοδος του εξυπηρετητή `StartApplication` όντως καλείται στη χρονική στιγμή 1. Μπορείτε επίσης να δείτε ότι η εφαρμογή του πελάτη ξεκινάει τη χρονική στιγμή 2, όπως ζητήσαμε στο σενάριο.

Μπορείτε να παρακολουθήσετε την πρόοδο της εξομοίωσης από την κλήση `ScheduleTransmit` στον πελάτη, που καλεί την `Send` στην επανάκληση `HandleRead` στην εφαρμογή του εξυπηρετητή. Σημειώστε ότι ο παρερχόμενος χρόνος για την αποστολή του πακέτου στη σύνδεση είναι 3.69 δευτερόλεπτα. Μπορείτε να δείτε επίσης το μήνυμα καταγραφής του εξυπηρετητή που αναφέρει ότι το πακέτο έφυγε και στη συνέχεια, μετά από την καθυστέρηση του καναλιού, βλέπετε την άφιξη του πακέτου στον πελάτη μέσω της μεθόδου `HandleRead`.

Υπάρχουν επίσης πολλά που συμβαίνουν κατά τη διάρκεια της εξομοίωσης τα οποία δεν εμφανίζονται. Μπορείτε πολύ εύκολα να παρακολουθήσετε ολόκληρη τη διαδικασία αν θέσετε τη μεταβλητή `NS_LOG` στην παρακάτω τιμή,

```
$ export 'NS_LOG=*=level_all|prefix_func|prefix_time'
```

Ο αστερίσκος στην παραπάνω εντολή είναι ο τελεστής που δηλώνει ότι θέλουμε να ενεργοποιηθεί η καταγραφή σε όλα τα συστατικά στοιχεία που χρησιμοποιούνται στην εξομοίωση. Δεν θα συμπεριλάβουμε εδώ την έξοδο (μια που αυτή παράγει 1265 γραμμές απλά για ένα πακέτο), αλλά μπορείτε να ανακατευθύνετε την πληροφορία αυτή σε ένα αρχείο και να το ανοίξετε με κάποιον επεξεργαστή κειμένου,

```
$ ./waf --run scratch/myfirst > log.out 2>&1
```

Προσωπικά χρησιμοποιώ αυτή τη φλύαρη μέθοδο καταγραφής όταν παρουσιάζεται ένα πρόβλημα και δεν έχω την παραμικρή ιδέα που βρίσκεται το λάθος. Μπορώ να ακολουθήσω τη ροή της εκτέλεσης του κώδικα πολύ εύκολα χωρίς να χρειάζεται να θέσω σημεία διακοπής (breakpoints) ή να εξετάσω βήμα-βήμα τον κώδικα στον αποσφαλματωτή. Μπορώ απλά να ανοίξω την έξοδο στον αγαπημένο μου επεξεργαστή κειμένου και να ψάξω για πράγματα που περιμένω να συμβαίνουν, αλλά και για πράγματα που δεν περιμένω να συμβαίνουν. Όταν έχω μια γενική ιδέα του τι πάει λάθος, μεταβαίνω σε έναν αποσφαλματωτή για μια πλήρη εξέταση του προβλήματος.

Αυτού του είδους η έξοδος μπορεί να είναι ιδιαίτερα χρήσιμη όταν το σενάριο κάνει κάτι τελείως απρόβλεπτο. Αν χρησιμοποιήσετε απλά τον αποσφαλματωτή, μπορείτε να παραβλέψετε τελείως μια απρόβλεπτη συμπεριφορά. Με την καταγραφή μπορούμε να την εντοπίσουμε γρήγορα.

### 5.1.3 Προσθήκη Καταγραφής σε Κώδικα

Μπορείτε να προσθέσετε νέες καταγραφές στις εξομοιώσεις σας καλώντας το στοιχείο καταγραφής μέσω διαφόρων μακροεντολών. Ας το επιχειρήσουμε στο σενάριο εξομοίωσης `myfirst.cc` που βρίσκεται στον φάκελο `scratch`.

Θυμηθείτε ότι έχουμε ορίσει ένα στοιχείο καταγραφής σε εκείνο το σενάριο:

```
NS_LOG_COMPONENT_DEFINE ("FirstScriptExample");
```

Γνωρίζετε τώρα ότι μπορείτε να ενεργοποιήσετε όλες τις δυνατές καταγραφές για αυτό το στοιχείο, θέτοντας τη μεταβλητή περιβάλλοντος `NS_LOG` σε κάποιο επίπεδο. Ας προχωρήσουμε στην προσθήκη καταγραφής στο σενάριο. Η μακροεντολή που προσθέτει καταγραφή σε επίπεδο πληροφοριακών μηνυμάτων είναι η `NS_LOG_INFO`. Θέλουμε να προσθέσουμε ένα μήνυμα (πριν αρχίσουμε να δημιουργούμε κόμβους) που αναφέρει ότι το σενάριο δημιουργεί μια τοπολογία "Creating Topology". Αυτό γίνεται όπως δείχνουμε στον παρακάτω κώδικα,

Ανοίξτε το `scratch/myfirst.cc` σε έναν επεξεργαστή κειμένου και προσθέστε τη γραμμή,

```
NS_LOG_INFO ("Creating Topology");
```

αμέσως πριν από τις γραμμές,

```
NodeContainer nodes;
nodes.Create (2);
```

Τώρα ας οικοδομήσουμε το σενάριο χρησιμοποιώντας το waf και καθαρίζοντας τη μεταβλητή `NS_LOG` ώστε να απενεργοποιήσουμε το torrent της καταγραφής που είχαμε προηγουμένως ενεργοποιήσει:

```
$ ./waf
$ export NS_LOG=
```

Αν τρέξετε το σενάριο τώρα,

```
$ ./waf --run scratch/myfirst
```

δεν θα δείτε το νέο μήνυμα, αφού το σχετικό στοιχείο καταγραφής (`FirstScriptExample`) δεν έχει ενεργοποιηθεί. Για να δείτε το μήνυμά σας θα πρέπει να το ενεργοποιήσετε με επίπεδο καταγραφής μεγαλύτερο ή ίσο με `NS_LOG_INFO`. Αν θέλετε απλά να δείτε το συγκεκριμένο επίπεδο καταγραφής, μπορείτε να το ενεργοποιήσετε ως εξής,

```
$ export NS_LOG=FirstScriptExample=info
```

Αν τρέξετε τώρα το σενάριο, θα δείτε το μήνυμα καταγραφής "Creating Topology",

```
Waf: Entering directory `/home/craigdo/repos/ns-3-allinone/ns-3-dev/build'
Waf: Leaving directory `/home/craigdo/repos/ns-3-allinone/ns-3-dev/build'
'build' finished successfully (0.404s)
Creating Topology
Sent 1024 bytes to 10.1.1.2
Received 1024 bytes from 10.1.1.1
Received 1024 bytes from 10.1.1.2
```

## 5.2 Χρησιμοποιώντας ορίσματα γραμμής εντολών

### 5.2.1 Παρακάμπτοντας Προκαθορισμένα Ορίσματα

Ένας άλλος τρόπος που μπορείτε να αλλάξετε τον τρόπο που τα *ns-3* σενάρια συμπεριφέρονται, χωρίς να χρειάζεται επεξεργασία και οικοδόμηση, είναι μέσω *ορισμάτων γραμμής εντολών*. Παρέχουμε ένα μηχανισμό που να αναλύσει τα ορίσματα γραμμής εντολών και αυτόματα θέτει τις τοπικές και καθολικές μεταβλητές με βάση τα ορίσματα αυτά.

Το πρώτο βήμα για τη χρήση του συστήματος ορισμάτων γραμμής εντολών, είναι να δηλώσουμε τον αναλυτή γραμμής εντολών. Αυτό γίνεται πολύ απλά (στο κύριο πρόγραμμα σας) όπως στον ακόλουθο κώδικα,

```
int
main (int argc, char *argv[])
{
  ...

  CommandLine cmd;
  cmd.Parse (argc, argv);

  ...
}
```

Αυτό το απλό απόσπασμα δύο γραμμών είναι πραγματικά πολύ χρήσιμο από μόνο του. Ανοίγει την πόρτα για τα συστήματα καθολικών μεταβλητών και `Attributes` του *ns-3*. Προσθέστε αυτές τις δύο γραμμές κώδικα στο σενάριο `scratch/first.cc` στην αρχή της `main`. Οικοδομήστε το σενάριο και τρέξτε το, αλλά ζητήστε βοήθεια από το σενάριο με τον ακόλουθο τρόπο,

```
$ ./waf --run "scratch/myfirst --PrintHelp"
```

Αυτό θα ζητήσει από τον Waf να τρέξει το σενάριο `scratch/myfirst` και να περάσει το όρισμα γραμμής εντολών `--PrintHelp` στο σενάριο. Τα εισαγωγικά απαιτούνται για να ορίσουμε ποιο από τα προγράμματα παίρνει το κάθε όρισμα. Ο αναλυτής της γραμμής εντολών θα δει το όρισμα `--PrintHelp` και θα αποκριθεί με,

```
Waf: Entering directory `/home/craigdo/repos/ns-3-allinone/ns-3-dev/build'
Waf: Leaving directory `/home/craigdo/repos/ns-3-allinone/ns-3-dev/build'
'build' finished successfully (0.413s)
TcpL4Protocol:TcpStateMachine()
CommandLine:HandleArgument(): Handle arg name=PrintHelp value=
--PrintHelp: Print this help message.
--PrintGroups: Print the list of groups.
--PrintTypeIds: Print all TypeIds.
--PrintGroup=[group]: Print all TypeIds of group.
--PrintAttributes=[typeid]: Print all attributes of typeid.
--PrintGlobals: Print the list of globals.
```

Ας επικεντρωθούμε στην επιλογή `--PrintAttributes`. Έχουμε ήδη υπαινιχθεί για το σύστημα `Ορισμάτων` *ns-3* ενώ ακολουθούσαμε βήμα-βήμα το σενάριο `first.cc`. Αν κοιτάξουμε τις ακόλουθες γραμμές κώδικα,

```
PointToPointHelper pointToPoint;
pointToPoint.SetDeviceAttribute ("DataRate", StringValue ("5Mbps"));
pointToPoint.SetChannelAttribute ("Delay", StringValue ("2ms"));
```

παρατηρούμε ότι το `DataRate` είναι στην πραγματικότητα ένα `Όρισμα` του *PointToPointNetDevice*'. Ας χρησιμοποιήσουμε τον αναλυτή ορισμάτων γραμμής εντολών για να παρατήσουμε τα `Attributes` του PointToPointNetDevice. Η λίστα βοήθειας αναφέρει ότι πρέπει να παρέχουμε ένα `TypeId`. Αυτό αντιστοιχεί στο όνομα της κλάσης στην οποία ανήκουν τα `Attributes`. Σε αυτή την περίπτωση θα είναι

ns3::PointToPointNetDevice. Αν το τυπώσουμε,

```
$ ./waf --run "scratch/myfirst --PrintAttributes=ns3::PointToPointNetDevice"
```

Το σύστημα θα τυπώσει όλα τα Attributes αυτού του είδους συσκευών δικτύου. Μεταξύ των Attributes θα δείτε είναι και το ακόλουθο,

```
--ns3::PointToPointNetDevice::DataRate=[32768bps]:
  The default data rate for point to point links
```

Αυτή είναι η προεπιλεγμένη τιμή που θα χρησιμοποιηθεί όταν δημιουργείται στο σύστημα μία PointToPointNetDevice. Εμείς παρακάμψαμε αυτή την προεπιλογή με την ρύθμιση Attribute στο PointToPointHelper. Ας χρησιμοποιήσουμε τις προεπιλεγμένες τιμές για τις συσκευές point-to-point και τα κανάλια με τη διαγραφή των κλήσεων SetDeviceAttribute και SetChannelAttribute από το myfirst.cc στον κατάλογο scratch.

Το σενάριό σας πρέπει τώρα να δηλώσει το PointToPointHelper και να μην κάνει κάποια set ενέργεια όπως στο ακόλουθο παράδειγμα,

```
...

NodeContainer nodes;
nodes.Create (2);

PointToPointHelper pointToPoint;

NetDeviceContainer devices;
devices = pointToPoint.Install (nodes);

...
```

Ας οικοδομήσουμε το νέο σενάριο με Waf (./waf) επιτρέποντας κάποια καταγραφή από την εφαρμογή διακομιστή UDP echo και ενεργοποιώντας το πρόθεμα ώρας.

```
$ export 'NS_LOG=UdpEchoServerApplication=level_all|prefix_time'
```

Αν τρέξουμε το σενάριο, θα πρέπει να δούμε την ακόλουθη έξοδο,

```
Waf: Entering directory `/home/craigdo/repos/ns-3-allinone/ns-3-dev/build'
Waf: Leaving directory `/home/craigdo/repos/ns-3-allinone/ns-3-dev/build'
'build' finished successfully (0.405s)
0s UdpEchoServerApplication:UdpEchoServer()
1s UdpEchoServerApplication:StartApplication()
Sent 1024 bytes to 10.1.1.2
2.25732s Received 1024 bytes from 10.1.1.1
2.25732s Echoing packet
Received 1024 bytes from 10.1.1.2
10s UdpEchoServerApplication:StopApplication()
UdpEchoServerApplication:DoDispose()
UdpEchoServerApplication:~UdpEchoServer()
```

Θυμηθείτε ότι την τελευταία φορα που είδαμε το χρόνο εξομοίωσης όπου το πακέτο παρελήφθηκε από τον διακομιστή, ήταν στα 2.00369 δευτερόλεπτα.

```
2.00369s UdpEchoServerApplication:HandleRead(): Received 1024 bytes from 10.1.1.1
```

Τώρα λαμβάνει το πακέτο στα 2.25732 δευτερόλεπτα. Η αλαγή αυτή οφείλεται στη μείωση του ρυθμού μετάδοσης του PointToPointNetDevice από τα 5 megabits ανά δευτερόλεπτο στην προκαθορισμένη τιμή των 32768 bits ανά δευτερόλεπτο.

Αν παρείχαμε το νέο DataRate μέσω της γραμμής εντολών, θα μπορούσαμε να επιταχύνουμε την εξομοίωσή μας και πάλι. Αυτό γίνεται με τον ακόλουθο τρόπο,

```
$ ./waf --run "scratch/myfirst --ns3::PointToPointNetDevice::DataRate=5Mbps"
```

Αυτό θα ορίσει την προκαθορισμένη τιμή του `DataRate Attribute` πάλι σε 5 megabits ανά δευτερόλεπτο. Εκπλαγήκατε από το αποτέλεσμα; Φαίνεται ότι για να επαναφέρουμε την αρχική συμπεριφορά του σεναρίου, θα πρέπει να ρυθμίσουμε την καθυστέρηση του καναλιού στην ταχύτητα του φωτός. Μπορούμε να ζητήσουμε από το σύστημα γραμμής εντολών να εκτυπώσει τα `Attributes` του καναλιού, ακριβώς όπως κάναμε για την δικτυακή συσκευή:

```
$ ./waf --run "scratch/myfirst --PrintAttributes=ns3::PointToPointChannel"
```

Ανακαλύπτουμε ότι το `Delay Attribute` του καναλιού είναι ενεργοποιημένο με τον ακόλουθο τρόπο:

```
--ns3::PointToPointChannel::Delay=[0ns]:
  Transmission delay through the channel
```

Μπορούμε λοιπόν να θέσουμε και τις δύο προκαθορισμένες τιμές μέσω του συστήματος γραμμής εντολών,

```
$ ./waf --run "scratch/myfirst
  --ns3::PointToPointNetDevice::DataRate=5Mbps
  --ns3::PointToPointChannel::Delay=2ms"
```

όπου επαναφέρουμε τον χρονισμό που είχαμε όταν θέσαμε το `DataRate` και το `Delay` στο σενάριο:

```
Waf: Entering directory `/home/craigdo/repos/ns-3-allinone/ns-3-dev/build'
Waf: Leaving directory `/home/craigdo/repos/ns-3-allinone/ns-3-dev/build'
'build' finished successfully (0.417s)
0s UdpEchoServerApplication:UdpEchoServer()
1s UdpEchoServerApplication:StartApplication()
Sent 1024 bytes to 10.1.1.2
2.00369s Received 1024 bytes from 10.1.1.1
2.00369s Echoing packet
Received 1024 bytes from 10.1.1.2
10s UdpEchoServerApplication:StopApplication()
UdpEchoServerApplication:DoDispose()
UdpEchoServerApplication:~UdpEchoServer()
```

Σημειώστε ότι το πακέτο λαμβάνεται και πάλι από το διακομιστή στα 2.00369 δευτερόλεπτα. Στην ουσία θα μπορούσαμε να ορίσουμε με αυτόν τον τρόπο οποιαδήποτε από τα `Attributes` τα οποία χρησιμοποιούνται στο σενάριο. Ειδικότερα, θα μπορούσαμε να θέσουμε το `UdpEchoClient Attribute MaxPackets` σε κάποια διαφορετική τιμή από τη μονάδα.

Πώς θα το πραγματοποιούσατε αυτό; Κάντε μια δοκιμή. Να θυμάστε ότι πρέπει να σχολιάσετε το μέρος που αντικαθιστά το προεπιλεγμένο `Attribute` και ορίσετε ρητά το `MaxPackets` στο σενάριο. Στη συνέχεια θα πρέπει να ξαναοικοδομήσετε το σενάριο. Θα πρέπει επίσης να βρείτε τη σύνταξη για να ορίσετε τη νέα προεπιλεγμένη τιμή της ιδιότητας, χρησιμοποιώντας τη βοήθεια της γραμμής εντολών. Μόλις έχετε καταλάβει αυτό το βήμα, θα πρέπει να είστε σε θέση να ελέγχετε τον αριθμό των πακέτων που αντανακλώνται από τη γραμμή εντολών. Μιας που είμαστε καλά παιδιά, θα σας πούμε ότι η γραμμή εντολών σας θα πρέπει να μοιάζει κάπως έτσι,

```
$ ./waf --run "scratch/myfirst
  --ns3::PointToPointNetDevice::DataRate=5Mbps
  --ns3::PointToPointChannel::Delay=2ms
  --ns3::UdpEchoClient::MaxPackets=2"
```

### 5.2.2 Συνδέοντας τις δικές σας τιμές

Μπορείτε να προσθέσετε τις δικές σας συνδέσεις στο σύστημα γραμμής εντολών. Αυτό γίνεται με έναν απλό τρόπο, απλά χρησιμοποιώντας τη μέθοδο `AddValue` στον αναλυτή γραμμής εντολών.

Ας χρησιμοποιήσουμε αυτή τη λειτουργία για να ορίσουμε με έναν τελείως διαφορετικό τρόπο τον αριθμό των πακέτων που αντανακλώνται. Ας προσθέσουμε στη συνάρτηση `main` μία τοπική μεταβλητή με το όνομα `nPackets`. Θα την αρχικοποιήσουμε στην τιμή 1 για να ταυτιστεί με την προηγούμενη προκαθορισμένη τιμή. Για να επιτρέψουμε στον αναλυτή γραμμής εντολών να τροποποιήσει την τιμή αυτή, πρέπει να συνδέσουμε την τιμή στον αναλυτή. Αυτό το κάνουμε με την προσθήκη μιας κλήσης στην `AddValue`. Αλλάξτε το σενάριο `scratch/myfirst.cc` έτσι ώστε να αρχίζει με αυτόν τον κώδικα,

```
int
main (int argc, char *argv[])
{
  uint32_t nPackets = 1;

  CommandLine cmd;
  cmd.AddValue("nPackets", "Number of packets to echo", nPackets);
  cmd.Parse (argc, argv);

  ...
```

Κυλήστε το σενάριο προς τα κάτω μέχρι το σημείο όπου θέτουμε το όρισμα `MaxPackets` και αλλάξτε το έτσι ώστε να δείχνει στη μεταβλητή `nPackets` αντί να παίρνει την τιμή 1 όπως δείχνουμε παρακάτω,

```
echoClient.SetAttribute ("MaxPackets", UintegerValue (nPackets));
```

Τώρα αν τρέξετε το σενάριο και παρέχετε το όρισμα `--PrintHelp`, θα μπορείτε να δείτε στην οθόνη βοήθειας το νέο σας `User Argument`.

```
$ ./waf --run "scratch/myfirst --PrintHelp"

Waf: Entering directory `/home/craigdo/repos/ns-3-allinone/ns-3-dev/build'
Waf: Leaving directory `/home/craigdo/repos/ns-3-allinone/ns-3-dev/build'
'build' finished successfully (0.403s)
--PrintHelp: Print this help message.
--PrintGroups: Print the list of groups.
--PrintTypeIds: Print all TypeIds.
--PrintGroup=[group]: Print all TypeIds of group.
--PrintAttributes=[typeid]: Print all attributes of typeid.
--PrintGlobals: Print the list of globals.
User Arguments:
    --nPackets: Number of packets to echo
```

Αν θέλετε να καθορίσετε τον αριθμό των πακέτων που αντανακλώνται, μπορείτε να θέσετε το όρισμα `--nPackets` στην γραμμή εντολών,

```
$ ./waf --run "scratch/myfirst --nPackets=2"
```

Θα πρέπει να εμφανίζεται τώρα

```
Waf: Entering directory `/home/craigdo/repos/ns-3-allinone/ns-3-dev/build'
Waf: Leaving directory `/home/craigdo/repos/ns-3-allinone/ns-3-dev/build'
'build' finished successfully (0.404s)
0s UdpEchoServerApplication:UdpEchoServer()
1s UdpEchoServerApplication:StartApplication()
Sent 1024 bytes to 10.1.1.2
2.25732s Received 1024 bytes from 10.1.1.1
2.25732s Echoing packet
Received 1024 bytes from 10.1.1.2
Sent 1024 bytes to 10.1.1.2
3.25732s Received 1024 bytes from 10.1.1.1
3.25732s Echoing packet
Received 1024 bytes from 10.1.1.2
```

```
10s UdpEchoServerApplication:StopApplication()
UdpEchoServerApplication:DoDispose()
UdpEchoServerApplication:~UdpEchoServer()
```

Έχετε αντανακλάσει τώρα δύο πακέτα. Φαίνεται ιδιαίτερα εύκολο, έτσι δεν είναι;

Αν είστε ένας χρήστης *ns-3* λοιπόν, μπορείτε να χρησιμοποιείτε το σύστημα ορισμάτων γραμμής εντολών για να ελέγχετε τα `Attributes` και τις μεταβλητές συστήματος. Αν είστε ο συγγραφέας ενός μοντέλου, μπορείτε να προσθέτετε νέα `Attributes` στα `Objects` σας και αυτά θα είναι αυτόματα διαθέσιμα στους χρήστες σας για να θέσουν τιμές μέσω του συστήματος γραμμής εντολών. Αν είστε ο συγγραφέας ενός σεναρίου, μπορείτε να προσθέτετε νέες μεταβλητές στα σενάριά σας και να τις συνδέετε στο σύστημα γραμμής εντολών χωρίς ιδιαίτερο κόπο.

## 5.3 Χρησιμοποιώντας το Σύστημα Ιχνηλασίας

Το όλο νόημα της εξομοίωσης είναι να παράγουμε έξοδο για μελλοντικές μελέτες, και το σύστημα ιχνηλασίας του *ns-3* είναι ένας πρωταρχικός μηχανισμός για το σκοπό αυτό. Αφού το *ns-3* είναι ένα πρόγραμμα σε γλώσσα C++, μπορούν να χρησιμοποιηθούν τυποποιημένες λειτουργίες για τη παραγωγή εξόδου από προγράμματα C++.

```
#include <iostream>
...
int main ()
{
  ...
  std::cout << "The value of x is " << x << std::endl;
  ...
}
```

Θα μπορούσατε ακόμη και να χρησιμοποιήσετε τη μονάδα καταγραφής για να προσθέσετε κάποια δομή στη λύση σας. Υπάρχουν πολλά γνωστά προβλήματα που δημιουργούνται από τέτοιες προσεγγίσεις και έτσι παρέχουμε ένα υποσύστημα ιχνηλασίας γεγονότων για να αντιμετωπίσουμε τα θέματα που θεωρήσαμε ότι ήταν σημαντικά.

Οι βασικοί στόχοι του συστήματος ανίχνευσης του *ns-3* είναι:

- Για τα βασικά του καθήκοντα, το σύστημα ανίχνευσης θα πρέπει να επιτρέπει στο χρήστη να παράγει τυποποιημένη ιχνηλασία για δημοφιλείς πηγές ιχνηλασίας, και να προσαρμόζει ποια αντικείμενα δημιουργούν την ιχνηλασία;
- Οι μέσοι χρήστες θα πρέπει να είναι σε θέση να επεκτείνουν το σύστημα ιχνηλασίας ώστε να τροποποιούν τη μορφή εξόδου που παράγεται, ή να εισάγουν νέες πηγές ιχνηλασίας, χωρίς να αλλάζουν τον πυρήνα του προσομοιωτή;
- Οι προχωρημένοι χρήστες μπορούν να τροποποιήσουν τον πυρήνα προσομοιωτή ώστε να προσθέτουν νέες πηγές και καταβόθρες ιχνηλασίας.

Το σύστημα ιχνηλασίας *ns-3* είναι χτισμένο στις έννοιες των ανεξάρτητων πηγών και καταβοθρών ιχνηλασίας, και ενός ενιαίου μηχανισμού για τη σύνδεση πηγών σε καταβόθρες. Οι πηγές ίχνους είναι οντότητες οι οποίες μπορούν να σηματοδοτήσουν γεγονότα που συμβαίνουν σε μια εξομοίωση και παρέχουν πρόσβαση σε ενδιαφέροντα δεδομένα. Για παράδειγμα, μια πηγή ίχνους θα μπορούσε να υποδείξει πότε ένα πακέτο λαμβάνεται από μία δικτυακή συσκευή και να παρέχει πρόσβαση στα περιεχόμενα του πακέτου για τους ενδιαφερόμενες καταβόθρες ίχνους.

Οι πηγές ίχνους δεν είναι χρήσιμες από μόνες τους, θα πρέπει να είναι “συνδεδεμένες” με άλλα κομμάτια κώδικα που κάνουν πραγματικά κάτι χρήσιμο με τις πληροφορίες που παρέχονται από την καταβόθρα. Οι καταβόθρες ίχνους είναι οι καταναλωτές των γεγονότων και των δεδομένων που παρέχονται από τις πηγές ίχνους. Για παράδειγμα, κάποιος θα μπορούσε να δημιουργήσει μία καταβόθρα ίχνους που θα εκτύπωνε ενδιαφέροντα μέρη του ληφθέντος πακέτου (όταν συνδέεται με την πηγή ίχνους του προηγούμενου παραδείγματος).

Το σκεπτικό για αυτή την ρητή διαίρεση είναι να επιτρέψει στους χρήστες να επισυνάπτουν νέους τύπους καταβοθρών στις υπάρχουσες πηγές ιχνηλασίας, χωρίς να απαιτείται επεξεργασία και επαναμεταγλωτισμός του πυρήνα του εξομοιωτή. Έτσι, στο παραπάνω παράδειγμα, ο χρήστης μπορεί με την επεξεργασία μόνο του σεναρίου του χρήστη να καθορίσει μια νέα καταβόθρα ιχνηλασίας στο σενάριό του και να το επισυνάψει σε μια υπάρχουσα πηγή ιχνηλασίας που ορίζεται στον πυρήνα εξομοίωσης.

Σε αυτόν τον οδηγό, θα εξετάσουμε κάποιες προκαθορισμένες πηγές και καταβόθρες και θα δείξουμε πως μπορούν να προσαρμοστούν με μια μικρή προσπάθεια. Δείτε το εγχειρίδιο ns-3 ή τις ενότητες how-to για πληροφορίες σχετικά με τη διαμόρφωση προηγμένης ιχνηλασίας, συμπεριλαμβανομένων της επέκτασης του χώρου ονομάτων ιχνηλασίας και της δημιουργίας νέων πηγών ιχνηλασίας.

### 5.3.1 Ιχνηλασία Ascii

Το *ns-3* παρέχει λειτουργικότητα βοήθειας που συμπεριλαμβάνει το σύστημα ιχνηλασίας χαμηλού επιπέδου για να σας βοηθήσει με τις λεπτομέρειες που εμπλέκονται στη διαμόρφωση μερικών ευκατανόητων ιχνών πακέτων. Αν ενεργοποιήσετε αυτή τη λειτουργία, θα δείτε την εξόδο σε αρχεία ASCII — εξού και το όνομα. Για όσους είναι εξοικειωμένοι με την έξοδο του *ns-2*, το ίχνος αυτού του είδους είναι ανάλογο με το `out.tr` που παράγεται από πολλά σενάρια.

Ας πάμε κατευθείαν να προσθέσουμε κάποια έξοδο ιχνηλασίας ASCII στο σενάριό μας `scratch/myfirst.cc`. Ακριβώς πριν από την κλήση προς `Simulator::Run ()`, προσθέστε τις ακόλουθες γραμμές κώδικα:

```
AsciiTraceHelper ascii;
pointToPoint.EnableAsciiAll (ascii.CreateFileStream ("myfirst.tr"));
```

Όπως και σε πολλά άλλα ιδιώματα *ns-3*, αυτός ο κώδικας χρησιμοποιεί ένα βοηθητικό αντικείμενο για να δημιουργήσει ίχνη ASCII. Η δεύτερη γραμμή περιέχει δύο ένθετες κλήσεις μεθόδων. Η "εσωτερική" μέθοδος, `CreateFileStream()` χρησιμοποιεί ένα ανώνυμο ιδίωμα αντικειμένου για να δημιουργήσει ένα αντικείμενο ρεύματος αρχείου στη στοίβα (χωρίς όνομα αντικειμένου) και να το περάσει στην καλούμενη μέθοδο. Θα επιστρέψουμε στο σημείο αυτό αργότερα, αλλά αυτό που πρέπει να ξέρετε στο σημείο αυτό είναι ότι δημιουργείτε ένα αντικείμενο που αντιπροσωπεύει ένα αρχείο με το όνομα "myfirst.tr" και το διαβιβάζετε στο `ns-3`. Λέτε στο `ns-3` να ασχοληθεί με τα θέματα χρόνου ζωής του δημιουργούμενου αντικειμένου και επίσης να χειριστεί τα προβλήματα του δημιούργησε ένας ελάχιστα γνωστός περιορισμός της C++ για ofstream αντικείμενα που σχετίζονται με την αντιγραφή των κατασκευαστών.

Η εξωτερική κλήση προς την `EnableAsciiAll()`, λέει στον βοηθό ότι θέλετε να ενεργοποιήσετε την ιχνηλασία ASCII σε όλες τις point-to-point συσκευές της εξομοίωσής σας. Και θέλετε το παρεχόμενο ίχνος καταβόθρας να γράψει πληροφορίες σε μορφή ASCII σχετικά με την κίνηση πακέτων.

Για όσους είναι εξοικειωμένοι με το *ns-2*, τα ιχνηλατημένα γεγονότα είναι ισοδύναμα με τα δημοφιλή σημεία ίχνους που καταγράφουν "+", "-", "d", και "r" γεγονότα.

Μπορείτε τώρα να χτίσετε το σενάριο και να το εκτελέσετε από τη γραμμή εντολών:

```
$ ./waf --run scratch/myfirst
```

Ακριβώς όπως έχετε ήδη δει πολλές φορές, θα δείτε κάποια μηνύματα από το Waf και στη συνέχεια το μήνυμα "'build' finished successfully" με κάποιο αριθμό μηνυμάτων από το πρόγραμμα εκτέλεσης.

Κατά την εκτέλεση το πρόγραμμα θα έχει δημιουργήσει ένα αρχείο με το όνομα `myfirst.tr`. Εξαιτίας του τρόπου με τον οποίο λειτουργεί το Waf, το αρχείο δεν έχει δημιουργηθεί στο τοπικό κατάλογο, αλλά στον προκαθορισμένο κατάλογο ανώτατου επιπέδου του αποθέματος. Αν θέλετε να ελέγχετε που αποθηκεύονται τα ίχνη, μπορείτε να χρησιμοποιήσετε την επιλογή `--cwd` του Waf για να το καθορίσετε. Εμείς δεν το κάναμε, έτσι πρέπει να αλλάξουμε κατάλογο και να πάμε στον αρχικό κατάλογο του αποθέματος και να ανοίξουμε το αρχείο ίχνους ASCII `myfirst.tr` με τον αγαπημένο σας επεξεργαστή κειμένου.

### Αναλύοντας Ίχνη Ascii

Στο αρχείο αυτό υπάρχει ένα μεγάλο πλήθος πληροφοριών σε μια αρκετά πυκνή μορφή, αλλά το πρώτο πράγμα που μπορείτε να παρατηρήσετε είναι ότι υπάρχει ένας πλήθος από ξεχωριστές γραμμές. Ίσως είναι δύσκολο να το δείτε ξεκάθαρα αν δεν διευρύνει το μέγεθος του παραθύρου σας σημαντικά.

Κάθε γραμμή στο αρχείο αντιστοιχεί σε ένα *ίχνος γεγονότος*. Σε αυτήν την περίπτωση εντοπίζουμε τα γεγονότα ιχνηλασίας στην *ουρά εκπομπής* που βρίσκεται σε κάθε δικτυακή συσκευή point-to-point στην εξομοίωση. Η ουρά εκπομπής είναι μια ουρά μέσω της οποίας πρέπει να περάσει κάθε πακέτο που προορίζεται για ένα κανάλι point-to-point. Σημειώστε ότι κάθε γραμμή στο αρχείο παρακολούθησης αρχίζει με ένα μοναχικό χαρακτήρα (έχει έναν κενό χαρακτήρα αμέσως μετά). Αυτός ο χαρακτήρας έχει την ακόλουθη έννοια:

- `+`: Μια λειτουργία τοποθέτησης στην ουρά συνέβη στην ουρά συσκευής;
- `-`: Μια λειτουργία απομάκρυνσης από την ουρά συνέβη στην ουρά συσκευής;
- `d`: Ένα πακέτο απορρίφθηκε, συνήθως επειδή η ουρά ήταν πλήρης;
- `r`: Ένα πακέτο παρελήφθη από την δικτυακή συσκευή.

Ας ρίξουμε μια πιο λεπτομερή ματιά στην πρώτη γραμμή του αρχείου παρακολούθησης. Θα την τμηματοποιήσουμε (τοποθετώντας εσοχές για λόγους σαφήνειας) με αριθμός αναφοράς στην αριστερή πλευρά:

```
1  +
2  2
3  /NodeList/0/DeviceList/0/$ns3::PointToPointNetDevice/TxQueue/Enqueue
4  ns3::PppHeader (
5    Point-to-Point Protocol: IP (0x0021))
6    ns3::Ipv4Header (
7      tos 0x0 ttl 64 id 0 protocol 17 offset 0 flags [none]
8      length: 1052 10.1.1.1 > 10.1.1.2)
9      ns3::UdpHeader (
10       length: 1032 49153 > 9)
11       Payload (size=1024)
```

Το πρώτο τμήμα αυτού του διευρυμένου γεγονότος ίχνους (αριθμός αναφοράς 0) είναι η λειτουργία. Έχουμε ένα χαρακτήρα +, οπότε αυτό αντιστοιχεί σε μια λειτουργία *τοποθέτησης στην ουρά* στην ουρά εκπομπής. Το δεύτερο τμήμα (αναφορά 1) είναι ο χρόνος εξομοίωσης που εκφράζεται σε δευτερόλεπτα. Ίσως να θυμάστε ότι ζητήσαμε από το `UdpEchoClientApplication` να ξεκινήσετε την αποστολή πακέτων στα δύο δευτερόλεπτα. Εδώ βλέπουμε την επιβεβαίωση ότι αυτό πράγματι συμβαίνει.

Το επόμενο τμήμα του ίχνους του παραδείγματος (αναφορά 2) μας δείχνει από ποια πηγή ίχνους προήλθε αυτό το γεγονός (εκφράζεται στο χώρο ονομάτων εντοπισμού). Μπορείτε να σκεφτείτε ότι ο χώρος ονομάτων του εντοπισμού είναι παρόμοιος με τον χώρο ονομάτων αρχείων. Η ρίζα του χώρου ονομάτων είναι η `NodeList`. Αυτό αντιστοιχεί σε ένα δοχείο διαχειρίζεται το | NS3 | κωδικός πυρήνα που περιέχει το σύνολο των κόμβων που είναι δημιουργήθηκε σε ένα σενάριο. Ακριβώς όπως ένα σύστημα αρχείων μπορεί να έχει καταλόγους κάτω από το ρίζα, μπορεί να έχουμε τους αριθμούς κόμβου στο `NodeList`. Το κορδόνι " / NodeList / 0" αναφέρεται, επομένως, στον κόμβο μηδενικής στην `NodeList` οποία συνήθως σκεφτόμαστε ως «κόμβος 0". Σε κάθε κόμβο υπάρχει μια λίστα συσκευές που έχουν εγκατασταθεί. Αυτή η λίστα εμφανίζεται δίπλα στο χώρο ονομάτων. Μπορείτε να δείτε ότι αυτό το γεγονός ίχνος προέρχεται από `DeviceList/0` η οποία είναι η συσκευή μηδενικής εγκατεστημένο στον κόμβο.

Το επόμενο αλφαριθμητικό, `$ns3::PointToPointNetDevice` σας λέει τι είδους συσκευή είναι στη μηδενική θέση στη λίστα συσκευών για τον κόμβο μηδέν. Θυμηθείτε ότι η λειτουργία + στην αναφορά 00 σημαίνει ότι μια λειτουργία τοποθέτησης στην ουρά συνέβη στην ουρά μεταδόσεως της συσκευής. Αυτό αντικατοπτρίζεται στα τελικά τμήματα της "διαδρομής ίχνους" τα οποίο είναι `TxQueue/Enqueue`.

Τα υπόλοιπα τμήματα στο ίχνος πρέπει να είναι αρκετά έξυπνα. Οι αναφορές 3-4 υποδεικνύουν ότι το πακέτο είναι εμφωλιασμένο στο πρωτόκολλο point-to-point. Οι αναφορές 5-7 δείχνουν ότι το πακέτο έχει μια επικεφαλίδα IPv4 και προήλθε από τη διεύθυνση IP 10.1.1.1 και έχει προορισμό την 10.1.1.2. Οι αναφορές 8-9 δείχνουν ότι

αυτό το πακέτο έχει μια επικεφαλίδα UDP και, τέλος, η αναφορά 10 δείχνει ότι το ωφέλιμο φορτίο είναι τα αναμενόμενα 1024 bytes.

Η επόμενη γραμμή στο αρχείο ίχνος δείχνει το ίδιο πακέτο που απομακρύνεται από την ουρά μετάδοσης στον ίδιο κόμβο.

Η τρίτη γραμμή στο αρχείο ίχνος δείχνει το πακέτο που λήφθηκε από τη δικτυακή συσκευή μέσω της αντήχησης του εξυπηρετητή. Αναπαράγουμε αυτό το συμβάν παρακάτω.

```
r
2.25732
/NodeList/1/DeviceList/0/$ns3::PointToPointNetDevice/MacRx
  ns3::Ipv4Header (
    tos 0x0 ttl 64 id 0 protocol 17 offset 0 flags [none]
    length: 1052 10.1.1.1 > 10.1.1.2)
    ns3::UdpHeader (
      length: 1032 49153 > 9)
      Payload (size=1024)
```

Παρατηρήστε ότι η λειτουργία ανίχνευσης είναι πλέον `r` και ο χρόνος εξομοίωσης έχει αυξηθεί σε 2.25732 δευτερόλεπτα. Αν έχετε ακολουθήσει τα βήματα του οδηγού προσεκτικά, αυτό σημαίνει ότι έχετε αφήσει το `DataRate` των δικτυακών συσκευών και το κανάλι `Delay` στις προεπιλεγμένες τιμές τους. Αυτή τη φορά θα πρέπει να είστε εξοικειωμένοι μια που το έχετε ξαναδεί σε προηγούμενη ενότητα.

Η είσοδος χώρος ονομάτων της πηγής ίχνους (αναφορά 02) έχει αλλάξει για να επισημάνει ότι το γεγονός έρχεται από τον κόμβο 1 (`/NodeList/1`) και η λήψη πακέτου της πηγή ίχνους (`/MacRx`). Θα πρέπει να είναι αρκετά εύκολο για σας να ακολουθήσετε την πρόοδο του πακέτου μέσω της τοπολογίας κοιτάζοντας τα υπόλοιπα ίχνη του αρχείου.

Οι βοηθοί συσκευών *ns-3* μπορούν επίσης να χρησιμοποιηθούν για τη δημιουργία αρχείων ίχνους σε μορφή `.pcap`. Το αρκτικόλεξο pcap αντιστοιχεί στη σύλληψη πακέτων (packet capture) και συνήθως γράφεται με μικρά γράμματα. Είναι στην πραγματικότητα μια διεπαφή προγράμματος που περιλαμβάνει τον ορισμό του είδους αρχείου `.pcap`. Το πιο δημοφιλές πρόγραμμα που μπορεί να εμφανίσει αυτό το είδος αρχείου είναι το Wireshark (παλαιότερα ονομαζόταν Ethereal). Ωστόσο, υπάρχουν πολλοί αναλυτές ίχνους κίνησης που χρησιμοποιούν αυτή τη μορφή πακέτων. Ενθαρρύνουμε τους χρήστες να εκμεταλλευτούν τα πολλά διαθέσιμα εργαλεία για την ανάλυση ιχνών pcap. Σε αυτόν τον οδηγό, θα επικεντρωθούμε στην προβολή ιχνών pcap με το tcpdump.

Ο κωδικός που χρησιμοποιούμε για να ενεργοποιήσουμε την ιχνηλασία pcap είναι μιας γραμμής.

```
pointToPoint.EnablePcapAll ("myfirst");
```

Εισάγετε αυτή τη γραμμή του κώδικα μετά τον κωδικό ιχνηλασίας ASCII που μόλις προσθέσατε στο `scratch/myfirst.cc`. Παρατηρήστε ότι έχουμε περάσει μόνο το αλφαριθμητικό "myfirst," και όχι "myfirst.pcap" ή κάτι παρόμοιο. Αυτό συμβαίνει επειδή η παράμετρος είναι ένα πρόθεμα, δεν είναι ένα πλήρες όνομα του αρχείου. Ο βοηθός στην ουσία θα δημιουργήσει ένα αρχείο ίχνους για κάθε συσκευή point-to-point στην εξομοίωση. Τα ονόματα των αρχείων θα χτιστούν χρησιμοποιώντας το πρόθεμα, τον αριθμό κόμβου, τον αριθμό της συσκευής και μια κατάληξη ".pcap".

Στο σενάριο του παραδείγματός μας, θα δούμε τελικά αρχεία με όνομα "myfirst-0-0.pcap" και "myfirst-1-0.pcap" που είναι τα ίχνη pcap για τον κόμβο 0-συσκευή 0 και κόμβο 1-συσκευή 0, αντίστοιχα.

Μόλις έχετε προσθέσει τη γραμμή του κώδικα για να ενεργοποιήσετε την ιχνηλασία pcap, μπορείτε να εκτελέσετε το σενάριο με το συνήθη τρόπο:

```
$ ./waf --run scratch/myfirst
```

Αν κοιτάξετε στον κατάλογο κορυφής της διανομής σας, θα πρέπει τώρα να βλέπετε τρία αρχεία καταγραφής: `myfirst.tr` είναι το αρχείο ίχνους ASCII που έχουμε εξετάσει προηγουμένως. Τα `myfirst-0-0.pcap` και `myfirst-1-0.pcap` είναι τα νέα αρχεία pcap που μόλις δημιουργήσαμε.

### Ανάγνωση εξόδου με tcpdump

Το πιο εύκολο βήμα που μπορούμε κάνουμε σε αυτό το σημείο θα είναι να χρησιμοποιήσουμε το `tcpdump` να δούμε τα `pcap` αρχεία.

```
$ tcpdump -nn -tt -r myfirst-0-0.pcap
reading from file myfirst-0-0.pcap, link-type PPP (PPP)
2.000000 IP 10.1.1.1.49153 > 10.1.1.2.9: UDP, length 1024
2.514648 IP 10.1.1.2.9 > 10.1.1.1.49153: UDP, length 1024

tcpdump -nn -tt -r myfirst-1-0.pcap
reading from file myfirst-1-0.pcap, link-type PPP (PPP)
2.257324 IP 10.1.1.1.49153 > 10.1.1.2.9: UDP, length 1024
2.257324 IP 10.1.1.2.9 > 10.1.1.1.49153: UDP, length 1024
```

Μπορείτε να δείτε στο dump του αρχείου `myfirst-0-0.pcap` (η συσκευή του πελάτη) ότι η το πακέτο αντανάκλασης στέλνεται στα 2 δευτερόλεπτα στην εξομοίωση. Αν κοιτάξετε το δεύτερο dump (`myfirst-1-0.pcap`) μπορείτε να δείτε ότι το πακέτο λαμβάνεται σε 2.257324 δευτερόλεπτα. Μπορείτε να δείτε το πακέτο που αντανακλάται πίσω σε 2.257324 δευτερόλεπτα στο δεύτερο dump, και, τέλος, μπορείτε να δείτε το πακέτο που παραλαμβάνεται πίσω στον πελάτη στο πρώτο dump σε 2.514648 δευτερόλεπτα.

### Ανάγνωση εξόδου με το Wireshark

Εάν δεν είστε εξοικειωμένοι με το Wireshark, υπάρχει μια ιστοσελίδα από την οποία μπορείτε να κατεβάσετε τα προγράμματα και την τεκμηρίωση: http://www.wireshark.org/.

Το Wireshark είναι ένα γραφικό περιβάλλον χρήστη, το οποίο μπορεί να χρησιμοποιηθεί για την εμφάνιση αυτών των αρχείων ίχνους. Εάν έχετε διαθέσιμο το Wireshark, μπορείτε να ανοίξετε κάθε αρχείο ίχνους και να εμφανίσετε τα περιεχόμενά του σαν να είχαν συλληφθεί τα πακέτα χρησιμοποιώντας έναν οσφρητή πακέτων (*packet sniffer*).

ΚΕΦΑΛΑΙΟ

ΕΞΙ

# ΔΗΜΙΟΥΡΓΙΑ ΤΟΠΟΛΟΓΙΩΝ

## 6.1 Δημιουργώντας μια Τοπολογία Δικτύου Αρτηρίας

Σε αυτή την ενότητα πρόκειται να επεκτείνουμε την ικανότητά μας να διαχειριζόμαστε τις συσκευές δικτύου του *ns-3* και τα κανάλια, ώστε να καλύψουμε ένα παράδειγμα ενός δικτύου αρτηρίας. Ο *ns-3* παρέχει μια δικτυακή συσκευή και ένα κανάλι που εμείς αποκαλούμε CSMA (Carrier Sense Multiple Access).

Η CSMA συσκευή του *ns-3* μοντελοποιεί ένα απλό δίκτυο στο πνεύμα του Ethernet. Ένα πραγματικό Ethernet χρησιμοποιεί το CSMA/CD (Carrier Sense Multiple Access with Collision Detection) σχήμα με εκθετικά αυξανόμενη οπισθοχώρηση για τη διεκδίκηση του διαμοιραζόμενου μέσου μετάδοσης. Η CSMA συσκευή του *ns-3* και το αντίστοιχο κανάλι μοντελοποιούν μόνο ένα υποσύνολο από αυτά.

Όπως έχουμε δει αντικείμενα βοηθών για τοπολογίες σημείου-προς-σημείο όταν κατασκευάζαμε τέτοιες τοπολογίες, σε αυτή την ενότητα θα δούμε ισοδύναμους CSMA βοηθούς τοπολογίας. Η εμφάνιση και η λειτουργία αυτών των βοηθών θα πρέπει να σας φαίνεται αρκετά οικεία.

Παραθέτουμε ένα παραδειγματικό σενάριο στον κατάλογο examples/tutorial. Το σενάριο αυτό «πατάει» πάνω στο σενάριο `first.cc` και προσθέτει ένα δίκτυο CSMA στην προσομοίωση σημείου-προς-σημείο που έχουμε ήδη εξετάσει. Ανοίξτε το `examples/tutorial/second.cc` στον επεξεργαστή κειμένου της προτίμησής σας. Θα πρέπει να έχετε ήδη δει αρκετό κώδικα του *ns-3* ώστε να μπορείτε να καταλάβετε τα περισσότερα από όσα συμβαίνουν σε αυτό το παράδειγμα, αλλά εμείς θα δούμε ολόκληρο το σενάριο και θα εξετάσουμε κάποια από τα αποτελέσματα.

Όπως και στο παράδειγμα του `first.cc` (και σε όλα τα παραδείγματα του ns-3), το αρχείο ξεκινάει με μια γραμμή κατάστασης για τον emacs και κάποιες κοινές δηλώσεις GPL.

Ο πραγματικός κώδικας ξεκινάει με τη φόρτωση αρχείων συμπερίληψης ενοτήτων (module includes), όπως έγινε και στο παράδειγμα `first.cc`.

```
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/csma-module.h"
#include "ns3/internet-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/applications-module.h"
#include "ns3/ipv4-global-routing-helper.h"
```

Ένα πράγμα που μπορεί να είναι εκπληκτικά χρήσιμο είναι ένα μικρό κομμάτι τέχνης σε ASCII, το οποίο δείχνει ένα «σχέδιο» της τοπολογίας δικτύου που κατασκευάζεται στο παράδειγμα. Θα βρείτε ένα παρόμοιο «σχέδιο» στα περισσότερα από τα παραδείγματά μας.

Σε αυτήν την περίπτωση, μπορείτε να δείτε ότι πρόκειται να επεκτείνουμε το παράδειγμά μας από σημείο-προς-σημείο (ο σύνδεσμος μεταξύ των κόμβων n0 και n1 παρακάτω) συνδέοντας ένα δίκτυο αρτηρίας στη δεξιά του πλευρά. Παρατηρήστε ότι αυτή είναι η προεπιλεγμένη τοπολογία δικτύου καθώς μπορείτε ουσιαστικά να αλλάξετε τον αριθμό των κόμβων που δημιουργούνται στο LAN. Εάν θέσετε τη nCsma στην τιμή ένα, θα υπάρχουν

συνολικά δύο κόμβοι στο LAN (CSMA κανάλι) — ένας απαιτούμενος κόμβος και ένας «επιπλέον» κόμβος. Εξ ορισμού υπάρχουν τρεις «επιπλέον» κόμβοι, όπως φαίνεται παρακάτω:

```
// Default Network Topology
//
//       10.1.1.0
// n0 -------------- n1   n2   n3   n4
//    point-to-point  |    |    |    |
//                    ================
//                      LAN 10.1.2.0
```

Έπειτα χρησιμοποιείται (`used`) ο χώρος ονομάτων του ns-3 και ορίζεται ένα στοιχείο καταγραφής. Όλα αυτά είναι όπως ήταν και στο `first.cc`, οπότε δεν υπάρχει κάτι καινούργιο ακόμα.

```
using namespace ns3;

NS_LOG_COMPONENT_DEFINE ("SecondScriptExample");
```

Το κυρίως πρόγραμμα ξεκινά με λίγο διαφορετική εξέλιξη. Χρησιμοποιούμε μια σημαία verbose για να καθορίσουμε εάν θα ενεργοποιηθούν τα στοιχεία καταγραφής των `UdpEchoClientApplication` και `UdpEchoServerApplication`. Αυτή η σημαία τίθεται εξ ορισμού ως αληθής (τα στοιχεία καταγραφής είναι ενεργοποιημένα) αλλά μας επιτρέπει να απενεργοποιήσουμε την καταγραφή κατά τη διάρκεια της προς-τα-πίσω δοκιμής αυτού του παραδείγματος.

Θα δείτε μερικό οικείο κώδικα, ο οποίος θα σας επιτρέψει να αλλάξετε τον αριθμό των συσκευών στο CSMA δίκτυο μέσω ορισμάτων στη γραμμή εντολών. Κάναμε κάτι παρόμοιο όταν επιτρέψαμε να αλλάξει ο αριθμός των πακέτων που στέλνονται στην ενότητα με τα ορίσματα της γραμμής εντολών. Η τελευταία γραμμή διασφαλίζει ότι έχετε τουλάχιστον έναν «επιπλέον» κόμβο.

Ο κώδικας αποτελείται από παραλλαγές του API που έχουμε εξετάσει πιο πριν, οπότε θα πρέπει να είστε πλήρως άνετοι με τον ακόλουθο κώδικα σε αυτό το σημείο του οδηγού.

```
bool verbose = true;
uint32_t nCsma = 3;

CommandLine cmd;
cmd.AddValue ("nCsma", "Number of \"extra\" CSMA nodes/devices", nCsma);
cmd.AddValue ("verbose", "Tell echo applications to log if true", verbose);

cmd.Parse (argc, argv);

if (verbose)
  {
    LogComponentEnable("UdpEchoClientApplication", LOG_LEVEL_INFO);
    LogComponentEnable("UdpEchoServerApplication", LOG_LEVEL_INFO);
  }

nCsma = nCsma == 0 ? 1 : nCsma;
```

Το επόμενο βήμα είναι η δημιουργία δύο κόμβων, τους οποίους θα συνδέσουμε μέσω ενός συνδέσμου σημείου-προς-σημείο. Χρησιμοποιείται ο `NodeContainer` για να το κάνει αυτό, ακριβώς όπως το έκανε και στο `first.cc`.

```
NodeContainer p2pNodes;
p2pNodes.Create (2);
```

Έπειτα, δηλώνουμε άλλον ένα `NodeContainer`, ο οποίος θα περιέχει τους κόμβους που θα είναι μέρος του δικτύου αρτηρίας (CSMA). Αρχικά, πρέπει να δημιουργήσουμε το αντικείμενο container αυτό καθαυτό.

```
NodeContainer csmaNodes;
csmaNodes.Add (p2pNodes.Get (1));
csmaNodes.Create (nCsma);
```

Η επόμενη γραμμή κώδικα παίρνει (Gets) τον πρώτο κόμβο (δηλαδή σα να έχει ένα ευρετήριο που να περιέχει έναν κόμβο) από τον container κόμβων σημείου-προς-σημείο και τον προσθέτει στον container των κόμβων που θα δεχτούν τις CSMA συσκευές. Ο εν λόγω κόμβος πρόκειται να καταλήξει με μια συσκευή σημείου-προς-σημείο *και* μια CSMA συσκευή. Έπειτα δημιουργούμε έναν αριθμό από «επιπλέον» κόμβους που συνθέτουν το υπόλοιπο του CSMA δικτύου. Δεδομένου ότι έχουμε ήδη έναν κόμβο στο CSMA δίκτυο – εκείνον που θα έχει και μια δικτυακή συσκευή σημείου-προς-σημείο και μια CSMA, ο αριθμός των «επιπλέον» κόμβων ισούται με τον αριθμό των κόμβων που επιθυμείτε στο κομμάτι του CSMA πλην ενός.

Το επόμενο κομμάτι κώδικα θα πρέπει να σας είναι ήδη πολύ οικείο. Δημιουργούμε έναν PointToPointHelper και καθορίζουμε τα σχετικά προεπιλεγμένα Attributes, έτσι ώστε να δημιουργήσουμε έναν πομπό με ταχύτητα μετάδοσης πέντε megabit ανά δευτερόλεπτο στις συσκευές που δημιουργήθηκαν με τη χρήση του βοηθού και μια καθυστέρηση δύο μιλιδευτερολέπτων στα κανάλια που δημιουργήθηκαν από τον βοηθό.

```
PointToPointHelper pointToPoint;
pointToPoint.SetDeviceAttribute ("DataRate", StringValue ("5Mbps"));
pointToPoint.SetChannelAttribute ("Delay", StringValue ("2ms"));

NetDeviceContainer p2pDevices;
p2pDevices = pointToPoint.Install (p2pNodes);
```

Έπειτα δημιουργούμε ένα NetDeviceContainer για να καταγράφει τις δικτυακές συσκευές σημείου-προς-σημείο και εγκαθιστούμε (Install) τις συσκευές στους κόμβους σημείου-προς-σημείο.

Αναφέραμε προηγουμένως πως πρόκειται να δείτε έναν βοηθό για CSMA συσκευές και κανάλια, και οι επόμενες γραμμές εισάγουν αυτά τα στοιχεία. Ο CsmaHelper λειτουργεί όπως ένας PointToPointHelper, μόνο που δημιουργεί και συνδέει CSMA συσκευές και κανάλια. Στην περίπτωση ενός ζεύγους CSMA συσκευής και καναλιού, παρατηρήστε ότι ο ρυθμός δεδομένων καθορίζεται μέσω ενός Attribute του *καναλιού* αντί για κάποιο Attribute της συσκευής. Αυτό συμβαίνει επειδή ένα πραγματικό CSMA δίκτυο δεν επιτρέπει την ανάμειξη, για παράδειγμα, συσκευών 10Base-T και 100Base-T σε ένα δεδομένο κανάλι. Πρώτα θέτουμε το ρυθμό δεδομένων στα 100 megabit ανά δευτερόλεπτο, και μετά θέτουμε την καθυστέρηση του καναλιού στα 6560 νανοδευτερόλεπτα (έχοντας επιλέξει αυθαίρετα ότι χρειάζεται 1 νανοδευτερόλεπτο ανά πόδι αντί ανά τμήματα 100 μέτρων). Σημειώστε ότι μπορείτε να καθορίσετε ένα Attribute χρησιμοποιώντας τον ενδογενή τύπο δεδομένων του.

```
CsmaHelper csma;
csma.SetChannelAttribute ("DataRate", StringValue ("100Mbps"));
csma.SetChannelAttribute ("Delay", TimeValue (NanoSeconds (6560)));

NetDeviceContainer csmaDevices;
csmaDevices = csma.Install (csmaNodes);
```

Με τον ίδιο τρόπο που δημιουργήσαμε ένα NetDeviceContainer ώστε να περιέχει τις συσκευές που δημιουργούνται από τον PointToPointHelper, έτσι δημιουργούμε ένα NetDeviceContainer για να περιέχει τις συσκευές που δημιουργούνται από τον CsmaHelper μας. Καλούμε τη μέθοδο Install του CsmaHelper ώστε να εγκαταστήσουμε τις συσκευές στους κόμβους του csmaNodes NodeContainer.

Τώρα έχουμε δημιουργήσει τους κόμβους μας, τις συσκευές και τα κανάλια μας, αλλά δεν έχουμε καμία εγκατεστημένη στοίβα πρωτοκόλλου. Όπως και στο σενάριο του first.cc, θα χρησιμοποιήσουμε τον InternetStackHelper για να εγκαταστησουμε αυτές τις στοίβες.

```
InternetStackHelper stack;
stack.Install (p2pNodes.Get (0));
stack.Install (csmaNodes);
```

Θυμηθείτε ότι πήραμε έναν από τους κόμβους από τον container p2pNodes και τον προσθέσαμε στον container

csmaNodes. Έτσι χρειάζεται μόνο να εγκαταστήσουμε τις στοίβες στον εναπομείναντα κόμβου του p2pNodes, και σε όλους τους κόμβους του container csmaNodes, ώστε να καλύψουμε όλους τους κόμβους της προσομοίωσης.

Όπως και στο παράδειγμα του first.cc, πρόκειται να χρησιμοποιήσουμε τον Ipv4AddressHelper για την ανάθεση των IP διευθύνσεων στις διεπαφές των συσκευών μας. Αρχικά χρησιμοποιούμε το δίκτυο 10.1.1.0 για να δημιουργήσουμε τις δύο διευθύνσεις που χρειάζονται για τις δύο σημείο-προς-σημείο συσκευές μας.

```
Ipv4AddressHelper address;
address.SetBase ("10.1.1.0", "255.255.255.0");
Ipv4InterfaceContainer p2pInterfaces;
p2pInterfaces = address.Assign (p2pDevices);
```

Θυμηθείτε ότι αποθηκεύουμε τις δημιουργημένες διεπαφές σε ένα container ώστε να κάνουμε πιο εύκολη την ανάκτηση πληροφορίας διευθυνσιοδότησης αργότερα, για χρήση κατά το στήσιμο των εφαρμογών.

Τώρα πρέπει να αναθέσουμε IP διευθύνσεις στις διεπαφές των CSMA συσκευών μας. Η λειτουργία πραγματοποιείται όπως και στην περίπτωση σημείου-προς-σημείο, με τη διαφορά ότι τώρα πραγματοποιούμε τη λειτουργία σε ένα container που έχει ένα μεταβλητό αριθμό από CSMA συσκευές — θυμηθείτε ότι κάναμε τον αριθμό των CSMA συσκευών μεταβλητό μέσω ορίσματος στη γραμμή εντολών. Οι CSMA συσκευές θα συσχετιστούν σε αυτήν την περίπτωση με τις IP διευθύνσεις από τη διεύθυνση δικτύου 10.1.2.0, όπως βλέπετε παρακάτω.

```
address.SetBase ("10.1.2.0", "255.255.255.0");
Ipv4InterfaceContainer csmaInterfaces;
csmaInterfaces = address.Assign (csmaDevices);
```

Τώρα έχουμε δημιουργήσει μια τοπολογία, αλλά χρειαζόμαστε εφαρμογές. Αυτή η ενότητα θα είναι κατά βάση παρόμοια με το τμήμα των εφαρμογών στο first.cc, αλλά εδώ θα εκκινήσουμε τον εξυπηρετητή σε έναν από τους κόμβους ο οποίος έχει μια CSMA συσκευή , και τον πελάτη στον κόμβο που έχει μόνο μία συσκευή σημείου-προς-σημείο.

Αρχικά, στήνουμε έναν εξυπηρετητή echo. Δημιουργούμε έναν UdpEchoServerHelper και παρέχουμε μια απαιτούμενη τιμή για Attribute στον δημιουργό, η οποία είναι ο αριθμός port του εξυπηρετητή. Θυμηθείτε ότι αυτό το port μπορεί να αλλάξει αργότερα, εάν το επιθυμείτε, με τη χρήση της μεθόδου SetAttribute, αλλά εμείς απαιτούμε να δίνεται ως όρισμα στον δημιουργό.

```
UdpEchoServerHelper echoServer (9);

ApplicationContainer serverApps = echoServer.Install (csmaNodes.Get (nCsma));
serverApps.Start (Seconds (1.0));
serverApps.Stop (Seconds (10.0));
```

Θυμηθείτε ότι ο csmaNodes NodeContainer περιέχει έναν από τους κόμβους που δημιουργήθηκαν στο δίκτυο σημείου-προς-σημείο και nCsma «επιπλέον» κόμβους. Εκεί που θέλουμε να φτάσουμε είναι στον τελευταίο από τους "επιπλέον" κόμβους. Οπότε, ο εύκολος τρόπος για να το σκεφτούμε είναι πως εάν δημιουργήσουμε έναν «επιπλέον» CSMA κόμβο, τότε αυτός θα βρίσκεται στην πρώτη θέση του container csmaNodes. Επαγωγικά, εάν δημιουργήσουμε nCsma «επιπλέον» κόμβους, ο τελευταίος θα είναι στη θέση nCsma. Μπορείτε να το δείτε αυτό στο Get της πρώτης γραμμής του κώδικα.

Η εφαρμογή του πελάτη εγκαθίσταται ακριβώς όπως και στο σενάριο του first.cc. Ξανά, παρέχουμε τα απαιτούμενα Attributes στον UdpEchoClientHelper μέσα στον δημιουργό (σε αυτή την περίπτωση την απομακρυσμένη διεύθυνση και το port). Λέμε στον πελάτη να στέλνει πακέτα στον εξυπηρετητή που μόλις έχουμε εγκαταστήσει στον τελευταίο από τους «επιπλέον» CSMA κόμβους. Εγκαθιστούμε τον πελάτη στον αριστερότερο κόμβο του σημείου-προς-σημείο κομματιού που φαίνεται στην απεικόνιση της τοπολογίας.

```
UdpEchoClientHelper echoClient (csmaInterfaces.GetAddress (nCsma), 9);
echoClient.SetAttribute ("MaxPackets", UintegerValue (1));
echoClient.SetAttribute ("Interval", TimeValue (Seconds (1.0)));
echoClient.SetAttribute ("PacketSize", UintegerValue (1024));
```

```
ApplicationContainer clientApps = echoClient.Install (p2pNodes.Get (0));
clientApps.Start (Seconds (2.0));
clientApps.Stop (Seconds (10.0));
```

Καθώς έχουμε στήσει στην ουσία ένα διαδίκτυο εδώ, χρειαζόμαστε κάποια μορφή διαδικτυακής δρομολόγησης. Ο *ns-3* παρέχει αυτό που εμείς αποκαλούμε «καθολική δρομολόγηση» (global routing), προκειμένου να σας βοηθήσει. Η καθολική δρομολόγηση εκμεταλλεύεται το γεγονός ότι ολόκληρο το διαδίκτυο είναι προσβάσιμο μέσα στην προσομοίωση και εκτελείται διαμέσου όλων των κόμβων που έχουν δημιουργηθεί για την προσομοίωση — αναλαμβάνει την δύσκολη δουλειά του καθορισμού της δρομολόγησης για εσάς χωρίς να χρειάζεται να ρυθμίσετε εσείς δρομολογητές.

Βασικά, αυτό που συμβαίνει είναι ότι κάθε κόμβος συμπεριφέρεται σα να ήταν ένας δρομολογητής OSPF, ο οποίος επικοινωνεί άμεσα και μαγικά με όλους τους άλλους δρομολογητές στο παρασκήνιο. Κάθε κόμβος παράγει δηλώσεις των συνδέσεων και τις μεταδίδει κατευθείαν σε έναν διαχειριστή καθολικής δρομολόγησης, ο οποίος χρησιμοποιεί αυτή την καθολική πληροφορία για να κατασκευάσει τους πίνακες δρομολόγησης σε κάθε κόμβο. Η εγκατάσταση μιας τέτοιας μορφής δρομολόγησης γίνεται σε μία γραμμή:

```
Ipv4GlobalRoutingHelper::PopulateRoutingTables ();
```

Έπειτα ενεργοποιούμε την καταγραφή pcap. Η πρώτη γραμμή του κώδικα που ενεργοποιεί την καταγραφή pcap στον βοηθό σημείου-προς-σημείο θα πρέπει να σας είναι οικεία μέχρι τώρα. Η δεύτερη γραμμή ενεργοποιεί την καταγραφή pcap στον CSMA βοηθό και υπάρχει μία επιπλέον παράμετρος που δεν έχετε συναντήσει ακόμα.

```
pointToPoint.EnablePcapAll ("second");
csma.EnablePcap ("second", csmaDevices.Get (1), true);
```

Το CSMA δίκτυο είναι ένα πολλαπλό σημείο-προς-σημείο δίκτυο. Αυτό σημαίνει ότι μπορούν να υπάρχουν (και όντως υπάρχουν σε αυτήν την περίπτωση) πολλαπλά τερματικά σημεία σε ένα κοινόχρηστο μέσο. Κάθε ένα από αυτά τα τερματικά σημεία έχει μια δικτυακή συσκευή που συσχετίζεται με αυτό. Υπάρχουν δύο βασικές εναλλακτικές για τη συλλογή πληροφορίας ιχνών από ένα τέτοιο δίκτυο. Η μία είναι η δημιουργία ενός αρχείου καταγραφής για κάθε μία δικτυακή συσκευή και η αποθήκευση μόνο των πακέτων που μεταδίδονται ή καταναλώνονται από αυτήν την δικτυακή συσκευή. Μια άλλη εναλλακτική είναι η επιλογή μίας από τις συσκευές και η μετάβασή της σε μεικτή κατάσταση. Έτσι, αυτή μόνο η συσκευή «παρακολουθεί» (sniff) το δίκτυο για όλα τα πακέτα και τα αποθηκεύει σε ένα μοναδικό αρχείο pcap. Με αυτόν τον τρόπο λειτουργεί, για παράδειγμα, το `tcpdump`. Εκείνη η τελική παράμετρος λέει στον CSMA βοηθό εάν πρέπει ή όχι να κανονίσει ώστε να δεσμεύει πακέτα σε μεικτή κατάσταση.

Σε αυτό το παράδειγμα, θα επιλέξουμε μία από τις συσκευές στο CSMA δίκτυο και θα της ζητήσουμε να εκτελέσει μία μεικτή παρακολούθηση (promiscuous sniff) του δικτύου, προσομοιώνοντας έτσι αυτό που θα έκανε το `tcpdump`. Εάν ήσασταν σε ένα μηχάνημα με Linux, θα μπορούσατε να δώσετε την εντολή `tcpdump -i eth0` προκειμένου να πάρετε τα ίχνη. Σε αυτήν την περίπτωση, προσδιορίζουμε τη συσκευή χρησιμοποιώντας την `csmaDevices.Get(1)`, η οποία επιλέγει την πρώτη συσκευή στον container. Θέτοντας την τελευταία παράμετρο ως αληθή ενεργοποιούνται οι μεικτές καταγραφές (promiscuous captures).

Το τελευταίο μέρος του κώδικα απλά τρέχει και καθαρίζει μετά την προσομοίωση, όπως και στο παράδειγμα του `first.cc`.

```
  Simulator::Run ();
  Simulator::Destroy ();
  return 0;
}
```

Προκειμένου να εκτελέσετε αυτό το παράδειγμα, αντιγράψτε το παράδειγμα σεναρίου του `second.cc` στον κατάλογο scratch και χρησιμοποιήστε το waf για το build όπως κάνατε και στο παράδειγμα `first.cc`. Εάν είστε στον υψηλότερου επιπέδου κατάλογο του αποθετηρίου, απλά πληκτρολογήστε,

```
$ cp examples/tutorial/second.cc scratch/mysecond.cc
$ ./waf
```

Προειδοποίηση: χρησιμοποιούμε το αρχείο `second.cc` ως ένα από τα τεστ οπισθοδρόμησής μας προκειμένου να επικυρώσουμε ότι δουλεύει όπως ακριβώς πιστεύουμε ότι πρέπει να δουλεύει, ώστε η εμπειρία σας με τον παρόντα οδηγό να είναι θετική. Αυτό σημαίνει ότι υπάρχει ήδη ένα εκτελέσιμο αρχείο με το όνομα `second` σε αυτό το project. Για να αποφύγετε οποιαδήποτε σύγχυση σχετικά με το τι εκτελείτε, παρακαλούμε κάντε τη μετονομασία σε `mysecond.cc` που προτείνεται παραπάνω.

Εάν ακολουθείτε αυτόν τον οδηγό με θρησκευτική ευλάβεια (το κάνετε, έτσι δεν είναι;) θα έχετε ακόμα τη μεταβλητή NS_LOG τεθειμένη, οπότε καθαρίστε αυτή τη μεταβλητή και εκτελέστε το πρόγραμμα.

```
$ export NS_LOG=
$ ./waf --run scratch/mysecond
```

Δεδομένου ότι έχουμε στήσει τις εφαρμογές UDP echo για καταγραφή, ακριβώς όπως κάναμε και στο `first.cc`, θα δείτε μια παρόμοια έξοδο όταν τρέξετε το σενάριο.

```
Waf: Entering directory `/home/craigdo/repos/ns-3-allinone/ns-3-dev/build'
Waf: Leaving directory `/home/craigdo/repos/ns-3-allinone/ns-3-dev/build'
'build' finished successfully (0.415s)
Sent 1024 bytes to 10.1.2.4
Received 1024 bytes from 10.1.1.1
Received 1024 bytes from 10.1.2.4
```

Θυμηθείτε ότι το πρώτο μήνυμα, "`Sent 1024 bytes to 10.1.2.4`", είναι ο UDP echo πελάτης που στέλνει ένα πακέτο στον εξυπηρετητή. Στην προκειμένη περίπτωση, ο εξυπηρετητής είναι σε ένα διαφορετικό δίκτυο (10.1.2.0). Το δεύτερο μήνυμα, "`Received 1024 bytes from 10.1.1.1`", είναι από τον UDP echo εξυπηρετητή, και δημιουργήθηκε όταν αυτός έλαβε το echo πακέτο. Το τελικό μήνυμα, "`Received 1024 bytes from 10.1.2.4`", είναι από τον echo πελάτη, και δείχνει ότι αυτός έχει λάβει το echo μήνυμα από τον εξυπηρετητή.

Εάν τώρα πάτε και δείτε στον κατάλογο του υψηλότερου επιπέδου, θα βρείτε τρία αρχεία καταγραφής ιχνών:

```
second-0-0.pcap  second-1-0.pcap  second-2-0.pcap
```

Ας δούμε για μια στιγμή την ονομασία αυτών των αρχείων. Όλα έχουν την ίδια μορφή, `<name>-<node>-<device>.pcap`. Για παράδειγμα, το πρώτο αρχείο στην παραπάνω απαρίθμηση είναι το `second-0-0.pcap`, το οποίο είναι το αρχείο ιχνών pcap από τον κόμβο μηδέν, από τη συσκευή μηδέν. Αυή είναι η συσκευή σημείου-προς-σημείο στον κόμβο μηδέν. Το αρχείο `second-1-0.pcap` είναι το αρχείο ιχνών pcap για τη συσκευή μηδέν στον κόμβο ένα, η οποία είναι επίσης μια δικτυακή συσκευή σημείου-προς-σημείο. Και το αρχείο `second-2-0.pcap` είναι το αρχείο ιχνών pcap για τη συσκευή μηδέν στον κόμβο δύο.

Εάν ανατρέξετε πίσω στην απεικόνιση της τοπολογίας στην αρχή αυτής της ενότητας, θα δείτε ότι ο κόμβος μηδέν είναι ο αριστερότερος κόμβος στη σύνδεση σημείου-προς-σημείο, και ο κόμβος ένα είναι ο κόμβος που έχει τόσο μια σημείου-προς-σημείο συσκευή όσο και μια CSMA συσκευή. Θα δείτε ότι ο κόμβος δύο είναι ο πρώτος «επιπλέον» κόμβος στο CSMA δίκτυο και ότι η συσκευή μηδέν του έχει επιλεγεί ως η συσκευή που θα καταγράψει τα ίχνη στην μεικτή κατάσταση.

Τώρα, ας ακολουθήσουμε το echo πακέτο μέσα στο διαδίκτυο. Αρχικά, εκτελέστε το tcpdump του αρχείου ιχνών για τον αριστερότερο κόμβο σημείου-προς-σημείο — τον κόμβο μηδέν.

```
$ tcpdump -nn -tt -r second-0-0.pcap
```

Θα πρέπει να σας εμφανιστούν τα περιεχόμενα του αρχείου pcap:

```
reading from file second-0-0.pcap, link-type PPP (PPP)
2.000000 IP 10.1.1.1.49153 > 10.1.2.4.9: UDP, length 1024
2.017607 IP 10.1.2.4.9 > 10.1.1.1.49153: UDP, length 1024
```

Η πρώτη γραμμή του αποτελέσματος δείχνει ότι ο τύπος σύνδεσης είναι PPP (point-to-point ή σημείο-προς-σημείο) κάτι το οποίο αναμέναμε. Έπειτα θα δείτε ότι το echo πακέτο φεύγει από τον κόμβο μηδέν μέσω της συσκευής που αντιστοιχίζεται στην IP διεύθυνση 10.1.1.1 με κατεύθυνση προς την IP διεύθυνση 10.1.2.4 (ο δεξιότερος κόμβος

του CSMA). Αυτό το πακέτο θα κινηθεί πάνω από τη σύνδεση σημείου-προς-σημείο και θα παραληφθεί από τη δικτυακή συσκευή σημείου-προς-σημείο στον κόμβο ένα. Ας ρίξουμε μια ματιά:

```
$ tcpdump -nn -tt -r second-1-0.pcap
```

Θα πρέπει τώρα να βλέπετε το αποτέλεσμα των ιχνών pcap της άλλης πλευράς του συνδέσμου σημείου-προς-σημείο:

```
reading from file second-1-0.pcap, link-type PPP (PPP)
2.003686 IP 10.1.1.1.49153 > 10.1.2.4.9: UDP, length 1024
2.013921 IP 10.1.2.4.9 > 10.1.1.1.49153: UDP, length 1024
```

Εδώ βλέπουμε ότι ο τύπος σύνδεσης είναι επίσης PPP όπως θα περιμέναμε. Βλέπετε ότι το πακέτο από την IP διεύθυνση 10.1.1.1 (το οποίο στάλθηκε τη χρονική στιγμή 2.000000 δευτερολέπτων) με κατεύθυνση προς την IP διεύθυνση 10.1.2.4 εμφανίστηκε στη διεπαφή αυτή. Τώρα, εσωτερικά σε αυτόν τον κόμβο, το πακέτο θα προωθηθεί στη διεπαφή CSMA και θα πρέπει να το δούμε να εξέρχεται από τη συσκευή με κατεύθυνση προς τον τελικό του προορισμό.

Θυμηθείτε ότι επιλέξαμε τον κόμβο δύο ως τον κόμβο που θα παρακολουθήσει (promiscuous sniffer) το CSMA δίκτυο, οπότε ας δούμε στο second-2-0.pcap για να διαπιστώσουμε αν είναι όντως εκεί.

```
$ tcpdump -nn -tt -r second-2-0.pcap
```

Θα πρέπει να βλέπετε τώρα τα αποτελέσματα για τον κόμβο δύο, και τη συσκευή μηδέν:

```
reading from file second-2-0.pcap, link-type EN10MB (Ethernet)
2.007698 ARP, Request who-has 10.1.2.4 (ff:ff:ff:ff:ff:ff) tell 10.1.2.1, length 50
2.007710 ARP, Reply 10.1.2.4 is-at 00:00:00:00:00:06, length 50
2.007803 IP 10.1.1.1.49153 > 10.1.2.4.9: UDP, length 1024
2.013815 ARP, Request who-has 10.1.2.1 (ff:ff:ff:ff:ff:ff) tell 10.1.2.4, length 50
2.013828 ARP, Reply 10.1.2.1 is-at 00:00:00:00:00:03, length 50
2.013921 IP 10.1.2.4.9 > 10.1.1.1.49153: UDP, length 1024
```

Όπως μπορείτε να δείτε, ο τύπος σύνδεσης είναι τώρα “Ethernet”. Κάτι νέο έχει προκύψει, ωστόσο. Το δίκτυο αρτηρίας χρειάζεται το ARP, το Πρωτόκολλο Ανάλυσης Διευθύνσεων (Address Resolution Protocol). Ο κόμβος ένα γνωρίζει ότι χρειάζεται να στείλει το πακέτο στην IP διεύθυνση 10.1.2.4, αλλά δεν ξέρει τη MAC διεύθυνση του αντίστοιχου κόμβου. Εκπέμπει στο δίκτυο CSMA (ff:ff:ff:ff:ff:ff) αναζητώντας τη συσκευή με IP διεύθυνση 10.1.2.4. Σε αυτήν την περίπτωση, ο δεξιότερος κόμβος απαντάει λέγοντας ότι βρίσκεται στη MAC διεύθυνση 00:00:00:00:00:06. Σημειώστε ότι ο κόμβος δύο δεν εμπλέκεται άμεσα σε αυτήν την ανταλλαγή, αλλά παρακολουθεί το δίκτυο και αναφέρει κάθε κίνηση που ανιχνεύει.

Η ανταλλαγή αυτή φαίνεται στις ακόλουθες γραμμές,

```
2.007698 ARP, Request who-has 10.1.2.4 (ff:ff:ff:ff:ff:ff) tell 10.1.2.1, length 50
2.007710 ARP, Reply 10.1.2.4 is-at 00:00:00:00:00:06, length 50
```

Τότε η συσκευή ένα στον κόμβο ένα προχωρά και στέλνει το echo πακέτο στον UDP echo εξυπηρετητή στην IP διεύθυνση 10.1.2.4.

```
2.007803 IP 10.1.1.1.49153 > 10.1.2.4.9: UDP, length 1024
```

Ο εξυπηρετητής λαμβάνει το αίτημα echo και αντιστρέφει την κατεύθυνση του πακέτου προσπαθώντας να το στείλει πίσω στην πηγή. Ο εξυπηρετητής γνωρίζει ότι αυτή η διεύθυνση είναι σε ένα άλλο δίκτυο το οποίο μπορεί να προσπελάσει μέσω της IP διεύθυνσης 10.1.2.1. Αυτό συμβαίνει επειδή έχουμε ενεργοποιήσει την καθολική δρομολόγηση και τα έχει ξεκαθαρίσει όλα αυτά εκ μέρους μας. Αλλά, ο κόμβος του echo εξυπηρετητή δεν γνωρίζει τη MAC διεύθυνση του πρώτου CSMA κόμβου, οπότε πρέπει να εκτελέσει το πρωτόκολλο ARP, όπως χρειάστηκε να κάνει και ο πρώτος CSMA κόμβος.

```
2.013815 ARP, Request who-has 10.1.2.1 (ff:ff:ff:ff:ff:ff) tell 10.1.2.4, length 50
2.013828 ARP, Reply 10.1.2.1 is-at 00:00:00:00:00:03, length 50
```

Ο εξυπηρετητής στέλνει τότε το echo πακέτο πίσω στον κόμβο προώθησης.

```
2.013921 IP 10.1.2.4.9 > 10.1.1.1.49153: UDP, length 1024
```

Κοιτώντας πίσω στον δεξιότερο κόμβο του συνδέσμου σημείου-προς-σημείο,

```
$ tcpdump -nn -tt -r second-1-0.pcap
```

Μπορείτε να δείτε τώρα το πακέτο που έχει γίνει echo να επιστρέφει μέσω του συνδέσμου σημείου-προς-σημείο στην τελευταία γραμμή των αποτελεσμάτων καταγραφής.

```
reading from file second-1-0.pcap, link-type PPP (PPP)
2.003686 IP 10.1.1.1.49153 > 10.1.2.4.9: UDP, length 1024
2.013921 IP 10.1.2.4.9 > 10.1.1.1.49153: UDP, length 1024
```

Τέλος, μπορείτε να δείτε πίσω στον κόμβο που ξεκίνησε το echo

```
$ tcpdump -nn -tt -r second-0-0.pcap
```

και να διαπιστώσετε ότι το πακέτο που έγινε echo φτάνει πίσω στην πηγή τη χρονική στιγμή των 2.007602 δευτερολέπτων,

```
reading from file second-0-0.pcap, link-type PPP (PPP)
2.000000 IP 10.1.1.1.49153 > 10.1.2.4.9: UDP, length 1024
2.017607 IP 10.1.2.4.9 > 10.1.1.1.49153: UDP, length 1024
```

Εν τέλει, θυμηθείτε ότι προσθέσαμε την δυνατότητα ελέγχου του αριθμού των CSMA συσκευών στη προσομοίωση μέσω ορίσματος στη γραμμή εντολών. Μπορείτε να αλλάξετε αυτό το όρισμα με τον ίδιο τρόπο όπως τότε που εξετάσαμε την αλλαγή του αριθμού των πακέτων που γίνονται echo στο παράδειγμα `first.cc`. Δοκιμάστε να τρέξετε το πρόγραμμα με το αριθμό των «επιπλέον» συσκευών να είναι ίσος με τέσσερεις:

```
$ ./waf --run "scratch/mysecond --nCsma=4"
```

Θα πρέπει τώρα να βλέπετε τα εξής:

```
Waf: Entering directory `/home/craigdo/repos/ns-3-allinone/ns-3-dev/build'
Waf: Leaving directory `/home/craigdo/repos/ns-3-allinone/ns-3-dev/build'
'build' finished successfully (0.405s)
At time 2s client sent 1024 bytes to 10.1.2.5 port 9
At time 2.0118s server received 1024 bytes from 10.1.1.1 port 49153
At time 2.0118s server sent 1024 bytes to 10.1.1.1 port 49153
At time 2.02461s client received 1024 bytes from 10.1.2.5 port 9
```

Παρατηρήστε ότι ο εξυπηρετητής echo έχει επανατοποθετηθεί πλέον στον τελευταίο από τους CSMA κόμβους, ο οποίος είναι στη διεύθυνση 10.1.2.5 αντί της προεπιλεγμένης, 10.1.2.4.

Είναι πιθανό να μην είστε ικανοποιημένοι με ένα αρχείο καταγραφής που δημιουργείται από έναν παρατηρητή στο CSMA δίκτυο. Μπορεί να θέλετε πραγματικά να λάβετε ίχνη από μια μόνο συσκευή και να μην ενδιαφέρεστε για κάποια άλλη κίνηση στο δίκτυο. Αυτό μπορείτε να το κάνετε αρκετά εύκολα.

Ας δούμε το `scratch/mysecond.cc` και ας προσθέσουμε εκείνον τον κώδικα που θα μας επιτρέψει να είμαστε πιο συγκεκριμένοι. Οι βοηθοί του `ns-3` παρέχουν μεθόδους που δέχονται έναν αριθμό κόμβου και έναν αριθμό συσκευής ως παραμέτρους. Προχωρήστε και αντικαταστήστε τις κλήσεις `EnablePcap` με τις παρακάτω κλήσεις.

```
pointToPoint.EnablePcap ("second", p2pNodes.Get (0)->GetId (), 0);
csma.EnablePcap ("second", csmaNodes.Get (nCsma)->GetId (), 0, false);
csma.EnablePcap ("second", csmaNodes.Get (nCsma-1)->GetId (), 0, false);
```

Γνωρίζουμε ότι θέλουμε να δημιουργήσουμε ένα αρχείο pcap με το βασικό όνομα “second”, και επίσης γνωρίζουμε ότι η συσκευή που μας ενδιαφέρει και στις δύο περιπτώσεις πρόκειται να είναι η μηδέν, κατά συνέπεια οι παράμετροι δεν είναι και πολύ ενδιαφέρουσες.

Προκειμένου να βρείτε τον αριθμό του κόμβου, έχετε δύο επιλογές: πρώτον, οι κόμβοι είναι αριθμημένοι με μονότονα αύξοντα τρόπο, ξεκινώντας από το μηδέν, με τη σειρά σύμφωνα με την οποία τους δημιουργείτε. Ένας τρόπος για να πάρετε τον αριθμό ενός κόμβου είναι να βρείτε τον αριθμό αυτό «μηχανικά», μελετώντας τη σειρά της δημιουργίας των κόμβων. Εάν ρίξετε μια ματιά στην απεικόνιση της τοπολογίας του δικτύου στην αρχή του αρχείου, θα δείτε ότι το κάναμε ήδη αυτό για εσάς, και θα διαπιστώσετε ότι ο τελευταίος CSMA κόμβος είναι ο κόμβος με τον αριθμό `nCsma + 1`. Αυτή η προσέγγιση μπορεί να καταστεί ενοχλητικά δύσκολη σε μεγαλύτερες προσομοιώσεις.

Ένας εναλλακτικός τρόπος, τον οποίο χρησιμοποιούμε εδώ, είναι το να διαπιστώσετε ότι οι `NodeContainers` περιέχουν δείκτες προς αντικείμενα `Node` του *ns-3*. Το αντικείμενο `Node` έχει μια μέθοδο που ονομάζεται `GetId`, η οποία επιστρέφει την ID του κόμβου, η οποία είναι ο αριθμός του κόμβου που ψάχνουμε. Ας πάμε να δούμε στο Doxygen για το `Node` και ας εντοπίσουμε αυτή τη μέθοδο, η οποία βρίσκεται στο χαμηλότερο επίπεδο του πυρήνα του *ns-3* που έχουμε φτάσει μέχρι στιγμής. Κάποιες φορές θα χρειαστεί όντως να ψάξετε με μεγάλη επιμέλεια για να βρείτε χρήσιμα πράγματα.

Μεταβείτε στην τεκμηρίωση του Doxygen για την έκδοσή σας (θυμηθείτε ότι μπορείτε να τη βρείτε στον ιστότοπο του project). Μπορείτε να βρείτε την τεκμηρίωση του `Node` ψάχνοντας στην καρτέλα "Classes" και κατεβαίνοντας κάτω στην "Class List" μέχρι να βρείτε το `ns3::Node`. Επιλέξτε το `ns3::Node` και θα μεταβείτε στην τεκμηρίωση για την κλάση `Node`. Εάν κατεβείτε κάτω στη μέθοδο `GetId` και την επιλέξετε, θα μεταβείτε σε μια λεπτομερή τεκμηρίωση της μεθόδου αυτής. Η χρήση της μεθόδου `GetId` μπορεί να κάνει τον προσδιορισμό του αριθμού ενός κόμβου πολύ ευκολότερο σε πολύπλοκες τοπολογίες.

Ας διαγράψουμε τα παλιά αρχεία καταγραφής από τον κατάλογο υψηλότερου επιπέδου, ώστε να αποφύγουμε οποιαδήποτε σύγχυση σχετικά με το τι συμβαίνει.

```
$ rm *.pcap
$ rm *.tr
```

Εάν κάνετε build το νέο σενάριο και εκτελέσετε την προσομοίωση θέτοντας τη μεταβλητή `nCsma` στο 100,

```
$ ./waf --run "scratch/mysecond --nCsma=100"
```

θα δείτε την ακόλουθη έξοδο:

```
Waf: Entering directory `/home/craigdo/repos/ns-3-allinone/ns-3-dev/build'
Waf: Leaving directory `/home/craigdo/repos/ns-3-allinone/ns-3-dev/build'
'build' finished successfully (0.407s)
At time 2s client sent 1024 bytes to 10.1.2.101 port 9
At time 2.0068s server received 1024 bytes from 10.1.1.1 port 49153
At time 2.0068s server sent 1024 bytes to 10.1.1.1 port 49153
At time 2.01761s client received 1024 bytes from 10.1.2.101 port 9
```

Σημειώστε ότι ο echo εξυπηρετητής βρίσκεται τώρα στη διεύθυνση 10.1.2.101, γεγονός που οφείλεται στο ότι έχουμε 100 «επιπλέον» CSMA κόμβους, μαζί με τον echo εξυπηρετητή που βρίσκεται στον τελευταίο από αυτούς. Εάν ζητήσετε τη λίστα με τα αρχεία pcap στον κατάλογο υψηλότερου επιπέδου θα δείτε τα εξής:

```
second-0-0.pcap  second-100-0.pcap  second-101-0.pcap
```

Το αρχείο καταγραφής `second-0-0.pcap` είναι η «αριστερότερη» συσκευή σημείου-προς-σημείο, η οποία είναι η πηγή του πακέτου echo. Το αρχείο `second-101-0.pcap` αντιστοιχεί στην δεξιότερη CSMA συσκευή στην οποία βρίσκεται ο εξυπηρετητής echo. Μπορεί να παρατηρήσατε ότι η τελευταία παράμετρος κατά την κλήση για ενεργοποίηση της pcap καταγραφής στον κόμβο του echo εξυπηρετητή είναι τεθειμένη ως ψευδής. Αυτό σημαίνει ότι τα ίχνη που συγκεντρώνονται σε αυτόν τον κόμβο ήταν σε μη-μεικτή κατάσταση.

Για να διευκρινίσουμε τη διαφορά μεταξύ μεικτών και μη-μεικτών ιχνών, ζητήσαμε επιπλέον ένα μη-μεικτό ίχνος για τον κόμβο δίπλα από τον τελευταίο. Ρίξτε μια ματιά στο `tcpdump` για το αρχείο `second-100-0.pcap`.

```
$ tcpdump -nn -tt -r second-100-0.pcap
```

Μπορείτε τώρα να δείτε ότι ο κόμβος 100 είναι πράγματι ένας παρατηρητής κατά την ανταλλαγή echo. Τα μόνα πακέτα που λαμβάνει είναι οι αιτήσεις ARP οι οποίες εκπέμπονται σε όλο το CSMA δίκτυο.

```
reading from file second-100-0.pcap, link-type EN10MB (Ethernet)
2.006698 ARP, Request who-has 10.1.2.101 (ff:ff:ff:ff:ff:ff) tell 10.1.2.1, length 50
2.013815 ARP, Request who-has 10.1.2.1 (ff:ff:ff:ff:ff:ff) tell 10.1.2.101, length 50
```

Δείτε τώρα το `tcpdump` για το αρχείο `second-101-0.pcap`.

```
$ tcpdump -nn -tt -r second-101-0.pcap
```

Μπορείτε τώρα να δείτε ότι ο κόμβος 101 είναι πραγματικά ο παραλήπτης της ανταλλαγής echo.

```
reading from file second-101-0.pcap, link-type EN10MB (Ethernet)
2.006698 ARP, Request who-has 10.1.2.101 (ff:ff:ff:ff:ff:ff) tell 10.1.2.1, length 50
2.006698 ARP, Reply 10.1.2.101 is-at 00:00:00:00:00:67, length 50
2.006803 IP 10.1.1.1.49153 > 10.1.2.101.9: UDP, length 1024
2.013803 ARP, Request who-has 10.1.2.1 (ff:ff:ff:ff:ff:ff) tell 10.1.2.101, length 50
2.013828 ARP, Reply 10.1.2.1 is-at 00:00:00:00:00:03, length 50
2.013828 IP 10.1.2.101.9 > 10.1.1.1.49153: UDP, length 1024
```

## 6.2 Μοντέλα, Χαρακτηριστικά και Πραγματικότητα

Αυτό το σημείο ενδείκνυται ώστε να κάνουμε μια μικρή παρένθεση και να σημειώσουμε κάτι σημαντικό. Μπορεί να είναι φανερό σε εσάς ή μπορεί και όχι, μα όποτε κάποιος πραγματοποιεί μια προσομοίωση, είναι σημαντικό να καταλαβαίνει ακριβώς τι μοντελοποιείται και τι όχι. Είναι δελεαστικό, για παράδειγμα, να σκεφτεί κανείς ότι οι CSMA συσκευές και τα κανάλια που χρησιμοποιήθηκαν στην προηγούμενη ενότητα είναι σαν αληθινές συσκευές Ethernet. Και να περιμένει ότι το αποτέλεσμα μιας προσομοίωσης θα αντανακλά άμεσα το τι θα συμβεί σε ένα πραγματικό Ethernet. Δεν είναι, όμως, έτσι τα πράγματα.

Ένα μοντέλο είναι, εξ ορισμού, μια αφαίρεση της πραγματικότητας. Είναι τελικά στην κρίση του συγγραφέα του σεναρίου προσομοίωσης το να καθορίσει το λεγόμενο «εύρος της ακρίβειας» και «την περιοχή της εφαρμοσιμότητας» της προσομοίωσης συνολικά, και κατά συνέπεια των συστατικών μερών της.

Σε μερικές περιπτώσεις, όπως στο `Csma`, μπορεί να είναι σχετικά εύκολο το να καθοριστεί τι *δεν* μπορεί να μοντελοποιηθεί. Διαβάζοντας την περιγραφή του μοντέλου (`csma.h`) μπορείτε να διαπιστώσετε ότι δεν υπάρχει ανίχνευση συγκρούσεων στο CSMA μοντέλο, και να αποφασίσετε το κατά πόσο εφαρμόσιμη θα είναι η χρήση του στην προσομοίωσή σας ή τι προειδοποιήσεις μπορεί να θέλετε να συμπεριλάβετε στα αποτελέσματά σας. Σε άλλες περιπτώσεις, μπορεί να είναι αρκετά εύκολη η ρύθμιση συμπεριφορών που ενδέχεται να μη συμβαδίζουν με την πραγματικότητα. Το να αφιερώσετε χρόνο εξετάζοντας μερικές τέτοιες περιπτώσεις θα αποδειχτεί ότι αξίζει τον κόπο, καθώς και το να εξετάσετε πόσο εύκολα μπορείτε να παρεκκλίνετε εκτός των ορίων της πραγματικότητας στις προσομοιώσεις σας.

Όπως θα έχετε δει, ο *ns-3* παρέχει `Attributes` τα οποία ένας χρήστης μπορεί εύκολα να θέσει ώστε να αλλάξει τη συμπεριφορά του μοντέλου. Θεωρήστε δύο από τα `Attributes` της `CsmaNetDevice`: το `Mtu` και το `EncapsulationMode`. Το `Mtu` χαρακτηριστικό υποδηλώνει τη Μέγιστη Μονάδα Μετάδοσης (Maximum Transmission Unit) της συσκευής. Είναι το μέγεθος της μεγαλύτερης Μονάδας Δεδομένων του Πρωτοκόλλου (Protocol Data Unit ή PDU) που μπορεί να στείλει η συσκευή.

Το MTU είναι εξ ορισμού στα 1500 byte στην `CsmaNetDevice`. Αυτή η προεπιλογή αντιστοιχεί σε έναν αριθμό που υπάρχει στο πρότυπο RFC 894, “A Standard for the Transmission of IP Datagrams over Ethernet Networks”. Ο αριθμός προέρχεται όντως από το μέγιστο μέγεθος πακέτου για δίκτυα τύπου 10Base5 (full-spec Ethernet) – 1518 byte. Εάν αφαιρέσετε το πλεόνασμα της DIX ενθυλάκωσης (DIX encapsulation overhead) για τα Ethernet πακέτα (18 byte) θα καταλήξετε να έχετε μέγιστο δυνατό μέγεθος δεδομένων (MTU) ίσο με 1500 byte. Κάποιοι μπορεί να παρατηρήσουν ότι το `MTU` για δίκτυα IEEE 802.3 είναι 1492 byte. Αυτό συμβαίνει επειδή η ενθυλάκωση LLC/SNAP προσθέτει ένα επιπρόσθετο βάρος από byte στο πλεόνασμα του πακέτου. Και στις δύο περιπτώσεις, το υποκείμενο υλικό μπορεί να στείλει μόνο 1518 byte, αλλά το μέγεθος των δεδομένων είναι διαφορετικό.

Προκειμένου να καθορίσουμε την κατάσταση ενθυλάκωσης, η `CsmaNetDevice` παρέχει ένα `Attribute` που καλείται `EncapsulationMode`, το οποίο μπορεί να πάρει τις τιμές `Dix` ή `Llc`. Αυτές αντιστοιχούν στην πλαισίωση Ethernet και LLC/SNAP κατ' αναλογία.

Αν αφήσει κάποιος το `Mtu` στα 1500 byte και αλλάξει την κατάσταση ενθυλάκωσης σε `Llc`, το αποτέλεσμα θα είναι ένα δίκτυο το οποίο ενθυλακώνει PDU των 1500 byte σε LLC/SNAP πλαίσια, που έχει σα συνέπεια την ύπαρξη πακέτων των 1526 byte, κάτι που θα ήταν ανεπιτρεπτό σε πολλά δίκτυα, καθώς αυτά μπορούν να μεταδώσουν το πολύ 1518 byte ανά πακέτο. Πιθανότατα αυτό να είχε ως αποτέλεσμα μια προσομοίωση που κατά περίεργο τρόπο δεν θα αντανακλά την πραγματικότητα που μπορεί εσείς να περιμένατε.

Για να κάνουμε πιο περίπλοκη την όλη υπόθεση, υπάρχουν τεράστια πλαίσια (1500 < MTU <= 9000 byte) και υπερ-τεράστια (MTU > 9000 byte) πλαίσια που δεν είναι επίσημα επικυρωμένα από την IEEE, αλλά είναι διαθέσιμα για κάποια δίκτυα υψηλών ταχυτήτων (Gigabit) και NIC. Κάποιος θα μπορούσε να αφήσει την κατάσταση ενθυλάκωσης στην επιλογή `Dix`, και να θέσει το `Attribute Mtu` σε μια `CsmaNetDevice` στα 64000 byte – ακόμα και αν ένα σχετικό `CsmaChannel DataRate` ήταν καθορισμένο στα 10 megabit ανά δευτερόλεπτο. Αυτό θα μοντελοποιούσε στην ουσία έναν διακόπτη Ethernet, φτιαγμένο από δίκτυα 10Base5 της δεκαετίας του 1980 συνδεδεμένα με συσκευές vampire tap που υποστηρίζουν υπερ-τεράστια πακέτα. Αυτό σίγουρα δεν είναι κάτι που έχει φτιαχτεί στο παρελθόν, ούτε και πρόκειται να φτιαχτεί, αλλά είναι κάτι που μπορείτε πολύ εύκολα να ρυθμίσετε εσείς.

Στο προηγούμενο παράδειγμα, χρησιμοποιήσατε τη γραμμή εντολών για να δημιουργήσετε μια προσομοίωση η οποία είχε 100 `Csma` κόμβους. Θα μπορούσατε το ίδιο απλά να έχετε δημιουργήσει μια προσομοίωση με 500 κόμβους. Εάν μοντελοποιούσατε πραγματικά το παραπάνω δίκτυο 10Base5 με τις vampire tap συσκευές, το μέγιστο μήκος ενός full-spec Ethernet καλωδίου είναι 500 μέτρα, με ελάχιστη απόσταση μεταξύ συσκευών τα 2.5 μέτρα. Κάτι που σημαίνει ότι θα μπορούσαν να υπάρχουν μόνο 200 τέτοιες συσκευές σε ένα πραγματικό δίκτυο. Επίσης, θα μπορούσατε αρκετά εύκολα να φτιάξετε και ένα δίκτυο εκτός των προτύπων με τον ίδιο τρόπο. Κάτι τέτοιο μπορεί να έχει ή να μην έχει έχει ως αποτέλεσμα μια ουσιαστική προσομοίωση, δεδομένου και του τι προσπαθείτε να μοντελοποιήσετε.

Παρόμοιες καταστάσεις μπορούν να προκύψουν σε πολλά σημεία στον *ns-3* και σε οποιαδήποτε προσομοίωση. Για παράδειγμα, μπορεί να είστε σε θέση να τοποθετήσετε κόμβους με τέτοιο τρόπο ώστε να καταλαμβάνουν τον ίδιο χώρο την ίδια στιγμή, ή μπορεί να είστε σε θέση να ρυθμίσετε του ενισχυτές ή τα επίπεδα θορύβου ώστε να παραβιάζουν τους βασικούς νόμους της Φυσικής.

Ο *ns-3* ευνοεί γενικά την ευελιξία, και πολλά μοντέλα θα σας επιτρέψουν να θέσετε ελεύθερα `Attributes` χωρίς να προσπαθήσουν να σας επιβάλλουν οποιαδήποτε αυθαίρετη συνοχή ή συγκεκριμένη υποκείμενη προδιαγραφή.

Αυτό που θα πρέπει να κρατήσετε εσείς είναι ότι ο *ns-3* θα σας παρέχει μια υπερ-ευέλικτη βάση πάνω στην οποία μπορείτε να πειραματιστείτε. Εξαρτάται από εσάς το αν θα κατανοήσετε τι ζητάτε από το σύστημα να κάνει και αν θα διασφαλίσετε ότι οι προσομοιώσεις που δημιουργείτε έχουν κάποιο νόημα και κάποια σύνδεση με κάποια πραγματικότητα που καθορίζεται από εσάς.

## 6.3 Δημιουργώντας μια Τοπολογία Ασύρματου Δικτύου

Σε αυτήν την ενότητα θα επεκτείνουμε τις γνώσεις μας για τις δικτυακές συσκευές και τα κανάλια του *ns-3* ώστε να καλύψουμε ένα παράδειγμα ενός ασύρματου δικτύου. Ο *ns-3* παρέχει ένα σύνολο από μοντέλα τύπου 802.11 που επιχειρούν να παρέχουν μια ακριβή υλοποίηση MAC-επιπέδου των προδιαγραφών του 802.11, και ένα «όχι-και-τόσο-αργό» μοντέλο φυσικού επιπέδου σύμφωνα με τις προδιαγραφές του 802.11a.

Ακριβώς όπως έχουμε δει στα αντικείμενα βοηθών τοπολογίας και σημείου-προς-σημείο και CSMA όταν κατασκευάζαμε τοπολογίες σημείου-προς σημείο, θα δούμε ανάλογους βοηθούς τοπολογίας `Wifi` σε αυτήν την ενότητα. Η εμφάνιση και η λειτουργία αυτών των βοηθών θα πρέπει να σας είναι αρκετά οικεία.

Σας παρέχουμε ένα παραδειγματικό σενάριο στον κατάλογο `examples/tutorial`. Το σενάριο αυτό βασίζεται πάνω στο σενάριο `second.cc` και προσθέτει ένα δίκτυο Wifi. Προχωρήστε και ανοίξτε το αρχείο `examples/tutorial/third.cc` στον επεξεργαστή κειμένου της προτίμησής σας. Θα έχετε ήδη δει αρκετό κώδικα του *ns-3* ώστε πλέον να μπορείτε να καταλάβετε τα περισσότερα από αυτά που συμβαίνουν σε αυτό

το παράδειγμα, αλλά υπάρχουν και μερικά νέα πράγματα, οπότε θα περιηγηθούμε κατά μήκος ολόκληρου του σενάριου και θα εξετάσουμε κάποια από τα αποτελέσματά του.

Όπως και στο παράδειγμα `second.cc` (και σε όλα τα παραδείγματα του *ns-3*) το αρχείο ξεκινά με μια γραμμή κατάστασης για τον emacs και κάποιες κοινές δηλώσεις GPL.

Ρίξτε μια ματιά στην ASCII τέχνη (που παρατίθεται παρακάτω), η οποία δείχνει την προεπιλεγμένη τοπολογία δικτύου που κατασκευάζεται στο παράδειγμα. Μπορείτε να δείτε ότι πρόκειται να επεκτείνουμε το παράδειγμά μας συνδέοντας ένα ασύρματο δίκτυο στην αριστερή πλευρά. Παρατηρήστε ότι αυτή είναι μια προεπιλεγμένη τοπολογία δικτύου, καθώς μπορείτε στην ουσία να αλλάξετε τον αριθμό των κόμβων που δημιουργούνται στα ενσύρματα και ασύρματα δίκτυα. Όπως και στην περίπτωση του σεναρίου `second.cc`, εάν αλλάξετε τη `nCsma`, θα σας δώσει έναν αριθμό από «επιπλέον« CSMA κόμβους. Με παρόμοιο τρόπο, μπορείτε να θέσετε τη μεταβλητή `nWifi` ώστε να ελέγχετε πόσοι `STA` κόμβοι (station ή σταθμοί) θα δημιουργηθούν στην προσομοίωση. Πάντα θα υπάρχει ένας κόμβος `AP` (access point ή σημείο πρόσβασης) στο ασύρματο δίκτυο. Εξ ορισμού, υπάρχουν τρεις «επιπλέον» CSMA κόμβου και τρεις ασύρματοι `STA` κόμβοι.

Ο κώδικας ξεκινά με τη φόρτωση αρχείων συμπερίληψης ενοτήτων, όπως έγινε και στο παράδειγμα `second.cc`. Υπάρχουν μερικές νέες συμπεριλήψεις κώδικα που αντιστοιχούν στην ενότητα του Wifi και στην ενότητα της κινητικότητας (mobility), για τις οποίες θα πούμε παρακάτω.

```
#include "ns3/core-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/network-module.h"
#include "ns3/applications-module.h"
#include "ns3/wifi-module.h"
#include "ns3/mobility-module.h"
#include "ns3/csma-module.h"
#include "ns3/internet-module.h"
```

Ακολουθεί η απεικόνιση της τοπολογίας δικτύου:

```
// Default Network Topology
//
//   Wifi 10.1.3.0
//                 AP
//  *    *    *    *
//  |    |    |    |    10.1.1.0
// n5   n6   n7   n0 -------------- n1   n2   n3   n4
//                   point-to-point  |    |    |    |
//                                   ================
//                                     LAN 10.1.2.0
```

Μπορείτε να δείτε ότι προσθέτουμε μια καινούργια δικτυακή συσκευή στον κόμβο στην αριστερή πλευρά της σύνδεσης σημείου-προς-σημείο, η οποία γίνεται το σημείο πρόσβασης για το ασύρματο δίκτυο. Ένας αριθμός από ασύρματους STA κόμβους δημιουργείται ώστε να γεμίσει το νέο δίκτυο με διεύθυνση 10.1.3.0, όπως φαίνεται στην αριστερή πλευρά της απεικόνισης.

Μετά την απεικόνιση, χρησιμοποιείται ο χώρος ονομάτων του `ns-3` και ορίζεται ένα στοιχείο καταγραφής. Όλα αυτά θα πρέπει να σας είναι αρκετά οικεία πλέον.

```
using namespace ns3;

NS_LOG_COMPONENT_DEFINE ("ThirdScriptExample");
```

Το κυρίως πρόγραμμα ξεκινά όπως και στο `second.cc` με την προσθήκη μερικών παραμέτρων της γραμμής εντολών για την ενεργοποίηση ή απενεργοποίηση των στοιχείων καταγραφής και για την αλλαγή του αριθμού των συσκευών που δημιουργούνται.

```
bool verbose = true;
uint32_t nCsma = 3;
```

```
uint32_t nWifi = 3;

CommandLine cmd;
cmd.AddValue ("nCsma", "Number of \"extra\" CSMA nodes/devices", nCsma);
cmd.AddValue ("nWifi", "Number of wifi STA devices", nWifi);
cmd.AddValue ("verbose", "Tell echo applications to log if true", verbose);

cmd.Parse (argc,argv);

if (verbose)
  {
    LogComponentEnable("UdpEchoClientApplication", LOG_LEVEL_INFO);
    LogComponentEnable("UdpEchoServerApplication", LOG_LEVEL_INFO);
  }
```

Όπως και σε όλα τα προηγούμενα παραδείγματα, το επόμενο βήμα είναι η δημιουργία δύο κόμβων που θα ενώνονται μέσω ενός συνδέσμου σημείου-προς-σημείο.

```
NodeContainer p2pNodes;
p2pNodes.Create (2);
```

Έπειτα, συναντάμε έναν παλιό μας γνώριμο. Δημιουργούμε έναν `PointToPointHelper` και θέτουμε τα σχετικά προεπιλεγμένα `Attributes`, έτσι ώστε να δημιουργήσουμε έναν πομπό με ταχύτητα μετάδοσης πέντε megabit ανά δευτερόλεπτο πάνω στις συσκευές που δημιουργήσαμε με τη βοήθεια του βοηθού, και για να ορίσουμε καθυστέρηση δύο μιλιδευτερολέπτων στα κανάλια που δημιουργήθηκαν από τον βοηθό. Έπειτα εγκαθιστούμε τις συσκευές στους κόμβους και τα κανάλια ανάμεσά τους.

```
PointToPointHelper pointToPoint;
pointToPoint.SetDeviceAttribute ("DataRate", StringValue ("5Mbps"));
pointToPoint.SetChannelAttribute ("Delay", StringValue ("2ms"));

NetDeviceContainer p2pDevices;
p2pDevices = pointToPoint.Install (p2pNodes);
```

Στη συνέχεια, δηλώνουμε άλλον ένα `NodeContainer`, προκειμένου να περιέχει τους κόμβους που θα είναι μέρος του δικτύου αρτηρίας (CSMA).

```
NodeContainer csmaNodes;
csmaNodes.Add (p2pNodes.Get (1));
csmaNodes.Create (nCsma);
```

Η επόμενη γραμμή κώδικα παίρνει (`Gets`) τον πρώτο κόμβο (δηλαδή σα να έχει ένα ευρετήριο που να περιέχει έναν) από τον container των κόμβων σημείου-προς-σημείο και τον προσθέτει στον container των κόμβων οι οποίοι θα δεχτούν μετέπειτα τις CSMA συσκευές. Ο εν λόγω κόμβος πρόκειται να καταλήξει να έχει μια συσκευή σημείου-προς-σημείο και μια CSMA συσκευή. Έπειτα δημιουργούμε έναν αριθμός από «επιπλέον» κόμβους, οι οποίοι συνθέτουν το υπόλοιπο του CSMA δικτύου.

Έπειτα δημιουργούμε έναν `CsmaHelper` και θέτουμε τα `Attributes` του όπως κάναμε και στο προηγούμενο παράδειγμα. Δημιουργούμε ένα `NetDeviceContainer` για να καταγράψουμε τις δικτυακές συσκευές CSMA που δημιουργήθηκαν και στη συνέχεια εγκαθιστούμε τις CSMA συσκευές στους επιλεγμένους κόμβους.

```
CsmaHelper csma;
csma.SetChannelAttribute ("DataRate", StringValue ("100Mbps"));
csma.SetChannelAttribute ("Delay", TimeValue (NanoSeconds (6560)));

NetDeviceContainer csmaDevices;
csmaDevices = csma.Install (csmaNodes);
```

Μετά πρόκειται να δημιουργήσουμε τους κόμβους που θα είναι μέρος του Wifi δικτύου. Θα δημιουργησουμε έναν

αριθμό από κόμβους-«σταθμούς», όπως προσδιορίζεται από το όρισμα στη γραμμή εντολών, και θα χρησιμοποιήσουμε τον «αριστερότερο» κόμβο του συνδέσμου σημείου-προς-σημείο ως τον κόμβο για το σημείο πρόσβασης.

```
NodeContainer wifiStaNodes;
wifiStaNodes.Create (nWifi);
NodeContainer wifiApNode = p2pNodes.Get (0);
```

Το επόμενο κομμάτι κώδικα κατασκευάζει τις συσκευές wifi και το διασυνδετικό κανάλι ανάμεσα σε αυτούς τους wifi κόμβους. Αρχικά, ρυθμίζει τους βοηθούς φυσικού επιπέδου (PHY) και καναλιού:

```
YansWifiChannelHelper channel = YansWifiChannelHelper::Default ();
YansWifiPhyHelper phy = YansWifiPhyHelper::Default ();
```

Για λόγους απλότητας, αυτός ο κώδικας χρησιμοποιεί την προεπιλεγμένη ρύθμιση του φυσικού επιπέδου και μοντέλα καναλιού που έχουν τεκμηριωθεί στην τεκμηρίωση API στο Doxygen για τις μεθόδους `YansWifiChannelHelper::Default` και `YansWifiPhyHelper::Default`. Μόλις δημιουργηθούν αυτά τα αντικείμενα, δημιουργούμε ένα αντικείμενο καναλιού και το συσχετίζουμε με τον διαχειριστή αντικειμένων μας του φυσικού επιπέδου, ώστε να σιγουρευτούμε ότι όλα τα αντικείμενα που δημιουργούνται στο φυσικό επίπεδο από τον `YansWifiPhyHelper` μοιράζονται το ίδιο βασικό κανάλι, που σημαίνει ότι μοιράζονται το ίδιο ασύρματο μέσο και μπορούν να επικοινωνήσουν και να παρέμβουν:

```
phy.SetChannel (channel.Create ());
```

Μόλις ρυθμιστεί και ο βοηθός φυσικού επιπέδου (PHY), μπορούμε να επικεντρωθούμε στο MAC επίπεδο. Εδώ επιλέγουμε να δουλέψουμε με non-Qos MAC, οπότε χρησιμοποιούμε ένα αντικείμενο NqosWifiMacHelper για να θέσουμε τις παραμέτρους που αφορούν το MAC.

```
WifiHelper wifi = WifiHelper::Default ();
wifi.SetRemoteStationManager ("ns3::AarfWifiManager");

NqosWifiMacHelper mac = NqosWifiMacHelper::Default ();
```

Η μέθοδος `SetRemoteStationManager` λέει στον βοηθό τον τύπο του αλγορίθμου για τον έλεγχο του ρυθμού που πρέπει να χρησιμοποιήσει. Εδώ, ζητάει από τον βοηθό να χρησιμοποιήσει τον αλγόριθμο AARF — λεπτομέρειες σχετικά με αυτόν υπάρχουν, φυσικά, στο Doxygen.

Έπειτα, ρυθμίζουμε τον τύπο του MAC, το SSID του δικτύου υποδομής που θέλουμε να στήσουμε και διασφαλίζουμε ότι οι σταθμοί μας δεν πραγματοποιούν ενεργητικές ανιχνεύσεις:

```
Ssid ssid = Ssid ("ns-3-ssid");
mac.SetType ("ns3::StaWifiMac",
  "Ssid", SsidValue (ssid),
  "ActiveProbing", BooleanValue (false));
```

Αυτός ο κώδικας αρχικά δημιουργεί ένα αντικείμενο τύπου 802.11 SSID, το οποίο θα χρησιμοποιηθεί για να τεθεί η τιμή του “Ssid” `Attribute` της υλοποίησης του MAC επιπέδου. Το συγκεκριμένο είδος επιπέδου MAC που θα δημιουργηθεί από τον βοηθό καθορίζεται μέσω `Attribute` ως τύπου “ns3::StaWifiMac”. Η χρήση του `NqosWifiMacHelper` θα διασφαλίσει ότι το `Attribute` “QosSupported” για τα δημιουργηθέντα αντικείμενα MAC θα τεθεί ως ψευδές. Ο συνδυασμός αυτών των δύο ρυθμίσεων σημαίνει ότι το επόμενο αντικείμενο MAC που θα δημιουργηθεί θα είναι ένας σταθμός (STA) non-QoS non-AP σε μια BSS υποδομή (π.χ. μία BSS με ένα AP). Τέλος, το `Attribute` “ActiveProbing” τίθεται ως ψευδές. Αυτό σημαίνει ότι δε θα στέλνονται αιτήματα ανίχνευσης από τα MAC που δημιουργούνται από αυτόν τον βοηθό.

Μόλις ρυθμιστούν πλήρως όλες οι παράμετροι που αφορούν τους σταθμούς, τόσο στο MAC όσο και στο φυσικό επίπεδο, μπορούμε να καλέσουμε τη γνωστή μας μέθοδο `Install` για να δημιουργήσουμε τις συσκευές wifi σε αυτούς τους σταθμούς:

```
NetDeviceContainer staDevices;
staDevices = wifi.Install (phy, mac, wifiStaNodes);
```

Έχουμε ρυθμίσει το Wifi για όλους τους STA κόμβους μας, και τώρα χρειάζεται να ρυθμίσουμε τον AP κόμβο (σημείο πρόσβασης). Ξεκινάμε αυτή τη διαδικασία αλλάζοντας τα προεπιλεγμένα `Attributes` του `NqosWifiMacHelper` ώστε να ανταποκρίνονται στις απαιτήσεις του AP.

```
mac.SetType ("ns3::ApWifiMac",
             "Ssid", SsidValue (ssid));
```

Σε αυτή την περίπτωση, ο `NqosWifiMacHelper` θα δημιουργήσει επίπεδα MAC του "ns3::ApWifiMac", με το τελευταίο να ορίζει ότι πρέπει να δημιουργηθεί ένα MAC επίπεδο ρυθμισμένο ως AP, με τον τύπο βοηθού να υποδηλώνει ότι το `Attribute` "QosSupported" πρέπει να τεθεί ως ψευδές - απενεργοποιώντας την υποστήριξη τύπου 802.11e/WMM-style QoS στα AP που θα δημιουργηθούν.

Οι επόμενες γραμμές δημιουργούν ένα μοναδικό AP το οποίο μοιράζεται το ίδιο σύνολο από `Attributes` φυσικού επιπέδου (και καναλιού) με τους σταθμούς:

```
NetDeviceContainer apDevices;
apDevices = wifi.Install (phy, mac, wifiApNode);
```

Σε αυτό το σημείο θα προσθέσουμε τα μοντέλα κινητικότητάς μας. Θέλουμε οι STA σταθμοί να είναι κινητοί, περιπλανώμενοι μέσα στα πλαίσια ενός περιοριστικού κουτιού, και θέλουμε να κάνουμε τον AP κόμβο σταθερό. Χρησιμοποιούμε τον `MobilityHelper` για να διευκολυνθούμε. Αρχικά, δημιουργούμε ένα αντικείμενο `MobilityHelper` και θέτουμε κάποια `Attributes` που ελέγχουν τη λειτουργία του «κατανεμητή θέσεων» (position allocator).

```
MobilityHelper mobility;

mobility.SetPositionAllocator ("ns3::GridPositionAllocator",
  "MinX", DoubleValue (0.0),
  "MinY", DoubleValue (0.0),
  "DeltaX", DoubleValue (5.0),
  "DeltaY", DoubleValue (10.0),
  "GridWidth", UintegerValue (3),
  "LayoutType", StringValue ("RowFirst"));
```

Αυτός ο κώδικας λέει στον βοηθό κινητικότητας να χρησιμοποιήσει ένα δισδιάστατο πλέγμα για να τοποθετήσει αρχικά τους STA κόμβους. Εξερευνήστε ελεύθερα το Doxygen ψάχνοντας για την κλάση `ns3::GridPositionAllocator` για να δείτε ακριβώς τι γίνεται.

Έχουμε τοποθετήσει τους κόμβους μας στο αρχικό πλέγμα, αλλά τώρα πρέπει να τους πούμε πώς να κινηθούν. Επιλέγουμε το `RandomWalk2dMobilityModel`, το οποίο βάζει τους κόμβους να κινούνται προς μια τυχαία κατεύθυνση, με τυχαία ταχύτητα, μέσα σε ένα οριοθετημένο κουτί.

```
mobility.SetMobilityModel ("ns3::RandomWalk2dMobilityModel",
  "Bounds", RectangleValue (Rectangle (-50, 50, -50, 50)));
```

Τώρα λέμε στο `MobilityHelper` να εγκαταστήσει τα μοντέλα κινητικότητας στους STA κόμβους.

```
mobility.Install (wifiStaNodes);
```

Θέλουμε το σημείο πρόσβασης να παραμείνει σε μια καθορισμένη θέση κατά τη διάρκεια της προσομοίωσης. Αυτό το επιτυγχάνουμε θέτοντας ως μοντέλο κινητικότητας για αυτόν τον κόμβο το `ns3::ConstantPositionMobilityModel`:

```
mobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");
mobility.Install (wifiApNode);
```

Τώρα πλέον έχουμε δημιουργήσει τους κόμβους μας, τις συσκευές και τα κανάλια μας, έχουμε επιλέξει τα μοντέλα κινητικότητας για τους Wifi κόμβους, αλλά δεν έχουμε καμία στοίβα πρωτοκόλλου. Ακριβώς όπως κάναμε και πολλές φορές πριν, θα χρησιμοποιήσουμε τον `InternetStackHelper` για να εγκαταστήσουμε αυτές τις στοίβες.

```
InternetStackHelper stack;
stack.Install (csmaNodes);
stack.Install (wifiApNode);
stack.Install (wifiStaNodes);
```

Όπως και στο παράδειγμα του `second.cc`, θα χρησιμοποιήσουμε τον `Ipv4AddressHelper` για να αναθέσουμε IP διευθύνσεις στις διεπαφές των συσκευών μας. Αρχικά θα χρησιμοποιήσουμε το δίκτυο 10.1.1.0 για να δημιουργήσουμε τις δύο διευθύνσεις που χρειαζονται οι δύο συσκευές μας σημείου-προς-σημείο. Έπειτα χρησιμοποιούμε το δίκτυο 10.1.2.0 για να αναθέσουμε διευθύνσεις στο CSMA δίκτυο, και έπειτα αναθέτουμε διευθύνσεις από το δίκτυο 10.1.3.0 τόσο στις STA συσκευές όσο και στην AP συσκευή στο ασύρματο δίκτυο.

```
Ipv4AddressHelper address;

address.SetBase ("10.1.1.0", "255.255.255.0");
Ipv4InterfaceContainer p2pInterfaces;
p2pInterfaces = address.Assign (p2pDevices);

address.SetBase ("10.1.2.0", "255.255.255.0");
Ipv4InterfaceContainer csmaInterfaces;
csmaInterfaces = address.Assign (csmaDevices);

address.SetBase ("10.1.3.0", "255.255.255.0");
address.Assign (staDevices);
address.Assign (apDevices);
```

Τοποθετούμε τον εξυπηρετητή echo στον «δεξιότερο» κόμβο της απεικόνισης στην αρχή του αρχείου. Το έχουμε κάνει και πιο πριν αυτό.

```
UdpEchoServerHelper echoServer (9);

ApplicationContainer serverApps = echoServer.Install (csmaNodes.Get (nCsma));
serverApps.Start (Seconds (1.0));
serverApps.Stop (Seconds (10.0));
```

Τοποθετούμε και τον πελάτη echo στον τελευταίο STA κόμβο που δημιουργήσαμε, κατευθύνοντάς τον προς τον εξυπηρετητή του CSMA δικτύου. Έχουμε επίσης δει παρόμοιες λειτουργίες και παλιότερα.

```
UdpEchoClientHelper echoClient (csmaInterfaces.GetAddress (nCsma), 9);
echoClient.SetAttribute ("MaxPackets", UintegerValue (1));
echoClient.SetAttribute ("Interval", TimeValue (Seconds (1.0)));
echoClient.SetAttribute ("PacketSize", UintegerValue (1024));

ApplicationContainer clientApps =
  echoClient.Install (wifiStaNodes.Get (nWifi - 1));
clientApps.Start (Seconds (2.0));
clientApps.Stop (Seconds (10.0));
```

Αφότου έχουμε χτίσει ένα διαδίκτυο εδώ, χρειάζεται να ενεργοποιήσουμε τη δρομολόγηση διαδικτύου όπως κάναμε και στο σενάριο του παραδείγματος στο `second.cc`.

```
Ipv4GlobalRoutingHelper::PopulateRoutingTables ();
```

Ένα πράγμα που μπορεί να εκπλήσσει κάποιους χρήστες είναι το γεγονός ότι η προσομοίωση που μόλις δημιουργήσαμε δε θα σταματήσει ποτέ «εκ των πραγμάτων». Αυτό οφείλεται στο ότι ζητήσαμε από το ασύρματο σημείο πρόσβασης να παράγει beacons. Θα παράγει beacons για πάντα, και αυτό θα έχει ως αποτέλεσμα να προγραμματίζονται ασταμάτητα γεγονότα προσομοίωσης για το μέλλον, οπότε πρέπει να πούμε στον προσομοιωτή να σταματήσει, παρόλο που μπορεί να έχουν προγραμματιστεί γεγονότα δημιουργίας beacon. Η ακόλουθη γραμμή κώδικα λέει στον προσομοιωτή να σταματήσει έτσι ώστε να μην προσομοιώνουμε beacons για πάντα, μπαίνοντας με αυτόν τον τρόπο σε έναν κατ' ουσίαν ατέρμονα βρόχο.

```
Simulator::Stop (Seconds (10.0));
```

Δημιουργούμε τόσα ίχνη καταγραφής έτσι ώστε να καλύψουμε και τα τρία δίκτυα:

```
pointToPoint.EnablePcapAll ("third");
phy.EnablePcap ("third", apDevices.Get (0));
csma.EnablePcap ("third", csmaDevices.Get (0), true);
```

Αυτές οι τρεις γραμμές κώδικα θα ξεκινήσουν την καταγραφή pcap και στους δύο κόμβους σημείου-προς-σημείο που λειτουργούν ως η ραχοκοκαλιά μας, θα αρχίσουν μια καταγραφή μεικτής κατάστασης στο Wifi δίκτυο, και μια μεικτή καταγραφή στο CSMA δίκτυο. Αυτό θα μας επιτρέψει να δούμε όλη την κίνηση με τη βοήθεια του ελάχιστου αριθμού αρχείων ιχνών.

Τέλος, τρέχουμε όντως την προσομοίωση, καθαρίζουμε και έπειτα βγαίνουμε από το πρόγραμμα.

```
  Simulator::Run ();
  Simulator::Destroy ();
  return 0;
}
```

Για να τρέξετε αυτό το παράδειγμα, θα πρέπει να αντιγράψετε το σενάριο `third.cc` στον κατάλογο scratch και να χρησιμοποιήσετε το Waf για να κάνετε build, όπως κάνατε και στο παράδειγμα `second.cc`. Εάν είστε στον κατάλογο του υψηλότερου επιπέδου του αποθετηρίου, θα πληκτρολογήσετε,

```
$ cp examples/tutorial/third.cc scratch/mythird.cc
$ ./waf
$ ./waf --run scratch/mythird
```

Ξανά, από τη στιγμή που έχετε θέσει τις εφαρμογές UDP echo όπως κάναμε στο σενάριο `second.cc`, θα δείτε μία παρόμοια έξοδο.

```
Waf: Entering directory `/home/craigdo/repos/ns-3-allinone/ns-3-dev/build'
Waf: Leaving directory `/home/craigdo/repos/ns-3-allinone/ns-3-dev/build'
'build' finished successfully (0.407s)
At time 2s client sent 1024 bytes to 10.1.2.4 port 9
At time 2.01796s server received 1024 bytes from 10.1.3.3 port 49153
At time 2.01796s server sent 1024 bytes to 10.1.3.3 port 49153
At time 2.03364s client received 1024 bytes from 10.1.2.4 port 9
```

Θυμηθείτε ότι το πρώτο μήνυμα, "`Sent 1024 bytes to 10.1.2.4`", είναι ο πελάτης UDP echo που στέλνει ένα πακέτο στον εξυπηρετητή. Σε αυτή την περίπτωση, ο πελάτης είναι στο ασύρματο δίκτυο (10.1.3.0). Το δεύτερο μήνυμα, "`Received 1024 bytes from 10.1.3.3`", είναι από τον UDP echo εξυπηρετητή, και δημιουργήθηκε όταν αυτός έλαβε το echo πακέτο. Το τελικό μήνυμα, "`Received 1024 bytes from 10.1.2.4`", είναι από τον πελάτη echo, και δείχνει ότι αυτός έχει λάβει το echo πακέτο του πίσω από τον εξυπηρετητή.

Εάν τώρα πάτε και δείτε στον κατάλογο του υψηλότερου επιπέδου, θα βρείτε τέσσερα αρχεία ιχνών από την προσομοίωση, δύο από τον κόμβο μηδέν και δύο από τον κόμβο ένα:

```
third-0-0.pcap  third-0-1.pcap  third-1-0.pcap  third-1-1.pcap
```

Το αρχείο "third-0-0.pcap" αντιστοιχεί στη συσκευή σημείου-προς-σημείο στον κόμβο μηδέν – στην αριστερή πλευρά της «ραχοκοκαλιάς». Το αρχείο "third-1-0.pcap" αντιστοιχεί στη συσκευή σημείου-προς-σημείο στον κόμβο ένα – στην δεξιά πλευρά της "ραχοκοκαλιάς". Το αρχείο "third-0-1.pcap" θα είναι το μεικτό (κατάσταση παρακολούθησης) ίχνος από το Wifi δίκτυο και το αρχείο "third-1-1.pcap" θα είναι το μεικτό ίχνος από το CSMA δίκτυο. Μπορείτε να το επιβεβαιώσετε αυτό εξετάζοντας τον κώδικα;

Από τη στιγμή που ο echo πελάτης είναι στο Wifi δίκτυο, ας αρχίσουμε από εκεί. Ας δούμε στο μεικτό (σε κατάσταση παρακολούθησης) ίχνος που καταγράψαμε σε αυτό το δίκτυο.

```
$ tcpdump -nn -tt -r third-0-1.pcap
```

Θα πρέπει να δείτε κάποια περιεχόμενα σχετικά με το Wifi που δεν έχετε ξαναδεί προηγουμένως:

```
reading from file third-0-1.pcap, link-type IEEE802_11 (802.11)
0.000025 Beacon (ns-3-ssid) [6.0* 9.0 12.0 18.0 24.0 36.0 48.0 54.0 Mbit] IBSS
0.000308 Assoc Request (ns-3-ssid) [6.0 9.0 12.0 18.0 24.0 36.0 48.0 54.0 Mbit]
0.000324 Acknowledgment RA:00:00:00:00:00:08
0.000402 Assoc Response AID(0) :: Successful
0.000546 Acknowledgment RA:00:00:00:00:00:0a
0.000721 Assoc Request (ns-3-ssid) [6.0 9.0 12.0 18.0 24.0 36.0 48.0 54.0 Mbit]
0.000737 Acknowledgment RA:00:00:00:00:00:07
0.000824 Assoc Response AID(0) :: Successful
0.000968 Acknowledgment RA:00:00:00:00:00:0a
0.001134 Assoc Request (ns-3-ssid) [6.0 9.0 12.0 18.0 24.0 36.0 48.0 54.0 Mbit]
0.001150 Acknowledgment RA:00:00:00:00:00:09
0.001273 Assoc Response AID(0) :: Successful
0.001417 Acknowledgment RA:00:00:00:00:00:0a
0.102400 Beacon (ns-3-ssid) [6.0* 9.0 12.0 18.0 24.0 36.0 48.0 54.0 Mbit] IBSS
0.204800 Beacon (ns-3-ssid) [6.0* 9.0 12.0 18.0 24.0 36.0 48.0 54.0 Mbit] IBSS
0.307200 Beacon (ns-3-ssid) [6.0* 9.0 12.0 18.0 24.0 36.0 48.0 54.0 Mbit] IBSS
```

Μπορείτε να δείτε ότι ο τύπος σύνδεσης είναι τώρα ο 802.11, όπως θα περιμένατε. Μπορείτε πιθανώς να καταλάβετε τι γίνεται και να βρείτε τα πακέτα του IP echo αιτήματος και της απάντησης σε αυτό το ίχνος. Την πλήρη ανάλυση των ιχνών αυτών σας την αφήνουμε ως άσκηση.

Τώρα, δείτε στο αρχείο pcap της αριστερής πλευράς του συνδέσμου σημείου-προς-σημείο,

```
$ tcpdump -nn -tt -r third-0-0.pcap
```

Ξανά, θα δείτε μερικά γνώριμα περιεχόμενα:

```
reading from file third-0-0.pcap, link-type PPP (PPP)
2.008151 IP 10.1.3.3.49153 > 10.1.2.4.9: UDP, length 1024
2.026758 IP 10.1.2.4.9 > 10.1.3.3.49153: UDP, length 1024
```

Αυτό είναι το echo πακέτο που πηγαίνει από αριστερά προς τα δεξιά (από το Wifi στο CSMA) και ξανά πίσω διαμέσου του συνδέσμου σημείου-προς-σημείο.

Τώρα, δείτε στο αρχείο pcap της δεξιάς πλευράς του συνδέσμου σημείου-προς-σημείο,

```
$ tcpdump -nn -tt -r third-1-0.pcap
```

Ξανά, θα δείτε μερικά γνώριμα περιεχόμενα:

```
reading from file third-1-0.pcap, link-type PPP (PPP)
2.011837 IP 10.1.3.3.49153 > 10.1.2.4.9: UDP, length 1024
2.023072 IP 10.1.2.4.9 > 10.1.3.3.49153: UDP, length 1024
```

Αυτό είναι επίσης το echo πακέτο που πηγαίνει από τα αριστερά προς τα δεξιά (από το Wifi στο CSMA) και πάλι πίσω διαμέσου του συνδέσμου σημείου-προς-σημείο, με λίγο διαφορετικούς χρονισμούς όπως πιθανόν να αναμένατε.

Ο echo εξυπηρετητής βρίσκεται στο CSMA δίκτυο, οπότε ας ρίξουμε μια ματιά στο μεικτό ίχνος εκεί:

```
$ tcpdump -nn -tt -r third-1-1.pcap
```

Θα πρέπει να βλέπετε μερικά γνώριμα περιεχόμενα:

```
reading from file third-1-1.pcap, link-type EN10MB (Ethernet)
2.017837 ARP, Request who-has 10.1.2.4 (ff:ff:ff:ff:ff:ff) tell 10.1.2.1, length 50
2.017861 ARP, Reply 10.1.2.4 is-at 00:00:00:00:00:06, length 50
```

```
2.017861 IP 10.1.3.3.49153 > 10.1.2.4.9: UDP, length 1024
2.022966 ARP, Request who-has 10.1.2.1 (ff:ff:ff:ff:ff:ff) tell 10.1.2.4, length 50
2.022966 ARP, Reply 10.1.2.1 is-at 00:00:00:00:00:03, length 50
2.023072 IP 10.1.2.4.9 > 10.1.3.3.49153: UDP, length 1024
```

Αυτό θα πρέπει να είναι εύκολα κατανοητό. Εάν έχετε ξεχάσει τι και πώς, επιστρέψτε πίσω και κοιτάξτε στα όσα είπαμε στο παράδειγμα `second.cc`. Είναι η ίδια ακολουθία.

Τώρα, αφιερώσαμε αρκετό χρόνο καθορίζοντας τα μοντέλα κινητικότητας για το ασύρματο δίκτυο και έτσι θα ήταν κρίμα να κλείσουμε χωρίς καν να δείξουμε ότι όντως οι STA κόμβοι κινούνται κατά τη διάρκεια της προσομοίωσης. Ας το κάνουμε αυτό εξετάζοντας τον πηγαίο κώδικα της καταγραφής των αλλαγών πορείας του `MobilityModel`. Πρόκειται απλά για μια γρήγορη ματιά στην λεπτομερή ενότητα σχετικά με την ιχνηλασία που ακολουθεί μετέπειτα, μα το σημείο αυτό φαίνεται ιδανικό για να δούμε ένα σχετικό παράδειγμα.

Όπως αναφέρθηκε στην ενότητα «Μικρορυθμίσεις», το σύστημα ιχνηλασίας του *ns-3* χωρίζεται σε πηγές ιχνηλασίας (trace source) και καταβόθρες ιχνηλασίας (trace sinks), και εμείς παρέχουμε μεθόδους για τη σύνδεση αυτών των δύο. Θα χρησιμοποιήσουμε την προκαθορισμένη πηγή ιχνηλασίας των αλλαγών πορείας για το μοντέλο κινητικότητας ώστε να πυροδοτήσουμε τα γεγονότα ιχνηλασίας. Θα χρειαστεί να συντάξουμε μια καταβόθρα ιχνηλασίας ώστε να τη συνδέσουμε με την πηγή, η οποία θα μας εμφανίζει μερικές ωραίες πληροφορίες. Παρά τη φήμη περί δυσκολίας αυτού του πράγματος, στην πραγματικότητα είναι εξαιρετικά απλό. Απλά πριν το κυρίως πρόγραμμα του σεναρίου `scratch/mythird.cc` (π.χ. αμέσως μετά τη δήλωση `NS_LOG_COMPONENT_DEFINE`), προσθέστε την ακόλουθη συνάρτηση:

```
void
CourseChange (std::string context, Ptr<const MobilityModel> model)
{
  Vector position = model->GetPosition ();
  NS_LOG_UNCOND (context <<
    " x = " << position.x << ", y = " << position.y);
}
```

Αυτός ο κώδικας τραβάει τις πληροφορίες τοποθεσίας από το μοντέλο κινητικότητας και καταγράφει άνευ όρων τις συντεταγμένες x και y του κόμβου. Θα τα ρυθμίσουμε έτσι ώστε αυτή η μέθοδος να καλείται κάθε φορά που ο ασύρματος κόμβος με τον echo πελάτη αλλάζει θέση. Αυτό το πετυχαίνουμε με τη χρήση της συνάρτησης `Config::Connect`. Προσθέστε τις ακόλουθες γραμμές κώδικα στο σενάριο, ακριβώς πριν από την κλήση `Simulator::Run`.

```
std::ostringstream oss;
oss <<
  "/NodeList/" << wifiStaNodes.Get (nWifi - 1)->GetId () <<
  "/$ns3::MobilityModel/CourseChange";

Config::Connect (oss.str (), MakeCallback (&CourseChange));
```

Αυτό που κάνουμε εδώ είναι ότι δημιουργούμε μια ακολουθία που περιέχει το μονοπάτι του χώρου ονομάτων που αφορά στην ιχνηλασία του γεγονότος στο οποίο θέλουμε να συνδεθούμε. Αρχικά, πρέπει να καταλάβουμε ποιος είναι ο κόμβος που θέλουμε, χρησιμοποιώντας τη μέθοδο `GetId` όπως περιγράφθηκε νωρίτερα. Στην περίπτωση του εξ ορισμού αριθμού CSMA και ασύρματων κόμβων, προκύπτει ότι ο ζητούμενος κόμβος είναι ο κόμβος επτά και το μονοπάτι του χώρου ονομάτων που αφορά στην ιχνηλασία για το μοντέλο κινητικότητας θα μοιάζει κάπως έτσι:

```
/NodeList/7/$ns3::MobilityModel/CourseChange
```

Με βάση τα όσα είπαμε στην ενότητα περί ιχνηλασίας, μπορείτε να παρέμβετε έτσι ώστε το μονοπάτι ιχνηλασίας να αναφέρεται στον έβδομο κόμβο της καθολικής NodeList. Αυτό ορίζει κάτι το οποίο αποκαλείται ενσωματωμένο (aggregated) αντικείμενο τύπου `ns3::MobilityModel`. Το πρόθεμα του δολαρίου υποδηλώνει ότι το MobilityModel είναι ενσωματωμένο στον κόμβο επτά. Το τελευταιό μέρος του μονοπατιού σημαίνει ότι βρισκόμαστε στο γεγονός “CourseChange” αυτού του μοντέλου.

Κάνουμε τη σύνδεση μεταξύ της πηγής ιχνηλασίας στον κόμβο επτά με την καταβόθρα ιχνηλασίας μας, καλώντας την `Config::Connect` και περνώντας ως όρισμα το μονοπάτι του χώρου ονομάτων. Μόλις γίνει αυτό, κάθε γεγονός αλλαγής πορείας στον κόμβο επτά θα καταλήγει στην καταβόθρα ιχνηλασίας μας, η οποία με τη σειρά της θα εκτυπώνει τη νέα θέση.

Εάν τώρα τρέξετε την προσομοίωση, θα δείτε ότι οι αλλαγές πορείας εμφανίζονται καθώς συμβαίνουν.

```
'build' finished successfully (5.989s)
/NodeList/7/$ns3::MobilityModel/CourseChange x = 10, y = 0
/NodeList/7/$ns3::MobilityModel/CourseChange x = 10.3841, y = 0.923277
/NodeList/7/$ns3::MobilityModel/CourseChange x = 10.2049, y = 1.90708
/NodeList/7/$ns3::MobilityModel/CourseChange x = 10.8136, y = 1.11368
/NodeList/7/$ns3::MobilityModel/CourseChange x = 10.8452, y = 2.11318
/NodeList/7/$ns3::MobilityModel/CourseChange x = 10.9797, y = 3.10409
At time 2s client sent 1024 bytes to 10.1.2.4 port 9
At time 2.01796s server received 1024 bytes from 10.1.3.3 port 49153
At time 2.01796s server sent 1024 bytes to 10.1.3.3 port 49153
At time 2.03364s client received 1024 bytes from 10.1.2.4 port 9
/NodeList/7/$ns3::MobilityModel/CourseChange x = 11.3273, y = 4.04175
/NodeList/7/$ns3::MobilityModel/CourseChange x = 12.013, y = 4.76955
/NodeList/7/$ns3::MobilityModel/CourseChange x = 12.4317, y = 5.67771
/NodeList/7/$ns3::MobilityModel/CourseChange x = 11.4607, y = 5.91681
/NodeList/7/$ns3::MobilityModel/CourseChange x = 12.0155, y = 6.74878
/NodeList/7/$ns3::MobilityModel/CourseChange x = 13.0076, y = 6.62336
/NodeList/7/$ns3::MobilityModel/CourseChange x = 12.6285, y = 5.698
/NodeList/7/$ns3::MobilityModel/CourseChange x = 13.32, y = 4.97559
/NodeList/7/$ns3::MobilityModel/CourseChange x = 13.1134, y = 3.99715
/NodeList/7/$ns3::MobilityModel/CourseChange x = 13.8359, y = 4.68851
/NodeList/7/$ns3::MobilityModel/CourseChange x = 13.5953, y = 3.71789
/NodeList/7/$ns3::MobilityModel/CourseChange x = 12.7595, y = 4.26688
/NodeList/7/$ns3::MobilityModel/CourseChange x = 11.7629, y = 4.34913
/NodeList/7/$ns3::MobilityModel/CourseChange x = 11.2292, y = 5.19485
/NodeList/7/$ns3::MobilityModel/CourseChange x = 10.2344, y = 5.09394
/NodeList/7/$ns3::MobilityModel/CourseChange x = 9.3601, y = 4.60846
/NodeList/7/$ns3::MobilityModel/CourseChange x = 8.40025, y = 4.32795
/NodeList/7/$ns3::MobilityModel/CourseChange x = 9.14292, y = 4.99761
/NodeList/7/$ns3::MobilityModel/CourseChange x = 9.08299, y = 5.99581
/NodeList/7/$ns3::MobilityModel/CourseChange x = 8.26068, y = 5.42677
/NodeList/7/$ns3::MobilityModel/CourseChange x = 8.35917, y = 6.42191
/NodeList/7/$ns3::MobilityModel/CourseChange x = 7.66805, y = 7.14466
/NodeList/7/$ns3::MobilityModel/CourseChange x = 6.71414, y = 6.84456
/NodeList/7/$ns3::MobilityModel/CourseChange x = 6.42489, y = 7.80181
```

ΚΕΦΑΛΑΙΟ

# ΕΠΤΑ

# ΙΧΝΗΛΑΣΙΑ

## 7.1 Ιστορικό

Όπως αναφέρεται στο *Χρησιμοποιώντας το Σύστημα Ιχνηλασίας*, το νόημα της λειτουργίας μιας προσομοίωσης *ns-3* είναι να παράγει έξοδο για μελέτη. Έχετε δύο βασικές στρατηγικές για την απόκτηση εξόδου από *ns-3*: τη χρήση γενικών μαζικών μηχανισμών παραγωγής και την ανάλυση του περιεχομένου τους για να εξαχθούν ενδιαφέρουσες πληροφορίες, ή με κάποιο τρόπο την ανάπτυξη ενός μηχανισμού εξόδου που αποπνέει ακριβώς (και ίσως μόνο) τις ζητούμενες πληροφορίες.

Η χρήση μαζικών μηχανισμών εξόδου έχει το πλεονέκτημα ότι δεν απαιτεί αλλαγές στον *ns-3*, αλλά μπορεί να απαιτήσει τη συγγραφή σεναρίων για την ανάλυση και το φιλτράρισμα των δεδομένων του ενδιαφέροντος. Συχνά, PCAP ή `NS_LOG` μηνύματα εξόδου που συγκεντρώνονται κατά τη διάρκεια προσομοιώσεων τρέχουν και ξεχωριστά τρέχουν μέσα από σενάρια που χρησιμοποιούν `grep`, `sed` ή `awk` για να αναλύσουν τα μηνύματα και να μειώσουν και να μετατρέψουν τα δεδομένα σε μία διαχειρίσιμη μορφή. Τα προγράμματα πρέπει να είναι γραμμένα ώστε να μετατρέπονται, έτσι αυτό δεν έρχεται δωρεάν. Η έξοδος του `NS_LOG` δεν θεωρείται μέρος του *ns-3* API, και μπορεί να αλλάξει χωρίς προειδοποίηση μεταξύ των εκδόσεων. Επιπλέον, η έξοδος του `NS_LOG` είναι διαθέσιμη μόνο σε εκδόσεις εντοπισμού σφαλμάτων, έτσι επικαλούμενη επιβάλλει ποινή απόδοσης. Φυσικά, αν η πληροφορία που ενδιαφέρει δεν υπάρχει σε κανένα από τους προκαθορισμένους μηχανισμούς εξόδου, η προσέγγιση αυτή αποτυγχάνει.

Εάν χρειάζεστε να προσθέσετε μερικές εξειδικευμένες πληροφορίες στους μαζικούς μηχανισμούς, αυτό σίγουρα μπορεί να γίνει΄ και αν χρησιμοποιήσετε έναν από τους *ns-3* μηχανισμούς, μπορείτε να πάρετε τον κώδικά σας προστιθέμενο ως εισφορά.

Ο *ns-3* παρέχει έναν άλλο μηχανισμό, που ονομάζεται Tracing, ο οποίος αποφεύγει ορισμένα από τα προβλήματα που συνδέονται με τους μαζικούς μηχανισμούς εξόδου. Έχει αρκετά σημαντικά πλεονεκτήματα. Κατ ‘αρχάς, μπορείτε να μειώσετε την ποσότητα των δεδομένων που θα πρέπει να διαχειριστείτε από μόνο τον εντοπισμό των εκδηλώσεων που σας ενδιαφέρουν (για τις μεγάλες προσομοιώσεις, τοποθετώντας τα πάντα στο δίσκο για την μετα-επεξεργασία μπορεί να δημιουργήσει I / O σημεία συμφόρησης). Δεύτερον, αν χρησιμοποιείτε αυτή τη μέθοδο, μπορείτε να ελέγξετε τη μορφοποίηση της εξόδου άμεσα, έτσι ώστε να αποφευχθεί το στάδιο της μετα-επεξεργασίας με `sed`, `awk`, `perl` ή `python` σενάρια. Αν επιθυμείτε, η παραγωγή σας μπορεί να διαμορφωθεί άμεσα σε μορφή αποδεκτή από gnuplot, για παράδειγμα (βλέπε επίσης *GnuplotHelper*). Μπορείτε να προσθέσετε άγκιστρα στον πυρήνα που μπορούν να προσπελαστούν από άλλους χρήστες, αλλά οι οποίες δε θα παράγουν καμία πληροφορία εκτός αν σας ζητηθεί ρητά να το πράξετε. Για αυτούς τους λόγους, πιστεύουμε ότι το σύστημα ανίχνευσης *ns-3* είναι ο καλύτερος τρόπος για να πάρετε πληροφορίες από μια προσομοίωση και είναι επίσης, ως εκ τούτου ένας από τους πιο σημαντικούς μηχανισμούς για να καταλάβουμε τον *ns-3*.

### 7.1.1 Εξειδικευμένα εργαλεία

Υπάρχουν πολλοί τρόποι για να πάρετε πληροφορίες από ένα πρόγραμμα. Ο πιο απλός τρόπος είναι να εκτυπώσετε μόνο τις πληροφορίες απευθείας στην κανονική έξοδο, όπως και στην

```
#include <iostream>
...
void
SomeFunction (void)
{
  uint32_t x = SOME_INTERESTING_VALUE;
  ...
  std::cout << "The value of x is " << x << std::endl;
  ...
}
```

Κανείς δεν πρόκειται να σας αποτρέψει από το να πηγαίνετε βαθιά στον πυρήνα του *ns-3* και προσθέτοντας τις δηλώσεις εκτύπωσης. Αυτό είναι τρομερά εύκολο να γίνει και μετά, έχετε τον πλήρη έλεγχο του δικού σας υποκατάστημα *ns-3*. Αυτό πιθανόν να μην αποδειχθεί ότι είναι ικανοποιητικό σε μακροπρόθεσμη βάση, όμως.

Καθώς ο αριθμός των καταστάσεων εκτύπωσης αυξάνει στα προγράμματά σας, το έργο της αντιμετώπισης του μεγάλου αριθμού των αποτελεσμάτων θα γίνονται όλο και περισσότερο περίπλοκα. Τελικά, μπορεί να αισθανθείτε την ανάγκη να ελέγχετε ό,τι πληροφορίες εκτυπώνετε με κάποιο τρόπο, ίσως ενεργοποιώντας και απενεργοποιώντας ορισμένες κατηγορίες εκτυπώσεις, ή αυξάνοντας ή μειώνοντας το την ποσότητα των πληροφοριών που θέλετε. Εάν συνεχίσουμε αυτήν την πορεία σας μπορεί να ανακαλύψετε ότι έχετε εκ νέου σε εφαρμογή το μηχανισμό `NS_LOG` (βλέπε *Χρησιμοποιώντας την Ενότητα Καταγραφής*). Προκειμένου να αποφευχθεί αυτό, ένα από τα πρώτα πράγματα που θα μπορούσατε να εξετάσετε είναι να χρησιμοποιείσετε μόνο του το `NS_LOG`.

Μας αναφέρθηκε παραπάνω ότι ένας τρόπος για να πάρετε πληροφορίες από τον *ns-3* είναι να αναλύσει τις υπάρχουσες εξόδους `NS_LOG` για ενδιαφέρουσες πληροφορίες. Αν ανακαλύψετε ότι χρειάζεστε κάποια εξειδικευμένη πληροφορία που δεν είναι παρών σε υφιστάμενες εξόδους αρχείων καταγραφής, μπορείτε να επεξεργαστείτε τον πυρήνα του *ns-3* και απλά προσθέστε ενδιαφέρουσες πληροφορίες σας στη ροή εξόδου. Τώρα, αυτό είναι σίγουρα καλύτερο από την προσθήκη των δικών σας δηλώσεων εκτύπωσης, δεδομένου ότι ακολουθεί ο *ns-3* συμβάσεις κωδικοποίησης και θα μπορούσε ενδεχομένως να είναι χρήσιμο σε άλλους ανθρώπους ως ένα patch στον υφιστάμενο πυρήνα.

Ας πάρουμε ένα τυχαίο παράδειγμα. Αν θέλετε να προσθέσετε περισσότερη καταγραφή στον *ns-3* υποδοχή TCP (`tcp-socket-base.cc`) θα μπορούσατε απλά να προσθέσετε ένα νέο μήνυμα κάτω στην εφαρμογή. Σημειώστε ότι στο `TcpSocketBase::ReceivedAck()` δεν υπάρχει log μήνυμα για την περίπτωση του no ACK. Θα μπορούσατε απλά να προσθέσετε ένα, αλλάζοντας τον κώδικα. Εδώ είναι η αρχική

```
/** Process the newly received ACK */
void
TcpSocketBase::ReceivedAck (Ptr<Packet> packet, const TcpHeader& tcpHeader)
{
  NS_LOG_FUNCTION (this << tcpHeader);

  // Received ACK. Compare the ACK number against highest unacked seqno
  if (0 == (tcpHeader.GetFlags () & TcpHeader::ACK))
    { // Ignore if no ACK flag
    }
  ...
```

Για να συνδεθείτε στην περίπτωση του no ACK, μπορείτε να προσθέσετε ένα νέο " NS_LOG_LOGIC" στο " if" σώμα δήλωσης

```
/** Process the newly received ACK */
void
TcpSocketBase::ReceivedAck (Ptr<Packet> packet, const TcpHeader& tcpHeader)
{
  NS_LOG_FUNCTION (this << tcpHeader);

  // Received ACK. Compare the ACK number against highest unacked seqno
  if (0 == (tcpHeader.GetFlags () & TcpHeader::ACK))
```

```
    { // Ignore if no ACK flag
      NS_LOG_LOGIC ("TcpSocketBase " << this << " no ACK flag");
    }
  ...
```

Αυτό μπορεί να φαίνεται αρκετά απλό και ικανοποιητικό με την πρώτη ματιά, αλλά κάτι που πρέπει να δούμε είναι ότι θα πρέπει να γράψετε κώδικα για να προσθέσετε δηλώσεις `NS_LOG` και θα πρέπει επίσης να γράψετε κώδικα (όπως στο `grep`, `sed` ή `awk` σενάρια) για να αναλύσει το αρχείο καταγραφής εξόδου, προκειμένου να απομονώσουν τα στοιχεία σας. Αυτό οφείλεται στο γεγονός ότι, ακόμη και αν έχετε κάποιο έλεγχο πάνω στο τι είναι η έξοδος από το σύστημα καταγραφής, έχετε μόνο τον έλεγχο στο συγκεκριμένο επίπεδο log, το οποίο είναι συνήθως ένα ολόκληρο αρχείο πηγαίου κώδικα.

Αν θέλετε να προσθέσετε κώδικα σε μια υπάρχουσα μονάδα, θα πρέπει επίσης να ακολουθείτε την έξοδο που κάθε άλλος προγραμματιστής έχει βρει ενδιαφέρουσα. Μπορείτε να διαπιστώσετε ότι, προκειμένου να πάρετε τις λίγες πληροφορίες που χρειάζεστε, μπορεί να χρειαστείτε να εντρυφήσετε μέσα από την τεράστια ποσότητα μηνυμάτων που προέρχονται από ξένα μηνύματα που δεν παρουσιάζουν κανένα ενδιαφέρον για εσάς. Μπορεί να αναγκαστείτε να αποθηκεύσετε τεράστια αρχεία καταγραφής στο δίσκο και να τα επεξεργαστείτε με σκοπό να κάνετε την δουλειά σας.

Δεδομένου ότι δεν υπάρχουν εγγυήσεις στον *ns-3* σχετικά με τη σταθερότητα της εξόδου `NS_LOG`, μπορείτε επίσης να ανακαλύψετε ότι τα κομμάτια της παραγωγής εξόδου τα οποία είναι για εξαφάνιση ή για αλλαγή μεταξύ διαφορετικών εκδόσεων. Αν εξαρτάστε στη δομή της παραγωγής, μπορείτε να βρείτε και άλλα μηνύματα που προστίθενται ή διαγράφονται τα οποία μπορεί να επηρεάσουν την ανάλυση του κώδικα.

Τέλος, η έξοδος `NS_LOG` είναι διαθέσιμη μόνο σε εκδόσεις εντοπισμού σφαλμάτων, δεν μπορείτε να πάρετε συνδεθείτε εξόδου από βελτιστοποιημένη χτίζει, που τρέχουν περίπου δύο φορές πιο γρήγορα. Στηριζόμενη στην `NS_LOG` επιβάλλει ποινή απόδοσης.

Για τους λόγους αυτούς, θεωρούμε τις εκτυπώσεις στο `std::cout` και τα μηνύματα `NS_LOG` να είναι γρήγορα και απλοί τρόποι για να πάρετε περισσότερες πληροφορίες από τον *ns-3*, αλλά δεν είναι κατάλληλο για σοβαρή δουλειά.

Είναι επιθυμητό να έχουμε μια σταθερή εγκατάσταση, χρησιμοποιώντας σταθερά APIs που επιτρέπουν σε κάποιον να φτάσει στον πυρήνα του συστήματος και να πάρει μόνο τις πληροφορίες που απαιτούνται. Είναι επιθυμητό να είναι σε θέση να το κάνει αυτό χωρίς να χρειάζεται να αλλάξει και να μεταγλωττίσει ξανά τον πυρήνα του συστήματος. Ακόμα καλύτερα θα είναι ένα σύστημα που κοινοποίησε τον κωδικό χρήστη, όταν ένα στοιχείο του ενδιαφέροντος αλλάξει ή μια ενδιαφέρουσα εκδήλωση έγινε έτσι ο χρήστης θα έχει αυτά που του χρειάζονται.

Το σύστημα εντοπισμού του *ns-3* έχει σχεδιαστεί για να λειτουργεί προς αυτή την κατεύθυνση και είναι καλά ενσωματωμένο με το *Attribute* και *Config* υποσυστήματα που επιτρέπουν την σχετικά απλή χρήση σεναρίων.

## 7.2 Επισκόπηση

Το σύστημα ανίχνευσης του *ns-3* είναι χτισμένο στις έννοιες των ανεξάρτητων απο τον εντοπισμό πηγών και τον εντοπισμό συλλεκτών, μαζί με ένα ενιαίο μηχανισμό για τη σύνδεση πηγών σε συλλέκτες.

Πηγές εντοπισμού είναι οντότητες που μπορούν να σηματοδοτήσουν τα γεγονότα που συμβαίνουν σε μια προσομοίωση και να παρέχουν πρόσβαση σε ενδιαφέροντα υποκείμενα δεδομένα. Για παράδειγμα, μια πηγή ίχνους θα μπορούσε να υποδείξει πότε ένα πακέτο παραλαμβάνεται από μια συσκευή δικτύου και να παρέχει πρόσβαση στα περιεχόμενα του πακέτου για τους ενδιαφερόμενους συλλέκτες εντοπισμού. Μια πηγή ίχνους μπορεί επίσης να αναφέρει πότε μια ενδιαφέρουσα αλλαγή κατάστασης συμβαίνει σε ένα μοντέλο. Για παράδειγμα, το παράθυρο συμφόρησης του μοντέλου TCP είναι πρώτος υποψήφιος για μια πηγή ίχνους. Κάθε φορά που αλλάζει το παράθυρο συμφόρησης που είναι συνδεδεμένο με συλλέκτες ίχνους ενημερώνεται με την παλαιά και νέα τιμή.

Οι πηγές εντοπισμού δεν είναι χρήσιμες από μόνες τους. Θα πρέπει να συνδεθούν με άλλα κομμάτια του κώδικα που κάνουν πραγματικά κάτι χρήσιμο με τις πληροφορίες που παρέχονται από την πηγή. Οι οντότητες που

καταναλώνουν πληροφορίες ίχνους ονομάζονται συλλέκτες ίχνους. Οι πηγές εντοπισμού είναι γεννήτριες των δεδομένων και οι συλλέκτες ίχνους είναι οι καταναλωτές. Αυτή η ρητή κατανομή επιτρέπει για ένα μεγάλο αριθμό πηγών ίχνους να είναι διάσπαρτα σε όλο το σύστημα σε χώρους που συγγραφείς μοντέλων πιστεύουν ότι μπορεί να είναι χρήσιμο. Η τοποθέτηση πηγών ίχνους εισάγει μία πολύ μικρή γενικά εκτέλεση.

Μπορεί να υπάρχουν μηδέν ή περισσότεροι καταναλωτές από ίχνη γεγονότων που παράγεται από μια πηγή ίχνους. Κάποιος μπορεί να σκεφτεί μια πηγή ίχνους, ως ένα είδος point-to-multipoint σύνδεσης πληροφοριών. Ο κώδικάς σας ψάχνει για ίχνη γεγονότων από ένα συγκεκριμένο κομμάτι του πηγαίου κώδικα, θα μπορούσε ευτυχώς να συνυπάρχει με άλλους κώδικες να κάνει κάτι εντελώς διαφορετικό από την ίδια πληροφορία.

Αν ένας χρήστης δεν συνδέσει ένα συλλέκτη ίχνους σε μια από αυτές τις πηγές, τίποτα δεν θα υπάρχει στην έξοδο. Με τη χρήση του συστήματος εντοπισμού, τόσο εσείς όσο και άλλοι άνθρωποι είναι συνδεδεμένοι με την ίδια πηγή ίχνους παίρνουν ακριβώς αυτό που θέλουν και μόνο ό,τι θέλουν έξω από το σύστημα. Ούτε εσείς επηρεάζετε κάθε άλλο χρήστη αλλάζοντας ποιά πληροφορία είναι έξοδος από το σύστημα. Αν συμβεί να προσθέσετε μια πηγή ίχνους, το έργο σας ως καλός πολίτης ανοικτού κώδικα μπορεί να επιτρέψει σε άλλους χρήστες για την παροχή νέων υπηρεσιών κοινής ωφελείας που είναι ίσως πολύ χρήσιμο συνολικά, χωρίς να κάνει οποιεσδήποτε αλλαγές στον πυρήνα *ns-3*.

### 7.2.1 Απλό Παράδειγμα

Ας πάρουμε μερικά λεπτά και βήμα βήμα ακολουθήστε ένα απλό παράδειγμα εντοπισμού. Θα χρειαστείτε την Επανάκληση να καταλάβετε τι συμβαίνει στο παράδειγμα, οπότε πρέπει να πάρετε μια μικρή παράκαμψη αμέσως.

#### Επανάκληση

Ο στόχος του συστήματος επανάκλησης *ns-3* είναι να επιτρέψει σε ένα κομμάτι του κώδικα να καλέσει μια συνάρτηση (ή μέθοδο σε C++), χωρίς καμία συγκεκριμένη μεταξύ των μονάδων εξάρτηση. Αυτό σημαίνει ότι, τελικά, θα πρέπει να έχετε κάποιο είδος εμμεσότητας – αντιμετωπίζεις τη διεύθυνση της κληθήσας συνάρτησης ως μια μεταβλητή. Η μεταβλητή αυτή ονομάζεται μεταβλητή(pointer-to-function). Η σχέση μεταξύ της συνάρτησης και της μεταβλητής(pointer-to-function) πραγματικά δεν διαφέρει από αυτήν του αντικειμένου(object) και του δείκτη προς το αντικείμενο(pointer-to-object).

Στη C, το κανονικό παράδειγμα της μεταβλητής(pointer-to-function) δείκτη-συνάρτησης είναι ένας δείκτης-σε-συνάρτηση-επιστρέφοντας-ακέραιο (PFI - pointer-to-function-returning-integer). Λαμβάνοντας μία παράμετρο `int`, όπως,

```
int (*pfi)(int arg) = 0;
```

(Αλλά διαβάστε το C++-FAQ Section 33 πριν συντάξετε κώδικα σαν αυτόν!) Αυτό που μπορείτε να πάρετε από αυτό είναι μια μεταβλητή που ονομάζεται απλά `pfi` που έχει την τιμή 0. Αν θέλετε να αρχικοποιήσετε αυτό το δείκτη σε κάτι σημαντικό, θα πρέπει να έχετε μια συνάρτηση με μια υπογραφή που να ταιριάζουν. Σε αυτήν την περίπτωση, θα μπορείτε να προσφέρετε μια συνάρτηση που μοιάζει με

```
int MyFunction (int arg) {}
```

Εάν έχετε αυτό το στόχο, μπορείτε να προετοιμάσετε τη μεταβλητή στο σημείο της συνάρτησή σας

```
pfi = MyFunction;
```

Στη συνέχεια μπορείτε να καλέσετε την MyFunction έμμεσα χρησιμοποιώντας την πιο υποβλητική μορφή της κλήσης

```
int result = (*pfi) (1234);
```

Αυτό είναι ενδεικτικό, απο τη στιγμη που διαφοροποιήστε στο δείκτη συνάρτησης ακριβώς όπως θα κάνατε την διαφορά στον κάθε δείκτη. Συνήθως, όμως, οι άνθρωποι θα επωφεληθούν από το γεγονός ότι ο μεταγλωττιστής(compiler) ξέρει τι συμβαίνει και θα χρησιμοποιήσει μόνο μια μικρότερη μορφή

```
int result = pfi (1234);
```

Αυτό μοιάζει σαν να καλείτε μια συνάρτηση που ονομάζεται `pfi`, αλλά ο μεταγλωττιστής(compiler) είναι αρκετά έξυπνος για να ξέρει να καλέσει μέσω της μεταβλητής `pfi` έμμεσα τη συνάρτηση `MyFunction`.

Θεωρητικά, αυτό είναι σχεδόν ακριβώς πώς λειτουργεί το σύστημα εντοπισμού. Βασικά, ένα ίχνος καταβόθρας *είναι* μια επανάκληση. Όταν ένα ίχνος καταβόθρας(trace sink) εκφράζει ενδιαφέρον λαμβάνοντας γεγονότα ίχνους, η ίδια προσθέτει ως επανάκληση σε έναν κατάλογο Επανακλήσεων εσωτερικά διατηρημένα από την πηγή ίχνους. Όταν μια ενδιαφέρουσα εκδήλωση συμβαίνει, η πηγή ίχνους επικαλείται τον χειριστή της `operator(...)` παρέχοντας μηδέν ή περισσότερα ορίσματα. Ο χειριστής `operator(...)` περιπλανιέται τελικά κάτω στο σύστημα και κάνει κάτι σημαντικό όπως η έμμεση κλήση που μόλις είδατε, παρέχοντας μηδέν ή περισσότερες παραμέτρους, έτσι όπως ακριβώς και η κλήση για `pfi` παραπάνω περάσει μία παράμετρο για την συνάρτηση στόχο `MyFunction`.

Η σημαντική διαφορά ότι το σύστημα εντοπισμού προσθέτει, είναι ότι για κάθε πηγή ίχνους υπάρχει μια εσωτερική λίστα Επανακλήσεων. Αντί να κάνουμε απλώς μια έμμεση κλήση, μια πηγή ίχνους μπορεί να καλέσει πολλές Επανακλήσεις. Όταν μία καταβόθρα ίχνους εκφράζει το ενδιαφέρον σε ειδοποιήσεις από μια πηγή ίχνους, ουσιαστικά φροντίζει μόνο να προσθέσει τη δική του συνάρτηση στη λίστα επανάκλησης.

Εάν ενδιαφέρεστε για περισσότερες λεπτομέρειες σχετικά με το πώς είναι πραγματικά τοποθετημένα στον *ns-3*, μη διστάσετε να μελετήσετε την ενότητα επανάκλησης του Εγχειριδίου(Manual) *ns-3*.

### Οδηγός Διασύνδεσης: `fourth.cc`

Έχουμε προβάλει κάποιο κώδικα για να εφαρμόσουμε αυτό που είναι πραγματικά το πιο απλό παράδειγμα του εντοπισμού που μπορεί να συναρμολογηθεί. Μπορείτε να βρείτε τον κώδικα σε αυτό τον κατάλογο ως `fourth.cc`. Ας δούμε μέσα από αυτό

```
/* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
/*
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License version 2 as
 * published by the Free Software Foundation;
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
 */

#include "ns3/object.h"
#include "ns3/uinteger.h"
#include "ns3/traced-value.h"
#include "ns3/trace-source-accessor.h"

#include <iostream>

using namespace ns3;
```

Το μεγαλύτερο μέρος αυτού του κώδικα θα πρέπει να είναι αρκετά γνωστό σε εσάς. Όπως αναφέρθηκε παραπάνω, το σύστημα ίχνους κάνει βαριά χρήση του Αντικειμένου και του Χαρακτηριστικού στα συστήματα( Object and Attribute systems), έτσι θα πρέπει να τα συμπεριλάβετε. Τα δύο πρώτα περιεχόμενα φέρουν πάνω σε δηλώσεις για τα συστήματα αυτά ρητά. Θα μπορούσατε να χρησιμοποιήσετε τον πυρήνα κεφαλίδα ενότητας για να πάρετε τα πάντα με τη μία, αλλά κάνουμε τα περιεχόμενα ρητά εδώ για να επεξηγήσουμε πόσο πραγματικά απλό είναι αυτό όλο.

Το αρχείο, `traced-value.h` φέρνει τις απαιτούμενες δηλώσεις για τον εντοπισμό των δεδομένων που υπακούει στη σημασιολογική αξία. Σε γενικές γραμμές, η σημασιολογική αξία ακριβώς σημαίνει ότι μπορείτε να περάσετε το ίδιο το αντικείμενο γύρω, αντί να μεταθέσετε τη διεύθυνση του αντικειμένου. Αυτό σημαίνει πραγματικά ότι θα είστε σε θέση να εντοπίζετε όλες τις αλλαγές που γίνονται σε ένα TracedValue σε ένα πολύ απλό τρόπο.

Δεδομένου ότι το σύστημα ανίχνευσης είναι ενσωματωμένο με Χαρακτηριστικά, και τα Χαρακτηριστικά δουλεύουν με Αντικείμενα, πρέπει να υπάρχει ένας *ns-3* `Object` για την πηγή ίχνους που υπάρχει. Το επόμενο απόσπασμα κώδικα δηλώνει και ορίζει ένα απλό Αντικείμενο που μπορούμε να εργαστούμε.

```
class MyObject : public Object
{
public:
  static TypeId GetTypeId (void)
  {
    static TypeId tid = TypeId ("MyObject")
      .SetParent (Object::GetTypeId ())
      .AddConstructor<MyObject> ()
      .AddTraceSource ("MyInteger",
                       "An integer value to trace.",
                       MakeTraceSourceAccessor (&MyObject::m_myInt),
                       "ns3::Traced::Value::Int32Callback")
      ;
    return tid;
  }

  MyObject () {}
  TracedValue<int32_t> m_myInt;
};
```

Οι δύο σημαντικές γραμμές κώδικα, παραπάνω, σε σχέση με τον εντοπισμό είναι το `.AddTraceSource` και η `TracedValue` δήλωση `m_myInt`.

Το `.AddTraceSource` παρέχει τα "άγκιστρα" που χρησιμοποιούνται για τη σύνδεση της πηγής ίχνους με τον έξω κόσμο μέσω του συστήματος Config. Το πρώτο όρισμα είναι ένα όνομα για αυτήν την πηγή ίχνους, το οποίο το καθιστά ορατό στο σύστημα Config. Το δεύτερο όρισμα είναι μία βοήθεια απο string. Τώρα κοιτάξτε τον τρίτο όρισμα, στην πραγματικότητα εστίασε στο *όρισμα* από το τρίτο όρισμα: `&MyObject::m_myInt`. Αυτή είναι η TracedValue η οποία προστίθεται στην κλάση(class), είναι πάντα ένα μέλος κλάσης δεδομένων. (Το τελευταίο όρισμα είναι το όνομα `typedef` για τον τύπο TracedValue, ως συμβολοσειρά(string). Αυτό χρησιμοποιείται για να δημιουργήσετε τεκμηρίωση για τη σωστή υπογραφή Επανάκλησης συνάρτησης, η οποία είναι χρήσιμη ειδικά για πιο γενικούς τύπους Επανακλήσεων.)

Η δήλωση `TracedValue<>` παρέχει την υποδομή που οδηγεί την διαδικασία επανάκλησης. Κάθε φορά που η υποκείμενη αξία είναι αλλαγμένη ο μηχανισμός TracedValue θα παρέχει τόσο την παλαιά όσο και την νέα τιμή της μεταβλητής, σε αυτή την περίπτωση μία αξία `int32_t`. Η συνάρτηση της καταβόθρας ίχνους για το TracedValue θα χρειαστεί την υπογραφή

```
void (* TracedValueCallback)(const int32_t oldValue, const int32_t newValue);
```

Όλες οι καταβόθρες ίχνους συνδέοντας αυτή την πηγή ίχνους πρέπει να έχουν αυτή την υπογραφή. Θα συζητήσουμε παρακάτω πώς μπορείτε να προσδιορίσετε την απαιτούμενη υπογραφή επανάκλησης σε άλλες περιπτώσεις.

Συνεχίζοντας με το `fourth.cc` βλέπουμε

```
void
IntTrace (int32_t oldValue, int32_t newValue)
{
  std::cout << "Traced " << oldValue << " to " << newValue << std::endl;
}
```

Αυτός είναι ο ορισμός μιάς καταβόθρας ίχνους. Αντιστοιχεί άμεσα με την υπογραφή της συνάρτησης επανάκλησης. Μόλις συνδεθεί, η συνάρτηση αυτή θα καλείται όταν το `TracedValue` αλλάξει.

Έχουμε δει τώρα την πηγή ίχνους και την καταβόθρα ίχνους. Αυτό που απομένει είναι ο κώδικας να συνδέσει την πηγή στην καταβόθρα, η οποία συμβαίνει στο `main`

```
int
main (int argc, char *argv[])
{
  Ptr<MyObject> myObject = CreateObject<MyObject> ();
  myObject->TraceConnectWithoutContext ("MyInteger", MakeCallback(&IntTrace));

  myObject->m_myInt = 1234;
}
```

Εδώ εμείς πρώτα δημιουργούμε το παράδειγμα MyObject στο οποίο η πηγή ίχνους υπάρχει.

Το επόμενο βήμα, το `TraceConnectWithoutContext`, αποτελεί τη σύνδεση μεταξύ της πηγής ίχνους και της καταβόθρας ίχνους. Το πρώτο όρισμα είναι ακριβώς το όνομα της πηγής ίχνους "MyInteger", είδαμε παραπάνω. Παρατήρησε την συνάρτηση πρότυπο `MakeCallback`. Αυτή η συνάρτηση κάνει τη λειτουργία που απαιτείται για να δημιουργήσει το υποκείμενο Αντικείμενο επανάκλησης *ns-3* και το συνδέουν με την συνάρτηση `IntTrace`. Το `TraceConnect` κάνει την σχέση μεταξύ της παρεχόμενης συνάρτησής σας και υπερφορτωμένα `operator()` στην εντοπισμένη μεταβλητή που αναφέρεται από το Χαρακτηριστικό "MyInteger". Μετά από αυτή την ένωση, η πηγή ίχνους θα πάρει "φωτιά" στην παρεχόμενη συνάρτηση επανάκλησης.

Ο κώδικας για να κάνει όλα αυτά να συμβούν είναι, φυσικά, μη-τετριμμένο, αλλά η ουσία είναι ότι οργανώνετε για κάτι που μοιάζει ακριβώς όπως το παράδειγμα παραπάνω `pfi()` να κληθεί από την πηγή ίχνους. Η δήλωση του `TracedValue<int32_t> m_myInt;` στο ίδιο το Αντικείμενο εκτελεί τη λειτουργία που απαιτείται για την παροχή των υπερφορτωμένων τελεστών ανάθεσης που θα χρησιμοποιήσει ο `operator()` για να επικαλεστεί πραγματικά την επανάκληση με τις επιθυμητές παραμέτρους. Το `.AddTraceSource` εκτελεί τη λειτουργία για να συνδέσετε την Επανάκληση στο σύστημα Config, και το " TraceConnectWithoutContext" εκτελεί τη λειτουργία για να συνδέσετε τη συνάρτησή σας με την πηγή ίχνους, η οποία καθορίζεται με βάση το όνομα του Χαρακτηριστικού.

Ας αγνοήσουμε για λίγο το κομμάτι σχετικά με το περιεχόμενο.

Τέλος, η γραμμή αποδίδοντας μία αξία σε `m_myInt`

```
myObject->m_myInt = 1234;
```

θα πρέπει να ερμηνευθεί ως επίκληση του `operator=` για τη μεταβλητή μέλους `m_myInt` με τον ακέραιο `1234` πέρασε ως μία παράμετρος.

Από τη στιγμή που το `m_myInt` είναι ένα `TracedValue`, ο φορέας αυτός ορίζεται να εκτελέσει μία επανάκληση που επιστρέφει κενό και παίρνει δύο ακέραιες τιμές ως παραμέτρους — μια παλιά τιμή και μια νέα τιμή για τον εν λόγω ακέραιο. Αυτή είναι ακριβώς η υπογραφή συνάρτηση για την συνάρτηση επανάκλησης που παρείχαμε — `IntTrace`.

Για να συνοψίσουμε, μια πηγή ίχνους είναι, στην ουσία, μια μεταβλητή που κρατά μια λίστα επανακλήσεων(callbacks). Μία καταβόθρα ίχνους είναι μια συνάρτηση που χρησιμοποιείται ως στόχος της επανάκλησης. Τα συστήματα πληροφόρησης τύπου Χαρακτηριστικό και Αντικείμενο χρησιμοποιούνται για να παρέχουν έναν τρόπο για να συνδέσετε πηγές ίχνους για τον εντοπισμό καταβόθρων. Η ενέργεια "χτυπήματος" μιάς πηγής ίχνους εκτελείται σε ένα φορέα στην πηγή ίχνους που εκτοξεύει επανακλήσεις. Αυτά τα αποτελέσματα επανακλήσεων

στη καταβόθρα ίχνους τα οποία καταχωρούν ενδιαφέρον στην πηγή καλούνται με τις παραμέτρους που παρέχονται από την πηγή.

Αν τώρα οικοδομήσουμε και να τρέξουμε αυτό το παράδειγμα,

```
$ ./waf --run fourth
```

θα δείτε την έξοδο από την συνάρτηση `IntTrace` να εκτελεί το συντομότερο δυνατό η πηγή ίχνους χτύπημα:

```
Traced 0 to 1234
```

Όταν έχουμε την εκτέλεση του κώδικα, `myObject->m_myInt = 1234;`, η πηγή ίχνους εκτελείται γρήγορα και παρέχει αυτόματα τις τιμές πριν και μετά στη καταβόθρα ίχνους. Η συνάρτηση `IntTrace` στη συνέχεια εκτύπωσε αυτό στην κανονική έξοδο.

### 7.2.2 Σύνδεση με Config

Η κλήση του `TraceConnectWithoutContext` φαίνεται στο παραπάνω απλό παράδειγμα το οποίο χρησιμοποιείται στην πραγματικότητα πολύ σπάνια στο σύστημα. Πιο τυπικά, το υποσύστημα `Config` χρησιμοποιείται για να επιλέξετε μια πηγή ίχνους στο σύστημα, χρησιμοποιώντας αυτό που ονομάζεται *Config path*. Είδαμε ένα παράδειγμα απο αυτό στην προηγούμενη ενότητα, όπου γαντζώθηκε στην "CourseChange" εκδήλωση, όταν πειραματιζόμασταν με το `third.cc`.

Υπενθυμίζουμε ότι ορίσαμε μία καταβόθρα ίχνους για να εκτυπώσουμε πληροφορίες και να αλλάξει πληροφορίες απο την κινητικότητα των μοντέλων απο την προσομοίωσή μας. Θα πρέπει τώρα να είναι πολύ πιο σαφές σε σας τι κάνει αυτή η συνάρτηση

```
void
CourseChange (std::string context, Ptr<const MobilityModel> model)
{
  Vector position = model->GetPosition ();
  NS_LOG_UNCOND (context <<
    " x = " << position.x << ", y = " << position.y);
}
```

Όταν συνδέσαμε την πηγή ίχνους "CourseChange" στην παραπάνω καταβόθρα ίχνους, χρησιμοποιήσαμε ένα Config path για να προσδιορίσουμε την πηγή όταν κανονίσαμε μια σύνδεση μεταξύ της προ-καθορισμένης πηγής ίχνους και της νέας καταβόθρας ίχνους

```
std::ostringstream oss;
oss << "/NodeList/"
    << wifiStaNodes.Get (nWifi - 1)->GetId ()
    << "/$ns3::MobilityModel/CourseChange";

Config::Connect (oss.str (), MakeCallback (&CourseChange));
```

Ας προσπαθήσουμε να καταλάβουμε του τι θεωρείται μερικές φορές σχετικά μυστηριώδης κώδικας. Για τους σκοπούς της συζήτησης, ας υποθέσουμε ότι ο αριθμός κόμβου(Node) που επιστρέφεται από το `GetId ()` είναι "7". Σε αυτήν την περίπτωση, η διαδρομή ανωτέρω αποδεικνύεται ότι είναι

```
"/NodeList/7/$ns3::MobilityModel/CourseChange"
```

Το τελευταίο τμήμα της διαδρομής config path πρέπει να είναι `Attribute` ενός `Object`. Στην πραγματικότητα, αν είχατε πρακτικά ένα δείκτη στο `Object` που έχει το "CourseChange" `Attribute`, θα μπορούσατε να γράψετε αυτό, ακριβώς όπως κάναμε στο προηγούμενο παράδειγμα. Τυπικά αποθηκεύουμε δείκτες στους (κόμβους) `Nodes` σε ένα NodeContainer. Στο παράδειγμα `third.cc`, οι κόμβοι του ενδιαφέροντος είναι αποθηκευμένοι στο NodeContainer `wifiStaNodes`. Στην πραγματικότητα, βάζοντας τη διαδρομή μαζί, χρησιμοποιήσαμε αυτό το container για να πάρει ένα `Ptr<Node>` που είχαμε συνηθίσει να λέμε `GetId()`. Θα μπορούσαμε να χρησιμοποιήσουμε αυτό το `Ptr<Node>` να καλέσει άμεσα μια μέθοδο Σύνδεσης

```
Ptr<Object> theObject = wifiStaNodes.Get (nWifi - 1);
theObject->TraceConnectWithoutContext ("CourseChange", MakeCallback (&CourseChange));
```

Στο παράδειγμα `third.cc`, θέλαμε πραγματικά ένα πρόσθετο "περιεχόμενο" για να παραδοθεί μαζί με τις παραμέτρους Επανάκλησης (η οποία θα εξηγηθεί παρακάτω) έτσι θα μπορούσαμε να χρησιμοποιήσουμε πραγματικά τον ακόλουθο ισοδύναμο κώδικα

```
Ptr<Object> theObject = wifiStaNodes.Get (nWifi - 1);
theObject->TraceConnect ("CourseChange", MakeCallback (&CourseChange));
```

Αυτό μας δίνει ότι ο εσωτερικός κώδικας για `Config::ConnectWithoutContext` και `Config::Connect` βρήκε πραγματικά μια `Ptr<Object>` και κάλεσε την κατάλληλη μέθοδο `TraceConnect` στο χαμηλότερο επίπεδο .

Οι συναρτήσεις του `Config` λαμβάνουν μια διαδρομή που αντιπροσωπεύει μια αλυσίδα από δείκτες `Object`. Κάθε τμήμα του μονοπατιού απαντάει σε ένα Χαρακτηριστικό Αντικείμενο. Το τελευταίο τμήμα είναι το Χαρακτηριστικό του ενδιαφέροντος, και πριν από τα τμήματα πρέπει να είναι τυπωμένα να περιέχουν ή να βρούν Αντικείμενα. Ο κώδικας `Config` αναλύει και "περπατάει" αυτό το μονοπάτι μέχρι να φτάσει στο τελικό τμήμα της διαδρομής. Στη συνέχεια ερμηνεύει το τελευταίο τμήμα ως `Attribute` στο τελευταίο Αντικείμενο που βρέθηκε, περπατώντας τη διαδρομή. Οι συναρτήσεις `Config` τότε καλούν την κατάλληλη `TraceConnect` ή μέθοδο `TraceConnectWithoutContext` για το τελικό Αντικείμενο. Ας δούμε τι θα συμβεί σε λίγο με περισσότερες λεπτομέρειες, όταν η παραπάνω διαδρομή περπάτησε.

Η κορυφαία χαρακτήρας "/" στη διαδρομή αναφέρεται σε ένα λεγόμενο namespace. Μία από τις προκαθορισμένες namespaces στο σύστημα config είναι "NodeList", η οποία είναι μια λίστα με όλους τους κόμβους στην προσομοίωση. Τα στοιχεία στην λίστα αναφέρονται στους δείκτες της λίστας, έτσι "/ NodeList / 7" αναφέρεται στον όγδοο Κόμβο στη λίστα των κόμβων που δημιουργούνται κατά τη διάρκεια της προσομοίωσης (ανάκληση δεικτών ξεκινούν από το *0'). Αυτή η αναφορά είναι στην πραγματικότητα ένα "Ptr<Node>'* και έτσι είναι μια υποκατηγορία ενός `ns3::Object`.

Όπως περιγράφεται στην ενότητα Object Model της *ns-3* Εγχειρίδιο(Manual), κάνουμε εκτεταμένη χρήση της συνάθροισης αντικειμένου. Αυτό μας επιτρέπει να σχηματίσουμε τη σύνδεση μεταξύ διαφορετικών Αντικειμένων χωρίς την οικοδόμηση ενός δέντρου πολύπλοκης κληρονομικότητας ή να προ-αποφασίσουμε ποια αντικείμενα θα είναι μέρος ενός Κόμβου. Κάθε Αντικείμενο σε ένα σύνολο μπορεί να επιτευχθεί από τα άλλα Αντικείμενα.

Στο παράδειγμά μας, το επόμενο τμήμα της διαδρομής που έχει περπατηθεί ξεκινά με το χαρακτήρα "$". Αυτό δηλώνει στο σύστημα config ότι το τμήμα είναι το όνομα ενός τύπου Αντικειμένου, ώστε η κλήση `GetObject` θα πρέπει να ψάχνει για αυτό το είδος. Μας βγάζει ότι η `MobilityHelper` χρησιμοποιείται στο `third.cc` και κανονίζει στο σύνολο ή συνδέει, ένα μοντέλο κινητικότητας σε κάθε ασύρματο Κόμβο `Nodes`. Όταν προσθέσετε το "$" ρωτάτε για ένα άλλο Αντικείμενο που έχει πιθανώς προηγουμένως αθροιστεί. Μπορείτε να σκεφτείτε αυτό ως εναλλαγή δεικτών από την αρχική Ptr<Node>, όπως καθορίζεται από το "/NodeList/7" στο ίδιο συνδεδεμένο μοντέλο κινητικότητάς του — το οποίο είναι τύπου `ns3::MobilityModel`. Εάν είστε εξοικειωμένοι με το `GetObject`, ζητήσαμε από το σύστημα να κάνετε τα εξής

```
Ptr<MobilityModel> mobilityModel = node->GetObject<MobilityModel> ()
```

Είμαστε τώρα στο τελευταίο Αντικείμενο στο μονοπάτι, έτσι ώστε να στρέψουμε την προσοχή μας προς τα Χαρακτηριστικά αυτού του Αντικειμένου. Η κλάση `MobilityModel` ορίζει ένα Χαρακτηριστικό που ονομάζεται "CourseChange". Μπορείτε να δείτε αυτό κοιτάζοντας τον πηγαίο κώδικα σε `src/mobility/model/mobility-model.cc` και αναζητώντας για "CourseChange" στο αγαπημένο σας επεξεργαστή(editor). Θα πρέπει να βρείτε

```
.AddTraceSource ("CourseChange",
                 "The value of the position and/or velocity vector changed",
                 MakeTraceSourceAccessor (&MobilityModel::m_courseChangeTrace),
                 "ns3::MobilityModel::CourseChangeCallback")
```

το οποίο θα έπρεπε να φαίνεται εξοικειωμένο σε αυτό το σημείο.

Αν κοιτάξετε για την αντίστοιχη δήλωση της υποκείμενης εντοπισμένης μεταβλητής στο `mobility-model.h` θα βρείτε

```
TracedCallback<Ptr<const MobilityModel> > m_courseChangeTrace;
```

Ο τύπος της δήλωσης `TracedCallback` προσδιορίζει `m_courseChangeTrace` ως μία ειδική λίστα των Επανακλήσεων που μπορεί να συνδεθεί με τις συναρτήσεις Config που περιγράφονται παραπάνω. Το `typedef` για την υπογραφή συνάρτησης επανάκλησης είναι επίσης ορισμένο στο αρχείο κεφαλίδας

```
typedef void (* CourseChangeCallback)(Ptr<const MobilityModel> * model);
```

Η κλάση `MobilityModel` έχει σχεδιαστεί για να είναι μια βασική κλάση που παρέχει μια κοινή διεπαφή για όλες τις συγκεκριμένες υποκατηγορίες. Εάν κάνετε αναζήτηση προς τα κάτω στο τέλος του αρχείου, θα δείτε μια μέθοδο που λέγεται `NotifyCourseChange()`

```
void
MobilityModel::NotifyCourseChange (void) const
{
  m_courseChangeTrace(this);
}
```

Οι παραγόμενες κλάσεις θα καλέσουν σε αυτή τη μέθοδο κάθε φορά που κάνουν μια αλλαγή πορείας για την υποστήριξη ανίχνευσης. Η μέθοδος αυτή επικαλείται τον `operator()` σχετικά με το υποκείμενο `m_courseChangeTrace`, το οποίο, με τη σειρά του, επικαλείται το σύνολο των εγγεγραμμένων Επανακλήσεων, καλώντας όλες τις καταβόθρες ίχνους που έχουν εγγραφεί με ενδιαφέρον για την πηγή ίχνους καλώντας μία συνάρτηση Config.

Έτσι, στο παράδειγμα `third.cc` κοιτάξαμε, κάθε φορά που μια αλλαγή πορείας γίνεται σε μία απο τις εγκαταστημένες περιπτώσεις `RandomWalk2dMobilityModel`, θα υπάρχει μία κλήση `NotifyCourseChange()` που καλεί επάνω σε μία κλάση βάσης " MobilityModel". Όπως είδαμε παραπάνω, αυτό επικαλείται `operator()` στη `m_courseChangeTrace`, η οποία με τη σειρά της, καλεί οποιαδήποτε εγγεγραμμένη καταβόθρα ίχνους. Στο παράδειγμα, ο μοναδικός κώδικας, καταγράφοντας ένα ενδιαφέρον ήταν ο κώδικας που έδωσε το μονοπάτι Config. Ως εκ τούτου, η συνάρτηση `CourseChange` που γαντζώθηκε από τον αριθμό του κόμβου επτά θα είναι η μόνη που ονομάζεται Επανάκληση.

Το τελευταίο κομμάτι του παζλ είναι το "περιεχόμενο". Υπενθυμίζουμε ότι είδαμε μια έξοδο που αναζητά κάτι σαν το παρακάτω από το `third.cc`

```
/NodeList/7/$ns3::MobilityModel/CourseChange x = 7.27897, y =
2.22677
```

Το πρώτο μέρος της εξόδου είναι το περιεχόμενο. Είναι απλά η διαδρομή μέσω της οποίας ο κώδικας config βρίσκεται στη πηγή ίχνους. Στην περίπτωση που εμείς κοιτάξαμε ότι μπορεί να υπάρχει οποιοσδήποτε αριθμός των πηγών ίχνους στο σύστημα απαντά σε οποιονδήποτε αριθμό των κόμβων με τα μοντέλα κινητικότητας. Πρέπει να υπάρχει κάποιος τρόπος να προσδιορίσει ποια πηγή ίχνους είναι στην πραγματικότητα αυτός που έβαλε φωτιά στην Επανάκληση. Ο εύκολος τρόπος είναι να συνδέσετε με `Config::Connect`, αντί για `Config::ConnectWithoutContext`

### 7.2.3 Βρίσκοντας Πηγές

Το πρώτο ερώτημα που έρχεται αναπόφευκτα για τους νέους χρήστες του συστήματος Ανίχνευσης είναι, *"Εντάξει, ξέρω ότι πρέπει να υπάρχουν πηγές ίχνους στον πυρήνα προσομοίωσης, αλλά πώς μπορώ να μάθω τι ίχνους πηγές είναι διαθέσιμες για εμένα;"*

Το δεύτερο ερώτημα είναι, *"Εντάξει, βρήκα μια πηγή ίχνους, πώς μπορώ να καταλάβω το μονοπάτι Config για να χρησιμοποιήσω όταν συνδεθώ σε αυτό;"*

Το τρίτο ερώτημα είναι, *"Εντάξει, βρήκα μια πηγή ίχνους και το μονοπάτι Config, πώς μπορώ να καταλάβω ποιο είναι το είδος επιστροφής και ποια πρέπει να είναι τα επίσημα ορίσματα της συνάρτησης επανάκλησής μου;"*

Το τέταρτο ερώτημα είναι, *"Εντάξει, εγώ τα πληκτρολόγησα όλα σωστά και πήρα αυτό το απίστευτα παράξενο μήνυμα σφάλματος, τι στον κόσμο σημαίνει αυτό;"*

Θα τις απαντήσουμε κάθε μια από αυτές με τη σειρά.

### 7.2.4 Διαθέσιμες Πηγές

*Εντάξει, ξέρω ότι πρέπει να υπάρχουν πηγές ίχνους στον πυρήνα προσομοίωσης, αλλά πώς μπορώ να μάθω τι πηγές ίχνους είναι διαθέσιμες για εμένα;*

Η απάντηση στο πρώτο ερώτημα βρίσκεται στην τεκμηρίωση *ns-3* API. Αν πάτε στην ιστοσελίδα του έργου, ns-3 project, θα βρείτε ένα σύνδεσμο στο "Documentation" στη γραμμή πλοήγησης. Αν επιλέξετε αυτό το σύνδεσμο, θα μεταφερθείτε στη σελίδα τεκμηρίωσης. Υπάρχει μια σύνδεση με τα "Latest Release" που θα σας μεταφέρει στην τεκμηρίωση για την τελευταία σταθερή έκδοση του *ns-3*. Εάν επιλέξετε το σύνδεσμο "API Documentation", θα πρέπει να ληφθούν για την *ns-3* API σελίδα τεκμηρίωσης.

Στην πλαϊνή γραμμή θα πρέπει να δείτε μια ιεραρχία που αρχίζει

- ns-3
- ns-3 Documentation
- All TraceSources
- All Attributes
- All GlobalValues

Η λίστα που μας ενδιαφέρει είναι "All TraceSources". Προχωρήστε και επιλέξτε το σύνδεσμο. Θα δείτε, ίσως όχι και τόσο έκπληκτα, μια λίστα με όλες τις πηγές ίχνους διαθέσιμο στον *ns-3*.

Ως παράδειγμα, μετακινηθείτε προς τα κάτω για να `ns3::MobilityModel`. Θα βρείτε μια καταχώρηση για

```
CourseChange: The value of the position and/or velocity vector changed
```

Θα πρέπει να αναγνωρίσουμε αυτό ως πηγή ίχνους που χρησιμοποιείται στο παράδειγμα `third.cc`. Θα είναι χρήσιμη η περιεργασία της λίστας.

### 7.2.5 Μονοπάτια Config

*Εντάξει, βρήκα μια πηγή ίχνους, πώς μπορώ να καταλάβω την πορεία Config για να την χρησιμοποιήσω όταν συνδεθώ σε αυτή;*

Εάν γνωρίζετε ποιο αντικείμενο σας ενδιαφέρει, το τμήμα "Detailed Description" για την κλάση θα εμφανίσει όλες τις διαθέσιμες πηγές ίχνους. Για παράδειγμα, ξεκινώντας από τη λίστα των "All TraceSources", κάντε κλικ στο σύνδεσμο `ns3::MobilityModel`, το οποίο θα σας μεταφέρει στην τεκμηρίωση για την κλάση `MobilityModel`. Σχεδόν στην κορυφή της σελίδας είναι μία γραμμή σύντομης περιγραφής της κλάσης, που καταλήγει σε ένα σύνδεσμο "More...". Κάντε κλικ σε αυτό το σύνδεσμο για να παρακάμψετε την περίληψη API και πηγαίνετε στο "Detailed Description" της κλάσης. Στο τέλος της περιγραφής, θα είναι (έως) τρεις λίστες:

- **Config Paths**: μια λίστα των τυπικών διαδρομών Config για αυτή την κλάση.
- **Attributes** μια λίστα με όλα τα χαρακτηριστικά που παρέχονται από αυτή την κλάση.
- **TraceSources**: μια λίστα με όλα τα διαθέσιμα TraceSources από αυτή την κλάση.

Πρώτα θα συζητήσουμε τις διαδρομές Config.

Ας υποθέσουμε ότι έχετε μόλις βρεί την πηγή ίχνους "CourseChange" στη λίστα "All TraceSources" και θέλετε να βρείτε πώς να συνδεθείτε με αυτή. Ξέρετε ότι χρησιμοποιείτε (και πάλι, από το παράδειγμα `third.cc`) ένα `ns3::RandomWalk2dMobilityModel`. Έτσι, είτε να κάνετε κλικ στο όνομα της κλάσης στη λίστα "All

TraceSources", ή να βρείτε `ns3::RandomWalk2dMobilityModel` στην "Class List". Είτε έτσι είτε αλλιώς θα πρέπει τώρα να εξετάσουμε τη σελίδα "ns3::RandomWalk2dMobilityModel Class Reference".

Αν τώρα μετακινηθείτε προς τα κάτω στην ενότητα "Detailed Description", μετά από τον ενοποιημένη λίστα των μεθόδων και χαρακτηριστικών κλάσης (ή απλά κάντε κλικ στο σύνδεσμο "More..." στο τέλος της κλάσης σύντομη περιγραφή στο πάνω μέρος της σελίδας) θα δείτε τη συνολική τεκμηρίωση για την κλάση. Συνεχίζοντας προς τα κάτω, βρείτε τη λίστα "Config Paths":

> **Config Paths**
>
> `ns3::RandomWalk2dMobilityModel` is accessible through the following paths with `Config::Set` and `Config::Connect`:
>
> - "/NodeList/[i]/$ns3::MobilityModel/$ns3::RandomWalk2dMobilityModel"

Η τεκμηρίωση σας λέει πώς να φτάσετε στο Αντικείμενο `RandomWalk2dMobilityModel`. Συγκρίνετε τη σειρά πάνω από τη σειρά που πράγματι χρησιμοποιήσαμε στο παράδειγμα κώδικα

```
"/NodeList/7/$ns3::MobilityModel"
```

Η διαφορά αυτή οφείλεται στο γεγονός ότι οι δύο κλήσεις `GetObject` υπονοούν στη σειρά που βρίσκονται στην τεκμηρίωση. Η πρώτη, για `$ns3::MobilityModel` θα ζητήσει το σύνολα για την βάση της κλάσης. Η δεύτερη κλήση `GetObject`, για `$ns3::RandomWalk2dMobilityModel`, χρησιμοποιείται για να ρίχνει την βάση της κλάσης για την συγκεκριμένη κλάση εφαρμογής. Η τεκμηρίωση δείχνει τις δύο αυτές λειτουργίες για εσάς. Αποδεικνύεται ότι η πραγματική πηγή ίχνους που ψάχνετε βρίσκεται στη βάση της κλάσης.

Δείτε πιο κάτω στην ενότητα "Detailed Description" για τη λίστα των πηγών ίχνους. Θα βρείτε

> No TraceSources are defined for this type.
>
> **TraceSources defined in parent class "ns3::MobilityModel"**
>
> - **CourseChange**: The value of the position and/or velocity vector changed.
>
>   Callback signature: `ns3::MobilityModel::CourseChangeCallback`

Αυτό είναι ακριβώς ό,τι χρειάζεται να ξέρετε. Η πηγή ίχνους του ενδιαφέροντος βρίσκεται στο `ns3::MobilityModel` (που ξέρατε ούτως ή άλλως). Το ενδιαφέρον πράγμα σε αυτό το κομμάτι της τεκμηρίωσης του API λέει ότι δεν χρειάζονται επιπλέον απορρίμματα στο μονοπάτι config παραπάνω για να φτάσουμε στην συγκεκριμένη κλάση, δεδομένου ότι η πηγή ίχνους είναι στην πραγματικότητα στην βάση της κλάσης. Ως εκ τούτου, η πρόσθετη `GetObject` δεν απαιτείται και μπορείτε απλά να χρησιμοποιήσετε τη διαδρομή

```
"/NodeList/[i]/$ns3::MobilityModel"
```

που ταιριάζει απόλυτα με το παράδειγμα διαδρομή

```
"/NodeList/7/$ns3::MobilityModel"
```

Παρεμπιπτόντως, ένας άλλος τρόπος για να βρείτε το μονοπάτι Config είναι το `grep` γύρω στον κώδικα βάσης *ns-3* για κάποιον που έχει ήδη καταλάβει. Πρέπει πάντα να προσπαθείτε να αντιγράψετε τον κώδικα εργασίας κάποιου άλλου πριν ξεκινήσετε να γράφετε τα δικά σας. Δοκιμάστε κάτι σαν:

```
$ find . -name '*.cc' | xargs grep CourseChange | grep Connect
```

και μπορείτε να βρείτε την απάντησή σας, μαζί με τον κώδικα εργασίας. Για παράδειγμα, στην περίπτωση αυτή, `src/mobility/examples/main-random-topology.cc` έχει κάτι που σας περιμένει να χρησιμοποιήσετε

```
Config::Connect ("/NodeList/*/$ns3::MobilityModel/CourseChange",
  MakeCallback (&CourseChange));
```

Θα επανέλθω σε αυτό το παράδειγμα σε λίγο.

### 7.2.6 Στίγματα Επανάκλησης

*Εντάξει, βρήκα μια πηγή ίχνους και το μονοπάτι Config, πώς μπορώ να υπολογίσω ποιος είναι ο τύπος επιστροφής και τα επίσημα ορίσματα της συνάρτησης επανάκλησης μου;*

Ο ευκολότερος τρόπος είναι να εξετάσετε την υπογραφή επανάκλησης `typedef`, η οποία δίνεται στη "Callback signature" της πηγής ίχνους στο "Detailed Description" της κλάσης, όπως φαίνεται παραπάνω.

Η επανάληψη καταχώρησης πηγής ίχνους της "CourseChange" από `ns3::RandomWalk2dMobilityModel` έχουμε:

- **CourseChange**: The value of the position and/or velocity vector changed.

  Callback signature: `ns3::MobilityModel::CourseChangeCallback`

Η υπογραφή ή το στίγμα επανάκλησης δίνεται ως ένας σύνδεσμος με τη σχετική `typedef`, όπου βρίσκουμε

> `typedef void (* CourseChangeCallback)(const std::string context, Ptr<const MobilityModel> * model);`
>
> **TracedCallback** signature for course change notifications.
>
> Αν η επανάκληση συνδέεται με τη χρήση `ConnectWithoutContext` παραλείψτε το `context` όρισμα από το στίγμα.
>
> **Parameters**:
>
> [in] context The context string supplied by the Trace source.
> [in] model The MobilityModel which is changing course.

Όπως και παραπάνω, για να δείτε αυτό κατά τη χρήση `grep` γύρω στον κώδικα βάσης *ns-3* για παράδειγμα. Το παραπάνω παράδειγμα, από `src/mobility/examples/main-random-topology.cc`, συνδέει την πηγή ίχνους "CourseChange" στην συνάρτηση `CourseChange` στο ίδιο αρχείο

```
static void
CourseChange (std::string context, Ptr<const MobilityModel> model)
{
  ...
}
```

Σημειώστε ότι αυτή η συνάρτηση:

- Επιστρέφει `void`.

Αν, κατά τύχη, το στίγμα επανάκλησης δεν έχει τεκμηριωθεί, και δεν υπάρχουν παραδείγματα για εργασία, καθόρισε το σωστό στίγμα της συνάρτησης επανάκλησης που μπορεί να είναι, επίσης, δύσκολο πραγματικά να υπολογιστεί από τον πηγαίο κώδικα.

Πριν προβώντας σε μια περιδιάβαση του κώδικα, θα είμαι ευθύς και απλά να σας πω έναν απλό τρόπο για να το καταλάβετε: Η τιμή επιστροφής της επανάκλησής σας θα είναι πάντα `void`. Η επίσημη λίστα παραμέτρων για μια `TracedCallback` μπορεί να βρεθεί από τη λίστα της παραμέτρου προτύπου στη δήλωση. Υπενθυμίζεται ότι για το τρέχον παράδειγμά μας, αυτό είναι `mobility-model.h`, όπου βρήκαμε προηγουμένως

```
TracedCallback<Ptr<const MobilityModel> > m_courseChangeTrace;
```

Υπάρχει μια αντιστοιχία ένα-προς-ένα μεταξύ της λίστας προτύπου παραμέτρου στη δήλωση και τα επίσημα ορίσματα της συνάρτησης επανάκλησης. Εδώ, υπάρχει μια παράμετρος προτύπου, η οποία είναι ένα `Ptr<const MobilityModel>`. Αυτό σας λέει ότι χρειάζεστε μια συνάρτηση που επιστρέφει κενό και παίρνει ένα `Ptr<const MobilityModel>`. Για παράδειγμα

```
void
CourseChange (Ptr<const MobilityModel> model)
{
```

```
  ...
}
```

Αυτό είναι το μόνο που χρειάζεστε, αν θέλετε να `Config::ConnectWithoutContext`. Αν θέλετε ένα περιεχόμενο, θα πρέπει να `Config::Connect` και να χρησιμοποιήσετε μια συνάρτηση Επανάκλησης που παίρνει ένα περιεχόμενο string, τότε τα ορίσματα πρότυπα

```
void
CourseChange (const std::string context, Ptr<const MobilityModel> model)
{
  ...
}
```

Αν θέλετε να εξασφαλίσετε ότι η συνάρτησή σας `CourseChangeCallback` είναι ορατή μόνο σε τοπικό αρχείο σας, μπορείτε να προσθέσετε τη λέξη-κλειδί `static` και να καταλήξετε σε

```
static void
CourseChange (const std::string path, Ptr<const MobilityModel> model)
{
  ...
}
```

το οποίο είναι ακριβώς αυτό που χρησιμοποιείται στο παράδειγμα `third.cc`.

### Εφαρμογή

Αυτή η ενότητα είναι εντελώς προαιρετική. Πρόκειται να είναι μια δύσκολη διαδρομή, ειδικά για όσους δεν είναι εξοικειωμένοι με τις λεπτομέρειες των προτύπων. Ωστόσο, εάν μπορείτε να πάρετε μέσα από αυτό, θα έχετε μια πολύ καλή λαβή για πολλά από τα *ns-3* ιδιώματα χαμηλού επιπέδου.

Έτσι, και πάλι, ας υπολογίσουμε τι στίγμα της συνάρτησης επανάκλησης απαιτείται για την πηγή ίχνους "CourseChange". Αυτό πρόκειται να είναι οδυνηρό, αλλά χρειάζεται να το κάνετε αυτό μια φορά. Μετά μπορείτε να πάρετε μέσα από αυτό, θα είστε σε θέση να εξετάσετε ένα `TracedCallback` και να το κατανοήσετε.

Το πρώτο πράγμα που πρέπει να εξετάσουμε είναι η δήλωση της πηγής ίχνους. Θυμηθείτε ότι αυτό είναι `mobility-model.h`, όπου έχουμε διαπιστώσει στο παρελθόν

```
TracedCallback<Ptr<const MobilityModel> > m_courseChangeTrace;
```

Αυτή η δήλωση είναι για ένα πρότυπο. Η παράμετρος πρότυπο είναι μέσα στη παρένθεση, ώστε πραγματικά ενδιαφερόμαστε να ανακαλύψουμε τι είναι το `TracedCallback<>`. Αν δεν έχετε απολύτως καμία ιδέα για το πού αυτό θα μπορούσε να βρεθεί, το `grep` είναι ο φίλος σου.

Πρόκειται πιθανώς να ασχοληθούμε για κάποιο είδος δήλωσης της πηγής *ns-3*, οπότε πρώτη αλλαγή στον κατάλογο `src`. Στη συνέχεια, γνωρίζουμε ότι η δήλωση αυτή θα πρέπει να είναι σε κάποιο είδος του αρχείου header, έτσι απλά `grep` για να το χρησιμοποιείτε:

```
$ find . -name '*.h' | xargs grep TracedCallback
```

Θα δείτε 303 γραμμές να πετούν από (σας δείχνω αυτό μέσω `wc` για να δείτε πόσο άσχημο ήταν). Αν και αυτά μπορεί να φαίνονται πολλά, αλλά αυτά δεν είναι πραγματικά πολλά. Απλά διοχέτευσε την έξοδο μέσω `more` και ξεκινήστε τη σάρωση μέσα από αυτό. Στην πρώτη σελίδα, θα δείτε κάποια πολύ ύποπτα πρότυπα - να αναζητούν πράγματα.

```
TracedCallback<T1,T2,T3,T4,T5,T6,T7,T8>::TracedCallback ()
TracedCallback<T1,T2,T3,T4,T5,T6,T7,T8>::ConnectWithoutContext (c ...
TracedCallback<T1,T2,T3,T4,T5,T6,T7,T8>::Connect (const CallbackB ...
TracedCallback<T1,T2,T3,T4,T5,T6,T7,T8>::DisconnectWithoutContext ...
TracedCallback<T1,T2,T3,T4,T5,T6,T7,T8>::Disconnect (const Callba ...
TracedCallback<T1,T2,T3,T4,T5,T6,T7,T8>::operator() (void) const ...
```

```
TracedCallback<T1,T2,T3,T4,T5,T6,T7,T8>::operator() (T1 a1) const ...
TracedCallback<T1,T2,T3,T4,T5,T6,T7,T8>::operator() (T1 a1, T2 a2 ...
TracedCallback<T1,T2,T3,T4,T5,T6,T7,T8>::operator() (T1 a1, T2 a2 ...
TracedCallback<T1,T2,T3,T4,T5,T6,T7,T8>::operator() (T1 a1, T2 a2 ...
TracedCallback<T1,T2,T3,T4,T5,T6,T7,T8>::operator() (T1 a1, T2 a2 ...
TracedCallback<T1,T2,T3,T4,T5,T6,T7,T8>::operator() (T1 a1, T2 a2 ...
TracedCallback<T1,T2,T3,T4,T5,T6,T7,T8>::operator() (T1 a1, T2 a2 ...
```

Αποδεικνύεται ότι όλο αυτό προέρχεται από το αρχείο κεφαλίδας `traced-callback.h` που ακούγεται πολύ ελπιδοφόρα. Στη συνέχεια μπορείτε να ρίξετε μια ματιά στο `mobility-model.h` και να δούμε ότι υπάρχει μια γραμμή που επιβεβαιώνει αυτό το προαίσθημα

```
#include "ns3/traced-callback.h"
```

Φυσικά, θα μπορούσατε να έχετε πάει σε αυτό από την άλλη κατεύθυνση και να αρχίσετε κοιτάζοντας τα περιεχόμενα του `mobility-model.h` και παρατηρώντας το περιεχόμενο του `traced-callback.h` και να συμπεράνετε ότι αυτό πρέπει να είναι το αρχείο που θέλετε.

Σε κάθε περίπτωση, το επόμενο βήμα είναι να ρίξετε μια ματιά στο `src/core/model/traced-callback.h` στο αγαπημένο σας επεξεργαστή κειμένου για να δείτε τι συμβαίνει.

Θα δείτε ένα σχόλιο στην αρχή του αρχείου που θα πρέπει να είναι παρήγορο:

> An ns3::TracedCallback has almost exactly the same API as a normal ns3::Callback but instead of forwarding calls to a single function (as an ns3::Callback normally does), it forwards calls to a chain of ns3::Callback.

Αυτό θα πρέπει να ακούγεται πολύ οικείο και να ξέρετε ότι είστε στο σωστό δρόμο.

Λίγο μετά από αυτό το σχόλιο, θα βρείτε

```
template<typename T1 = empty, typename T2 = empty,
         typename T3 = empty, typename T4 = empty,
         typename T5 = empty, typename T6 = empty,
         typename T7 = empty, typename T8 = empty>
class TracedCallback
{
  ...
```

Αυτό σας λέει ότι TracedCallback είναι templated κλάση. Έχει οκτώ πιθανές τύπου παραμέτρους με προκαθορισμένες τιμές. Πηγαίνετε πίσω και να την συγκρίνετε με τη δήλωση που προσπαθείτε να καταλάβετε

```
TracedCallback<Ptr<const MobilityModel> > m_courseChangeTrace;
```

Το `typename T1` στην templated δήλωση της κλάσης αντιστοιχεί στο `Ptr<const MobilityModel>` στην παραπάνω δήλωση. Όλες οι άλλες παράμετροι τύπου παραμένουν ως προεπιλογές. Κοιτάζοντας τον κατασκευαστή πραγματικά δεν σας λέει πολλά. Το ένα μέρος όπου μπορείτε να έχετε δει μια σύνδεση μεταξύ της συνάρτησης επανάκλησής σας και το σύστημα εντοπισμού είναι σε `Connect` και οι συναρτήσεις `ConnectWithoutContext`. Αν μετακινηθείτε προς τα κάτω, θα δείτε μια μέθοδο `ConnectWithoutContext` εδώ

```
template<typename T1, typename T2,
         typename T3, typename T4,
         typename T5, typename T6,
         typename T7, typename T8>
void
TracedCallback<T1,T2,T3,T4,T5,T6,T7,T8>::ConnectWithoutContext ...
{
  Callback<void,T1,T2,T3,T4,T5,T6,T7,T8> cb;
  cb.Assign (callback);
```

```
  m_callbackList.push_back (cb);
}
```

Είστε τώρα στην κοιλιά του κτήνους. Όταν το πρότυπο έχει παραδείγματα για τη δήλωση παραπάνω, ο compiler θα αντικαταστήσει τον `T1` με `Ptr<const MobilityModel>`.

```
void
TracedCallback<Ptr<const MobilityModel>::ConnectWithoutContext ... cb
{
  Callback<void, Ptr<const MobilityModel> > cb;
  cb.Assign (callback);
  m_callbackList.push_back (cb);
}
```

Μπορείτε να δείτε τώρα την εφαρμογή του όλα όσα έχουμε μιλήσει. Ο κώδικας δημιουργεί μία επανάκληση του σωστού τύπου και αναθέτει τη συνάρτησή σας σε αυτό. Αυτό είναι το ισοδύναμο του `pfi = MyFunction` που συζητήσαμε στην αρχή του παρόντος τμήματος. Ο κώδικας στη συνέχεια προσθέτει την Επανάκληση στην λίστα των Επανακλήσεων για αυτήν την πηγή. Το μόνο που μένει είναι να δούμε τον ορισμό της Επανάκλησης. Χρησιμοποιώντας το ίδιο τέχνασμα `grep` όπως συνηθίζαμε να βρούμε `TracedCallback`, θα είστε σε θέση να βρείτε ότι το αρχείο `./core/callback.h` είναι αυτό που πρέπει να εξετάσουμε.

Αν κοιτάξετε κάτω από το αρχείο, θα δείτε μια πολύ πιθανώς σχεδόν ακατανόητο κώδικα προτύπου. Εσείς τελικά θα καταλήξετε σε κάποια API Τεκμηρίωση για την κλάση Επανάκλησης πρότυπο, όμως. Ευτυχώς, υπάρχουν κάποια αγγλικά:

> **Callback** template class.
>
> This class template implements the Functor Design Pattern. It is used to declare the type of a **Callback**:
>
> - the first non-optional template argument represents the return type of the callback.
> - the reminaining (optional) template arguments represent the type of the subsequent arguments to the callback.
> - up to nine arguments are supported.

Προσπαθούμε να καταλάβουμε τι

```
Callback<void, Ptr<const MobilityModel> > cb;
```

σημαίνει η δήλωση. Τώρα είμαστε σε θέση να καταλάβουμε ότι το πρώτο (μη προαιρετικό) πρότυπο όρισμα , `void`, αντιπροσωπεύει τον τύπο επιστροφής της Επανάκλησης. Το δεύτερο (προαιρετικά) πρότυπο όρισμα, `Ptr<const MobilityModel>` αντιπροσωπεύει τον τύπο του πρώτου ορίσματος επανάκλησης.

Η Επανάκληση σε ερώτηση είναι η συνάρτησή σας για να λαμβάνετε τα γεγονότα ίχνους. Από αυτό μπορείτε να συμπεράνετε ότι χρειάζεστε μια συνάρτηση που επιστρέφει `void` και παίρνει ένα `Ptr<const MobilityModel>`. Για παράδειγμα,

```
void
CourseChangeCallback (Ptr<const MobilityModel> model)
{
  ...
}
```

Αυτό είναι το μόνο που χρειάζεστε, αν θέλετε να `Config::ConnectWithoutContext`. Αν θέλετε ένα πλαίσιο, θα πρέπει να `Config::Connect` και να χρησιμοποιήσετε μια συνάρτηση Επανάκλησης που παίρνει ένα περιβάλλον συμβολοσειράς. Αυτό οφείλεται στο γεγονός ότι η συνάρτηση `Connect` θα παρέχει το περιεχόμενο για εσάς. Θα χρειαστείτε

```
void
CourseChangeCallback (std::string context, Ptr<const MobilityModel> model)
{
```

```
  ...
}
```

Αν θέλετε να διασφαλίσετε ότι το `CourseChangeCallback` είναι ορατό μόνο σε τοπικό αρχείο σας, μπορείτε να προσθέσετε τη λέξη-κλειδί `static` και να καταλήξετε σε

```
static void
CourseChangeCallback (std::string path, Ptr<const MobilityModel> model)
{
  ...
}
```

το οποίο είναι ακριβώς αυτό που χρησιμοποιήσαμε στο παράδειγμα `third.cc`. Ίσως θα πρέπει να πάτε πίσω και να διαβάσετε πάλι την προηγούμενη ενότητα.

Εάν ενδιαφέρεστε για περισσότερες λεπτομέρειες σχετικά με την εφαρμογή των Επανακλήσεων, μη διστάσετε να ρίξετε μια ματιά στο manual του *ns-3*. Αποτελούν ένα από τα πιο συχνά χρησιμοποιούμενα κατασκευάσματα στα τμήματα χαμηλού επιπέδου του *ns-3*. Είναι, κατά τη γνώμη μου, ένα πολύ κομψό πράγμα.

### 7.2.7 TracedValues

Νωρίτερα σε αυτή την ενότητα, παρουσιάσαμε ένα απλό κομμάτι του κώδικα που χρησιμοποιείται στο `TracedValue<int32_t>` να αποδείξει τα βασικά στοιχεία του κώδικα ανίχνευσης. Εμείς απλά προσπαθήσαμε να καταλάβουμε το τι είναι πραγματικά η TracedValue και πώς να βρείτε τον τύπο επιστροφής και τα επίσημα ορίσματα για την επανάκληση.

Όπως αναφέραμε, το αρχείο, `traced-value.h` φέρνει στις απαιτούμενες δηλώσεις για τον εντοπισμό των δεδομένων που υπακούει στη σημασιολογική αξία. Σε γενικές γραμμές, η σημασιολογική αξία απλά σημαίνει ότι μπορείτε να περάσετε το ίδιο το αντικείμενο γύρω του, αντί να περάσεις την διεύθυνση του αντικειμένου. Επεκτείνουμε την απαίτηση να συμπεριλάβουμε το σύνολο των φορέων ανάθεσης-στιλ που είναι προκαθορισμένα σε plain-old-data (POD) τύπους:

| | |
|---|---|
| `operator=` (assignment) | |
| `operator*=` | `operator/=` |
| `operator+=` | `operator-=` |
| `operator++` (both prefix and postfix) | |
| `operator--` (both prefix and postfix) | |
| `operator<<=` | `operator>>=` |
| `operator&=` | `operator\|=` |
| `operator%=` | `operator^=` |

Αυτό σημαίνει πραγματικά ότι θα είστε σε θέση να εντοπίζετε όλες τις αλλαγές που γίνονται με τη χρήση των εν λόγω φορέων σε αντικείμενο C++ που έχει σημασιολογική αξία.

Η δήλωση `TracedValue<>` που είδαμε παραπάνω παρέχει την υποδομή που επιβαρύνει τους φορείς που αναφέρονται ανωτέρω και κινεί τη διαδικασία επανάκλησης. Σχετικά με τη χρήση οποιουδήποτε από τα παραπάνω φορέων με ένα `TracedValue` θα παρέχει τόσο την παλιά όσο και τη νέα τιμή της μεταβλητής, σε αυτή την περίπτωση μία αξία `int32_t`. Με την επιθεώρηση της δήλωσης `TracedValue`, γνωρίζουμε ότι η συνάρτηση της καταβόθρας ίχνους θα έχει ορίσματα `(const int32_t oldValue, const int32_t newValue)`. Ο τύπος επιστροφής για μία συνάρτηση Επανάκλησης `TracedValue` είναι πάντα `void`, έτσι ώστε το αναμενόμενο στίγμα Επανάκλησης θα είναι

```
void (* TracedValueCallback)(const int32_t oldValue, const int32_t newValue);
```

Το `.AddTraceSource` στη μέθοδο `GetTypeId` παρέχει το "αγκίστρι" που χρησιμοποιείται για τη σύνδεση της πηγής ίχνους με τον έξω κόσμο μέσω του συστήματος Config. Έχουμε ήδη συζητήσει τα τρία πρώτα ορίσματα στο `AddTraceSource`: το όνομα του Χαρακτηριστικού για το σύστημα Config, μια συμβολοσειρά βοήθεια, και τη διεύθυνση του μέλους κλάσης δεδομένων TracedValue.

Το τελευταίο όρισμα συμβολοσειρών, στο παράδειγμα “ns3::Traced::Value::Int32”, είναι το όνομα ενός `typedef` για το στίγμα της συνάρτησης επανάκλησης. Χρειαζόμαστε αυτές τις υπογραφές-στίγματα που θα καθοριστούν, και να δώσουμε το πλήρως αναγνωρισμένο όνομα τύπου στο `AddTraceSource`, έτσι ώστε η τεκμηρίωση API μπορεί να συνδέσει μια πηγή ίχνους στο στίγμα της συνάρτησης. Για το στίγμα TracedValue είναι απλό, για TracedCallbacks έχουμε ήδη δει τα API Έγγραφα που πραγματικά βοηθούν.

## 7.3 Πραγματικό Παράδειγμα

Ας κάνουμε ένα παράδειγμα από ένα βιβλίο, από τα πιο γνωστά βιβλία σχετικά με το πρωτόκολλο TCP γύρω. Είναι ένα κλασικό βιβλίο “TCP/IP Illustrated, Volume 1: The Protocols,” από τον W. Richard Stevens. Απλά άφησα το βιβλίο ανοιχτό και έτρεξα ένα ωραίο σύνολο αριθμών τόσο το παράθυρο συμφόρησης όσο και τη σειρά συναρτήσει του χρόνου στη σελίδα 366. Ο Stevens το αποκαλεί αυτό “Figure 21.10. Value of cwnd and send sequence number while data is being transmitted.” Ας δημιουργήσουμε πάλι το μέρος cwnd αυτού του συνόλου σε *ns-3* με τη χρήση του συστήματος εντοπισμού και `gnuplot`.

### 7.3.1 Διαθέσιμες Πηγές

Το πρώτο πράγμα που πρέπει να σκεφτούμε είναι το πώς θέλουμε να βγάλουμε τα στοιχεία έξω. Τι είναι αυτό που χρειαζόμαστε να εντοπίσουμε; Ας συμβουλευτούμε τη λίστα “All Trace Sources” για να δούμε με τι θα εργαστούμε. Θυμηθείτε ότι αυτό βρίσκεται στο *ns-3* API Documentation. Αν μετακινηθείτε μέσα στη λίστα, θα βρείτε τελικά:

**ns3::TcpNewReno**

- **CongestionWindow**: The TCP connection’s congestion window
- **SlowStartThreshold**: TCP slow start threshold (bytes)

Αποδεικνύεται ότι η *ns-3* TCP implementation υπάρχει (κυρίως) στο αρχείο `src/internet/model/tcp-socket-base.cc` ενώ οι παραλλαγές ελέγχου συμφόρησης στα αρχεία, όπως `src/internet/model/tcp-newreno.cc`. Αν δεν γνωρίζετε αυτό το *a priori*, μπορείτε να χρησιμοποιήσετε το αναδρομικό `grep` τέχνασμα:

```
$ find . -name '*.cc' | xargs grep -i tcp
```

Θα βρείτε τη σελίδα μετά τη σελίδα των περιεχομένων του TCP που δείχνουν προς αυτό το αρχείο.

Φέρνοντας την τεκμηρίωση της κλάσης για `TcpNewReno` και παρακάμπτοντας τη λίστα των TraceSources θα βρείτε

**TraceSources**

- **CongestionWindow**: The TCP connnection’s congestion window

  Callback signature: **ns3::Traced::Value::Uint322Callback**

Κάνοντας κλικ στο σύνδεσμο επανάκλησης `typedef` βλέπουμε την υπογραφή και ξέρουμε τώρα να περιμένουμε

```
typedef void(* ns3::Traced::Value::Int32Callback)(const int32_t oldValue, const int32_t newValue)
```

Θα πρέπει να καταλάβετε τώρα αυτόν τον κώδικα εντελώς. Αν έχουμε ένα δείκτη προς το `TcpNewReno`, μπορούμε να `TraceConnect` στη πηγή ίχνους “CongestionWindow” αν παρέχουμε τον κατάλληλο στόχο επανάκλησης. Αυτό είναι το ίδιο είδος της πηγής ίχνους που είδαμε στο απλό παράδειγμα κατά την έναρξη του παρόντος τμήματος, εκτός από το ότι μιλάμε για `uint32_t` αντί `int32_t`. Και ξέρουμε ότι πρέπει να παρέχουμε μια συνάρτηση επανάκλησης με αυτή την υπογραφή.

### 7.3.2 Βρίσκοντας Παραδείγματα

Είναι πάντα καλύτερο να προσπαθήσουμε να βρούμε τον κώδικα που δουλεύει, και μπορείτε να τροποποιήσετε, αντί να αρχίσετε από το μηδέν. Έτσι, η πρώτη σειρά των εργασιών είναι τώρα να βρείτε κάποιο κώδικα που ενώνεται ήδη στη πηγή ίχνους "CongestionWindow" και να δούμε αν μπορούμε να το τροποποιήσουμε. Ως συνήθως, ο `grep` είναι ο φίλος σας:

```
$ find . -name '*.cc' | xargs grep CongestionWindow
```

Θα επισημάνω μερικούς ελπιδοφόρους υποψηφίους: `examples/tcp/tcp-large-transfer.cc` και `src/test/ns3tcp/ns3tcp-cwnd-test-suite.cc`.

Δεν έχουμε επισκεφθεί κάποιο κώδικα δοκιμής ακόμα, οπότε ας ρίξουμε μια ματιά εκεί. Θα βρείτε συνήθως ότι ο κώδικας της δοκιμής είναι αρκετά μηδαμινός, έτσι αυτό είναι ίσως ένα πολύ καλό στοίχημα. Ανοίξτε `src/test/ns3tcp/ns3tcp-cwnd-test-suite.cc` στο αγαπημένο σας επεξεργαστή και αναζητήστε για "CongestionWindow". Θα βρείτε,

```
ns3TcpSocket->TraceConnectWithoutContext ("CongestionWindow",
  MakeCallback (&Ns3TcpCwndTestCase1::CwndChange, this));
```

Αυτό θα έπρεπε να είναι γνωστό σε σας. ναφερθήκαμε παραπάνω ότι αν είχαμε ένα δείκτη στο `TcpNewReno`, θα μπορούσαμε να `TraceConnect` στη πηγή ίχνους "CongestionWindow". Αυτό είναι ακριβώς αυτό που έχουμε εδώ. Έτσι, αποδεικνύεται ότι αυτή η γραμμή του κώδικα κάνει ακριβώς αυτό που θέλουμε. Ας πάμε μπροστά και να εξάγουμε τον κώδικα που χρειαζόμαστε από τη συνάρτηση αυτή (`Ns3TcpCwndTestCase1::DoRun (void)`). Αν κοιτάξετε αυτή τη συνάρτηση, θα διαπιστώσετε ότι μοιάζει ακριβώς όπως ένα σενάριο *ns-3*. Αποδεικνύεται ότι είναι ακριβώς αυτό που είναι. Πρόκειται για ένα σενάριο που εκτελείται από το πλαίσιο της δοκιμής, ώστε να μπορούμε να το τραβήξουμε ακριβώς έξω και να το τυλίξουμε σε `main` αντί για `DoRun`. Αντί να περπατήσετε μέσα από αυτό, βήμα, βήμα, παρέχουμε το αρχείο που προκύπτει από το porting της δοκιμής πίσω σε ένα μητρικό σενάριο *ns-3* – `examples/tutorial/fifth.cc`.

### 7.3.3 Δυναμικές Πηγές Εντοπισμού

Το παράδειγμα " fifth.cc" καταδεικνύει ένα εξαιρετικά σημαντικό κανόνα που πρέπει να καταλάβετε πριν από τη χρήση κάθε είδους πηγή ίχνους: θα πρέπει να βεβαιωθείτε ότι ο στόχος της εντολής `Config::Connect` υπάρχει πριν προσπαθήσετε να το χρησιμοποιήσετε. Αυτό δεν είναι διαφορετικό από το να λέμε ένα αντικείμενο πρέπει να αρχικοποιείται πριν προσπαθήσετε να το καλέσετε. Αν και αυτό μπορεί να φαίνεται προφανές, όταν δηλώθηκε αυτός ο τρόπος, κάνοντας τρικλοποδιά σε πολλούς ανθρώπους που προσπαθούν να χρησιμοποιήσουν το σύστημα για πρώτη φορά.

Ας επιστρέψουμε στα βασικά για μια στιγμή. Υπάρχουν τρεις βασικές φάσεις εκτέλεσης που υπάρχουν σε κάθε σενάριο *ns-3*. Η πρώτη φάση μερικές φορές ονομάζεται "Configuration Time" ή "Setup Time", και υπάρχει κατά τη διάρκεια της περιόδου, όταν η συνάρτηση `main` από το σενάριό σας τρέχει, αλλά πριν ο `Simulator::Run` έχει καλεστεί. Η δεύτερη φάση μερικές φορές ονομάζεται "Simulation Time" και υπάρχει κατά τη διάρκεια της χρονικής περιόδουπου είναι ενεργά εκτελέσιμα τα γεγονότα του `Simulator::Run`. Μετά την ολοκλήρωση της εκτέλεσης της προσομοίωσης, `Simulator::Run` θα επιστρέψει στον έλεγχο πίσω στη συνάρτηση `main`. Όταν συμβαίνει αυτό, τα σενάρια εισέρχονται σε αυτό το τι μπορεί να ονομάζεται "Teardown Phase", η οποία είναι όταν οι κατασκευές και τα αντικείμενα που δημιουργήθηκαν κατά τη διάρκεια της εγκατάστασης ληφθούν χώρια και κυκλοφορήσουν.

Ίσως το πιο κοινό λάθος που γίνεται στην προσπάθειά τους να χρησιμοποιήσουν το σύστημα εντοπισμού υποθέτει ότι οι οντότητες που δημιουργούνται δυναμικά *κατά τη διάρκεια της προσομοίωσης* είναι διαθέσιμες κατά τη διάρκεια της ρύθμισης. Ειδικότερα, ένας `Socket` *ns-3* είναι ένα δυναμικό αντικείμενο που δημιουργείται συχνά από `Applications` που επικοινωνούν μεταξύ `Nodes`. Ένα `Application` *ns-3* έχει πάντα ένα "Start Time" και ένα "Stop Time" που συνδέονται με αυτό. Στη συντριπτική πλειονότητα των περιπτώσεων, ένα `Application` δεν θα επιχειρήσει να δημιουργήσει ένα δυναμικό αντικείμενο έως ότου η μέθοδός του `StartApplication`

καλείται σε κάποιο “Start Time”. Αυτό γίνεται για να εξασφαλιστεί ότι η προσομοίωση είναι πλήρως διαμορφωμένη πριν το App προσπαθήσει να κάνει κάτι (ό,τι θα συνέβαινε αν προσπαθούσε να συνδεθεί σε έναν κόμβο που δεν υπήρχε ακόμα κατά τη διάρκεια της ρύθμισης). Ως αποτέλεσμα, κατά τη διάρκεια της φάσης διαμόρφωσης δεν μπορείτε να συνδέσετε μια πηγή ίχνους σε μία καταβόθρα ίχνους αν μία από αυτές δημιουργείται δυναμικά κατά τη διάρκεια της προσομοίωσης.

Οι δύο λύσεις για αυτό το connundrum είναι

#. Δημιουργήστε ένα συμβάν προσομοιωτή που εκτελείται μετά το δυναμικό αντικείμενο που έχει δημιουργηθεί και συνδέστε το ίχνος, όταν εκτελείται αυτό το γεγονός, ή

#. Δημιουργήστε το δυναμικό αντικείμενο κατά το χρόνο διαμόρφωσης, κρατήστε’το στη συνέχεια, και δώστε το αντικείμενο στο σύστημα για να το χρησιμοποιήσει κατά τη διάρκεια της προσομοίωσης.

Πήραμε τη δεύτερη προσέγγιση στο παράδειγμα `fifth.cc`. Αυτή η απόφαση μας απαίτησε να δημιουργήσουμε το `MyApp Application`, ο ολόκληρος σκοπός της οποίας είναι να ρίξετε μια `Socket` ως παράμετρο.

### 7.3.4 Πέρασμα: `fifth.cc`

Τώρα, ας ρίξουμε μια ματιά στο πρόγραμμα παράδειγμα που κατασκευάσαμε με ανατομή το τεστ παράθυρο συμφόρησης. Ανοίξτε το `examples/tutorial/fifth.cc` στο αγαπημένο σας επεξεργαστή. Θα πρέπει να δείτε κάποιο γνωστό κώδικα

```
/* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
/*
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License version 2 as
 * published by the Free Software Foundation;
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Include., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
 */

#include <fstream>
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/internet-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/applications-module.h"

using namespace ns3;

NS_LOG_COMPONENT_DEFINE ("FifthScriptExample");
```

Όλα αυτά έχουν καλυφθεί, γι ‘αυτό δεν θα το αναμασήσουμε. Οι επόμενες γραμμές του κώδικα είναι η εικόνα του δικτύου και ένα σχόλιο αντιμετώπισης του προβλήματος που περιγράφηκε παραπάνω με το `Socket`.

```
// ============================================================================
//
//         node 0                  node 1
//   +----------------+    +----------------+
//   |    ns-3 TCP    |    |    ns-3 TCP    |
//   +----------------+    +----------------+
```

```
//   |    10.1.1.1    |    |    10.1.1.2    |
//   +----------------+    +----------------+
//   | point-to-point |    | point-to-point |
//   +----------------+    +----------------+
//           |                      |
//           +----------------------+
//                5 Mbps, 2 ms
//
//
// We want to look at changes in the ns-3 TCP congestion window.  We need
// to crank up a flow and hook the CongestionWindow attribute on the socket
// of the sender.  Normally one would use an on-off application to generate a
// flow, but this has a couple of problems.  First, the socket of the on-off
// application is not created until Application Start time, so we wouldn't be
// able to hook the socket (now) at configuration time.  Second, even if we
// could arrange a call after start time, the socket is not public so we
// couldn't get at it.
//
// So, we can cook up a simple version of the on-off application that does what
// we want.  On the plus side we don't need all of the complexity of the on-off
// application.  On the minus side, we don't have a helper, so we have to get
// a little more involved in the details, but this is trivial.
//
// So first, we create a socket and do the trace connect on it; then we pass
// this socket into the constructor of our simple application which we then
// install in the source node.
// ============================================================================
//
```

Αυτό θα πρέπει επίσης να είναι αυτονόητο.

Το επόμενο μέρος είναι η δήλωση του `MyApp` `Application` που έχουμε βάλει μαζί για να καταστεί δυνατή η `Socket` για να δημιουργηθούν κατά το χρόνο διαμόρφωσης.

```
class MyApp : public Application
{
public:

  MyApp ();
  virtual ~MyApp();

  void Setup (Ptr<Socket> socket, Address address, uint32_t packetSize,
    uint32_t nPackets, DataRate dataRate);

private:
  virtual void StartApplication (void);
  virtual void StopApplication (void);

  void ScheduleTx (void);
  void SendPacket (void);

  Ptr<Socket>     m_socket;
  Address         m_peer;
  uint32_t        m_packetSize;
  uint32_t        m_nPackets;
  DataRate        m_dataRate;
  EventId         m_sendEvent;
  bool            m_running;
  uint32_t        m_packetsSent;
```

```
};
```

### Ξεκινώντας/Σταματώντας Εφαρμογών

Αξίζει τον κόπο να περάσετε λίγο χρόνο εξηγώντας πώς τα γεγονότα πραγματικά ξεκίνησαν στο σύστημα. Αυτή είναι μία άλλη αρκετά βαθιά εξήγηση, και μπορεί να αγνοηθεί αν δεν κάνετε σχεδιασμό για τα εγχειρήματα μέσα στα σπλάχνα του συστήματος. Είναι χρήσιμο, ωστόσο, ότι η συζήτηση αγγίζει σχετικά με το πώς ορισμένα πολύ σημαντικά μέρη εργασίας στον *ns-3* και εκθέτει κάποια σημαντικά ιδιώματα. Εάν σχεδιάζετε για την εφαρμογή νέων μοντέλων ίσως πρέπει να καταλάβετε αυτή την ενότητα.

Ο πιο συνηθισμένος τρόπος για να ξεκινήσετε την άντληση των γεγονότων είναι να ξεκινήσει μια `Application`. Αυτό γίνεται ως αποτέλεσμα των ακόλουθων (ελπίζουμε) οικείων γραμμών του σεναρίου *ns-3*

```
ApplicationContainer apps = ...
apps.Start (Seconds (1.0));
apps.Stop (Seconds (10.0));
```

Ο κώδικας της εφαρμογής (βλέπε `src/network/helper/application-container.h` αν σας ενδιαφέρει) κάνει επαναλήψεις μέσω των εφαρμογών και των κλήσεών του,

```
app->SetStartTime (startTime);
```

ως αποτέλεσμα της κλήσης `apps.Start` και

```
app->SetStopTime (stopTime);
```

ως αποτέλεσμα της κλήσης `apps.Stop`.

Το τελικό αποτέλεσμα αυτών των κλήσεων είναι ότι θέλουμε να έχουμε τον προσομοιωτή να κάνει αυτόματα τις κλήσεις στις `Applications` για να τους πεί πότε θα ξεκινήσουν και να σταματήσουν. Στην περίπτωση της `MyApp`, κληρονομεί από την κλάση `Application` και παρακάμπτει `StartApplication`, και `StopApplication`. Αυτές είναι οι συναρτήσεις που θα κληθούν από τον προσομοιωτή την κατάλληλη στιγμή. Στην περίπτωση της `MyApp` θα διαπιστώσετε ότι η `MyApp::StartApplication` κάνει την αρχική `Bind`, και `Connect` στην υποδοχή και στη συνέχεια ξεκινά η ροή δεδομένων καλώντας την `MyApp::SendPacket`. Η `MyApp::StopApplication` σταματάει να παράγει πακέτα, ακυρώνοντας κάθε γεγονώς που εκκρεμεί και στη συνέχεια κλείνει την υποδοχή.

Ένα από τα ωραία πράγματα για τον *ns-3* είναι ότι μπορείτε να αγνοήσετε εντελώς τις λεπτομέρειες της εφαρμογής του πώς το `Application` σας "automagically" καλείται από τον προσομοιωτή στο σωστό χρόνο. Αλλά δεδομένου ότι έχουμε ήδη αποτολμήσει βαθιά μέσα στον *ns-3* ήδη, ας πάμε για αυτό.

Αν κοιτάξετε στο `src/network/model/application.cc` θα διαπιστώσετε ότι η μέθοδος `SetStartTime` από ένα `Application` θέτει μόνο τη μεταβλητή μέλος `m_startTime` και τη μέθοδο `SetStopTime` καθορίζει ακριβώς το `m_stopTime`. Από εκεί, χωρίς κάποιες συμβουλές, η διαδρομή τελειώνει.

Το κλειδί για να πάρει το μονοπάτι ξανά είναι να γνωρίζουμε ότι υπάρχει μια παγκόσμια λίστα με όλους τους κόμβους του συστήματος. Κάθε φορά που δημιουργείτε ένα κόμβο σε μια προσομοίωση, ένας δείκτης σε αυτόν τον κόμβο προστίθεται στο παγκόσμιο `NodeList`.

Ρίξτε μια ματιά σε αυτό `src/network/model/node-list.cc` και ψάξτε για αυτό `NodeList::Add`. Η δημόσια στατική εφαρμογή καλεί σε μια ιδιωτική εφαρμογή που ονομάζεται `NodeListPriv::Add`. Αυτό είναι ένα σχετικά κοινό idom στον *ns-3*. Έτσι, ρίξτε μια ματιά στο `NodeListPriv::Add`. Εκεί θα βρείτε,

```
Simulator::ScheduleWithContext (index, TimeStep (0), &Node::Initialize, node);
```

Αυτό σας λέει ότι κάθε φορά που ένας Κόμβος δημιουργείται σε μια προσομοίωση, ως παρενέργεια, μια κλήση στη μέθοδο του κόμβου `Initialize` έχει προγραμματιστεί για εσάς που συμβαίνει σε χρόνο μηδέν. Μην διαβάζετε πάρα πολύ σε αυτό το όνομα, ακόμα. Αυτό δεν σημαίνει ότι ο Κόμβος πρόκειται να αρχίσει να κάνει κάτι, κι αυτό

μπορεί να ερμηνευθεί ως μια ενημερωτική κλήση στον Κόμβο λέγοντάς του ότι η προσομοίωση έχει ξεκινήσει, δεν είναι μια κλήση για δράση λέγοντας ο κόμβος να αρχίσει να κάνει κάτι.

Έτσι, το `NodeList::Add` έμμεσα προγραμματίζει μια κλήση στο `Node::Initialize` σε χρόνο μηδέν για να συμβουλεύσει ένα νέο Κόμβο που η προσομοίωση έχει ξεκινήσει. Αν κοιτάξετε στο `src/network/model/node.h``δε θα βρείτε μια μέθοδο που ονομάζεται ``Node::Initialize`. Αποδεικνύεται ότι η μέθοδος `Initialize` κληρονομείται από την κλάση `Object`. Όλα τα αντικείμενα του συστήματος μπορεί να ενημερώνονται όταν αρχίζει η προσομοίωση, και τα αντικείμενα της κλάσης Node είναι μόνο ένα είδος αυτών των αντικειμένων.

Ρίξτε μια ματιά εδώ `src/core/model/object.cc` και αναζητήστε για `Object::Initialize`. Αυτός ο κώδικας δεν είναι τόσο απλός όσο αναμένεται τη στιγμή που ο *ns-3* `Objects` υποστηρίζει τη συσσωμάτωση. Ο κώδικας στη `Object::Initialize` κάνει επαναλήψεις και στη συνέχεια μέσω όλων των αντικειμένων που έχουν συγκεντρωτικά μαζί και καλεί τη μέθοδο `DoInitialize`. Αυτό είναι άλλο ένα ιδίωμα που είναι πολύ συχνό στον *ns-3*, μερικές φορές ονομάζεται "template design pattern.": μια δημόσια μέθοδο μη-εικονική API, η οποία παραμένει σταθερή σε όλες τις εφαρμογές, και καλεί μια ιδιωτική μέθοδο εικονικής εφαρμογής που κληρονόμησε και έχει εφαρμοστεί απο υποκατηγορίες. Τα ονόματα είναι συνήθως κάτι σαν `MethodName` για το κοινό API και `DoMethodName` για το ιδιωτικό API.

Αυτό μας λέει ότι πρέπει να κοιτάξουμε για μια μέθοδο `Node::DoInitialize` στο `src/network/model/node.cc` για τη μέθοδο που θα συνεχίσει το μονοπάτι μας. Εάν εντοπίσετε τον κώδικα, θα βρείτε μια μέθοδο που κάνει επαναλήψεις μέσα από όλες τις συσκευές στον κόμβο και στη συνέχεια όλες τις εφαρμογές στον κόμβο καλώντας `device->Initialize` και `application->Initialize` αντίστοιχα.

Ίσως γνωρίζετε ήδη ότι οι κλάσεις `Device` και `Application` κληρονομούν από την κλάση `Object` και έτσι το επόμενο βήμα θα είναι να εξετάσουμε τι συμβαίνει όταν καλείται το `Application::DoInitialize`. Ρίξτε μια ματιά σε `src/network/model/application.cc` και θα βρείτε

```
void
Application::DoInitialize (void)
{
  m_startEvent = Simulator::Schedule (m_startTime, &Application::StartApplication, this);
  if (m_stopTime != TimeStep (0))
    {
      m_stopEvent = Simulator::Schedule (m_stopTime, &Application::StopApplication, this);
    }
  Object::DoInitialize ();
}
```

Εδώ,ερχόμαστε τελικά στο τέλος της διαδρομής. Αν το έχετε κρατήσει όλο "ευθεία", όταν εφαρμόζετε ένα *ns-3* `Application`, η νέα εφαρμογή σας κληρονομεί από την κλάση `Application`. Μπορείτε να παρακάμψετε τις μεθόδους `StartApplication` και `StopApplication` και παρέχει μηχανισμούς για την εκκίνηση και τη διακοπή της ροής των δεδομένων από το νέο `Application` σας. Όταν ένας Κόμβος δημιουργείται στην προσομοίωση, προστίθεται σε ένα παγκόσμιο `NodeList`. Η πράξη της προσθήκης ενός Κόμβου σε αυτό το `NodeList` προκαλεί ένα γεγονώς του προσομοιωτή που έχει προγραμματιστεί για το χρόνο μηδέν το οποίο καλεί την μέθοδο `Node::Initialize` από το νέο προστιθέμενο κόμβο που θα καλείται όταν ξεκινά η προσομοίωση. Δεδομένου ότι ο Κόμβος κληρονομεί από `Object`, αυτό καλεί τη μέθοδο `Object::Initialize` στον κόμβο που, με τη σειρά του, καλεί τις μεθόδους `DoInitialize` για όλα τα `Objects` που συγκεντρώνονται σε κόμβους (σκεφτείτε μοντέλα κινητικότητας). Δεδομένου ότι ο Κόμβος `Object` έχει παρακαμφθεί το `DoInitialize`, η μέθοδος καλείται όταν ξεκινά η προσομοίωση. Η μέθοδος `Node::DoInitialize` καλεί τις μεθόδους `Initialize` όλων των `Applications` στον κόμβο. Δεδομένου ότι τα `Applications` είναι επίσης `Objects`, αυτό προκαλεί να καλείται το `Application::DoInitialize`. Όταν καλείται το `Application::DoInitialize`, προγραμματίζει τα γεγονότα για το `StartApplication` και το `StopApplication` καλεί την `Application`. Αυτές οι κλήσεις έχουν σχεδιαστεί για να ξεκινήσει και να σταματήσει τη ροή των δεδομένων από τις `Application`

Αυτό ήταν ένα άλλο αρκετά μακρύ ταξίδι, αλλά χρειάζεται να γίνει μια φορά, και καταλαβαίνετε τώρα ένα άλλο

πολύ βαθύ κομμάτι του *ns-3*.

### Η Εφαρμογή MyApp

Το `MyApp Application` χρειάζεται έναν κατασκευαστή και καταστροφέα, φυσικά

```
MyApp::MyApp ()
  : m_socket (0),
    m_peer (),
    m_packetSize (0),
    m_nPackets (0),
    m_dataRate (0),
    m_sendEvent (),
    m_running (false),
    m_packetsSent (0)
{
}

MyApp::~MyApp()
{
  m_socket = 0;
}
```

Η ύπαρξη του επόμενου κομματιού του κώδικα είναι ολόκληρος λόγος για τον οποίο γράψαμε αυτό το `Application` στην πρώτη θέση.

```
void
MyApp::Setup (Ptr<Socket> socket, Address address, uint32_t packetSize,
                    uint32_t nPackets, DataRate dataRate)
{
  m_socket = socket;
  m_peer = address;
  m_packetSize = packetSize;
  m_nPackets = nPackets;
  m_dataRate = dataRate;
}
```

Αυτός ο κώδικας θα πρέπει να είναι αρκετά αυτονόητος. Κάνουμε αρχικοποίηση μεταβλητών μέλους. Το σημαντικό από την άποψη του εντοπισμού είναι η `Ptr<Socket> socket` που χρειαζόμασταν για να παρέχει στην εφαρμογή κατά τη διάρκεια της ρύθμισης. Υπενθυμίζουμε ότι πρόκειται να δημιουργήσουμε το `Socket` ως `TcpSocket` (το οποίο υλοποιείται από το `TcpNewReno`) και πιάνεται απο την πηγή ίχνους του "CongestionWindow" πριν από τη διοχέτευση προς την μέθοδο `Setup`.

```
void
MyApp::StartApplication (void)
{
  m_running = true;
  m_packetsSent = 0;
  m_socket->Bind ();
  m_socket->Connect (m_peer);
  SendPacket ();
}
```

Ο παραπάνω κώδικας είναι ο προσπελάσιμος της εφαρμογή `Application::StartApplication` που θα κληθεί αυτόματα από τον προσομοιωτή για να ξεκινήσει η `Application` να τρέχει την κατάλληλη στιγμή. Μπορείτε να δείτε ότι κάνει μία λειτουργία `Socket Bind`. Εάν είστε εξοικειωμένοι με Berkeley Sockets αυτό δεν πρέπει να αποτελεί έκπληξη. Εκτελεί τις απαιτούμενες εργασίες για την τοπική πλευρά της σύνδεσης ακριβώς όπως μπορείτε να φανταστείτε. Το ακόλουθο `Connect` θα κάνει ό,τι χρειάζεται για να δημιουργήσει μια σύνδεση

με το TCP σε Address m_peer. Θα πρέπει τώρα να είναι σαφές για ποιό λόγο θα πρέπει να αναβάλει πολύ αυτό το χρόνο προσομοίωσης, αφού ο Connect θα χρειαστεί ένα πλήρως λειτουργικό δίκτυο για να ολοκληρωθεί. Μετά τον Connect, η Application ξεκινά στη συνέχεια τη δημιουργία γεγονότων προσομοίωσης καλώντας SendPacket.

Το επόμενο κομμάτι του κώδικα εξηγεί στο Application πώς να σταματήσουμε να δημιουργούμε γεγονότα προσομοίωσης.

```
void
MyApp::StopApplication (void)
{
  m_running = false;

  if (m_sendEvent.IsRunning ())
    {
      Simulator::Cancel (m_sendEvent);
    }

  if (m_socket)
    {
      m_socket->Close ();
    }
}
```

Κάθε φορά που μια εκδήλωση προσομοίωσης έχει προγραμματιστεί, ένα Event δημιουργείται. Αν το Event εκκρεμεί η εκτέλεσή του, η μέθοδος της IsRunning θα επιστρέψει true. Σε αυτόν τον κώδικα, αν IsRunning() επιστρέφει true, εμείς Cancel το γεγονός που αφαιρεί από την ουρά του προσομοιωτή εκδήλωσης. Με τον τρόπο αυτό, θα σπάσει την αλυσίδα των γεγονότων που η Application χρησιμοποιεί για να κρατήσει την αποστολή Packets και η Application πηγαίνει ήσυχη. Αφού ηρεμήσει η Application εμείς Close την υποδοχή που κατεδαφίζει τη σύνδεση TCP.

Η υποδοχή είναι στην πραγματικότητα διαγραμένη από τον καταστροφέα, όταν η m_socket = 0 εκτελείται. Αυτό αφαιρεί την τελευταία αναφορά στην υποκείμενη Ptr<Socket> που προκαλεί τον καταστροφέα του αντικειμένου που πρόκειται να κληθεί.

Υπενθυμίζουμε ότι StartApplication κάλεσε SendPacket για να ξεκινήσει η αλυσίδα των γεγονότων που περιγράφει τη συμπεριφορά Application.

```
void
MyApp::SendPacket (void)
{
  Ptr<Packet> packet = Create<Packet> (m_packetSize);
  m_socket->Send (packet);

  if (++m_packetsSent < m_nPackets)
    {
      ScheduleTx ();
    }
}
```

Εδώ, μπορείτε να δείτε ότι SendPacket κάνει ακριβώς αυτό. Δημιουργεί ένα Packet και στη συνέχεια κάνει ένα Send η οποία, αν ξέρετε Berkeley Sockets, είναι πιθανώς ακριβώς αυτό που περιμένατε να δείτε.

Είναι ευθύνη του Application να κρατήσει τον προγραμματισμό της αλυσίδας των γεγονότων, έτσι ώστε οι επόμενες γραμμές καλούν τη ScheduleTx να προγραμματίσει μια άλλη περίπτωση μετάδοσης (μία SendPacket) μέχρι το Application αποφασίσει ότι έχει στείλει αρκετά.

```
void
MyApp::ScheduleTx (void)
{
```

```
  if (m_running)
    {
      Time tNext (Seconds (m_packetSize * 8 / static_cast<double> (m_dataRate.GetBitRate ())));
      m_sendEvent = Simulator::Schedule (tNext, &MyApp::SendPacket, this);
    }
}
```

Εδώ, μπορείτε να δείτε ότι το `ScheduleTx` κάνει ακριβώς αυτό. Εάν η `Application` τρέχει (αν το `StopApplication` δεν έχει κληθεί) θα προγραμματίσετε μια νέα εκδήλωση, η οποία καλεί `SendPacket` ξανά. Ο αναγνώστης προειδοποίησης θα εντοπίσει κάτι που σκοντάφτουν και νέοι χρήστες. Ο ρυθμός δεδομένων ενός `Application` είναι ακριβώς αυτός. Δεν έχει τίποτε να κάνει με το ρυθμό δεδομένων ενός υποκείμενου `Channel`. Αυτός είναι ο ρυθμός με τον οποίο η `Application` παράγει κομμάτια(bits). Δεν λαμβάνει υπόψη οποιαδήποτε επιβάρυνση για τα διάφορα πρωτόκολλα ή τα κανάλια που χρησιμοποιεί για τη μεταφορά των δεδομένων. Εάν ορίσετε το ρυθμό δεδομένων ενός `Application` στον ίδιο ρυθμό μετάδοσης δεδομένων ως υποκείμενο `Channel` θα υπερχειλίσει η μνήμης σας.

### Πηγές Ίχνους

Ο σκοπός αυτής της εργασίας είναι οι επανακλήσεις από το TCP υποδεικνύοντας ότι το παράθυρο συμφόρησης έχει ενημερωθεί. Το επόμενο τμήμα κώδικα υλοποιεί την αντίστοιχη πηγή ίχνους:

```
static void
CwndChange (uint32_t oldCwnd, uint32_t newCwnd)
{
  NS_LOG_UNCOND (Simulator::Now ().GetSeconds () << "\t" << newCwnd);
}
```

Αυτό θα πρέπει να σας φαίνεται αρκετά οικείο, οπότε δεν θα εμβαθύνουμε σε λεπτομέρειες. Η συνάρτηση αυτή απλά καταγράφει την τρέχουσα ώρα προσομοίωσης και τη νέα τιμή του παραθύρου συμφόρησης κάθε φορά που αυτή αλλάζει. Μπορείτε πιθανώς να φανταστείτε ότι θα μπορούσατε να φορτώσει τα αποτελέσματα που προκύπτουν σε ένα πρόγραμμα γραφικών (gnuplot ή Excel) και να δείτε αμέσως ένα ωραίο γράφημα της συμπεριφοράς του παραθύρου συμφόρησης σε σχέση με τον χρόνο.

Προσθέσαμε ένα νέο ίχνος νεροχύτη για να δείξουμε που απορρίπτονται πακέτα. Πρόκειται να προσθέσουμε ένα πρότυπο μοντέλο σφάλματος στον εν λόγω κώδικα, γι 'αυτό θα θέλαμε να το δούμε στη πράξη.

```
static void
RxDrop (Ptr<const Packet> p)
{
  NS_LOG_UNCOND ("RxDrop at " << Simulator::Now ().GetSeconds ());
}
```

Αυτή η πηγή ίχνους θα πρέπει να συνδεθεί με τη "PhyRxDrop" πηγή ίχνους του από άκρο σε άκρο NetDevice. Αυτή η πηγή ίχνους πυροδοτείται όταν ένα πακέτο απορρίπτεραι από το φυσικό στρώμα ενός " NetDevice". Εάν μεταβείται στο (`src/point-to-point/model/point-to-point-net-device.cc`), θα δείτε ότι αυτή η πηγή ίχνος αναφέρεται στο `PointToPointNetDevice::m_phyRxDropTrace`. Αν στη συνέχεια να αναζητήσετε στο `src/point-to-point/model/point-to-point-net-device.h` για αυτή τη μεταβλητή μέλος, θα διαπιστώσετε ότι έχει δηλωθεί ως `TracedCallback<Ptr<const Packet> >`. Αυτό σας δείχνει 'ότι ο στόχος επανάκλησης θα πρέπει να είναι μια συνάρτηση που επιστρέφει κενό και παίρνει μια μοναδική παράμετρο η οποία είναι μια `Ptr<const Packet>` (υποθέτοντας ότι χρησιμοποιούμε `ConnectWithoutContext`) - ακριβώς αυτό που έχουμε και παραπάνω.

### Κυρίως Πρόγραμμα

Ο παρακάτω κώδικας θα πρέπει να σας είναι πολύ οικείος:

```
int
main (int argc, char *argv[])
{
  NodeContainer nodes;
  nodes.Create (2);

  PointToPointHelper pointToPoint;
  pointToPoint.SetDeviceAttribute ("DataRate", StringValue ("5Mbps"));
  pointToPoint.SetChannelAttribute ("Delay", StringValue ("2ms"));

  NetDeviceContainer devices;
  devices = pointToPoint.Install (nodes);
```

Αυτό δημιουργεί δύο κόμβους με ένα από άκρο σε άκρο κανάλι μεταξύ τους, όπως φαίνεται στην εικόνα στην αρχή του αρχείου.

Οι επόμενες γραμμές κώδικα δείχνουν κάτι νέο. Αν παρακολουθήσουμε μια σύνδεση που συμπεριφέρεται άρτια, θα καταλήξουμε με ένα μονοτονικά αύξων παράθυρο συμφόρησης. Για να δούμε κάποια ενδιαφέρουσα συμπεριφορά, θα πρέπει να εισάγουμε σφάλματα συνδέσμων που θα απορρίπτουν πακέτα, θα προκαλούν διπλά ACKs και θα ενεργοποιήσουν τις πιο ενδιαφέρουσες συμπεριφορές του παραθύρου συμφόρησης.

Ο *ns-3* παρέχει αντικείμενα τύπου `ErrorModel` που μπορούν να συνδεθούν σε `Channels`. Χρησιμοποιούμε το " RateErrorModel" το οποίο μας επιτρέπει να εισάγουμε σφάλματα σε ένα `Channel` για ένα δοσμένο *ρυθμό*.

```
Ptr<RateErrorModel> em = CreateObject<RateErrorModel> ();
em->SetAttribute ("ErrorRate", DoubleValue (0.00001));
devices.Get (1)->SetAttribute ("ReceiveErrorModel", PointerValue (em));
```

The above code instantiates a `RateErrorModel` Object, and we set the "ErrorRate" `Attribute` to the desired value. We then set the resulting instantiated `RateErrorModel` as the error model used by the point-to-point `NetDevice`. This will give us some retransmissions and make our plot a little more interesting.

```
InternetStackHelper stack;
stack.Install (nodes);

Ipv4AddressHelper address;
address.SetBase ("10.1.1.0", "255.255.255.252");
Ipv4InterfaceContainer interfaces = address.Assign (devices);
```

The above code should be familiar. It installs internet stacks on our two nodes and creates interfaces and assigns IP addresses for the point-to-point devices.

Since we are using TCP, we need something on the destination Node to receive TCP connections and data. The `PacketSink` `Application` is commonly used in *ns-3* for that purpose.

```
uint16_t sinkPort = 8080;
Address sinkAddress (InetSocketAddress(interfaces.GetAddress (1), sinkPort));
PacketSinkHelper packetSinkHelper ("ns3::TcpSocketFactory",
  InetSocketAddress (Ipv4Address::GetAny (), sinkPort));
ApplicationContainer sinkApps = packetSinkHelper.Install (nodes.Get (1));
sinkApps.Start (Seconds (0.));
sinkApps.Stop (Seconds (20.));
```

This should all be familiar, with the exception of,

```
PacketSinkHelper packetSinkHelper ("ns3::TcpSocketFactory",
  InetSocketAddress (Ipv4Address::GetAny (), sinkPort));
```

This code instantiates a `PacketSinkHelper` and tells it to create sockets using the class `ns3::TcpSocketFactory`. This class implements a design pattern called "object factory" which is a commonly

used mechanism for specifying a class used to create objects in an abstract way. Here, instead of having to create the objects themselves, you provide the PacketSinkHelper a string that specifies a TypeId string used to create an object which can then be used, in turn, to create instances of the Objects created by the factory.

The remaining parameter tells the Application which address and port it should Bind to.

The next two lines of code will create the socket and connect the trace source.

```
Ptr<Socket> ns3TcpSocket = Socket::CreateSocket (nodes.Get (0),
  TcpSocketFactory::GetTypeId ());
ns3TcpSocket->TraceConnectWithoutContext ("CongestionWindow",
  MakeCallback (&CwndChange));
```

The first statement calls the static member function Socket::CreateSocket and provides a Node and an explicit TypeId for the object factory used to create the socket. This is a slightly lower level call than the PacketSinkHelper call above, and uses an explicit C++ type instead of one referred to by a string. Otherwise, it is conceptually the same thing.

Once the TcpSocket is created and attached to the Node, we can use TraceConnectWithoutContext to connect the CongestionWindow trace source to our trace sink.

Recall that we coded an Application so we could take that Socket we just made (during configuration time) and use it in simulation time. We now have to instantiate that Application. We didn't go to any trouble to create a helper to manage the Application so we are going to have to create and install it "manually". This is actually quite easy:

```
Ptr<MyApp> app = CreateObject<MyApp> ();
app->Setup (ns3TcpSocket, sinkAddress, 1040, 1000, DataRate ("1Mbps"));
nodes.Get (0)->AddApplication (app);
app->Start (Seconds (1.));
app->Stop (Seconds (20.));
```

The first line creates an Object of type MyApp – our Application. The second line tells the Application what Socket to use, what address to connect to, how much data to send at each send event, how many send events to generate and the rate at which to produce data from those events.

Next, we manually add the MyApp Application to the source Node and explicitly call the Start and Stop methods on the Application to tell it when to start and stop doing its thing.

We need to actually do the connect from the receiver point-to-point NetDevice drop event to our RxDrop callback now.

```
devices.Get (1)->TraceConnectWithoutContext("PhyRxDrop", MakeCallback (&RxDrop));
```

It should now be obvious that we are getting a reference to the receiving Node NetDevice from its container and connecting the trace source defined by the attribute "PhyRxDrop" on that device to the trace sink RxDrop.

Finally, we tell the simulator to override any Applications and just stop processing events at 20 seconds into the simulation.

```
  Simulator::Stop (Seconds(20));
  Simulator::Run ();
  Simulator::Destroy ();

  return 0;
}
```

Recall that as soon as Simulator::Run is called, configuration time ends, and simulation time begins. All of the work we orchestrated by creating the Application and teaching it how to connect and send data actually happens during this function call.

As soon as Simulator::Run returns, the simulation is complete and we enter the teardown phase. In this case, Simulator::Destroy takes care of the gory details and we just return a success code after it completes.

### 7.3.5 Running `fifth.cc`

Since we have provided the file `fifth.cc` for you, if you have built your distribution (in debug mode since it uses `NS_LOG` – recall that optimized builds optimize out `NS_LOG`) it will be waiting for you to run.

```
$ ./waf --run fifth
Waf: Entering directory `/home/craigdo/repos/ns-3-allinone-dev/ns-3-dev/build'
Waf: Leaving directory `/home/craigdo/repos/ns-3-allinone-dev/ns-3-dev/build'
'build' finished successfully (0.684s)
1       536
1.0093  1072
1.01528 1608
1.02167 2144
...
1.11319 8040
1.12151 8576
1.12983 9112
RxDrop at 1.13696
...
```

You can probably see immediately a downside of using prints of any kind in your traces. We get those extraneous waf messages printed all over our interesting information along with those RxDrop messages. We will remedy that soon, but I'm sure you can't wait to see the results of all of this work. Let's redirect that output to a file called `cwnd.dat`:

```
$ ./waf --run fifth > cwnd.dat 2>&1
```

Now edit up "cwnd.dat" in your favorite editor and remove the waf build status and drop lines, leaving only the traced data (you could also comment out the `TraceConnectWithoutContext("PhyRxDrop", MakeCallback (&RxDrop));` in the script to get rid of the drop prints just as easily.

You can now run gnuplot (if you have it installed) and tell it to generate some pretty pictures:

```
$ gnuplot
gnuplot> set terminal png size 640,480
gnuplot> set output "cwnd.png"
gnuplot> plot "cwnd.dat" using 1:2 title 'Congestion Window' with linespoints
gnuplot> exit
```

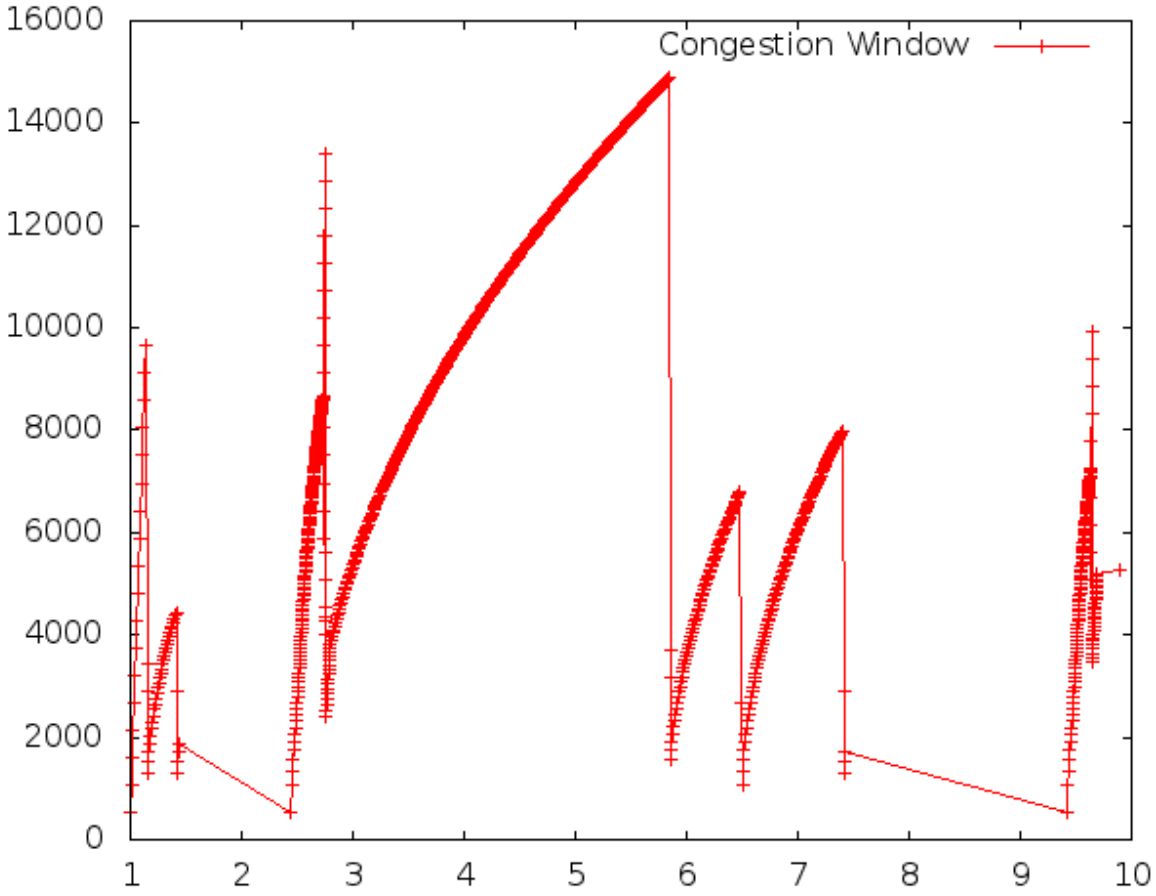
You should now have a graph of the congestion window versus time sitting in the file "cwnd.png" that looks like:

### 7.3.6 Using Mid-Level Helpers

In the previous section, we showed how to hook a trace source and get hopefully interesting information out of a simulation. Perhaps you will recall that we called logging to the standard output using `std::cout` a "blunt instrument" much earlier in this chapter. We also wrote about how it was a problem having to parse the log output in order to isolate interesting information. It may have occurred to you that we just spent a lot of time implementing an example that exhibits all of the problems we purport to fix with the *ns-3* tracing system! You would be correct. But, bear with us. We're not done yet.

One of the most important things we want to do is to is to have the ability to easily control the amount of output coming out of the simulation; and we also want to save those data to a file so we can refer back to it later. We can use the mid-level trace helpers provided in *ns-3* to do just that and complete the picture.

We provide a script that writes the cwnd change and drop events developed in the example `fifth.cc` to disk in separate files. The cwnd changes are stored as a tab-separated ASCII file and the drop events are stored in a PCAP file. The changes to make this happen are quite small.


Congestion Window
16000
14000
12000
10000
8000
6000
4000
2000
0
1
2
3
4
5
6
7
8
9
10

### Walkthrough: sixth.cc

Let's take a look at the changes required to go from fifth.cc to sixth.cc. Open examples/tutorial/sixth.cc in your favorite editor. You can see the first change by searching for CwndChange. You will find that we have changed the signatures for the trace sinks and have added a single line to each sink that writes the traced information to a stream representing a file.

```
static void
CwndChange (Ptr<OutputStreamWrapper> stream, uint32_t oldCwnd, uint32_t newCwnd)
{
  NS_LOG_UNCOND (Simulator::Now ().GetSeconds () << "\t" << newCwnd);
  *stream->GetStream () << Simulator::Now ().GetSeconds () << "\t" << oldCwnd << "\t" << newCwnd << s
}

static void
RxDrop (Ptr<PcapFileWrapper> file, Ptr<const Packet> p)
{
  NS_LOG_UNCOND ("RxDrop at " << Simulator::Now ().GetSeconds ());
  file->Write(Simulator::Now(), p);
}
```

We have added a "stream" parameter to the CwndChange trace sink. This is an object that holds (keeps safely alive) a C++ output stream. It turns out that this is a very simple object, but one that manages lifetime issues for the stream and solves a problem that even experienced C++ users run into. It turns out that the copy constructor for std::ostream is marked private. This means that std::ostreams do not obey value semantics and cannot be used in any mechanism that requires the stream to be copied. This includes the *ns-3* callback system, which as you may recall, requires objects that obey value semantics. Further notice that we have added the following line in the CwndChange trace sink implementation:

```
*stream->GetStream () << Simulator::Now ().GetSeconds () << "\t" << oldCwnd << "\t" << newCwnd << st
```

This would be very familiar code if you replaced *stream->GetStream () with std::cout, as in:

```
std::cout << Simulator::Now ().GetSeconds () << "\t" << oldCwnd << "\t" << newCwnd << std::endl;
```

This illustrates that the Ptr<OutputStreamWrapper> is really just carrying around a std::ofstream for you, and you can use it here like any other output stream.

A similar situation happens in RxDrop except that the object being passed around (a Ptr<PcapFileWrapper>) represents a PCAP file. There is a one-liner in the trace sink to write a timestamp and the contents of the packet being dropped to the PCAP file:

```
file->Write(Simulator::Now(), p);
```

Of course, if we have objects representing the two files, we need to create them somewhere and also cause them to be passed to the trace sinks. If you look in the main function, you will find new code to do just that:

```
AsciiTraceHelper asciiTraceHelper;
Ptr<OutputStreamWrapper> stream = asciiTraceHelper.CreateFileStream ("sixth.cwnd");
ns3TcpSocket->TraceConnectWithoutContext ("CongestionWindow", MakeBoundCallback (&CwndChange, stream)

...

PcapHelper pcapHelper;
Ptr<PcapFileWrapper> file = pcapHelper.CreateFile ("sixth.pcap", std::ios::out, PcapHelper::DLT_PPP);
devices.Get (1)->TraceConnectWithoutContext("PhyRxDrop", MakeBoundCallback (&RxDrop, file));
```

In the first section of the code snippet above, we are creating the ASCII trace file, creating an object responsible for managing it and using a variant of the callback creation function to arrange for the object to be passed to the sink. Our

ASCII trace helpers provide a rich set of functions to make using text (ASCII) files easy. We are just going to illustrate the use of the file stream creation function here.

The `CreateFileStream` function is basically going to instantiate a `std::ofstream` object and create a new file (or truncate an existing file). This `std::ofstream` is packaged up in an *ns-3* object for lifetime management and copy constructor issue resolution.

We then take this *ns-3* object representing the file and pass it to `MakeBoundCallback()`. This function creates a callback just like `MakeCallback()`, but it "binds" a new value to the callback. This value is added as the first argument to the callback before it is called.

Essentially, `MakeBoundCallback(&CwndChange, stream)` causes the trace source to add the additional "stream" parameter to the front of the formal parameter list before invoking the callback. This changes the required signature of the `CwndChange` sink to match the one shown above, which includes the "extra" parameter `Ptr<OutputStreamWrapper> stream`.

In the second section of code in the snippet above, we instantiate a `PcapHelper` to do the same thing for our PCAP trace file that we did with the `AsciiTraceHelper`. The line of code,

```
Ptr<PcapFileWrapper> file = pcapHelper.CreateFile ("sixth.pcap",
"w", PcapHelper::DLT_PPP);
```

creates a PCAP file named "sixth.pcap" with file mode "w". This means that the new file is truncated (contents deleted) if an existing file with that name is found. The final parameter is the "data link type" of the new PCAP file. These are the same as the PCAP library data link types defined in `bpf.h` if you are familar with PCAP. In this case, `DLT_PPP` indicates that the PCAP file is going to contain packets prefixed with point to point headers. This is true since the packets are coming from our point-to-point device driver. Other common data link types are DLT_EN10MB (10 MB Ethernet) appropriate for csma devices and DLT_IEEE802_11 (IEEE 802.11) appropriate for wifi devices. These are defined in `src/network/helper/trace-helper.h` if you are interested in seeing the list. The entries in the list match those in `bpf.h` but we duplicate them to avoid a PCAP source dependence.

A *ns-3* object representing the PCAP file is returned from `CreateFile` and used in a bound callback exactly as it was in the ASCII case.

An important detour: It is important to notice that even though both of these objects are declared in very similar ways,

```
Ptr<PcapFileWrapper> file ...
Ptr<OutputStreamWrapper> stream ...
```

The underlying objects are entirely different. For example, the `Ptr<PcapFileWrapper>` is a smart pointer to an *ns-3* Object that is a fairly heavyweight thing that supports Attributes and is integrated into the Config system. The `Ptr<OutputStreamWrapper>`, on the other hand, is a smart pointer to a reference counted object that is a very lightweight thing. Remember to look at the object you are referencing before making any assumptions about the "powers" that object may have.

For example, take a look at `src/network/utils/pcap-file-wrapper.h` in the distribution and notice,

```
class PcapFileWrapper : public Object
```

that class `PcapFileWrapper` is an *ns-3* Object by virtue of its inheritance. Then look at `src/network/model/output-stream-wrapper.h` and notice,

```
class OutputStreamWrapper : public
SimpleRefCount<OutputStreamWrapper>
```

that this object is not an *ns-3* Object at all, it is "merely" a C++ object that happens to support intrusive reference counting.

The point here is that just because you read `Ptr<something>` it does not necessarily mean that `something` is an *ns-3* Object on which you can hang *ns-3* Attributes, for example.

Now, back to the example. If you build and run this example,

```
$ ./waf --run sixth
```

you will see the same messages appear as when you ran "fifth", but two new files will appear in the top-level directory of your *ns-3* distribution.

```
sixth.cwnd  sixth.pcap
```

Since "sixth.cwnd" is an ASCII text file, you can view it with `cat` or your favorite file viewer.

```
1       0       536
1.0093  536     1072
1.01528 1072    1608
1.02167 1608    2144
...
9.69256 5149    5204
9.89311 5204    5259
```

You have a tab separated file with a timestamp, an old congestion window and a new congestion window suitable for directly importing into your plot program. There are no extraneous prints in the file, no parsing or editing is required.

Since "sixth.pcap" is a PCAP file, you can fiew it with `tcpdump`.

```
reading from file sixth.pcap, link-type PPP (PPP)
1.136956 IP 10.1.1.1.49153 > 10.1.1.2.8080: Flags [.], seq 17177:17681, ack 1, win 32768, options [TS
1.403196 IP 10.1.1.1.49153 > 10.1.1.2.8080: Flags [.], seq 33280:33784, ack 1, win 32768, options [TS
...
7.426220 IP 10.1.1.1.49153 > 10.1.1.2.8080: Flags [.], seq 785704:786240, ack 1, win 32768, options
9.630693 IP 10.1.1.1.49153 > 10.1.1.2.8080: Flags [.], seq 882688:883224, ack 1, win 32768, options
```

You have a PCAP file with the packets that were dropped in the simulation. There are no other packets present in the file and there is nothing else present to make life difficult.

It's been a long journey, but we are now at a point where we can appreciate the *ns-3* tracing system. We have pulled important events out of the middle of a TCP implementation and a device driver. We stored those events directly in files usable with commonly known tools. We did this without modifying any of the core code involved, and we did this in only 18 lines of code:

```
static void
CwndChange (Ptr<OutputStreamWrapper> stream, uint32_t oldCwnd, uint32_t newCwnd)
{
  NS_LOG_UNCOND (Simulator::Now ().GetSeconds () << "\t" << newCwnd);
  *stream->GetStream () << Simulator::Now ().GetSeconds () << "\t" << oldCwnd << "\t" << newCwnd << s
}

...

AsciiTraceHelper asciiTraceHelper;
Ptr<OutputStreamWrapper> stream = asciiTraceHelper.CreateFileStream ("sixth.cwnd");
ns3TcpSocket->TraceConnectWithoutContext ("CongestionWindow", MakeBoundCallback (&CwndChange, stream)

...

static void
RxDrop (Ptr<PcapFileWrapper> file, Ptr<const Packet> p)
{
  NS_LOG_UNCOND ("RxDrop at " << Simulator::Now ().GetSeconds ());
  file->Write(Simulator::Now(), p);
}

...
```

```
PcapHelper pcapHelper;
Ptr<PcapFileWrapper> file = pcapHelper.CreateFile ("sixth.pcap", "w", PcapHelper::DLT_PPP);
devices.Get (1)->TraceConnectWithoutContext("PhyRxDrop", MakeBoundCallback (&RxDrop, file));
```

## 7.4 Trace Helpers

The *ns-3* trace helpers provide a rich environment for configuring and selecting different trace events and writing them to files. In previous sections, primarily *Δημιουργία Τοπολογιών*, we have seen several varieties of the trace helper methods designed for use inside other (device) helpers.

Perhaps you will recall seeing some of these variations:

```
pointToPoint.EnablePcapAll ("second");
pointToPoint.EnablePcap ("second", p2pNodes.Get (0)->GetId (), 0);
csma.EnablePcap ("third", csmaDevices.Get (0), true);
pointToPoint.EnableAsciiAll (ascii.CreateFileStream ("myfirst.tr"));
```

What may not be obvious, though, is that there is a consistent model for all of the trace-related methods found in the system. We will now take a little time and take a look at the "big picture".

There are currently two primary use cases of the tracing helpers in *ns-3*: device helpers and protocol helpers. Device helpers look at the problem of specifying which traces should be enabled through a (node, device) pair. For example, you may want to specify that PCAP tracing should be enabled on a particular device on a specific node. This follows from the *ns-3* device conceptual model, and also the conceptual models of the various device helpers. Following naturally from this, the files created follow a `<prefix>-<node>-<device>` naming convention.

Protocol helpers look at the problem of specifying which traces should be enabled through a protocol and interface pair. This follows from the *ns-3* protocol stack conceptual model, and also the conceptual models of internet stack helpers. Naturally, the trace files should follow a `<prefix>-<protocol>-<interface>` naming convention.

The trace helpers therefore fall naturally into a two-dimensional taxonomy. There are subtleties that prevent all four classes from behaving identically, but we do strive to make them all work as similarly as possible; and whenever possible there are analogs for all methods in all classes.

| | PCAP | ASCII |
|---|---|---|
| Device Helper | ✓ | ✓ |
| Protocol Helper | ✓ | ✓ |

We use an approach called a `mixin` to add tracing functionality to our helper classes. A `mixin` is a class that provides functionality when it is inherited by a subclass. Inheriting from a mixin is not considered a form of specialization but is really a way to collect functionality.

Let's take a quick look at all four of these cases and their respective `mixins`.

### 7.4.1 Device Helpers

#### PCAP

The goal of these helpers is to make it easy to add a consistent PCAP trace facility to an *ns-3* device. We want all of the various flavors of PCAP tracing to work the same across all devices, so the methods of these helpers are inherited by device helpers. Take a look at `src/network/helper/trace-helper.h` if you want to follow the discussion while looking at real code.

The class `PcapHelperForDevice` is a `mixin` provides the high level functionality for using PCAP tracing in an *ns-3* device. Every device must implement a single virtual method inherited from this class.

```
virtual void EnablePcapInternal (std::string prefix, Ptr<NetDevice> nd, bool promiscuous, bool explic
```

The signature of this method reflects the device-centric view of the situation at this level. All of the public methods inherited from class `PcapUserHelperForDevice` reduce to calling this single device-dependent implementation method. For example, the lowest level PCAP method,

```
void EnablePcap (std::string prefix, Ptr<NetDevice> nd, bool promiscuous = false, bool explicitFilena
```

will call the device implementation of `EnablePcapInternal` directly. All other public PCAP tracing methods build on this implementation to provide additional user-level functionality. What this means to the user is that all device helpers in the system will have all of the PCAP trace methods available; and these methods will all work in the same way across devices if the device implements `EnablePcapInternal` correctly.

#### Methods

```
void EnablePcap (std::string prefix, Ptr<NetDevice> nd, bool promiscuous = false, bool explicitFilena
void EnablePcap (std::string prefix, std::string ndName, bool promiscuous = false, bool explicitFile
void EnablePcap (std::string prefix, NetDeviceContainer d, bool promiscuous = false);
void EnablePcap (std::string prefix, NodeContainer n, bool promiscuous = false);
void EnablePcap (std::string prefix, uint32_t nodeid, uint32_t deviceid, bool promiscuous = false);
void EnablePcapAll (std::string prefix, bool promiscuous = false);
```

In each of the methods shown above, there is a default parameter called `promiscuous` that defaults to `false`. This parameter indicates that the trace should not be gathered in promiscuous mode. If you do want your traces to include all traffic seen by the device (and if the device supports a promiscuous mode) simply add a true parameter to any of the calls above. For example,

```
Ptr<NetDevice> nd;
...
helper.EnablePcap ("prefix", nd, true);
```

will enable promiscuous mode captures on the `NetDevice` specified by `nd`.

The first two methods also include a default parameter called `explicitFilename` that will be discussed below.

You are encouraged to peruse the API Documentation for class `PcapHelperForDevice` to find the details of these methods; but to summarize ...

- You can enable PCAP tracing on a particular node/net-device pair by providing a `Ptr<NetDevice>` to an `EnablePcap` method. The `Ptr<Node>` is implicit since the net device must belong to exactly one Node. For example,

  ```
  Ptr<NetDevice> nd;
  ...
  helper.EnablePcap ("prefix", nd);
  ```

- You can enable PCAP tracing on a particular node/net-device pair by providing a `std::string` representing an object name service string to an `EnablePcap` method. The `Ptr<NetDevice>` is looked up from the name string. Again, the `<Node>` is implicit since the named net device must belong to exactly one Node. For example,

  ```
  Names::Add ("server" ...);
  Names::Add ("server/eth0" ...);
  ...
  helper.EnablePcap ("prefix", "server/ath0");
  ```

- You can enable PCAP tracing on a collection of node/net-device pairs by providing a `NetDeviceContainer`. For each `NetDevice` in the container the type is checked. For each device of the proper type (the same type as is managed by the device helper), tracing is enabled. Again, the `<Node>` is implicit since the found net device must belong to exactly one Node. For example,

```
NetDeviceContainer d = ...;
...
helper.EnablePcap ("prefix", d);
```

- You can enable PCAP tracing on a collection of node/net-device pairs by providing a `NodeContainer`. For each Node in the `NodeContainer` its attached `NetDevices` are iterated. For each `NetDevice` attached to each Node in the container, the type of that device is checked. For each device of the proper type (the same type as is managed by the device helper), tracing is enabled.

  ```
  NodeContainer n;
  ...
  helper.EnablePcap ("prefix", n);
  ```

- You can enable PCAP tracing on the basis of Node ID and device ID as well as with explicit `Ptr`. Each Node in the system has an integer Node ID and each device connected to a Node has an integer device ID.

  ```
  helper.EnablePcap ("prefix", 21, 1);
  ```

- Finally, you can enable PCAP tracing for all devices in the system, with the same type as that managed by the device helper.

  ```
  helper.EnablePcapAll ("prefix");
  ```

#### Filenames

Implicit in the method descriptions above is the construction of a complete filename by the implementation method. By convention, PCAP traces in the *ns-3* system are of the form `<prefix>-<node id>-<device id>.pcap`

As previously mentioned, every Node in the system will have a system-assigned Node id; and every device will have an interface index (also called a device id) relative to its node. By default, then, a PCAP trace file created as a result of enabling tracing on the first device of Node 21 using the prefix "prefix" would be `prefix-21-1.pcap`.

You can always use the *ns-3* object name service to make this more clear. For example, if you use the object name service to assign the name "server" to Node 21, the resulting PCAP trace file name will automatically become, `prefix-server-1.pcap` and if you also assign the name "eth0" to the device, your PCAP file name will automatically pick this up and be called `prefix-server-eth0.pcap`.

Finally, two of the methods shown above,

```
void EnablePcap (std::string prefix, Ptr<NetDevice> nd, bool promiscuous = false, bool explicitFilena
void EnablePcap (std::string prefix, std::string ndName, bool promiscuous = false, bool explicitFile
```

have a default parameter called `explicitFilename`. When set to true, this parameter disables the automatic filename completion mechanism and allows you to create an explicit filename. This option is only available in the methods which enable PCAP tracing on a single device.

For example, in order to arrange for a device helper to create a single promiscuous PCAP capture file of a specific name `my-pcap-file.pcap` on a given device, one could:

```
Ptr<NetDevice> nd;
...
helper.EnablePcap ("my-pcap-file.pcap", nd, true, true);
```

The first `true` parameter enables promiscuous mode traces and the second tells the helper to interpret the `prefix` parameter as a complete filename.

### ASCII

The behavior of the ASCII trace helper `mixin` is substantially similar to the PCAP version. Take a look at `src/network/helper/trace-helper.h` if you want to follow the discussion while looking at real code.

The class `AsciiTraceHelperForDevice` adds the high level functionality for using ASCII tracing to a device helper class. As in the PCAP case, every device must implement a single virtual method inherited from the ASCII trace `mixin`.

```
virtual void EnableAsciiInternal (Ptr<OutputStreamWrapper> stream,
                                  std::string prefix,
                                  Ptr<NetDevice> nd,
                                  bool explicitFilename) = 0;
```

The signature of this method reflects the device-centric view of the situation at this level; and also the fact that the helper may be writing to a shared output stream. All of the public ASCII-trace-related methods inherited from class `AsciiTraceHelperForDevice` reduce to calling this single device- dependent implementation method. For example, the lowest level ascii trace methods,

```
void EnableAscii (std::string prefix, Ptr<NetDevice> nd, bool explicitFilename = false);
void EnableAscii (Ptr<OutputStreamWrapper> stream, Ptr<NetDevice> nd);
```

will call the device implementation of `EnableAsciiInternal` directly, providing either a valid prefix or stream. All other public ASCII tracing methods will build on these low-level functions to provide additional user-level functionality. What this means to the user is that all device helpers in the system will have all of the ASCII trace methods available; and these methods will all work in the same way across devices if the devices implement `EnablAsciiInternal` correctly.

#### Methods

```
void EnableAscii (std::string prefix, Ptr<NetDevice> nd, bool explicitFilename = false);
void EnableAscii (Ptr<OutputStreamWrapper> stream, Ptr<NetDevice> nd);

void EnableAscii (std::string prefix, std::string ndName, bool explicitFilename = false);
void EnableAscii (Ptr<OutputStreamWrapper> stream, std::string ndName);

void EnableAscii (std::string prefix, NetDeviceContainer d);
void EnableAscii (Ptr<OutputStreamWrapper> stream, NetDeviceContainer d);

void EnableAscii (std::string prefix, NodeContainer n);
void EnableAscii (Ptr<OutputStreamWrapper> stream, NodeContainer n);

void EnableAsciiAll (std::string prefix);
void EnableAsciiAll (Ptr<OutputStreamWrapper> stream);

void EnableAscii (std::string prefix, uint32_t nodeid, uint32_t deviceid, bool explicitFilename);
void EnableAscii (Ptr<OutputStreamWrapper> stream, uint32_t nodeid, uint32_t deviceid);
```

You are encouraged to peruse the API Documentation for class `AsciiTraceHelperForDevice` to find the details of these methods; but to summarize ...

- There are twice as many methods available for ASCII tracing as there were for PCAP tracing. This is because, in addition to the PCAP-style model where traces from each unique node/device pair are written to a unique file, we support a model in which trace information for many node/device pairs is written to a common file. This means that the <prefix>-<node>-<device> file name generation mechanism is replaced by a mechanism to refer to a common file; and the number of API methods is doubled to allow all combinations.

- Just as in PCAP tracing, you can enable ASCII tracing on a particular (node, net-device) pair by providing a `Ptr<NetDevice>` to an `EnableAscii` method. The `Ptr<Node>` is implicit since the net device must belong to exactly one Node. For example,

```
Ptr<NetDevice> nd;
...
helper.EnableAscii ("prefix", nd);
```

- The first four methods also include a default parameter called `explicitFilename` that operate similar to equivalent parameters in the PCAP case.

  In this case, no trace contexts are written to the ASCII trace file since they would be redundant. The system will pick the file name to be created using the same rules as described in the PCAP section, except that the file will have the suffix `.tr` instead of `.pcap`.

- If you want to enable ASCII tracing on more than one net device and have all traces sent to a single file, you can do that as well by using an object to refer to a single file. We have already seen this in the "cwnd" example above:

  ```
  Ptr<NetDevice> nd1;
  Ptr<NetDevice> nd2;
  ...
  Ptr<OutputStreamWrapper> stream = asciiTraceHelper.CreateFileStream ("trace-file-name.tr");
  ...
  helper.EnableAscii (stream, nd1);
  helper.EnableAscii (stream, nd2);
  ```

  In this case, trace contexts *are* written to the ASCII trace file since they are required to disambiguate traces from the two devices. Note that since the user is completely specifying the file name, the string should include the `,tr` suffix for consistency.

- You can enable ASCII tracing on a particular (node, net-device) pair by providing a `std::string` representing an object name service string to an `EnablePcap` method. The `Ptr<NetDevice>` is looked up from the name string. Again, the `<Node>` is implicit since the named net device must belong to exactly one Node. For example,

  ```
  Names::Add ("client" ...);
  Names::Add ("client/eth0" ...);
  Names::Add ("server" ...);
  Names::Add ("server/eth0" ...);
  ...
  helper.EnableAscii ("prefix", "client/eth0");
  helper.EnableAscii ("prefix", "server/eth0");

  This would result in two files named ``prefix-client-eth0.tr`` and
  ``prefix-server-eth0.tr`` with traces for each device in the
  respective trace file.  Since all of the ``EnableAscii`` functions
  are overloaded to take a stream wrapper, you can use that form as
  well::

  Names::Add ("client" ...);
  Names::Add ("client/eth0" ...);
  Names::Add ("server" ...);
  Names::Add ("server/eth0" ...);
  ...
  Ptr<OutputStreamWrapper> stream = asciiTraceHelper.CreateFileStream ("trace-file-name.tr");
  ...
  helper.EnableAscii (stream, "client/eth0");
  helper.EnableAscii (stream, "server/eth0");
  ```

  This would result in a single trace file called `trace-file-name.tr` that contains all of the trace events for both devices. The events would be disambiguated by trace context strings.

- You can enable ASCII tracing on a collection of (node, net-device) pairs by providing a `NetDeviceContainer`. For each `NetDevice` in the container the type is checked. For each device of the proper type (the same type as is managed by the device helper), tracing is enabled. Again, the `<Node>` is implicit since the found net device must belong to exactly one Node. For example,

  ```
  NetDeviceContainer d = ...;
  ...
  helper.EnableAscii ("prefix", d);

  This would result in a number of ASCII trace files being created,
  each of which follows the ``<prefix>-<node id>-<device id>.tr``
  convention.
  ```

  Combining all of the traces into a single file is accomplished similarly to the examples above:

  ```
  NetDeviceContainer d = ...;
  ...
  Ptr<OutputStreamWrapper> stream = asciiTraceHelper.CreateFileStream ("trace-file-name.tr");
  ...
  helper.EnableAscii (stream, d);
  ```

- You can enable ASCII tracing on a collection of (node, net-device) pairs by providing a `NodeContainer`. For each Node in the `NodeContainer` its attached `NetDevices` are iterated. For each `NetDevice` attached to each Node in the container, the type of that device is checked. For each device of the proper type (the same type as is managed by the device helper), tracing is enabled.

  ```
  NodeContainer n;
  ...
  helper.EnableAscii ("prefix", n);
  ```

  This would result in a number of ASCII trace files being created, each of which follows the `<prefix>-<node id>-<device id>.tr` convention. Combining all of the traces into a single file is accomplished similarly to the examples above.

- You can enable PCAP tracing on the basis of Node ID and device ID as well as with explicit `Ptr`. Each Node in the system has an integer Node ID and each device connected to a Node has an integer device ID.

  ```
  helper.EnableAscii ("prefix", 21, 1);
  ```

  Of course, the traces can be combined into a single file as shown above.

- Finally, you can enable PCAP tracing for all devices in the system, with the same type as that managed by the device helper.

  ```
  helper.EnableAsciiAll ("prefix");
  ```

  This would result in a number of ASCII trace files being created, one for every device in the system of the type managed by the helper. All of these files will follow the `<prefix>-<node id>-<device id>.tr` convention. Combining all of the traces into a single file is accomplished similarly to the examples above.

#### Filenames

Implicit in the prefix-style method descriptions above is the construction of the complete filenames by the implementation method. By convention, ASCII traces in the *ns-3* system are of the form `<prefix>-<node id>-<device id>.tr`

As previously mentioned, every Node in the system will have a system-assigned Node id; and every device will have an interface index (also called a device id) relative to its node. By default, then, an ASCII trace file created as a result of enabling tracing on the first device of Node 21, using the prefix "prefix", would be `prefix-21-1.tr`.

You can always use the *ns-3* object name service to make this more clear. For example, if you use the object name service to assign the name "server" to Node 21, the resulting ASCII trace file name will automatically become, `prefix-server-1.tr` and if you also assign the name "eth0" to the device, your ASCII trace file name will automatically pick this up and be called `prefix-server-eth0.tr`.

Several of the methods have a default parameter called `explicitFilename`. When set to true, this parameter disables the automatic filename completion mechanism and allows you to create an explicit filename. This option is only available in the methods which take a prefix and enable tracing on a single device.

### 7.4.2 Protocol Helpers

#### PCAP

The goal of these `mixins` is to make it easy to add a consistent PCAP trace facility to protocols. We want all of the various flavors of PCAP tracing to work the same across all protocols, so the methods of these helpers are inherited by stack helpers. Take a look at `src/network/helper/trace-helper.h` if you want to follow the discussion while looking at real code.

In this section we will be illustrating the methods as applied to the protocol `Ipv4`. To specify traces in similar protocols, just substitute the appropriate type. For example, use a `Ptr<Ipv6>` instead of a `Ptr<Ipv4>` and call `EnablePcapIpv6` instead of `EnablePcapIpv4`.

The class `PcapHelperForIpv4` provides the high level functionality for using PCAP tracing in the `Ipv4` protocol. Each protocol helper enabling these methods must implement a single virtual method inherited from this class. There will be a separate implementation for `Ipv6`, for example, but the only difference will be in the method names and signatures. Different method names are required to disambiguate class `Ipv4` from `Ipv6` which are both derived from class `Object`, and methods that share the same signature.

```
virtual void EnablePcapIpv4Internal (std::string prefix,
                                     Ptr<Ipv4> ipv4,
                                     uint32_t interface,
                                     bool explicitFilename) = 0;
```

The signature of this method reflects the protocol and interface-centric view of the situation at this level. All of the public methods inherited from class `PcapHelperForIpv4` reduce to calling this single device-dependent implementation method. For example, the lowest level PCAP method,

```
void EnablePcapIpv4 (std::string prefix, Ptr<Ipv4> ipv4, uint32_t interface, bool explicitFilename =
```

will call the device implementation of `EnablePcapIpv4Internal` directly. All other public PCAP tracing methods build on this implementation to provide additional user-level functionality. What this means to the user is that all protocol helpers in the system will have all of the PCAP trace methods available; and these methods will all work in the same way across protocols if the helper implements `EnablePcapIpv4Internal` correctly.

##### Methods

These methods are designed to be in one-to-one correspondence with the Node- and `NetDevice`- centric versions of the device versions. Instead of Node and `NetDevice` pair constraints, we use protocol and interface constraints.

Note that just like in the device version, there are six methods:

```
void EnablePcapIpv4 (std::string prefix, Ptr<Ipv4> ipv4, uint32_t interface, bool explicitFilename =
void EnablePcapIpv4 (std::string prefix, std::string ipv4Name, uint32_t interface, bool explicitFile
void EnablePcapIpv4 (std::string prefix, Ipv4InterfaceContainer c);
void EnablePcapIpv4 (std::string prefix, NodeContainer n);
void EnablePcapIpv4 (std::string prefix, uint32_t nodeid, uint32_t interface, bool explicitFilename);
void EnablePcapIpv4All (std::string prefix);
```

You are encouraged to peruse the API Documentation for class PcapHelperForIpv4 to find the details of these methods; but to summarize ...

- You can enable PCAP tracing on a particular protocol/interface pair by providing a Ptr<Ipv4> and interface to an EnablePcap method. For example,

```
Ptr<Ipv4> ipv4 = node->GetObject<Ipv4> ();
...
helper.EnablePcapIpv4 ("prefix", ipv4, 0);
```

- You can enable PCAP tracing on a particular node/net-device pair by providing a std::string representing an object name service string to an EnablePcap method. The Ptr<Ipv4> is looked up from the name string. For example,

```
Names::Add ("serverIPv4" ...);
...
helper.EnablePcapIpv4 ("prefix", "serverIpv4", 1);
```

- You can enable PCAP tracing on a collection of protocol/interface pairs by providing an Ipv4InterfaceContainer. For each Ipv4 / interface pair in the container the protocol type is checked. For each protocol of the proper type (the same type as is managed by the device helper), tracing is enabled for the corresponding interface. For example,

```
NodeContainer nodes;
...
NetDeviceContainer devices = deviceHelper.Install (nodes);
...
Ipv4AddressHelper ipv4;
ipv4.SetBase ("10.1.1.0", "255.255.255.0");
Ipv4InterfaceContainer interfaces = ipv4.Assign (devices);
...
helper.EnablePcapIpv4 ("prefix", interfaces);
```

- You can enable PCAP tracing on a collection of protocol/interface pairs by providing a NodeContainer. For each Node in the NodeContainer the appropriate protocol is found. For each protocol, its interfaces are enumerated and tracing is enabled on the resulting pairs. For example,

```
NodeContainer n;
...
helper.EnablePcapIpv4 ("prefix", n);
```

- You can enable PCAP tracing on the basis of Node ID and interface as well. In this case, the node-id is translated to a Ptr<Node> and the appropriate protocol is looked up in the node. The resulting protocol and interface are used to specify the resulting trace source.

```
helper.EnablePcapIpv4 ("prefix", 21, 1);
```

- Finally, you can enable PCAP tracing for all interfaces in the system, with associated protocol being the same type as that managed by the device helper.

```
helper.EnablePcapIpv4All ("prefix");
```

#### Filenames

Implicit in all of the method descriptions above is the construction of the complete filenames by the implementation method. By convention, PCAP traces taken for devices in the *ns-3* system are of the form "<prefix>-<node id>-<device id>.pcap". In the case of protocol traces, there is a one-to-one correspondence between protocols and Nodes. This is because protocol Objects are aggregated to Node Objects. Since there is no global protocol id in the system, we

use the corresponding Node id in file naming. Therefore there is a possibility for file name collisions in automatically chosen trace file names. For this reason, the file name convention is changed for protocol traces.

As previously mentioned, every Node in the system will have a system-assigned Node id. Since there is a one-to-one correspondence between protocol instances and Node instances we use the Node id. Each interface has an interface id relative to its protocol. We use the convention "<prefix>-n<node id>-i<interface id>.pcap" for trace file naming in protocol helpers.

Therefore, by default, a PCAP trace file created as a result of enabling tracing on interface 1 of the Ipv4 protocol of Node 21 using the prefix "prefix" would be "prefix-n21-i1.pcap".

You can always use the *ns-3* object name service to make this more clear. For example, if you use the object name service to assign the name "serverIpv4" to the Ptr<Ipv4> on Node 21, the resulting PCAP trace file name will automatically become, "prefix-nserverIpv4-i1.pcap".

Several of the methods have a default parameter called `explicitFilename`. When set to true, this parameter disables the automatic filename completion mechanism and allows you to create an explicit filename. This option is only available in the methods which take a prefix and enable tracing on a single device.

### ASCII

The behavior of the ASCII trace helpers is substantially similar to the PCAP case. Take a look at `src/network/helper/trace-helper.h` if you want to follow the discussion while looking at real code.

In this section we will be illustrating the methods as applied to the protocol `Ipv4`. To specify traces in similar protocols, just substitute the appropriate type. For example, use a `Ptr<Ipv6>` instead of a `Ptr<Ipv4>` and call `EnableAsciiIpv6` instead of `EnableAsciiIpv4`.

The class `AsciiTraceHelperForIpv4` adds the high level functionality for using ASCII tracing to a protocol helper. Each protocol that enables these methods must implement a single virtual method inherited from this class.

```
virtual void EnableAsciiIpv4Internal (Ptr<OutputStreamWrapper> stream,
                                      std::string prefix,
                                      Ptr<Ipv4> ipv4,
                                      uint32_t interface,
                                      bool explicitFilename) = 0;
```

The signature of this method reflects the protocol- and interface-centric view of the situation at this level; and also the fact that the helper may be writing to a shared output stream. All of the public methods inherited from class `PcapAndAsciiTraceHelperForIpv4` reduce to calling this single device- dependent implementation method. For example, the lowest level ASCII trace methods,

```
void EnableAsciiIpv4 (std::string prefix, Ptr<Ipv4> ipv4, uint32_t interface, bool explicitFilename =
void EnableAsciiIpv4 (Ptr<OutputStreamWrapper> stream, Ptr<Ipv4> ipv4, uint32_t interface);
```

will call the device implementation of `EnableAsciiIpv4Internal` directly, providing either the prefix or the stream. All other public ASCII tracing methods will build on these low-level functions to provide additional user-level functionality. What this means to the user is that all device helpers in the system will have all of the ASCII trace methods available; and these methods will all work in the same way across protocols if the protocols implement `EnablAsciiIpv4Internal` correctly.

#### Methods

```
void EnableAsciiIpv4 (std::string prefix, Ptr<Ipv4> ipv4, uint32_t interface, bool explicitFilename =
void EnableAsciiIpv4 (Ptr<OutputStreamWrapper> stream, Ptr<Ipv4> ipv4, uint32_t interface);

void EnableAsciiIpv4 (std::string prefix, std::string ipv4Name, uint32_t interface, bool explicitFile
void EnableAsciiIpv4 (Ptr<OutputStreamWrapper> stream, std::string ipv4Name, uint32_t interface);
```

```
void EnableAsciiIpv4 (std::string prefix, Ipv4InterfaceContainer c);
void EnableAsciiIpv4 (Ptr<OutputStreamWrapper> stream, Ipv4InterfaceContainer c);

void EnableAsciiIpv4 (std::string prefix, NodeContainer n);
void EnableAsciiIpv4 (Ptr<OutputStreamWrapper> stream, NodeContainer n);

void EnableAsciiIpv4All (std::string prefix);
void EnableAsciiIpv4All (Ptr<OutputStreamWrapper> stream);

void EnableAsciiIpv4 (std::string prefix, uint32_t nodeid, uint32_t deviceid, bool explicitFilename);
void EnableAsciiIpv4 (Ptr<OutputStreamWrapper> stream, uint32_t nodeid, uint32_t interface);
```

You are encouraged to peruse the API Documentation for class `PcapAndAsciiHelperForIpv4` to find the details of these methods; but to summarize ...

- There are twice as many methods available for ASCII tracing as there were for PCAP tracing. This is because, in addition to the PCAP-style model where traces from each unique protocol/interface pair are written to a unique file, we support a model in which trace information for many protocol/interface pairs is written to a common file. This means that the <prefix>-n<node id>-<interface> file name generation mechanism is replaced by a mechanism to refer to a common file; and the number of API methods is doubled to allow all combinations.
- Just as in PCAP tracing, you can enable ASCII tracing on a particular protocol/interface pair by providing a `Ptr<Ipv4>` and an `interface` to an `EnableAscii` method. For example,

  ```
  Ptr<Ipv4> ipv4;
  ...
  helper.EnableAsciiIpv4 ("prefix", ipv4, 1);
  ```

  In this case, no trace contexts are written to the ASCII trace file since they would be redundant. The system will pick the file name to be created using the same rules as described in the PCAP section, except that the file will have the suffix ".tr" instead of ".pcap".
- If you want to enable ASCII tracing on more than one interface and have all traces sent to a single file, you can do that as well by using an object to refer to a single file. We have already something similar to this in the "cwnd" example above:

  ```
  Ptr<Ipv4> protocol1 = node1->GetObject<Ipv4> ();
  Ptr<Ipv4> protocol2 = node2->GetObject<Ipv4> ();
  ...
  Ptr<OutputStreamWrapper> stream = asciiTraceHelper.CreateFileStream ("trace-file-name.tr");
  ...
  helper.EnableAsciiIpv4 (stream, protocol1, 1);
  helper.EnableAsciiIpv4 (stream, protocol2, 1);
  ```

  In this case, trace contexts are written to the ASCII trace file since they are required to disambiguate traces from the two interfaces. Note that since the user is completely specifying the file name, the string should include the ",tr" for consistency.
- You can enable ASCII tracing on a particular protocol by providing a `std::string` representing an object name service string to an `EnablePcap` method. The `Ptr<Ipv4>` is looked up from the name string. The `<Node>` in the resulting filenames is implicit since there is a one-to-one correspondence between protocol instances and nodes, For example,

  ```
  Names::Add ("node1Ipv4" ...);
  Names::Add ("node2Ipv4" ...);
  ...
  helper.EnableAsciiIpv4 ("prefix", "node1Ipv4", 1);
  helper.EnableAsciiIpv4 ("prefix", "node2Ipv4", 1);
  ```

This would result in two files named "prefix-nnode1Ipv4-i1.tr" and "prefix-nnode2Ipv4-i1.tr" with traces for each interface in the respective trace file. Since all of the EnableAscii functions are overloaded to take a stream wrapper, you can use that form as well:

```
Names::Add ("node1Ipv4" ...);
Names::Add ("node2Ipv4" ...);
...
Ptr<OutputStreamWrapper> stream = asciiTraceHelper.CreateFileStream ("trace-file-name.tr");
...
helper.EnableAsciiIpv4 (stream, "node1Ipv4", 1);
helper.EnableAsciiIpv4 (stream, "node2Ipv4", 1);
```

This would result in a single trace file called "trace-file-name.tr" that contains all of the trace events for both interfaces. The events would be disambiguated by trace context strings.

- You can enable ASCII tracing on a collection of protocol/interface pairs by providing an `Ipv4InterfaceContainer`. For each protocol of the proper type (the same type as is managed by the device helper), tracing is enabled for the corresponding interface. Again, the `<Node>` is implicit since there is a one-to-one correspondence between each protocol and its node. For example,

```
NodeContainer nodes;
...
NetDeviceContainer devices = deviceHelper.Install (nodes);
...
Ipv4AddressHelper ipv4;
ipv4.SetBase ("10.1.1.0", "255.255.255.0");
Ipv4InterfaceContainer interfaces = ipv4.Assign (devices);
...
...
helper.EnableAsciiIpv4 ("prefix", interfaces);
```

This would result in a number of ASCII trace files being created, each of which follows the <prefix>-n<node id>-i<interface>.tr convention. Combining all of the traces into a single file is accomplished similarly to the examples above:

```
NodeContainer nodes;
...
NetDeviceContainer devices = deviceHelper.Install (nodes);
...
Ipv4AddressHelper ipv4;
ipv4.SetBase ("10.1.1.0", "255.255.255.0");
Ipv4InterfaceContainer interfaces = ipv4.Assign (devices);
...
Ptr<OutputStreamWrapper> stream = asciiTraceHelper.CreateFileStream ("trace-file-name.tr");
...
helper.EnableAsciiIpv4 (stream, interfaces);
```

- You can enable ASCII tracing on a collection of protocol/interface pairs by providing a `NodeContainer`. For each Node in the `NodeContainer` the appropriate protocol is found. For each protocol, its interfaces are enumerated and tracing is enabled on the resulting pairs. For example,

```
NodeContainer n;
...
helper.EnableAsciiIpv4 ("prefix", n);
```

This would result in a number of ASCII trace files being created, each of which follows the <prefix>-<node id>-<device id>.tr convention. Combining all of the traces into a single file is accomplished similarly to the examples above.

- You can enable PCAP tracing on the basis of Node ID and device ID as well. In this case, the node-id is translated

to a `Ptr<Node>` and the appropriate protocol is looked up in the node. The resulting protocol and interface are used to specify the resulting trace source.

```
helper.EnableAsciiIpv4 ("prefix", 21, 1);
```

Of course, the traces can be combined into a single file as shown above.

- Finally, you can enable ASCII tracing for all interfaces in the system, with associated protocol being the same type as that managed by the device helper.

```
helper.EnableAsciiIpv4All ("prefix");
```

This would result in a number of ASCII trace files being created, one for every interface in the system related to a protocol of the type managed by the helper. All of these files will follow the <prefix>-n<node id>-i<interface.tr convention. Combining all of the traces into a single file is accomplished similarly to the examples above.

#### Filenames

Implicit in the prefix-style method descriptions above is the construction of the complete filenames by the implementation method. By convention, ASCII traces in the *ns-3* system are of the form "<prefix>-<node id>-<device id>.tr"

As previously mentioned, every Node in the system will have a system-assigned Node id. Since there is a one-to-one correspondence between protocols and nodes we use to node-id to identify the protocol identity. Every interface on a given protocol will have an interface index (also called simply an interface) relative to its protocol. By default, then, an ASCII trace file created as a result of enabling tracing on the first device of Node 21, using the prefix "prefix", would be "prefix-n21-i1.tr". Use the prefix to disambiguate multiple protocols per node.

You can always use the *ns-3* object name service to make this more clear. For example, if you use the object name service to assign the name "serverIpv4" to the protocol on Node 21, and also specify interface one, the resulting ASCII trace file name will automatically become, "prefix-nserverIpv4-1.tr".

Several of the methods have a default parameter called `explicitFilename`. When set to true, this parameter disables the automatic filename completion mechanism and allows you to create an explicit filename. This option is only available in the methods which take a prefix and enable tracing on a single device.

## 7.5 Summary

*ns-3* includes an extremely rich environment allowing users at several levels to customize the kinds of information that can be extracted from simulations.

There are high-level helper functions that allow users to simply control the collection of pre-defined outputs to a fine granularity. There are mid-level helper functions to allow more sophisticated users to customize how information is extracted and saved; and there are low-level core functions to allow expert users to alter the system to present new and previously unexported information in a way that will be immediately accessible to users at higher levels.

This is a very comprehensive system, and we realize that it is a lot to digest, especially for new users or those not intimately familiar with C++ and its idioms. We do consider the tracing system a very important part of *ns-3* and so recommend becoming as familiar as possible with it. It is probably the case that understanding the rest of the *ns-3* system will be quite simple once you have mastered the tracing system

ΚΕΦΑΛΑΙΟ

# ΟΚΤΩ

# ΣΥΛΛΟΓΗ ΔΕΔΟΜΕΝΩΝ

Το τελευταίο κεφάλαιο του οδηγού μας παρουσιάζει κάποια συστατικά μέρη που προστέθηκαν στον *ns-3* κατά την έκδοση 3.18, και τα οποία είναι ακόμα υπό ανάπτυξη. Όπως επίσης είναι υπό ανάπτυξη και αυτό το μέρος του οδηγού.

## 8.1 Κίνηση

Ένας από τους κύριους στόχους της εκτέλεσης προσομοιώσεων είναι η δημιουργία δεδομένων εξόδου, είτε για ερευνητικούς σκοπούς είτε απλά για την εκμάθηση του συστήματος. Στο προηγούμενο κεφάλαιο, εισαγάγαμε το υποσύστημα ιχνηλασίας (tracing) και το παράδειγμα `sixth.cc` από το οποίο παράγονται PCAP ή ASCII αρχεία ιχνών. Αυτά τα ίχνη είναι πολύτιμα για την ανάλυση δεδομένων με χρήση ποικιλίας εξωτερικών εργαλείων, και για πολλούς χρήστες, καθώς τα δεδομένα εξόδου είναι ένα μέσο που προτιμάται για τη συλλογή δεδομένων (για ανάλυση από εξωτερικά εργαλεία).

Ωστόσο, υπάρχουν επίσης περιπτώσεις χρήσης που έχουν να κάνουν με περισσότερα από την απλή δημιουργία αρχείων ιχνηλασίας, συμπεριλαμβανομένων και των ακόλουθων:

- δημιουργία δεδομένων, που δεν καταγράφονται καλά σε ίχνη PCAP ή ASCII, όπως είναι τα δεδομένα εκτός των πακέτων (π.χ. μεταβάσεις καταστάσεων μηχανής σύμφωνα με πρωτόκολλα)
- μεγάλες προσομοιώσεις, για τις οποίες οι απαιτήσεις εισόδου-εξόδου σε χωρητικότητα για τη δημιουργία αρχείων ιχνηλασίας είναι απαγορευτικές ή επιβαρυντικές, και
- η ανάγκη για αναγωγή δεδομένων ή υπολογισμό *σε πραγματικό χρόνο* (online), κατά τη διάρκεια της εκτέλεσης της προσομοίωσης. Ένα καλό παράδειγμα σχετικά με αυτό είναι ο ορισμός μιας τερματικής συνθήκης για την προσομοίωση, ώστε να καθορίσετε το πότε να σταματήσει, όταν έχει λάβει αρκετά δεδομένα ώστε να σχηματίσει ένα αρκετά περιορισμένο διάστημα εμπιστοσύνης γύρω από την εκτίμηση κάποιας παραμέτρου.

Το πλαίσιο συλλογής δεδομένων του *ns-3* έχει σχεδιαστεί ώστε να παρέχει αυτές τις επιπρόσθετες δυνατότητες πέρα από αποτελέσματα που βασίζονται σε ίχνη. Συνιστούμε στους αναγνώστες που ενδιαφέρονται για αυτό το θέμα να συμβουλευτούν το εγχειρίδιο του *ns-3* για μια πιο λεπτομερή εξέταση του πλαισίου αυτού. Για τώρα, συνοψίζουμε μέσω ενός παραδείγματος κάποιες από τις δυνατότητες ανάπτυξης.

## 8.2 Παράδειγμα

Το παράδειγμα του οδηγού `examples/tutorial/seventh.cc` μοιάζει με το παράδειγμα `sixth.cc` που εξετάσαμε προηγουμένως, πέρα από κάποιες αλλαγές. Αρχικά, έχει ενεργοποιηθεί η υποστήριξη για IPv6 με μια επιλογή μέσω τερματικού:

```
CommandLine cmd;
cmd.AddValue ("useIpv6", "Use Ipv6", useV6);
cmd.Parse (argc, argv);
```

Αν ο χρήστης κάνει την επιλογή `useIpv6`, το πρόγραμμα θα εκτελεστεί χρησιμοποιώντας το IPv6 αντί του IPv4. Η επιλογή `help`, διαθέσιμη σε όλα τα προγράμματα του *ns-3* που υποστηρίζουν το αντικείμενο CommandLine όπως φαίνεται παραπάνω, μπορεί να καλεστεί ως ακολούθως (παρακαλούμε προσέξτε τη χρήση των διπλών εισαγωγικών):

```
./waf --run "seventh --help"
```

η οποία παράγει:

```
ns3-dev-seventh-debug [Program Arguments] [General Arguments]

Program Arguments:
    --useIpv6:  Use Ipv6 [false]

General Arguments:
    --PrintGlobals:              Print the list of globals.
    --PrintGroups:               Print the list of groups.
    --PrintGroup=[group]:        Print all TypeIds of group.
    --PrintTypeIds:              Print all TypeIds.
    --PrintAttributes=[typeid]:  Print all attributes of typeid.
    --PrintHelp:                 Print this help message.
```

Αυτή η προεπιλογή (η χρήση του IPv4, καθώς η useIpv6 έχει τεθεί ως false) μπορεί να αλλάξει μέσω εναλλαγής της δυαδικής τιμής της ως ακολούθως:

```
./waf --run "seventh --useIpv6=1"
```

και δείτε το PCAP αρχείο που έχει δημιουργηθεί, για παράδειγμα με την εντολή `tcpdump`:

```
tcpdump -r seventh.pcap -nn -tt
```

Αυτή ήταν μια σύντομη παρέκβαση σχετικά με την υποστήριξη του IPv6 και την γραμμή εντολών, η οποία επίσης παρουσιάστηκε νωρίτερα σε αυτόν τον οδηγό. Για ένα παράδειγμα με εξ ολοκλήρου χρήση της γραμμής εντολών, σας παρακαλούμε να δείτε το `src/core/examples/command-line-example.cc`.

Τώρα πίσω στη συλλογή δεδομένων. Στον κατάλογο `examples/tutorial/`, πληκτρολογήστε την ακόλουθη εντολή: `diff -u sixth.cc seventh.cc`, και εξετάστε κάποιες από τις νέες γραμμές αυτής της diff:

```
+  std::string probeType;
+  std::string tracePath;
+  if (useV6 == false)
+    {
   ...
+      probeType = "ns3::Ipv4PacketProbe";
+      tracePath = "/NodeList/*/$ns3::Ipv4L3Protocol/Tx";
+    }
+  else
+    {
   ...
+      probeType = "ns3::Ipv6PacketProbe";
+      tracePath = "/NodeList/*/$ns3::Ipv6L3Protocol/Tx";
+    }
 ...
+   // Use GnuplotHelper to plot the packet byte count over time
+   GnuplotHelper plotHelper;
+
```

```
+   // Configure the plot.  The first argument is the file name prefix
+   // for the output files generated.  The second, third, and fourth
+   // arguments are, respectively, the plot title, x-axis, and y-axis labels
+   plotHelper.ConfigurePlot ("seventh-packet-byte-count",
+                             "Packet Byte Count vs. Time",
+                             "Time (Seconds)",
+                             "Packet Byte Count");
+
+   // Specify the probe type, trace source path (in configuration namespace), and
+   // probe output trace source ("OutputBytes") to plot.  The fourth argument
+   // specifies the name of the data series label on the plot.  The last
+   // argument formats the plot by specifying where the key should be placed.
+   plotHelper.PlotProbe (probeType,
+                         tracePath,
+                         "OutputBytes",
+                         "Packet Byte Count",
+                         GnuplotAggregator::KEY_BELOW);
+
+   // Use FileHelper to write out the packet byte count over time
+   FileHelper fileHelper;
+
+   // Configure the file to be written, and the formatting of output data.
+   fileHelper.ConfigureFile ("seventh-packet-byte-count",
+                             FileAggregator::FORMATTED);
+
+   // Set the labels for this formatted output file.
+   fileHelper.Set2dFormat ("Time (Seconds) = %.3e\tPacket Byte Count = %.0f");
+
+   // Specify the probe type, probe path (in configuration namespace), and
+   // probe output trace source ("OutputBytes") to write.
+   fileHelper.WriteProbe (probeType,
+                          tracePath,
+                          "OutputBytes");
+
    Simulator::Stop (Seconds (20));
    Simulator::Run ();
    Simulator::Destroy ();
```

Ο προσεκτικός αναγνώστης θα ‘χει ήδη παρατηρήσει ότι, όταν δοκιμάζαμε την ιδιότητα σχετικά με το IPv6 στη γραμμή εντολών παραπάνω, εκείνο το αρχείο `seventh.cc` είχε δημιουργήσει αρκετά νέα αρχεία εξόδου:

```
seventh-packet-byte-count-0.txt
seventh-packet-byte-count-1.txt
seventh-packet-byte-count.dat
seventh-packet-byte-count.plt
seventh-packet-byte-count.png
seventh-packet-byte-count.sh
```

Αυτά δημιουργήθηκαν από τις πρόσθετες δηλώσεις που εισήχθησαν παραπάνω. Πιο συγκεκριμένα, από έναν GnuplotHelper και έναν FileHelper. Αυτά τα δεδομένα παρήχθησαν συνδέοντας τα μέρη για τη συλλογή δεδομένων σε πηγές ιχνών του *ns-3*, και μέσω εισαγωγής των δεδομένων σε ένα ήδη διαμορφωμένο `gnuplot` και σε ένα διαμορφωμένο αρχείο κειμένου. Στα επόμενα τμήματα, θα εξετάσουμε κάθε ένα από αυτά.

## 8.3 GnuplotHelper

Ο GnuplotHelper είναι ένα αντικείμενο-βοηθός του *ns-3* που στοχεύει στην παραγωγή γραφικών παραστάσεων `gnuplot` με όσο το δυνατόν λιγότερες δηλώσεις γίνεται, για συνηθισμένες περιπτώσεις. Συνδέει πηγές ιχνών του *ns-3* με τύπους δεδομένων που υποστηρίζονται από το σύστημα συλλογής δεδομένων. Δεν υποστηρίζονται όλοι οι τύποι δεδομένων που αφορούν τις πηγές ιχνών του *ns-3*, αλλά υποστηρίζονται πολλοί από τους συνηθισμένους τύπους ιχνηλασίας, συμπεριλαμβανομένων των TracedValues μαζί με τύπους απλών παλιών δεδομένων (plain old data ή POD).

Ας δούμε την έξοδο που προκύπτει από αυτόν τον βοηθό:

```
seventh-packet-byte-count.dat
seventh-packet-byte-count.plt
seventh-packet-byte-count.sh
```

Το πρώτο είναι ένα αρχείο δεδομένων gnuplot με μια σειρά από χρονοσημάνσεις διαχωρισμένες με κενά και καταμετρήσεις των byte των πακέτων. Θα καλύψουμε παρακάτω το πως καθορίστηκε η συγκεκριμένη έξοδος δεδομένων, αλλά τώρα ας συνεχίσουμε με τα αρχεία εξόδου. Το αρχείο `seventh-packet-byte-count.plt` είναι ένα αρχείο γραφικής παράστασης gnuplot, που μπορεί να ανοιχτεί μέσω του gnuplot. Οι αναγνώστες που κατανοούν τη σύνταξη του gnuplot μπορούν να δουν ότι αυτό θα παράξει ένα διαμορφωμένο PNG αρχείο ως έξοδο με το όνομα `seventh-packet-byte-count.png`. Τέλος, ένα μικρό σενάριο κελύφους `seventh-packet-byte-count.sh` εκτελεί αυτό το αρχείο γραφικής παράστασης μέσω του gnuplot για να παράξει το επιθυμητό PNG (το οποίο μπορείτε να δείτε μέσω κάποιου επεξεργαστή εικόνων). Πρόκειται για την εντολή:

```
sh seventh-packet-byte-count.sh
```

που θα παράξει το `seventh-packet-byte-count.png`. Γιατί δεν παρήχθη εξαρχής αυτό το αρχείο PNG; Η απάντηση σε αυτό είναι ότι παρέχοντας το αρχείο plot, ο χρήστης μπορεί να ρυθμίσει χειροκίνητα το αποτέλεσμα εάν το επιθυμεί, πριν να παραχθεί το PNG.

Ο τίτλος της PNG εικόνας δηλώνει ότι η γραφική παράσταση είναι μια παράσταση "Καταμέτρησης Byte Πακέτων προς Χρόνο", και ότι σχεδιάζει τα δεδομένα που έχουν ανιχνευθεί σε αντιστοιχία με το μονοπάτι της πηγής ιχνηλασίας:

```
/NodeList/*/$ns3::Ipv6L3Protocol/Tx
```

Σημειώστε τον χαρακτήρα-μπαλαντέρ στο μονοπάτι των ιχνών. Συνολικά, αυτό που αποτυπώνει αυτή η γραφική παράσταση είναι η αναπαράσταση των byte των πακέτων που παρατηρούνται στην πηγή ιχνών μετάδοσης του αντικειμένου Ipv6L3Protocol: σε μεγάλο βαθμό TCP πακέτα των 596 byte στη μια κατεύθυνση, και TCP επιβεβαιώσεις των 60 byte στην άλλη (σε αυτή την πηγή ιχνών αντιστοιχήθηκαν πηγές ιχνών από δύο κόμβους).

Πώς καθορίστηκε αυτό; Χρειάζονται μερικές δηλώσεις. Αρχικά, το αντικείμενο GnuplotHelper πρέπει να δηλωθεί και να ρυθμιστεί:

```
+  // Use GnuplotHelper to plot the packet byte count over time
+  GnuplotHelper plotHelper;
+
+  // Configure the plot.  The first argument is the file name prefix
+  // for the output files generated.  The second, third, and fourth
+  // arguments are, respectively, the plot title, x-axis, and y-axis labels
+  plotHelper.ConfigurePlot ("seventh-packet-byte-count",
+                            "Packet Byte Count vs. Time",
+                            "Time (Seconds)",
+                            "Packet Byte Count");
```

Μέχρι αυτό το σημείο, έχει ρυθμιστεί μια άδεια γραφική παράσταση. Το πρόθεμα του ονόματος του αρχείου είναι το πρώτο όρισμα, ο τίτλος της γραφικής είναι το δεύτερο, η ετικέτα του άξονα των X είναι το τρίτο, και η ετικέτα του άξονα των Υ το τέταρτο όρισμα.

Το επόμενο βήμα είναι να καθορίσουμε τα δεδομένα, και εδώ είναι που συνδέεται η πηγή ιχνών. Αρχικά, σημειώστε παραπάνω ότι στο πρόγραμμα δηλώσαμε μερικές μεταβλητές για μετέπειτα χρήση:

```
+  std::string probeType;
+  std::string tracePath;
+  probeType = "ns3::Ipv6PacketProbe";
+  tracePath = "/NodeList/*/$ns3::Ipv6L3Protocol/Tx";
```

Τις χρησιμοποιούμε εδώ:

```
+  // Specify the probe type, trace source path (in configuration namespace), and
+  // probe output trace source ("OutputBytes") to plot.  The fourth argument
+  // specifies the name of the data series label on the plot.  The last
+  // argument formats the plot by specifying where the key should be placed.
+  plotHelper.PlotProbe (probeType,
+                        tracePath,
+                        "OutputBytes",
+                        "Packet Byte Count",
+                        GnuplotAggregator::KEY_BELOW);
```

Τα πρώτα δύο ορίσματα είναι το όνομα του τύπου probe και το μονοπάτι της πηγής ιχνών. Αυτά τα δύο είναι μάλλον τα δυσκολότερα να οριστούν, όταν προσπαθείτε να χρησιμοποιήσετε αυτό το πλαίσιο εργασίας για να σχεδιάσετε άλλα ίχνη. Το probe ίχνος εδώ είναι η `Tx` πηγή ιχνών της κλάσης `Ipv6L3Protocol`. Αν εξετάσουμε την υλοποίηση αυτής της κλάσης (`src/internet/model/ipv6-l3-protocol.cc`), θα παρατηρήσουμε το:

```
.AddTraceSource ("Tx", "Send IPv6 packet to outgoing interface.",
                 MakeTraceSourceAccessor (&Ipv6L3Protocol::m_txTrace))
```

Αυτό μας λέει ότι το `Tx` είναι ένα όνομα για τη μεταβλητή `m_txTrace`, η οποία έχει μια δήλωση ως εξής:

```
/**
 * \brief Callback to trace TX (transmission) packets.
 */
TracedCallback<Ptr<const Packet>, Ptr<Ipv6>, uint32_t> m_txTrace;
```

Προκύπτει ότι αυτή η συγκεκριμένη υπογραφή πηγής ιχνών υποστηρίζεται από μια κλάση Probe (αυτό που χρειαζόμαστε εδώ) της κλάσης Ipv6PacketProbe. Δείτε τα αρχεία `src/internet/model/ipv6-packet-probe.{h,cc}`.

Έτσι, στην παραπάνω δήλωση PlotProbe, βλέπουμε ότι η δήλωση συνδέει την πηγή ιχνών (που ταυτοποιείται από την συμβολοσειρά του μονοπατιού) με έναν αντίστοιχο τύπο *ns-3* Probe του `Ipv6PacketProbe`. Εάν δεν υποστηρίζαμε αυτόν τον τύπο probe (ώστε να αντιστοιχίζεται με την υπογραφή της πηγής ιχνών), δε θα μπορούσαμε να χρησιμοποιήσουμε αυτή τη δήλωση (παρόλο που θα μπορούσαν να χρησιμοποιηθούν μερικές πιο πολύπλοκες και χαμηλότερου επιπέδου δηλώσεις, όπως περιγράφεται στο εγχειρίδιο).

Η Ipv6PacketProbe εξάγει, η ίδια, μερικές πηγές ιχνών που συλλέγουν τα δεδομένα από το ανιχνευθέν αντικείμενο Packet:

```
TypeId
Ipv6PacketProbe::GetTypeId ()
{
  static TypeId tid = TypeId ("ns3::Ipv6PacketProbe")
    .SetParent<Probe> ()
    .AddConstructor<Ipv6PacketProbe> ()
    .AddTraceSource ( "Output",
                      "The packet plus its IPv6 object and interface that serve as the output for thi
                      MakeTraceSourceAccessor (&Ipv6PacketProbe::m_output))
    .AddTraceSource ( "OutputBytes",
                      "The number of bytes in the packet",
```

```
                    MakeTraceSourceAccessor (&Ipv6PacketProbe::m_outputBytes))
  ;
  return tid;
}
```

Το τρίτο όρισμα της PlotProbe δήλωσής μας προσδιορίζει ότι ενδιαφερόμαστε για έναν αριθμό από byte σε αυτό το πακέτο: ειδικότερα, στην πηγή ιχνών "OutputBytes" της Ipv6PacketProbe. Τέλος, τα δύο τελευταία ορίσματα της δήλωσης παρέχουν το υπόμνημα της γραφικής παράστασης για αυτή τη σειρά δεδομένων ("Packet Byte Count"), και μία προαιρετική δήλωση μορφοποίησης του gnuplot (GnuplotAggregator::KEY_BELOW) ότι θέλουμε το υπόμνημα της γραφικής παράστασης να εισαχθεί κάτω από την παράσταση. Άλλες επιλογές περιλαμβάνουν τα NO_KEY, KEY_INSIDE, και KEY_ABOVE.

## 8.4 Υποστηριζόμενοι Τύποι Ιχνών

Οι ακόλουθες καταγεγραμμένες τιμές υποστηρίζονται από Probes έως και τη στιγμή που γράφεται το παρόν κείμενο:

| Τύπος TracedValue | Τύπος Probe | Αρχείο |
|---|---|---|
| double | DoubleProbe | stats/model/double-probe.h |
| uint8_t | Uinteger8Probe | stats/model/uinteger-8-probe.h |
| uint16_t | Uinteger16Probe | stats/model/uinteger-16-probe.h |
| uint32_t | Uinteger32Probe | stats/model/uinteger-32-probe.h |
| bool | BooleanProbe | stats/model/uinteger-16-probe.h |
| ns3::Time | TimeProbe | stats/model/time-probe.h |

Οι ακόλουθοι τύποι TraceSource υποστηρίζονται από τα Probes έως και τη στιγμή που γράφεται το παρόν κείμενο:

| Τύπος TracedSource | Τύπος Probe | Έξοδος Probe | Αρχείο |
|---|---|---|---|
| Ptr<const Packet> | PacketProbe | OutputBytes | network/utils/packet-probe.h |
| Ptr<const Packet>, Ptr<Ipv4>, uint32_t | Ipv4PacketProbe | OutputBytes | internet/model/ipv4-packet-probe.h |
| Ptr<const Packet>, Ptr<Ipv6>, uint32_t | Ipv6PacketProbe | OutputBytes | internet/model/ipv6-packet-probe.h |
| Ptr<const Packet>, Ptr<Ipv6>, uint32_t | Ipv6PacketProbe | OutputBytes | internet/model/ipv6-packet-probe.h |
| Ptr<const Packet>, const Address& | ApplicationPacketProbe | OutputBytes | applications/model/application-packet-probe.h |

Όπως είναι φανερό, μόνο μερικές πηγές ιχνών υποστηρίζονται, και όλες προσανατολίζονται στην εξαγωγή του μεγέθους του Packet (σε byte). Ωστόσο, οι περισσότεροι από τους βασικούς τύπους δεδομένων που είναι διαθέσιμοι ως TracedValues μπορούν να υποστηριχθούν από αυτούς τους βοηθούς.

## 8.5 FileHelper

Η κλάση FileHelper είναι απλά μια παραλλαγή του προηγούμενου παραδείγματος με τον GnuplotHelper. Το εν λόγω πρόγραμμα παρέχει διαμορφωμένη έξοδο των ίδιων χρονοσημασμένων δεδομένων, όπως η παρακάτω:

```
Time (Seconds) = 9.312e+00     Packet Byte Count = 596
Time (Seconds) = 9.312e+00     Packet Byte Count = 564
```

Δύο αρχεία παρέχονται, ένα για τον κόμβο "0" και ένα για τον κόμβο "1", όπως μπορείτε να δείτε στα ονόματα των αρχείων. Ας δούμε τον κώδικα κομμάτι-κομμάτι:

```
+   // Use FileHelper to write out the packet byte count over time
+   FileHelper fileHelper;
+
+   // Configure the file to be written, and the formatting of output data.
+   fileHelper.ConfigureFile ("seventh-packet-byte-count",
+                             FileAggregator::FORMATTED);
```

Το πρόθεμα του αρχείου για τον βοηθό αρχείων είναι το πρώτο όρισμα, και ένας προσδιοριστής της διαμόρφωσης είναι το επόμενο. Κάποιες άλλες επιλογές διαμόρφωσης περιλαμβάνουν τα PACE_SEPARATED, COMMA_SEPARATED και TAB_SEPARATED. Οι χρήστες μπορούν να αλλάξουν τη μορφοποίηση (εάν το FORMATTED καθορίζεται) με μια συμβολοσειρά διαμόρφωσης όπως η παρακάτω:

```
+
+   // Set the labels for this formatted output file.
+   fileHelper.Set2dFormat ("Time (Seconds) = %.3e\tPacket Byte Count = %.0f");
```

Εν τέλει, πρέπει να προσδεθεί η πηγή ιχνών που μας ενδιαφέρει. Πάλι, χρησιμοποιούνται οι μεταβλητές probeType και tracePath σε αυτό το παράδειγμα, και η έξοδος της πηγής ιχνών “OutputBytes” του probe συνδέεται:

```
+
+   // Specify the probe type, trace source path (in configuration namespace), and
+   // probe output trace source ("OutputBytes") to write.
+   fileHelper.WriteProbe (probeType,
+                          tracePath,
+                          "OutputBytes");
+
```

Τα πεδία χαρακτήρων-μπαλαντέρ σε αυτόν τον προσδιοριστή πηγής ιχνών αντιστοιχούν σε δύο πηγές ιχνών. Εν αντιθέσει με το παράδειγμα του GnuplotHelper, στο οποίο δύο σειρές δεδομένων παρατέθηκαν στην ίδια γραφική παράσταση, εδώ δύο ξεχωριστά αρχεία αποθηκεύονται στον δίσκο.

## 8.6 Σύνοψη

Η υποστήριξη για τη συλλογή δεδομένων είναι καινούργια από την έκδοση ns-3.18, και έχει προστεθεί η βασική υποστήριξη για την παροχή αποτελεσμάτων σε χρονικές σειρές (time series output). Το βασικό πρότυπο που περιγράφεται παραπάνω μπορεί να αναπαραχθεί και μέσα στα περιθώρια της υποστήριξης των υπάρχοντων probe και πηγών ιχνών. Περισσότερες δυνατότητες, συμπεριλαμβανομένης της στατιστικής επεξεργασίας, θα προστεθούν σε μελλοντικές εκδόσεις.

ΚΕΦΑΛΑΙΟ

# ΕΝΝΕΑ

# ΚΑΤΑΚΛΕΙΔΑ

## 9.1 Για το μέλλον

Αυτό το έγγραφο προορίζεται ώστε να είναι ένα «ζωντανό» έγγραφο. Ελπίζουμε και περιμένουμε ότι θα επεκταθεί με τον καιρό προκειμένου να καλύπτει όλο και περισσότερες πτυχές του *ns-3*.

Η συγγραφή κεφαλαίων του εγχειριδίου και του οδηγού χρήσης δεν είναι κάτι για το οποίο τρελαινόμαστε όλοι, αλλά είναι κάτι πολύ σημαντικό για το εγχείρημά μας. Αν είστε ειδικός σε κάποιον από αυτούς τους τομείς, παρακαλούμε σκεφτείτε εάν θέλετε να συνεισφέρετε στον *ns-3* παρέχοντας ένα από αυτά τα κεφάλαια, ή όποιο άλλο κεφάλαιο μπορεί να θεωρείτε ότι είναι σημαντικό.

## 9.2 Συνοψίζοντας

Ο *ns-3* είναι ένα μεγάλο και πολύπλοκο σύστημα. Είναι αδύνατο να καλυφθούν όλα τα πράγματα που θα χρειαστεί να ξέρετε σε ένα μικρό οδηγό. Προτρέπουμε τους αναγνώστες που θέλουν να μάθουν περισσότερα να διαβάσουν την ακόλουθη επιπρόσθετη τεκμηρίωση:

- Το εγχειρίδιο του *ns-3*
- Την τεκμηρίωση για τη βιβλιοθήκη μοντέλων του *ns-3*
- Το *ns-3* Doxygen (τεκμηρίωση του API)
- Το wiki του *ns-3*

– Η ομάδα ανάπτυξης του *ns-3*.