



คู่มือการใช้โปรแกรมจำลองระบบเครือข่าย

Network Simulator 3 (NS3)

เบื้องต้นเพื่อใช้ในการจำลองเครือข่ายเฉพาะกิจที่สามารถถูกควบคุมได้
ด้วยเอสดีเอ็นคอนโทรลเลอร์

ผศ.ดร. สุเมธ ประภาวัต

ได้รับทุนสนับสนุนงานวิจัยจากเงินรายได้ ประจำปีงบประมาณ 2563

คณะเทคโนโลยีสารสนเทศ

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง



รายงานวิจัยฉบับสมบูรณ์

คู่มือการใช้โปรแกรมจำลองระบบเครือข่าย

Network Simulator 3 (NS3)

เบื้องต้นเพื่อใช้ในการจำลองเครือข่ายเฉพาะกิจที่สามารถถูกควบคุมได้

ด้วยเอสดีเอ็นคอนโทรลเลอร์

ผศ.ดร. สุเมธ ประภาวัต

ได้รับทุนสนับสนุนงานวิจัยจากเงินรายได้ ประจำปีงบประมาณ 2563

คณะเทคโนโลยีสารสนเทศ

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

สารบัญ

หน้า

สารบัญ	i
บทที่ 1 บทนำ	2
บทที่ 2 การติดตั้งโปรแกรม NS3.....	4
2.1 การติดตั้งแพ็คเกจที่จำเป็นต่อการใช้งานโปรแกรม NS3.....	4
2.2 การติดตั้งโปรแกรมจำลองเครือข่าย NS3.....	5
2.3 การติดตั้งโปรแกรม Eclipse เพื่อใช้งานร่วมกับโปรแกรม NS3 และการตั้งค่าเพิ่มเติมเพื่อ การใช้งาน Debugger	10
บทที่ 3 ภาพรวมในการใช้งานโปรแกรม NS3 เบื้องต้น	18
3.1 ภาพรวมการใช้งานโปรแกรม NS3	18
3.2 การทดลองเริ่มเรียกใช้งานสคริปต์พื้นฐานเพื่อการจำลองระบบเครือข่าย	19
3.3 การสร้างโหนดในเครือข่ายเบื้องต้น และการกำหนดคุณสมบัติของอุปกรณ์ (Conceptual Overview).....	21
3.4 การเรียกใช้งาน Network Animator (NetAnim).....	24
3.5 การวัดประสิทธิภาพผ่านไฟล์ประเภท Ascii Tracing	28
3.6 คำแนะนำเบื้องต้นสำหรับการแก้ไขข้อผิดพลาดเพื่อการพัฒนาต่อยอดโปรโตคอล	31
บทที่ 4 ชุดคำสั่งเพื่อการจำลองระบบเครือข่ายเฉพาะกิจที่ถูกควบคุมได้ด้วยเอสดีเอ็น คอนโทรลเลอร์แบบอินแบนด์	35
4.1 การตั้งค่าเพื่อการใช้งานอุปกรณ์เครือข่ายไร้สายเบื้องต้น และการติดตั้งโปรโตคอลการ สื่อสารเพื่อสร้างเครือข่ายไร้สายเฉพาะกิจ (Mobile Ad Hoc Networks).....	35
4.2 การทำงานร่วมกันระหว่างเครือข่ายเฉพาะกิจเคลื่อนที่ และเอสดีเอ็นคอนโทรลเลอร์	41
บทที่ 5 แนะนำแหล่งอ้างอิงเพื่อการศึกษาเพิ่มเติม	53
5.1 Open flow V 1.3.....	53
บรรณานุกรม.....	54

บทที่ 1

บทนำ

โปรแกรมจำลองระบบเครือข่ายนั้นเป็นเครื่องมือหนึ่งที่ช่วยทำให้นักศึกษา หรือนักวิจัยสามารถศึกษาการทำงานของโพรโทคอลเครือข่ายต่าง ๆ ได้อย่างมีประสิทธิภาพผ่านการจำลองอุปกรณ์เครือข่ายขึ้นมาใช้งานได้เป็นจำนวนมาก ยกตัวอย่างเช่นโปรแกรม Network Simulator 3 (NS3) ที่อนุญาตให้ผู้ใช้งานสามารถสร้าง และกำหนดเงื่อนไขค่าพารามิเตอร์ต่าง ๆ ให้กับอุปกรณ์เครือข่ายที่ถูกสร้างขึ้น กำหนดรูปแบบการเชื่อมต่อระหว่างอุปกรณ์ รวมไปถึงการติดตั้งโพรโทคอลเพื่อการสื่อสารต่าง ๆ ที่โปรแกรมมีชุดคำสั่งรองรับ โดยตัวโปรแกรมจะแสดงผลการทำงานของจำลองออกมาได้ทั้งในรูปแบบของกราฟิกอย่างง่าย และไฟล์ข้อความที่ใช้ระบุเหตุการณ์ที่เกิดขึ้นทั้งหมดระหว่างการจำลองซึ่งถูกเรียกว่า Trace File โดยผลลัพธ์ที่ได้จากการจำลองนั้นจะสามารถถูกนำไปวิเคราะห์เพื่อใช้ในการศึกษาประสิทธิภาพของระบบเครือข่ายได้ นอกจากนี้ผู้ใช้งานยังสามารถเข้าไปแก้ไขชุดคำสั่งบนโปรแกรม NS3 เพื่อการออกแบบโพรโทคอลใหม่ ๆ ที่ผู้ใช้ได้คิดค้นขึ้นได้อีกด้วย ข้อดีอีกหนึ่งประการของโปรแกรม NS3 นั้นคือการที่ตัวโปรแกรมในปัจจุบันมีการถูกปรับปรุงอยู่ตลอดเวลา รวมไปถึงมีชุมชนสำหรับการพูดคุย และแลกเปลี่ยนความรู้ที่มีพลวัตรอย่างต่อเนื่อง จึงทำให้โปรแกรม NS3 มีความน่าสนใจที่จะถูกนำมาใช้งานมากยิ่งขึ้น

อย่างไรก็ตามโปรแกรม NS3 นั้นมีการทำงานอยู่บนระบบปฏิบัติการ Linux ซึ่งมีพื้นฐานการเขียนชุดคำสั่งด้วยภาษา C++ ซึ่งเป็นภาษาที่มีลักษณะเป็น Object Oriented นอกจากนี้ยังสามารถถูกควบคุมได้ด้วยชุดคำสั่งภาษา Python ซึ่งผู้ที่มีความสนใจจะใช้งานระบบ ควรจะมีความรู้พื้นฐานเกี่ยวกับภาษาโปรแกรมมิ่งข้างต้น ซึ่งจะช่วยให้ผู้ใช้ทำความเข้าใจชุดคำสั่งบนโปรแกรม NS3 รวมไปถึงการแก้ไขสคริปต์ต่าง ๆ เพื่อตั้งค่าการจำลองเครือข่ายได้อย่างถูกต้อง นอกจากนี้ในการจำลองระบบเครือข่ายนั้น ผู้ใช้ควรจะมีความรู้เกี่ยวกับทฤษฎีบนระบบเน็ตเวิร์กที่สนใจเพื่อที่จะทำให้กำหนดเงื่อนไขต่าง ๆ บนระบบเครือข่ายจำลองได้อย่างถูกต้องแม่นยำ

คู่มือฉบับนี้ถูกจัดทำขึ้นโดยมีวัตถุประสงค์เพื่อช่วยให้นักศึกษา หรือผู้ที่สนใจในการใช้งานโปรแกรม NS3 สามารถเริ่มต้นใช้งานโปรแกรมอย่างสะดวกมากยิ่งขึ้น โดยคู่มือฉบับนี้จะมีคำแนะนำเริ่มตั้งแต่การติดตั้งโปรแกรม การใช้งานในระดับพื้นฐาน การอธิบายชุดคำสั่งต่าง ๆ ที่สำคัญ การเรียกใช้งานสคริปต์เพื่อการจำลองเครือข่ายเบื้องต้น รวมไปถึงแนะนำแหล่งอ้างอิงที่จะช่วยให้เกิดการพัฒนาต่อยอดการใช้งานโปรแกรมในระดับที่สูงขึ้น

เนื่องจากคณะผู้วิจัยได้ทำการเขียนคู่มือฉบับนี้ระหว่างการศึกษาการทำงานร่วมกันระหว่างระบบเครือข่ายเฉพาะกิจเคลื่อนที่ และเทคโนโลยีเครือข่ายแบบเอสดีเอ็น ดังนั้นในคู่มือฉบับนี้จะมีการอธิบายชุดคำสั่งที่ใช้ในการสร้างระบบเครือข่ายเฉพาะกิจที่สามารถถูกควบคุมได้ด้วยเอสดีเอ็นคอนโทรลเลอร์ซึ่งถูกพัฒนาขึ้นด้วยโปรแกรม NS3 เพิ่มเติมเข้ามาในช่วงท้ายของคู่มือฉบับนี้

ผู้เขียนหวังเป็นอย่างยิ่งว่าคู่มือฉบับนี้จะสามารถช่วยให้ผู้ที่สนใจใช้งานโปรแกรม NS3 ได้ทำ
ความรู้จัก และเข้าใจการใช้งานตัวโปรแกรมได้อย่างรวดเร็วมากยิ่งขึ้น เพื่อช่วยทำให้ผู้ใช้งานนำ
โปรแกรม NS3 ไปต่อยอดในการจำลองระบบเครือข่ายที่ต้องการได้ โดยเฉพาะกับงานวิจัยที่เกี่ยวข้อง
กับเครือข่ายไร้สาย หรือเครือข่ายเฉพาะกิจเคลื่อนที่

บทที่ 2

การติดตั้งโปรแกรม NS3

ในการติดตั้งโปรแกรม NS3 นั้นผู้ใช้งานสามารถไปดาวน์โหลดไฟล์ที่ใช้สำหรับการติดตั้งโดยไม่เสียค่าใช้จ่ายได้ที่ <https://www.nsnam.org/releases/> ซึ่งบนลิงก์ดังกล่าวจะแสดงเวอร์ชันล่าสุดของโปรแกรม โดยในขณะที่ผู้เขียนได้จัดทำคู่มือฉบับนี้คือโปรแกรมในเวอร์ชัน NS 3.33 โดยที่โปรแกรม NS3 นั้นจะสามารถติดตั้งลงบนระบบปฏิบัติการ LINUX หรือ MACOS ได้โดยตรง โดยในทุกๆระบบปฏิบัติการนั้นจะต้องมีการติดตั้งภาษา Python เวอร์ชันตั้งแต่ 3.5 ขึ้นไป

ในคู่มือฉบับนี้จะนำเสนอการติดตั้งโปรแกรม NS3 เวอร์ชันล่าสุดลงบนระบบปฏิบัติการ Linux Mint 20 Ulyana (Cinnamon Desktop) เป็นตัวอย่าง หากผู้ใช้งานต้องการติดตั้งโปรแกรม NS3 ลงบนระบบปฏิบัติการอื่น ผู้ใช้งานสามารถศึกษารายละเอียดในการติดตั้งเพิ่มเติมได้จากลิงก์ <https://www.nsnam.org/wiki/Installation>

2.1 การติดตั้งแพ็คเกจที่จำเป็นต่อการใช้งานโปรแกรม NS3

ก่อนหน้าที่จะติดตั้งโปรแกรม NS3 นั้นผู้เขียนแนะนำให้ผู้ใช้ติดตั้งแพ็คเกจเหล่านี้เอาไว้ล่วงหน้า (ในกรณีที่มีการติดตั้งแพ็คเกจที่จำเป็นบางส่วนขาดหายไป จะมีการแสดงผล Error ออกมาในช่วงการเรียกคำสั่งเพื่อทดสอบผลการติดตั้งโปรแกรม) โดยในส่วนนี้ผู้เขียนจะแสดงรายชื่อแพ็คเกจขั้นต่ำที่จำเป็นต่อการใช้งาน NS3 ในเบื้องต้นเท่านั้น หากผู้ใช้งานต้องการติดตั้งส่วนเสริมต่าง ๆ เพิ่มเติม เพื่อการใช้งานฟังก์ชันอื่น ๆ ของโปรแกรม ผู้ใช้สามารถศึกษาข้อมูลได้จากเว็บไซต์ <https://www.nsnam.org/wiki/Installation> โดยรายละเอียดของแพ็คเกจนั้นมีดังต่อไปนี้

1. Python 3 โดยสามารถติดตั้งได้ตามคำสั่งต่อไปนี้
 - apt-get install g++ python3
 - apt-get install g++ python3 python3-dev pkg-config sqlite3
 - apt-get install python3-setuptools git
2. qt5 เพื่อการใช้งาน Network Animator ซึ่งเป็นโปรแกรมที่ทำหน้าที่นำไฟล์นามสกุล xml ที่ได้จากจำลองระบบเครือข่ายมาทำการประมวล และแสดงผลออกมาในรูปกราฟิกอย่างง่าย โดยสามารถติดตั้งได้ตามคำสั่งต่อไปนี้
 - apt-get install qt5-default mercurial
3. แพ็คเกจที่จำเป็นต่อการใช้ Bake tool ซึ่งเป็นตัวช่วยเพื่อใช้ในการติดตั้งโปรแกรม NS3 โดยสามารถติดตั้งได้ตามคำสั่งต่อไปนี้
 - apt-get install autoconf cvs bzip2 unrar

4. แพ็กเกจที่จำเป็นต่อการใช้ Debugging tool โดยสามารถติดตั้งได้ตามคำสั่งต่อไปนี้
 - apt-get install gdb valgrind
5. แพ็กเกจที่จำเป็นต่อการอ่าน Trace file ที่มีสกุล .pcap โดยสามารถติดตั้งได้ตามคำสั่งต่อไปนี้
 - apt-get install tcpdump
6. แพ็กเกจที่จำเป็นต่อการใช้งาน Python Binding โดยสามารถติดตั้งได้ตามคำสั่งต่อไปนี้
 - apt-get install cmake libc6-dev libc6-dev-i386 libclang-6.0-dev llvm-6.0-dev automake
 - apt-get install python3-pip
 - python3 -m pip install --user cxxfilt
7. แพ็กเกจที่จำเป็นต่อการสร้างไฟล์สกุล xml โดยสามารถติดตั้งได้ตามคำสั่งต่อไปนี้
 - apt-get install libxml2 libxml2-dev
 - pip3 install pygccxml
8. แพ็กเกจที่จำเป็นต่อการใช้งาน Openflow module สำหรับการใช้งานเทคโนโลยี SDN โดยสามารถติดตั้งได้ตามคำสั่งต่อไปนี้
 - apt-get install libboost-all-dev
 - apt-get install libboost-filesystem-dev
9. แพ็กเกจที่จำเป็นต่อการเพิ่มความแม่นยำในการจำลองโมดูล 802.11b WiFi error models โดยสามารถติดตั้งได้ตามคำสั่งต่อไปนี้
 - apt-get install gsl-bin libgsl-dev libgsl23 libgslcblas0
10. แพ็กเกจที่จำเป็นต่อการใช้งาน style check program ในโฟลเดอร์ utils/check-style.py โดยสามารถติดตั้งได้ตามคำสั่งต่อไปนี้
 - apt-get install uncrustify

2.2 การติดตั้งโปรแกรมจำลองเครือข่าย NS3

หลังจากการติดตั้งแพ็กเกจที่จำเป็นเรียบร้อยแล้ว ในส่วนนี้จะแสดงตัวอย่างการติดตั้งโปรแกรมจำลองเครือข่าย NS3 ซึ่งสามารถติดตั้งได้จาก 2 วิธีการ

1. การติดตั้งแบบอัตโนมัติผ่านชุดคำสั่ง Bake วิธีการนี้จะเป็นวิธีการที่ค่อนข้างสะดวกต่อการใช้งานสำหรับผู้เริ่มต้น

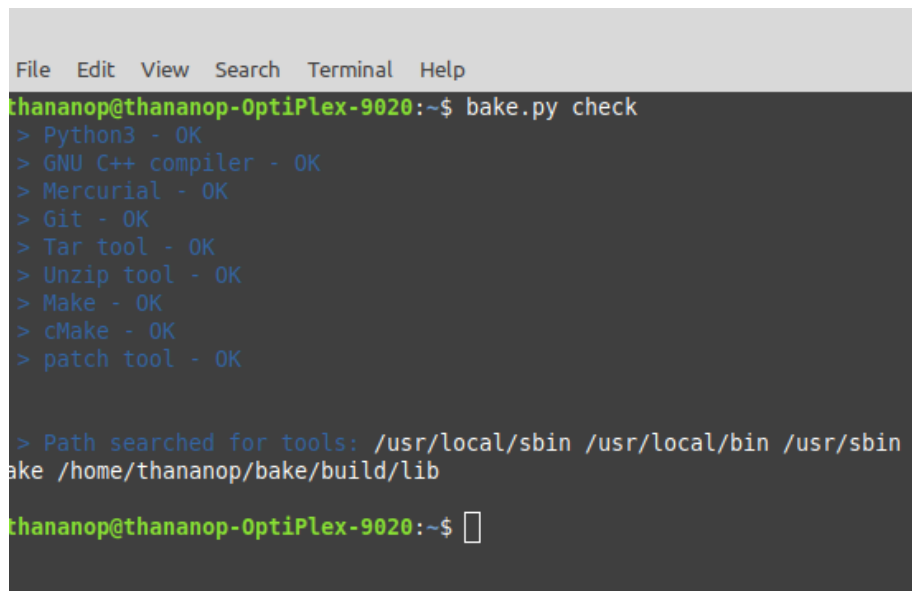
2. การติดตั้งโปรแกรมด้วยตัวเอง วิธีการนี้จะมีรายละเอียดมากกว่า โดยผู้ใช้งานจะต้องทำการดาวน์โหลดโปรแกรมจากเว็บไซต์ <http://www.nsnam.org/releases> โดยตรง และเรียกใช้คำสั่งในการติดตั้งเป็นลำดับขั้นตอน อย่างไรก็ตามการเลือกใช้การดาวน์โหลดซอร์สไฟล์โดยตรงจากเว็บไซต์นั้น จะมาพร้อมด้วย Feature เพื่อการตรวจสอบ และ Validate โปรแกรมที่มีความละเอียดมากยิ่งขึ้น

โดยในคู่มือฉบับนี้จะแสดงวิธีการติดตั้งด้วยการใช้เครื่องมือ Bake ที่จะช่วยอำนวยความสะดวกให้แก่ผู้ใช้ โดยในขั้นแรกผู้ใช้งานจะต้องทำการติดตั้งชุดโค้ด Bake ด้วยคำสั่งต่อไปนี้

```
>> git clone https://gitlab.com/nsnam/bake
```

หลังจากทำการติดตั้งชุดคำสั่ง Bake เสร็จเรียบร้อยแล้ว หากผู้ใช้งานต้องการเพิ่มความสะดวกต่อการเรียกใช้งาน ผู้ใช้งานสามารถแก้ไข Directory path เพื่อการเรียกใช้งานได้ตามคำสั่งต่อไปนี้

```
export BAKE_HOME=`pwd`/bake
export PATH=$PATH:$BAKE_HOME
export PYTHONPATH=$PYTHONPATH:$BAKE_HOME
```



```
File Edit View Search Terminal Help
thananop@thananop-OptiPlex-9020:~$ bake.py check
> Python3 - OK
> GNU C++ compiler - OK
> Mercurial - OK
> Git - OK
> Tar tool - OK
> Unzip tool - OK
> Make - OK
> cMake - OK
> patch tool - OK

> Path searched for tools: /usr/local/sbin /usr/local/bin /usr/sbin /
ake /home/thananop/bake/build/lib
thananop@thananop-OptiPlex-9020:~$
```

ภาพที่ 2.1 การตรวจสอบการติดตั้งแพ็คเกจที่จำเป็นต่อการใช้งานโปรแกรม NS3 ผ่าน bake.py

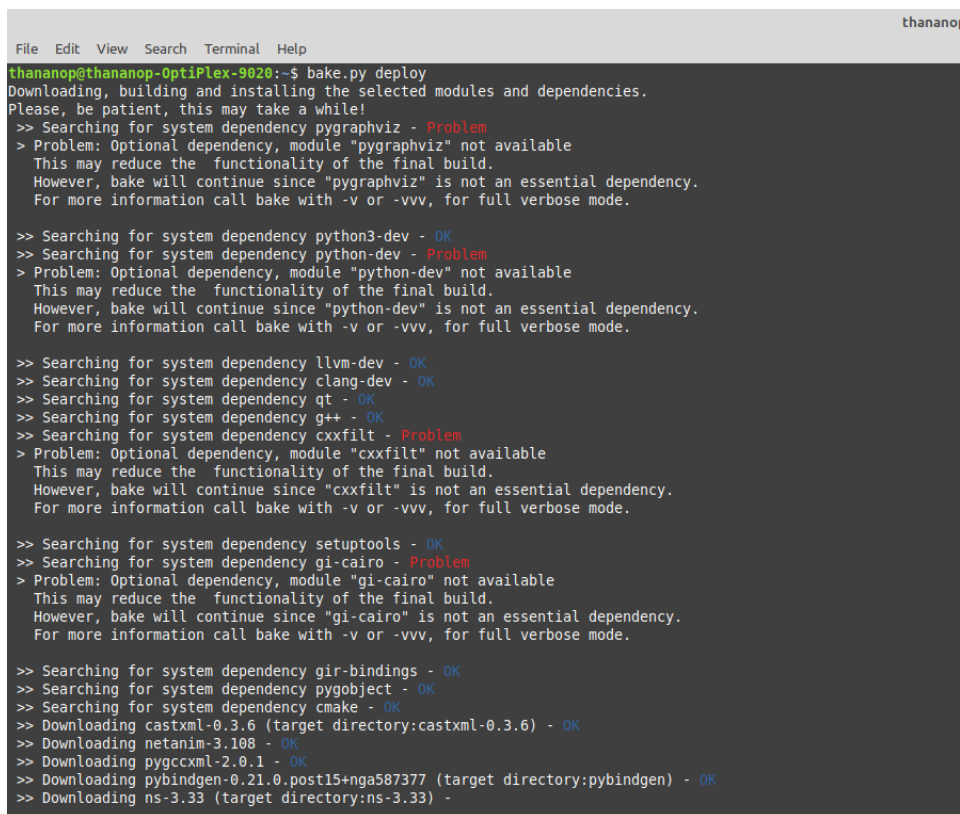
จากนั้นให้ผู้ใช้งานเรียกใช้คำสั่ง `>bake.py check` เพื่อทำการตรวจสอบว่าผู้ใช้งานได้ติดตั้งแพ็คเกจที่จำเป็นต่อการใช้งานโปรแกรม NS3 อย่างครบถ้วนหรือไม่โดยระบบจะแสดงผลลัพธ์

ตามภาพที่ 2.1 หากการติดตั้งแพ็คเกจครบถ้วน (ในกรณีที่ติดตั้งแพ็คเกจไม่ครบ ให้ผู้ใช้ติดตั้งแพ็คเกจที่ขาดไปตามที่เกิดการแจ้งเตือนจากหน้าจอ CLI) หลังจากนั้นให้ผู้ใช้ทำการเรียกคำสั่งต่อไปนี้

```
> bake.py configure -e ns-3.33
```

โดยในคำสั่งนี้จะเป็นการตั้งค่าให้ชุดโค้ดเตรียมพร้อมจะติดตั้งโปรแกรม NS3 ในเวอร์ชันที่ 3.33 ในกรณีที่เวอร์ชันของโปรแกรมได้ถูกปรับปรุงแก้ไขในอนาคต หรือผู้ใช้ต้องการติดตั้งโปรแกรมในเวอร์ชันที่ต่ำกว่า ผู้ใช้จะต้องเปลี่ยนการตั้งค่าให้สอดคล้องกับเวอร์ชันที่ผู้ใช้ต้องการ จากนั้นให้ผู้ใช้ทำการเรียกคำสั่งต่อไปนี้ เพื่อเริ่มกระบวนการติดตั้งโปรแกรม NS3 แบบอัตโนมัติ

```
> bake.py deploy
```



```
File Edit View Search Terminal Help
thananop@thananop-OptiPlex-9020:~$ bake.py deploy
Downloading, building and installing the selected modules and dependencies.
Please, be patient, this may take a while!
>> Searching for system dependency pygraphviz - Problem
> Problem: Optional dependency, module "pygraphviz" not available
This may reduce the functionality of the final build.
However, bake will continue since "pygraphviz" is not an essential dependency.
For more information call bake with -v or -vvv, for full verbose mode.

>> Searching for system dependency python3-dev - OK
>> Searching for system dependency python-dev - Problem
> Problem: Optional dependency, module "python-dev" not available
This may reduce the functionality of the final build.
However, bake will continue since "python-dev" is not an essential dependency.
For more information call bake with -v or -vvv, for full verbose mode.

>> Searching for system dependency llvm-dev - OK
>> Searching for system dependency clang-dev - OK
>> Searching for system dependency qt - OK
>> Searching for system dependency g++ - OK
>> Searching for system dependency cxxfilt - Problem
> Problem: Optional dependency, module "cxxfilt" not available
This may reduce the functionality of the final build.
However, bake will continue since "cxxfilt" is not an essential dependency.
For more information call bake with -v or -vvv, for full verbose mode.

>> Searching for system dependency setuptools - OK
>> Searching for system dependency gi-cairo - Problem
> Problem: Optional dependency, module "gi-cairo" not available
This may reduce the functionality of the final build.
However, bake will continue since "gi-cairo" is not an essential dependency.
For more information call bake with -v or -vvv, for full verbose mode.

>> Searching for system dependency gir-bindings - OK
>> Searching for system dependency pygobject - OK
>> Searching for system dependency cmake - OK
>> Downloading castxml-0.3.6 (target directory:castxml-0.3.6) - OK
>> Downloading netanim-3.108 - OK
>> Downloading pygccxml-2.0.1 - OK
>> Downloading pybindgen-0.21.0.post15+nga587377 (target directory:pybindgen) - OK
>> Downloading ns-3.33 (target directory:ns-3.33) -
```

ภาพที่ 2.2 การติดตั้งโปรแกรม NS3 ผ่าน bake.py

หลังจากที่เรียกใช้คำสั่ง deploy แล้ว หน้าจอ CLI จะแสดงผลลัพธ์ตามภาพที่ 2.2 ซึ่งแสดงให้เห็นว่าตัวชุดคำสั่ง bake จะทำการตรวจสอบแพ็คเกจที่จำเป็นอีกครั้งก่อนติดตั้งโปรแกรม NS3 โดยผู้ใช้อาจจะสามารถสังเกตได้ว่ามีข้อความ “Problem” ถูกแสดงเอาไว้ที่รายละเอียด

ของบางแพ็คเกจซึ่งเป็นการระบุว่าแพ็คเกจเหล่านี้ไม่ได้ถูกติดตั้งเอาไว้ แต่เนื่องจากแพ็คเกจเหล่านี้ไม่ใช่แพ็คเกจที่จำเป็นต่อการติดตั้งโปรแกรม NS3 ดังนั้นชุดคำสั่ง bake จึงยังคงทำงานต่อไปได้ หากผู้ใช้ต้องการติดตั้งชุดแพ็คเกจที่ถูกแนะนำทั้งหมด ผู้ใช้สามารถตรวจสอบสถานะการติดตั้งแพ็คเกจได้จากการเรียกใช้คำสั่ง

```
> bake.py show
```

โดยผู้ใช้สามารถทำการติดตั้งชุดแพ็คเกจที่ระบบแนะนำทั้งหมดเพื่อให้ทุก ๆ ส่วนขยายของโปรแกรม NS3 สามารถใช้งานได้ก่อนจะทำการติดตั้งโปรแกรมได้เช่นกัน หลังจากการติดตั้งเสร็จสมบูรณ์ระบบจะแสดงผลบนหน้าต่าง CLI ตามภาพที่ 2.3

```
>> Searching for system dependency llvm-dev - OK
>> Searching for system dependency clang-dev - OK
>> Searching for system dependency qt - OK
>> Searching for system dependency g++ - OK
>> Searching for system dependency cxxfilt - Problem
> Problem: Optional dependency, module "cxxfilt" not available
This may reduce the functionality of the final build.
However, bake will continue since "cxxfilt" is not an essential dependency.
For more information call bake with -v or -vvv, for full verbose mode.

>> Searching for system dependency setup tools - OK
>> Searching for system dependency gi-cairo - Problem
> Problem: Optional dependency, module "gi-cairo" not available
This may reduce the functionality of the final build.
However, bake will continue since "gi-cairo" is not an essential dependency.
For more information call bake with -v or -vvv, for full verbose mode.

>> Searching for system dependency gir-bindings - OK
>> Searching for system dependency pygobject - OK
>> Searching for system dependency cmake - OK
>> Downloading castxml-0.3.6 (target directory: castxml-0.3.6) - (Nothing to do, source directory already exists) - OK
>> Downloading netanim-3.108 - (Nothing to do, source directory already exists) - OK
>> Downloading pygccxml-2.0.1 - (Nothing to do, source directory already exists) - OK
>> Downloading pybindgen-0.21.0.post15+nga587377 (target directory: pybindgen) - (Nothing to do, source directory already exists) - OK
>> Downloading ns-3.33 (target directory: ns-3.33) - OK
>> Building castxml-0.3.6 - OK
>> Building netanim-3.108 - OK
>> Building pygccxml-2.0.1 - Problem
> Problem: Optional dependency, module "pygccxml-2.0.1" failed
This may reduce the functionality of the final build.
However, bake will continue since "pygccxml-2.0.1" is not an essential dependency.
For more information call bake with -v or -vvv, for full verbose mode.

>> Building pybindgen-0.21.0.post15+nga587377 - OK
>> Building ns-3.33 - OK
thananop@thananop-OptiPlex-9020:~$
```

ภาพที่ 2.3 หน้าจอแสดงการติดตั้งโปรแกรม NS3 เสร็จสมบูรณ์

หลังจากติดตั้งโปรแกรม NS3 เรียบร้อยแล้ว ในขั้นตอนต่อไปจะเป็นการตรวจสอบความสมบูรณ์ของการติดตั้งโปรแกรม เพื่อยืนยันว่าตัวโปรแกรมสามารถใช้งานได้อย่างปกติหรือไม่ โดยให้ผู้ใช้ย้ายไปที่ Directory ของโฟลเดอร์ที่โปรแกรมถูกติดตั้งเอาไว้ สำหรับตัวอย่างนี้โปรแกรม NS3 จะถูกเก็บเอาไว้ที่ /source/ns3 จากนั้นให้ผู้ใช้เรียกใช้คำสั่งดังต่อไปนี้

```
> ./waf configure --enable-tests --enable-examples
```

โดยชุดคำสั่งดังกล่าวจะเป็นการตั้งค่าให้โปรแกรม NS3 สามารถถูกตรวจสอบได้ผ่านชุดคำสั่งของไพธอน (ซึ่งจะถูก disable ไว้แบบ Default) จากนั้นจึงเรียกใช้งานชุดโค้ดไพธอนเพื่อทำการตรวจสอบความถูกต้องของการติดตั้งโปรแกรมผ่านคำสั่งต่อไปนี้

```
> ./test.py
```

โดยคำสั่งดังกล่าวจะทำการส่งคำสั่งเพื่อรันชุดสคริปต์ทดสอบทั้งหมดเพื่อตรวจสอบว่าทุก ๆ โมดูลบนโปรแกรม NS3 นั้นสามารถทำงานได้อย่างถูกต้อง หากการทดสอบทั้งหมดสำเร็จจุล่งจะมีการแสดงผลบนหน้าจอ CLI ตามภาพที่ 2.4 โดยผู้ใช้อาจจะสามารถสังเกตได้ว่ามีสคริปต์ที่ใช้ในการทดสอบบางส่วนที่ถูกข้ามการทดสอบไป เนื่องจากไม่มีการติดตั้งส่วนต่อขยาย NSC (Network Simulation Cradle) ในโปรแกรม (เนื่องจากส่วนต่อขยายนี้ได้หยุดการพัฒนาไปแล้ว) ซึ่งการที่ไม่ได้ทำการติดตั้งส่วนขยายดังกล่าวจะไม่ส่งผลต่อการทำงานของโปรแกรมแต่อย่างใด

```
[665/680] PASS: Example src/wimax/examples/wimax-ipv4
[666/680] PASS: Example src/wimax/examples/wimax-multicast
[667/680] PASS: Example examples/tutorial/first.py
[668/680] PASS: Example examples/wireless/wifi-ap.py
[669/680] PASS: Example examples/wireless/mixed-wired-wireless.py
[670/680] PASS: Example examples/routing/simple-routing-ping6.py
[671/680] PASS: Example src/bridge/examples/csma-bridge.py
[672/680] PASS: Example src/core/examples/sample-simulator.py
[673/680] PASS: Example src/wifi/examples/wifi-manager-example --wifiManager=Ideal --standard=802.11ax-5GHz -
ientNss=1 --stepTime=0.1
[674/680] PASS: Example src/wifi/examples/wifi-manager-example --wifiManager=Ideal --standard=802.11ax-2.4GHz
ientNss=2 --stepTime=0.1
[675/680] PASS: Example src/wifi/examples/wifi-manager-example --wifiManager=Ideal --standard=802.11ax-5GHz -
clientNss=1 --stepTime=0.1
[676/680] PASS: Example src/wifi/examples/wifi-manager-example --wifiManager=Ideal --standard=802.11ax-5GHz -
clientNss=1 --stepTime=0.1
[677/680] PASS: Example src/flow-monitor/examples/wifi-olsr-flowmon.py
[678/680] PASS: Example src/wifi/examples/wifi-bianchi --validate --phyRate=6 --nMinStas=5 --nMaxStas=10 --du
[679/680] PASS: Example src/wifi/examples/wifi-manager-example --wifiManager=Ideal --standard=802.11ax-5GHz -
clientNss=4 --stepTime=0.1
[680/680] PASS: Example src/wifi/examples/wifi-manager-example --wifiManager=Ideal --standard=802.11ax-5GHz -
clientNss=4 --stepTime=0.1
677 of 680 tests passed (677 passed, 3 skipped, 0 failed, 0 crashed, 0 valgrind errors)
List of SKIPPed tests:
  ns3-tcp-cwnd (requires NSC)
  ns3-tcp-interoperability (requires NSC)
  nsc-tcp-loss (requires NSC)
thananop@thananop-OptiPlex-9020:~/source/ns-3.33$
```

ภาพที่ 2.4 หน้าจอแสดงการทดสอบโปรแกรม NS3 เสร็จสมบูรณ์

ในขั้นตอนต่อไปจะเป็นการทดสอบขั้นสุดท้าย ด้วยการเรียกใช้สคริปต์คำสั่งบนโปรแกรม NS3 จากผู้ใช้โดยตรงโดยการเรียกคำสั่งต่อไปนี้

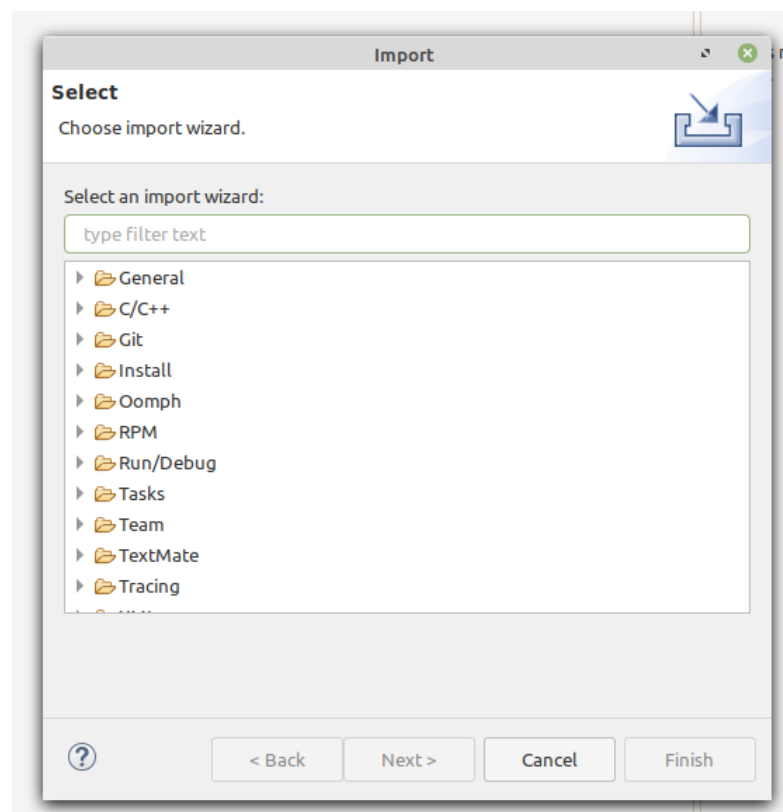
```
> ./waf --run hello-simulator
```

หากหน้าจอ CLI มีการแสดงผลด้วยข้อความ “Hello Simulator” นั้นแสดงถึงการติดตั้งโปรแกรมทั้งหมดนั้นเสร็จสมบูรณ์ ในส่วนถัดไปจะเป็นการอธิบายการใช้งานโปรแกรม NS3 ในเบื้องต้น รวมไปถึงการสร้างไฟล์ที่จะใช้ในการจำลองระบบเครือข่าย

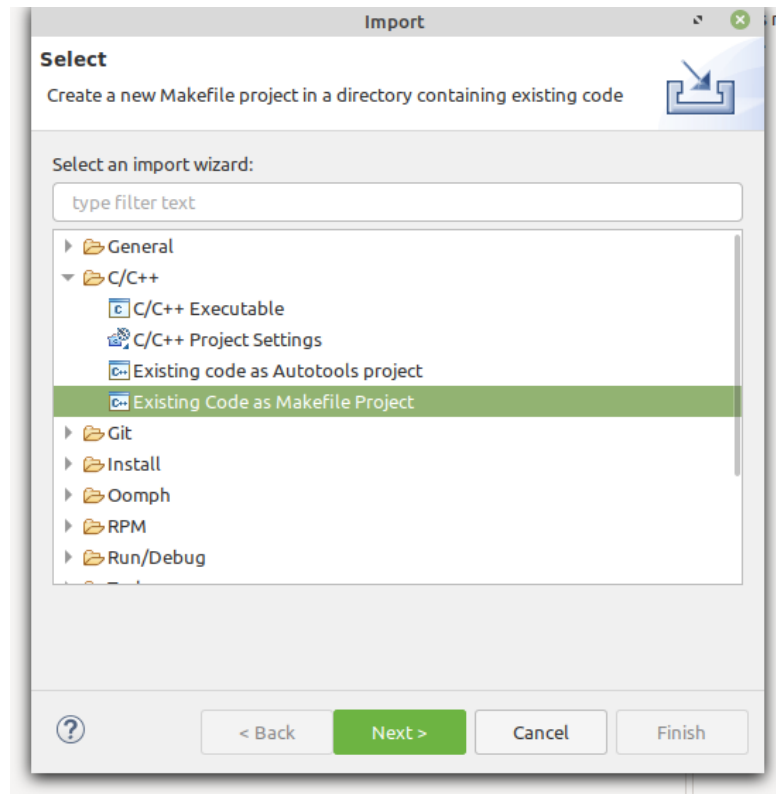
2.3 การติดตั้งโปรแกรม Eclipse เพื่อใช้งานร่วมกับโปรแกรม NS3 และการตั้งค่าเพิ่มเติมเพื่อการใช้งาน Debugger

เพื่อความสะดวกต่อการใช้งาน อ่าน และแก้ไขโค้ดบนโปรแกรม NS3 ผู้ใช้สามารถติดตั้งโปรแกรม Editor เพื่อเข้ามาช่วยจัดระเบียบชุดคำสั่งต่าง ๆ บนโปรแกรมได้ โดยในส่วนตัวของผู้เขียนนั้นเลือกใช้งานโปรแกรม Eclipse เป็นโปรแกรม Editor

ในส่วนนี้ผู้เขียนจะแสดงวิธีการติดตั้งโปรแกรม Eclipse รวมไปถึงแสดงการตั้งค่าต่าง ๆ ที่สำคัญ ในขั้นแรกนั้นให้ผู้ใช้ดาวน์โหลด และติดตั้งโปรแกรม Eclipse สำหรับการพัฒนาภาษา C/C++ เวอร์ชันล่าสุด โดยสามารถดาวน์โหลดตัว Installer สำหรับติดตั้งโปรแกรมได้จาก <https://www.eclipse.org/downloads>

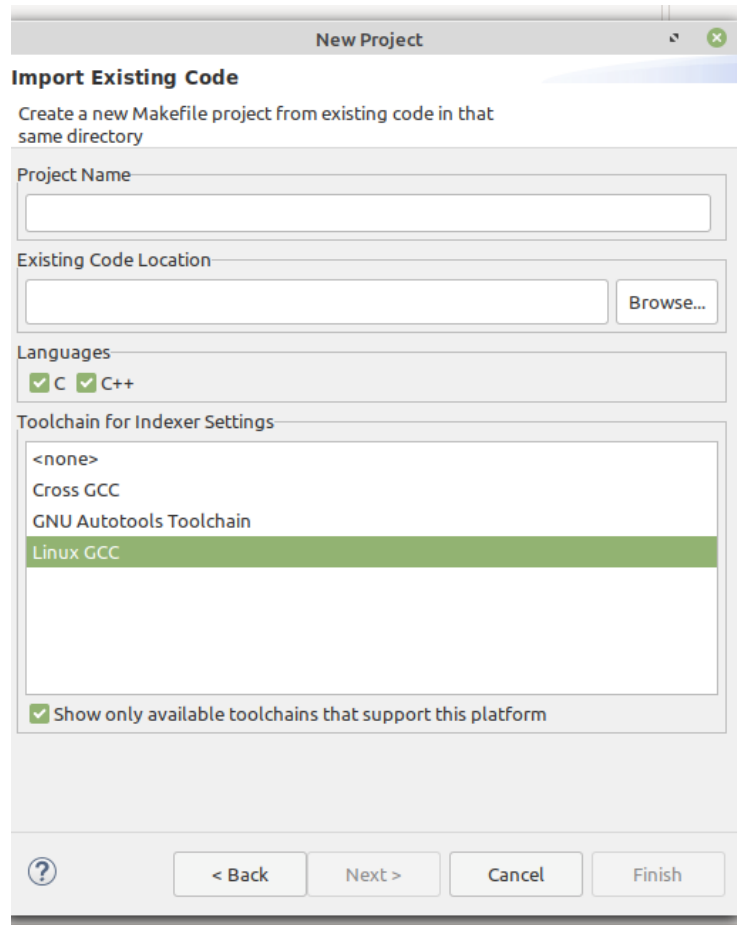


ภาพที่ 2.5 หน้าจอ Import Project บนโปรแกรม Eclipse



ภาพที่ 2.6 หน้าจอแสดงตัวเลือกเพื่อการ Import Existing Code บนโปรแกรม Eclipse

หลังจากที่ผู้ใช้ทำการติดตั้งโปรแกรมเรียบร้อยแล้ว ต่อไปผู้ใช้เปิดโปรแกรมขึ้นมา และเลือกคำสั่ง File -> Import ซึ่งตัวโปรแกรมจะแสดงผลตามภาพที่ 2.5 ต่อไปให้ผู้ใช้เลือกตัวเลือกตามภาพที่ 2.6 จากนั้นให้ผู้ใช้กดปุ่ม Next เพื่อไปยังหน้าถัดไป

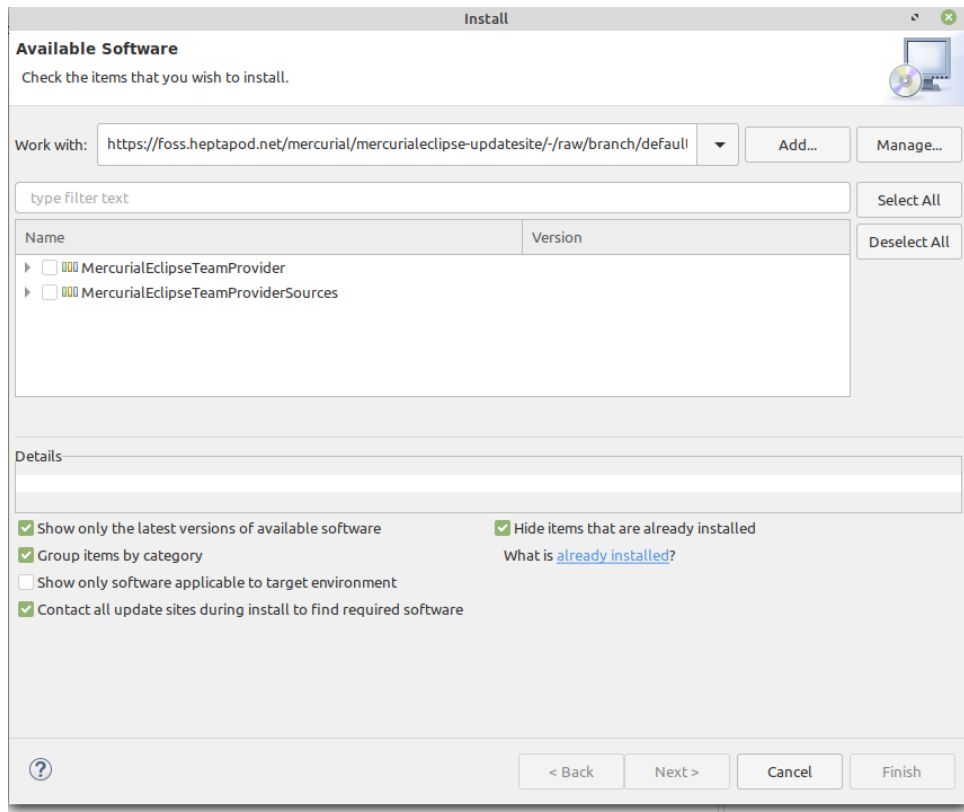


ภาพที่ 2.7 หน้าจอแสดงรายละเอียดการ Import Existing Code บนโปรแกรม Eclipse

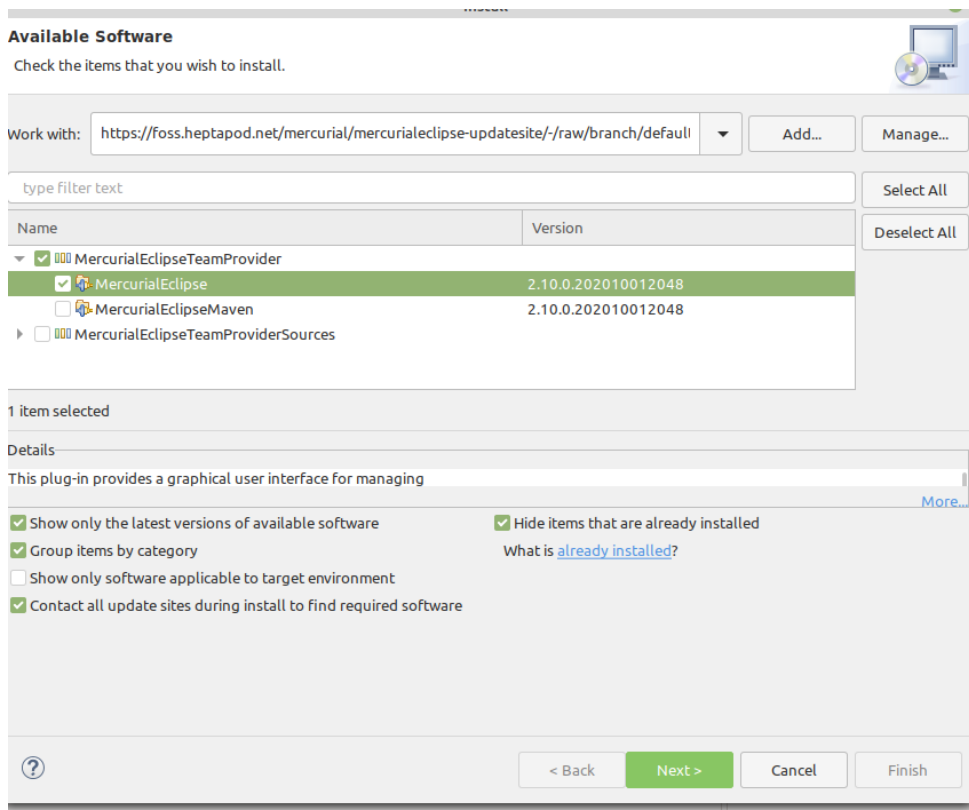
ในหน้าต่างนี้ให้ผู้ใช้กดปุ่ม Browse เพื่อเลือกโฟลเดอร์ที่เก็บชุดคำสั่งของโปรแกรม NS3 โดยในการติดตั้งของผู้เขียนจะเก็บไว้ที่โฟลเดอร์ source/ns-3.33 ในส่วนของการเลือก Toolchain for indexer Settings ผู้เขียนเลือกใช้เป็น Linux GCC ตามที่แสดงไว้ในภาพที่ 2.7 หลังจากนั้นให้ผู้ใช้กดปุ่ม Finish ก็จะเสร็จสิ้นกระบวนการเปิดโปรเจกต์ NS3 ขึ้นมาบนโปรแกรม Eclipse

จากนั้นผู้ใช้จะต้องติดตั้งส่วนต่อขยาย Mercurial เพิ่มเติมซึ่งจำเป็นต่อการใช้งานโปรแกรม โดยให้ผู้ใช้เลือกปุ่ม Help -> Install New software จากบน Toolbar ของโปรแกรม จากนั้นให้กรอก URL ดังต่อไปนี้ลงไปในฟิลด์ Work with: และกดปุ่ม Add ตามภาพที่ 2.8

<https://foss.heptapod.net/mercurial/mercurialeclipse-updatesite/-/raw/branch/default/p2>

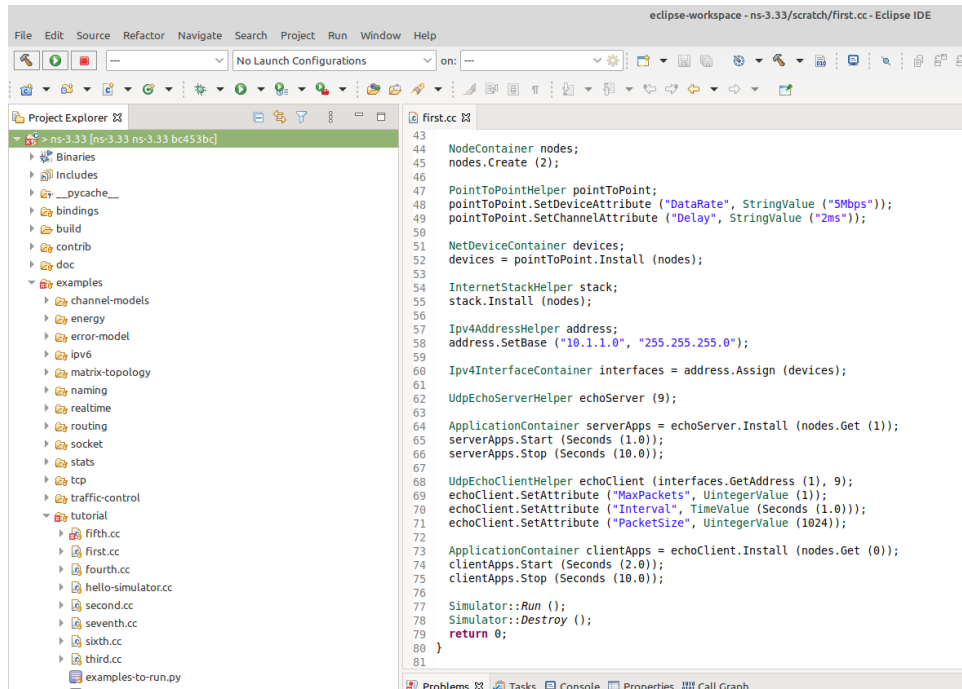


ภาพที่ 2.8 หน้าต่างที่ใช้เพื่อการติดตั้งส่วนขยาย Mercurial ลงไปบนโปรแกรม Eclipse

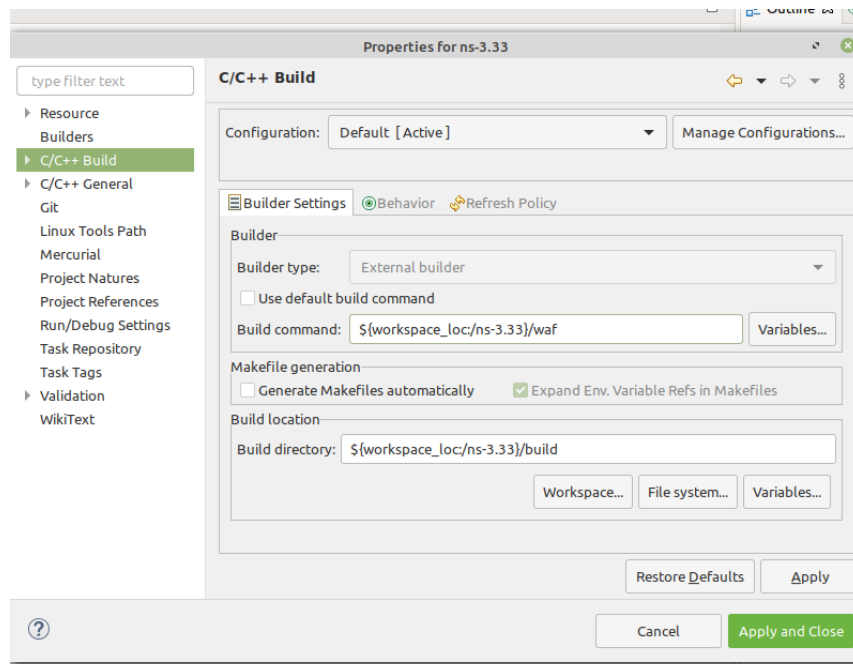


ภาพที่ 2.9 หน้าต่างที่ใช้เพื่อการติดตั้งส่วนขยาย Mercurial ลงไปบนโปรแกรม Eclipse (ต่อ)

จากนั้นจะปรากฏแท็บให้เลือกติดตั้งส่วนต่อขยายขึ้นมา ให้ผู้ใช้เลือก Mercurial eclipse และกดปุ่ม Next ตามภาพที่ 2.9 หลังจากนั้นผู้ใช้จะต้องยอมรับข้อตกลงของการใช้งานส่วนต่อขยาย และทำการติดตั้ง หลังจากการติดตั้งส่วนขยายเสร็จสิ้น ตัวโปรแกรมจะทำการปิดและเปิดใหม่เพื่อให้การติดตั้งเสร็จสมบูรณ์

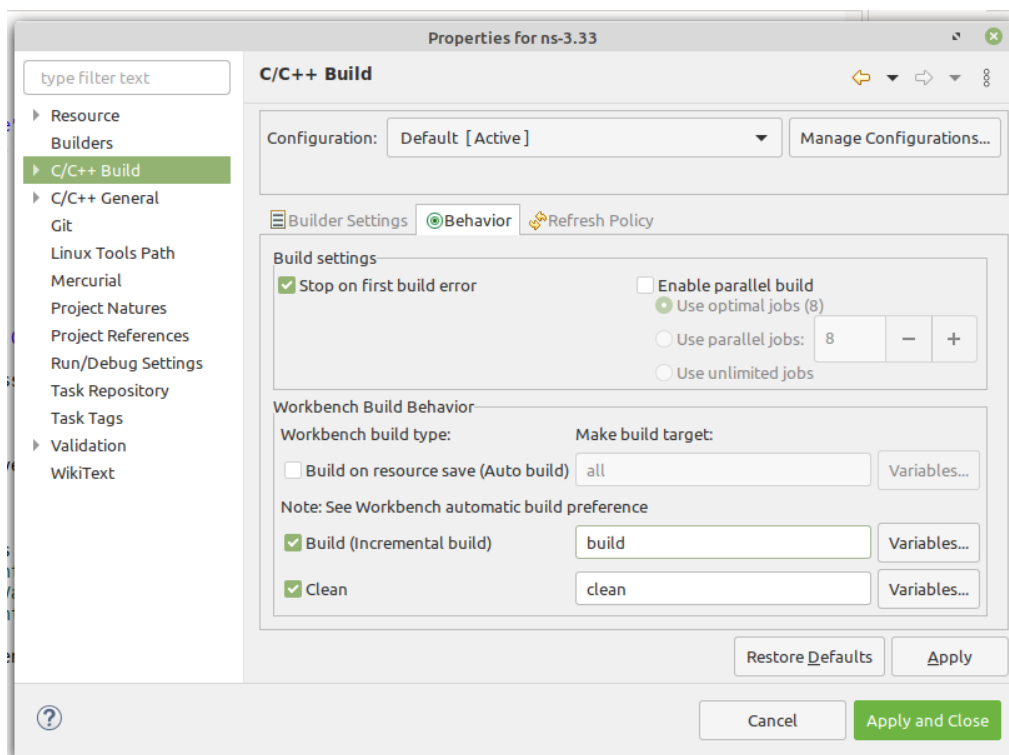


ภาพที่ 2.10 โฟลเดอร์โปรเจกต์ NS3.33



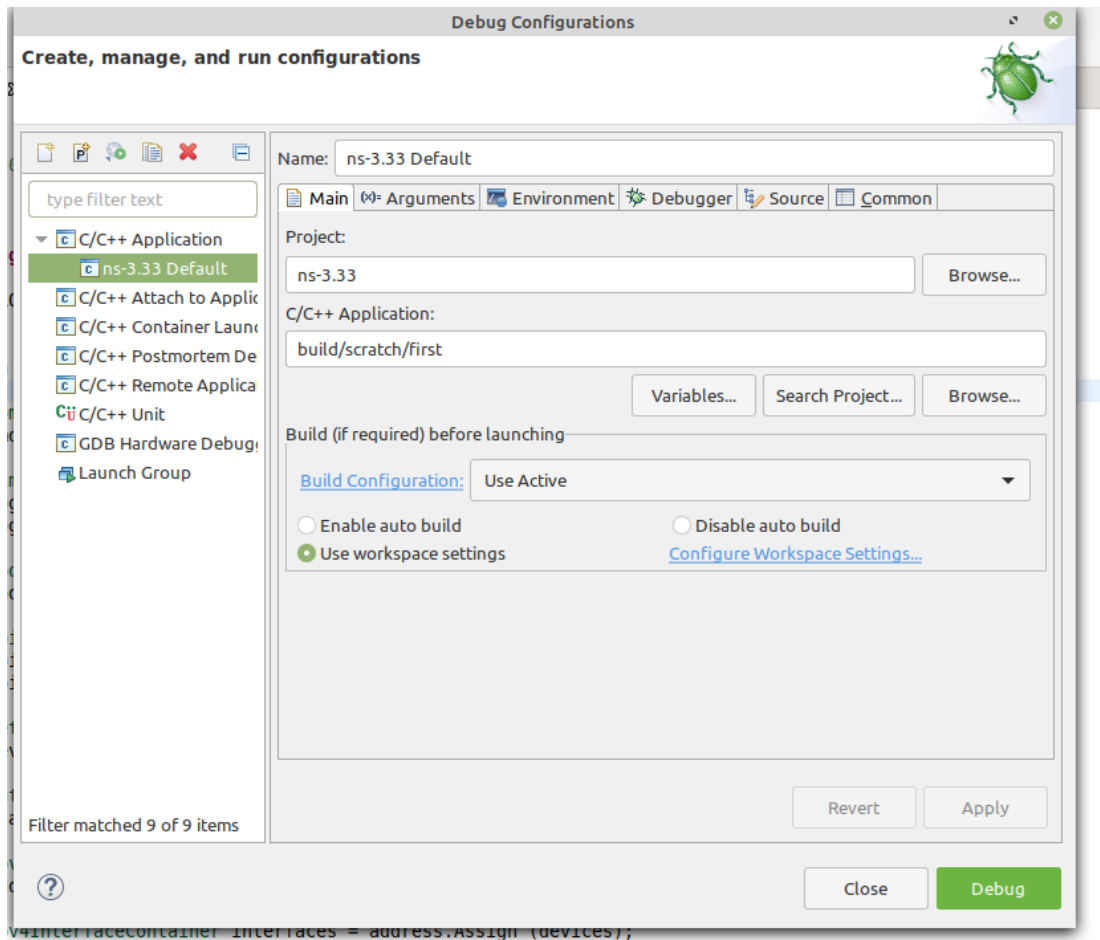
ภาพที่ 2.11 หน้าต่างการตั้งค่าเพื่อการ Build โปรเจกต์

จากนั้นจะเป็นการตั้งค่า Waf Builder บนโปรแกรม Eclipse เพื่อให้ผู้ใช้สามารถคอมไพล์ชุดคำสั่งต่าง ๆ ของ NS3 ผ่านโปรแกรม eclipse ได้โดยตรง โดยให้ผู้ใช้คลิกขวาที่โปรเจกต์ NS3 ตามภาพที่ 2.10 แล้วคลิกที่ตัวเลือก Property จากนั้นบนหน้าต่าง Property ให้ผู้ใช้เลือกไปที่แท็บ C/C++ Build แล้วทำการยกเลิกการเลือก “Use default build command” ออก จากนั้นให้กรอกข้อมูลตามภาพที่ 2.11 ซึ่งค่า Directory Path ของแต่ละเครื่องจะขึ้นอยู่กับการติดตั้งโปรแกรม NS3 ของผู้ใช้งานว่ากำหนดให้ไปอยู่ที่โฟลเดอร์ใด จากนั้นให้ผู้ใช้ไปที่แท็บ “Behavior” แล้วเปลี่ยนค่าในฟิลด์ “Build (Incremental build)” จาก all เป็น build ตามภาพที่ 2.12 จากนั้นให้ผู้ใช้เลือก Apply and Close



ภาพที่ 2.12 หน้าต่างการตั้งค่าเพื่อการ Build โปรเจกต์ (แท็บ Behavior)

ในส่วนสุดท้ายนั้นคือการตั้งค่า Debugger สำหรับโปรแกรม Eclipse เพื่อใช้ในการช่วยค้นหาปัญหา หรือไล่เรียงลำดับเหตุการณ์การทำงานของชุดคำสั่งต่าง ๆ บนโปรแกรม NS3 ซึ่งจะมีประโยชน์อย่างมากต่อการพัฒนาชุดคำสั่งในอนาคต

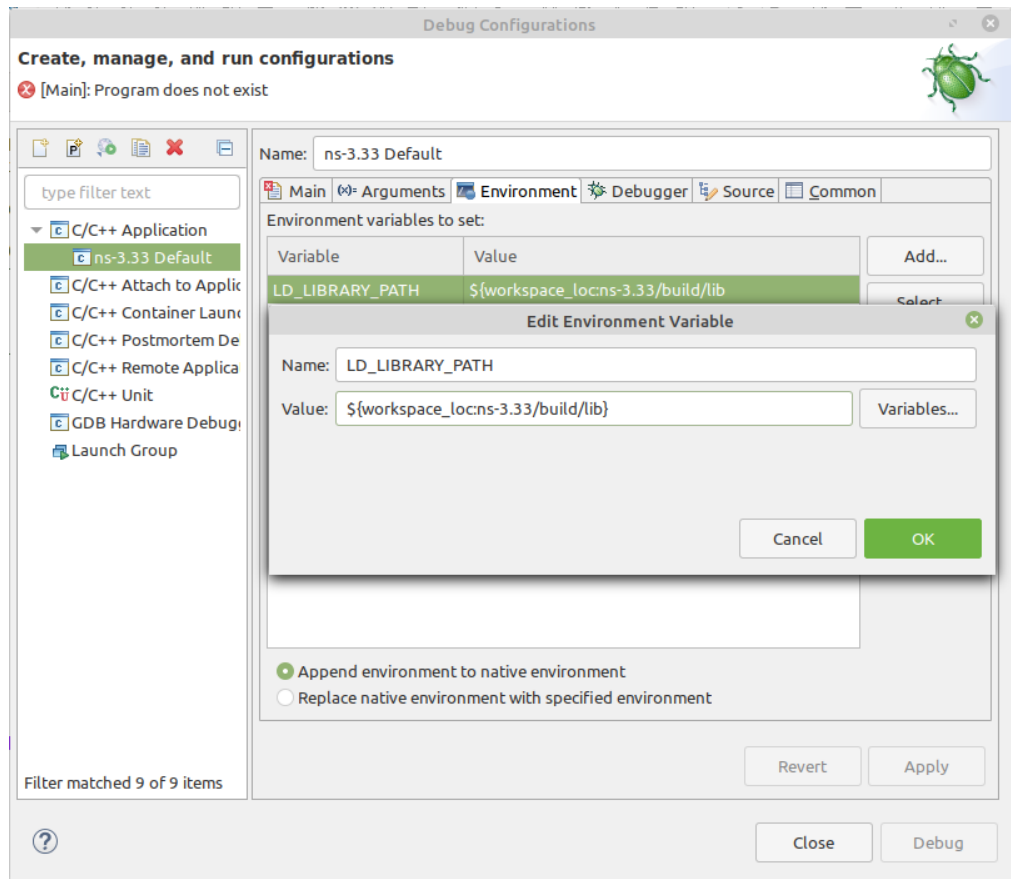


ภาพที่ 2.13 หน้าต่างการตั้งค่าเพื่อการ Debug โปรเจกต์

โดยให้ผู้ใช้ไปที่แท็บ Run -> Debug Configurations และตั้งชื่อใหม่ให้กับตัว Debug ตามภาพที่ 2.13 (ฟิลด์ Name) โดยในส่วนของฟิลด์ “C/C++ Application” จะเป็นการกำหนด Directory Path ของไฟล์ที่เราต้องการจะ Debug (ในตัวอย่างนี้คือไฟล์ first ในโฟลเดอร์ scratch) จากนั้นให้เลือกไปยังแท็บ Environment และกดปุ่ม Add ตามข้อมูลในภาพที่ 2.14 จากนั้นให้กดปุ่ม Apply เป็นอันเสร็จสิ้นการตั้งค่า

หลังจากนั้นผู้ใช้สามารถทำการทดลองการ Debug ได้ด้วยการกดปุ่ม F11 หากการตั้งค่าทั้งหมดถูกต้องตัวโปรแกรม Eclipse จะร้องขอการสลับมุมมองไปยังมุมมองการ Debug ซึ่งแสดงว่าการตั้งค่าเสร็จสมบูรณ์

โดยเมื่อผู้ใช้ต้องการจาก Debug สคริปต์ที่ต้องการให้ผู้ใช้เข้ามาปรับค่า Application ให้สอดคล้องกับสคริปต์ที่ผู้ใช้ต้องการจะ Debug ก่อนที่ผู้ใช้จะทำการเรียกใช้คำสั่งเพื่อการ Debug ต่อไป



ภาพที่ 2.14 หน้าต่างการตั้งค่าเพื่อการ Debug โปรเจกต์ (แท็บ Environment)

บทที่ 3

ภาพรวมในการใช้งานโปรแกรม NS3 เบื้องต้น

3.1 ภาพรวมการใช้งานโปรแกรม NS3

หากผู้ใช้งานเคยมีประสบการณ์จากการใช้งานโปรแกรม Packet Tracer เพื่อการจำลองระบบเครือข่ายอยู่แล้ว ความแตกต่างระหว่างโปรแกรม NS3 และโปรแกรม Packet Tracer นั้นคือ Packet Tracer จะมีลักษณะการกำหนดสถานการณ์ในการจำลอง และสร้างอุปกรณ์เครือข่ายผ่านวิธีการ drag and drop โดยที่ตัวโปรแกรมจะมี GUI ค่อยช่วยเหลือในการกำหนดค่าต่าง ๆ เพื่ออำนวยความสะดวกให้แก่ผู้ใช้งาน แต่ในส่วนของโปรแกรม NS3 นั้นการกำหนดค่าทั้งหมดจะต้องทำผ่านการเขียนสคริปต์เท่านั้น ซึ่งช่วยเพิ่มความยืดหยุ่นต่อการจำลองระบบเครือข่าย เนื่องจากผู้ใช้งานสามารถพัฒนาโมดูลจำเพาะ หรือโปรโตคอลการสื่อสารชนิดใหม่ภายในโปรแกรม NS3 และนำมาทดสอบเพื่อวัดประสิทธิภาพได้ ทำให้โปรแกรม NS3 มีความเหมาะสมต่อการใช้เป็นเครื่องมือที่เข้ามาช่วยในการทำวิจัยเป็นอย่างมาก อย่างไรก็ตามการที่โปรแกรม NS3 ไม่มีหน้าต่าง GUI คอยสนับสนุนก็อาจจะเป็นอุปสรรคต่อผู้เริ่มต้นใช้งานเช่นกัน

ในการจำลองระบบเครือข่ายบนโปรแกรม NS3 นั้น ผู้ใช้งานจะต้องกำหนดสถานการณ์ และสภาพแวดล้อมเพื่อการจำลองเครือข่าย ซึ่งจะประกอบไปด้วย รายละเอียดของอุปกรณ์เครือข่ายที่ติดตั้งอยู่บนโหนด จำนวนโหนด รูปแบบการเชื่อมต่อระหว่างโหนด รูปแบบการเคลื่อนที่ระยะเวลาในการจำลอง รวมไปถึงพฤติกรรมต่าง ๆ ของโหนด ผ่านการเขียนสคริปต์ภาษา C ขึ้นมา

โดยหลังจากการสร้างสคริปต์เพื่อกำหนดสถานการณ์เรียบร้อยแล้ว ผู้ใช้จะเรียกใช้งานสคริปต์ผ่านชุดคำสั่ง ./waf ซึ่งถูกพัฒนามาจากโปรแกรมภาษาไพธอน โดยผู้ใช้งานไปศึกษาข้อมูลเพิ่มเติมของชุดคำสั่งนี้ได้ผ่าน URL: <https://gitlab.com/ita1024/waf/> โดยในคู่มือฉบับนี้จะกล่าวถึงชุดคำสั่งพื้นฐานที่จำเป็นเท่านั้น หลังจากเรียกใช้ชุดคำสั่ง waf เพื่อเรียกใช้สคริปต์ที่ผู้ใช้สร้างขึ้นแล้ว ตัวโปรแกรม NS3 ก็จะเข้าไปเรียก Model หรือ Library ต่าง ๆ ที่เกี่ยวข้องกับ การจำลองเหตุการณ์มาสร้างอุปกรณ์ และเหตุการณ์จำลองตามที่ผู้ใช้กำหนด โดยผลลัพธ์ของการจำลองเครือข่ายจะถูกแสดงออกมาผ่านไฟล์ใน 3 รูปแบบซึ่งผู้ใช้สามารถกำหนดได้ในสคริปต์การจำลองอันได้แก่

1. ASCII File มีลักษณะเป็น Log ไฟล์ที่มีนามสกุลของไฟล์คือ .tr ที่จะแสดงทุก ๆ เหตุการณ์ที่เกิดขึ้นในการจำลองระบบเครือข่าย โดยจะถูกกำกับด้วยเวลาที่เกิดเหตุการณ์นั้น ๆ โดยผู้ใช้งานนำไฟล์นี้ไปสกัดเอาข้อมูลที่ผู้สนใจมาเพื่อใช้ในการวิเคราะห์ประสิทธิภาพของเครือข่ายได้

2. PCAP File เป็นไฟล์ที่สามารถนำไปใช้ร่วมกับโปรแกรม Wireshark เพื่อใช้ในการตรวจสอบการรับส่งข้อความต่าง ๆ ที่เกิดขึ้นจากการจำลองเครือข่าย และสามารถนำไฟล์ประเภทนี้ไปใช้เพื่อวิเคราะห์ประสิทธิภาพของเครือข่ายได้เช่นกัน
3. ANIM file เป็นไฟล์ที่จะบันทึกเหตุการณ์ต่าง ๆ ในการจำลองระบบเครือข่าย ออกมาเป็นไฟล์ซึ่งมีนามสกุล .xml โดยผู้ใช้สามารถเปิดไฟล์ .xml นี้ผ่านโปรแกรม Network Animator (NetAnim) เพื่อแสดงผลการจำลองเครือข่ายออกมาในรูปแบบกราฟิกอย่างง่าย ซึ่งจะช่วยให้ผู้ใช้สามารถตรวจสอบความถูกต้องของการกำหนดค่าต่าง ๆ ในสคริปต์ รวมไปถึงช่วยในการวิเคราะห์ผลลัพธ์ที่เกิดขึ้นจากการจำลองเครือข่ายได้

นอกจากไฟล์ที่ใช้ในการแสดงผลการจำลองทั้ง 3 รูปแบบที่ผู้เขียนได้นำเสนอไปแล้วนั้น ผู้ใช้ยังสามารถเก็บผลลัพธ์ของการจำลองผ่านการแทรกชุดคำสั่งเพื่อการเก็บค่าแอททริบิวท์ที่ผู้ใช้สนใจลงไปโปรแกรม NS3 โดยตรงได้เช่นกัน ซึ่งการเก็บข้อมูลในลักษณะนี้จะช่วยลดขั้นตอนการวิเคราะห์ Log file ลงได้ โดยผู้ใช้งานระบบสามารถเลือกวิธีการในการเก็บผลลัพธ์ได้อย่างอิสระ โดยสรุปแล้วขั้นตอนในการจำลองระบบเครือข่ายจะแบ่งออกเป็น 4 ขั้นตอนดังนี้

1. ผู้ใช้เขียนสคริปต์ที่ใช้กำหนดสถานการณ์การจำลองระบบเครือข่าย
2. ผู้ใช้ทำการจำลองระบบเครือข่ายผ่านการรันสคริปต์ด้วยชุดคำสั่ง WAF
3. ระบบจะสร้าง log file ตามที่ผู้ใช้กำหนด ซึ่งเก็บทุก ๆ เหตุการณ์ที่เกิดขึ้นจากการจำลองเครือข่าย
4. ผู้ใช้นำไฟล์ที่ได้ไปวิเคราะห์เพื่อใช้ในการสรุปผล และวิเคราะห์ประสิทธิภาพของเครือข่าย

โดยในส่วนถัดไปผู้เขียนจะนำเสนอทั้ง 4 ขั้นตอนเพื่อการจำลองระบบเครือข่ายผ่านตัวอย่างเบื้องต้นซึ่งโปรแกรม NS3 ได้จัดเตรียมเอาไว้ให้แก่ผู้ใช้งาน

3.2 การทดลองเริ่มเรียกใช้งานสคริปต์พื้นฐานเพื่อการจำลองระบบเครือข่าย

ผู้ใช้งานจะสามารถค้นหาไฟล์เพื่อทดลองการใช้งานโปรแกรม NS3 เบื้องต้นได้จาก Directory: ns3.33/examples ซึ่งการทำความเข้าใจกับสคริปต์การจำลองเหตุการณ์เหล่านี้จะช่วยให้ผู้ใช้มีความเข้าใจกับโปรแกรม NS3 มากยิ่งขึ้น โดยในการรันสคริปต์เหล่านี้ผู้ใช้สามารถเรียกใช้งานได้ด้วยการเรียกคำสั่งผ่าน Command Prompt ซึ่งมีรูปแบบดังต่อไปนี้

```
> ./waf --run "<Directory Path> / <script name> --<attribute name>=  
<attribute value>"
```

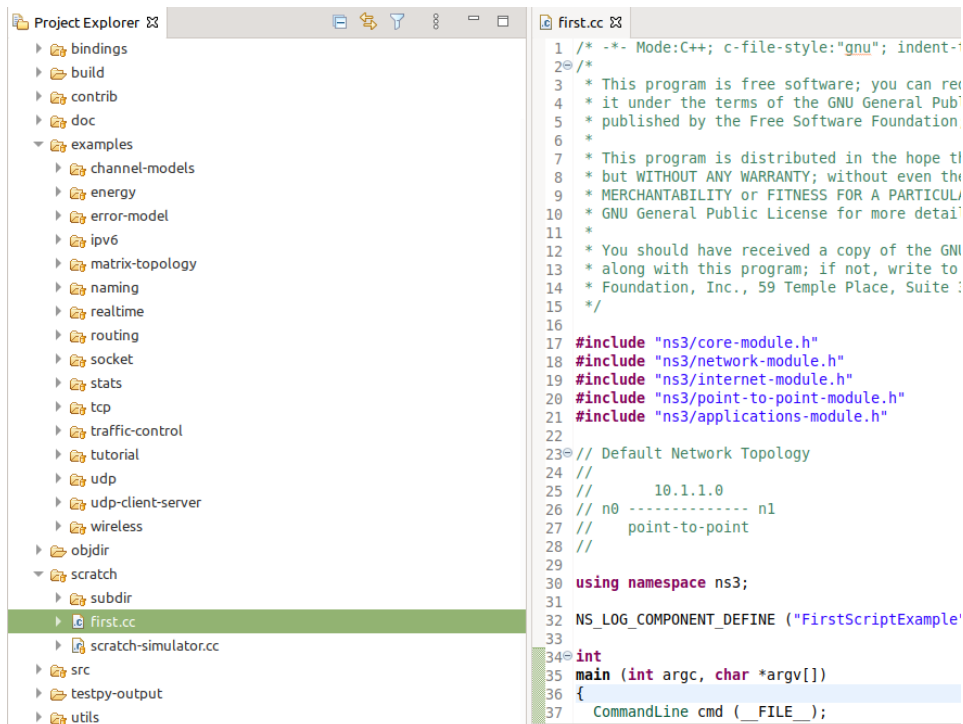
```
File Edit View Search Terminal Help
thananop@thananop-OptiPlex-9020:~/source/ns-3.33$ ./waf --run examples/tutorial/first
Waf: Entering directory `~/home/thananop/source/ns-3.33/build'
Waf: Leaving directory `~/home/thananop/source/ns-3.33/build'
Build commands will be stored in build/compile_commands.json
'build' finished successfully (0.884s)
At time +2s client sent 1024 bytes to 10.1.1.2 port 9
At time +2.00369s server received 1024 bytes from 10.1.1.1 port 49153
At time +2.00369s server sent 1024 bytes to 10.1.1.1 port 49153
At time +2.00737s client received 1024 bytes from 10.1.1.2 port 9
thananop@thananop-OptiPlex-9020:~/source/ns-3.33$
```

ภาพที่ 3.1 การทดสอบการรันสคริปต์ตัวอย่าง first.cc

ยกตัวอย่างเช่น หากผู้ใช้งานต้องการเรียกใช้งานสคริปต์ที่ชื่อว่า first ซึ่งถูกเก็บอยู่ใน Directory /example/tutorial ผู้ใช้สามารถเรียกใช้งานผ่านคำสั่งตามที่แสดงไว้ในภาพที่ 3.1 และเนื่องจากสคริปต์ first ไม่ต้องการการป้อนค่าตัวแปรจากผู้ใช้ ในการเรียกใช้คำสั่งจึงไม่ต้องป้อนค่าตัวแปรต่อท้าย และไม่จำเป็นต้องครอบชุดคำสั่งด้วยเครื่องหมาย Quotation (“”) แต่ในกรณีที่ผู้ใช้ต้องการจะป้อนค่าตัวแปรให้กับสคริปต์ (หากสคริปต์มีการกำหนดให้รับค่าตัวแปรได้) ก็จะมีรูปแบบการเรียกคำสั่ง ยกตัวอย่างเช่น

```
> ./waf --run "scratch/manet-routing-compare.cc --protocol=1"
```

โดยในการเรียกสคริปต์นี้ จะมีการใส่ค่าตัวแปรลงไปโดยกำหนดให้ค่าของตัวแปร protocol มีค่าเท่ากับ 1 เป็นต้น อย่างไรก็ตามในขั้นตอนของการพัฒนาโปรแกรมจำลองเครือข่ายซึ่งผู้ใช้อาจต้องการเขียนสคริปต์ด้วยตนเอง หรือการนำสคริปต์ตัวอย่างที่ถูกเก็บไว้ใน Directory /example มาแก้ไข ผู้เขียนแนะนำให้ผู้ใช้สร้าง หรือคัดลอกสคริปต์แล้วนำมาเก็บไว้ที่โฟลเดอร์ Scratch ตามภาพที่ 3.2 และเรียกใช้งานสคริปต์ผ่านโฟลเดอร์ scratch เพื่อป้องกันไม่ให้ไฟล์ตัวอย่างของระบบถูกแก้ไขโดยตรง



ภาพที่ 3.2 การคัดลอกไฟล์ first.cc เข้ามาเก็บไว้ในโฟลเดอร์ scratch เพื่อใช้งาน หรือแก้ไข

ในส่วนนี้จะเป็นการอธิบายการทำงานของสคริปต์ first.cc ซึ่งทำให้ปรากฏผลลัพธ์การจำลองผ่านหน้าจอ CLI ตามภาพที่ 3.1 โดยในสคริปต์ first.cc นั้นจะเป็นการสร้างโหนด 2 โหนดขึ้นมาในระบบ จากนั้นจึงเชื่อมต่อโหนดทั้ง 2 ตัวนี้เข้าด้วยกันผ่านการเชื่อมต่อแบบมีสาย (Point to point connection) จากนั้นจึงมีการเรียกใช้งานแอปพลิเคชัน <ECHO> เพื่อให้เกิดการสื่อสารที่มีความคล้ายคลึงกับคำสั่ง Ping ที่มีการใช้งานอยู่ทั่วไป โดยที่โหนดที่ถูกกำหนดให้เป็น Client จะส่งข้อความออกไปหาโหนดที่ทำหน้าที่เป็น Server หลังจากที่ได้รับข้อความก็จะทำการตอบข้อความกลับไปหาโหนดที่เป็น Client โดยผู้ใช้งานจะสามารถสังเกตผลลัพธ์จากหน้าจอ CLI ว่าโหนดต้นทางมีการส่งข้อความออกไปในวินาทีที่เท่าไร ไปหาโหนดที่มีไอพีแอดเดรสอะไร โดยผ่านพอร์ตใด และโหนดปลายทางได้รับข้อความในวินาทีที่เท่าไร และมีการตอบกลับเมื่อไหร่ โดยค่าทั้งหมดนี้ผู้ใช้งานจะสามารถกำหนดเองได้อย่างอิสระ ในส่วนถัดไปผู้ใช้งานจะใช้ตัวอย่าง first.cc เป็นตัวอย่างในการอธิบายการสร้างโหนด และการกำหนดค่าต่างๆ ของสคริปต์สำหรับโปรแกรม NS3

3.3 การสร้างโหนดในเครือข่ายเบื้องต้น และการกำหนดคุณสมบัติของอุปกรณ์ (Conceptual Overview)

ในการเขียนสคริปต์เพื่อกำหนดค่าการจำลองระบบเครือข่ายผ่านโปรแกรม NS3 นั้นจะมีองค์ประกอบหลัก ๆ ดังต่อไปนี้

1. Node หรือสามารถเรียกได้อีกอย่างว่า end devices เนื่องจากโปรแกรม NS3 ไม่ใช่โปรแกรมสำหรับการจำลองการเชื่อมต่ออุปกรณ์เครือข่ายที่มีการใช้งานอยู่จริง (ไม่มีชื่อโมเดล รุ่น หรือมาตรฐานมากำกับ) เช่นเดียวกับโปรแกรม Packet Tracer หรือ GNS3 เราจึงจะไม่เรียกอุปกรณ์ end devices ว่าเป็นโฮสต์ แต่จะเรียกเป็นโหนดแทน โดยผู้ใช้สามารถกำหนดคุณสมบัติต่าง ๆ ของโหนดที่สร้างขึ้นได้เองว่าจะให้มีคุณสมบัติอย่างไร
2. Channel หรือช่องทางการสื่อสารระหว่างโหนด โดยผู้ใช้สามารถกำหนดได้ว่าโหนดจะมีการสื่อสารกันอย่างไร ผ่านการเชื่อมต่อแบบมีสาย หรือแบบไร้สาย ซึ่งผู้ใช้สามารถกำหนดคุณลักษณะของช่องทางการสื่อสารเหล่านี้ได้เอง
3. Net Device (Network device) ซึ่งเปรียบเทียบกับ Network Interface Card ในความเป็นจริง โดยผู้ใช้สามารถติดตั้ง Net Device ลงไปบนโหนดเพื่อเพิ่มคุณลักษณะต่าง ๆ ให้กับอุปกรณ์ได้ ยกตัวอย่างเช่นหากผู้ใช้ต้องการสร้างโหนดที่สามารถเชื่อมต่อกับโหนดอื่นได้ทั้งแบบมีสายผ่านสายแลน และเชื่อมต่อแบบไร้สายผ่านเทคโนโลยีไวไฟ ผู้ใช้สามารถสร้าง Net Device 2 ชนิด และติดตั้งลงไปในโหนดที่สร้างขึ้นมาได้
4. Topology / Protocol Helper จะเป็นส่วนที่ใช้ในการกำหนดโปรโตคอลการสื่อสารระหว่างอุปกรณ์ รวมไปถึงการกำหนดค่าไอพีแอดเดรส หรือข้อมูลอื่น ๆ ที่จำเป็นต่อการสื่อสาร ซึ่งจะสอดคล้องกับมาตรฐานที่มีอยู่ในปัจจุบัน โดยผู้ใช้สามารถแก้ไข และปรับเปลี่ยนค่าเหล่านี้ได้
5. Mobility Model เป็นส่วนที่กำหนดตำแหน่ง และแพตเทิร์นการเคลื่อนที่ของโหนด ในส่วนนี้อาจจะมีผลเฉพาะการจำลองโหนดที่มีการสื่อสารแบบไร้สายเท่านั้น
6. Application เป็นส่วนที่จะกำหนดว่าโหนดจะส่งข้อมูลประเภทใดออกไป เป็นแบบ TCP หรือ UDP มีอัตราการส่งข้อมูล หรือขนาดข้อมูลเท่าใด รวมไปถึงการกำหนดรูปแบบการส่งข้อมูลว่าจะมีลักษณะอย่างไร ผู้ใช้สามารถเรียกใช้งานแอปพลิเคชันที่ตัวโปรแกรม NS3 เตรียมไว้ให้ และติดตั้งลงไปในโหนดเพื่อกำหนดให้โหนดมีพฤติกรรมตามที่ผู้ใช้กำหนด

เพื่อความเข้าใจที่มากยิ่งขึ้นผู้เขียนจะอธิบายการกำหนดค่าเหล่านี้ผ่านการยกตัวอย่างผ่านสคริปต์ first.cc

<ผู้เขียนแนะนำให้เปิดไฟล์ first.cc ประกอบการอ่านเพื่อจะได้เห็นภาพรวมของสคริปต์
ทั้งหมด>

โดยในขั้นแรก สคริปต์จะมีการประกาศคลาส NodeContainer ผ่านการประกาศตัวแปร “nodes” ก่อนที่จะเรียกใช้คำสั่ง Creates เพื่อสร้างโหนดขึ้นมาในระบบ โดยมีการรับตัวแปรเพื่อใช้ในการกำหนดจำนวนโหนดที่ถูกสร้างขึ้น (ซึ่งในที่นี้คือ 2 โหนด)

จากนั้นสคริปต์จะมีการประกาศคลาส PointToPointHelper เพื่อสร้างการเชื่อมต่อแบบมีสาย โดยผู้ใช้สามารถกำหนดอัตราเร็วในการส่งข้อมูล และความหน่วงในช่องทางการสื่อสารได้เพื่อความสมจริง

ถัดมาตัวสคริปต์จึงประกาศคลาส NetDevicesContainer เพื่อสร้าง Network Interface card (NIC) ขึ้นมาโดยการอ้างอิงข้อมูลการตั้งค่าการเชื่อมต่อจากคลาส PointToPointHelper และติดตั้ง NIC ลงไปบนโหนดทั้ง 2 ตัว หลังจากจบขั้นตอนนี้แล้วผู้ใช้งานก็จะได้โหนด 2 ตัวที่มีการเชื่อมต่อกันผ่านสายตามที่ผู้ใช้งานกำหนด

จากนั้นสคริปต์จึงมีการประกาศคลาส InternetStackHelper และติดตั้งลงบนโหนดผ่านคลาส IPv4InterfaceContainer ซึ่งจะเป็นการทำให้โหนดรู้จักโปรโตคอลพื้นฐานที่จำเป็นต่อการสื่อสารผ่านระบบอินเทอร์เน็ตเช่น ICMP protocol, ARP protocol เป็นต้น การติดตั้งคลาสนี้ลงบนโหนดจะช่วยทำให้ผู้ใช้สามารถสร้าง Log files เพื่อใช้ในการตรวจสอบผลลัพธ์ของการรับส่งข้อมูลในการจำลองได้ ในส่วนของคลาส IPv4AddressHelper จะเป็นตัวกำหนดค่าไอพีแอดเดรสให้กับโหนด ซึ่งในสคริปต์นี้จะเป็นการตั้งค่าไอพีแบบอัตโนมัติด้วยการเช็ต Base IP เอาไว้ โดยโหนดจะมีการตั้งค่าไอพีแอดเดรสแบบอัตโนมัติที่มีค่าเพิ่มขึ้นไปเรื่อย ๆ ตามจำนวนโหนดสูงสุดที่กำหนด (ระบุด้วย Subnet Mask)

จุดสังเกต: ผู้ใช้จะสามารถสังเกตได้ว่าตัวสคริปต์มักจะมีการตั้งค่าผ่านคลาส <Helper> ก่อนที่จะติดตั้งค่าเหล่านี้ลงบนโหนดผ่านคลาส <Container>

หลังจากตั้งค่าการเชื่อมต่อเรียบร้อยแล้ว ก็จะเป็นการตั้งค่าแอปพลิเคชันที่ใช้ในการติดต่อสื่อสารระหว่างโหนด โดยสคริปต์มีการประกาศคลาส UdpEchoServerHelper เพื่อกำหนดค่าในส่วนของโหนดที่ทำหน้าที่เป็นปลายทางในการรับข้อมูลโดยมีการกำหนดค่าตัวแปรเป็นค่าของพอร์ตที่จะใช้ในการรับข้อมูล จากนั้นจึงทำการติดตั้งลงบนโหนดที่ 2 (โหนดไอดีคือ 1) ผ่านการประกาศ ApplicationContainer โดยผู้ใช้จะสามารถตั้งค่าได้ว่าจะให้โหนดเริ่มรับข้อมูลที่วินาทีที่เท่าไร แล้วหยุดรับข้อมูลที่วินาทีที่เท่าไร ผ่านคำสั่ง Start และ Stop ตามลำดับ

ถัดมาจะเป็นการตั้งค่าในฝั่งของโหนดที่ทำหน้าที่เป็นผู้ส่งข้อความ โดยในฝั่งผู้ส่งจะสามารถกำหนดขนาดของแพ็กเก็ต จำนวนแพ็กเก็ต และความถี่ในการส่งข้อมูลได้ผ่านการประกาศคลาส echoClient และทำการตั้งค่า โดยคลาสนี้จะมีการรับค่าตัวแปรค่าแรกเป็นการ

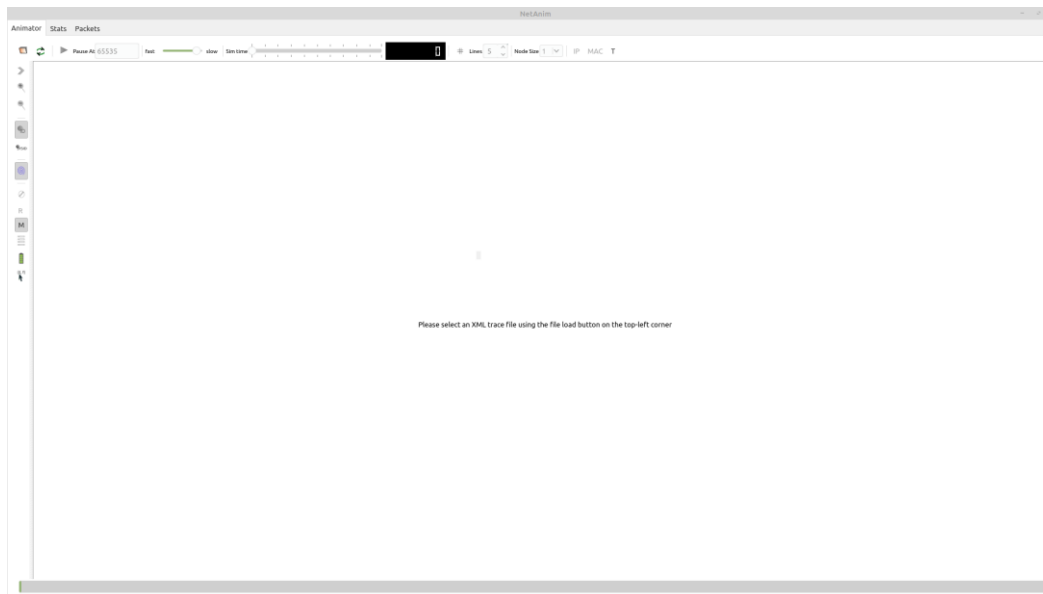
กำหนดไอพีแอดเดรสของโหนดปลายทาง และพอร์ตที่จะใช้ในการส่งข้อมูล หลังจากที่ได้ติดตั้งคุณลักษณะเหล่านี้ลงไปที่โหนดที่ 1 (โหนดไอดีคือ 0) แล้ว จากนั้นจะเป็นการสั่งให้เริ่มการจำลองระบบเครือข่าย และทำลายตัวจำลองเครือข่ายทิ้งผ่านคำสั่ง `Simulator::Run();` และ `Simulator::Destroy();` ตามลำดับ

ผู้ใช้สามารถศึกษาการทำงานของโปรแกรม NS3 เพิ่มเติมได้จากสคริปต์ในโฟลเดอร์ `example/tutorial` อื่น ๆ ในกรณีที่ผู้ใช้สนใจจะทำการศึกษาการทำงานของระบบเครือข่ายที่เกี่ยวข้องกับโหนดไร้สาย ผู้เขียนแนะนำให้ผู้ใช้เริ่มต้นศึกษาจากสคริปต์ `wifi-simple-adhoc.cc` ซึ่งถูกเก็บอยู่ในโฟลเดอร์ `example/wireless` ซึ่งในสคริปต์ดังกล่าวผู้ใช้จะได้เรียนรู้เกี่ยวกับการตั้งค่าโหนดเครือข่ายไร้สาย รวมไปถึงการกำหนดค่าพิกัดตำแหน่งของโหนด รวมไปถึงการตั้งค่ารูปแบบการเคลื่อนที่ของโหนดอีกด้วย โดยผู้เขียนจะกล่าวถึงรายละเอียดเหล่านี้ที่จำเป็นต่อการจำลองระบบเครือข่ายเฉพาะกิจเคลื่อนที่อีกครั้งในบทถัดไป

3.4 การเรียกใช้งาน Network Animator (NetAnim)

เมื่อผู้ใช้ทำการติดตั้งโปรแกรม NS3 ตามขั้นตอนที่กล่าวไว้ข้างต้นแล้ว ตัวชุดคำสั่ง `bake` จะมีการดาวน์โหลดโฟลเดอร์ `NetAnim-3.108` เข้ามาติดตั้งให้โดยอัตโนมัติ เพื่อใช้งานโปรแกรม `NetAnim` ผู้ใช้ต้องทำการคอลไฟล์ชุดคำสั่งเสียก่อน โดยให้ผู้ใช้เปลี่ยน `directory` ไปยังโฟลเดอร์ `netanim` จากนั้นจึงทำการเรียกใช้ชุดคำสั่งดังต่อไปนี้

```
>make clean
>qmake NetAnim.pro
>make
```



ภาพที่ 3.3 อินเทอร์เฟซโปรแกรม NetAnim

หลังจากรันชุดคำสั่งดังกล่าวแล้วผู้ใช้จะสามารถเรียกใช้โปรแกรม NetAnim ได้ผ่านการเรียกใช้คำสั่ง `./NetAnim` ซึ่งจะแสดงผลอินเทอร์เฟซออกมาตามภาพที่ 3.3

```

15 /
16
17 #include "ns3/core-module.h"
18 #include "ns3/network-module.h"
19 #include "ns3/internet-module.h"
20 #include "ns3/point-to-point-module.h"
21 #include "ns3/applications-module.h"
22 #include "ns3/netanim-module.h"
23
24 // Default Network Topology
25 //
26 //      10.1.1.0
27 // n0 ----- n1
28 //      point-to-point
29 //

```

ภาพที่ 3.4 การเรียกใช้ Library: netanim-module.h

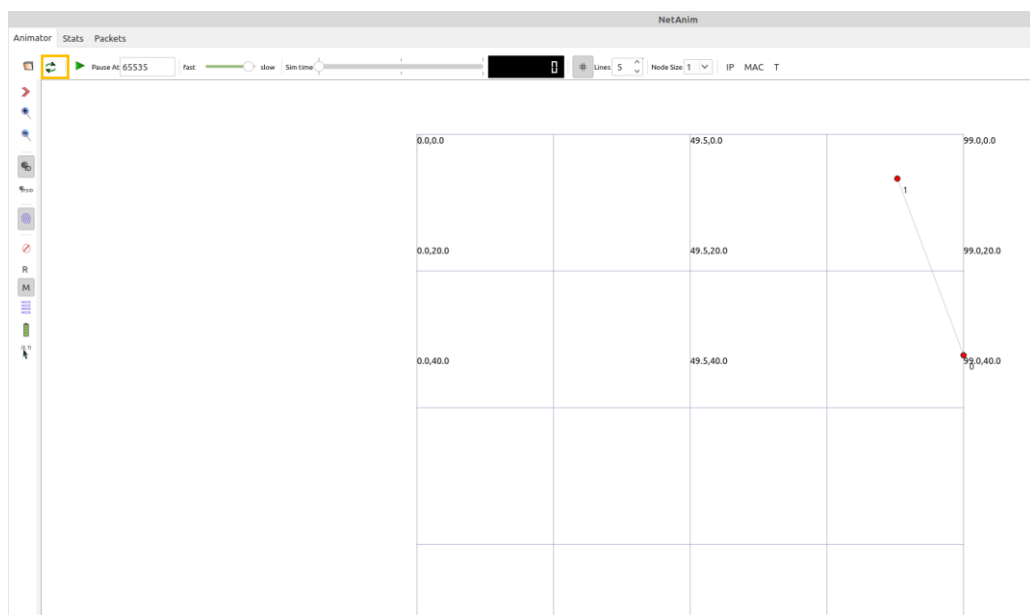
การที่จะแสดงผลการจำลองระบบเครือข่ายออกมาในรูปแบบของกราฟิกผ่านโปรแกรม NetAnim นั้นผู้ใช้จะต้องทำการสร้างไฟล์นามสกุล `.xml` ขึ้นมาระหว่างการรันสคริปต์เพื่อการจำลองเหตุการณ์โดยผู้ใช้จะต้องเพิ่มโค้ดในส่วนของการอ้างอิงเพื่อใช้งาน Library: `netanim-module.h` ตามคำสั่งในภาพที่ 3.4 จากนั้นผู้ใช้จะต้องแทรกโค้ดในการประกาศสร้างคลาส `AnimationInterface` เพื่อใช้ในการสร้างไฟล์ `.xml` ตามภาพที่ 3.5 โดยผู้เขียนได้ทำการเพิ่มชุด

โค้ดเหล่านี้ลงไปในสคริปต์ first.cc เป็นตัวอย่าง จากนั้นผู้เขียนจะทำการเรียกคำสั่งในการรันสคริปต์ first.cc อีกครั้งเพื่อการสร้างไฟล์ .xml ขึ้นมา

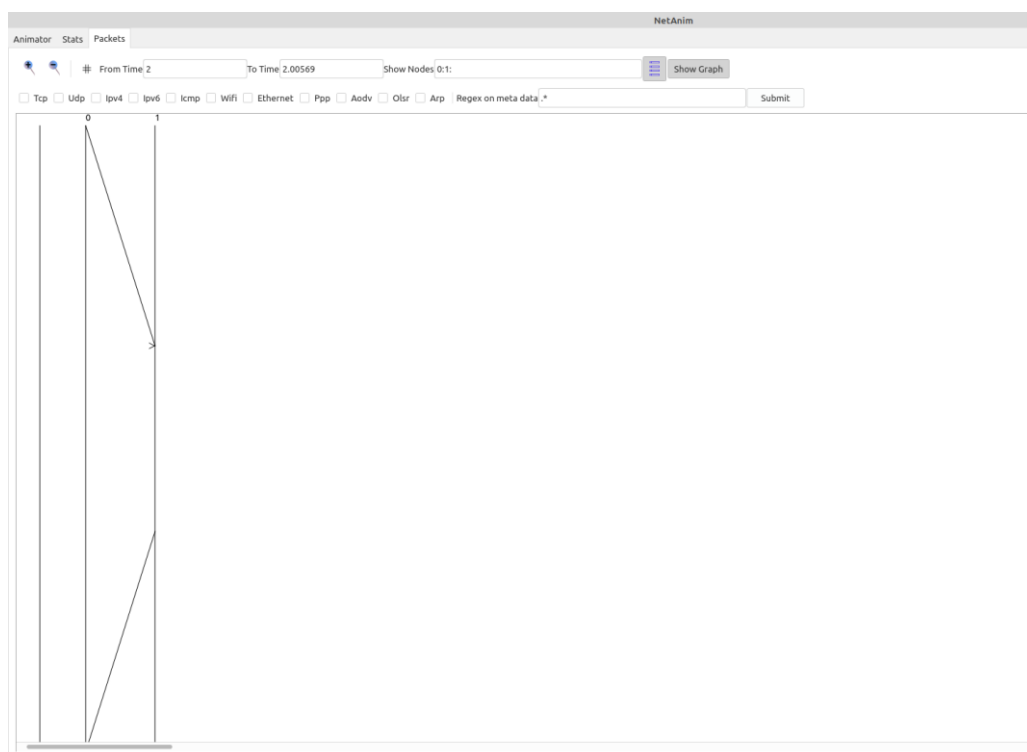
```
60
61 Ipv4InterfaceContainer interfaces = address.Assign (devices);
62
63 UdpEchoServerHelper echoServer (9);
64
65 ApplicationContainer serverApps = echoServer.Install (nodes.Get (1));
66 serverApps.Start (Seconds (1.0));
67 serverApps.Stop (Seconds (10.0));
68
69 UdpEchoClientHelper echoClient (interfaces.GetAddress (1), 9);
70 echoClient.SetAttribute ("MaxPackets", UintegerValue (1));
71 echoClient.SetAttribute ("Interval", TimeValue (Seconds (1.0)));
72 echoClient.SetAttribute ("PacketSize", UintegerValue (1024));
73
74 ApplicationContainer clientApps = echoClient.Install (nodes.Get (0));
75 clientApps.Start (Seconds (2.0));
76 clientApps.Stop (Seconds (10.0));
77
78
79 AnimationInterface anim ("first.xml");
80 Simulator::Run ();
81 Simulator::Destroy ();
82 return 0;
83 }
84
```

ภาพที่ 3.5 ชุดคำสั่งที่ใช้ในการสร้างไฟล์ .xml

ข้อควรระวัง: ผู้ใช้จะต้องทำการสร้างคลาส AnimationInterface ก่อนการเรียกคำสั่ง Simulation::run() เพื่อให้ไฟล์ .xml ถูกสร้างอย่างถูกต้อง



ภาพที่ 3.6 ผลการเรียกใช้ไฟล์ .xml ผ่านโปรแกรม NetAnim



ภาพที่ 3.7 แท็บ Packet บนโปรแกรม NetAnim เพื่อแสดงผลสรุปการส่งข้อมูลระหว่างโหนด

หลังจากการรันสคริปต์เพื่อการจำลองเครือข่ายเสร็จเรียบร้อยแล้ว ผู้ใช้จะได้ไฟล์ first.xml ซึ่งจะถูเก็บไว้ใน Directory /ns3.33 โดยผู้ใช้อาจจะสังเกตว่ามีข้อความเตือนปรากฏขึ้นบนหน้าต่าง CLI เนื่องจากผู้ใช้ไม่ได้กำหนดตำแหน่งให้โหนดทั้ง 2 ตัวเอาไว้ล่วงหน้า ซึ่งสามารถกำหนดได้ภายหลังหากผู้ใช้ต้องการ ในกรณีที่ผู้ใช้ไม่ได้ระบุตำแหน่งที่แน่นอนให้โหนด โหนดก็จะปรากฏบนพื้นที่การจำลองแบบสุ่ม

จากนั้นให้ผู้ใช้เปิดไฟล์ xml ที่ถูกสร้างขึ้นผ่านโปรแกรม NetAnim ด้วยการกดปุ่มไอคอน โพลเดอร์สีเหลืองที่อยู่มุมซ้ายบน ผู้ใช้ก็จะได้ผลลัพธ์ตามภาพที่ 3.6 โดยผู้ใช้จะสามารถกดปุ่ม “Play” ซึ่งมีลักษณะเป็นลูกศรสีเขียวเพื่อให้โปรแกรมแสดงผลการจำลองระบบเครือข่าย นอกจากนี้ผู้ใช้อังสามารถเลือกไปยังแท็บ Packet เพื่อดูภาพรวมของการส่งข้อมูลได้ตามภาพที่ 3.7 ซึ่งผู้ใช้อังสามารถกรองชนิดของข้อมูลที่ต้องการแสดงได้อีกด้วย

โปรแกรม NetAnim นั้นจะมีประโยชน์อย่างมากในการจำลองระบบเครือข่ายที่มีความซับซ้อนมากยิ่งขึ้น ซึ่งผู้ใช้สามารถกำหนดรายละเอียดในการแสดงผลเพิ่มเติมได้โดยการเขียนโค้ดคำสั่งเพื่อตั้งค่า Object ที่ถูกสร้างขึ้นจากคลาส AnimationInterface โดยผู้ใช้สามารถเรียนรู้การเขียนชุดคำสั่งได้จากการศึกษาสคริปต์ตัวอย่างการใช้งานที่ถูเก็บเอาไว้ใน Directory src/netanim/example

3.5 การวัดประสิทธิภาพผ่านไฟล์ประเภท Ascii Tracing

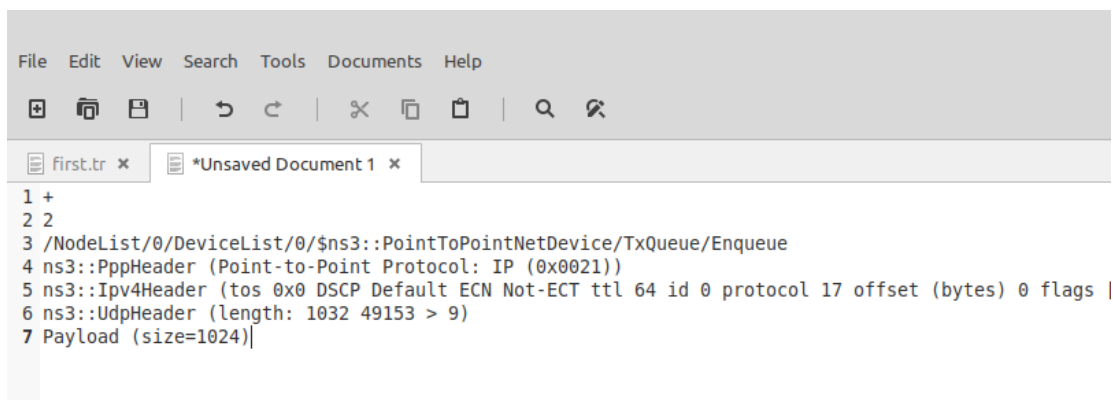
อีกส่วนหนึ่งที่มีความสำคัญต่อการใช้งานโปรแกรม NS3 นั้นคือการสกัดค่าผลลัพธ์การจำลองเพื่อนำมาแสดงผลเป็นตัวชี้วัดต่าง ๆ ที่บ่งบอกถึงประสิทธิภาพของระบบเครือข่ายที่ถูกจำลองขึ้นมา โดยในส่วนนี้ผู้เขียนจะแสดงวิธีการเก็บบันทึกไฟล์ Ascii trace ซึ่งมีนามสกุลไฟล์คือ .tr เพื่อนำเอามาใช้ในการประมวลผลในลำดับถัดไป โดยจะยกตัวอย่างจากการเพิ่มคำสั่งลงไปในสคริปต์ first.cc

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

```
AsciiTraceHelper ascii;
Ptr<OutputStreamWrapper> stream = ascii.CreateFileStream ("first.tr");
pointToPoint.EnableAsciiAll(stream);
```

```
AnimationInterface anim ("first.xml");
Simulator::Run ();
Simulator::Destroy ();
return 0;
}
```

ภาพที่ 3.8 ชุดคำสั่งที่ใช้ในการสร้าง Ascii trace file



ภาพที่ 3.9 ตัวอย่างเพื่อใช้ในการอธิบาย Ascii trace file format

โดยในการจะแทรกคำสั่งเพื่อให้เกิดการบันทึก trace file นั้นผู้ใช้จะต้องเพิ่มชุดคำสั่งในการตั้งค่า Trace file ตามภาพที่ 3.8 โดยชุดคำสั่งนี้จะเป็นการบันทึกทุกเหตุการณ์การรับส่งข้อมูลที่เกิดขึ้นทั้งหมดบน NIC ที่เราได้ทำการติดตั้งเอาไว้บนโหนดทั้ง 2 โหนด หลังจากที่ใช้สั่งรันสคริปต์อีกครั้งผู้ใช้จะทำการสร้างไฟล์ “first.tr” ขึ้นมา เมื่อผู้ใช้เปิดเข้าไปในไฟล์ ผู้ใช้จะสามารถสังเกตเห็นชุดข้อความซึ่งถูกใช้เพื่อระบุเหตุการณ์ต่าง ๆ ในเครือข่ายที่ถูกบันทึกเอาไว้ โดยข้อมูล

แต่ละส่วนจะถูกขึ้นไว้ด้วยการเว้นวรรค หากเรานำข้อความจากบรรทัดแรกมาแยกออกมาเป็นบรรทัด ๆ ก็จะได้ผลเป็นดังภาพที่ 3.9 ซึ่งเราสามารถแยกข้อความออกมาได้เป็นชุดข้อมูลทั้งหมด 7 ชุด โดยมีรายละเอียดดังต่อไปนี้

ข้อมูลที่แสดงในบรรทัดที่ 1 จะเป็นสัญลักษณ์ที่บ่งบอกถึงเหตุการณ์ที่เกิดขึ้นโดยจะแบ่งออกเป็นทั้งหมด 5 รูปแบบได้แก่

1. + (enqueue) หมายถึง เกิดเหตุการณ์ enqueue ที่อุปกรณ์
2. - (dequeue) หมายถึง เกิดเหตุการณ์ dequeue ที่อุปกรณ์
3. d (drop) หมายถึง เกิดการละทิ้งข้อความออกไปจากอุปกรณ์
4. t (transmit) หมายถึง เกิดการส่งข้อความออกไปจากอุปกรณ์
5. r (receive) หมายถึง เกิดการรับข้อความเข้ามาในอุปกรณ์

ข้อมูลที่แสดงในบรรทัดที่ 2 แสดงถึงช่วงเวลาที่เกิดเหตุการณ์ในบรรทัดแรก (หน่วยเป็นวินาที)

ข้อมูลที่แสดงในบรรทัดที่ 3 จะแสดงให้เห็นว่าเหตุการณ์ที่เกิดขึ้นนั้นเกิดที่โหนดใด และอุปกรณ์ใด โดยในเหตุการณ์นี้จะเกิดขึ้นที่โหนดไอดี 0 บนอุปกรณ์ไอดี 0 ซึ่งหมายถึง NIC เพราะมีอุปกรณ์ที่ติดตั้งบนโหนดนี้เพียงชิ้นเดียว (Point to Point Netdevice) โดยเหตุการณ์ที่เกิดขึ้นนั้นคือโหนดนั้นทำการเก็บข้อมูลเข้าไปใน Queue (/Enqueue) เพื่อรอการส่งออกไป (/Txqueue)

ข้อมูลที่แสดงในบรรทัดที่ 4 จะแสดงรายละเอียดของโปรโตคอลที่ถูกบรรจุอยู่ใน NIC ซึ่งในที่นี้คือ Point to Point

ข้อมูลที่แสดงในบรรทัดที่ 5 แสดงข้อมูลบน IPV4 Header

ข้อมูลที่แสดงในบรรทัดที่ 6 แสดงข้อมูลบน UDP Header

ข้อมูลที่แสดงในบรรทัดที่ 7 แสดงขนาดของข้อมูลที่ถูกบรรจุมาใน Payload

โดยสรุปก็คือในบรรทัดแรกของ Trace file นั้นคือเหตุการณ์ที่โหนดไอดี 0 นำแพ็กเก็ตที่ถูกสร้างใหม่ไปใส่ไว้ใน Queue เพื่อทำการรอส่งข้อมูล

โดยข้อมูลนั้นจะถูกส่งออกไปจากกระบวนการนำข้อมูลออกจาก Queue ในเหตุการณ์ที่เกิดขึ้นจาก Trace file บรรทัดที่ 2

Trace file บรรทัดที่ 3 แสดงให้เห็นว่าโหนดไอดี 1 ได้รับข้อมูลที่ถูกส่งมาจากโหนดไอดี 0 ที่เวลาวินาที่ 2.00369

โดย Trace file อีก 3 บรรทัดที่เหลือจะมีความคล้ายคลึงกับ 3 บรรทัดแรกแต่จะเป็นการส่งข้อมูลจากโหนดไอดี 1 กลับไปโหนดไอดี 0 แทน ตามแอปพลิเคชัน Echo ที่ถูกติดตั้งไว้

```
78
79
30
31   AsciiTraceHelper ascii;
32   Ptr<OutputStreamWrapper> stream = ascii.CreateFileStream ("first.tr");
33   stack.EnableAsciiIpv4All (stream);
34   //pointToPoint.EnableAsciiAll(stream);
35
36   AnimationInterface anim ("first.xml");
37   Simulator::Run ();
38   Simulator::Destroy ();
39   return 0;
40 }
```

ภาพที่ 3.10 การบันทึกเหตุการณ์ที่เกิดขึ้นจากการจำลองโพรโทคอล Ipv4 ลงบน trace file

นอกจากนั้นผู้ใช้อย่างยังสามารถบันทึกรายละเอียดของเหตุการณ์ต่าง ๆ ที่เกิดขึ้นบน OSI layer 3 ลงไปบน Trace file ได้โดยการเพิ่มคำสั่งตามภาพที่ 3.10 (stack เป็น object ที่ถูกสร้างจาก class (InternetStackHelper)) โดยผู้เขียนได้ทำการปิดการทำงานของโค้ดในส่วนการบันทึกข้อมูลจาก NIC ออกไปก่อนเพื่อให้การอ่านไฟล์มีความสะดวกมากยิ่งขึ้น ซึ่งผลลัพธ์ที่ได้ก็จะมี ความใกล้เคียงกับ Trace file เดิม เพียงแต่รายละเอียดของไฟล์จะถูกลดทอนลง โดยผู้ใช้งาน สามารถเลือกได้ว่าจะกำหนดให้สคริปต์บันทึกข้อมูลจากคลาสใดลงไปบน Trace file ได้บ้าง เพื่อที่จะทำให้ผู้ใช้ได้รับข้อมูลที่ต้องการในการนำไปใช้วัดประสิทธิภาพของเครือข่ายได้

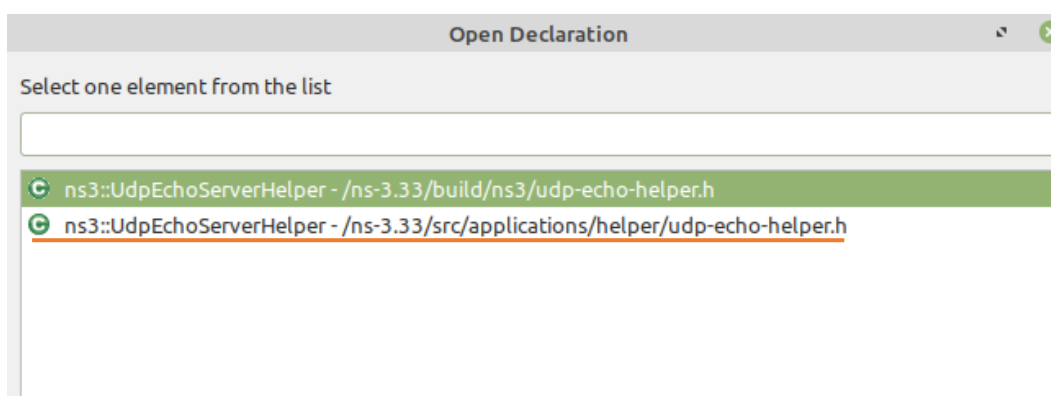
ตัวอย่างเช่นผู้ใช้งานจะสามารถคำนวณหาเวลาที่ใช้ในการส่งข้อมูลจากโหนด 0 ไปยังโหนด 1 ได้ หากมีการส่งข้อมูลมากกว่า 1 ครั้ง ผู้ใช้ก็สามารถนำ Trace file มาใช้วิเคราะห์เวลาที่ใช้ในการ ส่งข้อมูลโดยเฉลี่ยได้ แต่ในการจำลองเครือข่ายที่มีความซับซ้อนมากยิ่งขึ้น ขนาดของ Trace file นั้นจะมีขนาดใหญ่มากจนผู้ใช้ไม่สามารถวิเคราะห์ข้อมูลผลการจำลองได้ด้วยตนเอง ดังนั้น ผู้เขียนจึงแนะนำให้ผู้ใช้เขียนสคริปต์เพื่อเข้ามาช่วยประมวลผล Trace file เพื่อช่วยให้ผู้ใช้ สามารถสกัดผลลัพธ์ที่สนใจออกมาได้ หนึ่งในวิธีการที่ผู้เขียนแนะนำคือการใช้เครื่องมือ AWK Script ซึ่งมีความเรียบง่ายในการใช้งาน โดยผู้ใช้สามารถไปศึกษาเพิ่มเติมได้จาก “คู่มือการใช้ Network Simulator 2 (NS2) เบื้องต้นเพื่อใช้ในการจำลองเครือข่ายเฉพาะกิจเคลื่อนที่” ซึ่งถูก จัดเก็บไว้ที่ห้องปฏิบัติการ 534 คณะเทคโนโลยีสารสนเทศ สจล.

3.6 คำแนะนำเบื้องต้นสำหรับการแก้ไขชุดคำสั่งเพื่อการพัฒนาต่อยอดโปรโตคอล

ผู้ใช้อาจจะสามารถสังเกตได้ว่าผลลัพธ์ที่ผู้ใช้สามารถวิเคราะห์ได้จาก Trace File “first.tr” นั้นได้ถูกแสดงผลออกมาอยู่บนหน้าจอ CLI ของผู้ใช้อยู่แล้ว ข้อความเหล่านี้เกิดจากคำสั่ง LogComponentEnable ในไฟล์ first.cc ในบรรทัดก่อนหน้าที่จะมีการสร้างโหนดขึ้น โดยคำสั่งนี้จะอนุญาตให้เกิดการแสดงผล Log ที่ถูกสร้าง และเก็บไว้บนแอปพลิเคชัน UdpEchoClient / UdpEchoServer สิ่งที่น่าสนใจก็คือหากผู้ใช้ต้องการจะแก้ไขข้อความที่ถูกแสดงผลบนหน้าจอ CLI ที่เกิดจากแอปพลิเคชัน 2 ตัวนี้ต้องทำอย่างไร

การทำงานในส่วนนี้จะเกินขอบเขตของการแก้ไขสคริปต์ในการกำหนดค่าสภาพแวดล้อมในการจำลองเครือข่ายออกไป ไม่ใช่เพียงแต่การนำสิ่งที่โปรแกรม NS3 มีอยู่เดิมมาใช้ แต่ผู้ใช้งานจะต้องเข้าไปแก้ไขโมเดลที่ถูกนำมาใช้ในการจำลองการเชื่อมต่อ หรืออุปกรณ์โดยตรง ผู้เขียนจะยกตัวอย่างการเข้าไปแก้ไขแอปพลิเคชัน UdpEchoServer เป็นตัวอย่างให้เกิดการแสดงผลบนหน้าจอ CLI ที่เปลี่ยนไป โดยจะเริ่มจากการแนะนำการค้นหาไฟล์ที่กำหนดพฤติกรรมการแสดงผลข้อความบนหน้าจอ โดยเริ่มต้นการค้นหาผ่านสคริปต์ first.cc

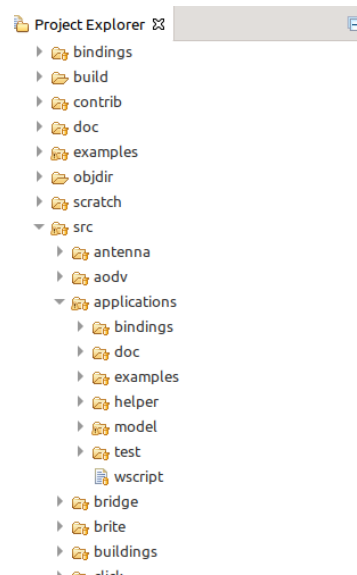
ในขั้นแรกผู้ใช้งานจะต้องทราบว่าโค้ดที่ทำหน้าที่แสดงผลข้อความขึ้นมาบนหน้าจอ CLI นั้นถูกเก็บอยู่ในไฟล์ใด ซึ่งในโปรแกรม Eclipse นั้นจะมีตัวช่วยในการค้นหาไฟล์ที่เกี่ยวข้องกับสคริปต์ที่ผู้ใช้งานกำลังใช้งาน โดยผู้ใช้งานจะสังเกตได้ว่าในสคริปต์ first.cc นั้นมีการประกาศเพื่อเรียกใช้งานคลาส UdpEchoServerHelper อยู่ ซึ่งโดยมากแล้วคลาส Helper เหล่านี้จะถูกใช้เพื่อสร้างการเชื่อมโยงระหว่างไฟล์หรือคลาส รวมไปถึงการทำให้ไฟล์สคริปต์สามารถเข้าถึงคลาสที่กำหนดและนำไปสร้างเป็น Object ได้



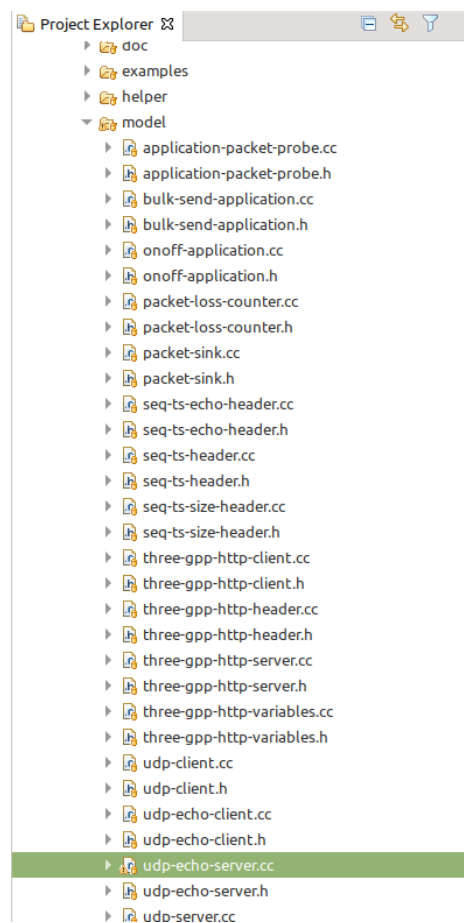
ภาพที่ 3.11 หน้าต่าง Open Declaration

หากผู้ใช้งานนำ Cursor ไปวางไว้ที่คำสั่งการเรียกคลาส และกดปุ่ม F3 ตัวโปรแกรม Eclipse จะแสดงให้เห็นถึง Directory Path ที่เกี่ยวข้องกับคลาสนั้นตามภาพที่ 3.11 โดยให้ผู้ใช้งานสังเกตไปที่โฟลเดอร์ src ตามที่ผู้เขียนได้ขีดเส้นใต้ไว้ในภาพ ซึ่งโฟลเดอร์ src นั้นจะเป็นส่วนที่เก็บโมเดลทุก

ชนิดที่โปรแกรม NS3 สามารถเรียกใช้งานได้ ซึ่งจะสังเกตได้ว่าคลาส UdpEchoServerHelper ถูกเก็บไว้ในโฟลเดอร์ applications



ภาพที่ 3.12 ซับโฟลเดอร์ทั้งหมดภายในโฟลเดอร์ applications



ภาพที่ 3.13 รายชื่อไฟล์ในซับโฟลเดอร์ model ภายในโฟลเดอร์ applications

```

162 Address localAddress;
163 while ((packet = socket->RecvFrom (from)))
164 {
165     socket->GetSockName (localAddress);
166     m_rxTrace (packet);
167     m_rxTraceWithAddresses (packet, from, localAddress);
168     if (InetSocketAddress::IsMatchingType (from))
169     {
170         NS_LOG_INFO ("At time " << Simulator::Now ().As (Time::S) << " server received " << packet->GetSize () <<
171             InetSocketAddress::ConvertFrom (from).GetIpv4 () << " port " <<
172             InetSocketAddress::ConvertFrom (from).GetPort ());
173     }
174     else if (Inet6SocketAddress::IsMatchingType (from))
175     {
176         NS_LOG_INFO ("At time " << Simulator::Now ().As (Time::S) << " server received " << packet->GetSize () <<
177             Inet6SocketAddress::ConvertFrom (from).GetIpv6 () << " port " <<
178             Inet6SocketAddress::ConvertFrom (from).GetPort ());
179     }
180     packet->RemoveAllPacketTags ();
181     packet->RemoveAllByteTags ();
182     NS_LOG_LOGIC ("Echoing packet");
183     socket->SendTo (packet, 0, from);
184     if (InetSocketAddress::IsMatchingType (from))
185     {
186         NS_LOG_INFO ("At time " << Simulator::Now ().As (Time::S) << " server sent " << packet->GetSize () << " by
187             InetSocketAddress::ConvertFrom (from).GetIpv4 () << " port " <<
188             InetSocketAddress::ConvertFrom (from).GetPort ());
189     }
190     else if (Inet6SocketAddress::IsMatchingType (from))
191     {
192         NS_LOG_INFO ("At time " << Simulator::Now ().As (Time::S) << " server sent " << packet->GetSize () << " by
193             Inet6SocketAddress::ConvertFrom (from).GetIpv6 () << " port " <<
194             Inet6SocketAddress::ConvertFrom (from).GetPort ());
195     }
196 }
197 }
198 }
199 }

```

ภาพที่ 3.14 ส่วนของคำสั่งที่ใช้ในการเก็บ Log ในไฟล์ udp-echo-server.cc

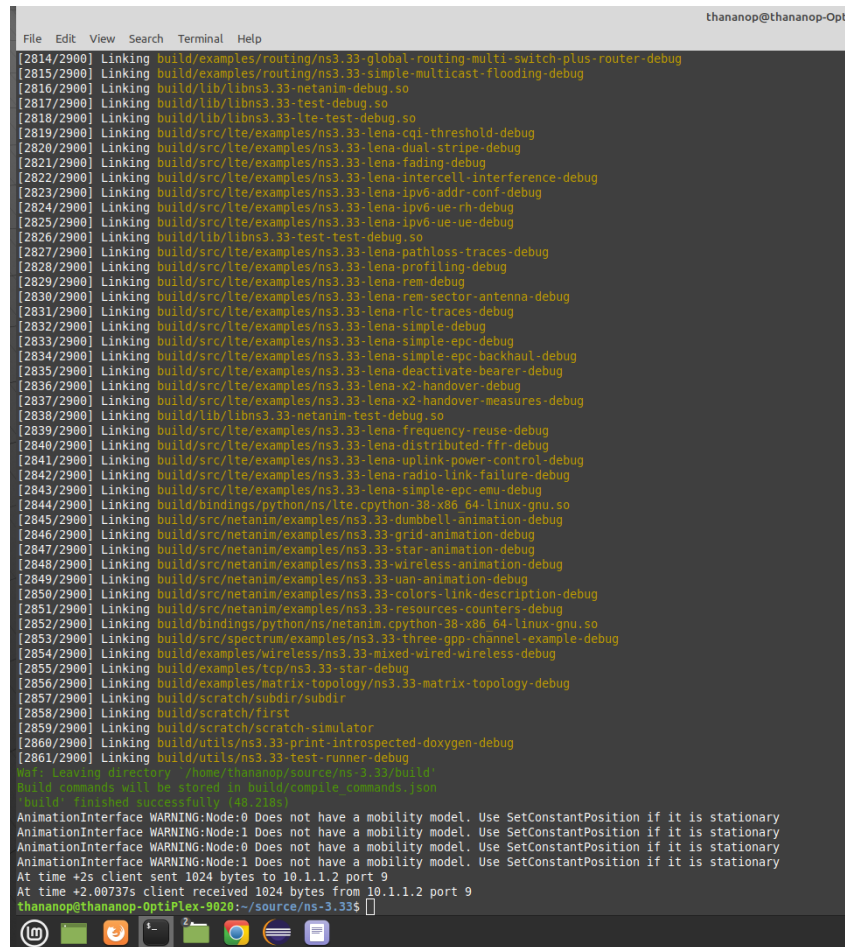
หากผู้ใช้ลองเปิดเข้าไปที่โฟลเดอร์ applications ตามภาพที่ 3.12 ก็จะพบว่า มี ซับ โฟลเดอร์แยกย่อยลงไปอีกหลายโฟลเดอร์ หนึ่งในนั้นคือโฟลเดอร์ helper ซึ่งเป็นที่อยู่ของคลาส UdpEchoServerHelper แต่โฟลเดอร์ที่เก็บไฟล์ที่กำหนดการทำงานจริง ๆ นั้นมักจะอยู่ที่ โฟลเดอร์ model หากผู้ใช้เปิดเข้าไปตามภาพที่ 3.13 ผู้ใช้ก็จะพบไฟล์ชื่อ udp-echo-server.cc ซึ่งเป็นไฟล์เป้าหมายที่เราต้องการ

เมื่อเปิดไฟล์ udp-echo-server.cc ขึ้นมาแล้วเลื่อนลงไปที่โค้ดบรรทัดที่ 170 เป็นต้นไปตามภาพที่ 3.14 ผู้ใช้จะพบส่วนของการเก็บ Log ที่เกิดขึ้นในไฟล์นี้ (ผู้ใช้งานสามารถค้นหาส่วนของโค้ดที่ต้องการได้ผ่านแพตเทิร์นที่ใช้ในการแสดงผลข้อความ) โดยผู้เขียนจะลองทำการปิดการใช้งานคำสั่งในการเก็บ Log (บรรทัดที่ 170-172 และ 189-191) ก่อนที่จะทำการรันสคริปต์ first.cc อีกครั้ง

ผู้ใช้งานจะสังเกตได้ว่าหลังจากรันสคริปต์ไปแล้ว โปรแกรม NS3 จะทำการคอมไพล์ไฟล์ใหม่ เนื่องจากได้มีการแก้ไขค่าในโมเดลของตัวโปรแกรม โดยผลลัพธ์ที่ปรากฏจากภาพที่ 3.15 นั้นจะแสดงให้เห็นว่า ชุดข้อความที่เกี่ยวข้องกับ UdpEchoServer นั้นไม่ปรากฏขึ้นมาบนหน้า CLI อีกต่อไป

ตัวอย่างการแก้ไขไฟล์แบบเริ่มต้นนี้อาจช่วยให้ผู้ใช้งานสามารถค้นหาไฟล์เป้าหมายที่ต้องการเข้าไปแก้ไขชุดโค้ด เพื่อการพัฒนาต่อยอดโปรแกรม NS3 ตามแต่วัตถุประสงค์ของผู้ใช้ได้

นอกเหนือจากการค้นหาไฟล์เป้าหมายเพื่อทำการแก้ไขแล้ว หากผู้ใช้ไม่สามารถคาดเดาเหตุการณ์ที่เกิดขึ้นจากการรันสคริปต์ หรือเจาะจงลงไปได้ว่าจะต้องไปจัดการกับไฟล์ใด หนึ่งในเครื่องมือที่สำคัญที่จะช่วยให้ผู้ใช้ทำความเข้าใจโค้ดบนโปรแกรม NS3 ได้มากยิ่งขึ้นนั่นคือการใช้งานเครื่องมือ Debugger ซึ่งผู้เขียนได้แสดงวิธีการตั้งค่าเอาไว้แล้วในบทก่อนหน้านี้ โดยผู้เขียนคาดหวังว่า ผู้ใช้งานโปรแกรม NS3 นั้นจะมีประสบการณ์การ Debug ชุดคำสั่งจากการเขียนโปรแกรมผ่านภาษาอื่น ๆ มาอยู่แล้ว ผู้เขียนจึงจะไม่ขอลงรายละเอียดในส่วนนี้เพิ่มเติม



```
File Edit View Search Terminal Help
[2814/2900] Linking build/examples/routing/ns3.33-global-routing-multi-switch-plus-router-debug
[2815/2900] Linking build/examples/routing/ns3.33-simple-multicast-flooding-debug
[2816/2900] Linking build/lib/libns3.33-netanim-debug.so
[2817/2900] Linking build/lib/libns3.33-test-debug.so
[2818/2900] Linking build/lib/libns3.33-lte-test-debug.so
[2819/2900] Linking build/src/lte/examples/ns3.33-lena-cqi-threshold-debug
[2820/2900] Linking build/src/lte/examples/ns3.33-lena-dual-stripe-debug
[2821/2900] Linking build/src/lte/examples/ns3.33-lena-fading-debug
[2822/2900] Linking build/src/lte/examples/ns3.33-lena-intercell-interference-debug
[2823/2900] Linking build/src/lte/examples/ns3.33-lena-ipv6-addr-conf-debug
[2824/2900] Linking build/src/lte/examples/ns3.33-lena-ipv6-ue-rh-debug
[2825/2900] Linking build/src/lte/examples/ns3.33-lena-ipv6-ue-ue-debug
[2826/2900] Linking build/lib/libns3.33-test-test-debug.so
[2827/2900] Linking build/src/lte/examples/ns3.33-lena-pathloss-traces-debug
[2828/2900] Linking build/src/lte/examples/ns3.33-lena-profiling-debug
[2829/2900] Linking build/src/lte/examples/ns3.33-lena-rem-debug
[2830/2900] Linking build/src/lte/examples/ns3.33-lena-rem-sector-antenna-debug
[2831/2900] Linking build/src/lte/examples/ns3.33-lena-rlc-traces-debug
[2832/2900] Linking build/src/lte/examples/ns3.33-lena-simple-debug
[2833/2900] Linking build/src/lte/examples/ns3.33-lena-simple-epc-debug
[2834/2900] Linking build/src/lte/examples/ns3.33-lena-simple-epc-backhaul-debug
[2835/2900] Linking build/src/lte/examples/ns3.33-lena-deactivate-bearer-debug
[2836/2900] Linking build/src/lte/examples/ns3.33-lena-x2-handover-debug
[2837/2900] Linking build/src/lte/examples/ns3.33-lena-x2-handover-measures-debug
[2838/2900] Linking build/lib/libns3.33-netanim-test-debug.so
[2839/2900] Linking build/src/lte/examples/ns3.33-lena-frequency-reuse-debug
[2840/2900] Linking build/src/lte/examples/ns3.33-lena-distributed-ffr-debug
[2841/2900] Linking build/src/lte/examples/ns3.33-lena-uplink-power-control-debug
[2842/2900] Linking build/src/lte/examples/ns3.33-lena-radio-link-failure-debug
[2843/2900] Linking build/src/lte/examples/ns3.33-lena-simple-epc-emu-debug
[2844/2900] Linking build/bindings/python/ns/lte.cpython-38-x86_64-linux-gnu.so
[2845/2900] Linking build/src/netanim/examples/ns3.33-dumbbell-animation-debug
[2846/2900] Linking build/src/netanim/examples/ns3.33-grid-animation-debug
[2847/2900] Linking build/src/netanim/examples/ns3.33-star-animation-debug
[2848/2900] Linking build/src/netanim/examples/ns3.33-wireless-animation-debug
[2849/2900] Linking build/src/netanim/examples/ns3.33-uav-animation-debug
[2850/2900] Linking build/src/netanim/examples/ns3.33-colors-link-description-debug
[2851/2900] Linking build/src/netanim/examples/ns3.33-resources-counters-debug
[2852/2900] Linking build/bindings/python/ns/netanim.cpython-38-x86_64-linux-gnu.so
[2853/2900] Linking build/src/spectrum/examples/ns3.33-three-gpp-channel-example-debug
[2854/2900] Linking build/examples/wireless/ns3.33-mixed-wired-wireless-debug
[2855/2900] Linking build/examples/tcp/ns3.33-star-debug
[2856/2900] Linking build/examples/matrix-topology/ns3.33-matrix-topology-debug
[2857/2900] Linking build/scratch/subdir/subdir
[2858/2900] Linking build/scratch/first
[2859/2900] Linking build/scratch/scratch-simulator
[2860/2900] Linking build/utlis/ns3.33-print-introspected-doxxygen-debug
[2861/2900] Linking build/utlis/ns3.33-test-runner-debug
waf: Leaving directory /home/thananop/source/ns-3.33/build
Build commands will be stored in build/compile_commands.json
'build' finished successfully (48.218s)
AnimationInterface WARNING:Node:0 Does not have a mobility model. Use SetConstantPosition if it is stationary
AnimationInterface WARNING:Node:1 Does not have a mobility model. Use SetConstantPosition if it is stationary
AnimationInterface WARNING:Node:0 Does not have a mobility model. Use SetConstantPosition if it is stationary
AnimationInterface WARNING:Node:1 Does not have a mobility model. Use SetConstantPosition if it is stationary
At time +2s client sent 1024 bytes to 10.1.1.2 port 9
At time +2.00737s client received 1024 bytes from 10.1.1.2 port 9
thananop@thananop-OptiPlex-9020:~/source/ns-3.33$
```

ภาพที่ 3.15 การส่งรันสคริปต์ first.cc หลังจากมีการแก้ไขไฟล์ในโฟลเดอร์โมเดล

หากผู้ใช้ต้องการค้นหาข้อมูลเพิ่มเติมว่า Module หรือ Class ที่ผู้ใช้สนใจนั้นมีหลักการทำงานอย่างไร (รวมไปถึงการตรวจสอบรายละเอียดของตัวแปรต่าง ๆ) ผู้ใช้สามารถเข้าไปค้นหาข้อมูลได้เพิ่มเติมจาก URL: <https://www.nsnam.org/doxygen/modules.html> ได้ โดยในบทถัดไปผู้เขียนจะอธิบายสคริปต์ที่เกี่ยวข้องกับการใช้งาน และการตั้งค่าอุปกรณ์เครือข่ายไร้สาย และโมดูลที่จำเป็นต่อการจำลองเครือข่ายเฉพาะกิจเคลื่อนที่ รวมไปถึงชุดคำสั่งที่ใช้สร้าง SDN Controller เพื่อใช้ในการควบคุมการค้นหาเส้นทางบนเครือข่ายเฉพาะกิจเคลื่อนที่ที่ผู้เขียนได้พัฒนาขึ้นมา

บทที่ 4

ชุดคำสั่งเพื่อการจำลองระบบเครือข่ายเฉพาะกิจที่ถูกควบคุมได้ด้วยเอสดีเอ็น คอนโทรลเลอร์แบบอินแบนด์

หลังจากที่ผู้ใช้ได้เรียนรู้การใช้งานโปรแกรม NS3 เบื้องต้นแล้ว ในบทนี้ผู้เขียนจะแนะนำวิธีการใช้โปรแกรม NS3 ในส่วนที่เกี่ยวข้องกับการจำลองระบบเครือข่ายเฉพาะกิจ ซึ่งผู้ใช้อาจต้องมีความเข้าใจต่อการสร้าง และตั้งค่าโหนดไร้สาย รวมไปถึงการกำหนดโปรโตคอลที่จะใช้ในการสื่อสารระหว่างโหนดโดยจะมีรายละเอียดดังนี้

4.1 การตั้งค่าเพื่อการใช้งานอุปกรณ์เครือข่ายไร้สายเบื้องต้น และการติดตั้งโปรโตคอลการสื่อสารเพื่อสร้างเครือข่ายไร้สายเฉพาะกิจ (Mobile Ad Hoc Networks)

```
198 int
199 main (int argc, char *argv[])
200 {
201     RoutingExperiment experiment;
202     std::string CSVfileName = experiment.CommandSetup (argc,argv);
203
204     //blank out the last output file and write the column headers
205     std::ofstream out (CSVfileName.c_str ());
206     out << "SimulationSecond," <<
207     "ReceiveRate," <<
208     "PacketsReceived," <<
209     "NumberOfSinks," <<
210     "RoutingProtocol," <<
211     "TransmissionPower" <<
212     std::endl;
213     out.close ();
214
215     int nSinks = 10;
216     double txp = 7.5;
217
218     experiment.Run (nSinks, txp, CSVfileName);
219 }
```

ภาพที่ 4.1 ฟังก์ชัน main ของสคริปต์ manet-routing-compare.cc

ในส่วนนี้ผู้เขียนจะขอยกตัวอย่างการสร้าง และการตั้งค่าโหนดไร้สายผ่านสคริปต์ตัวอย่าง manet-routing-compare.cc ซึ่งถูกเก็บเอาไว้ในโฟลเดอร์ examples/routing โดยภาพที่ 4.1 นั้นจะแสดงฟังก์ชัน main ของสคริปต์ โดยคลาส RoutingExperiment นั้นจะเป็นคลาสหลักที่ใช้ในการกำหนดค่าตัวแปร และ Method ต่าง ๆ ที่ใช้ในการจำลองระบบเครือข่าย โดยคำสั่ง CommandSetup นั้นจะถูกใช้ในการรับค่าตัวแปรจากผู้ใช้ แต่หากผู้ใช้ไม่ได้กำหนดค่าใด

ๆ ตอนสร้าง Object ขึ้นมา ตัวสคริปต์จะมีการกำหนดค่าเริ่มต้นให้แก่ตัวแปรเหล่านี้เอาไว้ล่วงหน้า จากนั้นในสคริปต์จะมีการกำหนดตัวแปรดังนี้

1. จำนวนโหนดที่จะทำหน้าที่ในการรับข้อมูล (โหนดปลายทาง) ด้วยการตั้งค่าตัวแปร nSinks
2. ค่าพลังงานที่โหนดจะใช้เพื่อการแพร่กระจายข้อความผ่านการตั้งค่าตัวแปร txp

จากนั้นจะมีการเรียกคำสั่ง Run เพื่อเริ่มการกำหนดค่าตัวแปรอื่น ๆ ก่อนเริ่มการจำลองเครือข่าย

```
Config::setDefault ("ns3::OnOffApplication::PacketSize",StringValue ("64"));
Config::setDefault ("ns3::OnOffApplication::DataRate", StringValue (rate));
```

ภาพที่ 4.2 คำสั่ง Config::setDefault

```
245 NodeContainer adhocNodes;
246 adhocNodes.Create (nWifis);
247
248 // setting up wifi phy and channel using helpers
249 WifiHelper wifi;
250 wifi.SetStandard (WIFI_STANDARD_80211b);
251
252 YansWifiPhyHelper wifiPhy;
253 YansWifiChannelHelper wifiChannel;
254 wifiChannel.SetPropagationDelay ("ns3::ConstantSpeedPropagationDelayModel");
255 wifiChannel.AddPropagationLoss ("ns3::FriisPropagationLossModel");
256 wifiPhy.SetChannel (wifiChannel.Create ());
257
258 // Add a mac and disable rate control
259 WifiMacHelper wifiMac;
260 wifi.SetRemoteStationManager ("ns3::ConstantRateWifiManager",
261                               "DataMode",StringValue (phyMode),
262                               "ControlMode",StringValue (phyMode));
263
264 wifiPhy.Set ("TxPowerStart",DoubleValue (txp));
265 wifiPhy.Set ("TxPowerEnd", DoubleValue (txp));
266
267 wifiMac.SetType ("ns3::AdhocWifiMac");
268 NetDeviceContainer adhocDevices = wifi.Install (wifiPhy, wifiMac, adhocNodes);
269
270 MobilityHelper mobilityAdhoc;
271 int64_t streamIndex = 0; // used to get consistent mobility across scenarios
272
273 ObjectFactory pos;
274 pos.SetTypeId ("ns3::RandomRectanglePositionAllocator");
275 pos.Set ("X", StringValue ("ns3::UniformRandomVariable[Min=0.0|Max=300.0]"));
276 pos.Set ("Y", StringValue ("ns3::UniformRandomVariable[Min=0.0|Max=1500.0]"));
```

ภาพที่ 4.3 การสร้างโหนดเครือข่ายไร้สาย

ภาพที่ 4.2 แสดงการเรียกใช้งานคำสั่ง Config::SetDefault โดยคำสั่งนี้จะทำการ Override ค่าตัวแปรเริ่มต้นบนคลาสที่กำหนดให้เป็นไปตามการตั้งค่าของผู้ใช้ จากนั้นจะเป็นการสร้าง โหนดเครือข่ายไร้สาย และการตั้งค่าต่าง ๆ ตามภาพที่ 4.3 โดยเริ่มต้นจากการสร้างโหนดเปล่า ๆ ขึ้นมาก่อนจากการเรียกใช้คลาส NodeContainer จากนั้นก็จะเริ่มการตั้งค่าอุปกรณ์เครือข่าย ก่อนที่จะติดตั้งลงไปในโหนดโดยมีรายละเอียดดังนี้

1. Class WifiHelper: ใช้ในการกำหนดรายละเอียดการเชื่อมต่อแบบไร้สาย โดยมีการ กำหนดมาตรฐานในการส่งข้อมูลด้วย method: SetStandard ในโค้ดบรรทัดที่ 250 และกำหนดอัตราการส่งข้อมูลด้วย method: SetRemoteStationManager ในโค้ด บรรทัดที่ 260-262
2. Class YansWifiChannelHelper: ใช้ในการกำหนดรายละเอียดของช่องทางการสื่อสาร แบบไร้สายว่าจะใช้โมเดลในการจำลองความหน่วงโมเดลแบบใดผ่าน method: SetPropagationDelay และจะใช้แบบจำลอง LossModel รูปแบบใดผ่าน method: AddPropagationLoss ในโค้ดบรรทัดที่ 254-255 โดยโปรแกรม NS3 จะมีโมเดลเพื่อ ใช้ในการจำลองรายละเอียดของช่องทางการสื่อสารไร้สายอยู่หลายรูปแบบซึ่งค่าของ method เหล่านี้จะสามารถเปลี่ยนแปลงได้ขึ้นอยู่กับผู้ใช้
3. Class YansWifiPhyHelper จะมีการใช้งานร่วมกับคลาสก่อนหน้าเพื่อใช้ในการกำหนด รายละเอียดของฮาร์ดแวร์ที่ใช้ในการสื่อสารแบบไร้สาย โดยการนำเอาข้อมูลช่องทาง สื่อสารที่กำหนดไว้ล่วงหน้ามาใช้ผ่าน method: SetChannel ในโค้ดบรรทัดที่ 256 รวมไปถึงการกำหนดพลังงานที่ใช้ในการส่งข้อมูลผ่านอุปกรณ์ผ่านโค้ดในบรรทัดที่ 264-265 ซึ่งพลังงานที่ใช้ในการส่งข้อมูลนั้นจะมีผลต่อระยะการสื่อสารของโหนดซึ่งจะถูก นำไปคำนวณผ่านสมการจำลอง LossModel ที่ถูกกำหนดเอาไว้ข้างต้น
4. Class WifiMacHelper เป็นการจำลองรายละเอียดของ Mac layer บนอุปกรณ์ เครือข่ายว่าจะให้โหนดทำงานอยู่ในโหมดใด โดยในบรรทัดที่ 267 จะมีการกำหนด Mac layer ให้มีรูปแบบในการสื่อสารแบบ Ad Hoc เพื่อใช้ในการจำลองเครือข่ายเฉพาะกิจ

จากนั้นจะเป็นการติดตั้งอุปกรณ์เครือข่ายไร้สายที่ถูกกำหนดค่าเอาไว้ลงไปในโหนดทุกโหนด ผ่านการประกาศคลาส NetDeviceContainer ในบรรทัดที่ 268


```

273 ObjectFactory pos;
274 pos.SetTypeId ("ns3::RandomRectanglePositionAllocator");
275 pos.Set ("X", StringValue ("ns3::UniformRandomVariable[Min=0.0|Max=300.0]"));
276 pos.Set ("Y", StringValue ("ns3::UniformRandomVariable[Min=0.0|Max=1500.0]"));
277
278 Ptr<PositionAllocator> taPositionAlloc = pos.Create ()->GetObject<PositionAllocator> ();
279 streamIndex += taPositionAlloc->AssignStreams (streamIndex);
280
281 std::stringstream ssSpeed;
282 ssSpeed << "ns3::UniformRandomVariable[Min=0.0|Max=" << nodeSpeed << "]";
283 std::stringstream ssPause;
284 ssPause << "ns3::ConstantRandomVariable[Constant=" << nodePause << "]";
285 mobilityAdhoc.SetMobilityModel ("ns3::RandomWaypointMobilityModel",
286                                "Speed", StringValue (ssSpeed.str ()),
287                                "Pause", StringValue (ssPause.str ()),
288                                "PositionAllocator", PointerValue (taPositionAlloc));
289 mobilityAdhoc.SetPositionAllocator (taPositionAlloc);
290 mobilityAdhoc.Install (adhocNodes);
291 streamIndex += mobilityAdhoc.AssignStreams (adhocNodes, streamIndex);
292 NS_UNUSED (streamIndex); // From this point, streamIndex is unused

```

ภาพที่ 4.4 ชุดคำสั่งที่ใช้ในการกำหนดตำแหน่งของโหนดไร้สาย

ในส่วนถัดไปจะเป็นการอธิบายชุดคำสั่งที่ใช้ในการกำหนดตำแหน่ง และรูปแบบการเคลื่อนที่ของโหนดในเครือข่ายผ่านชุดโค้ดในสคริปต์ตามภาพที่ 4.4 ด้วยการตั้งค่า Object “mobilityAdhoc” ที่ถูกสร้างจากคลาส MobilityHelper

การจัดวางตำแหน่งโหนดนั้นจะถูกกำหนดผ่านการใช้งาน Class: ObjectFactory ที่มีการกำหนดค่า TypeId ให้เป็นรูปแบบในการจัดวางโหนด ตัวอย่างในบรรทัดที่ 274 จะเป็นการประกาศการวางโหนดแบบสุ่มในพื้นที่ปิดรูปสี่เหลี่ยมผ่านการกำหนด TypeId ให้มีค่าเป็น “RandomRectanglePositionAllocator” โดยโค้ดในบรรทัดที่ 275-276 จะเป็นการกำหนดขอบเขตของพื้นที่ที่ใช้ในการจำลองซึ่งมีความกว้าง X และความยาว Y ซึ่งในตัวอย่างนี้คือ 300 x 1500 m จากนั้นจึงสร้าง Pointer ที่ชื่อว่า taPositionAlloc เพื่อชี้ไปยัง Object ที่ใช้ในการสุ่มตำแหน่งของโหนด ก่อนจะทำการตั้งค่าลงไปบนตัวแปร mobilityAdhoc ด้วยโค้ดบรรทัดที่ 289

ต่อไปจะเป็นการกำหนดรูปแบบการเคลื่อนที่ของโหนดด้วยโค้ดในบรรทัดที่ 285 ซึ่งจะต้องการค่าตัวแปร 4 ชนิดคือ

1. รูปแบบการเคลื่อนที่ของโหนด (ในตัวอย่างถูกกำหนดให้เป็น RandomWaypoint) ในส่วนนี้ผู้ใช้สามารถกำหนดรูปแบบการเคลื่อนที่ให้เปลี่ยนไปตามแบบจำลองอื่น ๆ ได้ ยกตัวอย่างเช่นรูปแบบการเคลื่อนที่ของมนุษย์ผ่านการเดิน หรือการเคลื่อนที่ของรถยนต์ที่จะเคลื่อนที่ผ่านแผนที่ในรูปแบบถนน เป็นต้น
2. ความเร็วของโหนด โดยในตัวอย่างจะมีการตั้งค่าการสุ่มความเร็วของโหนด และรูปแบบการกระจายเอาไว้ล่วงหน้าด้วยโค้ดบรรทัดที่ 281-282
3. ค่า Pause Time ซึ่งหมายถึงช่วงเวลาที่โหนดจะรักษาสถานะการเคลื่อนที่ และความเร็วเอาไว้ก่อนที่จะมีการสุ่มค่าใหม่เพื่อเปลี่ยนทิศทาง โดยในตัวอย่างจะมีการตั้งค่า

การสุ่มช่วงเวลาดังกล่าว และรูปแบบการกระจายเอาไว้ล่วงหน้าด้วยโค้ดบรรทัดที่ 283-284

4. ค่า Pointer ที่ชี้ไปยังตำแหน่งของโหนดที่จะกำหนดรูปแบบการเคลื่อนที่ให้ซึ่งในที่นี้คือ `taPositionAlloc` ที่ถูกกล่าวไว้ข้างต้น

หลังจากตั้งค่าตำแหน่ง และรูปแบบการเคลื่อนที่ของโหนดลงไปบน Object “mobilityAdhoc” เรียบร้อยแล้ว ผู้ใช้จะต้องทำการติดตั้ง `mobilityAdhoc` ลงไปบนโหนดที่ถูกสร้างขึ้นเอาไว้ล่วงหน้า (`adhocNodes`) ผ่านคำสั่ง `Install` ก็เป็นอันเสร็จสิ้นกระบวนการ

โดยผู้ใช้อาจสังเกตได้ว่าการสร้างตัวแปร `streamIndex` ขึ้นมาในสคริปต์นี้ ซึ่งตัวแปรดังกล่าวจะทำหน้าที่ในการควบคุม Pointer ที่เกี่ยวข้องกับการสุ่มตัวเลขที่ใช้ในการกำหนดค่าตำแหน่ง และค่าพารามิเตอร์ที่เกี่ยวข้องกับการเคลื่อนที่ของโหนด เพื่อให้แน่ใจว่าการสุ่มชุดตัวเลขที่เกิดขึ้นจะมีค่าไม่ซ้ำกัน

ภาพที่ 4.5 จะเป็นการกำหนดโปรโตคอลค้นหาเส้นทางที่จำเป็นต่อการสื่อสารให้กับโหนดในเครือข่ายเฉพาะกิจเคลื่อนที่ โดยในตัวอย่างนี้ผู้ใช้จะสามารถกำหนดโปรโตคอลที่จะใช้ในการจำลองผ่านการใส่ค่าตัวแปรในการเลือกโปรโตคอลเข้ามาผ่านทางหน้าจอ CLI ตัวอย่างเช่นหากผู้ใช้ต้องการจะจำลองการส่งข้อมูลระหว่างโหนดผ่านโปรโตคอล OLSR ผู้ใช้จะต้องสั่งรันสคริปต์ดังตัวอย่างต่อไปนี้

```
> ./waf --run "scratch/manet-routing-compare.cc --protocol=1"
```

โดยชุดโค้ดที่ใช้ในการติดตั้งโปรโตคอลลงไปบนโหนดนั้นจะอยู่ในบรรทัดที่ 325-326 ในกรณีที่ใช้ไม่กรอกข้อมูลเพื่อทำการเลือกโปรโตคอล ตัวสคริปต์จะทำการจำลองเครือข่ายด้วยการเรียกใช้โปรโตคอล AODV อย่างอัตโนมัติ

```

AodvHelper aodv;
OlsrHelper olsr;
DsdvHelper dsdv;
DsrHelper dsr;
DsrMainHelper dsrMain;
Ipv4ListRoutingHelper list;
InternetStackHelper internet;

switch (m_protocol)
{
case 1:
    list.Add (olsr, 100);
    m_protocolName = "OLSR";
    break;
case 2:
    list.Add (aodv, 100);
    m_protocolName = "AODV";
    break;
case 3:
    list.Add (dsdv, 100);
    m_protocolName = "DSDV";
    break;
case 4:
    m_protocolName = "DSR";
    break;
default:
    NS_FATAL_ERROR ("No such protocol:" << m_protocol);
}

if (m_protocol < 4)
{
    internet.SetRoutingHelper (list);
    internet.Install (adhocNodes);
}
else if (m_protocol == 4)
{
    internet.Install (adhocNodes);
    dsrMain.Install (dsr, adhocNodes);
}

NS_LOG_INFO ("assigning ip address");

```

ภาพที่ 4.5 ชุดคำสั่งที่ใช้ในการกำหนดโปรโตคอลค้นหาเส้นทาง

ภาพที่ 4.6 จะเป็นการกำหนดหมายเลขไอพี และการติดตั้งแอปพลิเคชัน ซึ่งผู้เขียนได้อธิบายไปแล้วในบทก่อนหน้านี้ ในส่วนของสคริปต์นี้จะมีการเปลี่ยนมาใช้แอปพลิเคชัน OnOff ซึ่งจะมีลักษณะการส่งแพ็กเก็ตตลอดเวลาในกรณีที่โหนดอยู่ในสถานะ On และจะหยุดส่งเมื่อโหนดอยู่ในสถานะ Off การตั้งค่าในลักษณะตามภาพคือการทำให้สถานะ Off มีค่าเป็น 0 ตลอดเวลานั้นหมายถึงการกำหนดให้โหนดต้นทางทำการส่งแพ็กเก็ตออกไปตลอดเวลาจนกว่าจะจบการจำลอง

ในส่วนของรูปในโค้ดบรรทัดที่ 345 นั้นจะเป็นการกำหนดสถานะของโหนดต้นทาง และปลายทาง ซึ่งกำกับด้วยตัวแปร nSinks ในตัวอย่างนี้เมื่อค่า nSinks ถูกกำหนดให้มีค่าเท่ากับ 10

นั่นหมายความว่า โหนด 0-9 จะทำหน้าที่เป็นโหนดปลายทาง ในขณะที่โหนด 10-19 จะทำหน้าที่เป็นโหนดต้นทาง

```
336 Ipv4AddressHelper addressAdhoc;  
337 addressAdhoc.SetBase ("10.1.1.0", "255.255.255.0");  
338 Ipv4InterfaceContainer adhocInterfaces;  
339 adhocInterfaces = addressAdhoc.Assign (adhocDevices);  
340  
341 OnOffHelper onoff1 ("ns3::UdpSocketFactory", Address ());  
342 onoff1.SetAttribute ("OnTime", StringValue ("ns3::ConstantRandomVariable[Constant=1.0]"));  
343 onoff1.SetAttribute ("OffTime", StringValue ("ns3::ConstantRandomVariable[Constant=0.0]"));  
344  
345 for (int i = 0; i < nSinks; i++)  
346 {  
347     Ptr<Socket> sink = SetupPacketReceive (adhocInterfaces.GetAddress (i), adhocNodes.Get (i));  
348  
349     AddressValue remoteAddress (InetSocketAddress (adhocInterfaces.GetAddress (i), port));  
350     onoff1.SetAttribute ("Remote", remoteAddress);  
351  
352     Ptr<UniformRandomVariable> var = CreateObject<UniformRandomVariable> ();  
353     ApplicationContainer temp = onoff1.Install (adhocNodes.Get (i + nSinks));  
354     temp.Start (Seconds (var->GetValue (100.0, 101.0)));  
355     temp.Stop (Seconds (TotalTime));  
356 }  
357
```

ภาพที่ 4.6 ชุดคำสั่งที่ใช้ในการกำหนดหมายเลขไอพีและติดตั้งแอปพลิเคชัน

```
//AsciiTraceHelper ascii;  
//Ptr<OutputStreamWrapper> osw = ascii.CreateFileStream ( (tr_name + ".tr").c_str());  
//wifiPhy.EnableAsciiAll (osw);  
AsciiTraceHelper ascii;  
MobilityHelper::EnableAsciiAll (ascii.CreateFileStream (tr_name + ".mob"));  
  
//Ptr<FlowMonitor> flowmon;  
//FlowMonitorHelper flowmonHelper;  
//flowmon = flowmonHelper.InstallAll ();
```

ภาพที่ 4.7 ชุดคำสั่งที่ใช้ในการสร้าง Mobility Ascii Trace File

อีกส่วนหนึ่งที่น่าสนใจสำหรับสคริปต์นี้คือการที่ผู้ใช้สามารถดึง Ascii Trace File ซึ่งมีข้อมูลของการเคลื่อนที่ของโหนดในเครือข่ายออกมาตามชุดคำสั่งที่ถูกแสดงไว้ในภาพที่ 4.7 ซึ่งมีประโยชน์ต่อการนำไปวิเคราะห์ผลการจำลองเครือข่ายที่เกี่ยวข้องต่อรูปแบบการเคลื่อนของโหนดได้ หรือนำแพตเทิร์นการเคลื่อนที่ซึ่งถูกสร้างขึ้นจากโปรแกรม NS3 ไปใช้ต่อกับโปรแกรมอื่นที่รองรับไฟล์นามสกุล .mob

4.2 การทำงานร่วมกันระหว่างเครือข่ายเฉพาะกิจเคลื่อนที่ และเอสดีเอ็นคอนโทรลเลอร์

หลังจากผู้ใช้ได้ทำความเข้าใจกับชุดโค้ดเพื่อการตั้งค่าการจำลองเครือข่ายเฉพาะกิจแบบพื้นฐานไปแล้ว ในส่วนนี้ผู้เขียนจะทำการอธิบายชุดโค้ดที่ผู้เขียนได้พัฒนาขึ้นเพื่อทำการควบคุมระบบเครือข่ายเฉพาะกิจเคลื่อนที่ผ่านการใช้งานเอสดีเอ็นคอนโทรลเลอร์ที่ โดยรายละเอียดทั้งหมดของงานวิจัยนี้ผู้ใช้งานสามารถที่จะไปศึกษาเพิ่มเติมได้จากรายงานฉบับเต็มที่แนบมากับคู่มือฉบับนี้

เนื่องจากผู้เขียนได้สร้างกลไกในการควบคุมเครือข่ายเฉพาะกิจผ่านเอสดีเอ็นคอนโทรลเลอร์ไว้ผ่านการแก้ไขไฟล์บางส่วนในโฟลเดอร์ใน Directory src/aodv/model โดยจะมีการแก้ไขไฟล์ aodv-routing-protocol.cc เป็นหลัก ดังนั้นหากผู้ใช้ต้องการจะศึกษา หรือพัฒนาต่อยอดโปรโทคอลที่ผู้เขียนได้พัฒนาขึ้น ผู้ใช้สามารถนำโฟลเดอร์ model ที่แนบมากับคู่มือฉบับนี้ลงไปติดตั้งแทนที่โฟลเดอร์ดั้งเดิมของโปรแกรม NS3 บน Directory ที่ผู้เขียนระบุไว้ก่อนหน้านี้ หรือผู้ใช้งานสามารถเลือกนำอิมเมจไฟล์ที่ผู้เขียนได้แนบมากับคู่มือฉบับนี้ไปเปิดใช้งานผ่านโปรแกรม Virtual Machine ต่าง ๆ ตัวอย่างเช่น VMware Workstation Player เพื่อพัฒนาต่อยอดโปรโทคอลภายใต้สภาพแวดล้อมของโปรแกรมที่ผู้เขียนได้จัดเตรียมไว้ให้ได้ในทันที

โดยในบทนี้ผู้เขียนจะทำการแนะนำชุดโค้ดที่ได้พัฒนาขึ้น รวมไปถึงสคริปต์ที่ใช้ในการจำลองระบบเครือข่ายเฉพาะกิจที่ถูกควบคุมได้ด้วยเอสดีเอ็น โดยผู้ใช้ที่ต้องการศึกษาเนื้อหาในส่วนนี้ควรจะมีพื้นฐานเกี่ยวกับหลักการทำงานของโปรโทคอล AODV [1] และเทคโนโลยี SDN [2-3] เพื่อช่วยให้ผู้ใช้สามารถทำความเข้าใจเนื้อหาได้มากยิ่งขึ้น

4.2.1 ชุดโค้ดส่วนขยายเพื่อการใช้งานเอสดีเอ็นคอนโทรลเลอร์เพื่อการค้นหาเส้นทางบนเครือข่ายเฉพาะ

หลังจากที่ผู้ใช้ทำการแทนที่โฟลเดอร์ model ตามที่ถูกระบุไว้ข้างต้นแล้ว ให้ผู้ใช้เปิดไฟล์ aodv-routing-protocol.cc ขึ้นมาเพื่อการประกอบการทำความเข้าใจเพิ่มเติมในส่วนนี้

โดยในชุดคำสั่งในเวอร์ชันนี้ จะกำหนดให้โหนดที่มีหมายเลขที่อยู่ไอพี 10.0.0.1 นั้นทำงานเป็นคอนโทรลเลอร์โหนดเสมอ ดังนั้นผู้ใช้งานจะต้องตั้งค่าสคริปต์โดยคำนึงถึงข้อจำกัดดังกล่าว เพื่อให้การทำงานของชุดโค้ดเป็นไปอย่างถูกต้อง

```
//AUNZ MOD 5 : if this BooleanValue is set to be true -> the SDN-Controller is enable while AODV fu
.AddAttribute ("EnableController", "Indicates whether a Controller enable.",
              BooleanValue (true),
              MakeBooleanAccessor (&RoutingProtocol::SetControllerEnable,
                                   &RoutingProtocol::GetControllerEnable),
              MakeBooleanChecker ())
//AUNZ MOD 13 : this function allow SDN-controller to sense the NB_list update immediately
// (assume that there are no delay or loss in this process) (for debugging)
.AddAttribute ("EnableGodmode", "Indicates whether a GodMode enable.",
              BooleanValue (false),
              MakeBooleanAccessor (&RoutingProtocol::SetGodModeEnable,
                                   &RoutingProtocol::GetGodModeEnable),
              MakeBooleanChecker ())
```

ภาพที่ 4.8 ตัวแปรที่กำหนดการเปิดใช้งานเอสดีเอ็นคอนโทรลเลอร์เพื่อการค้นหาเส้นทาง

หากผู้ใช้งานต้องการเรียกใช้โปรโทคอลการค้นหาเส้นทางบนเครือข่ายเฉพาะกิจที่สามารถถูกควบคุมได้ผ่านเอสดีเอ็นคอนโทรลเลอร์ ผู้ใช้จะต้องเลือกใช้งานโปรโทคอล AODV เป็น

โพรโทคอลการค้นหาเส้นทางเท่านั้น และผู้ใช้งานจะต้องกำหนดค่าตัวแปร “EnableController” ตามภาพที่ 4.8 ให้มีค่าเป็น true ซึ่งจะเป็นการปรับจากการทำงานของโพรโทคอล AODV แบบปกติ มาเป็นการค้นหาเส้นทางด้วยเอสดีเอ็นคอนโทรลเลอร์ โดยผู้ใช้งานสามารถเลือกแก้ไขตัวแปรในไฟล์ aodv-routing-protocol.cc โดยตรง หรือผ่านการกรอกข้อมูลตัวแปรในจังหวะที่จะเรียกใช้สคริปต์เพื่อการจำลองระบบเครือข่ายอย่างใดก็ได้

ในกรณีที่ผู้ใช้งานกำหนดค่าตัวแปรดังกล่าวให้มีค่าเป็น false ตัวชุดโค้ดดังกล่าวก็จะปรับกลับไปทำงานเป็นโพรโทคอล AODV แบบดั้งเดิม

ในส่วนถัดไปจะเป็นการอธิบายชุดคำสั่งที่สำคัญเพื่อการสร้างเอสดีเอ็นคอนโทรลเลอร์ และกลไกต่าง ๆ ที่เกี่ยวข้อง

เมื่อเอสดีเอ็นคอนโทรลเลอร์ถูกเรียกใช้งาน ทันทีที่ตัวซิมูเลเตอร์เริ่มทำงาน เอสดีเอ็นคอนโทรลเลอร์ก็จะเริ่มกระบวนการเก็บรวบรวมข้อมูลของโหนดต่าง ๆ ในเครือข่ายเฉพาะกิจเพื่อนำมาสร้างเป็นโทโพโลยีของเครือข่าย และสร้างเส้นทางให้โหนดต้นทางหากมีการร้องขอเพื่อที่จะให้โหนดคอนโทรลเลอร์มีการทำงานตามเงื่อนไขดังกล่าว ในขั้นแรกโหนดคอนโทรลเลอร์จะต้องมีการประกาศข้อความชนิดหนึ่งออกไปเป็นระยะ ๆ เพื่อระบุว่าโหนดคอนโทรลเลอร์นั้นมีตัวตนอยู่ในระบบการจำลอง โดยฟังก์ชันที่เกี่ยวข้องในการทำงานส่วนนี้คือฟังก์ชัน SendControllerAdvertisement ตามภาพที่ 4.9 โดยผู้เขียนได้ทำการดัดแปลง Hello packet ที่มีอยู่เดิมจากโพรโทคอล AODV มาใช้งานในส่วนนี้โดยมีการกำหนดประเภทของเฮดเดอร์ใหม่ให้มีค่าเป็น “AODVTYPE_CONTROLLER” และจะถูกเรียกในชื่อใหม่ว่า Controller Advertisement (C_{adv}) โดยแพ็กเก็ตนี้จะถูกแพร่กระจายออกไปทั่วทั้งเครือข่าย

```

2903 //AUNZ MOD 5 : Create SendControllerAdvertisement
2904 void
2905 RoutingProtocol::SendControllerAdvertisement ()
2906 {
2907     NS_LOG_FUNCTION (this);
2908     /* Broadcast a RREP with TTL = 1 with the RREP message fields set as follows:
2909     * Destination IP Address      The node's IP address.
2910     * Destination Sequence Number The node's latest sequence number.
2911     * Hop Count                   0
2912     * Lifetime                    AllowedHelloLoss * HelloInterval
2913     */
2914     ctl_seqNo++;
2915     for (std::map<Ptr<Socket>, Ipv4InterfaceAddress>::const_iterator j = m_socketAddresses.begin (); j
2916     {
2917         Ptr<Socket> socket = j->first;
2918         Ipv4InterfaceAddress iface = j->second;
2919         RrepHeader helloHeader (/*prefix size=*/ 0, /*hops=*/ 0, /*dst=*/ iface.GetLocal (), /*Controll
2920                                 /*origin=*/ iface.GetLocal (), /*lifetime=*/ Time (m_co
2921         Ptr<Packet> packet = Create<Packet> ();
2922         SocketIpTtlTag tag;
2923         tag.SetTtl (1);
2924         packet->AddPacketTag (tag);
2925         packet->AddHeader (helloHeader);
2926         TypeHeader tHeader (AODVTYPE_CONTROLLER);
2927         packet->AddHeader (tHeader);
2928         // Send to all-hosts broadcast if on /32 addr, subnet-directed otherwise
2929         Ipv4Address destination;
2930         if (iface.GetMask () == Ipv4Mask::GetOnes ())
2931         {
2932             destination = Ipv4Address ("255.255.255.255");
2933         }
2934         else
2935         {
2936             destination = iface.GetBroadcast ();
2937         }
2938         Time jitter = Time (MilliSeconds (m_uniformRandomVariable->GetInteger (0, 10)));
2939         Simulator::Schedule (jitter, &RoutingProtocol::SendTo, this, socket, packet, destination);
2940     }
2941 }
2942

```

ภาพที่ 4.9 ฟังก์ชัน SendControllerAdvertisement

```

1539 void
1540 RoutingProtocol::RecvControllerAdvertisement (Ptr<Packet> p, Ipv4Address receiver, Ipv4Address sender)
1541 {
1542     NS_LOG_FUNCTION (this);
1543     RrepHeader rrepHeader;
1544     p->RemoveHeader (rrepHeader);
1545     Ipv4Address dst = rrepHeader.GetDst ();
1546     NS_LOG_LOGIC ("RREP destination " << dst << " RREP origin " << rrepHeader.GetOrigin ());
1547     //HOP COUNT INCREMENT
1548     uint8_t hop = rrepHeader.GetHopCount () + 1;
1549     rrepHeader.SetHopCount (hop);
1550     Ptr<NetDevice> dev = m_ipv4->GetNetDevice (m_ipv4->GetInterfaceForAddress (receiver));
1551     RoutingTableEntry newEntry (/*device=*/ dev, /*dst=*/ dst, /*validSeqNo=*/ true, /*seqno=*/ rrepHeader
1552                                 /*iface=*/ m_ipv4->GetAddress (m_ipv4->GetInterfaceForAddress (receiver)),
1553                                 /*nextHop=*/ sender, /*lifeTime=*/ rrepHeader.GetLifeTime ());
1554     // Broadcast indicator
1555     bool sequence_number_equal = false;
1556     RoutingTableEntry toDst;
1557     if (m_routingTable.LookupRoute (dst, toDst))
1558     {
1559         sequence_number_equal = true;
1560     }
1561 }

```

ภาพที่ 4.10 ฟังก์ชัน RecvControllerAdvertisement

โหนดในเครือข่ายจะทำการตอบสนองต่อแพ็กเก็ต C_{adv} โดยการรับแพ็กเก็ตเข้ามา และใช้ข้อมูลจากแพ็กเก็ตเฮดเดอร์นั้นมาสร้างเส้นทางย้อนกลับไปหาโหนดคอนโทรลเลอร์ก่อนจะแพร่กระจายแพ็กเก็ตนี้ต่อไป และโหนดที่ได้รับแพ็กเก็ต C_{adv} จะทำการตั้งเวลาในการรายงานข้อมูล Link-state ของตัวเองไปยังคอนโทรลเลอร์โหนดเป็นระยะ ๆ โดยกำหนดให้

ช่วงเวลานั้นสอดคล้องกับค่าบในการประกาศ C_{adv} โดยพฤติกรรมดังกล่าวจะถูกควบคุมผ่าน ฟังก์ชัน RecvControllerAdvertisement ตามภาพที่ 4.10

```

2849 //AUNZ MOD 2 : Create SendNeighborList from SendHello
2850 void
2851 RoutingProtocol::SendNeighborList (std::string flag)
2852 {
2853     NS_LOG_FUNCTION (this);
2854
2855     RoutingTableEntry toController;
2856     Ipv4Address controller = "10.0.0.1";
2857     if(m_routingTable.LookupRoute (controller, toController)){
2858
2859         RrepHeader rrepHeader (/*prefix size=*/ 0, /*hops=*/ 0, /*dst=*/ m_ipv4->GetAddress(1,0).GetLocal
2860                                 /*origin=*/ controller,/*lifetime=*/ Time (m_contro
2861
2862         //AUNZ MOD 2 : Add payload Msg
2863         std::string nblast = m_routingTable.GetNBLIST();
2864         //AUNZ MOD 11: GOD MODE SET
2865         std::map<Ipv4Address, std::string>::iterator it = godnblast.find(m_ipv4->GetAddress(1,0).GetL
2866         if (it != godnblast.end()){
2867             it->second = nblast;
2868         }else{
2869             godnblast.insert(std::pair<Ipv4Address, std::string>( m_ipv4->GetAddress(1,0).GetLocal(),
2870         }
2871
2872         // AUNZ MOD 7 : Add all Dest Ip after "|" into Nblast to act as RREQ
2873         nblast.append("|");
2874         if(m_queue.GetSize()!=0){
2875             nblast.append(m_queue.GetQueueIp());
2876         }
2877
2878         // AUNZ MOD 12 : Append flag to NBLIST
2879         nblast.append(flag);
2880
2881         // AUNZ MOD 13: print check nblast after link break
2882         if(flag == "2"){
2883             std::cout << "this nblast is : " << nblast << "\n";
2884         }
2885     }

```

ภาพที่ 4.11 ฟังก์ชัน SendNeighborlist

หลังจากที่โหนดได้รับ C_{adv} แล้วโหนดนั้น ๆ ก็จะสามารถรับรู้เส้นทางในการติดต่อไปหา โหนดคอนโทรลเลอร์ได้ โดยโหนดจะทำการรายงานข้อมูล Link-state ของตัวเองผ่านการส่ง แพ็กเก็ต NB_list ไปให้โหนดคอนโทรลเลอร์ โดยการแนบข้อมูล Link-state ของตัวเองลงไปใน Payload ของแพ็กเก็ต ผ่านการเรียกใช้งานฟังก์ชัน SendNeighborlist ตามภาพที่ 4.11 โดยนอกเหนือจากข้อมูล Link-State แล้ว Payload ในแพ็กเก็ตนี้จะมีข้อมูลที่สำคัญ อีก 2 ฟิลด์ซึ่งก็คือ

1. หมายเลขไอพีของโหนดปลายทางที่โหนดดังกล่าวต้องการติดต่อด้วย ซึ่งฟิวด์นี้จะถูกใช้หากโหนดนั้นมีข้อมูลที่ต้องการจะส่งไปยังปลายทาง การแนบหมายเลขไอพีของโหนดปลายทางไปด้วยจะทำให้โหนดคอนโทรลเลอร์ทราบว่าโหนดดังกล่าวกำลังร้องขอเส้นทาง
2. Flag ซึ่งจะเป็นการบอกสถานะของแพ็กเก็ตซึ่งจะมีอยู่ทั้งหมด 3 รูปแบบ
 - a. Flag = 0 คือแพ็กเก็ต NB_list แบบปกติ

- b. Flag = 1 คือแพ็กเก็ต NB_list เพื่อใช้ในการร้องขอเส้นทางไปยังโหนดคอนโทรลเลอร์อีกครั้ง หากตรวจพบว่าเส้นทางที่มีอยู่เดิมไม่สามารถใช้งานได้ ซึ่งจำนวนครั้งในการร้องขอเส้นทางใหม่จะถูกกำหนดด้วยตัวแปร Token
- c. Flag = 2 คือแพ็กเก็ต NB_list ที่ใช้ในการแจ้งโหนดคอนโทรลเลอร์ให้ทราบว่าโหนดได้ทำการตรวจพบเส้นทางขาดจากการแจ้งจากโหนดอื่น และโหนดนั้น ๆ ได้รับผลกระทบจากการขาดของเส้นทางนั้นด้วย แพ็กเก็ตรูปแบบนี้จะถูกส่งออกไปเพื่อทำการปรับปรุงข้อมูล Link-state ที่ถูกเก็บไว้บนโหนดคอนโทรลเลอร์ให้สอดคล้องกับสถานการณ์ของเครือข่ายตลอดเวลา

โดยระหว่างการส่งต่อแพ็กเก็ต NB_list กลับไปหาโหนดคอนโทรลเลอร์ โหนดระหว่างทางก็จะทำการสร้างเส้นทางย้อนกลับไปหาโหนดซึ่งเป็นเจ้าของแพ็กเก็ต NB_list ด้วย การทำเช่นนี้จะทำให้โหนดคอนโทรลเลอร์สามารถติดต่อไปหาโหนดทุกโหนดในเครือข่ายได้ โดยผู้เขียนได้นำแพ็กเก็ต Route Reply (RREP) จากโปรโตคอล AODV มาปรับใช้งานเพื่อให้ได้ผลลัพธ์ตามที่ผู้เขียนออกแบบไว้

```

2201 void
2202 RoutingProtocol::ProcessUnicastNBList (RrepHeader const & rrepHeader, Ipv4Address receiver, std::string
2203 {
2204
2205
2206     NS_LOG_FUNCTION (this << "from " << rrepHeader.GetDst ());
2207
2208     // AUNZ MOD 7 Process data from Payload @ controller -> based on Mod 3
2209     if(m_ipv4->GetAddress(1,0).GetLocal() == "10.0.0.1"){
2210
2211         std::vector<std::string> processednb;
2212         std::vector<std::string> rreqlist;
2213         boost::split(processednb, nbdata, boost::is_any_of("|"));
2214
2215         RoutingTableEntry Dataplane;
2216         if (m_routingTable.LookupRoute (rrepHeader.GetDst (), Dataplane))
2217         {
2218             Dataplane.setNblist(processednb[0]);
2219             m_routingTable.Update (Dataplane);
2220         }
2221
2222         //AUNZ MOD 13 : Intermediate Node update NBLIST to Controller after route break -> then return
2223         if(processednb[2] == "2"){
2224             return;
2225         }
2226
2227         std::vector<std::string> buff_cal_path = Dataplane.getCalculatedPath();
2228         if(processednb[1] != ""){
2229             boost::split(rreqlist, processednb[1], boost::is_any_of(" "));
2230
2231             //AUNZ mod 11 -> retransmission Instruction if fail [bound with SendInstruction]
2232             rreqlist.push_back("new");
2233
2234             rreqcount++;
2235             if(1){
2236                 std::cout << "PATHREQCOUUNT " << rreqcount << "\n";
2237             }
2238
2239             //AUNZ MOD 12 -> emergency request -> send instruction with blacklist [No delay]
2240             //Check flag
2241             if(processednb[2] == "1"){
2242                 Simulator::Schedule (Seconds (0.1), &RoutingProtocol::SendInstruction, this, rrepHeader, rre
2243                 return;
2244
2245

```

ภาพที่ 4.12 ฟังก์ชัน ProcessUnicastNBList

หลังจากที่แพ็กเก็ต NB_List ถูกส่งกลับมาจนถึงโหนดคอนโทรลเลอร์แล้ว โหนดคอนโทรลเลอร์ก็จะนำข้อมูลจาก NB_list ไปประมวลผลเพื่อใช้ในการสร้าง หรือปรับปรุง โทโพลีของเครือข่าย และทำการค้นหาเส้นทางไปยังปลายทางหากมีการร้องขอผ่านการเรียกใช้ฟังก์ชัน ProcessUnicastNBList ตามภาพที่ 4.12

```

:60 void
:61 RoutingProtocol::SendInstruction(RrepHeader const & rrepHeader, std::vector<std::string> rreqlist, std:
:62
:63 //AUNZ mod 11 -> retransmission Instruction if it fail
:64 int new_req_flag=0;
:65 if(rreqlist[rreqlist.size()-1]=="new"){
:66     rreqlist.pop_back();
:67     //the retransmission may cause packet loss if the calculated path is not actually suitable
:68     new_req_flag = 1; //-> enable this line to allow Controller to retransmit Instruction
:69 }
:70
:71 for (unsigned int i = 0; i < rreqlist.size(); i++){
:72     IpV4Address dest(rreqlist[i].c_str());
:73
:74     //Aunz Mod 12
:75     //-----
:76     std::vector<std::string> blacklist;
:77     RoutingTableEntry checkcalculatedroute;
:78
:79     if(flag == "1"){
:80         if (m_routingTable.LookupRoute (rrepHeader.GetDst (), checkcalculatedroute)){
:81             if(checkcalculatedroute.getCalculatedPath().size()!=0){
:82                 blacklist = checkcalculatedroute.getCalculatedPath();
:83                 blacklist.pop_back(); // remove cost
:84                 blacklist.pop_back(); // remove source
:85                 blacklist.erase(blacklist.begin()); // remove dest
:86             }
:87         }
:88
:89         //Switch of Blacklist -> comment to enable blacklist mode
:90         blacklist.clear();
:91     }
:92     //-----
:93     std::vector<std::string> bestroute = Pathfinder(rrepHeader.GetDst(), dest, blacklist);
:94
:95     RoutingTableEntry savecalculatedroute;
:96
:97     if(bestroute.size()<=1){
:98         std::vector<std::string> resetroute;
:99         savecalculatedroute.setCalculatedPath(resetroute);
:00         m_routingTable.Update (savecalculatedroute);
:01         return;
:02     }
:03

```

ภาพที่ 4.13 ฟังก์ชัน SendInstruction

หากเกิดการร้องขอเส้นทางขึ้น โหนดคอนโทรลเลอร์จะทำการค้นหาเส้นทางผ่านการเรียกใช้ฟังก์ชัน SendInstruction ตามภาพที่ 4.13 โดยจะมีการเรียกใช้ฟังก์ชันย่อย Pathfinder ซึ่งเป็นการเรียกใช้ Dijkstra Algorithm เพื่อการค้นหาเส้นทางที่สั้นที่สุด ก่อนที่จะนำเส้นทางที่ได้มาใช้ในการสร้างแพ็กเก็ตชนิดใหม่ที่ชื่อว่า Instruction packet ที่จะถูกใช้ในการปรับปรุง หรือเปลี่ยนแปลงข้อมูลในตารางเส้นทางของโหนดเป้าหมายทั้งหมดที่ถูกเลือกให้ทำหน้าที่เป็นส่วนหนึ่งของเส้นทาง โดยเริ่มต้นจากการส่งแพ็กเก็ต Instruction ไปยังโหนดปลายทางไล่มาจนถึงโหนดต้นทาง เมื่อโหนดต้นทางได้รับแพ็กเก็ต Instruction แล้วโหนดต้นทางก็จะเริ่มต้นการส่งแพ็กเก็ตออกไปหาโหนดปลายทาง

โดยในฟังก์ชัน SendInstruction นี้จะมีตัวแปรอาร์เรย์ blacklist ที่จะมีการทำงานร่วมกับ NB_list ที่มี flag เป็น 1 (เส้นทางที่โหนดคอนโทรลเลอร์เคยสร้างให้ไม่สามารถใช้งานได้) โดยที่โหนดคอนโทรลเลอร์จะไม่สร้าง และส่งเส้นทางที่ประกอบด้วยโหนดกลุ่มเดิมซึ่งอาจมีผลต่อเส้นทางที่ขาดไปแล้วกลับไปให้โหนดต้นทางที่ร้องขอมาอีกจนกว่าจะถึงคาบการประกาศ C_{adv} รอบถัดไป โดยที่ผู้ใช้จะสามารถเลือกเปิดฟังก์ชันนี้ได้ผ่านการคอมเมนต์โค้ด blacklist.clear(); ออกไป

ฟังก์ชันที่ไม่ได้ถูกนำมาใช้จริงในชุดโค้ดนี้ ได้แก่ฟังก์ชัน ProcessNBList และ SendNeighborList ซึ่งผู้เขียนใช้ในการศึกษาการแก้ชุดคำสั่งเบื้องต้นเท่านั้น

ในกรณีที่ผู้ใช้ต้องการเรียนรู้วิธีการตั้ง Timer เพื่อการส่งข้อมูลออกไปเป็นรายคาบ ผู้ใช้สามารถศึกษาได้จากชุดฟังก์ชันที่เกี่ยวข้องกับการแพร่กระจายแพ็กเก็ต C_{adv} ได้

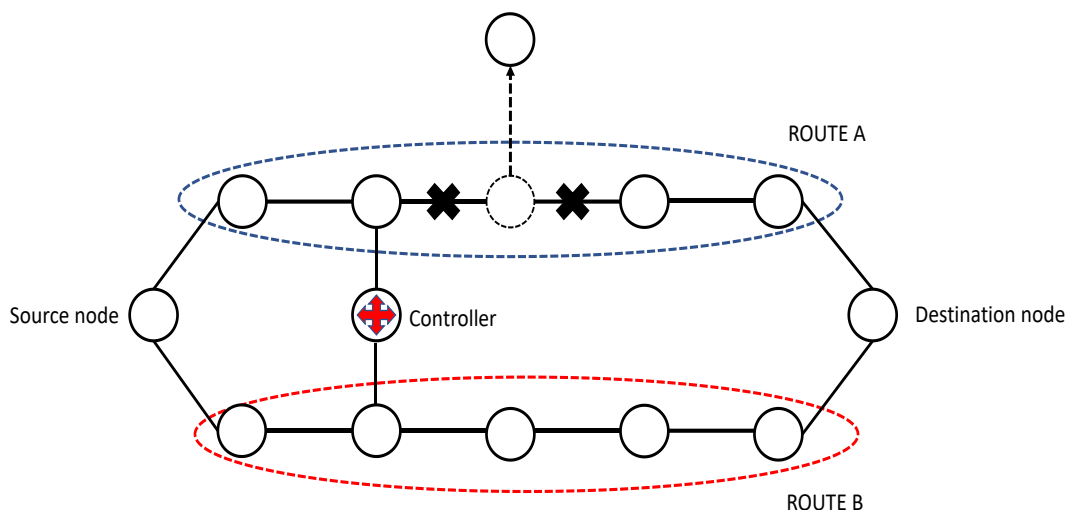
นอกเหนือจากไฟล์ aodv-routing-protocol.cc แล้ว จะยังมีอีก 2 ไฟล์หลัก ๆ ที่เกี่ยวข้องกับการพัฒนาเอสดีเอ็นคอนโทรลเลอร์เพื่อการค้นหาเส้นทางบนเครือข่ายเฉพาะกิจ ซึ่งก็คือ

1. aodv-ratable.cc ซึ่งจะเกี่ยวข้องกับการบันทึก หรือดึงข้อมูลจาก Routing Table ของโหนดมาใช้งาน
2. aodv-rqueue.cc จะเป็นชุดคำสั่งที่เกี่ยวข้องกับการจัดการคิวบนโหนด

4.2.2 สคริปต์เพื่อการเรียกใช้งานการจำลองระบบเครือข่ายเฉพาะกิจที่ถูกควบคุมได้ด้วย เอสดีเอ็นคอนโทรลเลอร์

โดยพื้นฐานแล้วชุดโค้ดนี้จะถูกนำไปใช้ในการกำหนด Routing Protocol บนทุก ๆ โหนดที่ถูกสร้างขึ้นในการจำลอง (ยกเว้นเสียแต่ว่าผู้ใช้จะทำการกำหนดค่าบนสคริปต์ที่แตกต่างออกไป) ดังนั้นผู้ใช้จะต้องทำการแก้ไขสคริปต์ และการตั้งค่าเงื่อนไขต่าง ๆ ให้สอดคล้องกับเงื่อนไขข้างต้น

ในส่วนนี้ผู้เขียนได้ทำการแนบไฟล์สคริปต์ที่จะใช้ในการจำลองการทำงานของเอสดีเอ็นคอนโทรลเลอร์เพื่อการค้นหาเส้นทางบนเครือข่ายเฉพาะกิจแบบพื้นฐานเอาไว้ด้วย ซึ่งไฟล์ดังกล่าวมีชื่อว่า sdn-manet-demo.cc โดยผู้ใช้สามารถคัดลอกสคริปต์นี้ไปไว้ที่โฟลเดอร์ scratch เพื่อทำการทดสอบการใช้งาน



ภาพที่ 4.14 โทโพโลยีของเครือข่ายที่ถูกสร้างขึ้นผ่านสคริปต์ sdn-manet-demo.cc

ภาพที่ 4.14 จะเป็นโทโพโลยีของเครือข่ายที่สคริปต์นี้ได้ทำการสร้างขึ้น ซึ่งมีการจัดวาง โหนดต้นทาง และปลายทางเอาไว้ตามภาพ ในส่วนของโหนดระหว่างทางนั้นจะถูกจัดวางไว้ เพื่อให้โหนดต้นทางสามารถมีเส้นทางในการติดต่อไปหาโหนดปลายทางได้ 2 เส้นทาง (เส้นทาง A และเส้นทาง B) ในส่วนของโหนดคอนโทรลเลอร์ ก็จะถูกกำหนดตำแหน่งเอาไว้ ให้สามารถติดต่อหาโหนดได้ผ่านโหนดทั้ง 2 โหนดจากทั้งโหนดในเส้นทาง A ด้านบน และ โหนดในเส้นทาง B ด้านล่าง โดยรูปแบบการกำหนดพิกัดของโหนดให้เจาะจงลงไปบนพื้นที่ การจำลองผ่านการกำหนดโดยผู้ใช้งานสามารถตรวจสอบได้จากชุดคำสั่งตามภาพที่ 4.15 โดยที่ พารามิเตอร์ในคลาส Vector คือพิกัดของโหนดในรูปแบบ (X,Y,Z) ซึ่งหมายความว่าผู้ใช้งาน สามารถกำหนดการวางตำแหน่งโหนดในรูปแบบ 3 มิติได้ ซึ่งจะมีความสำคัญหากผู้ใช้งาน ทำการจำลองระบบเครือข่ายเกี่ยวกับพาหนะที่สามารถเคลื่อนที่แบบ 3 มิติ เช่นโดรน เป็นต้น

```
//Set Controller Mobility
//Important Note: node ID=0 must be assigned to be a controller node in every setup
MobilityHelper mobility;
Ptr<ListPositionAllocator> positionAlloc = CreateObject<ListPositionAllocator> ();
positionAlloc->Add (Vector (800.0, 150.0, 0.0)); //Controller Node
positionAlloc->Add (Vector (2000.0, 150.0, 0.0)); // Destination Node
positionAlloc->Add (Vector (0.0, 150.0, 0.0)); // Source Node
positionAlloc->Add (Vector (0.0, 300.0, 0.0));
positionAlloc->Add (Vector (250.0, 300.0, 0.0));
positionAlloc->Add (Vector (500.0, 300.0, 0.0));
positionAlloc->Add (Vector (750.0, 300.0, 0.0));
positionAlloc->Add (Vector (1000.0, 300.0, 0.0)); // Moved node
positionAlloc->Add (Vector (1250.0, 300.0, 0.0));
positionAlloc->Add (Vector (1500.0, 300.0, 0.0));
positionAlloc->Add (Vector (1750.0, 300.0, 0.0));
positionAlloc->Add (Vector (2000.0, 300.0, 0.0));
positionAlloc->Add (Vector (0.0, 0.0, 0.0));
positionAlloc->Add (Vector (250.0, 0.0, 0.0));
positionAlloc->Add (Vector (500.0, 0.0, 0.0));
positionAlloc->Add (Vector (750.0, 0.0, 0.0));
positionAlloc->Add (Vector (1000.0, 0.0, 0.0));
positionAlloc->Add (Vector (1250.0, 0.0, 0.0));
positionAlloc->Add (Vector (1500.0, 0.0, 0.0));
positionAlloc->Add (Vector (1750.0, 0.0, 0.0));
positionAlloc->Add (Vector (2000.0, 0.0, 0.0));

mobility.SetPositionAllocator (positionAlloc);
mobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");
mobility.Install (nodes);
```

ภาพที่ 4.15 ชุดคำสั่งเพื่อกำหนดตำแหน่งของโหนด

เนื่องจากลำดับการแพร่กระจายข้อความต่าง ๆ ที่มีผลต่อการเลือกเส้นทางบนคอนโทรลเลอร์นั้นจะเกิดขึ้นแบบสุ่มเมื่อจำนวนฮอปของเส้นทาง A และ B มีค่าเท่ากัน หากผู้ใช้ทำการเปลี่ยนค่า Seed สำหรับใช้ในการสุ่มเหตุการณ์ ผลลัพธ์ของการจำลองก็จะเปลี่ยนไป โดยในสถานการณ์จำลองที่ผู้เขียนสร้างขึ้น (Seed = 12345) เอสดีเอ็นคอนโทรลเลอร์จะเลือกใช้เส้นทาง A ก่อน

การแก้ไขค่า Seed นั้นจะมีความจำเป็นหากผู้ใช้งานต้องการจำลองเหตุการณ์สุ่มจำนวนมาก ก่อนจะนำผลลัพธ์จากการสุ่มเหตุการณ์ในหลายรูปแบบมาหาประสิทธิภาพเฉลี่ยของโปรโตคอล หรือแนวคิดที่ถูกพัฒนาจากผู้ใช้

โดยเมื่อระยะเวลาผ่านไปถึงวินาทีที่ 32 จะมีโหนดหนึ่งในเส้นทางทำการเคลื่อนที่ออกจากจุดเดิม และกระตุ้นให้เอสดีเอ็นคอนโทรลเลอร์เปลี่ยนเส้นทางในการส่งข้อมูลให้โหนดต้นทาง โดยชุดคำสั่งเหล่านี้จะถูกแสดงไว้ตามภาพที่ 4.16

```
// a node (ID:7) is moved after 32 second has passed
Ptr<Node> node = nodes.Get (7);
Ptr<MobilityModel> mob = node->GetObject<MobilityModel> ();
Simulator::Schedule (Seconds (32), &MobilityModel::SetPosition, mob, Vector (500.0, 1000.0, 0.0));
```

ภาพที่ 4.16 ชุดคำสั่งเพื่อย้ายตำแหน่งของโหนดตามเวลาที่กำหนด

ภาพที่ 4.17 จะเป็นการขยายความชุดคำสั่งบางส่วนที่ผู้เขียนได้มีการปรับแก้ไข โดยการตั้งค่า PropagationLoss ให้มีค่าเป็น RangePropagationLossModel นั้นจะทำให้โหนดมีระยะการแพร่กระจายข้อความที่สามารถถูกกำหนดระยะจากผู้ใช้ได้โดยตรง (ในสคริปต์นี้ถูกกำหนดให้มีค่าเท่ากับ 250 เมตร) ในขณะที่โมเดลอื่น ๆ ผู้ใช้จะต้องป้อนค่าพลังงานที่ใช้ในการส่งข้อความเพื่อใช้ในการกำหนดระยะในการแพร่กระจาย

ในส่วนของการตั้งค่าแอปพลิเคชัน On-Off นั้น ผู้ใช้ได้มีการกำหนดค่า DataRate ให้มีค่าเท่ากับ 20480 bps เพื่อให้สามารถส่งข้อมูล 512 Byte ออกไปได้ 5 แพ็กเก็ตต่อวินาทีนั่นเอง

```

257 //Set Nodes transmission range to 250 m (radius)
258 //-----
259 Config::SetDefault( "ns3::RangePropagationLossModel::MaxRange", DoubleValue( step ) );
260 YansWifiChannelHelper wifiChannel;
261 wifiChannel.SetPropagationDelay( "ns3::ConstantSpeedPropagationDelayModel" );
262 wifiChannel.AddPropagationLoss( "ns3::RangePropagationLossModel" );
263 //-----
264
265 wifiPhy.SetChannel (wifiChannel.Create ());
266 WifiHelper wifi;
267 wifi.SetRemoteStationManager ("ns3::ConstantRateWifiManager", "DataMode", StringValue ("OfdmRate6M"));
268 devices = wifi.Install (wifiPhy, wifiMac, nodes);
269
270
271 if (pcap)
272 {
273     wifiPhy.EnablePcapAll (std::string ("aodv"));
274 }
275
276 //3) Install Internet Stacks
277 //(Since SDN controller is implemented on top of AODV routing protocols, the AODV routing helper is:
278 AodvHelper aodv;
279 InternetStackHelper stack;
280 stack.SetRoutingHelper (aodv); // has effect on the next Install ()
281 stack.Install (nodes);
282 Ipv4AddressHelper address;
283 address.SetBase ("10.0.0.0", "255.0.0.0");
284 interfaces = address.Assign (devices);
285
286 //On-Off Helper application (set to send 5 packet per sec (CBR))
287 int nSinks = 1;
288 OnOffHelper onoff1 ("ns3::UdpSocketFactory", Address ());
289 onoff1.SetAttribute ("OnTime", StringValue ("ns3::ConstantRandomVariable[Constant=1.0]"));
290 onoff1.SetAttribute ("OffTime", StringValue ("ns3::ConstantRandomVariable[Constant=0.0]"));
291 //512 Byte = 4092 bps / 20480 = 5 packet/s
292 onoff1.SetAttribute ("DataRate", DataRateValue (20480));

```

ภาพที่ 4.17 อธิบายขยายความชุดคำสั่งในบางส่วนเพิ่มเติม

```
thananop@thananop-OptiPlex-9020:~/source/ns-3.33$ ./waf --run scratch/sdn-manet-demo.cc
Waf: Entering directory `/home/thananop/source/ns-3.33/build'
Waf: Leaving directory `/home/thananop/source/ns-3.33/build'
Build commands will be stored in build/compile_commands.json
'build' finished successfully (1.149s)
Creating 21 nodes 250 m apart.
Starting simulation for 35 s ...
PATHREQCOUNT 1
PATHREQCOUNT 2
PATHREQCOUNT 3
PDR :94
thananop@thananop-OptiPlex-9020:~/source/ns-3.33$
```

ภาพที่ 4.18 ผลลัพธ์ของการเรียกใช้สคริปต์ sdn-manet-demo.cc

โดยผลลัพธ์ของการเรียกใช้สคริปต์นี้จะมีการแสดงผลข้อมูลออกมาบนหน้าต่าง CLI ตามภาพที่ 4.18 โดย PATHREQCOUNT นั้นคือจำนวนครั้งในการส่งการร้องขอเส้นทางไปที่คอนโทรลเลอร์ และค่า PDR นั้นคืออัตราการส่งสำเร็จของข้อมูลจากต้นทางไปยังปลายทาง นอกจากนี้ตัวสคริปต์ยังทำการสร้าง Trace File และ xml ไฟล์เพื่อใช้ในการวิเคราะห์ผลลัพธ์ออกมาตามที่ผู้เขียนได้กล่าวไว้ในบทก่อนหน้าอีกด้วย

บทที่ 5

แนะนำแหล่งอ้างอิงเพื่อการศึกษาเพิ่มเติม

5.1 Open flow V 1.3

ถึงแม้โปรแกรม NS3 จะมีโมดูลเพื่อการจำลองเอสดีเอ็นคอนโทรลเลอร์มาพร้อมใช้งาน แต่โมดูลดังกล่าวนั้นล้าสมัย และไม่ได้มีการปรับปรุงให้เข้ากับมาตรฐานปัจจุบันของเทคโนโลยี หากผู้ใช้งานต้องการใช้โปรแกรม NS3 เพื่อใช้ในการจำลองเอสดีเอ็นคอนโทรลเลอร์เพื่อการควบคุมอุปกรณ์เครือข่ายแบบมีสายที่ซับซ้อนการทำงานของสถาปัตยกรรมเอสดีเอ็น ตัวอย่างเช่น สวิตช์ หรือเราเตอร์ ผู้เขียนแนะนำให้ผู้ใช้ศึกษาข้อมูลเพิ่มเติมได้จาก URL: <http://www.lrc.ic.unicamp.br/ofswitch13/> ซึ่งในหน้าเว็บไซต์ดังกล่าวได้มีการเผยแพร่ชุดคำสั่งเพื่อสร้างโมดูลที่จำเป็นต่อการใช้งานร่วมกับสถาปัตยกรรมเอสดีเอ็นทั้งหมด รวมไปถึงคู่มือในการติดตั้งโมดูลนี้ลงไปในโปรแกรม NS3 และวิธีการใช้งานในเบื้องต้นเอาไว้อีกด้วย

บรรณานุกรม

- [1] C. Perkins, E. Belding-Royer, and S. Das, "Ad hoc On-Demand Distance Vector (AODV) Routing", RFC 3561, July 2003
- [2] W. Xia, Y. Wen, C. H. Foh, D. Niyato and H. Xie, "A Survey on Software-Defined Networking," in IEEE Communications Surveys & Tutorials, vol. 17, no. 1, pp. 27-51, Firstquarter 2015.
- [3] อนุชิตา มัชฌิมา, ชยุตม์ สว่าง, วรวัชร ณรงค์ขวนะ, ธนานพ ทองถาวร, และ สุเมธ ประภาวัต, "Network Management for Traffic Distribution using SDN Architecture," รายงานการประชุมวิชาการระดับประเทศด้านเทคโนโลยีสารสนเทศ ครั้งที่ 11 (National Conference on Information Technology: NCIT 2019), ต.ค. 2562.