دانشگاه آزاد اسلامی

واحد علوم و تحقیقات تهران

"گروه مهندسی کامپیوتر"

شبیه‌سازی شبکه‌های کامپیوتری

"آشنایی با بسته نرم‌افزاری NS-2"

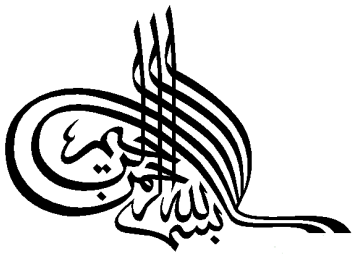
« گزارش فنی»

نگارش ۱,۰۰

تهیه کننده: سیامک عضدالملکی

Azods@Yahoo.com

بهار ۱۳۸۰



دانشگاه آزاد اسلامی

واحد علوم و تحقیقات تهران

گروه مهندسی کامپیوتر

شبیه‌سازی شبکه‌های کامپیوتری

آشنایی با بسته نرم‌افزاری NS-2

« گزارش فنی »

نگارش ۱,۰۰

تهیه کننده: سیامک عضدالملکی

Azods@Yahoo.com

بهار ۱۳۸۰

تقدیم به اساتید ارجمند
جناب‌آقای دکترعلی موقررحیم‌آبادی،
و جناب آقای دکترحسین پدرام

فهرست مطالب

پیشگفتار ۶

۱-مقدمه ۸

۲-مفاهیم پایه ۱۳

۲-۱- OTcl زبان کاربر ۱۳

۲-۲-مثال ساده شبیه‌سازی ۱۵

۲-۳-زمان‌بند رخداد ۱۹

۲-۴-اجزاء شبکه ۲۱

۲-۴-۱-گره و مسیریابی ۲۲

۲-۴-۲-پیوند ۲۳

۲-۴-۳-ناظر صف ۲۴

۲-۴-۴-مثالی از جریان بسته ۲۵

۲-۵-بسته ۲۶

۳-پس از شبیه‌سازی ۲۷

۳-۱-مثال تحلیل ردیابی ۲۷

۳-۲-مثالی از ناظر صف RED ۲۹

۴-توسعه بسته نرم افزاری NS ۳۲

۴-۱-چه چیزی در کجاست؟ ۳۲

۴-۲-پیوند OTcl ۳۴

۴-۲-۱-صدور رده های C++ به OTcl ۳۴

۴-۲-۲-صدور متغیرهای رده C++ به OTcl ۳۵

۴-۲-۳-صدور فرمانهای کنترلی شئ C++ به OTcl ۳۶

۴-۲-۴-اجرای یک فرمان OTcl از درون برنامه C++ ۳۷

۴-۲-۵-ترجمه، اجرا و آزمایش MyAgent ۳۷

۴-۳-افزودن کاربردها و کارگزارهای جدید ۳۸

۴-۴-افزودن صف جدید ۴۴

۴-۵-مثال شبکه های محلی ۴۶
۴-۶-مثال چند پراکنی ۴۷
۴-۷-مثال سرویس دهنده تور جهان گستر ۴۷

۵-مراجع ۴۸
پیوست ۱:دست نویس مربوط به یک شبیه‌سازی ساده ۴۹
پیوست ۲:دست نویس مربوط به یک شبیه‌سازی ساده با ردیابی ۵۱
پیوست ۳:دست نویس مربوط به صف RED ۵۳
پیوست ۴:متن برنامه مربوط به پیوند ۵۵
پیوست ۵:دست نویس مربوط به پیوند ۵۶
پیوست ۶:فایلهای کاربرد چند رسانه ۵۷
پیوست ۷:صف DtRrQueue ۶۵
پیوست ۸:دست نویس شبیه‌سازی شبکه های محلی ۶۸
پیوست ۹:دست نویس شبیه‌سازی چند پراکنی ۷۰
پیوست ۱۰:دست نویس شبیه‌سازی تور جهان گستر ۷۱

پیشگفتار

گسترش روزافزون شبکه‌های کامپیوتری و به طور مشخص *اینترنت* یکی از شاخص‌ترین و مهمترین رخدادهایی به شمار می‌رود که در دهه‌های اخیر ارتباطات را به خود معطوف داشته است. این رشد فزاینده علاوه بر آنکه در برگیرنده محیط اصلی و سنتی این شبکه (سکوهای دانشگاهی، تحقیقاتی و پژوهشی) بوده، مدتی است محیطهای روزمره اداری، تجاری و حتی خانگی را به سرعت در می‌نوردد. بدون شک این رشد و توسعه مرهون تحقیقات و پژوهشهایی است که به طور مستمر و پیوسته در این شاخه از علم کامپیوتر و سایر زمینه‌های مرتبط در سایر علوم صورت می‌پذیرد. یکی از زمینه‌های جالب توجه در شبکه‌های کامپیوتری فراهم آوردن امکان ارزیابی و سنجش ایده‌ها، پروتکلها، معماریها و پاسخ به سوالات متنوع در مورد آنها، قبل از تحقق و پیاده سازی فیزیکی یک شبکه و اجزاء آن است. در واقع شبیه‌سازی شبکه‌های کامپیوتری محیطی را در اختیار محققین و پژوهشگران قرار می‌دهد که با استفاده از آن پاسخ به پرسشهای "چه می‌شود اگر ...؟" [۱] بدون نیاز به پیاده سازی فیزیکی و با استفاده از مدل شبیه‌سازی شده، و با تقریب قابل قبول حاصل گردد.

بسته نرم‌افزاری ***شبیه‌ساز شبکه‌های کامپیوتری***[۲] یا NS-2 (نگارش ۲) یک شبیه‌ساز شبکه‌های کامپیوتری و مبتنی بر روش شبیه‌سازی رخدادهای گسسته[۳] و شئ گرا[۴] است. این بسته نرم‌افزاری دردانشگاه‌کالیفرنیا (برکلی)[۵] طراحی و پیاده سازی شده است. این نرم افزار توسط زبانهای برنامه سازی C++ و OTcl نوشته شده است و یکی از سودمندترین ابزار برای شبیه‌سازی شبکه‌های کامپیوتری محلی و گسترده می‌باشد. با وجود اینکه NS پس از آشنایی اولیه نسبتاً سهل الاستفاده است، ولی برای کاربران عادی، در نظر اول پیچیده و دشوار به نظر می‌رسد. شاید یکی از دلایل این امر فقدان راهنمایِ کاربران مناسب برای این بسته نرم‌افزاری باشد تا از طریق آن کاربران بتوانند با چهارچوب عملیاتی این بسته نرم‌افزاری و قابلیتهای اساسی آن آشنا شوند. در واقع مستندات گوناگون فعلی این بسته نرم افزاری نیز عموماً توسط تولیدکنندگان آن تهیه شده است که اطلاع عمیق و جزئی از آن داشته و با چنان سطحی از جزئیات تهیه شده که برای کاربران ماهر و حرفه‌ای این نرم‌افزار و نه کاربران عادی، مناسب است. هدف از تهیه این راهنمای سریع، آشنایی اولیه و پایه‌ای کاربران این بسته نرم افزاری با اجزاء آن و چگونگی عملکرد آن می‌باشد. پس از مطالعه این راهنمای سریع، کاربر قادر خواهد بود تا با روش برپایی[۶] یک شبیه‌سازی شبکه، روش دسترسی به اطلاعات بیشتر در مورد کدهای اجزاء شبکه[۷]، روش ایجاد یک جزء جدید و نظایر آن آشنا شود.

1 -"What … if …" Questions
2-Network Simulator
3 -Discrete Event Simulator
4 -Object Oriented
5 -University of California Berkeley
6 -Setup
7 -Network Components

روش بکارگرفته شده در این راهنمای سریع، آموزش از طریق ارائه مثال است. مؤلف سعی نموده تا با استفاده از تجربیات شخصی و تجربیات سایر کاربران و در قالب مثالهای کاربردی و توضیحات مربوطه، روش بهره‌گیری از این بسته نرم‌افزاری را ارائه نماید. با وجود اینکه در این راهنما تمام کاربردهای ممکن شبیه ساز تحت پوشش قرار نگرفته است، لیکن امید داریم که با مطالعه آن، کاربران قادر به آشنایی سریع و استفاده از این بسته نرم‌افزاری گردند.

در تهیه این گزارش فنی سعی شده است تا جای ممکن کم‌تر از واژه های انگلیسی استفاده شود و معادلهای انتخاب شده عموماً برگرفته از فرهنگ بسامدی واژگان کامپیوتر و انفورماتیک (شورای عالی انفورماتیک) و واژه نامه انجمن انفورماتیک ایران می‌باشد. با این وجود به سبب نامأنوس بودن برخی از برابرنهاده ها، معادل انگلیسی هر واژه در اولین کاربرد زیرنویسی شده است. در بعضی موارد نیز معادل انگلیسی در کاربردهای بعدی جهت یادآوری و سهولت درک مطلب مجدداً ذکر شده است.

هرچند که تلاش بسیار شده است که این گزارش بدون اشتباه باشد، ولی نگارنده به هیچ وجه نمی‌تواند ادعا کند که خطا و اشتباهی در آن وجود ندارد. لذا امیدوارم اساتید گرامی و دانشجویان محترم، اینجانب (Azods@Yahoo.com) را از نظرات اصلاحی خود مطلع کنند و اشتباهات احتمالی را یادآور شوند تا در نگارشهای بعد، نسبت به رفع نواقص اقدام گردد.

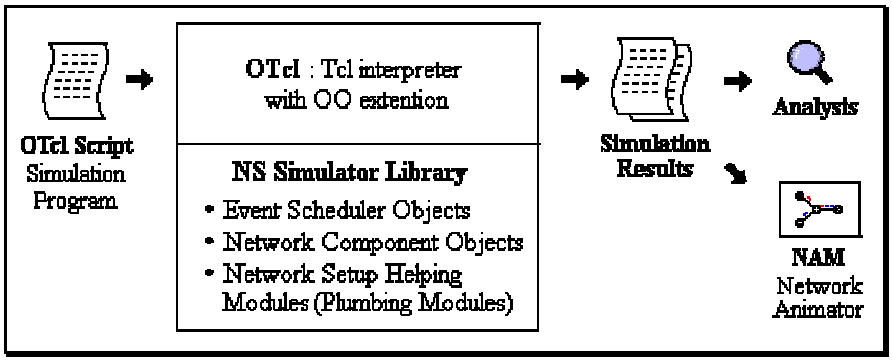
تهران – بهار ۱۳۸۰
سیامک عضدالملکی

۱-مقدمه

بسته نرم افزاری شبیه‌ساز شبکه‌های کامپیوتری، یک شبیه ساز مبتنی بر رخداد است که در دانشگاه کالیفرنیا (برکلی) طراحی و پیاده سازی شده است. این بسته نرم افزاری قادر است طیف وسیعی از پروتکلها و همبندیهای شبکه‌های محلی و گسترده را شبیه‌سازی نماید. لازم به تذکر است که به سبب فراگیری و محبوبیت اینترنت، و خانواده پروتکل TCP/IP به عنوان یکی از اجزاء تفکیک ناپذیر آن، این بسته نرم افزاری قابلیتهای قابل توجهی در شبیه‌سازی خانواده پروتکل فوق از خود نشان می‌دهد. این بسته نرم افزاری پروتکلهایی نظیر TCP، UDP ، رفتار منابع مولّد ترافیک نظیر پروتکلهای Telnet، FTP، WEB،CBR[1]، VBR[2]، مکانیزمهای مدیریت صف در مسیریابها[3] مانند Drop Tail،RED[4]، و CBQ[5] و الگوریتمهای مسیریابی نظیر دیجسترا[6] و امثالهم را مهیا می‌نماید. علاوه بر اینها NS مکانیزمهای چندپراکنی[7] و برخی از پروتکلهای دسترسی به رسانه[8] برای شبیه‌سازی شبکه‌های محلی را نیز در اختیار می‌گذارد. این پروژه در حال حاضر به عنوان بخشی از پروژه VINT[9] است. در کنار NS ابزار کمکی دیگری نیز در دست طراحی و پیاده سازی است و این ابزار امکان نمایش نتایج شبیه‌سازی، تحلیل، و تبدیل همبندیهای[10] شبکه توسط مولدهای رایج به شکل قابل استفاده در NS را در بر می‌گیرند. در حال حاضر نگارش دوم این بسته نرم افزاری با بهره گیری از قابلیتهای زبان برنامه نویسی C++ و زبان دست نویس[11] OTcl طراحی وپیاده سازی شده است. OTcl زبان دست نویس با ساختار نگارشی زبان Tcl و امکانات و تواناییهای شئ گرایی است. قابلیتهای شئ گرایی و افزوده شده به Tcl در دانشگاه MIT طراحی و پیاده سازی گردیده است.

در این راهنمای سریع کاربران، ابتدا به بررسی ساختار NS می‌پردازیم و به دنبال آن با ذکر مثالهای کاربردی با چگونگی استفاده عملی از این بسته نرم‌افزاری آشنا می‌شویم. تصاویر بکار گرفته شده در این راهنما، خصوصاً در زمینه ساختار NS و اجزاء شبکه از مجموعه اسلایدهای پنجمین کارگاه شبیه‌ساز NS و پروژه VINT گرفته شده و در مواردی با تغییرات جزئی مورد استفاده قرار گرفته است. برای کسب اطلاعات بیشتر در خصوص پروژه VINT و بسته نرم‌افزاری NS می‌توانید به نشانی http://www.isi.edu/nsnam مراجعه نمایید. همچنین در انتهای این گزارش فنی فهرستی از مراجع مفید و سودمند تهیه شده است.

1 -Constant Bit Rate Traffic
2 -Variable Bit Rate Traffic
3 -Router
4 -Random Early Detection
5 -Class Based Queue
6 -Dijkstra
7 -Multicasting
8 -Medium Access Control (MAC) Protocols
9 -Virtual Inter Network Test-bed
10 -Topologies
11 -Script



شکل ۱-۱ – ساختمان و اجزاء کلی NS-2

همانگونه که در شکل ۱-۱ ملاحظه می‌شود، در یک نگاه ساده شده از دید کاربر، بسته نرم‌افزاری NS یک مفسر دست نوشته‌های OTcl است که از کتابخانه اشیاء و اجزاء شبکه[۱]، زمان‌بند[۲] رخدادهای شبیه‌سازی، و واحدهای کتابخانه‌ای برپایی شبکه تشکیل شده است. واحدهای کتابخانه‌ای برپایی شبکه در واقع به صورت توابع شئ *شبیه ساز* پیاده‌سازی شده‌اند[۳]. به بیان دیگر به منظور استفاده از NS کاربر می‌بایست تنها برنامه‌ای به زبان دست نویس OTcl بنویسد. به منظور برپایی و شبیه‌سازی یک شبکه، کاربر برنامه ای به زبان OTclمی‌نویسد و در آن زمان‌بند وقایع را تنظیم کرده و با استفاده از توابع اتصال کتابخانه‌ای و اشیاء شبکه، همبندی مورد نظر شبکه خود را توصیف می کند. سپس با استفاده از توابع مربوط به زمان‌بند شبیه ساز، منابع مولد ترافیک و زمان شروع و خاتمه ارسال بسته‌ها توسط این منابع ترافیکی را تعیین می‌کند. یکی از تعابیری که در این بسته نرم افزاری برای مرحله برپایی شبکه به کار برده می شود، عبارت ***لوله کشی***[۴] است. در واقع کاربر در هنگام برپایی شبکه، مسیر داده‌ها را از طریق تنظیم اشاره‌گر یک شئ شبکه به شئ مناسب همسایه برقرار می‌کند (واین عمل لوله کشی تعبیر می‌شود). زمانی که کاربر بخواهد یک شئ جدید شبکه را بسازد، می‌تواند به راحتی کدهای مربوط به آن شئ جدید را بنویسد یا با بهره گیری از کتابخانه اشیاء یک شئ شبکه ترکیبی ایجاد نماید. ممکن است این فرآیند پیچیده به نظر رسد ولی در واقع *لوله کشی* واحدهای ساختمانی در زبان دست نویس OTcl بسیار ساده است. یکی از تواناییهای مهم و قابل توجه NS، توانایی *لوله کشی* و استفاده از اشیاء است.

یکی دیگر از اجزاء ساختمانی مهم در بسته نرم افزاری NS زمان‌بند رخدادها است. یک رخداد در NS در واقع یک شماره شناسایی بسته[۵] است که در زمان مشخص شده برای رخداد، منحصر به فرد بوده و همچنین اشاره گر به شیء‌ای است که آن رخداد را پردازش می‌کند. به بیان دیگر رخداد مجموعه شماره شناسایی و اشاره‌گر است. در NS زمان‌بند رخداد زمان شبیه‌سازی را

1 -Network Component Object Libraries
2 -Scheduler
3-در روش برنامه سازی شیء گرایی به این توابع Member Funciton گفته می شود.
4 -Plumbing
5 -Packet ID

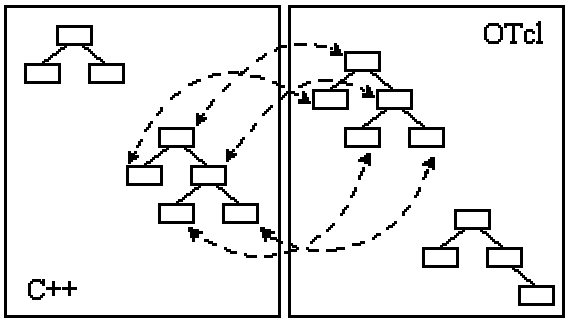
ردگیری می‌نماید و با فراخوانی عناصر مناسب شبکه، رخدادهای موجود در صف رخدادها برای زمان حال (زمان جاری شبیه ساز)را به سمت توابع مربوطه هدایت کرده و به این ترتیب پردازش لازم برروی بسته توسط تابع مربوطه و مناسب اعمال می‌شود. عناصر شبکه از طریق تبادل بسته‌ها با یکدیگر ارتباط برقرار می نمایند. تمام عناصر شبکه که برای پردازش بسته به زمان (تأخیر پردازش) نیاز دارند، با صدور یک رخداد به زمان‌بند رخدادها و انتظار وقوع آن رخداد عمل می‌کنند. با سر رسید زمان بروز رخداد، زمان‌بند با فراخوانی تابع مربوطه، عنصر شبکه مربوط به پردازش آن رخداد (که در این حالت مولد رخداد نیز بوده)را درگیر می نماید. به عنوان مثال یک راه گزین یا سوئیچ شبکه که رفتار یک سوئیچ با ۲۰ میکروثانیه تأخیر راه گزینی را شبیه‌سازی می‌نماید، به محض دریافت بسته، رخدادی برای همان بسته و ۲۰ میکروثانیه بعد را به زمان‌بند اعلام می‌نماید. بعد از گذشت ۲۰ میکروثانیه، زمان‌بند این رخداد را از صف رخدادها خارج کرده و آن را به عنصر سوئیچ ارسال می کند. سوئیچ نیز بسته را به عنصر پیوندخروجی[1] مناسب هدایت (ارسال) می‌کند.

کاربرد دیگر زمان‌بند رخدادها، زمان سنجی[2] است. به عنوان مثال پروتکل TCP برای اطلاع از سر آمدن زمان ارسال مجدد یک بسته[3] (وارسال مجدد آن بسته با شماره ترتیب قبلی در پروتکل TCP و یک شماره بسته جدید NS) به زمان سنجی نیاز دارد. زمان سنجها مشابه با گره‌های تأخیر برای اندازه گیری زمان از زمان‌بند رخدادها استفاده می نمایند. با این تفاوت که زمان سنجها، زمان یک مقدار زمانی منسوب به بسته را اندازه گیری کرده و پس از سر آمدن مدت زمان معینی پردازش خاصی برروی بسته انجام می دهند و رخداد جدیدی را شبیه‌سازی نمی کنند.

همانگونه که پیشتر ذکر شد، بسته نرم افزاری NS علاوه بر زبان دست نویس OTcl ، با زبان برنامه نویسی C++ نیز نوشته شده است. به دلیل افزایش کارایی، NS پیاده سازی مسیر داده را از پیاده سازی مسیر کنترل مجزا نموده است. به منظور کاهش زمان پردازش بسته و رخداد (نه زمان شبیه‌سازی)، زمان‌بند رخدادها و نیز اشیاء پایه و مقدماتی شبکه در مسیر دادها تماماً به زبان C++ نوشته شده اند. این اشیاء ترجمه شده، با استفاده از پیوند بین اشیاء، در دست نویسهای OTcl نیز در دسترس هستند. در واقع این پیوند، یک شئ نظیر در دست نویس OTcl برای شئ ترجمه شده در زبان C++ ایجاد می‌کند. به این ترتیب متغیرها و متدهای موجود در زبان C++ در اختیار دست نویس OTcl نیز قرار می‌گیرد و می توان اشیاء C++ را از طریق دست نویسها نیز کنترل نمود. همچنین امکان اضافه کردن متغیرها و متدهای جدید به اشیاء ترجمه شده در زبان C++ نیز مهیا می‌گردد. سایر اشیایی که در حین شبیه‌سازی نیازی به کنترل ندارند و یا به صورت داخلی مورد استفاده قرار می گیرند، به دست نویسها پیوند نخواهند خورد. شکل ۱-۲ مثالی از سلسله مراتب

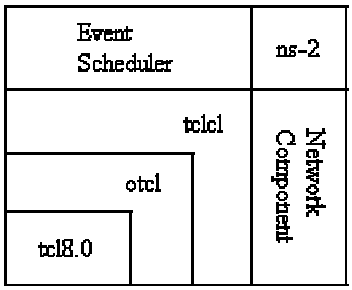
1 -Output Link
2 -Timer
3 -Retransmission timeout

اشیاء در زبان C++ و OTcl را نشان می‌دهد. نکته قابل توجه در این شکل آن است که سلسله مراتب اشیاء در C++ به طور مشابه با سلسله مراتب نظیر در OTcl مرتبط می شوند.



شکل ۱-۲ – دوگانی[1] C++ و OTcl

شکل ۱-۳ معماری عمومی بسته نرم افزاری NS را نشان می‌دهد. در این شکل یک کاربر عمومی NS را می توان در گوشهٔ پایین و سمت چپ شکل در نظر گرفت. این کاربر طراحی و اجرای شبیه‌سازی شبکه را با استفاده از دست نویس Tcl و کتابخانه اشیاء OTcl انجام می‌دهد. زمان‌بند وقایع و بیشتر عناصر شبکه به زبان C++پیاده سازی شده و با استفاده از پیوند خاصی در OTcl قابل استفاده هستند. این پیوند خاص با استفاده از tclcl پیاده سازی می شود. تمام این مجموعه در کنار یکدیگر محیط نرم افزاری NS را تشکیل می دهند که در واقع یک مفسر توسعه یافته Tcl با کتابخانه ای از عناصر شبکه است.



شکل ۱-۳-معماری نرم افزاریNS

در این بخش ساختار و معماری عمومی NS را بررسی کردیم. در اینجا ممکن است این سؤال مطرح شود که نتایج شبیه‌سازی به چه شکلی در اختیار محقق یا کاربر قرار می‌گیرد. همانگونه که در شکل ۱-۱ دیده می شود، هنگامی که شبیه‌سازی خاتمه می یابد، NS یک یا چند فایل متنی به عنوان خروجی تولید می‌کند که این فایلها داده‌های تفصیلی شبیه‌سازی را در بر دارند. این داده‌ها برای تحلیل شبیه‌سازی یا به عنوان ورودی ابزار نمایش داده های شبیه‌سازی نظیر NAM [2]قرار می‌گیرند. NAM نرم‌افزاری است که به عنوان بخشی از پروژه VINT پیاده سازی شده

1 -Duality
2 -Network Animator

است. این نرم افزار از یک رابط کاربر دوستانه [1] بهره می‌برد. این رابط کاربر شباهت زیادی به نرم‌افزارهای پخشCD صوتی دارد(دکمه های پخش، حرکت به جلو، حرکت به عقب، توقف موقت، و ...) و با استفاده از آن می توان نتایج شبیه‌سازی را با دقت زمانی مورد نظر مورد مطالعه و بررسی قرار داد. به علاوه این نرم افزار قادر است اطلاعات مربوط به گذردهی[2]، تعداد بسته‌های از دست رفته در هر پیوند ارتباطی و نظایر آن را در قالب نمودارهای گرافیکی نمایش و ارائه دهد.

1 -User Friendly
2 -Throughput

۲.مفاهیم پایه

در این بخش به بررسی مفاهیم پرکاربرد و اساسی در بسته نرم‌افزاری NS می‌پردازیم و بحث خود را با زبان دست نویس OTcl به عنوان رابط کاربر NS و سپس نمونه‌های عملی ادامه می‌دهیم.

۲-۱- OTcl : زبان کاربر

همانگونه که در بخش قبل ذکر شد، NS در واقع یک مفسر OTcl به همراه کتابخانه‌ای از اشیاء شبیه‌سازی شبکه است. به همین سبب آشنایی با زبان دست نویس OTcl برای استفاده از NS ضروری و اجتناب ناپذیر می‌باشد.(برای اطلاعات بیشتر در مورد این زبان دست نویس به مراجع ۸ و ۹ مراجعه کنید.) در این بخش مثالهایی از دست نویسهای OTcl و Tcl به منظور انتقال مفاهیم بنیادی و پایه‌ای این محیط دست نویس ارائه می‌شود. با استفاده از این مثالها ایده های اصلی برنامه سازی در OTcl به منظور برپایی همبندی شبکه و شبیه‌سازی آن منتقل می‌شود. این مثالها از پنجمین کارگاه معرفی NS استخراج شده است. از این بخش به بعد چنین فرض می‌شود که کاربر بسته نرم‌افزاری NS را نصب کرده و نیز با زبانهای برنامه سازی Cو C++ و مفاهیم اولیه برنامه سازی شئ گرا آشنایی دارد.

شکل ۲-۱ یک دست نویس کلی به زبان Tcl است. در این مثال نحوه ایجاد رویه[1] و فراخوانی آن، انتصاب مقادیر به متغیرها و نحوه ایجاد حلقه‌های تکرار نشان داده شده است. با عنایت بر اینکه OTcl گونهٔ گسترش یافته و شئ گرای زبان دست نویس Tcl است، این نکته را روشن می‌کند که تمامی دستورات Tcl در OTcl نیز قابل استفاده می‌باشند. در واقع ارتباط بین Tcl و OTcl مشابه ارتباط زبانهای C و C++ است. برای اجرای این دست نویس باید نام دست نویس (به عنوان مثال ex-tcl.tcl) را به عنوان پارامتر برنامه NS معرفی نمایید. در صورتی که tcl8.0 برروی کامپیوتر شما نصب شده باشد،فرمان tcl ex-tcl.tcl نیز همین عمل را انجام خواهد داد.

```
# Writing a procedure called "test"
proc test {} {
    set a 43
    set b 27
    set c [expr $a + $b]
    set d [expr [expr $a - $b] * $c]
    for {set k 0} {$k < 10} {incr k} {
  if {$k < 5} {
     puts "k < 5, pow = [expr pow($d, $k)]"
  } else {
     puts "k >= 5, mod = [expr $d % $k]"
  }
    }
}

# Calling the "test" procedure created above
test
```

شکل ۲-۱- یک دست نویس ساده Tcl

1 -Procdure

در زبان دست نویس Tcl کلمه کلیدی **proc** برای تعریف یک رویه به کار برده می شود. به دنبال این کلمه کلیدی، نام رویه و سپس پارامترهای رویه (در داخل { }) ذکر می‌شوند. کلمه کلیدی **set** برای انتصاب یک مقدار به یک متغیر به کار برده می شود. عبارت **[expr . . .]** سبب می‌شود که شبیه ساز مقدار عبارت داخل قلاب ([ ]) و مقابل کلمه کلیدی expr را محاسبه نماید. نکته قابل توجه دیگر آن است که برای دسترسی به مقدار یک متغیر می‌بایست از نماد $ قبل از نام متغیر استفاده نمود. (به خط if ($K < 5) در شکل ۲-۱ توجه کنید.) کلمه کلیدی **puts** رشته کاراکتری ذکر شده و محصور در نمادهای گیومه را برروی صفحه نمایش نشان می دهد. خروجی این مثال به شرح زیر است:

```
k < 5, pow = 1.0
k < 5, pow = 1120.0
k < 5, pow = 1254400.0
k < 5, pow = 1404928000.0
k < 5, pow = 1573519360000.0
k >= 5, mod = 0
k >= 5, mod = 4
k >= 5, mod = 0
k >= 5, mod = 0
k >= 5, mod = 4
```

شکل ۲-۲- خروجی مثال ۲-۱

مثال بعدی نمونه ای از برنامه سازی شئ گرا را در دست نویس های OTcl نشان می‌دهد. این مثال بسیار ساده است و چگونگی ایجاد و استفاده از یک شئ را نشان می‌دهد. در صورتی که شما یک کاربر معمولی و عادی NS باشید، احتمال نوشتن یک شئ جدید توسط شما خیلی کم خواهد بود ولی از آنجا که تمام اشیاء در NS چه توسط زبان برنامه نویسی C++ نوشته شده و به دست نویس OTcl پیوند خورده باشند و چه اساساً در OTcl ایجاد شده باشند؛ نهایتاً به صورت یک شئ OTcl رفتار می‌کنند، لذا درک اشیاء OTcl سودمند خواهد بود.

```
# Create a class call "mom" and
# add a member function call "greet"
Class mom
mom instproc greet {} {
    $self instvar age_
    puts "$age_ years old mom say:
    How are you doing?"
}
# Create a child class of "mom" called "kid"
# and overide the member function "greet"
Class kid -superclass mom
kid instproc greet {} {
    $self instvar age_
    puts "$age_ years old kid say:
    What's up, dude?"
}
# Create a mom and a kid object set each age
set a [new mom]
$a set age_ 45
set b [new kid]
$b set age_ 15
# Calling member function "greet" of each object
$a greet
$b greet
```

شکل ۲-۲-یک دست نویس ساده OTcl

مثال۲-۲ یک دست نویس OTcl است که دو کلاس (شئ)، mom و kid را تعریف کرده و شئ kid فرزند شئ mom می باشد. در هر دو شئ (کلاس)تابع greet نیز وجود دارد. پس از تعریف

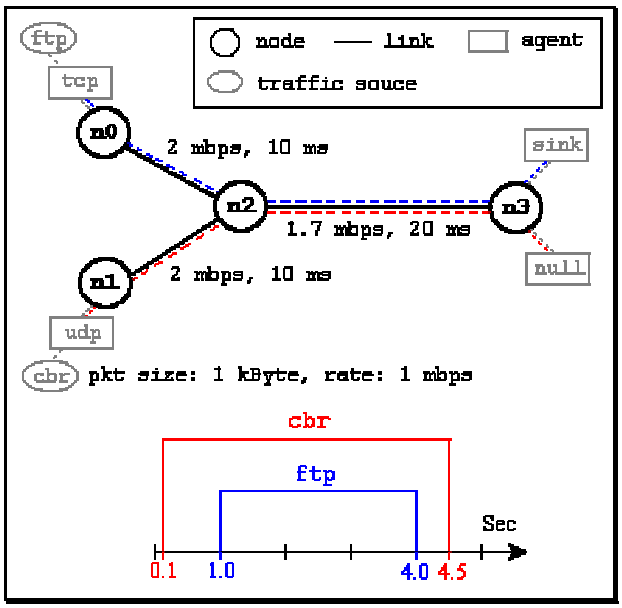
اشیاء نمونه‌ای[1] از هر یک توصیف می شود و متغیر age در هر یک مقداری را می پذیرد.(در کلاس mom مقدار ۴۵ و در کلاس kid مقدار ۱۵ در متغیر age جایگزین می‌شود) سپس تابع greet فراخوانی می‌شود. در این مثال کلمه کلیدی **Class** یک شئ ایجاد کرده و کلمه کلید **instproc** یک تابع (Member Function) را تعریف می‌نماید. برای برقراری رابطه توارث [2] از کلمه کلیدی **superclass** استفاده شده است. در هنگام تعریف توابع (متدها) کلمه کلیدی **$self** مشابه اشاره گر this در زبان برنامه سازی C++ عمل می‌کند. عبارت **instvar** کنترل می‌کند که آیا متغیر ذکر شده در این کلاس (شئ) یا در کلاس والد (شئ جد) تعریف شده است یاخیر. اگر متغیر تعریف شده باشد، همان متغیر ارجاع داده می‌شود و اگر تعریف نشده باشد، یک متغیر جدید تعریف می‌شود. در نهایت برای ایجاد یک شئ نمونه [3] از کلمه کلیدی **new** استفاده می شود. خروجی این مثال به شکل زیر است:

```
45 year old mom say:
    How are you doing?
15 year old kid say:
    What's up, dude?
```

شکل ۲-۳- خروجی مثال ۲-۲

۲-۲-مثال ساده شبیه‌سازی

در این بخش به بررسی یک مثال ساده شبیه‌سازی می‌پردازیم. شکل ۲-۴ یک سناریوی شبیه‌سازی ساده را نشان می‌دهد. در این مثال یک دست نویس ساده OTcl دیده می‌شود که یک پیکربندی ساده شبکه را ایجاد می کند. این پیکربندی سناریوی شبیه‌سازی شکل۲-۴ را اجرا می‌کند. برای اجرای این مثال فایلی به نام ns-simple.tcl را ایجاد کرده و دستورات مربوطه (پیوست ۱) را در آن وارد کنید. سپس با استفاده از فرمان ns ns-simple.tcl سناریوی شبیه‌سازی را اجرا کنید.



شکل ۲-۴- یک همبندی ساده شبکه و سناریوی شبیه‌سازی آن

1 -Instance
2 -Inheritance
3 -Instance

این پیکربندی شبکه از چهار گره (n0,n1,n2,n3) تشکیل شده‌است که در شکل ۲-۴ ملاحظه می‌شوند. پهنای باند خطوط ارتباطی دو طرفه n0 و n2 و نیز خطوط n1 و n2 ، ۲ مگابیت در ثانیه و تأخیر آنها ۱۰ میلی ثانیه است. پهنای باند خط ارتباطی n2 و n3 ، ۱/۷ مگابیت در ثانیه و تأخیر آن ۲۰ میلی ثانیه است. هر گره از یک صف با خاصیت Drop Tail (سر ریز از انتها) بهره می‌برد که حداکثر ظرفیت آن ۱۰ است. یک کارگزار[1] (سرویس گیرنده)TCP به گره n0 متصل شده است که یک ارتباط TCPرا با کارگزار گیرنده (سرویس دهنده) و متصل به گره n3 برقرار می‌کند. به صورت پیش فرض حداکثر اندازه بسته کارگزار TCP، ۱ کیلوبایت در نظر گرفته شده است. کارگزار سرویس دهنده بسته های تصدیق(ACK) [2]را تولید کرده و به فرستنده (سرویس‌گیرنده) ارسال می‌کند. به گره n1 یک کارگزار پروتکل UDP متصل شده است. این کارگزار نیز با سرویس دهنده null در گره n3 ارتباط دارد. کارگزار پوچ (Null) کارگزاری است که بسته های دریافتی را رها می‌کند. به کارگزارهای TCP و UDPبه ترتیب مولدهای ترافیک FTP و CBR متصل شده است. مولد ترافیک CBR به گونه‌ای پیکربندی شده است که ۱ کیلو بایت بسته را با نرخ ۱ مگابیت در ثانیه تولید می‌کند و مولد ترافیک ثابت CBR در زمان 0.1ثانیه شروع و در زمان 4.5ثانیه متوقف می‌شود و مولد ترافیک FTP نیز در ثانیه 1.0 شروع و در ثانیه 4.0 شبیه‌سازی متوقف می‌شود.

آنچه در ادامه می آید توضیح این دست نویس است. به صورت کلی هر دست نویس NS با ایجاد یک نمونه[3] شئ آغاز می شود.

[Set ns [new simulator : این دستور یک شئ *شبیه ساز* NS را ایجاد می کند و آن را به متغیر ns منصوب می کند. کارهایی که این خط انجام می دهد به این قرار است:

۱.ساختار بسته را مقدار دهی آغازین می کند.
۲.زمان‌بند را ایجاد می کند.
۳.ساختار آدرس دهی پیش فرض را انتخاب می کند.

شئ *شبیه ساز* دارای متدهایی است که عملیات زیر را انجام می دهند:

- اشیاء ترکیبی مانند گره ها و خطوط ارتباطی را ایجاد می‌کند.
- اشیاء مربوط به عناصر شبکه را به یکدیگر متصل می‌کند. (به عنوان مثال attach-agent)
- پارامترهای اجزاء شبکه به خصوص پارامترهای اشیاء ترکیبی را تنظیم می‌کند.
- ارتباط بین کارگزارها را برقرار می کند. (به عنوان مثال اتصال بین مولد ترافیک TCP و گیرنده را برقرار می‌کند.)
- پارامترهای نمایش توسط نرم افزار NAM را مشخص می‌کند.

1 -Agent
2 -Acknowledge
3 -Instance

اکثر متدهای این شئ، توابعی هستند که برای برپایی شبکه و زمان‌بندی مورد استفاده قرار می‌گیرند، با این وجود برخی از توابع برای نمایش نتایج توسط NAM تعبیه شده‌اند. پیاده‌سازی متدهای شئ شبیه ساز در فایل /ns-2/tcl/lib/ns-lib.tcl قرار دارند.

$ns color *fid color* :این عبارت رنگ بسته ها را برای یک جریان مشخص داده تعیین می‌کند که توسط پارامتر fid مشخص می‌شود. این متد از شئ *"شبیه ساز"*، تنها برای نمایش خروجی در زیر سیستم NAM به کاربرد می‌شود و در روند اصلی شبیه‌سازی بی تأثیر است.

$ns nametrace-all *file-descriptor* : این متد (تابع) به شبیه ساز فرمان می‌دهد که تمام اطلاعات و نتایج شبیه‌سازی را در قالب مناسب جهت استفاده در برنامه NAM و به عنوان ورودی این برنامه، ثبت کند. همچنین نام فایلی که این نتایج در آن ذخیره خواهد شد را به صورت پارامتر دریافت می‌کند. البته فرمانی که نتایج را در نهایت در فایل فوق ثبت می‌کند، *$ns* flush-trace نام دارد. به طور مشابه تابع trace-all نتایج شبیه‌سازی را به صورت معمولی (و نه در قالب ورودی مورد نیاز NAM) ذخیره می‌کند.

Proc *finish { }* : پس از آنکه این شبیه‌سازی با فرمان *$ns* at 5.0 *finish* خاتمه یافت، این رویه فراخوانی می‌شود. این رویه پردازشهای لازم پس از اجرای شبیه‌سازی را انجام می دهد.

Set *n0* [*$ns* node] :متد (تابع) node یک گره جدید ایجاد می کند. یک گره در شبیه ساز NS یک شئ ترکیبی است که ازنشانی و درگاه طبقه بندی کننده[1] ساخته شده است. کاربران می توانند با ایجاد یک نشانی و یک درگاه طبقه بندی کننده مجزا و سپس اتصال آنها به یکدیگر یک گره ایجاد کنند ولی این تابع تعریف گره را به شکل ساده تری انجام می دهد. برای اطلاع بیشتر از نحوهٔ ایجاد یک گره به فایلهای ns-2/tcl/libs/ns-lib.tcl و ns-2/tcl/libs/ns-node.tcl مراجعه کنید.

$ns duples-link *node1 node2 bandwidth delay queue-type* : این عبارت دوپیوند ساده[2] و یک طرفه با پهنای باند و تأخیر مشخص شده ایجاد می کند و دوگره معین شده را به یکدیگر متصل می نماید. در NS صف ارسال بسته‌ها (صف خروجی) در یک گره به عنوان بخشی از پیوند ارتباطی پیاده سازی می‌شود. بنابر این کاربر می بایست در هنگام تعریف و ایجاد یک پیوند، نوع صف را نیز مشخص نماید. در مثال فوق صف به کار رفته از نوع DropTail است. در صورتی که بخواهیم از صف دیگری مانند RED استفاده کنیم تنها می بایست عبارت DropTail را با عبارت REDتعویض کنیم. در بخش بعدی با چگونگی پیاده سازی یک پیوند در NS آشنا می شویم. یک پیوند[3]، مشابه یک گره، یک شئ ترکیبی است و کاربران می توانند اشیاء تشکیل دهنده آنها را به صورت مجزاء تعریف کرده و

1 -Classifier Port
2 -Simplex
3 -Link

سپس این اجزاء را به یکدیگر متصل کند. برای اطلاع بیشتر در مورد پیوند به فایلهای ns-2/tcl/libs/ns-lib.tcl و ns-2/tcl/libs/ns-link.tcl مراجعه کنید. نکته قابل توجه و حائز اهمیت آن است که می توان واحدهای (مولد) خطا[1] را نیز به یک پیوند اضافه نمود و به این ترتیب خطاهای خط ارتباطی نیز شبیه‌سازی می‌شوند.

$ns queue-limit *node1 node2 number* : این عبارت ظرفیت صفهای دو پیوند یک طرفه را مشخص می‌کند که گره های node1 و node2 را به یکدیگر متصل می کند. برای اطلاع بیشتر به فایلهای ns-2/tcl/libs/ns-lib.tcl و ns-2/tcl/libs/ns-link.tcl مراجعه کنید.

$ns duplex-link-op *node1 node2*: این عبارت و خطوط بعدی برای کنترل برنامه نمایش نتایج؛ NAM در نظر گرفته شده است. برای مشاهده اثر این خطوط می توانید آنها را موقتاً حذف کرده و تأثیر آن را مشاهده کنید.

پس از برپایی مقدماتی شبکه، قدم بعدی بر پایی کارگزاران ترافیک نظیر TCP و UDP و منابع ترافیک مانند FTP و CBR و به دنبال آن اتصال آنها به گره ها و کارگزاران متناظر است.

Set *tcp* [new *Agent/TCP*] : این عبارت نحوه ایجاد یک کارگزار TCP را نشان می دهد. در حالت کلی کاربران می توانند هر کارگزار و هر منبع ترافیک را با این روش ایجاد کند. کارگزاران و منابع ترافیک در واقع اشیاء ساده هستند (نه شئ ترکیبی) و اکثراً با استفاده از زبان برنامه سازی C++ پیاده سازی شده و به OTcl پیوند خورده‌اند. بنابراین متد خاصی از شئ شبیه ساز که مورد و نمونه‌ای از این اشیاء را ایجاد نماید وجود ندارد. برای ایجاد یک کارگزار یا یک منبع ترافیک کاربر می‌بایست نام کلاس (رده) این اشیاء را (نظیر Agent/TCP ، Agent/TCPSink، Application/FTP) ذکر کند. برای اطلاع بیشتر به فایل ns-2/tcl/libs/ns-default.tcl/ مراجعه کنید. این فایل مقدار پارامترهای پیکربندی پیش فرض برای اشیاء شبکه موجود در NS را در بردارد. بنابراین بررسی این فایل، شاخص مناسبی از اشیاء موجود در NS و پارامترهای قابل پیکربندی را ارائه می‌کند.

$ns attach-agent *node agent* : این تابع یک کارگزار تولید شده را به یک گره متصل می کند. در واقع کاری که این تابع انجام می دهد آن است که تابع Attach گره مورد نظر را فراخوانی کرده و این تابع کارگزار داده شده را به گره متصل می کند.

$ns connect *agent1 agent2* : پس از ایجاد دوکارگزار که قرار است با یکدیگر ارتباط برقرار نمایند،قدم بعدی فراهم نمودن یک ارتباط منطقی شبکه بین آنها است. این عمل از طریق تنظیم و جهت گیری آدرس مقصد به سمت آدرس درگاه دیگری فراهم می‌شود.

با فرض آنکه تمام پیکربندیهای مورد نیاز شبکه انجام شده باشد، مرحله بعدی نوشتن سناریوی شبیه‌سازی (یا به عبارت دیگر زمان‌بندی شبیه‌سازی) است. شئ ***شبیه‌ساز*** از متدهای متنوعی به منظور زمان‌بندی شبیه‌سازی بهره می برد ولی در اینجا به مهمترین توابع (متدهای) این شئ می پردازیم. این متدها از جمله متدهای پرکاربرد هستند.

1 -Error Modules

$ns at *time "String"*: این تابع از شئ شبیه ساز، زمان‌بند را وادار می کند که رشتهٔ کاراکتری پارامتر را ("*String*") را در زمان شبیه‌سازی‌داده‌شده،اجرا نماید. به عنوان مثال عبارت *$ns* at *0.1 "$cbr start"* زمان‌بند شبیه‌سازی را مجبور می‌کند که تابع srart از شئ مولد ترافیک cbr را در زمان 0.1 شبیه‌سازی فرا خواند. این تابع به عنوان یک مولد، ترافیک داده‌ای به شکل ثابت CBR را وارد شبکه می‌نماید. در NS معمولاً یک مولد ترافیک داده ها را ارسال نمی کند، بلکه تنها به کارگزارش[1] در لایهٔ پایین اطلاع می‌دهد که مقداری داده برای ارسال دارد، و این کارگزار است که از میزان داده ارسالی با توجه به نوع خط ارتباطی اطلاع پیدا کرده و بسته‌ها اطلاع پیدا می‌کند.

پس از پیکربندی شبکه، زمان‌بندی شبیه‌سازی و توصیف رویه های پس از شبیه‌سازی تنها چیزی که باقی می ماند اجرای شبیه‌سازی است که با فرمان *$ns* run انجام می شود.

۲-۳-زمان‌بند رخداد

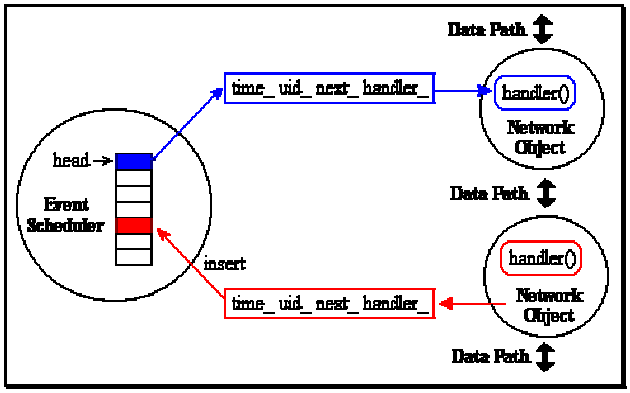
در این بخش به بررسی زمان‌بندهای رخداد گسسته بسته نرم افزاری NS می پردازیم. همانگونه که پیشتر نیز در فصلهای قبل گفتیم، مهمترین استفاده کننده زمان‌بند رخداد، اجزاء شبکه هستند که تأخیر پردازش بسته ها یا زمان سنجی های مورد نیاز را اجرا می‌کنند. شکل ۲-۵ نشان می‌دهد که هر شئ شبکه از یک زمان‌بند رخداد استفاده می‌کند. توجه کنید که شئ شبکه که یک رخداد را ایجاد می‌کند، همان عنصری است که بعداً (در زمان تعیین شده) رخداد را پردازش می کند. همچنین توجه داشته باشید که مسیر داده بین اشیاء شبکه از مسیر رخداد مجزا است. در واقع بسته ها از یک شئ شبکه به شئ دیگر توسط متد (تابع):

*send(Packet *p) {target_->recv(p)};*

ارسال و توسط متد (تابع):

Recv(Packet, Handler * h=0)*

دریافت می شوند.



شکل ۲-۵- زمان‌بند رخداد گسسته

در بسته نرم افزاری NS دو نوع زمان‌بند رخداد مختلف پیاده سازی شده است.این دو زمان‌بند، بلادرنگ[2] و غیربلادرنگ هستند. در مورد زمان‌بندی غیر بلادرنگ سه نوع پیاده سازی قابل

1 -Agent
2 -Real-time

استفاده است. این سه پیاده سازی عبارتنداز: لیست، هرم[1]، و تقویم[2] ولی از نظر منطقی تمام آنها یک کار را انجام می دهند. این تعدد پیاده سازی به دلیل حفظ و پشتیبانی سازگاری با نگارشهای[3] قبلی این نرم افزار است. در این میان زمان‌بندی تقویمی، زمان‌بند پیش فرض است. زمان‌بندهای بلادرنگ برای شبیه‌سازی های سخت افزاری [4] مورد استفاده قرار می گیرد. استفاده از این نوع زمان‌بند به شبیه ساز اجازه می دهد که با یک شبکه واقعی ارتباط برقرار نماید. در حال حاضر شبیه‌سازی سخت افزاری و ارتباط با ترافیک و جریان داده ها در یک شبکه واقعی در دست طراحی و پیاده سازی (در حال توسعه)است. با این وجود نگارش آزمایش آن قابل استفاده است. عبارات زیر نحوه استفاده از یک زمان‌بند رخداد مشخص را نشان می‌دهد.

```
…
set ns [new simulator]
$ns use-scheduler Heap
…
```

کاربرد دیگر زمان‌بند رخداد، همانگونه که از نامش بر می آید، زمان‌بندی رخدادهای شبیه‌سازی است. به عنوان مثال شروع یک کاربرد انتقال فایل مبتنی بر پروتکل FTP، تعیین زمان خاتمه شبیه‌سازی، ایجاد سناریوی شبیه‌سازی قبل از اجرای آن، مواردی هستند که توسط زمان‌بند رخدادها انجام می شود. یک شئ زمان‌بند رخداد دارای متدهایی برای زمان‌بندی شبیه‌سازی است. به عنوان مثال at *time "String"* در زمان مشخص شده یک رخداد ویژه به نام AtEvent را صادر می‌کند. یک "AtEvent" در واقع یک کلاس فرزند (زیرکلاس) از شئ Event است که تنها دارای یک متغیر اضافی به منظور نگهداری رشته کاراکتری فرمان "String" است. هنگامی که شبیه‌سازی شروع می‌شود و زمان تنظیم شده برای یک رخداد موجود در صف فرا می رسد، AtEvent به پردازشگر مناسب ارسال می شود. این پردازشگر یک بار ایجاد شده و تمام رخدادهای از این نوع را پردازش می کند. مثال زیر را در نظر بگیرید:

```
…
set ns [new simulation]
$ns use-scheduler Heap
$ns at 300.5 “Complete_sim”
…
…
proc Complete_sim { } {
…
…
}
```

از مثال فوق چنین بر می آید که متد at *time "String"* از متدها (توابع) شئ شبیه ساز است. ولی به خاطر داشته باشید که شئ شبیه ساز تنها به عنوان یک رابط کاربر عمل می‌کند و متدها و توابع اشیاء شبکه یا یک شئ زمان‌بند را فرا می‌خواند و این اشیاء به نوبه خود پردازش

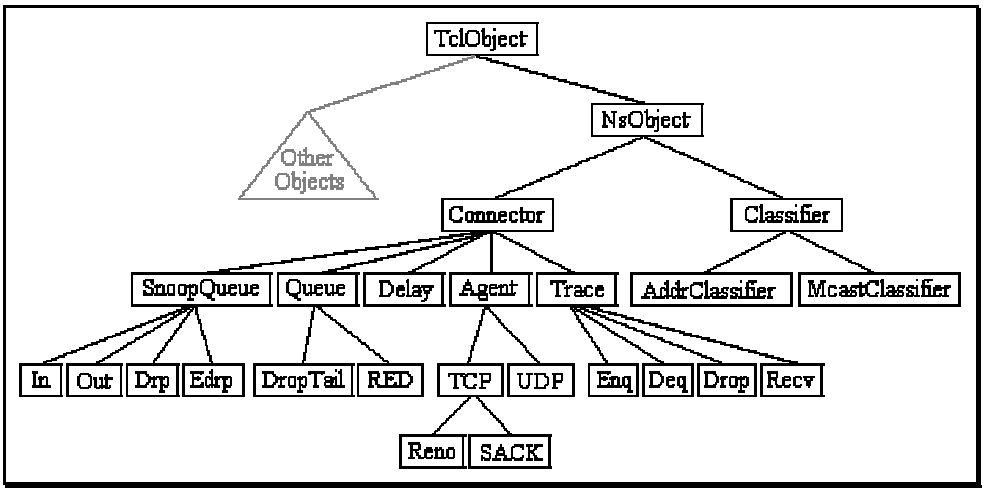
1 -Heap
2 -Calendar
3 -Backward Compatibility
4 -Emulation

اصلی و مورد نیاز را انجام می‌دهند. فهرست زیر بخشی از توابع شبیه ساز را نشان می‌دهد که با توابع زمانبند در ارتباط هستند:

Simulator instproc now	# زمان جاری را در قالب نگارشی زمان‌بند باز می گرداند.
Simulator instproc at args	# اجرای کد را در زمان مشخص شده زمان‌بندی می کند.
Simulator instproc ar-now args	# اجرای کد را در زمان فعلی زمان‌بندی می کند.
Simulator instproc after n args	# اجرای کد را بعد از گذشت n ثانیه زمان‌بندی می کند.
Simulator instporc run args	# شروع زمان‌بند
Simularot instproc halt	# خاتمه (خاتمه موقت) زمان‌بند

۲-۴-اجزاء شبکه

در این بخش به بررسی و مطالعه اجزا ساختمانی و به طور مشخص اجزاء شبکه در بسته نرم افزاری NS می پردازیم. شکل ۲-۶ بخشی از سلسله مراتب رده ها (کلاسهای) OTcl در بسته نرم‌افزاری NS را نشان می دهد. این شکل به درک اجزاء بنیادی شبکه در این نرم افزار کمک می‌نماید. برای آگاهی از ساختار کامل رده های موجود در این نرم افزار به مرجع ۶ مراجعه نمایید.



شکل ۲-۶- بخشی از ساختار درختی کلاسها در نرم افزار NS

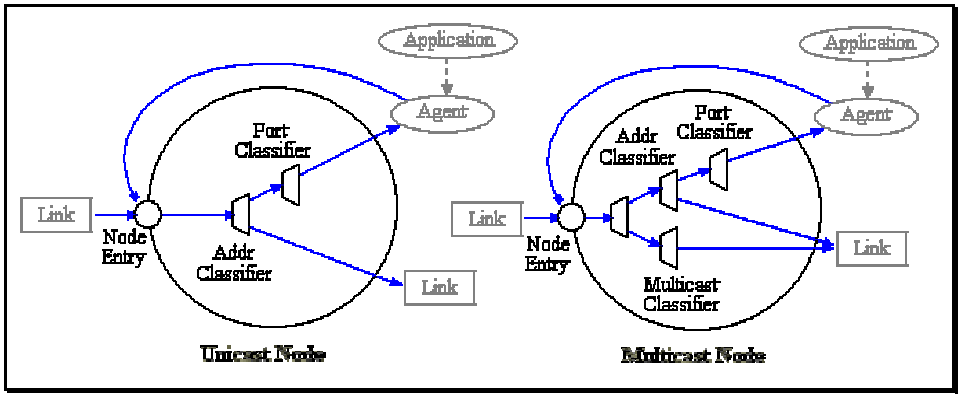
همانگونه که در این شکل ملاحظه می شود ریشه (جد) این ساختار سلسله مراتبی، کلاس TclObject است و سایر اشیاء (زمان‌بند، اجزاء شبکه، زمان سنج و حتی سایر اشیاء از جمله اشیاء مرتبط با NAM) از این کلاس منشعب شده اند. از سوی دیگر کلاس NsObject نیز والد تمام اشیاء اجزاء پایه شبکه است که پردازش بسته‌ها را بر عهده دارد و برخی از آنها نیز مانند گره و پیوند از نوع اشیاء ترکیبی هستند. اجزاء مقدماتی شبکه نیز به نوبه خود با توجه به تعداد مسیرهای داده خروجی به دو زیر کلاس اتصالات[1] و تفکیک کنندگان[2] یا متمایزگرها تقسیم می شوند. اشیاء پایه

1 -Connector
2 -Classifiers

شبکه که تنها دارای یک مسیر خروجی داده هستند در زیر کلاس اتصالات قرار گرفته و اشیاء راه گزینی که مسیرهای خروجی متعدد دارند (به عنوان مثال سوئیچها) تحت رده تفکیک‌کنندگان (متمایزگرها) جای می‌گیرند.

۲-۴-۱-گره و مسیریابی

یک گره یک شئ مرکب است. گره از یک ورودی گره ویک تفکیک‌کننده(متمایزگر) تشکیل می‌شود. شکل ۲-۷ یک گره را نشان می دهد. در بسته نرم افزاری NS دو نوع گره تک‌پراکنی[1] و چندپراکنی[2] وجود دارد. یک گره تک پراکنی یک تفکیک‌کننده(متمایزگر) آدرس دارد که مسیریابی تک پراکنی را انجام می دهد و یک تفکیک‌کنندۀ درگاه دارد. یک گره چند پراکنی علاوه بر این یک طبقه بندی کننده دارد که بسته های چندپراکنی را از بسته های تک پراکنی تفکیک می‌کند. همچنین یک تفکیک‌کنندۀ چند پراکنی دارد که مسیریابی چند پراکنی را انجام می‌دهد.



شکل ۲-۷- گره تک پراکنی و چند پراکنی

در بسته نرم افزاری NS گره های پیش فرض از نوع تک پراکنی هستند. برای ایجاد یک گره چند پراکنی کاربر می بایست به طور صریح در دست نویس OTcl ، دقیقاً پس از ایجاد شئ زمان‌بند، بیان نماید که تمام گره هایی را که ایجاد می شوند از نوع چند پراکنی خواهند بود. پس از تعیین نوع گره کاربر می تواند پروتکل مسیریابی خاصی را نیز به جای استفاده از پروتکل مسیریابی پیش فرض برای آن گره انتخاب نماید. پروتکلهای مسیریابی برای گره های چند پراکنی و تک پراکنی به صورت زیر خواهند بود:

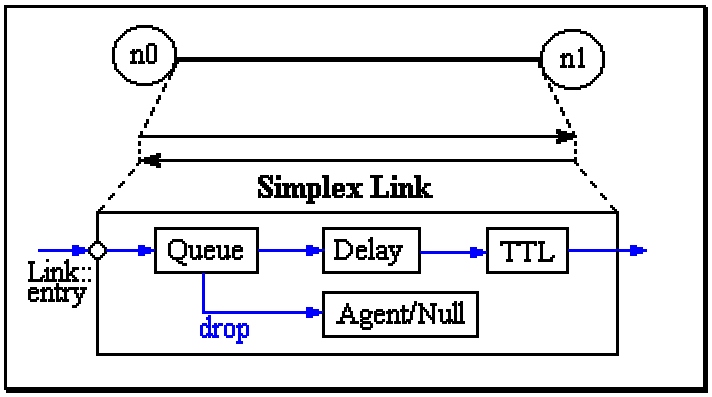
- Unicast:
  - *$ns* rtproto *type*
  - *type* : Static, Session, DB, cost, multi-path
- Multicast:
  - *$ns* multicast (right after set *$ns* [new Scheduler])
  - *$ns* mrtproto *type*
  - *type* : CtrMcast, DM, ST, BST

1 -Unicast
2 -Multicast

برای اطلاعات بیشتر در خصوص مسیریابی به مستندات بسته نرم افزاری NS مراجعه کنید.

۲-۴-۲- پیوند[1]

یکی دیگر از اشیاء مهم و اساسی در بسته نرم افزاری NS ، پیوند است. همانگونه که در شکل ۲-۸ دیده می شود، هنگامی که کاربر یک پیوند دو طرفه را با استفاده از متد(تابع) شئ شبیه ساز ایجاد می کند، در واقع دو پیوند ساده یک طرفه در هر یک از دو جهت مسیر انتقال داده ایجاد می‌گردند.



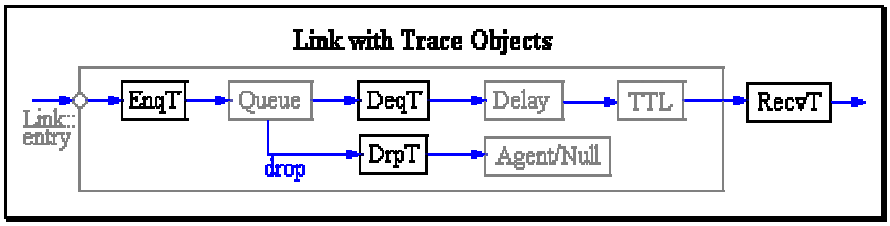
شکل۲-۸- پیوند

نکته مهم و قابل توجه این است که صف خروجی در یک گره نیز در واقع به عنوان بخشی از پیوند ساده یک طرفه پیاده سازی می‌شوند. بسته هایی که از صف خارج می شوند وارد شئ تأخیر شده تا تأخیر پیوند ارتباطی برروی آنها اعمال شود و بسته هایی که از صف سرریز کنند وارد کارگزار خاصی موسوم به کارگزار تهی (پوچ) [2] شده و آزاد می شوند. در نهایت شئ TTL[3]، پارامتر TTL را برای هر بسته دریافت شده محاسبه و روزآمد می‌سازد.

دربسته نرم افزاری NS فعالیتهای شبکه در پیوندهای ارتباطی یک طرفه ردیابی[4] می‌شوند. اگر در هنگام برپایی شبکه از شبیه ساز بخواهیم که ردیابی را فعال نماید (این کار را می توان با استفاده از دستور $ns trace-all file یا دستور$ns namtrace-all file انجام داد) پیوندهای ایجاد شده پس از صدور این فرمان دارای اشیایی برای ردگیری ترافیک نیز خواهند بود. شکل ۲-۹ یک پیوند دارای ردگیری را نشان می دهد. علاوه بر این کاربران می توانند با استفاده از فرمان create-trace بین یک مبدأ و مقصد مورد نظر ردیابی خاصی را انجام دهند. شکل کلی این فرمان به این صورت است:

Create-trace {*type file src dst*}

1 -Link
2 -Null Agent
3 -Time To Live
4 -Trace

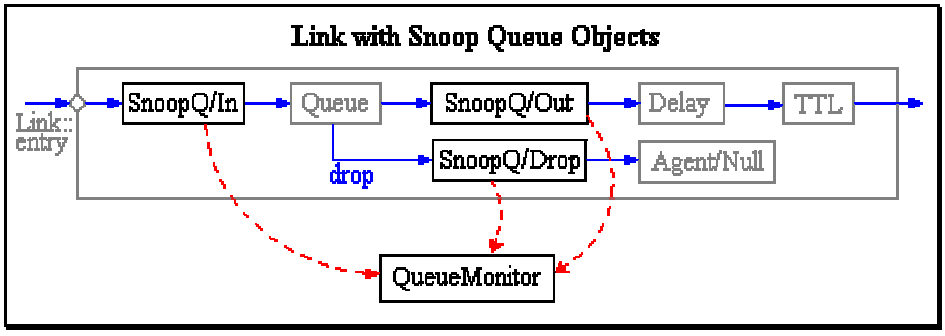


شکل ۲-۹- پیوند به همراه اشیاء ردیابی

هنگامی که هریک از اشیاء ردگیری (مانند EnqT ، DepT، DrpT و RecvT) بسته‌ای را دریافت می کنند، گزارش دریافت را در فایل ردیابی ثبت کرده و بدون مصرف زمان شبیه‌سازی (تأخیر)بسته را به شئ بعدی شبکه ارسال می کنند. در بخش تحلیل نتایج به ساختار وقایع ثبت شده در فایل ردگیری می پردازیم.

۲-۴-۳-ناظر صف[1]

اساساً اشیاء ردگیری برای ثبت زمان ورود بسته ها در جایی طراحی شده اند که این اشیاء قرار گرفته اند. با این وجود ممکن است کاربر مایل باشد اطلاعات بیشتری در مورد آنچه در داخل صف می گذرد بدست آورد. به عنوان مثال یک کاربر علاقه مند در مورد رفتار صفهای RED[2] ، ممکن است بخواهد میانگین اندازه صف و اندازه جاری صف را اندازه گیری کند. در واقع وی به یک ناظر صف احتیاج دارد. نظارت بر صف همانگونه که در شکل ۲-۱۰ ملاحظه می شود با استفاده از اشیاء ناظر صف حاصل می‌شود.



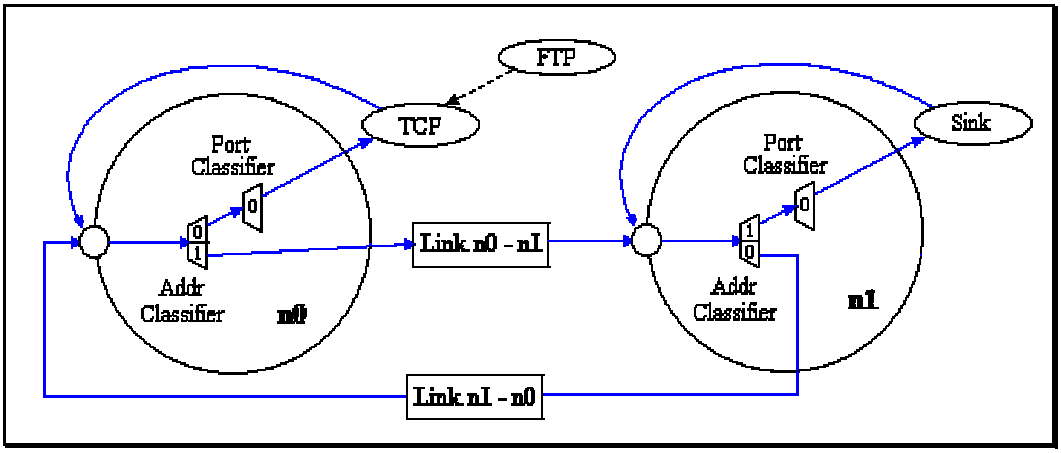
شکل ۲-۱۰- نظارت بر صف

هنگامی که یک بسته وارد می‌شود، شئ "Snoop Queue" این رخداد (ورود بسته) را به ناظر صف گزارش می‌دهد. ناظر صف با استفاده از این اطلاع بر صف نظارت می‌کند. در مثال نظارت صف RED یک ناظر از این نوع صف به کار گرفته شده است. (به بخش ۳-۲ مراجعه کنید) توجه کنید که نظارت بر صف می تواند در کنار ردیابی پیوند یا سایر اشیاء ردیابی انجام پذیرد.(ولی در شکل فوق اشیاء ردیابی مشاهده نمی شوند)

1 -Queue Monitor
2 -Random Early Detection

۲-۴-۴-مثالی از جریان بسته

تا به اینجا دو جزء از مهمترین اجزاء شبکه یعنی گره و پیوند بررسی شدند. شکل۲-۱۱ عناصر ساختمانی یک سناریوی شبیه‌سازی شبکه و جریان بسته ها را نشان می‌دهد. این شبکه از دو گره تشکیل شده است (n0 و n1) و آدرس شبکه این دو گره (در شبیه‌سازی) به ترتیب 0 و 1 است. یک کارگزار TCP به گره صفر متصل شده است که با استفاده از درگاه صفر با شئ گیرنده TCP متصل به گره یک تبادل داده ها را انجام می دهد. در نهایت نیز یک کاربرد انتقال فایل FTP (یک مولد ترافیک) به کارگزار TCP متصل شده است.



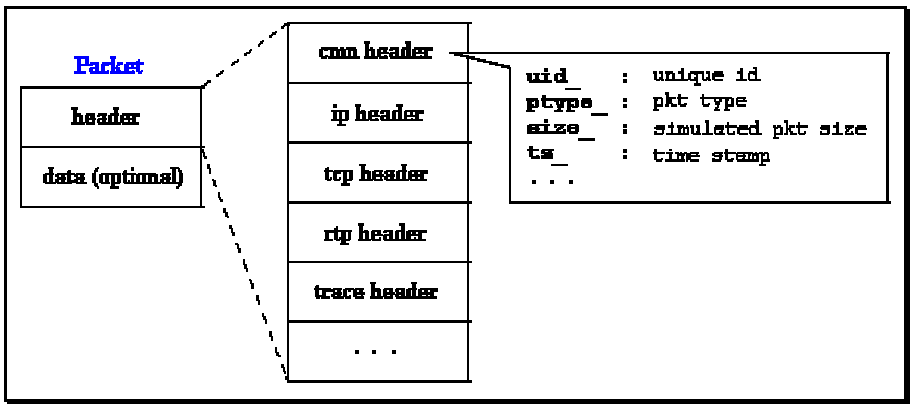
شکل ۲-۱۱- مثالی از جریان بسته ها

در این شکل مولد ترافیک FTP یک بسته را به مقصد گره ۱ (درگاه صفر) تولید می‌کند و این بسته به کارگزار TCP رفته و از آنجا توسط تفکیک کننده آدرس با توجه به اینکه برای گره ۱ آدرس دهی شده است وارد پیوند n0-n1 می شود. پس از سپری شدن تأخیر انتقال برروی پیوند مذکور، بسته وارد گره ۱ شده و در آنجا تفکیک کننده آدرس مجدداً بسته را دریافت و به درگاه صفر گیرنده هدایت می کند که یک TCP Sink می باشد. TCP Sink تصدیق بسته را به سوی گره صفر تولید می کند و این بسته توسط تفکیک کننده آدرس در گره ۱ وارد مسیر ارتباطی (پیوند) n1-n0 می شود. و نهایتاً توسط تفکیک کننده آدرس در گره صفر به کارگزار TCP می رسد و این روال برای سایر بسته ها ادامه می یابد.

توجه کنید که در این شکل رفتار واقعی پروتکل FTP برروی لایه انتقال TCPشرح داده نشده است و تنها هدف از توضیحات فوق شرح جزئیات داخلی یک مدل شبکه و جریان بسته ها در نرم افزار شبیه‌سازی NS بوده است.

۲-۵- بسته

در بسته نرم افزاری NS یک *بسته* از مجموعه ای از سرآیندها[1] و یک فضای اختیاری داده تشکیل شده است. (به شکل ۲-۱۲ نگاه کنید) همانگونه که پیشتر در بخش مثالهای ساده شبیه‌سازی گفتیم مقدار دهی آغازین یک بسته و قالب آن در هنگامی صورت می پذیرد که شئ *شبیه ساز* ایجاد می گردد. در این هنگام پشته ای از سرآیندهای ثبت شده (یا قابل استفاده) نظیر سرآیند IP ، سرآیندTCP، سرآیند RTP [2] و سرآیندهای ردیابی تعریف شده و محل شروع هر سرآیند ثبت می شود. به عبارت دیگر صرفنظر از اینکه آیا یک سرآیند خاص مورد استفاده قرار گیرد یا خیر، هنگامی که کارگزاری بسته‌ای را تخصیص دهد، مجموعه ای (پشته ای) از سرآیندها ثبت شده ایجاد شده و یک شئ شبکه با استفاده از محل شروع هر سرآیند در بسته (که قبلاً ثبت شده است و نقش آدرس شروع را ایفا می‌کند) می تواند دسترسی یا پردازشهای مورد نیاز را برروی بسته و سرآیند مورد نیاز انجام دهد.



شکل ۲-۱۲- ساختار بسته ها در NS

در اغلب موارد یک بسته تنها مجموعه ای از سرآیندها را در بر دارد و فضای داده در آن تهی است. لیکن یک بسته می‌تواند داده های واقعی پروتکل کاربرد را نیز با تخصیص فضای مورد نیاز در بر بگیرد. کارگزاران[3] و کاربردهایی که از این قابلیت استفاده می کنند، زیاد نیستند. اگر قصد دارید کاربردی را پیاده سازی کنید که در بستر شبکه با کاربرد دیگر صحبت (تبادل داده) نماید، می‌توان با استفاده از این فضای داده و تغییرات جزئی در پیاده سازی کارگزار زیرین و مربوطه به این هدف نائل گردید. رهیافت ممکن دیگر ایجاد یک سرآیند جدید برای کاربرد مورد نظر و اعمال تغییرات در کارگزار زیرین، به منظور نوشتن داده‌های دریافتی در سرآیند جدید خواهد بود. از این رهیافت در مثالهای فصلهای آتی استفاده خواهیم کرد. (به بخش ۴-۲-۶ مراجعه کنید.)

1 -Headers

2- پروتکل UDP از سرآیند RTP استفاده می کند.

3 -Agents

۳- پس از شبیه‌سازی

در این بخش به بررسی مثالهایی می پردازیم که در آنها پس از انجام شبیه‌سازی تحلیل و پردازش نتایج نیز انجام شده است. در واقعی یکی از اهداف اساسی شبیه سازی ارزیابی کارایی و محدودیتهای احتمالی روشی یا ایده ای است که محقق در حال بررسی آن می‌باشد. لذا پس از برپایی سناریوی شبیه سازی می‌بایست نتایج شبیه سازی مورد مطالعه و بررسی قرار گیرند.

۳-۱-مثال تحلیل ردیابی

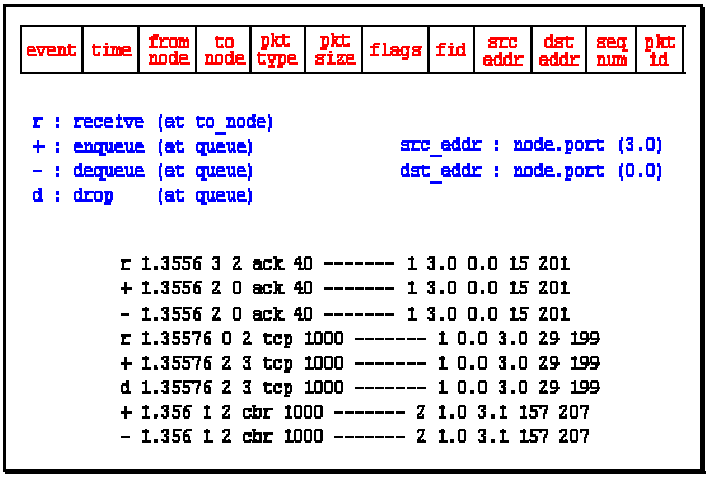
این بخش مثالی از تحلیل عملیات ردیابی[1] را نشان می دهد. مثال ۳-۱ همان دست نویس OTcl در بخش ۲-۲ است با این تفاوت که خطوطی برای ایجاد فایل ردیابی و نوشتن اطلاعات ردیابی به آن اضافه شده است. برای اجرای این دست نویس متن آن را که در پیوست ۲ آمده است را به عنوان پارامتر برنامه NS معرفی نمایید.

```
…
#Open the NAM trace file
set nf [open out.nam w]
$ns namtrace-all $nf

#Open the Trace file
set tf [open out.tr w]
$ns trace-all $tf

#Define a 'finish' procedure
proc finish {} {
        global ns nf tf
        $ns flush-trace
        #Close the NAM trace file
        close $nf
        #Close the Trace file
        close $tf
        #Execute NAM on the trace file
        exec nam out.nam &
        exit 0
}
…
```

شکل ۳-۱-نمونه فعال کردن ردیابی در دست نویس OTcl

با اجرای دست نویس فوق یک فایل ردیابی (با ساختار مناسب برای برنامه NAM) ایجاد می‌شود. همچنین فایل ردیابی دیگری (تحت نام "out.tr")برای تحلیل نتایج شبیه‌سازی توسط این دست نویس تولید می شود. شکل ۳-۲ ساختار فایل ردیابی و نمونه اطلاعات خروجی این مثال را نشان می‌دهد.

1 -Trace



شکل ۳-۲- ساختار ردیابی

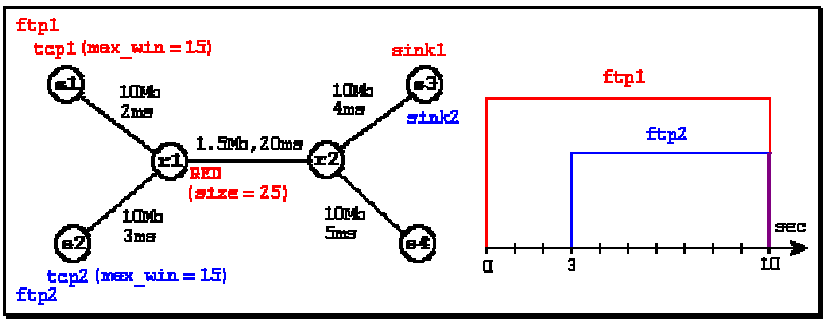
هر خط ردیابی با یک نماد، به عنوان توصیف گر رخداد (+, -, d, r) شروع می‌شود. سپس زمان رخداد بر حسب ثانیه و مبتنی بر زمان شبیه‌سازی ثبت می‌گردد. به دنبال آن شماره گره فرستنده و گیرنده می آید که از طریق آن دو پیوندی که رخداد برروی آن بروز کرده شناسایی می‌گردد.(به شکل ۲-۴ رجوع کنید) اطلاعات بعدی نوع بسته و اندازه آن برحسب بایت است. فیلد بعدی پرچم[1]است که در این شکل به صورت"-------" نشان داده شده است. در حال حاضر پرچمی که در NS پیاده سازی شده و پشتیبانی می گرددECN[2] است. فیلد بعدی شماره شناسایی جریان[3] یا fid است. با استفاده از این فیلد در پروتکل IPv6 کاربر می تواند هریک از جریانهای داده را از یکدیگر مجزا نماید. در صورتی که کاربرد فوق مورد نظر نباشد کاربر می تواند از این فیلد برای مقاصد تحلیل نتایج استفاده کند. این فیلد در محیط NAM در حکم شماره شناسایی رنگی است که جریان داده با آن رنگ نمایش داده می شود. دو فیلد بعدی نشانی فرستنده و گیرنده در قالب ***گره.درگاه*** [4]است. فیلد بعدی شماره ترتیب بسته های پروتکل شبکه یا انتقال است. توجه داشته باشید که علیرغم عدم وجود شماره ترتیب در پروتکل UDP، NS برای مقاصد اندازه گیری و تحلیلی، به این بسته ها نیز شماره ترتیب اختصاص می دهد. آخرین فیلد شماره منحصر به فردی است که هر بسته در NS خواهد داشت. با در دست داشتن نتایج یک شبیه‌سازی کاربر قادر خواهد بود با استفاده از برنامه های کمکی و دست نویسهای ساده و انتقال آن به محیطهای آماری و یا صفحات گسترده[5] نمودارهای گرافیکی مورد نظر را تهیه نماید.

در این بخش مثالی را دیدیم که در آن چگونگی ایجاد فایل ردیابی در NS و تفسیر و استخراج نتایج از آن ارائه شد. در مثال بعد موردی را مطالعه می کنیم که فرآیند پردازشهای پس از شبیه‌سازی در داخل دست نویس OTcl گنجانده شده است.

1 -Flag
2 -Explicit Congestion Notification
3 -Flow ID (Identification)
4 -Node.Port
5 -Spread sheet

۳-۲- مثالی از ناظر صف *RED*[1]

در این بخش به بررسی مثالی می پردازیم که در آن از یک ناظر صف استفاده شده است. شکل ۳-۳ مدل شبکه مورد نظر ما را نشان می دهد که با استفاده از یک دست نویس OTcl همبندی شبکه را برپاکرده و سناریوی شبیه‌سازی را اجرا می کند که در شکل ۳-۴ دیده می‌شود. توجه کنید که مدل صف به کار رفته در این مثال قادر است ۲۵ بسته را در خود نگهداری کند و برای پیوند ارتباطی r1-r2 در نظر گرفته شده است. هدف از این شبیه‌سازی مطالعه رفتار صف RED از طریق اندازه گیری پویای *اندازه جاری صف* و *میانگین طول صف* خواهد بود. برای اجرای این شبیه‌سازی می توانید دست نویس پیوست ۳ را به عنوان پارامتر خط فرمان ("ns red.tcl")به NS بفرستید.



شکل ۳-۳- مثالی از مدل شبکه برای مطالعه صف RED

```
set ns [new Simulator]
set node_(s1) [$ns node]
set node_(s2) [$ns node]
set node_(r1) [$ns node]
set node_(r2) [$ns node]
set node_(s3) [$ns node]
set node_(s4) [$ns node]


$ns duplex-link $node_(s1) $node_(r1) 10Mb 2ms DropTail
$ns duplex-link $node_(s2) $node_(r1) 10Mb 3ms DropTail
$ns duplex-link $node_(r1) $node_(r2) 1.5Mb 20ms RED
$ns queue-limit $node_(r1) $node_(r2) 25
$ns queue-limit $node_(r2) $node_(r1) 25
$ns duplex-link $node_(s3) $node_(r2) 10Mb 4ms DropTail
$ns duplex-link $node_(s4) $node_(r2) 10Mb 5ms DropTail

...

set tcp1 [$ns create-connection TCP/Reno $node_(s1) TCPSink
$node_(s3) 0]
$tcp1 set window_ 15
set tcp2 [$ns create-connection TCP/Reno $node_(s2) TCPSink
$node_(s3) 1]
$tcp2 set window_ 15
set ftp1 [$tcp1 attach-source FTP]
set ftp2 [$tcp2 attach-source FTP]


# Tracing a queue
set redq [[$ns link $node_(r1) $node_(r2)] queue]
set tchan_ [open all.q w]
```

1 -Random Early Detection

```
$redq trace curq_
$redq trace ave_
$redq attach $tchan_


$ns at 0.0 "$ftp1 start"
$ns at 3.0 "$ftp2 start"
$ns at 10 "finish"


# Define 'finish' procedure (include post-simulation processes)
proc finish {} {
    global tchan_
    set awkCode {
      {
          if ($1 == "Q" && NF>2) {
              print $2, $3 >> "temp.q";
              set end $2
          }
          else if ($1 == "a" && NF>2)
          print $2, $3 >> "temp.a";
      }
    }
    set f [open temp.queue w]
    puts $f "TitleText: red"
    puts $f "Device: Postscript"

    if { [info exists tchan_] } {
      close $tchan_
    }
    exec rm -f temp.q temp.a
    exec touch temp.a temp.q

    exec awk $awkCode all.q

    puts $f \"queue
    exec cat temp.q >@ $f
    puts $f \n\"ave_queue
    exec cat temp.a >@ $f
    close $f
    exec xgraph -bb -tk -x time -y queue temp.queue &
    exit 0
}

$ns run
```

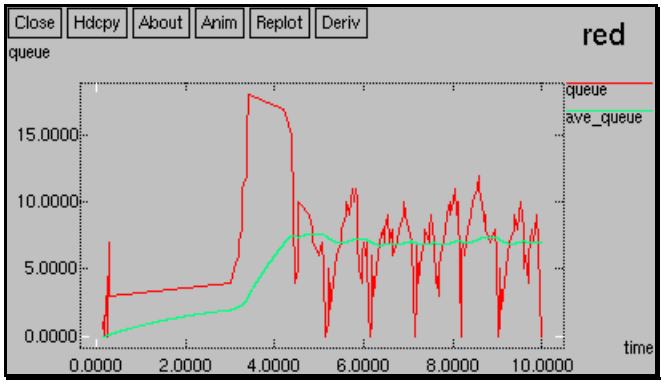
شکل ۳-۴-سناریوی شبیه سازی

در این مثال موارد جالب توجهی وجود دارد. اولین نکته آن است که برای ایجاد یک ارتباط مبتنی بر پروتکل TCP از متد (تابع)پیشرفته *create-connection* استفاده شده است. نکته دوم بررسی دقیق قسمتی است که برروی صف RED نظارت می کند. این کار با استفاده از متغیری انجام شده است که به شئ صف RED اشاره کرده و توابع ردیابی آن را برای نظارت بر اندازه جاری صف، *Curq_* و میانگین اندازه صف *avg_* فرا می خواند. سپس نتایج در فایل ردیابی به نام "all.q" ثبت می‌شوند.

ساختار خروجی ردیابی صف برای میانگین اندازه صف و اندازه جاری صف به شکل زیر است:

```
a       time    avg_q_size
Q       time    crnt_q_size
```

پس از خاتمه ردیابی و اجرای شبیه‌سازی فایل ردیابی "all.q" بسته شده و تنها کار باقیمانده نوشتن رویه ای است که پس از شبیه‌سازی نتایج حاصل را در قالب نمودار گرافیکی ترسیم نماید. در این مثال از ابزارهای awk و Xgraph برای تولید نمودار مورد نظر استفاده شده است و نتیجه کار در شکل ۳-۵ مشاهده می‌شود.



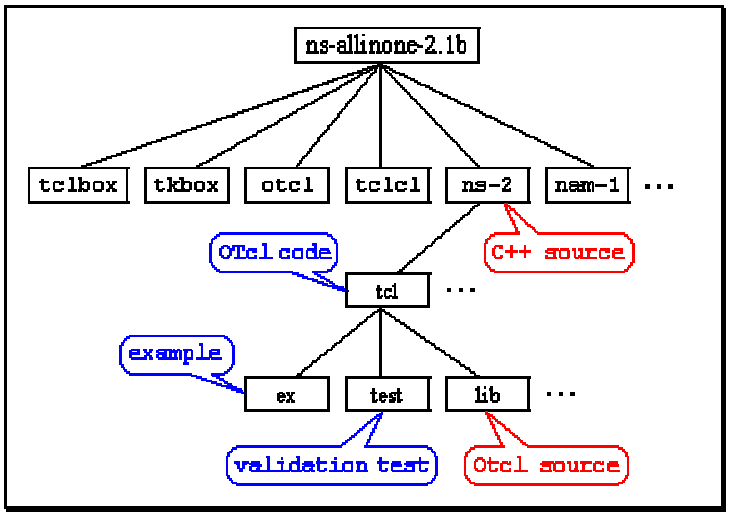
۳-۵-نمودار ردیابی صف RED

۴-توسعه بسته نرم افزاری NS

در این بخش به بررسی ساختار فایلها در بسته نرم افزاری NS پرداخته و به دنبال آن به معرفی روشها و تکنیکهای مربوط به گسترش این ابزار شبیه‌سازی اشاره خواهیم کرد.

۴-۱- چه چیزی در کجاست؟

قبل از آنکه وارد بحث گسترش و توسعه NS شویم لازم است تا به اختصار به بررسی فایلها و فهرستها[1] و اطلاعاتی که در بردارند بپردازیم. شکل ۴-۱ بخشی از ساختار سلسله مراتبی فهرستهای شبیه ساز را نشان می دهد[2].



شکل ۴-۱- بخشی ساختار فهرستهای بسته نرم افزاری NS

در میان تمام زیر فهرستهای این بسته نرم افزاری، زیرفهرست ns-2 محلی است که تمام پیاده سازی شبیه ساز (به زبان C++ یا OTcl) در آنجا ذخیره شده است. علاوه بر آن دست نویسهای OTcl به منظور آزمون اعتبار سنجی NS و دست نویسهای نمونه نیز در این فهرست قرار دارند. در واقع در داخل این فهرست، زیرفهرستی به نام tcl وجود دارد که برنامه‌های دست نویس OTcl (مثالها و آزمونهای اعتبار سنجی) در آن ذخیره شده‌اند. اغلب برنامه‌های نوشته شده به زبان C++ که پیاده سازی زمان‌بند رخداد و اجزاء پایهٔ شبکه (به استثناء موارد مرتبط با تورجهان گستر[3]) در زیر فهرست اصلی قرار دارند. به عنوان نمونه اگر بخواهید پیاده سازی کارگزار UDPرا مشاهده نمایید کافی به زیر فهرست"ns-allinone-2.1b/ns-2" مراجعه کرده و در آنجا فایلهای "udp.h" و "udp.cc" را باز کنید. برای اطلاع از ساختار سلسله مراتبی اجزاء شبکه به شکل ۲-۶ مراجعه کنید. در این بخش از این به بعد فرض می‌کنیم که زیر فهرست جاری، "ns-allinone-2.1b" می باشد.

1 -Directory

2-در این شکل فرض شده است که کاربر از بسته نرم افزاری NS نگارش 2.1b ("ns-allinone-2.1b") استفاده می نماید.

3 -World Wide Web (WWW)

از زیرفهرست Tcl، زیرفهرستهای دیگری نیز منشعب می شود. از میان این زیرفهرستها، lib زیرفهرستی است که متن دست نویسهای OTcl در خصوص اغلب اجزاء بنیادی و پایه ای پیاده سازی NS (نظیر کارگزار، گره، پیوند، بسته، نشانی (آدرس)، مسیریابی و نظایر آن) را در بر دارد. این زیرفهرست محلی است که کاربران و برنامه نویسان و توسعه دهندگان NS مراجعات زیادی به آن خواهند داشت. توجه داشته باشید که دست نویسهای OTcl برای پیاده سازی شبکه های محلی (LAN) ، تورجهان گستر(Web) و چندپراکنی[1] در زیرفهرستهای جداگانه ای قرار دارند. برخی از مهمترین فایلهای موجود در زیر فهرست "ns-2/tcl/lib" عبارتند از:

ns-lib.tcl : رده (کلاس)[2] شبیه ساز و اغلب متدها (توابع) آن به استثناء شبکه های محلی، تورجهان گستر، و چند پراکنی در این فایل قرار دارند. اگر بخواهد با توابع و متدهای شئ شبیه ساز آشنا شوید و نحوه عملکرد آنها را بررسی نمایید بهترین و مناسبترین محل این فایل خواهد بود.

ns-default.tcl : مقادیر پیش فرض پارامترهای قابل پیکربندی و مربوط به اجزاء شبکه در این فایل گردآوری شده اند. نظر به اینکه اغلب اجزاء شبکه به زبان برنامه سازی C++ پیاده سازی شده اند، پارامترهای قابل پیکربندی در واقع متغیرهای C++ هستند که از طریق پیوند OTcl [3] در اختیار دست نویسهای OTcl نیز قرار دارند. در بخش بعد در خصوص پیوند بین برنامه‌های C++ و دست نویسهای OTcl نکات کامل تری را مطرح می نماییم.

ns-packet.tcl : قالب سرآیند بسته‌ها و پیاده سازی و مقدار دهی آغازین آنها در این فایل انجام می شود. هنگامی که یک سرآیند جدید می سازید، می بایست آن سرآیند را در این فایل ثبت نموده و فرآیندها و روالهای لازم جهت درج آن در مجموعه سرآیندها را نیز بنویسید. مثالی از ایجاد یک سرآیند جدید در بخش *"افزودن یک کاربرد وکارگزار جدید"* قابل بررسی است.

سایر فایلهای OTcl : سایر فایلهای موجود در این زیرفهرست مربوط به پیاده سازی اجزاء ترکیبی شبکه و بخشهای کنترلی اشیاء شبکه به زبان C++ است. کاربرد FTP تماماً به زبان OTcl پیاده سازی شده است و متن آن در فایل "ns-source.tcl" قرار دارد.

دو زیرفهرست جالب دیگر برای کاربرانی که مایل به آشنایی با فرآیند طراحی یک شبیه‌سازی مشخص هستند، زیرفهرستهای ex و test است. زیرفهرست ex مثالهای متعددی از دست نویس‌های شبیه‌سازی را در بردارد و زیرفهرست test دست نویسهایی است که به منظور آزمون و اعتبار سنجی نگارش NS نصب شده برروی کامپیوترشما و اطمینان از صحت نصب آن مورد استفاده قرار می‌گیرد. این دست نویسها شبیه‌سازیهای متعددی را انجام داده و نتایج حاصل شده بر روی کامپیوتر شما را

1 -Multicast
2-Class
3 -Bind (*C++_variable_name , OTcl_variable_name)*

با نتایج معتبر مورد انتظار مقایسه می نماید. به این ترتیب در صورت کسب نتایج مشابه می توان از نصب صحیح بسته نرم افزاری NS برروی کامپیوتر خودتان اطمینان حاصل نمایید.

۴-۲- پیوند OTcl

با توجه به اینکه رده های اشیاء مورد نظر در مسیر داده به منظور افزایش کارایی می‌بایست به زبان برنامه‌سازی C++ نوشته شوند، لذا در هنگام گسترشNS از طریق افزودن یک شئ مقدماتی شبکه، معمولاً به پیوند بین C++ و OTcl نیاز خواهد داشت. در این بخش در قالب مثالهای متعدد پیوندهای موجود بین برنامه های C++ و دست نویسهای OTcl (پیوند C++/OTcl) را معرفی می‌کنیم. دراین بخش مثال ساده ای به نام "MyAgent" را معرفی می کنیم. این کارگزار به صورت نمونه بوده و رفتار یک کارگزار واقعی (یعنی ایجاد و ارسال بسته ها) را انجام نمی دهد با این وجود این فرض لطمه ای به کلیت بحث ما وارد نخواهد کرد. متن کامل این کارگزار که به زبان C++ نوشته شده است در پیوست ۴ قابل بررسی است. همچنین در انتهای این بخش یک دست نویس OTcl به منظور ارزیابی و آزمون این کارگزار[1] ارائه می شود.

۴-۲-۱- صدور[2] رده های C++ به OTcl

فرض کنید می خواهید یک رده از شئ جدید شبکه به نام MyAgent در زبان برنامه سازی C++ ایجاد کنید. این شئ را از رده Agent اقتباس می‌کنیم. همچنین می خواهیم این شئ را به گونه‌ای بسازیم که بتوانیم نمونه ای[3] از آن را در دست نویس OTcl نیز ایجاد کنیم. برای اینکار می‌بایست یک شئ ارتباطی به نام MyAgentClass بسازیم که این شئ از TclClass اقتباس شده باشد. این شئ پیوندی یک شئ OTcl (در این مثال با نام "Agent/MtAgentOTcl") ایجاد می‌کند و ارتباط بین شئ OTcl و شئ C++ (در این مثال "MyAgent") را برقرار می‌کند. این ارتباط از طریق اجرای متد (تابع)Create صورت می‌گیرد. شکل ۴-۲ تعریف MyAgent و رده ارتباط دهنده را نشان می‌دهد.

```
class MyAgent : public Agent {
public:
        MyAgent();
protected:
        int command(int argc, const char*const* argv);
private:
        int    my_var1;
        double my_var2;
        void   MyPrintFunc(void);
};
```

```
static class MyAgentClass : public TclClass {
public:
        MyAgentClass() : TclClass("Agent/MyAgentOtcl") {}
        TclObject* create(int, const char*const*) {
                return(new MyAgent());
        }
} class_my_agent;
```

شکل۴-۲- عنصر شبکه در زبان C++ و شئ پیوند

1 -MyAgent
2 -Export
3 -Instance

هنگامی که NS اجرا می شود، تابع سازنده[1] متغیر ایستای "*Class_My_Agent*" را اجرا می کند و به بنابراین یک نمونه از "MyAgentClass" ایجاد می شود. در این فرآیند رده (کلاس) "Agent/MyAgentOTcl" و متدهای مربوطه در فضای OTcl ایجاد می شوند. هر زمان که کاربر در فضای OTcl سعی کند نمونه جدیدی از این شئ را با استفاده از فرمان"new Agent/MyAgentOTcl" ایجاد کند،"MyAgentClass:create" فراخوانی شده و به طور خودکار یک نمونه از "MyAgent" را ایجاد کرده و نشانی (اشاره گر) آن را باز می گرداند. توجه داشته باشید که ایجاد یک شئ C++ از طریق OTcl به معنای آن نیست که می توانید از داخل دست نویس OTcl توابع آن شئ را فراخوانی کرده یا به متغیرهای آن دسترسی یابید.

۴-۲-۲- صدور متغیرهای رده C++ به OTcl

فرض کنید شئ جدید C++ شما، MyAgent دو متغیر پارامتری به نامهای my_var1 و my_var2 دارد و شما می خواهید این دو متغیر را به راحتی از داخل دست نویس OTcl (دست نویس ورودی شبیه‌سازی)تغییر دهید. برای اینکار می بایست از تابع انقیاد[2] استفاده کنید. این تابع را می‌بایست برای صدور هر متغیر فراخوانی نمود. تابع انقیاد یک متغیر جدید با نام داده شده (پارامتر اول) را در کلاس OTcl نظیر ("Agent/MyAgentOTcl") ایجاد می کند. علاوه بر آن ارتباط دو طرفه بین این دو متغیر را نیز فراهم می کند. شکل ۴-۳ انقیاد متغیرهای my_var1 و my_var2 را نشان می‌دهد.

```
MyAgent::MyAgent() : Agent(PT_UDP) {
        bind("my_var1_otcl", &my_var1);
        bind("my_var2_otcl", &my_var2);
}
```

شکل ۴-۳- انقیاد متغیرها

توجه کنید که توابع انقیاد در تابع سازنده "MyAgent" گنجانده شده اند به این ترتیب هنگامی که یک نمونه جدید از این شئ ایجاد شود تابع انقیاد نیز فراخوانی می شود. NS از چهار تابع انقیاد برای پنج نوع مختلف متغیرها به شرح زیر، پشتیبانی می کند:

نوع متغیر	نام تابع
متغیرهای صحیح یا حقیقی	Bind()
متغیر زمان (Time)	Bind_time()
متغیر پهنای باند(Bandwidth)	Bind_bw()
متغیر منطقی (Boolean)	Bind_bool()

1 -Constructor Function
2 -Binding

به این ترتیب کاربری که شبیه‌سازی را با دست نویسی به زبان OTcl طراحی و اجرا می‌کند می‌تواند به متغیرهای (پارامترهای پیکربندی) اجزاء شبکه که در زبان C++ پیاده سازی شده‌اند دسترسی یافته و آنها را تغییر دهد. توجه داشته باشید که هر زمان یک متغیر C++ را صادر کردید توصیه می‌شود که مقدارپیش فرض آن را در فایل "ns-2/tcl/lib/ns-lib.tcl" نیز تنظیم کنید. در غیر اینصورت هنگام ایجاد یک نمونه جدید از شئ مورد نظر، پیام هشداری نیز دریافت خواهید کرد.

۴-۲-۳- صدور فرمانهای کنترلی شئ C++ به OTcl

علاوه بر صدور اشیاء C++ به OTcl ممکن است بخواهیم کنترلهای خاصی برروی اشیاء را از داخل دست نویس OTcl انجام دهیم. به عبارت دیگر ممکن است بخواهیم یک شئ C++ را از طریق دست نویسهای OTcl کنترل کنیم. این کار با استفاده از یک متد (تابع) خاص به نام **"Command"** در شئ C++ (در این مثال MyAgent) میسر است. این تابع نقش یک مفسر فرامین OTcl را ایفا می‌کند. در حقیقت از دید کاربر، یک فرمان OTcl که در یک تابع یا متد شئC++ تعریف شده است، شبیه تابعی است که در شئ OTclمتناظر وجود دارد. شکل ۴-۴ مثالی از مفسر فرمان OTcl را نشان می دهد. در این مثال متد (تابع) commandبرای شئ MyAgent نوشته شده است.

```
int MyAgent::command(int argc, const char*const* argv) {
      if(argc == 2) {
            if(strcmp(argv[1], "call-my-priv-func") == 0) {
                  MyPrivFunc();
                  return(TCL_OK);
            }
      }
      return(Agent::command(argc, argv));
}
```

شکل ۴-۴-مفسر فرمان OTcl

هنگامی که نمونه[1] از شئ OTcl که با MyAgent منطبق است در فضای OTcl ایجاد می‌شود (به عنوان مثال از طریق فرمان:"set myagent [new Agent/MyAgentOTcl]") و کاربر سعی می‌کند یک تابع خاصی (متدی)از آن را فراخوانی کند، (به عنوان مثال:"$myagent call-my-priv-func") OTcl در میان توابع این شئ می گردد و اگر تابع فراخوانی شده پیدا نشود، تابع "MyAgent::**command**" را فراخوانی کرده و نام تابع مورد نظر را نیز به عنوان پارامتر به آن منتقل می کند. این پارامتر در قالب argc/arv منتقل می شود. به این ترتیب اگر عملیات خاصی برای این تابع OTcl تعبیه شده باشد، آن عملیات انجام می شود، و نتایج را باز می گرداند. در غیر اینصورت تابع command شئ جد[2] (والد) به صورت بازگشتی فراخوانی می شود تا تابع مورد نظر پیدا شود. در صورتی که تابع مورد بحث در هیچ یک از توابع والد نباشد، یک پیام خطا به شئ OTcl باز گردانده می شود و OTclنیز پیام خطارا به کاربر منعکس می کند. به این ترتیب یک کاربر در فضای OTcl می تواند رفتار یک شئ C++ را کنترل کند.

1 -Instance
2 -Ancestor

۴-۲-۴-اجرای یک فرمان OTcl از درون برنامه C++

هنگامی که یک شئ شبکه جدید در C++ می سازیم، ممکن است بخواهیم یک فرمان OTcl را نیز برای آن شئ اجرا کنیم. شکل ۴-۵ پیاده سازی تابع "MyPrivFunc" را به عنوان یکی از متدهای "MyAgent" نشان می دهد. این تابع مفسر OTcl را وادار می سازد تا مقادیر متغیرهای خصوصی[1] my_var1 و my_var2 را چاپ نماید.

```
void MyAgent::MyPrivFunc(void) {
        Tcl& tcl = Tcl::instance();
        tcl.eval("puts \"Message From MyPrivFunc\"");
        tcl.evalf("puts \"      my_var1 = %d\"", my_var1);
        tcl.evalf("puts \"      my_var2 = %f\"", my_var2);
}
```

شکل ۴-۵-اجرای فرمان OTcl توسط یک شئ C++

برای اجرای یک فرمان OTcl از داخل C++ می بایست آدرس "Tcl::instance()" را بدست آورد که یک متغیر ایستا است. به این ترتیب شما می توانید به توابعی دسترسی یابید که فرمانهای OTcl رادریافت و توسط مفسر OTcl اجرا نمایند (اولین خط تابع MyPrivFunc این کار را انجام می دهد). این مثال دو روش انتقال فرمان OTclبه مفسر را نشان می دهد. برای آگاهی از لیست کامل توابع انتقال فرمانهای OTcl به مستندات NS مراجعه نمایید.

۴-۲-۵- ترجمه[2]، اجرا و آزمایش "MyAgent"

تا به اینجا روشهای مختلف برقراری پیوند بین دست نویسهای OTcl و اشیاء ایجاد شده در محیط برنامه سازی C++ را با استفاده از نمونه "MyAgent" بررسی کردیم. با توجه به اینکه اجرا و آزمایش این نمونه می تواند به درک بیشتر خواننده از تکنیک پیوند OTcl و C++ کمک نماید در این قسمت رویه ای ارائه می نماییم که با استفاده از آن می توان "MyAgent" را ترجمه، اجرا و آزمایش نمود.

۱.فایل "ex-linkge.cc" را ایجاد و در فهرست ns-2 ذخیره کنید.
(متن این برنامه در پیوست ۴ ارائه شده است)
۲.فایل "Makefile" را باز کرده و در انتهای فهرست فایلهای مقصود[3] فایل "ex-linkage.o" را اضافه کنید.
۳.بسته نرم افزاری NS را با استفاده از فرمان Make مجدداً ترجمه کنید.

1 -Private member variable
2 -Compile
3 -Object Files

۴.دست نویس "ex-linkage.tcl" را ایجاد کنید. این فایل حاوی فرمانهای OTcl است که شئ MyAgent را آزمایش می کند. (متن این دست نویس در پیوست ۵ آمده است) همچنین شکل ۴-۶ این دست نویس و نتیجه ناشی از اجرای آن را نشان می دهد.
۵.دست نویس OTcl را با استفاده از فرمان"ns ex-linkage.tcl" اجرا کنید.

ex-linkage.tcl

```
# Create MyAgent (This will give two warning messages that
# no default vaules exist for my_var1_otcl and my_var2_otcl)
set myagent [new Agent/MyAgentOtcl]


# Set configurable parameters of MyAgent
$myagent set my_var1_otcl 2
$myagent set my_var2_otcl 3.14

# Give a command to MyAgent
$myagent call-my-priv-func
```

result

```
warning: no class variable Agent/MyAgentOtcl::my_var1_otcl
         see tcl-object.tcl in tclcl for info about this warning.
warning: no class variable Agent/MyAgentOtcl::my_var2_otcl

Message From MyPrivFunc
     my_var1 = 2
     my_var2 = 3.140000
```

شکل ۴-۶- دست نویس آزمون گر OTcl و خروجی آن

۴-۳-افزودن کاربردها[1] و کارگزارهای[2] جدید

در این قسمت از طریق ذکر یک مسئله با فرآیند ایجاد کاربردها و کارگزارهای جدید آشنا می شویم. فرض کنید می خواهیم یک کاربرد چند رسانه ای [3] برروی پروتکل UDP ایجاد کنیم. این کاربرد رفتار یک کاربرد چند رسانه ای مجازی را شبیه‌سازی می کند. این کاربرد به گونه ای است که از پنج زوج کدگذاری[4] و نرخ ارسال[5] موسوم به *سطوح پنجگانه نرخ رسانه* بهره می برد و با توجه به ازدحام[6] موجود در شبکه روش کدگذاری و نرخ ارسال را تغییر می دهد.

در این کاربرد فرض می کنیم هنگامی که ارتباط فرستنده و گیرنده بر قرار می‌شود، طرفین برروی یکی از سطوح پنجگانه توافق می کنند. این پنج سطح به ترتیب از صفر تا چهار شماره گذاری شده اند. برای سادگی فرض می کنیم که نرخ ارسال برای هریک از سطوح کد گذاری ثابت است. همچنین اندازه بسته ها صرفنظر از روش کدگذاری یکسان است.

1 -Applications
2 -Agents
3 -Multimedia
4 -Encoding
5 -Transmission Rate
6 -Congestion

این کاربرد به بیان ساده به این صورت عمل می کند. فرستنده با نرخ ارسال (و روش کدگذاری) شماره صفر اقدام به ارسال بسته ها می کند و با اعلام گیرنده روش ارسال (نرخ و کدگذاری) بعدی را انتخاب می کند. گیرنده وظیفه دارد که بر ازدحام شبکه نظارت کرده و الگوی مناسب را تعیین کند. برای تعیین ازدحام در شبکه یک نظارت تناوبی برای آگاهی از بسته های از دست رفته مورد استفاده قرار می گیرد. دوره تناوب این نظارت به اندازه هر RTT[1] ثانیه است. هنگامی که بسته از دست برود، ازدحام در شبکه آشکار شده و گیرنده مقیاس مناسب را به فرستنده اعلام می کند. در صورتی که ازدحام نیز صورت نگرفته باشد گیرنده مقدار مقیاس را یک واحد اضافه می کند.

قبل از پیاده سازی این کاربرد، کارگزار UDP را بررسی کرده و مشکل مهمی در این کارگزار آشکار می شود. با توجه به آنکه تخصیص بسته و ارسال آن توسط این کارگزار انجام می شود، لذا تمام اطلاعات مورد نیاز این کاربرد از نظر کارگزار UDP به عنوان جریان داده تلقی می شود. از سوی دیگر پیاده سازی UDP تنها بسته هایی را تخصیص می دهد که دارای سرآیند ثبت شده باشند. بنابراین می بایست پیاده سازی این کارگزار را به گونه ای تصحیح که داده های دریافت شده از کاربرد را ارسال کند. همچنین ممکن است بخواهیم از این کاربرد برای تحقیق بیشتر برروی مسیریابهای IP و مکانیزمهای مدیریت صف در آنها استفاده کنیم. بنابراین باید راهی داشته باشیم که این بسته ها را از سایر بسته ها متمایز نماییم. لذا می بایست کارگزار UDP را به نوعی تصحیح کنیم که نوع داده را در یکی از فیلدهای بدون استفاده ثبت نماید.

```
// Multimedia Header Structure
struct hdr_mm {
        int ack;      // is it ack packet?
        int seq;      // mm sequence number
        int nbytes;   // bytes for mm pkt
        double time;  // current time
        int scale;    // scale (0-4) associated with data rates

        // Packet header access functions
        static int offset_;
        inline static int& offset() { return offset_; }
        inline static hdr_mm* access(const Packet* p) {
                return (hdr_mm*) p->access(offset_);
        }
};
```

```
// Multimedia Header Class
static class MultimediaHeaderClass : public PacketHeaderClass {
public:
 MultimediaHeaderClass() : PacketHeaderClass("PacketHeader/Multimedia",
                                              sizeof(hdr_mm)) {
        bind_offset(&hdr_mm::offset_);
        }
} class_mmhdr;
```

شکل ۴-۷-ساختار سرآیند و کلاس MM در فایل "udp-mm.cc" , "udp-mm.h"

برای پیاده سازی این کاربرد از پیاده سازی CBR به عنوان نمونه استفاده کرده و سطوح پنجگانه را به آن اضافه می کنیم. پس از بررسی سلسله مراتب کلاسهای C++ نام کلاس این کاربرد را به صورت "MmApp" و به عنوان فرزند کلاس "Application" انتخاب می کنیم. نام منتاظر آن در OTcl ، به صورت"Application/MmApp" خواهد بود. رفتار فرستنده و گیرنده این کاربرد در

1 -Round Trip Time

"MmApp" پیاده سازی می شود. کارگزار تصحیح شده UDP که از MmApp پشتیبانی می کند، "UdpMmAgent" نامیده شده و فرزند کلاس "UdpAgent" خواهد بود. نام متناظر در سلسله مراتب اشیاء OTcl به صورت "Agent/UDP/UDPmm" است.

```
enum packet_t {
        PT_TCP,
        ...
        PT_Multimedia,
        PT_NTYPE // This MUST be the LAST one
};

class p_info {
public:
        p_info() {
                name_[PT_TCP]= "tcp";
                ...
                name_[PT_Multimedia] = "Multimedia";
                name_[PT_NTYPE]= "undefined";
        }
        ...
};
```

```
foreach prot {
        AODV
        ...
        Multimedia
} {
        add-packet-header $prot
}
```

شکل ۴-۸-افزودن به فایل "ns-packet.tcl"

MmApp Header : برای ارتباطات در سطح کاربرد، سرآیندی با نام hdr_mm در زبان C++ تعریف می‌کنیم. هرزمان که کاربرد، اطلاعاتی برای ارسال داشته باشد، این اطلاعات در قالب ساختار hdr_mm به کارگزار UdpMmAgent ارسال می شود. سپس این کارگزار با توجه به اندازه شبیه‌سازی شده از هر بسته داده، یک یا چند بسته تخصیص داده و داده ها را در سرآیند هر بسته می نویسد. شکل ۴-۷ تعریف این سرآیند و کلاس مربوطه را نشان می‌دهد. توجه کنید که کلاس "MultimediaHeaderClass" از کلاس "PacketHeaderClass" منشعب شده است. پس از تعریف این سرآیند در زبان برنامه نویسی C++ باید جایگاه این سرآیند را نیز در مجموعه سرآیندها به کمک متغییری[1] مشخص کنیم. این متغیر "off_mm_" است. این تغییرات (سرآیند جدید OTcl و متغییر off_mm_) می بایست در فایل ns-packet.tcl اضافه شوند.(شکل ۴-۸) به این ترتیب سرآیند جدید تعریف شده در مجموعه سرآیندهای بسته نرم افزاری NS اضافه می شود. به این ترتیب فرآیند ایجاد سرآیند خاتمه پیدا کرده و UdpMmAget با تعریف یک متغیر و انقیاد آن به off_mm_ به سرآیند جدید دسترسی خواهد داشت. برای اطلاع از ادامه کاربرد و کارگزار تغییر یافته UDP به پیوست ۶ مراجعه کنید.

1 -Offset Variable

MmApp Sender : فرستنده برای زمانبندی ارسال بسته بعدی کاربرد از یک زمان سنج[1] استفاده می کند. کلاس "SendTimer" که از کلاس "TimerHandler" گرفته شده است دارای تابعی به نام expire() است که متد send_mm_pkt() مربوط به شئ MmApp را فراخوانی می کند. به دنبال آن یک نمونه از این شئ را به عنوان عضو خصوصی[2] MmApp و تحت نام snd_timer_ تعریف می کنیم. شکل ۴-۹ پیاده سازی SendTimer را نشان می دهد. قبل از تنظیم این زمان سنج، MmApp زمان ارسال بعدی را با استفاده از نرخ ارسال جاری (یکی از سطوح پنجگانه) و اندازه بسته که در دست نویس ورودی شبیه‌سازی ذکر شده است، محاسبه می کند. MmApp هنگامی که یک بسته تصدیق را از گیرنده دریافت می کند، پارامترهای نرخ ارسال و کدگذاری را روزآمد می کند.

```
class SendTimer : public TimerHandler {
 public:
        SendTimer(MmApp* t) : TimerHandler(), t_(t) {}
        inline virtual void expire(Event*);
 protected:
        MmApp* t_;
};

void SendTimer::expire(Event*)
{
  t_->send_mm_pkt();
}
```

```
class MmApp : public Application {
public:
        MmApp();
        ...
private:
        ...
        SendTimer snd_timer_;
        ...
};
```

```
MmApp::MmApp() : running_(0), snd_timer_(this), ack_timer_(this)
{
  bind_bw("rate0_", &rate[0]);
  ...
  bind_bw("rate4_", &rate[4]);
  bind("pktsize_", &pktsize_);
  bind_bool("random_", &random_);
}
```

```
void MmApp::send_mm_pkt()
{
  hdr_mm mh_buf;

  if (running_) {
    ...
    agent_->sendmsg(pktsize_, (char*) &mh_buf);  // send to UDP

    // Reschedule the send_pkt timer
    double next_time_ = next_snd_time();
    if(next_time_ > 0) snd_timer_.resched(next_time_);
  }
}
```

شکل ۴-۹-پیاده سازی SendTimer

1 -Timer
2 -Private Member

MmAppReceiver : گیرنده از یک زمان سنج موسوم به "ack_timer_" استفاده می کند. این زمان سنج، موعد ارسال بسته تصدیق (ACK) [1] را زمانبندی می کند که تناوب آن با میانگین RTT برابر است. هنگامی که گیرنده بسته داده کاربرد را دریافت می کند، تعداد بسته های دریافت شده و تعداد بسته های گم شده را (با استفاده از شماره ترتیب) می شمارد. هنگامی که زمان سنج "ack_timer_" منقصی شود، متد send_ack_pkt از MmApp را فراخوانی می کند. این تابع با توجه به شمارش بسته های رسیده و گم شده یکی از سطوح پنجگانه را انتخاب کرده و شمارنده ها را نیز صفر کرده و یک بسته تصدیق با مقدار تعیین شده سطح ارسال جدید به فرستنده می فرستد. توجه کنید که گیرنده هیچ فاز برقراری اتصالی یا روالی برای خاتمه آن ندارد. بنابراین با دریافت اولین بسته آغاز شده و به تناوب بسته های تصدیق را ارسال کرده و هیچگاه متوقف نمی شود.

UdpMmAgent : روال UdpMmAgent که از کارگزار UdpAgent اقتباس شده است از ویژگیهای سه گانه زیر بهره می برد:

- اطلاعات دریافت شده از MmApp را در سرآیند بسته داده MM می نویسد. (و یا اطلاعات دریافت شده از بسته را خوانده و بهMmApp منتقل می کند)
- قطعه بندی[2] و جمع آوری مجدد[3] بسته ها را انجام می دهد. (کارگزار UDP تنها قطعه بندی را انجام می دهد.)
- مقدار ۱۵ (بالاترین مقدار اولویت) را در بیت اولویت (IPv6) براس بسته های MmApp تنظیم می کند.

"Agent.h" : برای اینکه کاربرد جدید و کارگزار مربوط به آن در بسته نرم افزاری NS قابل استفاده باشد، می‌بایست دو متد به صورت عمومی[4] به کلاس *کارگزار* "Agent" اضافه شود. در متد "Commnad" از کلاس "MmApp" یک فرمان OTcl به نام "attach-agent" تعریف شده است. هنگامی که این فرمان از دست نویس OTcl دریافت می شود، "MmApp" سعی می کند که خودش را به کارگزار زیرین متصل کند. قبل از این اتصال، تابع "supportMM()" را از کارگزار زیرین فراخوانی می‌کند تا پشتیبانی آن کارگزار از کاربرد چند رسانه را کنترل کند و در صورتی که پاسخ مثبت باشد، تابع "enableMM()" را فرا می خواند. علیرغم اینکه این دو تابع در کلاس "UdpMmApp" تعریف شده اند ولی در کلاس والد (یعنی "Agent") تعریف نشده اند و به این ترتیب دو تابع عمومی کلاس کارگزار فراخوانی می شوند. بنابراین هنگامی که برنامه ها را ترجمه می کنیم یک پیام خطا تولید می شود. با درج دو متد به عنوان رویه های عمومی (در فایل "agent.h") این مشکل بر طرف خواهد شد.

1 -Acknowledge
2 -Segmentation
3 -Re-Assembly
4 -Public

```
class Agent : public Connector {
 public:
        Agent(int pktType);
        ...
        virtual int supportMM() { return 0; }
        virtual void enableMM() {}
        virtual void sendmsg(int nbytes, const char *flags = 0);
        virtual void send(int nbytes) { sendmsg(nbytes); }
        ...
}
```

```
class Application : public Process {
public:
        Application();
        virtual void send(int nbytes);
        virtual void recv(int nbytes);
        virtual void recv_msg(int nbytes, const char *msg = 0){};
        virtual void resume();
        ...
};
```

شکل ۴-۱۰-افزودن دو تابع به کلاس Agent

تصحیح فایل "ns-default.tcl" : پس از پیاده سازی تمام اجزاء کاربرد و کارگزار آخرین کار تنظیم مقادیر پیش فرض برای پارامترهای پیکربندی در فایل "ns-default.tcl" است. مثالی از تنظیم مقادیر پیش فرض در شکل ۴-۱۱ دیده می شود. این مقادیر برای پارامترهای به کاربرده می شوند که توسط MmApp تعریف شده اند.

```
...
Application/MmApp set rate0_ 0.3mb
Application/MmApp set rate1_ 0.6mb
Application/MmApp set rate2_ 0.9mb
Application/MmApp set rate3_ 1.2mb
Application/MmApp set rate4_ 1.5mb

Application/MmApp set pktsize_ 1000
Application/MmApp set random_ false
...
```

شکل ۴-۱۱-تنظیم مقادیر پیش فرض پارامترها

قبل از ترجمه مجدد NS می توانید از رویه زیر برای کنترل مرحله به مرحله استفاده کنید:

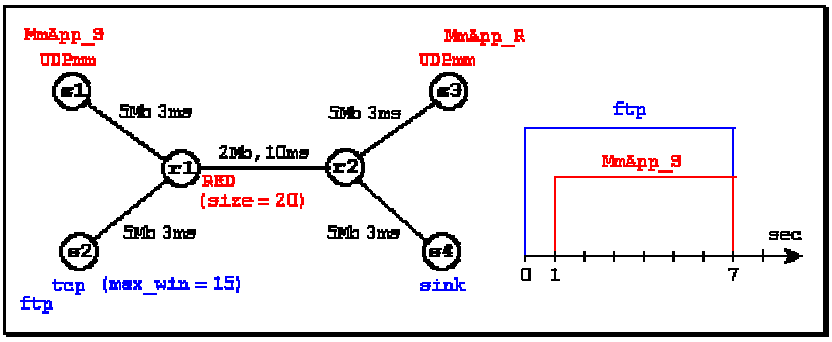
۱.فایلهای "udp-mm.cc", "udp-mm.h", "mm-app.cc" , "mm-app.h" را در زیر فهرست ns-2 ایجاد کنید. (پیوست ۶)

۲.سرآیند کاربرد جدید را در فایل "ns-packet.tcl" ثبت کنید.

۳.دو متد جدید برای کلاس کارگزار "Agent" را در فایل "agent.h" اضافه کنید.

۴.مقادیر پیش فرض پارامترها را در فایل "ns-default.tcl" درج کنید.

پس از انجام موارد فوق فایل Makefile را بازکرده و "mm-app.o" و "udp-mm.o" را در لیست فایلهای مقصد اضافه کنید و با استفاده از فرمان make ، NS را مجدداً ترجمه کنید.



شکل ۴-۱۲-همبندی و سناریوی شبیه‌سازی

شکل ۴-۱۲ یک همبندی و سناریوی شبیه‌سازی را به منظور آزمایش این کاربرد و کارگزارش نشان می دهد و در شکل ۴-۱۳ دست نویس شبیه‌سازی آزمایشی مشاهده می شود.

```
set ns [new Simulator]
…
$ns duplex-link $node_(r1) $node_(r2) 2Mb 10ms RED
…
#Setup RED queue parameter
$ns queue-limit $node_(r1) $node_(r2) 20
Queue/RED set thresh_ 5
Queue/RED set maxthresh_ 10
Queue/RED set q_weight_ 0.002
Queue/RED set ave_ 0
…
#Setup a MM UDP connection
set udp_s [new Agent/UDP/UDPmm]
set udp_r [new Agent/UDP/UDPmm]
$ns attach-agent $node_(s1) $udp_s
$ns attach-agent $node_(s3) $udp_r
$ns connect $udp_s $udp_r
$udp_s set packetSize_ 1000
$udp_r set packetSize_ 1000
$udp_s set fid_ 1
$udp_r set fid_ 1

#Setup a MM Application
set mmapp_s [new Application/MmApp]
set mmapp_r [new Application/MmApp]
$mmapp_s attach-agent $udp_s
$mmapp_r attach-agent $udp_r
$mmapp_s set pktsize_ 1000
$mmapp_s set random_ false
…
#Simulation Scenario
$ns at 0.0 "$ftp start"
$ns at 1.0 "$mmapp_s start"
$ns at 7.0 "finish"

$ns run
```

شکل ۴-۱۳- دست نویس آزمایش MmApp

۴-۴- افزودن صف جدید

در این قسمت صف خروجی یک مسیریاب را درنظر می گیریم. این صف از نوع Drop-Tail بوده و مکانیزم خارج کردن بسته ها از آن زمان‌بندی Round-Robin است. این مسیریاب دو نوع بسته را در صفهای خروجی خود نگهداری می کند، یک دسته بسته های معمولی و دیگری بسته هایی که دارای اولویت ۱۵ هستند. (به عنوان مثال بسته هایی که توسط کاربرد MmApp و برروی کارگزار

UDPmm در بخش قبل بررسی شدند.) در واقع در صف خروجی این مسیریاب بسته های با اولویت ۱۵ (بالاترین اولویت) در کنار سایر بسته ها در صف قرار می گیرند و مسیریاب قدیمی ترین بسته را از هریک از دو گروه و به نوبت خارج می کند.

```
class DtRrQueue : public Queue {
 public:
        DtRrQueue() {
                q1_ = new PacketQueue;
                q2_ = new PacketQueue;
                pq_ = q1_;
                deq_turn_ = 1;
        }

 protected:
        void enque(Packet*);
        Packet* deque();

        PacketQueue *q1_;    // First  FIFO queue
        PacketQueue *q2_;    // Second FIFO queue
        int deq_turn_;       // 1 for First queue 2 for Second
};
```

```
void DtRrQueue::enque(Packet* p)
{
  hdr_ip* iph = hdr_ip::access(p);

  // if IPv6 priority = 15 enqueue to queue1
  if (iph->prio_ == 15) {
    q1_->enque(p);
    if ((q1_->length() + q2_->length()) > qlim_) {
      q1_->remove(p);
      drop(p);
    }
  }
  else {
    q2_->enque(p);
    if ((q1_->length() + q2_->length()) > qlim_) {
      q2_->remove(p);
      drop(p);
    }
  }
}
```

```
Packet* DtRrQueue::deque()
{
  Packet *p;

  if (deq_turn_ == 1) {
        p =  q1_->deque();
        if (p == 0) {
            p = q2_->deque();
            deq_turn_ = 1;
        }
        else
            deq_turn_ = 2;
  }
  else {
       p =  q2_->deque();
       if (p == 0) {
           p = q1_->deque();
           deq_turn_ = 2;
       }
       else
           deq_turn_ = 1;
  }

  return (p);
}
```

شکل ۴-۱۴-پیاده سازی کلاس DtRrQueue

صف خروجی در این مسیریاب از دو صف منطقی با نظام FIFO تشکیل می شود. این دو صف را LQ1 و LQ2 می نامیم. اندازهٔ مجموع این دوصف با اندازه صف فیزیکی (PQ) برابر است، به عبارت دیگر رابطه LQ1 + LQ2 = PQ در مورد اندازه صفها برقراراست. برای پیاده سازی یک صف با رفتار صف گذاری[1] از نوع Drop-Tail، هنگامی که یک بسته برای ورود به صف دریافت می شود، مدیر صف

1 -Enqueue

مجموع LQ1+LQ2 را کنترل کرده و در صورتی که این مجموع از PQ کمتر باشد با توجه به فیلد اولویت بسته،آن را در صف مربوطه وارد می کند. برای پیاده سازی رفتار Round-Robin مدیر صف در هر نوبت یک بسته را از هر یک از دو صف LQ1 و LQ2، به ترتیب خارج می کند. به عبارت دیگر در صورتی که هر دوصف دارای بسته باشند، بسته ها به نرخ ۱:۱ (یک به یک) از صفها خارج می شوند.

به منظور پیاده سازی این صف، نام شئ مربوط به این صف را DtRrQueue[1] انتخاب کرده ایم. این شئ از شئ صف اقتباس شده است. نام متناظر آن در OTcl به صورت Queue/DTRR است. هنگامی که متد recv که در کلاس Queueپیاده سازی شده است بسته ای را دریافت می کند، تابع enqueu را فراخوانی کرده و هنگامی که پیوند ارتباطی بلوک (مسدود)نباشد، تابع dequeu را فرامی‌خواند. هنگامی که پیوند ارتباطی نیز از حالت بلوک خارج می شود نیز این تابع فراخوانی می‌شود. بنابراین می بایست این دو تابع برای شئ DtRrQueue مجدداً با توجه به نظام مورد نظر بازنویسی شوند. شکل ۴-۱۴ توصیف این کلاس و توابع enqueue و dequeue را نشان می دهد. برای اطلاعات بیشتر به پیوست ۷ مراجعه نمایید.

برای آزمایش این صف جدید دست نویس بخش قبلی را تغییر داده و در آن صفهای RED را با صفهایی از نوع DtRr جایگزین می کنیم. شکل ۴-۱۵ این دست نویس تغییر یافته را نشان می‌دهد.

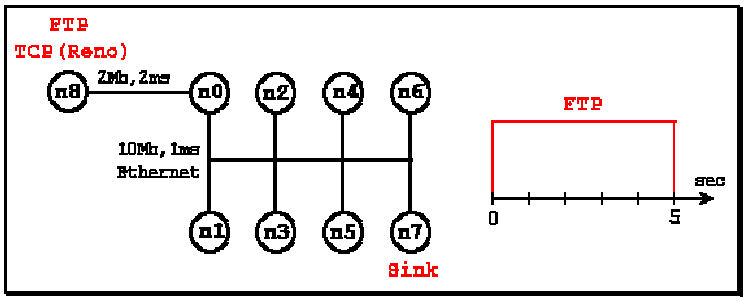
```
set ns [new Simulator]
…
$ns duplex-link $node_(s1) $node_(r1) 5Mb 3ms DropTail
$ns duplex-link $node_(s2) $node_(r1) 5Mb 3ms DropTail
$ns duplex-link $node_(r1) $node_(r2) 2Mb 10ms DTRR
$ns duplex-link $node_(s3) $node_(r2) 5Mb 3ms DropTail
$ns duplex-link $node_(s4) $node_(r2) 5Mb 3ms DropTail

#Set DTRR queue size to 20
$ns queue-limit $node_(r1) $node_(r2) 20
…
#Simulation Scenario
$ns at 0.0 "$ftp start"
$ns at 1.0 "$mmapp_s start"
$ns at 7.0 "finish"

$ns run
```

شکل ۴-۱۵- دست نویس آزمایش DtRrQueue

۴-۵- مثال شبکه های محلی

در این بخش به بررسی مثالی می پردازیم که یک شبکه محلی ساده را شبیه‌سازی می‌نماید. شکل ۴-۱۶ همبندی و سناریوی شبیه‌سازی را نشان می دهد. متن کامل دست نویس این شبیه‌سازی در پیوست ۸ می باشد. بررسی بیشتر این مثال به خواننده واگذار می شود.

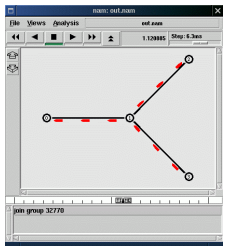
1 -Drop-Tail, Round Robin Queue



شکل ۴-۱۶-همبندی و سناریو شبیه‌سازی شبکه های محلی

۴-۶-مثال چند پراکنی

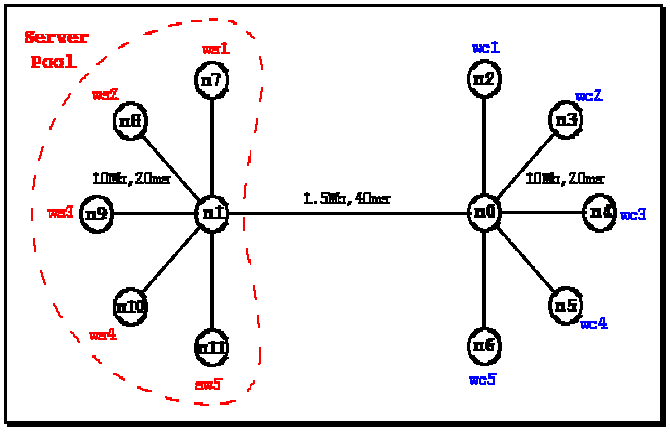
این مثال از پنجمین کارگاه شبیه ساز VINT/NSگرفته شده است و در آن سناریوی شبیه‌سازی و همبندی چندپراکنی مورد بررسی قرار می‌گیرد. شکل ۴-۱۷ خروجی تولید شده توسط نرم افزار NAM[1] را نشان می‌دهد. متن کامل دست نویس این مثال در پیوست ۹ آمده است.



شکل ۴-۱۷-خروجی نرم افزار NAM (شبیه‌سازی چند پراکنی)

۴-۷-مثال سرویس دهنده تورجهان گستر[2]

این مثال نیز از پنجمین کارگاه شبیه ساز VINT/NS گرفته شده است. همبندی شبکه مورد نظر در شکل ۴-۱۸ دیده می شود. متن کامل دست نویسهای مربوط به این شبیه‌سازی در پیوست ۱۰ آمده است. توجه داشته باشید که برای اجرای این دست نویس برروی کامپیوتر خودتان می‌بایست مسیر فایل "http-mod.tcl" را در ابتدای دست نویس به طور مناسب تغییر دهید.



شکل ۴-۱۸-همبندی شبکه در شبیه‌سازی تورجهان گستر

1 -Network Animator
2 -World Wide Web Server

۵-مراجع

در این بخش به بررسی مراجعی می پردازیم که کاربران علاقه مند می توانند برای دریافت اطلاعات بیشتر در مورد NSو سایر اجزاء عملیاتی به آنها مراجعه نمایند. نظر به اینکه تمام این مراجع برروی اینترنت قرار دارند لذا ممکن است در هنگامی که شما این گزارش را مطالعه می نمایید؛ این نشانی‌ها تغییر کرده باشند.

1. VINT project home page: http://www.isi.edu/nsnam/vint
2. NS Home page: http://www.isi.edu/nsnam/ns
3. NS Tutorial: http://www.isi.edu/nsnam/ns/tutorial/
4. NS Installation: http://www.isi.edu/nsnam/ns/ns-build.html
5. NS Manula page: http://www.isi.edu/nsnam/ns/ns-documentation.html
6. NS Class Hierarchy: http://www-sop.inria.fr/rodeo/personnel/Antoine.Clerget/ns
7. Tcl/Tk Quick Reference Guide: http://www.slac.stanford.edu/~raines/tkref.html
8. OTcl Tutorial (Berkeley Version): http://bmrc.berkeley.edu/research/cmt/cmtdoc/OTcl
9. OTcl Tutorial (MIT Version): ftp://ftp.tns.lcs.mit.edu/pub/OTcl/README.html
10. Network Animator (NAM): http://www.isi.edu/nsnam/nam

پیوست ۱ : دست نویس مربوط به یک شبیه‌سازی ساده
“ns_simple.tcl”

```
#Create a simulator object
set ns [new Simulator]

#Define different colors for data flows (for NAM)
$ns color 1 Blue
$ns color 2 Red

#Open the NAM trace file
set nf [open out.nam w]
$ns namtrace-all $nf

#Define a 'finish' procedure
proc finish {} {
        global ns nf
        $ns flush-trace
        #Close the NAM trace file
        close $nf
        #Execute NAM on the trace file
        exec nam out.nam &
        exit 0
}

#Create four nodes
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]

#Create links between the nodes
$ns duplex-link $n0 $n2 2Mb 10ms DropTail
$ns duplex-link $n1 $n2 2Mb 10ms DropTail
$ns duplex-link $n2 $n3 1.7Mb 20ms DropTail

#Set Queue Size of link (n2-n3) to 10
$ns queue-limit $n2 $n3 10

#Give node position (for NAM)
$ns duplex-link-op $n0 $n2 orient right-down
$ns duplex-link-op $n1 $n2 orient right-up
$ns duplex-link-op $n2 $n3 orient right

#Monitor the queue for link (n2-n3). (for NAM)
$ns duplex-link-op $n2 $n3 queuePos 0.5


#Setup a TCP connection
set tcp [new Agent/TCP]
$tcp set class_ 2
$ns attach-agent $n0 $tcp
set sink [new Agent/TCPSink]
$ns attach-agent $n3 $sink
$ns connect $tcp $sink
$tcp set fid_ 1

#Setup a FTP over TCP connection
set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ftp set type_ FTP


#Setup a UDP connection
set udp [new Agent/UDP]
$ns attach-agent $n1 $udp
set null [new Agent/Null]
$ns attach-agent $n3 $null
$ns connect $udp $null
$udp set fid_ 2

#Setup a CBR over UDP connection
set cbr [new Application/Traffic/CBR]
$cbr attach-agent $udp
```

```
$cbr set type_ CBR
$cbr set packet_size_ 1000
$cbr set rate_ 1mb
$cbr set random_ false


#Schedule events for the CBR and FTP agents
$ns at 0.1 "$cbr start"
$ns at 1.0 "$ftp start"
$ns at 4.0 "$ftp stop"
$ns at 4.5 "$cbr stop"

#Detach tcp and sink agents (not really necessary)
$ns at 4.5 "$ns detach-agent $n0 $tcp ; $ns detach-agent $n3 $sink"

#Call the finish procedure after 5 seconds of simulation time
$ns at 5.0 "finish"

#Print CBR packet size and interval
puts "CBR packet size = [$cbr set packet_size_]"
puts "CBR interval = [$cbr set interval_]"

#Run the simulation
$ns run
```

پیوست ۲ : دست نویس مربوط به یک شبیه‌سازی ساده با ردیابی
“ns_simple_trace.tcl”

```
#Create a simulator object
set ns [new Simulator]

#Define different colors for data flows (for NAM)
$ns color 1 Blue
$ns color 2 Red

#Open the NAM trace file
set nf [open out.nam w]
$ns namtrace-all $nf

#Open the Trace file
set tf [open out.tr w]
$ns trace-all $tf

#Define a 'finish' procedure
proc finish {} {
        global ns nf tf
        $ns flush-trace
        #Close the NAM trace file
        close $nf
        #Close the Trace file
        close $tf
        #Execute NAM on the trace file
        exec nam out.nam &
        exit 0
}

#Create four nodes
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]

#Create links between the nodes
$ns duplex-link $n0 $n2 2Mb 10ms DropTail
$ns duplex-link $n1 $n2 2Mb 10ms DropTail
$ns duplex-link $n2 $n3 1.7Mb 20ms DropTail

#Set Queue Size of link (n2-n3) to 10
$ns queue-limit $n2 $n3 10

#Give node position (for NAM)
$ns duplex-link-op $n0 $n2 orient right-down
$ns duplex-link-op $n1 $n2 orient right-up
$ns duplex-link-op $n2 $n3 orient right

#Monitor the queue for link (n2-n3). (for NAM)
$ns duplex-link-op $n2 $n3 queuePos 0.5


#Setup a TCP connection
set tcp [new Agent/TCP]
$tcp set class_ 2
$ns attach-agent $n0 $tcp
set sink [new Agent/TCPSink]
$ns attach-agent $n3 $sink
$ns connect $tcp $sink
$tcp set fid_ 1

#Setup a FTP over TCP connection
set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ftp set type_ FTP


#Setup a UDP connection
set udp [new Agent/UDP]
$ns attach-agent $n1 $udp
set null [new Agent/Null]
```

```
$ns attach-agent $n3 $null
$ns connect $udp $null
$udp set fid_ 2

#Setup a CBR over UDP connection
set cbr [new Application/Traffic/CBR]
$cbr attach-agent $udp
$cbr set type_ CBR
$cbr set packet_size_ 1000
$cbr set rate_ 1mb
$cbr set random_ false


#Schedule events for the CBR and FTP agents
$ns at 0.1 "$cbr start"
$ns at 1.0 "$ftp start"
$ns at 4.0 "$ftp stop"
$ns at 4.5 "$cbr stop"

#Detach tcp and sink agents (not really necessary)
$ns at 4.5 "$ns detach-agent $n0 $tcp ; $ns detach-agent $n3 $sink"

#Call the finish procedure after 5 seconds of simulation time
$ns at 5.0 "finish"

#Print CBR packet size and interval
puts "CBR packet size = [$cbr set packet_size_]"
puts "CBR interval = [$cbr set interval_]"

#Run the simulation
$ns run
```

پیوست ۳ : دست نویس مربوط به صف RED
“red.tcl”

```
# RED Queue Monitor Siulation Script

set ns [new Simulator]
#
# Create a simple six node topology:
#
#        s1                 s3
#         \                 /
# 10Mb,2ms \  1.5Mb,20ms   / 10Mb,4ms
#           r1 --------- r2
# 10Mb,3ms /               \ 10Mb,5ms
#         /                 \
#        s2                 s4
#
set node_(s1) [$ns node]
set node_(s2) [$ns node]
set node_(r1) [$ns node]
set node_(r2) [$ns node]
set node_(s3) [$ns node]
set node_(s4) [$ns node]

$ns duplex-link $node_(s1) $node_(r1) 10Mb 2ms DropTail
$ns duplex-link $node_(s2) $node_(r1) 10Mb 3ms DropTail
$ns duplex-link $node_(r1) $node_(r2) 1.5Mb 20ms RED
$ns queue-limit $node_(r1) $node_(r2) 25
$ns queue-limit $node_(r2) $node_(r1) 25
$ns duplex-link $node_(s3) $node_(r2) 10Mb 4ms DropTail
$ns duplex-link $node_(s4) $node_(r2) 10Mb 5ms DropTail

$ns duplex-link-op $node_(s1) $node_(r1) orient right-down
$ns duplex-link-op $node_(s2) $node_(r1) orient right-up
$ns duplex-link-op $node_(r1) $node_(r2) orient right
$ns duplex-link-op $node_(r1) $node_(r2) queuePos 0
$ns duplex-link-op $node_(r2) $node_(r1) queuePos 0
$ns duplex-link-op $node_(s3) $node_(r2) orient left-down
$ns duplex-link-op $node_(s4) $node_(r2) orient left-up


set tcp1 [$ns create-connection TCP/Reno $node_(s1) TCPSink $node_(s3) 0]
$tcp1 set window_ 15
set tcp2 [$ns create-connection TCP/Reno $node_(s2) TCPSink $node_(s3) 1]
$tcp2 set window_ 15
set ftp1 [$tcp1 attach-source FTP]
set ftp2 [$tcp2 attach-source FTP]

# Tracing a queue
set redq [[$ns link $node_(r1) $node_(r2)] queue]
set tchan_ [open all.q w]
$redq trace curq_
$redq trace ave_
$redq attach $tchan_


$ns at 0.0 "$ftp1 start"
$ns at 3.0 "$ftp2 start"
$ns at 10 "finish"

# Define 'finish' procedure (include post-simulation processes)
proc finish {} {
    global tchan_
    set awkCode {
        {
            if ($1 == "Q" && NF>2) {
                print $2, $3 >> "temp.q";
                set end $2
            }
            else if ($1 == "a" && NF>2)
            print $2, $3 >> "temp.a";
        }
    }
    set f [open temp.queue w]
```

```
    puts $f "TitleText: red"
    puts $f "Device: Postscript"

    if { [info exists tchan_] } {
        close $tchan_
    }
    exec rm -f temp.q temp.a
    exec touch temp.a temp.q

    exec awk $awkCode all.q

    puts $f \"queue
    exec cat temp.q >@ $f
    puts $f \n\"ave_queue
    exec cat temp.a >@ $f
    close $f
    exec xgraph -bb -tk -x time -y queue temp.queue &
    exit 0
}

$ns run
```

پیوست ۴ : متن برنامه مربوط به پیوند

“ex-linkage.cc”

```
// ex-linkage.cc
// Example of a simple and dull Agent that
// illustrates the use of OTcl linkages


#include <stdio.h>
#include <string.h>
#include "agent.h"


class MyAgent : public Agent {
public:
        MyAgent();
protected:
        int command(int argc, const char*const* argv);
private:
        int    my_var1;
        double my_var2;
        void   MyPrivFunc(void);
};


static class MyAgentClass : public TclClass {
public:
        MyAgentClass() : TclClass("Agent/MyAgentOTcl") {}
        TclObject* create(int, const char*const*) {
                return(new MyAgent());
        }
} class_my_agent;


MyAgent::MyAgent() : Agent(PT_UDP) {
       bind("my_var1_OTcl", &my_var1);
       bind("my_var2_OTcl", &my_var2);
}


int MyAgent::command(int argc, const char*const* argv) {
      if(argc == 2) {
           if(strcmp(argv[1], "call-my-priv-func") == 0) {
                 MyPrivFunc();
                 return(TCL_OK);
           }
      }
      return(Agent::command(argc, argv));
}


void MyAgent::MyPrivFunc(void) {
      Tcl& tcl = Tcl::instance();
      tcl.eval("puts \"Message From MyPrivFunc\"");
      tcl.evalf("puts \"     my_var1 = %d\"", my_var1);
      tcl.evalf("puts \"     my_var2 = %f\"", my_var2);
}
```

پیوست ۵ : دست نویس مربوط پیوند
“ex-linkage.tcl”

```
# ex-Linkage.tcl

# Create MyAgent (This will give two warning messages that
# no default vaules exist for my_var1_OTcl and my_var2_OTcl)
set myagent [new Agent/MyAgentOTcl]

# Set configurable parameters of MyAgent
$myagent set my_var1_OTcl 2
$myagent set my_var2_OTcl 3.14

# Give a command to MyAgent
$myagent call-my-priv-func
```

پیوست ۶ : فایلهای کاربرد چند رسانه

“mm-app.h, mm-app.cc, udp-mm.h, udp-mm.cc”

```
//
// File:      mm-app.h
//

#include "timer-handler.h"
#include "packet.h"
#include "app.h"
#include "udp-mm.h"

// This is used for receiver's received packet accounting
struct pkt_accounting {
        int last_seq;   // sequence number of last received MM pkt
        int last_scale; // rate (0-4) of last acked
        int lost_pkts;  // number of lost pkts since last ack
        int recv_pkts;  // number of received pkts since last ack
        double rtt;     // round trip time
};


class MmApp;


// Sender uses this timer to
// schedule next app data packet transmission time
class SendTimer : public TimerHandler {
 public:
       SendTimer(MmApp* t) : TimerHandler(), t_(t) {}
       inline virtual void expire(Event*);
 protected:
       MmApp* t_;
};


// Reciver uses this timer to schedule
// next ack packet transmission time
class AckTimer : public TimerHandler {
 public:
       AckTimer(MmApp* t) : TimerHandler(), t_(t) {}
       inline virtual void expire(Event*);
 protected:
       MmApp* t_;
};


// Mulitmedia Application Class Definition
class MmApp : public Application {
 public:
       MmApp();
       void send_mm_pkt();  // called by SendTimer:expire (Sender)
       void send_ack_pkt(); // called by AckTimer:expire (Receiver)
 protected:
       int command(int argc, const char*const* argv);
       void start();       // Start sending data packets (Sender)
       void stop();        // Stop sending data packets (Sender)
 private:
       void init();
       inline double next_snd_time();                          // (Sender)
       virtual void recv_msg(int nbytes, const char *msg = 0); // (Sender/Receiver)
       void set_scale(const hdr_mm *mh_buf);                   // (Sender)
       void adjust_scale(void);                                // (Receiver)
       void account_recv_pkt(const hdr_mm *mh_buf);            // (Receiver)
       void init_recv_pkt_accounting();                        // (Receiver)

       double rate[5];        // Transmission rates associated to scale values
       double interval_;      // Application data packet transmission interval
       int pktsize_;          // Application data packet size
       int random_;           // If 1 add randomness to the interval
       int running_;          // If 1 application is running
       int seq_;              // Application data packet sequence number
       int scale_;            // Media scale parameter
       pkt_accounting p_accnt;
```

```
        SendTimer snd_timer_;  // SendTimer
        AckTimer  ack_timer_;  // AckTimer
};
```

```
//
// File:      mm-app.cc
//

#include "random.h"
#include "mm-app.h"


// MmApp OTcl linkage class
static class MmAppClass : public TclClass {
 public:
  MmAppClass() : TclClass("Application/MmApp") {}
  TclObject* create(int, const char*const*) {
    return (new MmApp);
  }
} class_app_mm;


// When snd_timer_ expires call MmApp:send_mm_pkt()
void SendTimer::expire(Event*)
{
  t_->send_mm_pkt();
}


// When ack_timer_ expires call MmApp:send_ack_pkt()
void AckTimer::expire(Event*)
{
  t_->send_ack_pkt();
}


// Constructor (also initialize instances of timers)
MmApp::MmApp() : running_(0), snd_timer_(this), ack_timer_(this)
{
  bind_bw("rate0_", &rate[0]);
  bind_bw("rate1_", &rate[1]);
  bind_bw("rate2_", &rate[2]);
  bind_bw("rate3_", &rate[3]);
  bind_bw("rate4_", &rate[4]);
  bind("pktsize_", &pktsize_);
  bind_bool("random_", &random_);
}


// OTcl command interpreter
int MmApp::command(int argc, const char*const* argv)
{
  Tcl& tcl = Tcl::instance();

  if (argc == 3) {
    if (strcmp(argv[1], "attach-agent") == 0) {
      agent_ = (Agent*) TclObject::lookup(argv[2]);
      if (agent_ == 0) {
        tcl.resultf("no such agent %s", argv[2]);
        return(TCL_ERROR);
      }

      // Make sure the underlying agent support MM
      if(agent_->supportMM()) {
        agent_->enableMM();
      }
      else {
        tcl.resultf("agent \"%s\" does not support MM Application", argv[2]);
        return(TCL_ERROR);
      }

      agent_->attachApp(this);
      return(TCL_OK);
    }
  }
  return (Application::command(argc, argv));
}



void MmApp::init()
```

```
{
  scale_ = 0; // Start at minimum rate
  seq_ = 0;   // MM sequence number (start from 0)
  interval_ = (double)(pktsize_ << 3)/(double)rate[scale_];
}


void MmApp::start()
{
  init();
  running_ = 1;
  send_mm_pkt();
}


void MmApp::stop()
{
  running_ = 0;
}


// Send application data packet
void MmApp::send_mm_pkt()
{
  hdr_mm mh_buf;

  if (running_) {
    // the below info is passed to UDPmm agent, which will write it
    // to MM header after packet creation.
    mh_buf.ack = 0;            // This is a MM packet
    mh_buf.seq = seq_++;         // MM sequece number
    mh_buf.nbytes = pktsize_;  // Size of MM packet (NOT UDP packet size)
    mh_buf.time = Scheduler::instance().clock(); // Current time
    mh_buf.scale = scale_;                       // Current scale value
    agent_->sendmsg(pktsize_, (char*) &mh_buf);  // send to UDP

    // Reschedule the send_pkt timer
    double next_time_ = next_snd_time();
    if(next_time_ > 0) snd_timer_.resched(next_time_);
  }
}


// Schedule next data packet transmission time
double MmApp::next_snd_time()
{
  // Recompute interval in case rate or size chages
  interval_ = (double)(pktsize_ << 3)/(double)rate[scale_];
  double next_time_ = interval_;
  if(random_)
    next_time_ += interval_ * Random::uniform(-0.5, 0.5);
  return next_time_;
}


// Receive message from underlying agent
void MmApp::recv_msg(int nbytes, const char *msg = 0)
{
  if(msg) {
    hdr_mm* mh_buf = (hdr_mm*) msg;

    if(mh_buf->ack == 1) {
      // If received packet is ACK packet
      set_scale(mh_buf);
    }
    else {
      // If received packet is MM packet
      account_recv_pkt(mh_buf);
      if(mh_buf->seq == 0) send_ack_pkt();
    }
  }
}


// Sender sets its scale to what reciver notifies
void MmApp::set_scale(const hdr_mm *mh_buf)
{
```

```
  scale_ = mh_buf->scale;
}


void MmApp::account_recv_pkt(const hdr_mm *mh_buf)
{
  double local_time = Scheduler::instance().clock();

  // Calculate RTT
  if(mh_buf->seq == 0) {
    init_recv_pkt_accounting();
    p_accnt.rtt = 2*(local_time - mh_buf->time);
  }
  else
    p_accnt.rtt = 0.9 * p_accnt.rtt + 0.1 * 2*(local_time - mh_buf->time);

  // Count Received packets and Calculate Packet Loss
  p_accnt.recv_pkts ++;
  p_accnt.lost_pkts += (mh_buf->seq - p_accnt.last_seq - 1);
  p_accnt.last_seq = mh_buf->seq;
}


void MmApp::init_recv_pkt_accounting()
{
  p_accnt.last_seq = -1;
  p_accnt.last_scale = 0;
  p_accnt.lost_pkts = 0;
  p_accnt.recv_pkts = 0;
}


void MmApp::send_ack_pkt(void)
{
  double local_time = Scheduler::instance().clock();

  adjust_scale();

  // send ack message
  hdr_mm ack_buf;
  ack_buf.ack = 1;  // this packet is ack packet
  ack_buf.time = local_time;
  ack_buf.nbytes = 40;  // Ack packet size is 40 Bytes
  ack_buf.scale = p_accnt.last_scale;
  agent_->sendmsg(ack_buf.nbytes, (char*) &ack_buf);

  // schedul next ACK time
  ack_timer_.resched(p_accnt.rtt);
}


void MmApp::adjust_scale(void)
{
  if(p_accnt.recv_pkts > 0) {
    if(p_accnt.lost_pkts > 0)
      p_accnt.last_scale = (int)(p_accnt.last_scale / 2);
    else {
      p_accnt.last_scale++;
      if(p_accnt.last_scale > 4) p_accnt.last_scale = 4;
    }
  }
  p_accnt.recv_pkts = 0;
  p_accnt.lost_pkts = 0;
}
```

```
//
// File:      udp-mm.h
//

#ifndef ns_udp_mm_h
#define ns_udp_mm_h

#include "udp.h"
#include "ip.h"

// Multimedia Header Structure
struct hdr_mm {
        int ack;     // is it ack packet?
        int seq;     // mm sequence number
        int nbytes;  // bytes for mm pkt
        double time; // current time
        int scale;   // scale (0-4) associated with data rates

        // Packet header access functions
         static int offset_;
         inline static int& offset() { return offset_; }
         inline static hdr_mm* access(const Packet* p) {
                 return (hdr_mm*) p->access(offset_);
         }
};


// Used for Re-assemble segmented (by UDP) MM packet
struct asm_mm {
        int seq;     // mm sequence number
        int rbytes;  // currently received bytes
        int tbytes;  // total bytes to receive for MM packet
};


// UdpMmAgent Class definition
class UdpMmAgent : public UdpAgent {
public:
        UdpMmAgent();
        UdpMmAgent(packet_t);
        virtual int supportMM() { return 1; }
        virtual void enableMM() { support_mm_ = 1; }
        virtual void sendmsg(int nbytes, const char *flags = 0);
        void recv(Packet*, Handler*);
protected:
        int support_mm_; // set to 1 if above is MmApp
private:
        asm_mm asm_info; // packet re-assembly information
};

#endif
```

```
//
// File:      udp-mm.cc
//

#include "udp-mm.h"
#include "rtp.h"
#include "random.h"
#include <string.h>


int hdr_mm::offset_;

// Mulitmedia Header Class
static class MultimediaHeaderClass : public PacketHeaderClass {
public:
        MultimediaHeaderClass() : PacketHeaderClass("PacketHeader/Multimedia",
                                                    sizeof(hdr_mm)) {
                bind_offset(&hdr_mm::offset_);
        }
} class_mmhdr;


// UdpMmAgent OTcl linkage class
static class UdpMmAgentClass : public TclClass {
public:
        UdpMmAgentClass() : TclClass("Agent/UDP/UDPmm") {}
        TclObject* create(int, const char*const*) {
                return (new UdpMmAgent());
        }
} class_udpmm_agent;


// Constructor (with no arg)
UdpMmAgent::UdpMmAgent() : UdpAgent()
{
        support_mm_ = 0;
        asm_info.seq = -1;
}

UdpMmAgent::UdpMmAgent(packet_t type) : UdpAgent(type)
{
        support_mm_ = 0;
        asm_info.seq = -1;
}


// Add Support of Multimedia Application to UdpAgent::sendmsg
void UdpMmAgent::sendmsg(int nbytes, const char* flags)
{
        Packet *p;
        int n, remain;


        if (size_) {
                n = (nbytes/size_ + (nbytes%size_ ? 1 : 0));
                remain = nbytes%size_;
        }
        else
                printf("Error: UDPmm size = 0\n");

        if (nbytes == -1) {
                printf("Error:  sendmsg() for UDPmm should not be -1\n");
                return;
        }
        double local_time =Scheduler::instance().clock();
        while (n-- > 0) {
                p = allocpkt();
                if(n==0 && remain>0) hdr_cmn::access(p)->size() = remain;
                else hdr_cmn::access(p)->size() = size_;
                hdr_rtp* rh = hdr_rtp::access(p);
                rh->flags() = 0;
                rh->seqno() = ++seqno_;
                hdr_cmn::access(p)->timestamp() =
                    (u_int32_t)(SAMPLERATE*local_time);
                // to eliminate recv to use MM fields for non MM packets
                hdr_mm* mh = hdr_mm::access(p);
                mh->ack = 0;
```

```
                mh->seq = 0;
                mh->nbytes = 0;
                mh->time = 0;
                mh->scale = 0;
                // mm udp packets are distinguished by setting the ip
                // priority bit to 15 (Max Priority).
                if(support_mm_) {
                        hdr_ip* ih = hdr_ip::access(p);
                        ih->prio_ = 15;
                        if(flags) // MM Seq Num is passed as flags
                                memcpy(mh, flags, sizeof(hdr_mm));
                }
                // add "beginning of talkspurt" labels (tcl/ex/test-rcvr.tcl)
                if (flags && (0 ==strcmp(flags, "NEW_BURST")))
                        rh->flags() |= RTP_M;
                target_->recv(p);
        }
        idle();
}


// Support Packet Re-Assembly and Multimedia Application
void UdpMmAgent::recv(Packet* p, Handler*)
{
        hdr_ip* ih = hdr_ip::access(p);
        int bytes_to_deliver = hdr_cmn::access(p)->size();

        // if it is a MM packet (data or ack)
        if(ih->prio_ == 15) {
                if(app_) {  // if MM Application exists
                        // re-assemble MM Application packet if segmented
                        hdr_mm* mh = hdr_mm::access(p);
                        if(mh->seq == asm_info.seq)
                                asm_info.rbytes += hdr_cmn::access(p)->size();
                        else {
                                asm_info.seq = mh->seq;
                                asm_info.tbytes = mh->nbytes;
                                asm_info.rbytes = hdr_cmn::access(p)->size();
                        }
                        // if fully reassembled, pass the packet to application
                        if(asm_info.tbytes == asm_info.rbytes) {
                                hdr_mm mh_buf;
                                memcpy(&mh_buf, mh, sizeof(hdr_mm));
                                app_->recv_msg(mh_buf.nbytes, (char*) &mh_buf);
                        }
                }
                Packet::free(p);
        }
        // if it is a normal data packet (not MM data or ack packet)
        else {
                if (app_) app_->recv(bytes_to_deliver);
                Packet::free(p);
        }
}
```

پیوست ۷ : صف DtRrQueue[1]

“dtrr-queue.h, dtrr-queue.cc”

```
//
// File:      dtrr-queue.h
//

#include <string.h>
#include "queue.h"
#include "address.h"


class DtRrQueue : public Queue {
 public:
         DtRrQueue() {
            q1_ = new PacketQueue;
            q2_ = new PacketQueue;
            pq_ = q1_;
            deq_turn_ = 1;
      }

 protected:
         void enque(Packet*);
       Packet* deque();

       PacketQueue *q1_;   // First  FIFO queue
       PacketQueue *q2_;   // Second FIFO queue
       int deq_turn_;      // 1 for First queue 2 for Second
};
```

1 -Drop Tail Round Robin Queue

```
//
// File:      dtrr-queue.cc
//


#include "dtrr-queue.h"


static class DtRrQueueClass : public TclClass {
public:
        DtRrQueueClass() : TclClass("Queue/DTRR") {}
        TclObject* create(int, const char*const*) {
               return (new DtRrQueue);
      }
} class_dropt_tail_round_robin;



void DtRrQueue::enque(Packet* p)
{
  hdr_ip* iph = hdr_ip::access(p);

  // if IPv6 priority = 15 enqueue to queue1
  if (iph->prio_ == 15) {
    q1_->enque(p);
    if ((q1_->length() + q2_->length()) > qlim_) {
      q1_->remove(p);
      drop(p);
    }
  }
  else {
    q2_->enque(p);
    if ((q1_->length() + q2_->length()) > qlim_) {
      q2_->remove(p);
      drop(p);
    }
  }
}



Packet* DtRrQueue::deque()
{
  Packet *p;

  if (deq_turn_ == 1) {
    p =  q1_->deque();
    if (p == 0) {
      p = q2_->deque();
      deq_turn_ = 1;
    }
    else {
      deq_turn_ = 2;
    }
  }
  else {
    p =  q2_->deque();
    if (p == 0) {
      p = q1_->deque();
      deq_turn_ = 2;
    }
```

```
    else {
      deq_turn_ = 1;
    }
  }

  return (p);
}
```

پیوست ۸ : دست نویس شبیه‌سازی شبکه محلی

“ex-lan.tcl”

```
# Modified version of:
# "ns-2/tcl/ex/lantest.tcl"

set opt(tr)    "out.tr"
set opt(namtr) "out.nam"
set opt(seed)  0
set opt(stop)  5
set opt(node)  8

set opt(qsize) 100
set opt(bw)    10Mb
set opt(delay) 1ms
set opt(ll)    LL
set opt(ifq)   Queue/DropTail
set opt(mac)   Mac/Csma/Ca
set opt(chan)  Channel
set opt(tcp)   TCP/Reno
set opt(sink)  TCPSink

set opt(app)   FTP


proc finish {} {
        global ns opt trfd ntrfd

        $ns flush-trace
        close $trfd
         close $ntrfd
        exec nam $opt(namtr) &
        exit 0
}

proc create-trace {} {
        global ns opt

        set trfd [open $opt(tr) w]
        $ns trace-all $trfd
        return $trfd
}

proc create-namtrace {} {
         global ns opt

         set ntrfd [open $opt(namtr) w]
         $ns namtrace-all $ntrfd
}

proc create-topology {} {
        global ns opt
        global lan node source node0

        set num $opt(node)
        for {set i 0} {$i < $num} {incr i} {
                set node($i) [$ns node]
                lappend nodelist $node($i)
        }

        set lan [$ns newLan $nodelist $opt(bw) $opt(delay) \
                        -llType $opt(ll) -ifqType $opt(ifq) \
                        -macType $opt(mac) -chanType $opt(chan)]

        set node0 [$ns node]
        $ns duplex-link $node0 $node(0) 2Mb 2ms DropTail

        $ns duplex-link-op $node0 $node(0) orient right

}

## MAIN ##

set ns [new Simulator]
```

```
set trfd [create-trace]
set ntrfd [create-namtrace]

create-topology

set tcp0 [$ns create-connection TCP/Reno $node0 TCPSink $node(7) 0]
$tcp0 set window_ 15

set ftp0 [$tcp0 attach-app FTP]

$ns at 0.0 "$ftp0 start"
$ns at $opt(stop) "finish"

$ns run
```

پیوست ۹ : دست نویس شبیه‌سازی چند پراکنی

“ex-mcast.tcl”

```
# ex-mcast.tcl

set ns [new Simulator]
$ns multicast

set f [open out.tr w]
$ns trace-all $f
$ns namtrace-all [open out.nam w]

$ns color 1 red
# prune/graft packets
$ns color 30 purple
$ns color 31 green

set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]

# Use automatic layout
$ns duplex-link $n0 $n1 1.5Mb 10ms DropTail
$ns duplex-link $n1 $n2 1.5Mb 10ms DropTail
$ns duplex-link $n1 $n3 1.5Mb 10ms DropTail

$ns duplex-link-op $n0 $n1 orient right
$ns duplex-link-op $n1 $n2 orient right-up
$ns duplex-link-op $n1 $n3 orient right-down
$ns duplex-link-op $n0 $n1 queuePos 0.5

set mrthandle [$ns mrtproto DM {}]

set cbr0 [new Application/Traffic/CBR]
set udp0 [new Agent/UDP]
$cbr0 attach-agent $udp0
$ns attach-agent $n1 $udp0
$udp0 set dst_ 0x8001

set cbr1 [new Application/Traffic/CBR]
set udp1 [new Agent/UDP]
$cbr1 attach-agent $udp1
$udp1 set dst_ 0x8002
$udp1 set class_ 1
$ns attach-agent $n3 $udp1

set rcvr [new Agent/LossMonitor]
#$ns attach-agent $n3 $rcvr
$ns at 1.2 "$n2 join-group $rcvr 0x8002"
$ns at 1.25 "$n2 leave-group $rcvr 0x8002"
$ns at 1.3 "$n2 join-group $rcvr 0x8002"
$ns at 1.35 "$n2 join-group $rcvr 0x8001"

$ns at 1.0 "$cbr0 start"
$ns at 1.1 "$cbr1 start"

$ns at 2.0 "finish"

proc finish {} {
        global ns
        $ns flush-trace

        puts "running nam..."
        exec nam out.nam &
        exit 0
}

$ns run
```

پیوست ۱۰ : دست نویس شبیه‌سازی سرویس دهنده web

“ex-web.tcl, dumbbell.tcl”

```
# ex-web.tcl

# Initial setup
source ~/ns-allinone-2.1b4a/ns-2/tcl/http/http-mod.tcl
source dumbbell.tcl
global num_node n

set ns [new Simulator]
$ns set-address 7 24     ;# set-address <bits for node address> <bits for port>

# set up colors for nam
for {set i 1} {$i <= 30} {incr i} {
    set color [expr $i % 6]
    if {$color == 0} {
        $ns color $i blue
    } elseif {$color == 1} {
        $ns color $i red
    } elseif {$color == 2} {
        $ns color $i green
    } elseif {$color == 3} {
        $ns color $i yellow
    } elseif {$color == 4} {
        $ns color $i brown
    } elseif {$color == 5} {
        $ns color $i black
    }
}

# Create nam trace and generic packet trace
$ns namtrace-all [open out.nam w]
# trace-all is the generic trace we've been using
$ns trace-all [open out.tr w]

create_topology

############################ Modify From Here ######################
## Number of Pages per Session
set numPage 10
set httpSession1 [new HttpSession $ns $numPage [$ns picksrc]]
set httpSession2 [new HttpSession $ns $numPage [$ns picksrc]]

## Inter-Page Interval
## Number of Objects per Page
## Inter-Object Interval
## Number of Packets per Object
## have to set page specific attributes before createPage
## have to set object specific attributes after createPage
$httpSession1 setDistribution interPage_ Exponential 1 ;#in sec
$httpSession1 setDistribution pageSize_ Constant 1 ;# number of objects/page
$httpSession1 createPage
$httpSession1 setDistribution interObject_ Exponential 0.01 ;# in sec
$httpSession1 setDistribution objectSize_ ParetoII 10 1.2 ;# number of packets

# uses default
$httpSession2 createPage

$ns at 0.1 "$httpSession1 start" ;# in sec as well
$ns at 0.2 "$httpSession2 start"

$ns at 30.0 "finish"

proc finish {} {
        global ns
        $ns flush-trace
        puts "running nam..."
    # exec to run unix command
        exec nam out.nam &
        exit 0
}

# Start the simualtion
$ns run
```

```
# dumbbell.tcl
# Simple 4-node star topology
#        8    7       2    3
#           \ |       | /
#        9 -- 1 ----- 0 -- 4
#           / |       | \
#        10   11      6    5

proc create_topology {} {
global ns n num_node
set num_node 12

for {set i 0} {$i < $num_node} {incr i} {
        set n($i) [$ns node]
}

$ns set src_ [list 2 3 4 5 6]
$ns set dst_ [list 7 8 9 10 11]

# EDGES (from-node to-node length a b):
$ns duplex-link $n(0) $n(1) 1.5Mb 40ms DropTail
$ns duplex-link $n(0) $n(2) 10Mb 20ms DropTail
$ns duplex-link $n(0) $n(3) 10Mb 20ms DropTail
$ns duplex-link $n(0) $n(4) 10Mb 20ms DropTail
$ns duplex-link $n(0) $n(5) 10Mb 20ms DropTail
$ns duplex-link $n(0) $n(6) 10Mb 20ms DropTail
$ns duplex-link $n(1) $n(7) 10Mb 20ms DropTail
$ns duplex-link $n(1) $n(8) 10Mb 20ms DropTail
$ns duplex-link $n(1) $n(9) 10Mb 20ms DropTail
$ns duplex-link $n(1) $n(10) 10Mb 20ms DropTail
$ns duplex-link $n(1) $n(11) 10Mb 20ms DropTail

$ns duplex-link-op $n(0) $n(1) orient left
$ns duplex-link-op $n(0) $n(2) orient up
$ns duplex-link-op $n(0) $n(3) orient right-up
$ns duplex-link-op $n(0) $n(4) orient right
$ns duplex-link-op $n(0) $n(5) orient right-down
$ns duplex-link-op $n(0) $n(6) orient down
$ns duplex-link-op $n(1) $n(7) orient up
$ns duplex-link-op $n(1) $n(8) orient left-up
$ns duplex-link-op $n(1) $n(9) orient left
$ns duplex-link-op $n(1) $n(10) orient left-down
$ns duplex-link-op $n(1) $n(11) orient down
}
# end of create_topology
```