บทที่ 3

หลักการทำงานและการเพิ่มขยายโปรแกรม ns

3.1 ลักษณะทั่วไปและการจำลองด้วยโปรแกรม ns

โปรแกรมจำลอง Network Simulator หรือ ns[1] ใช้สำหรับการจำลองทางเครือข่ายคอมพิวเตอร์ (NS-2 Website, 2004) โดยการพัฒนาเริ่มต้นขึ้นในช่วงปี 1989 โดยการนำโปรแกรม Real Network Simulator ของมหาวิทยาลัย Cornell มาปรับปรุงและในปี 1995 ภายใต้การสนับสนุนของ DARPA (Defense Advanced Research Projects Agency) ผ่านโครงการ VINT (Virtual InterNeTwork) ที่เป็นความร่วมมือในการพัฒนาระหว่างมหาวิทยาลัย Southern Califonia (USC), Xerox Parc, Lawrence Berkeley National Laboratory (LBNL) และ มหาวิทยาลัย Califonia ที่ Berkeley โดยในปัจจุบัน USC เป็นผู้ดูแลหลัก

โปรแกรมจำลอง ns เป็นโปรแกรมจำลองที่พัฒนาขึ้นแบบโอเพนซอร์สและใช้แนวคิดของการโปรแกรมเชิงอ็อบเจกต์ จึงเปิดโอกาสให้นักพัฒนาได้สร้างแบบจำลองของโพรโทคอลทางเครือข่ายคอมพิวเตอร์ แล้วนำโพรโทคอลเหล่านั้นมาทำงานเป็นระบบร่วมกับโพรโทคอลอื่นๆ เพื่อการศึกษาและเปรียบเทียบสมรรถนะหรือข้อจำกัดของโพรโทคอลได้ในสภาพแวดล้อมที่กำหนดขึ้น

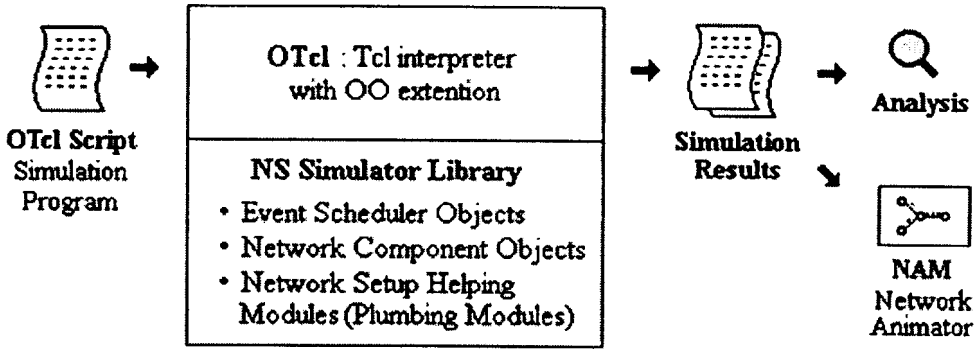
ในปัจจุบัน ns สามารถทำงานได้ทั้งบนระบบปฏิบัติการ Linux และ Windows และสนับสนุนแบบจำลองทางเครือข่ายจำนวนมาก (Fall and Varadhan, 2002) เช่น แบบจำลองของลิงค์ไม่ว่าจะเป็นแบบมีสายหรือไร้สาย แบบจำลองของโหนดทั้งที่เป็นคอมพิวเตอร์ เราเตอร์หรือสวิตซ์ และของโพรโทคอลต่างๆ ซึ่งสร้างขึ้นอย่างเป็นลำดับชั้น อาทิ ในชั้นทรานสปอร์ต ก็จะประกอบด้วย UDP TCP ฯลฯ และในชั้นโปรแกรมประยุกต์ ที่ประกอบด้วยโพรโทคอล Telnet FTP HTTP เป็นต้น

กระบวนการใช้งาน ns แสดงดังรูปที่ 3.1 โดยเริ่มจากการที่ผู้ใช้กำหนดค่าการทดลองโดยผ่านทาง OTcl สคริปต์ เช่นกำหนดลักษณะการเชื่อมต่อ แบนด์วิดท์ของลิงค์ โพรโทคอลในชั้นต่างๆ และเมื่อระบบทำงานเสร็จสิ้นก็จะได้แฟ้มผลลัพธ์ออกมา ซึ่งผู้ใช้จะต้องประยุกต์เครื่องมือเสริมอื่นๆ เช่น โปรแกรมตารางคำนวณ โปรแกรม awk หรือ ภาษา perl เพื่อวิเคราะห์แฟ้มผลลัพธ์ นอกจากนี้

---

[1] ns เป็นเครื่องหมายอันมีลิขสิทธิ์ ซึ่งหมายถึง โปรแกรมจำลอง Network Simulator (Fall and Varadhan, 2002)

ยังสามารถนำผลลัพธ์การจำลองมาโดยผ่านโปรแกรม Network AniMator หรือ NAM (NS-2 Website, 2004)



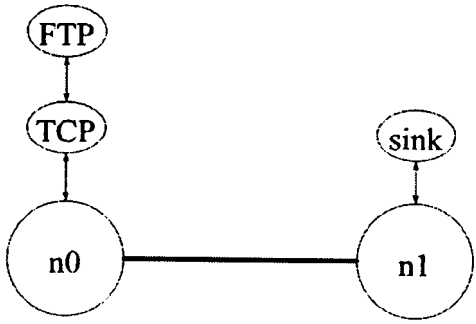
รูปที่ 3.1 แสดงลักษณะการใช้งานโปรแกรม ns

(ที่มา: Chung and Claypool, 2004)

```
1. set ns [new Simulator] //กำหนดโปรเซสการทำงาน
2. set f [open out.tr w] // กำหนดไฟล์สำหรับผลลัพธ์
3. $ns trace-all $f //กำหนดให้ผลลัพธ์เข้าสู่ไฟล์ f
4. set nf [open out.nam w] //บรรทัดที่ 4-5 กำหนดไฟล์สำหรับแสดงภาพ
5. $ns namtrace-all $nf    //เคลื่อนไหวของ NAM
6. set n0 [$ns node] //กำหนดโหนด n0
7. set n1 [$ns node] //กำหนดโหนด n1
8. $ns duplex-link $n0 $n2 5Mb 2ms DropTail //กำหนดการเชื่อมต่อระหว่างโหนด
9. set tcp [new Agent/TCP] //กำหนดค่าตัวแทนของ TCP
10. $ns attach-agent $n0 $tcp //ให้ TCP อยู่กับโหนด n1
11. set ftp [new Application/FTP] //กำหนดตัวแทนของ FTP
12. $ftp attach-agent $tcp //ให้ FTP เป็นโปรแกรมประยุค์ที่อยู่เหนือ TCPในบรรทัดที่ 9
13. set sink [new Agent/TCPSink] //กำหนดผู้ร้องขอการถ่ายโอนข้อมูล
14. $ns attach-agent $n1 $sink //ให้ผู้ร้องขออยู่บนโหนด n1
15. $ns connect $tcp $sink //เชื่อมต่อ tcp เข้ากับ sink
16. $ns at 1.2 "$ftp start" //การจำลองของ FTP ที่ simulation time = 1.2
```

รูปที่ 3.2 แสดงส่วนของโปรแกรม OTcl สคริปต์สาธิตการใช้ ns ในภาคปฏิบัติ

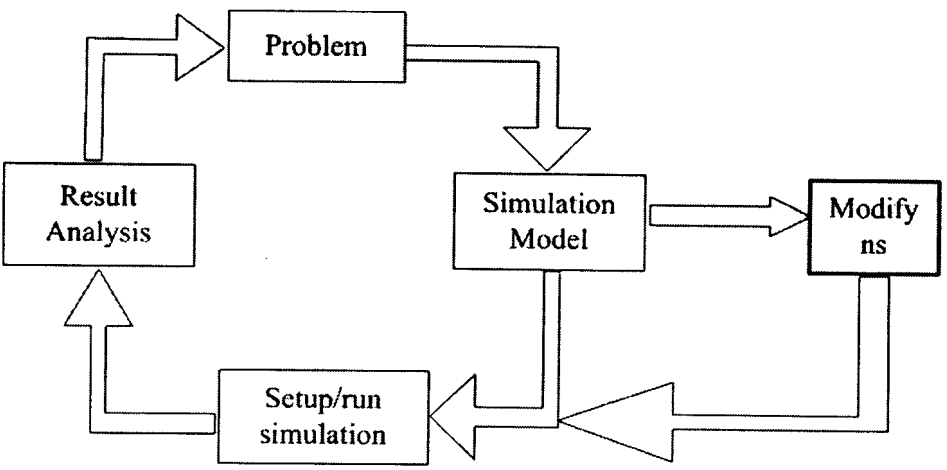
(ที่มา: Fall and Varadhan, 2002)



รูปที่ 3.3 แสดงผลทางกายภาพของส่วนของโปรแกรมในรูปที่ 3.2

ส่วนของโปรแกรมดังรูปที่ 3.2 จะเป็นตัวอย่างของการทดลองที่เกี่ยวกับ File Transfer Protocol (FTP) เพื่อสาธิตวิธีการใช้งาน ns ในทางปฏิบัติ ทั้งนี้จะสังเกตว่าผู้ใช้สามารถกำหนดค่าการทดลองจาก OTcl สคริปต์แต่ระบบจะมีการเชื่อมโยงไปยังโมดูลการทำงานจริงให้เอง ซึ่งการกำหนดค่าดังกล่าวจะได้ระบบการจำลองแสดงดังรูป 3.3

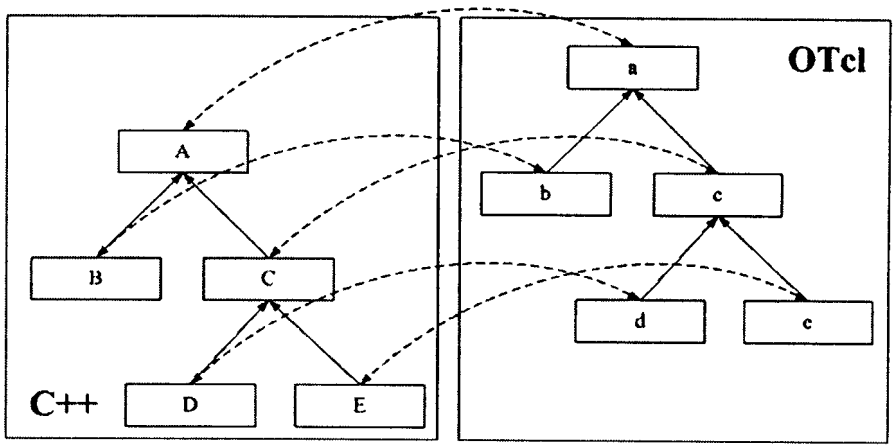
ในกรณีของการจำลองเรื่อง FTP นั้น ns ได้จัดเตรียมแบบจำลองไว้ให้แล้ว จึงสามารถทำการจำลองได้เลย แต่หากการจำลองของโพรโทคอลใดที่ ns ยังไม่ได้เตรียมแบบจำลองไว้ให้ นักพัฒนาก็จำเป็นที่จะต้องสร้างหรือปรับแต่ง ns ก่อนจึงจะทำการจำลองได้ ดังแสดงในรูปที่ 3.4 ซึ่งได้แสดงถึงวัฏจักรของการจำลองด้วย ns โดยทั่วไปโมดูลที่แทนแบบจำลองของโพรโทคอลใน ns จะต้องสร้างด้วยภาษา C++ เนื่องจากต้องการความเร็วในการประมวลผล แต่ในส่วนของการกำหนดสภาวะสำหรับการจำลองจะเขียนด้วย OTcl สคริปต์เพราะสามารถเขียนโปรแกรมได้เร็วกว่า (Chung and Claypool, 2004)



รูปที่ 3.4 แสดงวัฏจักรของการจำลองด้วย ns

(ที่มา: Chen and Haldar, 2002)

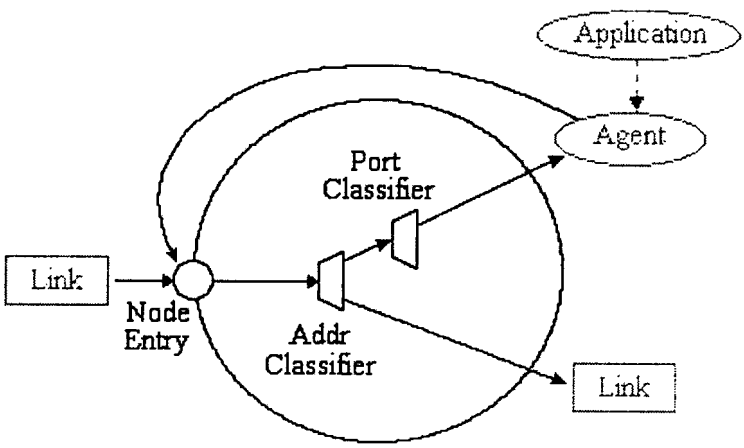
ดังนั้นโมดูลการทำงานของ ns สามารถสร้างขึ้นจากภาษา C++ โดยผ่านการเชื่อมโยงกับ OTcl เพื่อให้ผู้ใช้สามารถเรียกใช้งานได้ผ่านทางสคริปต์ OTcl ดังแสดงในรูปที่ 3.5 สำหรับส่วนการทำงานเชื่อมโยงภาษา C++ เข้ากับ OTcl นั้นเรียกว่า OTcl Linkage ซึ่งจะอธิบายในหัวข้อที่ 3.3



รูปที่ 3.5 แสดงการเชื่อมโยงระหว่างโมดูลภาษา C++ กับสคริปต์ OTcl

(ที่มา: Chung and Claypool, 2004)

3.2 โครงสร้างของโหนดและลิงค์



รูปที่ 3.6 แสดงโครงสร้างของโหนด

(ที่มา: Chung and Claypool, 2004)

องค์ประกอบหลักของระบบเครือข่ายคือ โหนดและลิงค์ โดยโหนดคือจุดที่มีการประมวลผลอาจเป็นเครื่องคอมพิวเตอร์ เราเตอร์ ฯลฯ และลิงค์คือสื่อสัญญาณที่จะนำพาข้อมูลจาก

ต้นทางไปสู่ปลายทาง ดังนั้นใน ns จึงได้สร้างโหนดและลิงค์ไว้และจัดเป็นโครงสร้างที่สำคัญสำหรับระบบการจำลองด้วย ns

โครงสร้างของโหนดแสดงดังรูป 3.6 โดยแต่ละโหนดเมื่อถูกสร้างขึ้นมาขณะทำการจำลองจะได้รับหมายเลขประจำโหนดที่แตกต่างกัน ซึ่งภายในประกอบด้วย

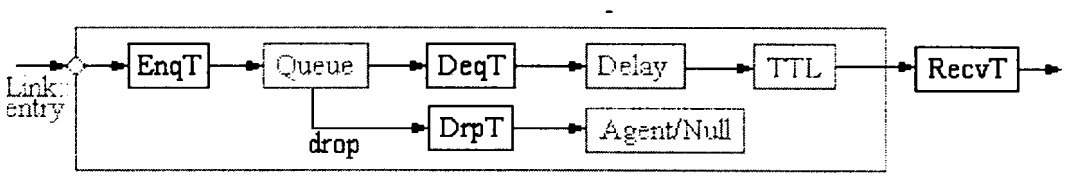
- Address Classifier ใช้สำหรับการตรวจสอบหมายเลขประจำโหนด เพื่อบ่งบอกความเป็นเจ้าของแพ็กเกต
- Port Classifier ใช้สำหรับตรวจสอบหมายเลขพอร์ตของแพ็กเกตที่เข้ามา
- ส่วนที่ใช้สำหรับการเชื่อมต่อกับโมดูลอื่นๆ เช่น ลิงค์ หรือ เอเจนต์ (ซึ่งเป็นตัวแทนของชั้นทรานสปอร์ต) การสร้างคลาสให้ทำงานเชื่อมต่อกันได้จะอธิบายในหัวข้อที่ 3.3

ส่วนต่อไปคือส่วนของลิงค์ซึ่งจะเป็นตัวแทนของสื่อสัญญาณต่างๆ โดยจะพิจารณาเรื่องความหน่วงแพร่กระจาย (Propagation Delay) และความหน่วงการส่ง (Transmission Delay) โดยความหน่วงการส่งอาจมองให้อยู่ในรูปแบนด์วิดท์ก็ได้ ดังสมการ

ความหน่วงการส่ง = ขนาดแพ็กเกต (บิต)/แบนด์วิดท์(บิตต่อวินาที

ดังนั้นจึงได้กำหนดให้ลิงค์มีองค์ประกอบดังนี้

- Queue เป็นจุดพักของข้อมูลก่อนจะถูกนำส่งไปยังโหนดต่อไป
- Delay เป็นหน่วยทำงานสำหรับสร้างความหน่วงแพร่กระจาย
- TTL เป็นส่วนที่ทำหน้าที่เกี่ยวกับตรวจสอบและเพิ่มค่า TTL
- Agent/Null เป็นส่วนรองรับแพ็กเกตที่ถูกคัดทิ้ง



รูปที่ 3.7 แสดงโครงสร้างของลิงค์

(ที่มา: Chung and Claypool, 2004)

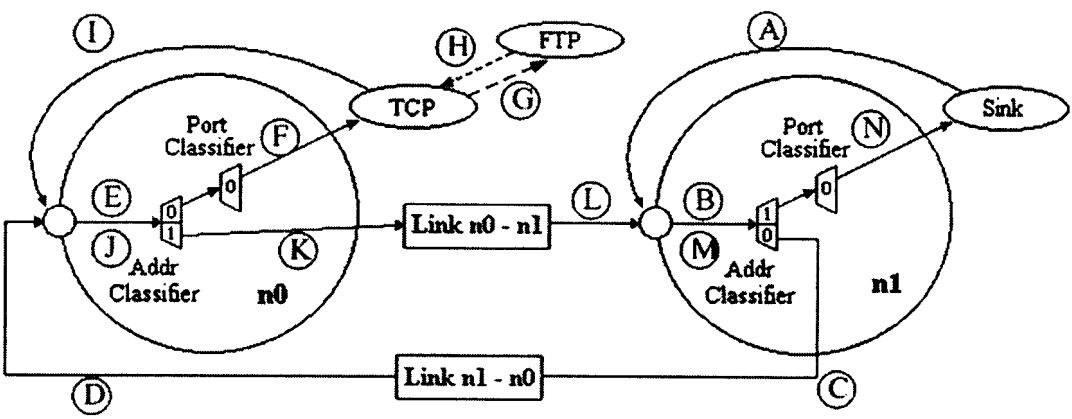
โดยในรูป 3.7 นั้นได้แสดงโครงสร้างของลิงค์ชนิด Simplex ซึ่งจะเห็นว่ารองรับการเคลื่อนที่ของข้อมูลในทิศทางเดียว อย่างไรก็ตาม ns ยังได้รองรับการทำงานของลิงค์แบบสองทางซึ่งเรียกว่า ลิงค์ชนิด Duplex ซึ่งก็คือการนำเอาลิงค์ชนิด Simplex 2 อ็อบเจกต์มาทำงานร่วมกัน

นอกจากนี้คลาสของลิงค์ยังมีคลาสย่อยที่ใช้สำหรับการบันทึกการทำงานของระบบ เช่น การบันทึกเวลาการมาถึงของแพ็กเกต เวลาที่แพ็กเกตถูกคัดทิ้ง โดยจะอยู่ในรูปแบบมาตรฐานที่กำหนดไว้แล้ว การทำงานส่วนนี้มีความสำคัญกับระบบการจำลองเพราะคือส่วนที่จะรายงานผลลัพธ์การจำลองของระบบออกมาให้ผู้ใช้ทราบนั่นเอง

โดยมีองค์ประกอบดังนี้

- EnqT สำหรับบันทึกข้อมูลแพ็กเกตที่มาถึงและได้รับการเข้าคิว
- DeqT สำหรับบันทึกข้อมูลแพ็กเกตที่ออกจากคิว
- DrpT สำหรับบันทึกข้อมูลแพ็กเกตที่ถูกคัดทิ้ง
- RecvT สำหรับบันทึกข้อมูลแพ็กเกตที่จะต้องจัดส่งให้กับโหนดต่อไป

ต่อไปจะแสดงขั้นตอนการทำงานเมื่อนำโหนด และลิงค์ มาทำงานร่วมกัน ซึ่งแสดงในรูปที่ 3.8 ซึ่งเป็นผลจากส่วนของโปรแกรม OTcl ได้แสดงไว้ในรูปที่ 3.2



รูปที่ 3.8 แสดงกลไกการทำงานของระบบการจำลอง FTP

(ที่มา: Chung and Claypool, 2004)

โดยเหตุการณ์ที่เกิดขึ้นในระบบอธิบายได้ดังต่อไปนี้

เหตุการณ์ A Sink ส่งข้อมูลร้องขอไฟล์ข้อมูลของ FTP ที่โหนด n0 ข้อมูลการต้องขอหากมีขนาด

เกินกว่า 1 แพ็กเกต ก็จะต้องถูกแบ่ง(fragment) ออกเป็นแพ็กเกตย่อยๆก่อน

เหตุการณ์ B ข้อมูลการร้องขอขณะนี้เข้าสู่โหนด n1 ซึ่ง Address Classifier จะพิจารณาว่าจะจัดส่งแพ็กเกตออกทางลิงค์ใด ซึ่ง Address Classifier จะใช้หมายเลขประจำโหนดเป็นเกณฑ์ในการจัดส่งแพ็กเกต และเนื่องจากแพ็กเกตนี้คือข้อมูลการร้องขอไฟล์ข้อมูลไปยังโหนด n0 แพ็กเกตนี้จึงถูกจัดส่งไปยัง Address Classifier หมายเลข 0

เหตุการณ์ C ข้อมูลถูกจัดส่งไปยังลิงค์ หากขณะนี้คิวว่างลิงค์ก็จะจัดส่งข้อมูลออกเข้าสู่หน่วย Delay ทันที แต่ถ้าพบว่าคิวไม่ว่างจะเข้าคิวไว้ก่อน (การเข้าคิวจัดการโดยหน่วย Queue)

เหตุการณ์ D ลิงค์จัดส่งแพ็กเกตออกจากคิว แล้วเข้าสู่หน่วย Delay ซึ่งจะทำหน้าที่คำนวณความหน่วงแพร่กระจาย

เหตุการณ์ E ลิงค์จัดส่งข้อมูลให้กับโหนด n0 ได้อย่างสมบูรณ์ หมายความว่าขณะนี้ทุกบิตของแพ็กเกตถึงโหนด n0 แล้ว และที่โหนด n0 จะทำการ Address Classify ซึ่งพบว่าเป็นแพ็กเกตที่ส่งถึงตนเองจึงรับแพ็กเกตนี้ไว้ และเข้าสู่ Port Classifier ซึ่งพบว่าแพ็กเกตนี้เป็นของ TCP

เหตุการณ์ F จัดส่งข้อมูลให้กับเอเจนต์ ซึ่งขณะนี้คือ TCP

เหตุการณ์ G TCP รับแพ็กเกต จนกระทั่งพบว่าแพ็กเกตที่รับเข้ามาเป็นกลุ่มข้อมูลที่สมบูรณ์แล้วจึงจัดส่งให้ FTP ทั้งนี้เพราะว่าหากข้อมูลการร้องขอใหญ่กว่า 1 แพ็กเกต แพ็กเกตที่จัดส่งจากต้นทางที่แสดงถึงการร้องขอไฟล์ข้อมูลอาจมีมากกว่า 1 แพ็กเกตก็เป็นได้ ซึ่งเป็นหน้าที่ของเอเจนต์ในชั้นทรานสปอร์ตจะต้องแบ่งกลุ่มและรวบรวม (Reassemble) แพ็กเกตกลับมาเป็นกลุ่มข้อมูลการร้องขอ

เหตุการณ์ H เมื่อ FTP รับข้อมูลการร้องขอ จึงทำการส่งข้อมูลไฟล์ที่ร้องขอให้กลับไปยังโหนด n1

เหตุการณ์ I TCP รับข้อมูลจาก FTP และทำการแบ่งกลุ่มข้อมูลเพื่อการส่ง และจัดส่งข้อมูลให้กับโหนด n0

เหตุการณ์ J แพ็กเกตเข้าสู่ Address Classifier ซึ่งโหนด n0 พบว่าไม่ใช่ข้อมูลของตนเอง และจะต้องจัดส่งให้กับโหนด n1

เหตุการณ์ K แพ็กเกตถูกจัดส่งไปยังลิงค์ หากขณะนี้คิวว่างลิงค์ก็จะจัดส่งข้อมูลออกเข้าสู่หน่วย Delay ทันที แต่ถ้าพบว่าคิวไม่ว่างจะเข้าคิวไว้ก่อน

เหตุการณ์ L ลิงค์นำแพ็กเกตออกจากคิว แล้วเข้าสู่หน่วย Delay ซึ่งจะทำหน้าที่คำนวณความหน่วงแพร่กระจายของแพ็กเกต

เหตุการณ์ M แพ็กเกตมาถึงโหนด n1 และพบว่าเป็นแพ็กเกตที่มีปลายทางเป็นโหนด n1

เหตุการณ์ N Sink ได้รับแพ็กเกต

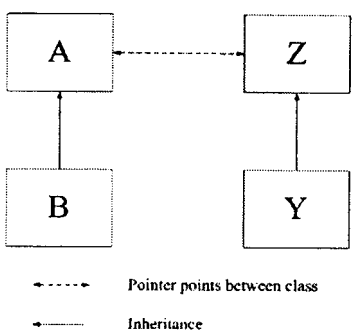
จากเหตุการณ์ตั้งแต่ A ถึง N เป็นการสรุปขั้นตอนการรับส่งข้อมูลที่เกิดขึ้น สำหรับรายละเอียดการจัดส่งข้อมูลจริงจะขึ้นอยู่กับโพรโทคอลที่ใช้ เช่น หากเป็น UDP ก็จะมีการจัดส่งและรับข้อมูลโดยไม่มีการส่งข้อมูลซ้ำ (Retransmission) แต่หากเป็น TCP ก็จะมีการส่งข้อมูลและรับข้อความตอบรับ (Acknowledgement) หากข้อความตอบรับไม่กลับมาในเวลากำหนดก็จะส่งข้อมูลซ้ำให้เพื่อเป็นการรับประกันความถูกต้องของการส่งข้อมูล ทั้งนี้รายละเอียดการจำลองของแต่และโพรโทคอลจะขึ้นอยู่กับข้อกำหนดที่แตกต่างกันออกไป

3.3 คลาสต้นแบบที่สำคัญ

ในส่วนนี้จะได้กล่าวถึงคลาสต้นแบบที่สำคัญกับวิทยานิพนธ์นี้ ซึ่งประกอบด้วย คลาส Application, TCP Application และ คลาส Random Variable ซึ่งในการวิจัยคลาสของ HLA จะต้องเข้าไปเกี่ยวพันด้วย นอกจากนี้ประเด็นสำคัญที่จะละเว้นเสียไม่ได้ 2 ประการ คือเรื่อง OTcl linkage ซึ่งจะเชื่อมโยงคลาสที่สร้างด้วย C++ เข้ากับ OTcl สคริปต์และเรื่องการสร้างความสัมพันธ์ระหว่างโมดูลโดยอาศัยคุณสมบัติภาวะหลายรูปแบบ (Polymorphism) ซึ่งเป็นพื้นฐานที่จะทำให้เข้าใจปฏิสัมพันธ์ที่เป็นระบบระหว่างคลาสต่างๆภายใน ns

3.3.1 ภาวะหลายรูปแบบ

ภาวะหลายรูปแบบหรือ Polymorphism เป็นเอกลักษณ์อันหนึ่งของการโปรแกรมเชิงอ็อบเจกต์ซึ่งทำให้เกิดความยืดหยุ่นในการสร้างและเพิ่มขยายซอฟต์แวร์ และใน ns ก็ได้ใช้แนวคิดดังกล่าวเพื่อสร้างระบบจำลองขึ้นมาโดยผ่านทางภาษา C++ ดังนั้นในเนื้อหาส่วนนี้จะได้กล่าวถึงแนวคิดดังกล่าว



รูปที่ 3.9 แสดงความสัมพันธ์ระหว่างคลาส A, B, Z และ Y

```
1. Class A {
2. public:
3.    A();
4.    virtual void AttachZ(Z*);
5.    virtual void Asend_msg(int);
6.    virtual void Arecv_msg(int);
7. protected:
8.    Z* zebra;
9. };
```

```
1. Class Z {
2. public:
3.    Z();
4.    void AttachA(A*);
5.    void Zsend_msg(int);
6.    void Zrecv_msg(int);
7. protected:
8.    A* ant;
9. };
```

รูปที่ 3.10 แสดงคลาสนามธรรม A และ B

โดยจะยกตัวอย่างความสัมพันธ์ของคลาสกลุ่มหนึ่ง ดังรูป 3.9 ทั้งนี้คลาส A และ Z จะเป็นคลาสนามธรรม(แสดงส่วนการทำงานในรูป 3.10) ซึ่งจะใช้เป็นโครงร่างสำหรับการเชื่อมต่อ โดยมีการประกาศที่สำคัญคือในบรรทัดที่ 8 สำหรับคลาส A ได้ประกาศตัวแปรพอยน์เตอร์ชื่อ zebra ที่ชี้ไปยังอ็อบเจกต์ของคลาส Z และในบรรทัดเดียวกันนี้ของคลาส Z ก็ได้ประกาศตัวแปรพอยน์เตอร์ชื่อ ant ที่ชี้ไปยังอ็อบเจกต์ของคลาส A แต่เนื่องจากคลาสทั้ง A และ Z เป็นคลาสนามธรรมจึงทำให้ไม่มีคำสั่งที่ทำให้พอยน์เตอร์ ant และ zebra ชี้ไปยังเป้าหมายได้ แต่ในรูปที่ 3.9 ได้แสดงลูกศรเส้นประสองทางซึ่งเชื่อม A และ Z เข้าด้วยกัน นั้นให้ความหมายว่าคลาส A และ Z ได้จัดเตรียมส่วนสำหรับการเชื่อมต่อเอาไว้แล้ว

```
1. Class B : public A {
2. public:
3.    B();
4.    virtual void AttachZ(Z*);
5.    virtual void Asend_msg(int);
6.    virtual void Arecv_msg(int);
7. };
```

```
1. Class Y : public Z {
2. public:
3.    Y();
4.    virtual void AttachA(A*);
5.    virtual void Zsend_msg(int);
6.    virtual void Zrecv_msg(int);
7. };
```

รูปที่ 3.11 แสดงการรับทอดของคลาส B จากคลาส A และ คลาส Y จากคลาส Z

```
1. void B::AttachZ(Z* target){
2.     zebra = target;
3. }
4.  void B::Asend_msg(int msg){
5.      zebra->Zrecv_msg(msg);
6.  }
7.  void B::Arecv_msg(int msg){
8.     printf("B hear:%d",msg);
9.  }
```

```
1. void Y::AttachA(A* target){
2.     zebra = target;
3. }
4.  void Y::Zsend_msg(int msg){
5.      ant->Arecv_msg(msg);
6.  }
7.  void Y::Zrecv_msg(int msg){
8.     printf("Y hear:%d",msg);
9.  }
```

รูปที่ 3.12 แสดงการเขียนโปรแกรมลงในส่วน Virtual Function ของคลาส B และ Y

สำหรับคลาสนามธรรม ส่วนการทำงานจริงจะไปอยู่ยังคลาสลูก ในที่นี้คือ คลาส B และ Y (แสดงส่วนคำสั่งสำหรับการรับทอดดังรูป 3.11) ทั้งสองได้รับทอดคุณสมบัติมาจากคลาส A และ Z ตามลำดับ ดังในรูปที่ 3.12 บรรทัดที่ 1-3 คือส่วนฟังก์ชัน AttachZ(Z*) ที่จะทำให้อ็อบเจกต์ของคลาส B ชี้ไปยังอ็อบเจกต์ของคลาส Y ได้ โดยการทำให้พอยน์เตอร์ zebra ชี้ไปยังที่เดียวกับ target

```
1. B b1;
2. Y y1;


3. b1.AttachZ(&zebra);
4. y1.AttachA(&ant);


5. b1->Asend_msg(5);
6. y1->Zsend_msg(10);
```

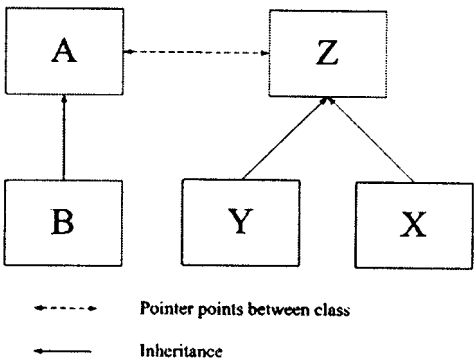
ผลลัพธ์

```
5. Y hear:5
6. B hear:10
```

รูปที่ 3.13 แสดงการสร้างอ็อบเจกต์ของคลาส B และ Y ซึ่งทั้งคู่ต่างเรียกใช้ซึ่งกันและกัน

ในรูปที่ 3.13 b1 และ y1 เป็นอ็อบเจกต์ของ B และ Y บรรทัดที่ 3-4 จะทำการเชื่อมอ็อบเจกต์ทั้งสองเข้าหากัน และบรรทัดที่ 5 b1 ได้ส่งข้อความถึง y1 จึงทำให้เกิดผลลัพธ์คือ Y ได้รับข้อความดังกล่าว ส่วนบรรทัดที่ 6 y1 ก็สามารถส่งข้อความถึง b1 ได้เช่นกัน

ต่อไปจะแสดงการเพิ่มคลาส X เข้าสู่ระบบโดยให้เป็นคลาสลูกของคลาส Z ดังแสดงในรูป 3.14 สำหรับส่วนของโปรแกรมของคลาส X แสดงในรูปที่ 3.15



รูปที่ 3.14 แสดงการเพิ่มเติมคลาส X สู่ระบบ

```
1. void X::AttachA(A* target){
2.     zebra = target;
3. }
4.  void X::Zsend_msg(int msg){
5.      ant->Arecv_msg(msg);
6.  }
7.  void X::Zrecv_msg(int msg){
8.     printf("X hear:%d",msg);
9.  }
```

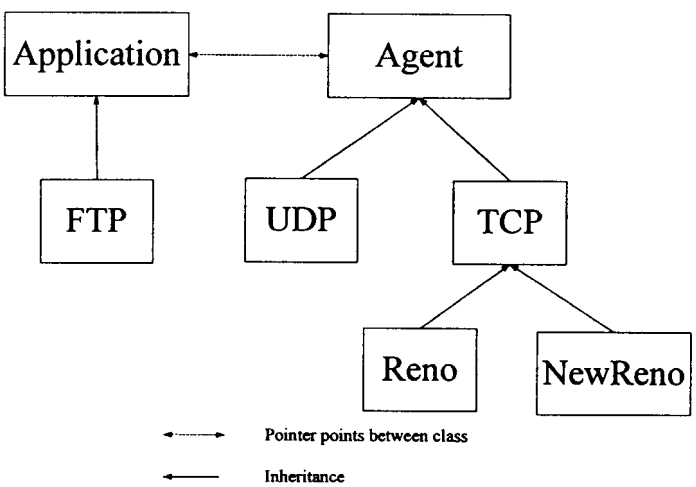
รูปที่ 3.15 แสดงการเขียนโปรแกรมลงในส่วน Virtual Function ของคลาส X

ส่วนของโปรแกรมหลักแสดงดังรูปที่ 3.16 ซึ่งแสดงความสามารถในการเชื่อมต่อถึงกันของอ็อบเจกต์ b1 และ x1 (บรรทัดที่ 3-4) และที่สำคัญในบรรทัดที่ 5 ยังมีการส่งข้อความถึงกัน และได้แสดงผลลัพธ์คือ X hear:5 ซึ่งแตกต่างจากผลลัพธ์เมื่อ b1 เชื่อมต่อกับ y1 แสดงให้เห็นว่าการเพิ่มคลาส X ที่เป็นคลาสใหม่เข้าสู่ระบบ เราไม่ต้องทำการเปลี่ยนแปลงส่วนเชื่อมต่อหรือฟังก์ชันของ

คลาส B แต่คลาส Y และ X จะจัดการกับการเรียกใช้และข้อมูลที่เข้ามาเอง หรืออาจมองในอีกแง่หนึ่งว่า แท้จริงอ็อบเจกต์ b1 นั้นสามารถคุยกับทุกๆอ็อบเจกต์ที่เป็นลูกหลานของคลาส Z ได้ทุกๆ อ็อบเจกต์เนื่องจาก b1 มีพอยน์เตอร์ zebra ซึ่งชี้ไปยังคลาส Z

1. B b1; 2. X x1; 3. b1.AttachZ(&x1); 4. x1.AttachA(&b1); 5. b1->Asend_msg(5); 6. x1->Zsend_msg(10);	ผลลัพธ์ 5. X hear:5 6. B hear:10

รูปที่ 3.16 แสดงการสร้างอ็อบเจกต์ของคลาส B และ X และมีการเรียกใช้ซึ่งกันและกัน



รูปที่ 3.17 แสดงโครงสร้างการเชื่อมต่อของ Application และ Agent

จากรูปที่ 3.14 และรูปที่ 3.17 พอที่จะเทียบเคียงโครงสร้างการเชื่อมต่อกันได้ โดยในรูป 3.17 นั้นได้แสดงโครงสร้างส่วนหนึ่งของ ns ซึ่งจะเห็นว่า FTP สามารถที่จะเชื่อมต่อกับ TCP Reno หรือ TCP New Reno ได้ โดย FTP จะมองเห็นโพรโทคอลที่เพิ่มเติมเข้ามาเป็นรูปหนึ่งของ Agent

ทั้งหมดที่กล่าวมาเกี่ยวกับเรื่องภาวะหลายรูปแบบนั้น ns ได้นำแนวทางดังกล่าวมาประยุกต์ใช้โดยตลอด โดยมีประโยชน์สำคัญคือ ทำให้การขยายระบบทำได้อย่างยืดหยุ่น

ในวิทยานิพนธ์นี้ซึ่งจะต้องสร้างโพรโทคอลในระดับชั้นประยุกต์ ซึ่งจะต้องรับทอดคุณสมบัติจากคลาส Application ดังนั้นจึงมั่นใจได้ว่าโมดูลที่จะสร้างขึ้นจะสามารถเรียกใช้ความสามารถของ ns ได้อย่างครบถ้วน เช่น การรับประกันความถูกต้องของการส่งผ่านข้อมูลในส่วนที่เกี่ยวกับ Time Management ซึ่งจะต้องทำการเชื่อมต่อ TCP ก็ย่อมสามารถทำได้ และส่วนของโมดูล Object Management และ Data Distribution Management ซึ่งจะพึ่งพาโพรโทคอล UDP ก็ใช้ได้อย่างไม่มีข้อยกเว้นเช่นกัน

สำหรับรายละเอียดที่เกี่ยวข้องกับโมดูล Object Management, Data Distribution Management และ Time Management จะกล่าวถึงต่อไปในบทที่4 ซึ่งว่าด้วยเรื่องการออกแบบและสร้าง สำหรับส่วนที่เหลือของบทนี้จะกล่าวถึงการทำงานของคลาส Application, Tcp Application และ Random Variable แต่สำหรับในหัวข้อที่ 3.3.2 จะกล่าวถึงวิธีการที่ ns ใช้สำหรับการประกาศชุดคำสั่งในคลาสภาษา C++ ให้สามารถเรียกใช้ได้ผ่านทาง OTcl สคริปต์

3.3.2 OTcl linkage

ดังที่ได้กล่าวถึงไว้ในตอนต้นดังแสดงในรูปที่ 3.5 ซึ่งได้แสดงให้เห็นถึงความสามารถของ ns ที่สามารถเชื่อมโยงส่วนการทำงานทั้งที่อยู่ในภาษา C++ หรือ OTcl เข้าด้วยกัน ทำให้ผู้ใช้สามารถเรียกใช้ส่วนการทำงานที่เป็นภาษา C++ ผ่าน OTcl สคริปต์หรือเรียก OTcl สคริปต์ผ่านทาง C++ ก็ได้ แต่โดยทั่วไปโมดูลที่มีการประมวลผลระดับหน่วยข้อมูล เช่น การคำนวณค่า Round Trip Time (RTT) หรือการพิจารณาค่า TTL เพื่อจะคัดแพ็กเกตทิ้ง สิ่งเหล่านี้จะเขียนด้วยภาษา C++ และกำหนดให้ผู้ใช้เรียกใช้ได้ผ่านทาง OTcl สคริปต์ได้ โดยอาศัยการทำงานของ OTcl linkage

```
1. class Fed : public Application {
2. public:
3.         Fed();
4. protected:
5.         int command(int argc, const char*const* argv);
6. private:
7.         virtual void   AttachSimObj(void);
8. };
```

รูปที่ 3.18 แสดงการประกาศคลาส Fed สำหรับส่วนการทำงานจริง

จากรูปที่ 3.18 เป็นตัวอย่างแสดงการสร้างคลาส Fed ซึ่งในที่นี้ไม่ได้แสดงให้เห็นรายละเอียดภายในเพียงแต่จะแสดงให้เห็นความเชื่อมโยงระหว่างคลาสภาษา C++ กับส่วนของ OTcl linkage เท่านั้น การสร้างคลาส Fed จะทำการรับทอดจาก Application และจะทำงานในชั้นโปรแกรมประยุกต์ แต่สำหรับผู้ใช้สามารถเรียกใช้คลาส Fed ผ่านทาง OTcl สคริปต์ได้ด้วยคำสั่ง set fed1 [new Application/Fed] ซึ่งเป็นการสร้างตัวแปร fed1 ขึ้นในพื้นที่ของ OTcl ซึ่งจะชี้ไปยังพื้นที่ของ C++ อีกทอดหนึ่ง การที่ ns รู้จักคำสั่ง [new Application/Fed] เป็นผลจากโปรแกรมบรรทัดที่ 3 รูป 3.19 และเมื่อทำคำสั่ง set fed1 [new Application/Fed] OTcl linkage ก็จะเรียกฟังชัน create ซึ่งจะจบด้วยการสร้างคลาส Fed ขึ้นด้วยคำสั่ง new Fed() ดังแสดงในบรรทัดที่ 5 ซึ่งจะคืนค่าพอยน์เตอร์ที่ชี้ไปยังอ็อบเจกต์ของคลาส Fed ให้กับตัวแปร fed1 ในพื้นที่ของ OTcl

```
1. static class FedClass : public TclClass {
2. public:
3.         FedClass() : TclClass("Application/Fed ") {}
4.         TclObject* create(int, const char*const*) {
5.                 return(new Fed());
6.         }
7. } class_Fed;
```

รูปที่ 3.19 แสดงการสร้าง FedClass เพื่อประกาศคลาส Fed ในพื้นที่ OTcl

เมื่อเกิดตัวแปร fed1 ขึ้นแล้วก็สามารถเรียกใช้ส่วนการทำงานที่เกี่ยวข้องคลาส Fed ได้ดังแสดงในรูปที่ 3.20 โดยการตรวจสอบค่าในฟังก์ชัน command ดังแสดงในบรรทัดที่ 1 เช่นเมื่อใช้คำสั่ง $fed1 attach-sim-obj $obj1 ระบบก็จะเข้าทำฟังก์ชัน command โดยสามารถเข้าถึงบรรทัดที่ 3-5 ได้ แต่หากเป็นคำสั่งอื่นๆ จะไม่สามารถหาพบเนื่องจากในที่นี้ Fed::command ได้นิยามคำสั่งไว้เพียงคำสั่งเดียวคือ attach-sim-obj แต่ OTcl ก็จะพยายามหาในส่วนต่อไปคือในส่วนฟังก์ชัน command ของคลาส Application ซึ่งเป็นคลาสแม่ด้วยคำสั่งในบรรทัดที่ 8

```
1. int Fed::command(int argc, const char*const* argv) {
2.     if(argc == 2) {
3.         if(strcmp(argv[1], "attach-sim-obj") == 0) {
4.             AttachSimObj();
5.             return(TCL_OK);
6.         }
7.     }
8.     return(Application::command(argc, argv));
9. }
```

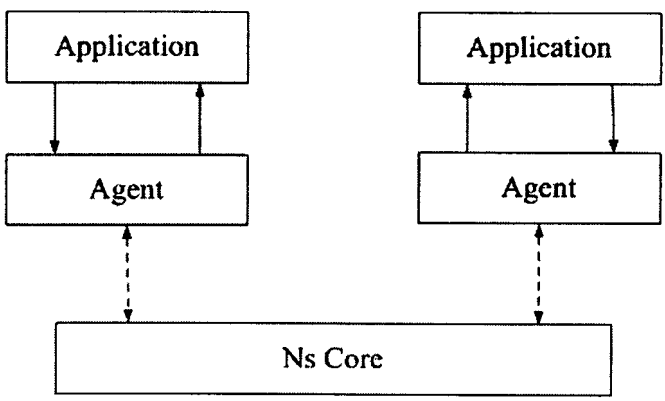
รูปที่ 3.20 แสดงส่วนของการแปลคำสั่งสำหรับคลาส Fed

ดังนั้นโดยสรุป OTcl linkage จะเป็นการสร้างการเชื่อมต่อระหว่างพื้นที่ OTcl เข้ากับโมดูลการทำงานที่เขียนขึ้นด้วย C++ ทำให้ผู้ใช้สามารถกำหนดค่าการทดลองด้วย OTcl สคริปต์ได้

3.3.3 คลาส Application และ Tcp Application

ในส่วนนี้จะกล่าวถึงการทำงานของคลาส Application รวมทั้งจะกล่าวถึงคลาส Tcp Application ไปด้วยในคราวเดียวกันเนื่องจาก Tcp Application นั้นเป็นคลาสที่รับทอดคุณสมบัติมาจากคลาส Application และได้เพิ่มเติมความสามารถให้สามารถขนถ่ายข้อมูลจริงได้

สำหรับในวิทยานิพนธ์นี้จะเน้นการวิจัยในส่วนที่เกี่ยวข้องกับ HLA ซึ่งเป็นอาจมองเป็นโพรโทคอลชั้นโปรแกรมประยุกต์เป็นหลัก แม้ว่าจะต้องเกี่ยวพันกับชั้นทรานสปอร์ต แต่ก็เป็นไปเพื่อการเรียกใช้งานเท่านั้น ดังนั้นมุมมองของผู้วิจัยจึงมองภาพส่วนอื่นๆเป็นนามธรรม การพรรณนาถึงโครงสร้างของคลาสต่างๆ จึงเกี่ยวข้องอยู่กับคลาสที่เกี่ยวกับโปรแกรมประยุกต์ ซึ่งในที่นี้คือคลาส Application และ Tcp Application อาจจะมีการพาดพิงถึงคลาส Agent ซึ่งเป็นตัวแทนของชั้นทรานสปอร์ตบ้าง แต่ก็เป็นไปเพื่อให้เกิดความกระจ่างชัดมากขึ้นเท่านั้น ดังแสดงในรูปที่ 3.21 ซึ่งจะเน้นตรงส่วน Application และ Agent และมองส่วนอื่นๆ เป็น Ns Core ซึ่งจะนำส่งข้อมูลไปยังเป้าหมายได้



รูปที่ 3.21 แสดงโครงร่างของโมดูลที่เกี่ยวเนื่องกับวิทยานิพนธ์นี้

ในส่วนของคลาส Application นั้นได้รับทอดคุณสมบัติมาจากคลาส Process โดยคลาส Application จะเป็นตัวแทนของโพรโทคอลชั้นโปรแกรมประยุกต์ ดังได้แสดงโครงสร้างทั้งหมดไว้ในรูป 3.22 ซึ่งจะแสดงถึงลักษณะเด่นของคลาสนี้ได้ 3 ประการ คือ

- การเชื่อมต่อกับคลาส Agent เพื่อเป็นช่องทางในการรับ/ส่งข้อมูลถึงกันได้
- ฟังก์ชันที่เกี่ยวกับการส่งข้อมูล
- ฟังก์ชันที่เกี่ยวกับการรับข้อมูล

```
1. class Application : public Process {
2. public:
3.     Application();
4.     virtual void send(int nbytes);
5.     virtual void recv(int nbytes);
6.     virtual void resume();
7. protected:
8.     virtual int command(int argc, const char*const* argv);
9.     virtual void start();
10.    virtual void stop();
11.    Agent *agent_;
12.    int enableRecv_;        // call Tcl recv or not
13.    int enableResume_;      // call Tcl resume or not
14. };
```

รูปที่ 3.22 แสดงโครงสร้างของคลาส Application

สำหรับการเชื่อมต่อกับคลาส Agent เพื่อเปิดช่องทางการติดต่อนั้น อาศัยพอยน์เตอร์ชื่อ agent_ ซึ่งถูกประกาศที่บรรทัดที่ 11 สำหรับฟังก์ชันส่งและรับอยู่ที่บรรทัดที่ 4 และ 5 ตามลำดับ ขณะนี้ขอให้สังเกตว่าความสำคัญของฟังก์ชันส่ง/รับข้อมูลไม่ได้อยู่ที่ว่าจะส่ง/รับข้อมูลอย่างไร แต่ความสำคัญอยู่ที่พารามิเตอร์ของฟังก์ชันการส่ง/และรับกลับเป็นเพียงขนาดของข้อมูล ดังแสดงในบรรทัดที่ 4 ว่า virtual void send(int nbytes) นั่นแสดงให้เห็นว่าคลาส Application ไม่สนับสนุนการส่งข้อมูลจริง การส่งข้อมูลจริงสำคัญอย่างไรจะกล่าวถึงอีกครั้งในบทที่ 4 ว่าด้วยการออกแบบ

สำหรับฟังก์ชันอื่นๆ สามารถอธิบายได้ดังนี้

- virtual void start() สำหรับกำหนดการทำงานของอ็อบเจกต์
- virtual void stop() สำหรับกำหนดการยุดทำงานของอ็อบเจกต์
- virtual void resume() สำหรับกำหนดการเริ่มทำงานใหม่

อนึ่ง ในวิทยานิพนธ์นี้ ไม่ได้ใช้ตัวแปร int enableRecv_ และ int enableResume_ ในบรรทัดที่ 12-13

สำหรับรายละเอียดการทำงานของฟังก์ชันที่สำคัญ คือส่วนส่ง/รับข้อมูลแสดงดังรูป 3.23 ส่วนของการส่งแสดงในบรรทัดที่ 1-4 ซึ่งจะเป็นว่าคลาส Application อาศัยการฟังก์ชันการส่งของคลาส Agent อีกทอดหนึ่ง สำหรับบรรทัดที่ 5-11 แสดงส่วนการรับข้อมูล ซึ่งจะมีการคืนค่าให้กับ OTcl แต่ไม่มีการนำจำนวนข้อมูลที่ได้ไปประมวลผลต่อ

จะเห็นว่าคลาส Application นั้นไม่สามารถนำข้อมูลจริงไปสู่ปลายทางได้ ns จึงได้เพิ่มคลาส Tcp Application ซึ่งในการเขียนโปรแกรมจะเขียนย่อเป็น TcpApp ซึ่งรับทอดมาจากคลาส Application ให้สามารถนำส่งข้อมูลจริงได้ แสดงโครงร่างในรูปที่ 3.24 สังเกตได้จากฟังก์ชันต้นแบบในบรรทัดที่ 7 virtual void send(int nbytes, AppData *data) นั้นกำหนดให้มีพารามิเตอร์ AppData *data ซึ่งจะเป็นพอยน์เตอร์ชี้ไปยังกลุ่มข้อมูลที่จะจัดส่ง

```
1. void Application::send(int nbytes)
2. {
3.      agent_->sendmsg(nbytes);
4. }
5. void Application::recv(int nbytes)
6. {
7.      if (! enableRecv_)
8.          return;
9.      Tcl& tcl = Tcl::instance();
10.     tcl.evalf("%s recv %d", name_, nbytes);
11. }
```

รูปที่ 3.23 แสดงส่วนการทำงานของฟังก์ชัน send และ recv

เนื่องจากการทำงานของคลาส TcpApp มีความสัมพันธ์กับโครงสร้างข้อมูล CBufList ซึ่งเขียนไว้ในบรรทัดที่ 28-29 ซึ่งเป็นโครงสร้างลิงค์ลิสต์ ร่วมกันทำงานกับฟังก์ชัน send(int nbytes, AppData *data) และ CBuf* rcvr_retrieve_data() ในส่วนนี้จึงจะใช้รูปที่ 3.25 ร่วมด้วยเพื่อการอธิบายที่ชัดเจนยิ่งขึ้น ดังนี้

ขั้นที่ 1 TcpApp ส่งข้อมูลด้วยคำสั่ง send(int nbytes, AppData *data) ซึ่งต่างจากคลาส Application ที่ส่งด้วย send(int nbytes) ส่วนของขนาดกลุ่มข้อมูล nbytes จะถูกจัดส่งให้ Agent ตามปกติ

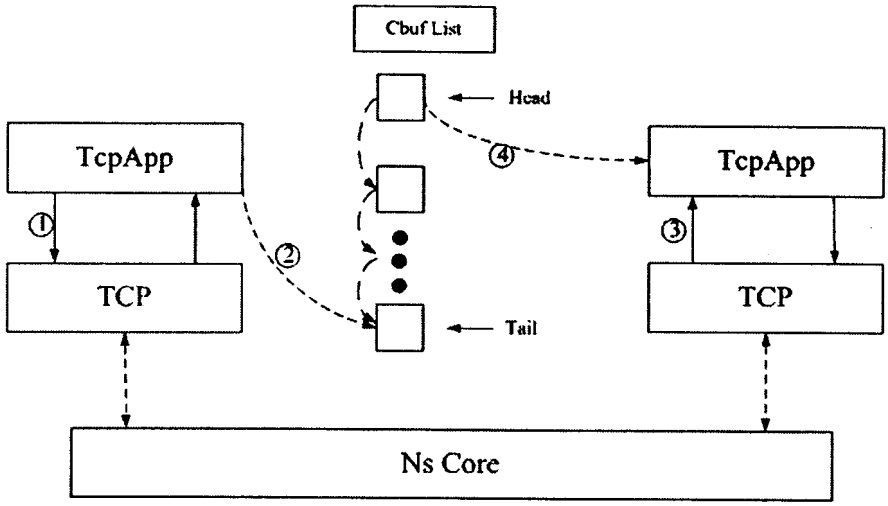
ขั้นที่ 2 สำหรับเนื้อหาของข้อมูลในที่นี้คือ AppData *data จะถูกส่งเข้า Cbuf List

ขั้นที่ 3 กลุ่มข้อมูลมาถึงเป้าหมายตามขนาดที่ระบุ ฟังก์ชัน recv(int nbytes) ถูกกระตุ้นให้รับข้อมูล

ขั้นที่ 4 นอกจาก recv(int nbytes) จะรับขนาดข้อมูลเข้ามาแล้ว ภายในยังมีการเรียกใช้ฟังก์ชัน rcvr_retrieve_data() เพื่อดึงข้อมูลออกจาก Cbuf List ซึ่งทำให้ที่ปลายทางได้รับทั้งขนาดข้อมูลขนาดข้อมูลและเนื้อหาของข้อมูลจริง

```
1. class TcpApp : public Application {
2. public:
3.     TcpApp(Agent *tcp);
4.     ~TcpApp();
5.
6.     virtual void recv(int nbytes);
7.     virtual void send(int nbytes, AppData *data);
8.
9.     void connect(TcpApp *dst) { dst_ = dst; }
10.
11.    virtual void process_data(int size, AppData* data);
12.    virtual AppData* get_data(int&, AppData*) {
13.        // Not supported
14.        abort();
15.        return NULL;
16.    }
17.    virtual void resume();
18. protected:
19.     virtual int command
20.     (int argc, const char*const*  argv);
21.     CBuf* rcvr_retrieve_data() {
22.             return cbuf_.detach();
23.     }
24.     virtual void start() { abort(); }
25.     virtual void stop() { abort(); }
26.
27.     TcpApp *dst_;
28.     CBufList cbuf_;
29.     CBuf *curdata_;
30.     int curbytes_;
31. };
```

รูปที่ 3.24 แสดงโครงร่างการทำงานของ TcpApp



รูปที่ 3.25 แสดงการทำงานของ TcpApp

นอกจากนี้เมื่อสามารถดึงเนื้อหาของข้อมูลมาได้แล้ว TcpApp ยังได้จัดเตรียมฟังก์ชัน virtual void process_data(int size, AppData* data) ไว้สำหรับประมวลผลเนื้อหาข้อมูลที่ได้รับมา สำหรับคลาสที่รับทอดจาก TcpApp สามารถเขียนโปรแกรมส่วน process_data ได้ใหม่ได้เนื่องจากฟังก์ชันได้ถูกประกาศไว้เป็น virtual สำหรับฟังก์ชัน connect(TcpApp*) ในบรรทัดที่ 9 ใช้สำหรับชี้คู่ของอ็อบเจกต์ของคลาส TcpApp ที่อยู่ที่อีกฝั่งหนึ่ง เนื่องจากทั้งสองต้องใช้ Cbuf List ร่วมกัน

นอกจากนี้ฟังก์ชัน get_data(), resume(), start(), stop() ของ TcpApp ขณะนี้ยังไม่ได้รับการเขียนโปรแกรมให้ทำงาน มีเพียงแต่การประกาศฟังก์ชันไว้เท่านั้น

3.3.4 คลาส Random Variable

ในระบบการจำลองตัวแปรสุ่ม (Random Variable) มีความสำคัญ เช่นถ้าต้องการจำลองการทอดลูกเต๋า ที่ความน่าจะเป็นของการออกแต่ละหน้าเท่ากัน ก็ต้องใช้ตัวแปรสุ่มที่ให้ผลลัพธ์ที่มีการแจกแจงแบบสม่ำเสมอ (Uniform Distribution) เป็นข้อมูลเข้าสำหรับการจำลอง สำหรับการจำลองอื่นๆ ก็จะมีลักษณะธรรมชาติของข้อมูลเข้าที่แตกต่างกันออกไป เช่น การทดสอบเรื่องคิวแบบ M/M/1 ซึ่งกำหนดให้เวลาการเข้าใช้หน่วยบริการมีการแจกแจงแบบเอ็กโปเนนเชียล (Exponential Distribution) ก็ต้องใช้ตัวแปรสุ่มที่ให้ผลลัพธ์ที่มีการแจกแจงแบบเอ็กโปเนนเชียล ดังนั้นตัวแปรสุ่ม

ที่ใช้สำหรับเป็นต้นกำเนิดข้อมูลของการจำลองจึงมีหลายรูปแบบ และการนำไปใช้ก็จะต้องมีบริบทและเงื่อนไขที่แตกต่างกันออกไป

สำหรับ ns ได้จัดเตรียมคลาสที่ทำงานเป็นตัวแปรสุ่มไว้หลายชนิด เช่น คลาส NormalRandomVariable, LogNormalRandomVariable, UniformRandomVariable และ ExponentialRandomVariable แต่คลาสเหล่านี้ทั้งหมดล้วนรับทอดคุณสมบัติมาจากคลาส RandomVariableiable ทั้งสิ้น ดังแสดงในรูปที่ 3.26 ฟังก์ชันที่สำคัญคือ

- virtual double value() ซึ่งจะทำให้ได้ผลลัพธ์การสุ่มออกมา
- virtual double avg() ซึ่งจะทำให้ได้ค่าเฉลี่ยของผลลัพธ์ของการสุ่มออกมา โดยค่าผลลัพธ์ที่ได้จะเท่ากันทุกครั้งที่เรียกใช้งาน

ทั้งสองฟังก์ชันถูกกำหนดให้เป็น Pure Virtual Function ซึ่งจะไม่มีโปรแกรมในคลาสนี้ แต่ส่วนโปรแกรมจะไปปรากฏอยู่ในคลาสลูก ซึ่งทำให้การทำงานของฟังก์ชันขึ้นอยู่กับคลาสลูก ซึ่งจะมีความแตกต่าง และไม่มีความสัมพันธ์ต่อกัน

ในบรรทัดที่ 7 คือการกำหนดฟังก์ชัน seed(char*) ซึ่งทำให้เมื่อเปลี่ยน seed จะได้ค่าลำดับของการสุ่มที่แตกต่างกันออกไป และในบรรทัดที่ 9 RNG* rng_ นั้นได้สร้างพอยน์เตอร์ให้ชี้ไปยังตัวสร้างเลขสุ่ม ซึ่งจะใช้เป็นข้อมูลเข้าของการสร้างตัวแปรสุ่มอีกทอดหนึ่ง

```
1. class RandomVariable : public TclObject {
2. public:
3.     RandomVariable();
4.     virtual double value() = 0;
5.     virtual double avg() = 0;
6.     int command(int argc, const char*const* argv);
7.     int seed(char *);
8. protected:
9.     RNG* rng_;
10.};
```

รูปที่ 3.26 แสดงโครงร่างของคลาส RandomVariable

3.4 สรุป

ในบทนี้ได้กล่าวถึงแนวคิด หลักการและการใช้งานโปรแกรมจำลอง ns โดยได้อธิบายรายละเอียดขององค์ประกอบภายในที่สำคัญ คือ โหนด ลิงค์ โพรโทคอลที่ใช้บ่อยและอ็อบเจกต์ที่ใช้สำหรับการบันทึกการทำงานของการจำลองซึ่งเป็นที่มาของแฟ้มผลลัพธ์ และในตอนท้ายยังได้กล่าวถึงคลาสต้นแบบที่วิทยานิพนธ์นี้เกี่ยวข้องด้วยทั้งในรูปแบบของการรับทอด เช่น คลาส Application หรือ Tcp Application และในรูปแบบของการทำงานร่วมกันเป็นลำดับชั้นโดยอาศัยแนวคิดเรื่องภาวะหลายรูปแบบสำหรับการอ้างอิงถึงกัน เช่น คลาส Random Variable ทั้งนี้เพื่อเป็นฐานสำหรับบทต่อไปที่ว่าด้วยการออกแบบโมดูลของ HLA