HYbrid Monitoring and Networks Simulation - HYMNS

Jon Crowcroft*, Isidor Kouvelas*, Lorenzo Vicisano*
Internet Research Institute
HP Laboratories Bristol
HPL-IRI -98-002
September, 1998

hipparch, darpa,
merci, hymn,
network monitoring,
NS6,
network simulation

This note is about some of the technical work that UCL CS people have done, partly Jon at HP's Internet Research Institute during the first quarter of 1998, but also previous work using NS under several projects, notably MERCI, HIPAPRCH and DARPA work.

The work described is a series of separate efforts to add protocols to NS, and the design effort to enhance NS to provide interworking between real and simulated (actual and virtual) systems.

Internal Accession Date Only

*University College London, Gower St., London, UK

HYbrid Monitoring and Networks Simulation - HYMNS

Jon Crowcroft
University College London, HP Labs, Bristol,

Isidor Kouvelas and Lorenzo Vicisano, UCL

Abstract

This note is about some of the technical work that UCL CS people have done, partly Jon at HP during the first quarter of 1998, but also previous work using NS under several projects, notably MERCI, HIPAPRCH and DARPA work.

The work described is a series of seperate efforts to add protocols to NS, and then a design effort to enhance NS to provide interworking between real and simulated (actual and virtual) systems. [1]

1 Introduction

The intention was to add features to a network simulator, NS[6] to allow one to run an experiment that was part simulated, part real. To do this the first step was to understand ns.[2]

The original (possibly misguided) goal was to enhance NS to allow experiments to be run that included both real and simulated nodes and agents.

In the next section we give a brief overview of NS. Then we look at the actual systems we were interested in studying i nthe real and simualted world to see what might motivate us to combine actual and virtual networks. Then we take a look at the way one might do this with NS. Finally, we stop and think about what we're trying to do and conclude that maybe, just maybe, its not the best idea in the world, or even in this "cube".

A side effect of this work would be to allow parallel and distributed simulations to be undertaken with NS. This goal appears to also be part of the VINT project, which is the collaboration in the US working mainly on NS.

2 Background on NS

NS is the network simulator from Berkeley written in otcl/C++.

It's based on the notion of adding some basic simulation classes to the otcl interpreter and then building a set of network specific simulated components, and finally, addign a number of protocol specific components.

The base classes include essentially the event and a set of schedulers, together with a set of methods that allow one to build on these.

The network classes added include:

[1] *"like to the lark at break of day arising sings hymns at Heaven's Gate"* - the old bill....shakespeare, to his dark lady, that is, not the sweeney.

[2] Let us begin with a quick look introduction to Ns.....now all turn to page 1 of your K&R.

node

link which includes the notion of connector

queue which includes the notion of drop,

delay

agent

timers

There are also a variety of support classes and methods for statistics and event tracing and monitoring. Each of the added classes makes use of these facilities to build up a protocol stack, and a networked system.

Thus on top of these base simulator classes, there are a number of protocols. Some of these are generic, and others are specific to a particular protocol problem, usually an Internet Protocol of some kind.

Firstly, there are routing, both unicast and multicast typically fairly simple models are in the base system (an SPF/Dijkstra, and a RPF multicast); then there are end to end, transport protocols (e.g. TCP, SRM); finally, session.

The experimentor is either a student learning about an existing protocol, and typically modifying topology configuration, or protocol runtime parameters, or they are a protocol designer, who implements a new protocol, and adds the appropriate C(C++) code modules, and any relevent Tcl (e.g. agent level code and interface code for the user) and then builds new network similator.

A simulation run is simply the execution of an ns with a tcl script tht drives it: initialise a topology of links, nodes, queues and agents, and then start or stop agents at various times, all the while, tracing events to a log file.

Before one constructs a simulation, one might use a variety of tools, such as a topology generator (or a network trace tool to generate a topology file (the tcl that describes the nodes and links and so on), although with dynamic routing, this might be created dynamically. One can use traces, but they are statically captured as arrays of times (and possibly packet sizes) to drive an agent.

After a run, the log file can be animated through an additional tool called *nam*, or else analysed using any tool you wish oto use - typically, a combination of perl, awk and gnuplot or xgraph are often used.

There is a simple agent, called a tap, which allows UDP packets to be captured from a socket on the host (i.e. ones sent by some UDP soruce outside the process that is the running ns instance, possible outside the host the NS process is running on.

3 UCL Experiments

UCL has developed several protocols in research projects which we wished to simulate aspects of since it is often hard to control enough features of the real network to be certain that a protocol exhibits the desired characteristics. Often the protocol is too complex to analyse too.

In fact,m even with nice research networks such as JAMES, LEARNET and the CAIRN, it is hard to control the traffic and topology.

As three example problems to look at, UCL CS researches have recently added three protocols to the world:

Multcp - Philippe Oechslin[4]

RLC - Lornezo Vicisano[2]

SOT - Isidor Kouvelas[3]

I guess we just followed the existing design patterns largely - the thre bits of UCL work (a weighted proportional fair TCP, a reliable multicast transport and self orgnising distributed transcoders) represent both approaches - the TCP stayred with the agent code as per existing TCPs - there's no reason really to change it much, unless we wanted to play with a lot of different variants within the same simulation run maybe

the rlc work and sot work involve a first pass at getting the protocol itself right (typically joint design of real implementation and simulator code, or even a large degree of shared code if all goes well) whereas the "realworld" agent would be soemthign that is part of an application, and therefore has a lot of realworld complexity tro do with the OS environment (e.g. in extremis, is something thast sits above winsock or sockets) whereas an angent in the simulation is simple, and therefore doa-ble in Tcl, but is also typically quite low on event complexity (start something up, stop it, ask it things) sites well in Tcl, while everythign thats hard to get right (looks like a state machine, and operates on packet arrival, timers firing) goes in C++

of course, on re-reading this, i think it should all be re-written in realtime smalltalk:-)

In SOT, we went for the split that most protocols in ns have and implemented the packet forwarding stuff in C++ and all the protocol control in otcl. To improve performance there are a couple of hacks to avoid calling otcl all the time by moving a bit of the control in C++. We used bound variables between otcl and C++. We didn't bother with memory consumption a lot although the random simulations we run were already hitting the limits of my machine (32 megs of RAM).

The goal of the actual pieces of work is

- multcp - follow TCP model
- rlc - like SRM
- sot - a bit like SRM but more application (session level) too

3.1 What we wanted to find out

In many pieces of simulation work there is a goal[3].

Soem common goals in modern protocol and network system design are thigns we wanted to explore in thinking about multcp, rlc and sot- read the papers cited for more details, but typical problems include:

- Fairness (to be unfair in multcp) - there are several definitions of fairness - we give to in the referenced work due to Kelly and Jain.
- Stability (e.g. of contro lscheme in multcp, sot and rlc) - Stability is a notion from contro ltheory, and is well defined in terms of z-transforms, and also intuition!
- Scaling (in memory, messages etc) (of messages in sot, of multicast route memory in rlc) - this is trickier.

[3] sadly, some folks will setout designing a simulation withotu a goal - this is like trying to prove that the sun won't rise tomorrow

- heterogeneity (of link speeds, but also of traffic load on different links and multicast subtrees for sot and rlc). This is the vagues notion, but is also very important in the Internet.

Note these are all dynamic! and adaptive...

4 Design for Modifications to NS to make HYMNS

- ns input to Rmon or other monitoring systems (e.g. tcpdump runnign at LAN sites) - need to generate topology and trace files and programs for where to put what into LAN probes
- Need to get format of files BACK from Rmon probes for
  1. analysis tool
  2. input to trace driver analysis (see nam and ns trace file formats from man page) docs.... (as code/C++ modules/tables for - see http traces and ITA for example formats)
  3. routing - an ns needs to create a "router" that exports IP net numbers for all the links in the system as real IP net numbers (could be a single CIDR prefix)
- a host running this needs to either be (discovered—configured as) a router as far as the other hosts/routers at a site are concerned, or be promiscuous.....

  This could, just could use built in topology+rip, and then it would autmoate itself. This would require
- need to create classes for real traffic (first test is to see if these work for IP and RIP:-) - these would be (for each IP protocol and above) a pair of methods, `realize` and `unrealize`, which would convert external protocol headers into the internal and vice versa, possibly mirroring some of the protocol state i nthe outside eworlds that was only partially present in the simulated world (e.g. actual sequence number in TCP, or virtual IP address and port space).

So basically, there is a new source and sink agent that we configure into an NS that has "real" components, which includes at tap, plus a module that receives packets, and runs a classifier. Action on packets inbound will be initially to discard them as unrecognized (unless we statically configure a topology of real and virtual components), unless they are route update packets. In the latter case, we use these to dynamically creates nodes and links in the NS. In any real scenario (e.g. mbone or unicast public internet) we might expect the topology thus discovered to grow somewhat large!

As we build routes to network numbers, so we can receive packets from real sources there to sinks in the simulation, or start to forward "packets" from simualted sources to real sinks

Of course, for this to work both directions, the simulated topology has to export a set of routes, advertising (apparently) real networks and subnets as reachable (in fact, we can be jolly good citezens here and export a single prefix of course - but the import will still be horrid). Since ns keeps node ids as simple integers, we need to *realize* and *unrealize* addresses.

Once sources and sinks are reachable, the interface agent can run classification on other protocols above IP such as UDP, TCP, SRM, RLC, etc...

5 Why was this doomed from the start?

Well, if we had read the paper by Vern and Sally[1] properly, we might have saved some time.

Basically, the problems with this idea include:

- adaption!

  why its not worth it (cite sally/kevin paper !) since adaption is name of game (multcp, rlc, sot) but visusaltion and background traffic traces are useful to first order (cite mark's mbone stats, but show how they are not part of "self-fulfilling design philosophy of transport/end2end design"

- timeliness

  Of course there is no hope of the simulation keeping pace with the real world accurately. Basically, any performance index that is going to depend on real wall clocks willl suffer in the HYMNS environment.

- repeatability

  The real world is disarmingly unique. Each time you encounter it, it is a different age (as are you, sometimes by a different amount!). This makes for low boredom thresholds, and some surprises. When it comes to simulation, (Raj Jain would be the 1.epsilonth to tell us), we'd like to have increasing confidence as we run each simulation. However, the real world (at least to non Bayesians) is not like that ...

5.1 So can we salvage anything from this Kitchen's Inc?

Well, yes actually - during this process (while looking at SOT and RLC), Mark Handley published a note[5] wherein he wrote, that we can monitor the mbone externally quite succesfully over a period using RTCP logs and mtrace (provided we filter out bogons from the log files).

Now the value of external monitoring (e.g. via Treno, ttcp, pathchar etc) may be limited in the short term, but in the large scale, in a lot of the cases we're interrested here, it may be good enough.

Of course, using the same code in simulation as in implementation does bring a high degree of *validation* to the process. This (especially given the open nature of lower level protocol implementation in the Internet Community) cannot be underestimated.

Which brings me to billing.....:-)

A (Un-)Realization

The source sink agent is based on Steve McCanne's tap, but should use raw IP interface where avaialble; an obvious way to do this is to define a new interface on a host (if multihoming o na phisycai interface is allowed); alternatives for other operating systems include using the SVID Link Provider Interface, writing a BSD Packet Filter driver, and using Microsoft's alternate Transport Level Provider interface in some smart way.

This then basically accepts IP packets, and runs a classifier on them.

It feeds to one very important piece of code, which constructs simulator anlogues of the real world, by creating a shadow set of nodes, links, connectors, queues and so on, many of which are much simpler versions of the existing ones in Ns.

The principal source agent that talks to this code is the dynamic routing modules for unicast and multicast, which have to be statically configured with a neighbour which is the shadow sink (packet

realizer) for route updates to the real world, and source to the routing modeule of route updates received from the real world.

Once "connectivity" is built, then other packets may flow [4].

Then we need to worry about the connectivity graph, and the classification and unrelaization of real packets destied for simulated sinks, and vice versa.

This means that for each simulated protocol, we need to shadow the relevant protocol state in the real world that is not adequately coverd in the simulated protocol - some examples spring to mind such as TCP ISS, and so on.

Certain behaviours occur in simulated time (e.g. RTT estimation) which are tricky to do without merging real RTT estimation from the "edge" of the simualtor to the real sink, with the simulated time.

Numerous other things will emerge if this system is really built!

B References etc

References

[1] Paxson, V., and Floyd, S., Why We Don't Know How To Simulate The Internet, Proceedings of the 1997 Winter Simulation Conference, December 1997. (see `http://ftp.ee.lbl.gov/floyd/papers.html`)

[2] L.Vicisano, L.Rizzo, J. Crowcroft Layered Congestion Avoidance for Reliable Multicast IEEE Infocom, San Francisco, April 1998. (see `ftp://cs.ucl.ac.uk/darpa/infocom98.ps`).

[3] I.Kouvelas, V.Hardman, J.Crowcroft Self Organising Transcoders submitted to NOSSDAV (see `http://www.cs.ucl.ac.uk/staff/I.Kouvelas/publications/sot.ps.gz`).

[4] Philippe Oechslin, J Crowcroft Weighted Proportional Fairness and Pricing for TCP submitted to ACM CCR (see `ftp://cs.ucl.ac.uk/darpa/multcp.ps`).

[5] Mark Handley's Mbone Measurements and nam animation thereof, See `http://north.east.isi.edu/mbonemon/`

[6] See `http://www-mash.cs.berkeley.edu/ns/` for the details about NS.

[4] actually, broadcast and multicast could obviously flow before this!