NETWORK TRAFFIC COLLECTION AND CHARACTERIZATION

By

Dahong Li

A Project Submitted to the Graduate

Faculty of Rensselaer Polytechnic Institute

in Partial Fulfillment of the

Requirements for the Degree of

MASTER OF COMPUTER SCIENCE

Approved:

Boleslaw K. Szymanski
Project Adviser

Rensselaer Polytechnic Institute
Troy, New York

April 1999
(For Graduation May 1999)

CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGMENT

The author would like to thank people who give help to make this project possible.

First, my sincere thanks goes to Professor Boleslaw K. Szymanski for his very patient instruction and considerable encouragement which were given to me during the completion of the project. His generous support and insightful guidance enable me to tackle this challenging research project.

Second, my thanks are due to Professor Shivkumar Kalyanaraman and Professor Kenneth S. Vastola for many constructive ideas, discussions and suggestions in group meetings, as well as colaborations of their groups.

I would like to thank our colleages in CS and ECSE Departments who co-work with me. I should thank Biplab Sikdar for many useful suggestions, discussions and collaboration. I should also thank Jiang Li, Bin Mo, David Harrison for their professionalcollaboration and many constructive suggestions and discussions.

Finally, many thanks to my wife Shan for her support and understanding.

CHAPTER 1
INTRODUCTION

1.1 Introduction

In the recent years the management of large-scale heterogenous networks has attract much more attention due to the fast growth in the global network usage and application. The complexity, heterogeneity and speed of the Next Generation Internet (NGI) require new , scalable approaches to network management and control.

The whole project[1], propose to develop a system of collaborative, on-line simulations which supports a suite of distributed network management and control functions.

Simulation has traditionally been a performance analysis tool used for validating models and measuring deviations from expected performance. According to this point, the whole project was proposed a three-stage approach:

1. Use on-line simulation to achieve desirable network protocol functions and demonstrate that autonomous on-line simulations at network nodes can add value to existing network protocols by operating on a larger time-scale and incorporating current and historical information.

2. Extend this system of autonomous simulators into a system of collaborative simulators which communicate judiciously over the network. The proposed system is different from what is commonly referred to as "distributed simulation" in which a large simulation is distributed among many sites to improve performance.

3. Demonstrate concrete network engineering applications of this collaborative on-line simulation facility.

The whole project was proposed to build a system of distributed, collaborative on-line simulator which will be integrated with a suite of network protocol functions such as advanced feedback-based traffic management, stabilizing internet routing, and admission control for premium network traffic classes. The goal will be to improve automatic network management due to increased awareness of the global and long-range temporal information about network state gathered by each node

and processed by the on-line simulators.

I am focused on design and implementation a traffic collection and characterization model, called traffic collection model. It can be integrated with traffic generation model, ERICA[5] algorithm and NS network simulator[2]. Three stage approach is proposed as following:

1. Simulated System. Design and implementation traffic collection in NS based on SNMP standard. Incorporate method for traffic generation based on collected traces.

2. Router-based System. Implement traffic collection in the CS department router using SNMP and implement traffic generation based on collected traces.

3. Experiment. Compare the traffic generated by the system with real traffic on the router. Use different trace period.

1.2 Network Simulator Software Package NS

Our traffic model sit on the network simulator (NS)[2] package developed in a DARPA funded research project as a real network simulation tool. NS is a discrete event simulator targeted at networking research. NS provides substantial support for simulation of TCL, routing and multicast protocol.

NS began as a variant of the REAL network simulator in 1989 and has evolved substantially over the past few years. The NS development effort is now an on-going effort of research and development.

We use the NS v2 as a real network simulation for this project. NS v2 is written in C++; it uses OTcl as a command and configuration interface. It decomposes the more complex object into simpler components for greater flexibility and comportablility; it uses OTcl, an object oriented version of Tcl, to make the configuration interface; the interface code to the OTcl interpreter is separate from the main simulator. We use it as a tool to facilitate our task and goals.

CHAPTER 2
SIMULATED SYSTEM

2.1 Traffic Collection Based on SNMP Standard

As the traffic collection model will implemented on a real network, the traffic information collected in simulation must available in real network. Our traffic collection model is based on the Simple Network Management Protocol (SNMP)[3]. The SNMP provides a standardized network management framework for enabling the control and monitoring of an internetwork. SNMP is based on Internet standards that define its three major components: the Structure of Management Information (SMI), the Management Information Base (MIB), and the protocol itself. The SMI specifics about the managed objects that the MIB will manage.

2.1.1 SNMP Managed Objects

Every managed object has a unique, administratively assigned name, an ASN.1 syntax for the abstract data structure that represents the object, and a corresponding encoding for its representation and communication using protocol. Figure 2.1 shows the highest common level at which the objects are represented by the SMI.

The objects are organized in group. There are system group, the interfaces table, the address translation group, the ip group, the ICMP group, TCP group,

OBJECT: object descriptor object identifier SYNTAX: ASN.1 syntax for object's abstract data structure DEFINITION: a description of the object in printable ASCII characters ACCESS: read-only or read-write or write-only or not-accessible STATUS: mandatory or optional or obsolete

Figure 2.1: Top-Level Managed Object Template

UDP group, EGP group and SNMP group. TCP group and UDP group have traffic objects. The traffic collection model that is being developed d only implement corresponding information collection for these SNMP traffic objects. There are four traffic objects in UDP group.

```
udpInDatagrams OBJECT-TYPE
    SYNTAX      Counter32
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
            "The total number of UDP datagrams delivered to UDP users."
    ::= { udp 1 }

udpOutDatagrams OBJECT-TYPE
    SYNTAX      Counter32
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
            "The total number of UDP datagrams sent from this entity."
    ::= { udp 4 }
```

The following traffic objects are in TCP group.

```
tcpInSegs OBJECT-TYPE
    SYNTAX      Counter32
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
            "The total number of segments received, including those
            received in error.  This count includes segments received on
            currently established connections."
    ::= { tcp 10 }

tcpOutSegs OBJECT-TYPE
```

```
SYNTAX      Counter32
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
        "The total number of segments sent, including those on
        current connections but excluding those containing only
        retransmitted octets."
::= { tcp 11 }
```

2.1.2 Traffic Collection in NS

NS is an object oriented simulator, written in C++, with an OTcl interpreter as a front-end[2]. The simulator supports a class hierarchy in C++, and a similar class hierarchy within the OTcl interpreter. The two hierarchies are closely related to each other; from the user's perspective, there is a one-to-one correspondence between a class in the interpreted hierarchy and one in the compiled hierarchy. The root of this hierarchy is the class TclObject. The interpreted class hierarchy is automatically established through methods defined in the class TclClass.

There are a number of ways of collecting output or trace data on a simulation. There are two primary but distinct types of monitoring capabilities currently supported by the simulator. The first, called traces, record each individual packets as it arrives, departs, or is dropped at a link or queue. Trace objects are configured into a simulation as nodes in the network topology, representing the destination of collected data. The other types of objects, called monitors, record counts of various interesting quantities such as packet and byte arrivals, departures, etc. Monitors can monitor counts associated with all packets, or on a per-flow basis using a flow monitor.

Monitors are supported by a separate set of objects that are created and inserted into the topology around queues. They provides a place where arrival statistics and times are gathered. Currently NS monitor support in OTcl consists of a number of specialized classes visible in OTcl but implemented in C++, combined with a set of Tcl helper procedures and classes defined in the NS library. The

following OTcl classes are supported by underlying C++ class.

SnoopQueue/In on input, collect a time/size sample (pass packet on)

SnoopQueue/Out on output, collect a time/size sample (pass packet on)

SnoopQueue/Drop on drop, collect a time/size sample (pass packet on)

QueueMonitor receive and aggregate collected samples from snooper

It is clear these monitor can not support SNMP traffic objects described above. A new more generic monitor is designed and implemented. It supports traffic objects in UDP group and TCP group with simplification. As traffic information in NS simulator is not exactly reflect real network, there are approximation exists when implementing traffic objects. The new generic object is called *Snoop/Recv* which can be inserted directly in-line in the network topology. It will supply traffic collection for TCP group and UDP group based on given command in OTcl. The class derivation is as following:

```
class SnoopRecv : public Snoop {...}
class Snoop : public Connector {
  ...
        protected:
                Monitor* qm_;
}

class Monitor : public TclObject {
...
virtual command(int argc, const char*const* argv);
}

int Monitor::command(int argc, const char*const* argv)
{
...
        if (strcmp(argv[1], "get-udpOutDatagrams") == 0) {
                if (udpOutDatagrams_)
                        tcl.resultf("%d", udpOutDatagrams_);
```

```
                else
                        tcl.resultf("");
                return (TCL_OK);
        }
        if (strcmp(argv[1], "get-udpInDatagrams") == 0) {
                if (udpInDatagrams_)
                        tcl.resultf("%d", udpInDatagrams_);
                else
                        tcl.resultf("");
                return (TCL_OK);
        }
}
```

The following example shows how to inserted the generic object into network topology to support traffic collection. Suppose we want to support udpInDatagrams object. First we will insert Snoop/Recv object before link destination entry object. We may have object sequence as following:

```
Before Insertion:
head_=[\$src exitpoint]->queue_->link_->ttl_......->[\$dst entry]
                                        ->drophead_->[null Agent]
After Insertion:
head_=[\$src exitpoint]->queue_->link_->ttl_->.....->SnoopRecv->[\$dst entry]
                                        ->drophead_->[null Agent]
```

The NS simulator require the monitor methods associated with the OTcl Link class. *Snoop/Recv* object is incorporated in init-monitor and attach-monitor. Two monitor help commands are implemented and may be used within simulation scripts to help in attaching monitor element for traffic collection.

sample-timeout-udp Support SNMP UDP group traffic object

sample-timeout-tcp Partially support SNMP TCP group traffic object

These two help functions will use command get-udpInDatagrams and get-udpOutDatagrams to get corresponding object information based on SNMP traffic objects.

2.2 Traffic Characterization for Self-similar Traffic Generation Model

To support traffic generation model of Self-Similar network traffic[4], another traffic parameter set is supported by the traffic collection model. The traffic collection model is able to collect traffic data at specific link with given start time and time interval. Traffic generation support self-similar traffic[4]. The traffic parameter set is Hurst parameter H, average packet arrival rate R, average packet size in bytes and correlation.

2.2.1 Parameter Definition and Property

The importance of self-similar processes lies in the fact that they provide an elegant explanation and interpretation of an empirical law that is commonly referred to the Hurst effect. Briefly, for a given set of observations $(X_k : k = 1, 2, \ldots, n)$ with sample mean $\overline{X}(n)$ and sample variance $S^2(n)$, the rescaled adjusted range statistic (or R/S statistic) is given by $\frac{R(n)}{S(n)} = \frac{1}{S(n)}[max(0, W_1, W_2, \ldots, W_n) - min(0, W_1, W_2, \ldots, W_n)]$, with $W_k = (X_1 + X_2 + \ldots + X_k) - k\overline{X}(n)(k \geq 1)$. The objective of the R/S analysis infers the degree of self-similarity H (Hurst parameter). While many naturally occurring time series appear to be well represented by the relation E[R(n)/S(n)]$\sim cn^H$, as $n \to \infty$. In the network measurements, we typically deal with time series with hundreds of thousands of observations and are therefore able to employ statistical and data analytic techniques that are impractical for small data sets. The graphical tools pox plots of R/S is applied to the collected data sets to calculate an estimate $\hat{a}H$ of the Hurst parameter of H.

The traffic characterization of collected traffic data will give us Hurst parameter H, average packet arrival rate R, average packet size and correlation. A set of observations $(X_k : k = 1, 2, \ldots, n)$ is given. The average packet arrival rate R is:

$$R = \frac{\sum_{k=1}^{n} X_k}{n \times t} \qquad t : timeinterval \tag{2.1}$$

The correlation is:

$$correlation = \frac{\sum_{k=2}^{n}(X_k - \overline{X})(X_{k-1} - \overline{X})}{\sum_{k=1}^{n}(X_k - \overline{X})^2} \tag{2.2}$$

The Hurst parameter[4] is given:

$$W_k = (X_1 + X_2 + \ldots + X_k) - k\overline{X}(n) \qquad (k \geq 1) \tag{2.3}$$

$$\frac{R(n)}{S(n)} = \frac{1}{S(n)}[max(0, W_1, W_2, \ldots, W_n) - min(0, W_1, W_2, \ldots, W_n)] \tag{2.4}$$

$$\hat{H} = \frac{\log R_10000/S_10000 - \log R_1000/S_1000}{\log 10000 - \log 1000} \tag{2.5}$$

2.2.2 Experiments

Traffic collection and characterization experiments is designed and measured on NS simulator with traffic collection monitor and self-similar traffic generator. Self-similar traffic is generated with specified Hurst parameter, rate, correlation, etc. A traffic collection monitor sit on the traffic link and collects the traffic data. The traffic data will feed in characterization model to get a new set parameter. The new set of parameter will compared with the specified parameter set.

Figure 2.2: Experiment Flow Chart

User
↓ parameter
Self-similar Traffic Generator
↓ traffic
Traffic Collection Monitor
↓ traffic trace
Traffic Characterization
↓ parameter

The experiments measurement are shown in table 2.1. It is clear that experiment 2 do not catch the pattern of the self-similar traffic. The reason is that the observation series is too coarse, $R \times interval = 56.8 \times 0.1 = 5.68$ compared to experiment one 0.0568. A picture is worth a thousand word. Figure 2.3 2.5 shows the analysis of experiments in rescaled adjusted range plot.

Table 2.1: Self-similar Traffic Collection and Characterization Experiments

Parameter	Experiment 1		Experiment 2		Experiment 3	
Hurst Parameter	0.92	0.9147	0.92	0.3722	0.92	0.9143
Correlation	0.356	0.0039	0.356	0.2214	0.356	0.027
Rate	5.68	6.74	56.8	70.85	56.8	52.45
Interval	N/A	0.01	N/A	0.1	N/A	0.001
Scale	5	4	5	4	5	4
Size	210	210.0	210	210.0	210	210.0

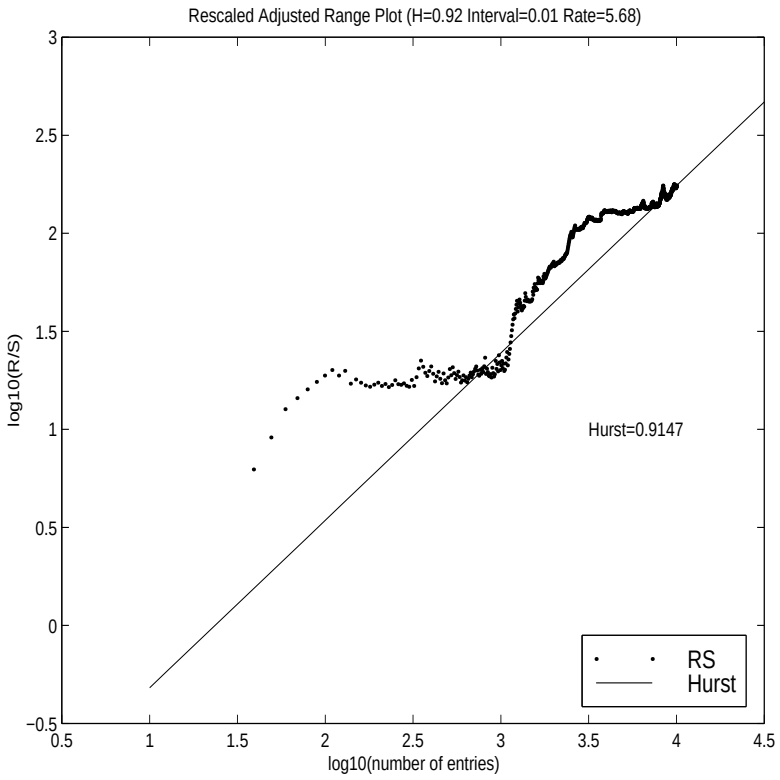

Figure 2.3: Checking the Self-similarity Property of Experiment 1

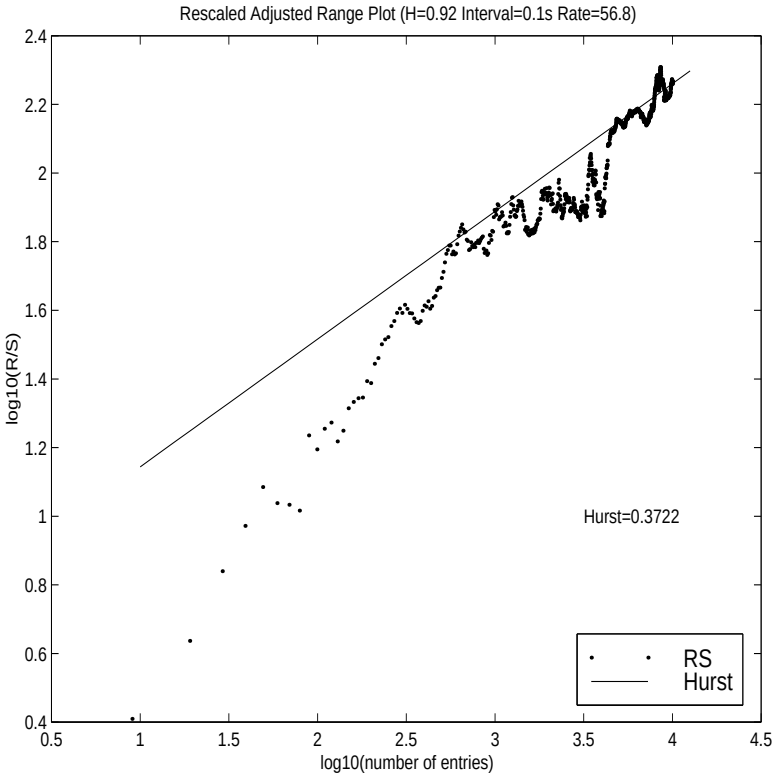

Figure 2.4: Checking the Self-similarity Property of Experiment 2

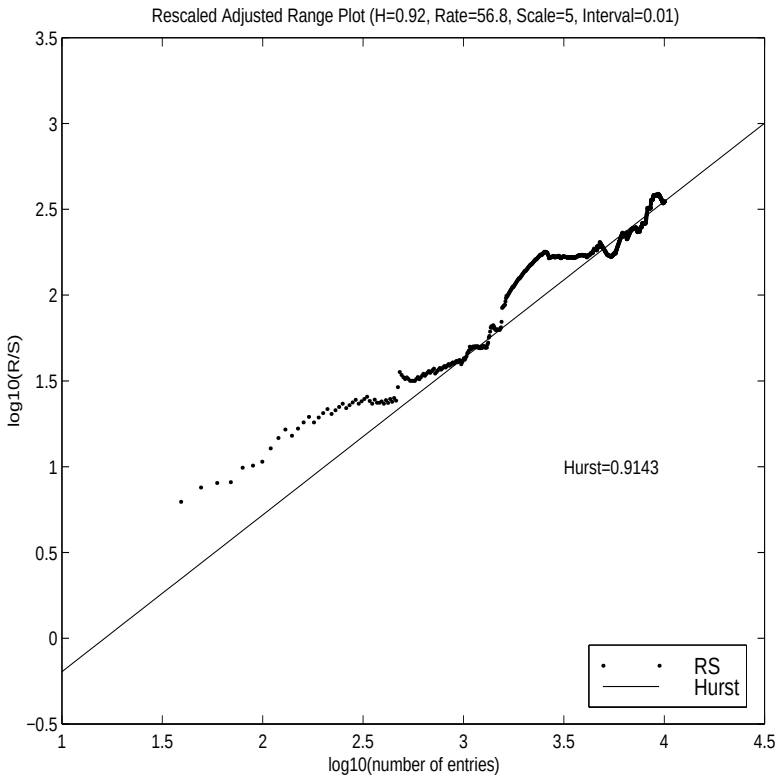

Figure 2.5: Checking the Self-similarity Property of Experiment 3

CHAPTER 3
Real Network Traffic Collection Based on SNMP

3.1 Introduction

Real network traffic collection are based on SNMP. Traffic objects defined in SNMP can be retrieved. A SNMP API package from Advent Inc. is utilized to realize real network traffic collection. A SNMP object (variable) retrieve tool is implemented in Java.

The SNMP variables are organized in hierarchy. The SNMP traffic objects udpInDatagrams, udpOutDatagram, tcpInSegs, tcpOutSegs in the hierarchy are shown in Figure 3.1. Every SNMP objects ID is defined in the hierarchy tree. For example, the object ID of udpOutDatagrams is iso.org.dod.internet.management.mib-2.udp.udpOutDatagrams. The corresponding numerical object ID of udpOutDatagrams is: 1.3.6.1.2.1.7.4.

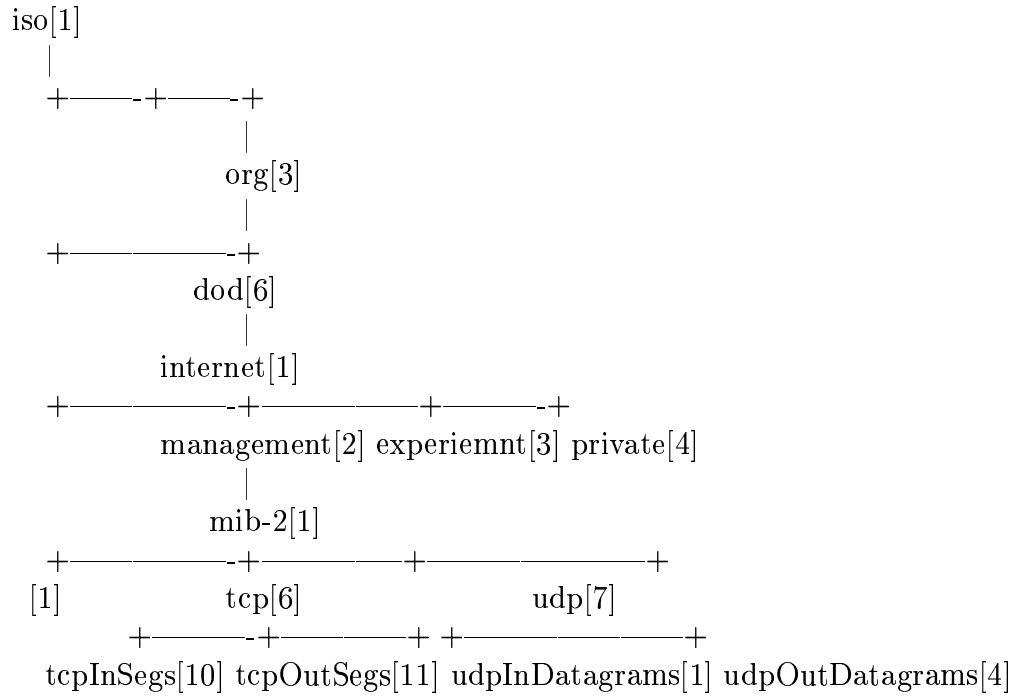

Figure 3.1: SNMP Objects Hierarchy

3.2 Experiment and Analysis

The SNMP object udpOutDatagrams is collected at CS router eggbeater.cs.rpi.edu at specified time interval. The raw data format is

924765534358 < −−time stamp

Object ID: .1.3.6.1.2.1.7.4.0

Counter: 244960 < −−object value

924765535371

Object ID: .1.3.6.1.2.1.7.4.0

Counter: 245111

Experiements collect SNMP object udpOutDatagrams at RPI Computer Science Department router eggbeater.cs.rpi.edu at time interval one second and eight second separately. In the experiment 1, ten thousand samples with time interval 1 second are collected at 2:00pm-5:00pm on April 22 Figure 3.2. Ten thousand samples with time interval 8 second are collected at 9:39pm May 2 - 8:07pm May 3 in experiment 2 Figure 3.3. The Hurst parameter is much below 0.5 which is observed for short term self-similar practically[4]. From the collected data timestamp, the outOutDatagrams update around every 92 seconds when we do the experiment 1. Altough we observes 10000 samples, it actually only catch around 110 variable update. We can say the experiment 1 is short term traffic collection. This is the reason that the experiment 1 not looks like self-similar. In experiment 2, it catches around 1700 variable updates. It is clear the traffic is self-similar as the Hurst parameter is above 0.7 which is a criteria for self-similar traffic parctically [4].

Table 3.1: SNMP udpOutDatagrams Collection from Router eggbeater.cs.rpi.edu

Parameter	Experiment 1	Experiment 2
Hurst Parameter	0.1659	0.8695
Interval(second)	1	8
Samples	10,000	10,000

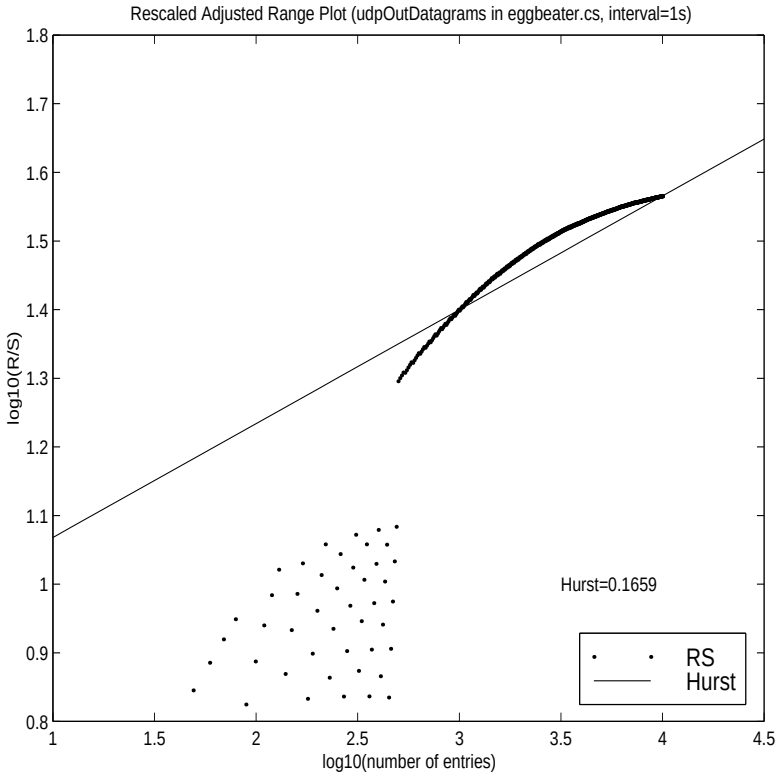

Figure 3.2: Hurst Parameter of Traffic object updOutDatagrams from eggbeater (1 second)

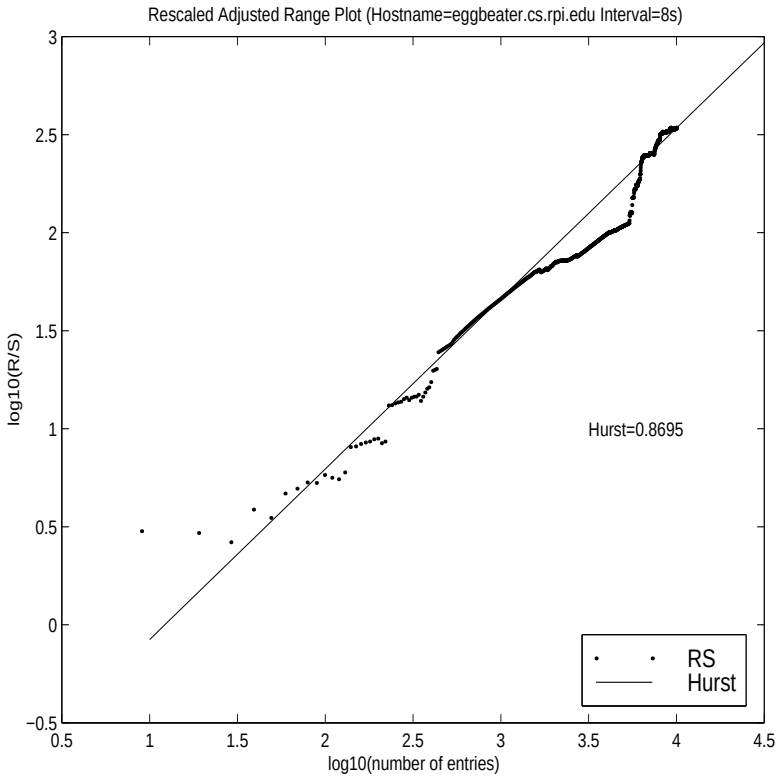

Figure 3.3: Hurst Parameter of Traffic object updOutDatagrams from eggbeater (8 second)

LITERATURE CITED

[1] Boleslaw K. Szymanski, Shivkumar Kalyanaraman, Kenneth S. Vastola, and Chuanyi Ji, "Network Management and Control Using On-Line Collaborative Simulation", project funded by DARPA-ITO. Contract number F19628-98-C-0057.

[2] Kevin Fall and Kannan Varadhan, "ns Notes and Documentation".

[3] Sean Harnedy, "Total SNMP: Exploring the Simple Network Management Protocol", v2, Prentice Hall, 1997.

[4] Will E. Leland, Murad S. Taqqu, Walter Willinger and Daniel V. Wilson, "On the Self-Similar Nature of Ethernet Traffic (Extended Version)", IEEE/ACM Transactions on networking, Vol 2. No. 1 February 1994.

[5] A. Charny, D.D. Clark, R. Jain, "Congestion Control with Explicit rate Indication", In Proceeding of ICC'95, June 1995.

APPENDIX A
SOURCE CODE

```
-----queue-monitor.h

#ifndef ns_queue_monitor_h
#define ns_queue_monitor_h

#include "integrator.h"
#include "connector.h"
#include "packet.h"

#define SNMP_
#define MAX_SIZE 1000000L

class QueueMonitor : public TclObject {
public:
QueueMonitor() : bytesInt_(NULL), pktsInt_(NULL), delaySamp_(NULL),
size_(0), pkts_(0),
parrivals_(0), barrivals_(0),
pdepartures_(0), bdepartures_(0),
pdrops_(0), bdrops_(0),
srcId_(0), dstId_(0), channel_(0)
{
bind("size_", &size_);
bind("pkts_", &pkts_);
bind("parrivals_", &parrivals_);
bind("barrivals_", &barrivals_);
bind("pdepartures_", &pdepartures_);
bind("bdepartures_", &bdepartures_);
```

```
bind("pdrops_", &pdrops_);
bind("bdrops_", &bdrops_);
bind("off_cmn_", &off_cmn_);
};

int size() const { return (size_); }
int pkts() const { return (pkts_); }
int parrivals() const { return (parrivals_); }
int barrivals() const { return (barrivals_); }
int pdepartures() const { return (pdepartures_); }
int bdepartures() const { return (bdepartures_); }
int pdrops() const { return (pdrops_); }
int bdrops() const { return (bdrops_); }
void printStats();
virtual void in(Packet*);
virtual void out(Packet*);
virtual void drop(Packet*);
#ifdef SNMP_
        // Packet received at queue target, virtual at [$dst entry]
virtual void recv(Packet*);
#endif
virtual void edrop(Packet*) { abort(); }; // not here
virtual int command(int argc, const char*const* argv);
protected:
Integrator *bytesInt_; // q-size integrator (bytes)
Integrator *pktsInt_; // q-size integrator (pkts)
Samples* delaySamp_; // stat samples of q delay
int size_; // current queue size (bytes)
int pkts_; // current queue size (packets)
// aggregate counters bytes/packets
int parrivals_;
```

```
int barrivals_;
int pdepartures_;
int bdepartures_;
int pdrops_;
int bdrops_;
int off_cmn_;
int srcId_;
int dstId_;
Tcl_Channel channel_;
#ifdef SNMP_
double sampleInterval_; // sample timeout interval
        int udpOutDatagrams_;  // The total # of UDP datagrams sent from
// this entity;
int udpInDatagrams_;  // The total number of UDP datagrams delivered to UDP users

int packetArrDest_; // packet arrived at monitor destination
private:
double samplePacketArrDest[MAX_SIZE];
int nSample; // number of samples
void compute_para();  // computer hurst parameter paraAverage, paraDeviation, para
double paraAverage;
double paraDeviation;
double paraHurst;
#endif
};

class SnoopQueue : public Connector {
public:
SnoopQueue() : qm_(0) {}
int command(int argc, const char*const* argv) {
if (argc == 3) {
```

```
if (strcmp(argv[1], "set-monitor") == 0) {
qm_ = (QueueMonitor*)
TclObject::lookup(argv[2]);
if (qm_ == NULL)
return (TCL_ERROR);
return (TCL_OK);
}
}
return (Connector::command(argc, argv));
}
 protected:
QueueMonitor* qm_;
};

class SnoopQueueIn : public SnoopQueue {
public:
void recv(Packet* p, Handler* h) {
qm_->in(p);
send(p, h);
}
};

class SnoopQueueOut : public SnoopQueue {
public:
void recv(Packet* p, Handler* h) {
qm_->out(p);
send(p, h);
}
};

class SnoopQueueDrop : public SnoopQueue {
```

```
public:
void recv(Packet* p, Handler* h) {
qm_->drop(p);
send(p, h);
}
};

#ifdef SNMP_
class SnoopQueueRecv : public SnoopQueue {
public:
void recv(Packet* p, Handler* h) {
qm_->recv(p);
send(p, h);
}
};
#endif

/*
 * "early drop" QueueMonitor.  Like a normal QueueMonitor,
 * but also supports the notion of "early" drops
 */


class EDQueueMonitor : public QueueMonitor {
public:
EDQueueMonitor() : ebdrops_(0), epdrops_(0) {
bind("ebdrops_", &ebdrops_);
bind("epdrops_", &epdrops_);
}
void edrop(Packet* p) {
hdr_cmn* hdr = (hdr_cmn*)p->access(off_cmn_);
```

```
ebdrops_ += hdr->size();
epdrops_++;
QueueMonitor::drop(p);
}
int epdrops() const { return (epdrops_); }
int ebdrops() const { return (ebdrops_); }
protected:
int ebdrops_;
int epdrops_;
};

class SnoopQueueEDrop : public SnoopQueue {
public:
void recv(Packet* p, Handler* h) {
qm_->edrop(p);
send(p, h);
}
};


#ifndef MAXFLOW
#define MAXFLOW 32
#endif

/*
 * a 'QueueMonitorCompat', which is used by the compat
 * code to produce the link statistics available in ns-1
 */

class QueueMonitorCompat : public QueueMonitor {
public:
```

```
QueueMonitorCompat();
void in(Packet*);
void out(Packet*);
void drop(Packet*);
int command(int argc, const char*const* argv);
protected:
void flowstats(int flowid); /* create a flowstats structure */
int off_ip_;

int pkts_[MAXFLOW];
int bytes_[MAXFLOW];
int drops_[MAXFLOW];
Samples *flowstats_[MAXFLOW];
};

#endif
```

-----queue-monitor.cc

```
#ifndef lint
static const char rcsid[] =
    "@(#) $Header: /usr/src/mash/repository/vint/ns-2/queue-monitor.cc,v 1.19 1998
#endif

#include "queue-monitor.h"

int QueueMonitor::command(int argc, const char*const* argv)
{
Tcl& tcl = Tcl::instance();
```

```
if (argc == 2) {
if (strcmp(argv[1], "get-bytes-integrator") == 0) {
if (bytesInt_)
tcl.resultf("%s", bytesInt_->name());
else
tcl.resultf("");
return (TCL_OK);
}
if (strcmp(argv[1], "get-pkts-integrator") == 0) {
if (pktsInt_)
tcl.resultf("%s", pktsInt_->name());
else
tcl.resultf("");
return (TCL_OK);
}
if (strcmp(argv[1], "get-delay-samples") == 0) {
if (delaySamp_)
tcl.resultf("%s", delaySamp_->name());
else
tcl.resultf("");
return (TCL_OK);
}
#ifdef SNMP_
                if (strcmp(argv[1], "get-udpOutDatagrams") == 0) {
                     if (udpOutDatagrams_)
tcl.resultf("%d", udpOutDatagrams_);
  else
tcl.resultf("");
return (TCL_OK);
  }
```

```
if (strcmp(argv[1], "get-udpInDatagrams") == 0) {
if (udpInDatagrams_)
tcl.resultf("%d", udpInDatagrams_);
else
tcl.resultf("");
return (TCL_OK);
}
  if (strcmp(argv[1], "new-sample-timeout") == 0) {
samplePacketArrDest[nSample] = packetArrDest_;
nSample++;
if (nSample > MAX_SIZE)
printf("%s", "Error: greater than array size (QueueMonitor)");
// Caluate paraAverage, paraDeviation, paraHurst
compute_para();
packetArrDest_ = 0;
return (TCL_OK);
}
if (strcmp(argv[1], "get-paraAverage") == 0) {
tcl.resultf("%f", paraAverage);
return (TCL_OK);
}
if (strcmp(argv[1], "get-paraDeviation") == 0) {
tcl.resultf("%f", paraDeviation);
return (TCL_OK);
}
if (strcmp(argv[1], "get-paraHurst") == 0) {
tcl.resultf("%f", paraHurst);
return (TCL_OK);
}
#endif
}
```

```
if (argc == 3) {
if (strcmp(argv[1], "set-bytes-integrator") == 0) {
bytesInt_ = (Integrator *)
TclObject::lookup(argv[2]);
if (bytesInt_ == NULL)
return (TCL_ERROR);
return (TCL_OK);
}
if (strcmp(argv[1], "set-pkts-integrator") == 0) {
pktsInt_ = (Integrator *)
TclObject::lookup(argv[2]);
if (pktsInt_ == NULL)
return (TCL_ERROR);
return (TCL_OK);
}
if (strcmp(argv[1], "set-delay-samples") == 0) {
delaySamp_ = (Samples*)
TclObject::lookup(argv[2]);
if (delaySamp_ == NULL)
return (TCL_ERROR);
return (TCL_OK);
}
if (strcmp(argv[1], "trace") == 0) {
int mode;
const char* id = argv[2];
channel_ = Tcl_GetChannel(tcl.interp(), (char*)id, &mode);
if (channel_ == 0) {
tcl.resultf("trace: can't attach %s for writing", id);
return (TCL_ERROR);
}
```

```
return (TCL_OK);
}
#ifdef SNMP_
                if (strcmp(argv[1], "set-udpOutDatagrams") == 0) {
            udpOutDatagrams_ = atoi(argv[2]);
   return (TCL_OK);
}
if (strcmp(argv[1], "set-udpInDatagrams") == 0) {
udpInDatagrams_ = atoi(argv[2]);
return (TCL_OK);
}
if (strcmp(argv[1], "set-sampleInterval") == 0) {
sampleInterval_ = atof(argv[2]);
return (TCL_OK);
}
if (strcmp(argv[1], "set-packetArrDest") == 0) {
packetArrDest_ = atoi(argv[2]);
return (TCL_OK);
}
#endif
}
if (argc == 4) {
if (strcmp(argv[1], "set-src-dst") == 0) {
srcId_ = atoi(argv[2]);
dstId_ = atoi(argv[3]);
return (TCL_OK);
}
}
return TclObject::command(argc, argv); // else control reaches end of
// non-void function, see? :-)
}
```

```
static class QueueMonitorClass : public TclClass {
 public:
QueueMonitorClass() : TclClass("QueueMonitor") {}
TclObject* create(int, const char*const*) {
return (new QueueMonitor());
}
} queue_monitor_class;

void
QueueMonitor::printStats() {
char wrk[500];
int n;
double now = Scheduler::instance().clock();
sprintf(wrk, "%-6.3f %d %d %d %d", now, srcId_, dstId_, size_, pkts_);
n = strlen(wrk);
wrk[n] = '\n';
wrk[n+1] = 0;
(void)Tcl_Write(channel_, wrk, n+1);
wrk[n] = 0;
}

// packet arrival to a queue
void QueueMonitor::in(Packet* p)
{
hdr_cmn* hdr = (hdr_cmn*)p->access(off_cmn_);
double now = Scheduler::instance().clock();
int pktsz = hdr->size();

barrivals_ += pktsz;
parrivals_++;
```

```
size_ += pktsz;
pkts_++;
if (bytesInt_)
bytesInt_->newPoint(now, double(size_));
if (pktsInt_)
pktsInt_->newPoint(now, double(pkts_));
if (delaySamp_)
hdr->timestamp() = now;
if (channel_)
printStats();

}

void QueueMonitor::out(Packet* p)
{
hdr_cmn* hdr = (hdr_cmn*)p->access(off_cmn_);
double now = Scheduler::instance().clock();
int pktsz = hdr->size();

size_ -= pktsz;
pkts_--;
bdepartures_ += pktsz;
pdepartures_++;
if (bytesInt_)
bytesInt_->newPoint(now, double(size_));
if (pktsInt_)
pktsInt_->newPoint(now, double(pkts_));
if (delaySamp_)
delaySamp_->newPoint(now - hdr->timestamp());
if (channel_)
printStats();
```

```
#ifdef SNMP_
// Bug: CBR traffic generator reset packet type, it always use UDP?
if (hdr->ptype() == PT_UDP || hdr->ptype() == PT_CBR)
udpOutDatagrams_++;
#endif
}

void QueueMonitor::drop(Packet* p)
{
hdr_cmn* hdr = (hdr_cmn*)p->access(off_cmn_);
double now = Scheduler::instance().clock();
int pktsz = hdr->size();

size_ -= pktsz;
pkts_--;
bdrops_ += pktsz;
pdrops_++;
if (bytesInt_)
bytesInt_->newPoint(now, double(size_));
if (pktsInt_)
pktsInt_->newPoint(now, double(pkts_));
if (channel_)
printStats();
}

#ifdef SNMP_
void QueueMonitor::recv(Packet* p)
{
hdr_cmn* hdr = (hdr_cmn*) p->access(off_cmn_);
int t = hdr->ptype();
```

```
   packetArrDest_++; // packets arrived at monitor destination side

        // Bug: the following packet type all use UDP ?????
if (t == PT_RTP || t == PT_CBR || t == PT_UDP || t == PT_EXP ||
    t == PT_PARETO)
udpInDatagrams_++;
}

void QueueMonitor::compute_para()
{
  double     mom1;      // First moment
  double     mom2;      // Second moment
  double     x_bar;     // Mean (X_bar value)
  double     s;         // Standard deviation (S value)
  double     w;         // W value
  double     r;         // R value
  double     min_w;     // Minimum W value
  double     max_w;     // Maximum W value
  double     rs_value;  // R/S value to be returned
  double     sum;       // Temporary sum value
  long int   i, j;      // Loop counters

  int N;
  double *X;

  N = nSample - 1;
  X = &samplePacketArrDest[1];

  // Loop to compute mean and standard deviation of X
  mom1 = mom2 = 0.0;
```

```
for (i=0; i<N; i++)
{
  mom1 = mom1 + (X[i] / N);
  mom2 = mom2 + (X[i] * X[i] / N);
}
x_bar = mom1;
s = sqrt(mom2 - mom1 * mom1);

paraAverage = (double) packetArrDest_ / sampleInterval_;
paraDeviation = s / sampleInterval_;

// Double loop to find minimum and maximum W values
min_w = max_w = 0.0;
for (i=0; i<N; i++)
{
  sum = 0.0;
  for (j=0; j<i; j++)
    sum = sum + X[j];

  w = sum - (i * x_bar);
  if (w > max_w) max_w = w;
  if (w < min_w) min_w = w;
}

// Compute R value as maximum W minus minimum W
r = max_w - min_w;

// Compute R/S value
rs_value = r / s;

paraHurst = rs_value;
```

```
}
#endif

static class SnoopQueueInClass : public TclClass {
public:
SnoopQueueInClass() : TclClass("SnoopQueue/In") {}
TclObject* create(int, const char*const*) {
return (new SnoopQueueIn());
}
} snoopq_in_class;

static class SnoopQueueOutClass : public TclClass {
public:
SnoopQueueOutClass() : TclClass("SnoopQueue/Out") {}
TclObject* create(int, const char*const*) {
return (new SnoopQueueOut());
}
} snoopq_out_class;

static class SnoopQueueDropClass : public TclClass {
public:
SnoopQueueDropClass() : TclClass("SnoopQueue/Drop") {}
TclObject* create(int, const char*const*) {
return (new SnoopQueueDrop());
}
} snoopq_drop_class;

#ifdef SNMP_
static class SnoopQueueRecvClass : public TclClass {
public:
SnoopQueueRecvClass() : TclClass("SnoopQueue/Recv") {}
```

```
TclObject* create(int, const char*const*) {
return (new SnoopQueueRecv());
}
} snoopq_recv_class;
#endif


static class SnoopQueueEDropClass : public TclClass {
public:
SnoopQueueEDropClass() : TclClass("SnoopQueue/EDrop") {}
TclObject* create(int, const char*const*) {
return (new SnoopQueueEDrop);
}
} snoopq_edrop_class;

static class QueueMonitorEDClass : public TclClass {
public:
QueueMonitorEDClass() : TclClass("QueueMonitor/ED") {}
TclObject* create(int, const char*const*) {
return (new EDQueueMonitor);
}
} queue_monitor_ed_class;

/*
 * a 'QueueMonitorCompat', which is used by the compat
 * code to produce the link statistics used available in ns-1
 *
 * in ns-1, the counters are the number of departures
 */

#include "ip.h"
```

```
QueueMonitorCompat::QueueMonitorCompat()
{
bind("off_ip_", &off_ip_);

memset(pkts_, 0, sizeof(pkts_));
memset(bytes_, 0, sizeof(bytes_));
memset(drops_, 0, sizeof(drops_));
memset(flowstats_, 0, sizeof(flowstats_));
}



/*
 * create an entry in the flowstats_ array.
 */

void
QueueMonitorCompat::flowstats(int flowid)
{
Tcl& tcl = Tcl::instance();

/*
 * here is the deal.  we are in C code.  we'd like to do
 *     flowstats_[flowid] = new Samples;
 * but, we want to create an object that can be
 * referenced via tcl.  (in particular, we want ->name_
 * to be valid.)
 *
 * so, how do we do this?
 *
 * well, the answer is, call tcl to create it.  then,
 * do a lookup on the result from tcl!
```

```
 */

tcl.evalf("new Samples");
flowstats_[flowid] = (Samples*)TclObject::lookup(tcl.result());
if (flowstats_[flowid] == 0) {
abort();
/*NOTREACHED*/
}
}


void QueueMonitorCompat::out(Packet* pkt)
{
hdr_cmn* hdr = (hdr_cmn*)pkt->access(off_cmn_);
hdr_ip* iph = (hdr_ip*)pkt->access(off_ip_);
double now = Scheduler::instance().clock();
int fid = iph->flowid();

if (fid >= MAXFLOW) {
abort();
/*NOTREACHED*/
}
// printf("QueueMonitorCompat::out(), fid=%d\n", fid);
bytes_[fid] += ((hdr_cmn*)pkt->access(off_cmn_))->size();
pkts_[fid]++;
if (flowstats_[fid] == 0) {
flowstats(fid);
}
flowstats_[fid]->newPoint(now - hdr->timestamp());
QueueMonitor::out(pkt);
}
```

```
void QueueMonitorCompat::in(Packet* pkt)
{
hdr_cmn* hdr = (hdr_cmn*)pkt->access(off_cmn_);
double now = Scheduler::instance().clock();
// QueueMonitor::in() *may* do this, but we always need it...
hdr->timestamp() = now;
QueueMonitor::in(pkt);
}

void QueueMonitorCompat::drop(Packet* pkt)
{

hdr_ip* iph = (hdr_ip*)pkt->access(off_ip_);
int fid = iph->flowid();
if (fid >= MAXFLOW) {
abort();
/*NOTREACHED*/
}
++drops_[fid];
QueueMonitor::drop(pkt);
}

int QueueMonitorCompat::command(int argc, const char*const* argv)
{
Tcl& tcl = Tcl::instance();
int fid;
if (argc == 3) {
fid = atoi(argv[2]);
if (strcmp(argv[1], "bytes") == 0) {
if (fid >= MAXFLOW) {
```

```
abort();
/*NOTREACHED*/
}
tcl.resultf("%d", bytes_[fid]);
return TCL_OK;
} else if (strcmp(argv[1], "pkts") == 0) {
if (fid >= MAXFLOW) {
abort();
/*NOTREACHED*/
}
tcl.resultf("%d", pkts_[fid]);
return TCL_OK;
} else if (strcmp(argv[1], "drops") == 0) {
if (fid >= MAXFLOW) {
abort();
/*NOTREACHED*/
}
tcl.resultf("%d", drops_[fid]);
return TCL_OK;
} else if (strcmp(argv[1], "get-class-delay-samples") == 0) {
if (fid >= MAXFLOW) {
abort();
/*NOTREACHED*/
}
if (flowstats_[fid] == 0) {
/*
 * instantiate one if user actually
 * cares enough to ask for it!
 *
 * (otherwise, need to return "",
 * and then special-case caller to
```

```
 * handle this null return.)
 */
flowstats(fid);
}
tcl.resultf("%s", flowstats_[fid]->name());
return TCL_OK;
}
}
return (QueueMonitor::command(argc, argv));
}

static class QueueMonitorCompatClass : public TclClass {
 public:
QueueMonitorCompatClass() : TclClass("QueueMonitor/Compat") {}
TclObject* create(int, const char*const*) {
return (new QueueMonitorCompat);
}
} queue_monitor_compat_class;

------ns-link.tcl
Class Link
Link instproc init { src dst } {
$self next

        #modified for interface code
$self instvar trace_ fromNode_ toNode_ source_ dest_ color_ oldColor_
set fromNode_ [$src getNode]
set toNode_ [$dst getNode]
        set source_ $src
        set dest_ $dst
set color_ "black"
```

```
set oldColor_ "black"

set trace_ ""
}

Link instproc head {} {
$self instvar head_
return $head_
}

Link instproc queue {} {
$self instvar queue_
return $queue_
}

Link instproc link {} {
$self instvar link_
return $link_
}

Link instproc src {} {
$self instvar source_
return $source_
}

Link instproc dst {} {
$self instvar dest_
return $dest_
}

Link instproc fromNode {} {
```

```
$self instvar fromNode_
return $fromNode_
}

Link instproc toNode {} {
$self instvar toNode_
return $toNode_
}

Link instproc cost c {
$self instvar cost_
set cost_ $c
}

Link instproc cost? {} {
$self instvar cost_
if ![info exists cost_] {
set cost_ 1
}
set cost_
}

Link instproc up { } {
$self instvar dynamics_ dynT_
if ![info exists dynamics_] return
$dynamics_ set status_ 1
if [info exists dynT_] {
foreach tr $dynT_ {
$tr format link-up {$src_} {$dst_}
set ns [Simulator instance]
$self instvar fromNode_ toNode_
```

```
$tr ntrace "l -t [$ns now] -s [$fromNode_ id] -d [$toNode_ id] -S UP"
$tr ntrace "v -t [$ns now] link-up [$ns now] [$fromNode_ id] [$toNode_ id]"
}
}
}

Link instproc down { } {
$self instvar dynamics_ dynT_
if ![info exists dynamics_] {
puts stderr "$class::$proc Link $self was not declared dynamic, and cannot be take
return
}
$dynamics_ set status_ 0
$self all-connectors reset
if [info exists dynT_] {
foreach tr $dynT_ {
$tr format link-down {$src_} {$dst_}
set ns [Simulator instance]
$self instvar fromNode_ toNode_
$tr ntrace "l -t [$ns now] -s [$fromNode_ id] -d [$toNode_ id] -S DOWN"
$tr ntrace "v -t [$ns now] link-down [$ns now] [$fromNode_ id] [$toNode_ id]"
}
}
}

Link instproc up? {} {
$self instvar dynamics_
if [info exists dynamics_] {
return [$dynamics_ status?]
} else {
return "up"
```

```
}
}

Link instproc all-connectors op {
foreach c [$self info vars] {
$self instvar $c
if ![info exists $c] continue
if [array size $c] continue
foreach var [$self set $c] {
if [catch "$var info class"] {
continue
}
if ![$var info class Node] { ;# $op on most everything
catch "$var $op";# in case var isn't a connector
}
}
}
}

Link instproc install-error {em} {
$self instvar link_
$em target [$link_ target]
$link_ target $em
}

# Map between two nodes and their link (if any).
Simulator instproc nodes-to-link {src dst} {
$self instvar link_
return $link_([$src id]:[$dst id])
}
Simulator instproc nodes-to-link? {src dst} {
```

```
$self instvar link_
return [info exists link_([$src id]:[$dst id])]
}

Class SimpleLink -superclass Link

SimpleLink instproc init { src dst bw delay q {lltype "DelayLink"} } {
$self next $src $dst
$self instvar link_ queue_ head_ toNode_ ttl_
$self instvar drophead_

        #added for interface code
        $self instvar ifacein_ ifaceout_
        set ifacein_ 0
        set ifaceout_ $dst

set queue_ $q
set link_ [new $lltype]
$link_ set bandwidth_ $bw
$link_ set delay_ $delay

$queue_ target $link_
$link_ target [$dst entry]

#set head_ $queue_ -> replace by the following
# xxx this is hacky
if { [[$q info class] info heritage ErrModule] == "ErrorModule" } {
set head_ [$q classifier]
} elseif { [$src info class] == "DuplexNetInterface" } {
                set head_ [$src exitpoint]
                set ifacein_ $head_
```

```
                $head_ target $queue_
        } else {
                set head_ $queue_
        }

set drophead_ [new Connector]
$drophead_ target [[Simulator instance] set nullAgent_]
$queue_ drop-target $drophead_

# XXX
# put the ttl checker after the delay
# so we don't have to worry about accounting
# for ttl-drops within the trace and/or monitor
# fabric
#
set ttl_ [new TTLChecker]
$ttl_ target [$link_ target]
$self ttl-drop-trace
$link_ target $ttl_
}

#
# should be called after SimpleLink::trace
#
SimpleLink instproc nam-trace { ns f } {
$self instvar enqT_ deqT_ drpT_ rcvT_ dynT_

#XXX
# we use enqT_ as a flag of whether tracing has been
# initialized
if [info exists enqT_] {
```

```
$enqT_ namattach $f
if [info exists deqT_] {
$deqT_ namattach $f
}
if [info exists drpT_] {
$drpT_ namattach $f
}
if [info exists rcvT_] {
$rcvT_ namattach $f
}
if [info exists dynT_] {
foreach tr $dynT_ {
$tr namattach $f
}
}
} else {
$self trace $ns $f "nam"
}
}

#
# Build trace objects for this link and
# update the object linkage
#
# create nam trace files if op == "nam"
#
SimpleLink instproc trace { ns f {op ""} } {

$self instvar enqT_ deqT_ drpT_ queue_ link_ head_ fromNode_ toNode_
$self instvar rcvT_ ttl_ trace_
$self instvar drophead_ ;# idea stolen from CBQ and Kevin
```

```
#snmp
$self instvar ns_ sampleInterval_
set ns_ $ns
set sampleInterval_ 0.1 ;# default sample interval time
#snmp end

set trace_ $f
set enqT_ [$ns create-trace Enque $f $fromNode_ $toNode_ $op]
set deqT_ [$ns create-trace Deque $f $fromNode_ $toNode_ $op]
set drpT_ [$ns create-trace Drop $f $fromNode_ $toNode_ $op]
set rcvT_ [$ns create-trace Recv $f $fromNode_ $toNode_ $op]

$self instvar drpT_ drophead_
set nxt [$drophead_ target]
$drophead_ target $drpT_
$drpT_ target $nxt

$queue_ drop-target $drophead_

# $drpT_ target [$queue_ drop-target]
# $queue_ drop-target $drpT_

$deqT_ target [$queue_ target]
$queue_ target $deqT_

#$enqT_ target $head_
#set head_ $enqT_       -> replaced by the following
        if { [$head_ info class] == "networkinterface" } {
    $enqT_ target [$head_ target]
    $head_ target $enqT_
```

```
    # puts "head is i/f"
        } else {
    $enqT_ target $head_
    set head_ $enqT_
    # puts "head is not i/f"
}

# put recv trace after ttl checking, so that only actually
# received packets are recorded
$rcvT_ target [$ttl_ target]
$ttl_ target $rcvT_

$self instvar dynamics_
if [info exists dynamics_] {
$self trace-dynamics $ns $f $op
}
}

SimpleLink instproc trace-dynamics { ns f {op ""}} {
$self instvar dynT_ fromNode_ toNode_
lappend dynT_ [$ns create-trace Generic $f $fromNode_ $toNode_ $op]
$self transit-drop-trace
$self linkfail-drop-trace
}

#SNMP snmp-sample-udp
#SimpleLink instproc snmp-sample-udp { } {
# $self instvar ns_ deqT_ drqT_ recT_ sampleInterval_ trace_
# $self instvar source_ dest_
#
#        set udpOutDatagram_ [$deqT_ get-udpOutDatagrams]
```

```
#
# if {$trace_ != 0} {
# puts $trace_ "[$ns_ now] [$source_ id] [$dest_ id] $udpOutDatagram_"
# }
# $ns_ at [expr [$ns_ now] + $sampleInterval_] "$self snmp-sample-udp"
#}
#snmp end

SimpleLink instproc ttl-drop-trace args {
$self instvar ttl_
if ![info exists ttl_] return
if {[llength $args] != 0} {
$ttl_ drop-target [lindex $args 0]
} else {
$self instvar drophead_
$ttl_ drop-target $drophead_
}
}

SimpleLink instproc transit-drop-trace args {
$self instvar link_
if {[llength $args] != 0} {
$link_ drop-target [lindex $args 0]
} else {
$self instvar drophead_
$link_ drop-target $drophead_
}
}

SimpleLink instproc linkfail-drop-trace args {
$self instvar dynamics_
```

```
if ![info exists dynamics_] return
if {[llength $args] != 0} {
$dynamics_ drop-target [lindex $args 0]
} else {
$self instvar drophead_
$dynamics_ drop-target $drophead_
}
}

#
# Trace to a callback function rather than a file.
#
SimpleLink instproc trace-callback {ns cmd} {
$self trace $ns {}
foreach part {enqT_ deqT_ drpT_ rcvT_} {
$self instvar $part
set to [$self set $part]
$to set callback_ 1
$to proc handle a "$cmd \$a"
}
}

#
# like init-monitor, but allows for specification of more of the items
# attach-monitors $insnoop $inqm $outsnoop $outqm $dropsnoop $dropqm
#
#SNMP modified
SimpleLink instproc attach-monitors { insnoop outsnoop dropsnoop qmon {recvsnoop '
#end
$self instvar drpT_ queue_ head_ snoopIn_ snoopOut_ snoopDrop_
$self instvar qMonitor_ drophead_
```

```
#SNMP
$self instvar snoopRecv_ ttl_
set snoopRecv_ $recvsnoop
#endif

set snoopIn_ $insnoop
set snoopOut_ $outsnoop
set snoopDrop_ $dropsnoop

$snoopIn_ target $head_
set head_ $snoopIn_

$snoopOut_ target [$queue_ target]
$queue_ target $snoopOut_

set nxt [$drophead_ target]
$drophead_ target $snoopDrop_
$snoopDrop_ target $nxt

#SNMP
$snoopRecv_ target [$ttl_ target]
$ttl_ target $snoopRecv_
#end SNMP

# if [info exists drpT_] {
# $snoopDrop_ target [$drpT_ target]
# $drpT_ target $snoopDrop_
# $queue_ drop-target $drpT_
# } else {
# $snoopDrop_ target [[Simulator instance] set nullAgent_]
# $queue_ drop-target $snoopDrop_
```

```
# }

$snoopIn_ set-monitor $qmon
$snoopOut_ set-monitor $qmon
$snoopDrop_ set-monitor $qmon
#SNMP
$snoopRecv_ set-monitor $qmon
#end SNMP
set qMonitor_ $qmon
}

#
# Insert objects that allow us to monitor the queue size
# of this link.  Return the name of the object that
# can be queried to determine the average queue size.
#
SimpleLink instproc init-monitor { ns qtrace sampleInterval} {
$self instvar qMonitor_ ns_ qtrace_ sampleInterval_

set ns_ $ns
set qtrace_ $qtrace
set sampleInterval_ $sampleInterval
set qMonitor_ [new QueueMonitor]

#SNMP modified
$self attach-monitors [new SnoopQueue/In] \
[new SnoopQueue/Out] [new SnoopQueue/Drop] \
$qMonitor_ [new SnoopQueue/Recv]
#endif

set bytesInt_ [new Integrator]
```

```
$qMonitor_ set-bytes-integrator $bytesInt_
set pktsInt_ [new Integrator]
$qMonitor_ set-pkts-integrator $pktsInt_
#SNMP
set udpOutDatagrams_ 0
     $qMonitor_ set-udpOutDatagrams $udpOutDatagrams_
set udpInDatagrams_ 0
$qMonitor_ set-udpInDatagrams $udpInDatagrams_
$qMonitor_ set-packetArrDest  0
$qMonitor_ set-sampleInterval $sampleInterval_
#endif

return $qMonitor_
}

SimpleLink instproc start-tracing { } {
$self instvar qMonitor_ ns_ qtrace_ sampleInterval_
$self instvar source_ dest_

if {$qtrace_ != 0} {
$qMonitor_ trace $qtrace_
}
$qMonitor_ set-src-dst [$source_ id] [$dest_ id]
}

SimpleLink instproc queue-sample-timeout { } {
$self instvar qMonitor_ ns_ qtrace_ sampleInterval_
$self instvar source_ dest_

set qavg [$self sample-queue-size]
if {$qtrace_ != 0} {
```

```
puts $qtrace_ "[$ns_ now] [$source_ id] [$dest_ id] $qavg"
}
$ns_ at [expr [$ns_ now] + $sampleInterval_] "$self queue-sample-timeout"
}

SimpleLink instproc queue-sample-timeout-udp { } {
$self instvar qMonitor_ ns_ qtrace_ sampleInterval_
$self instvar source_ dest_

      set udpOutDatagrams_ [$qMonitor_ get-udpOutDatagrams]
set udpInDatagrams_ [$qMonitor_ get-udpInDatagrams]

        if {$qtrace_ != 0} {
puts $qtrace_ "[$ns_ now] [$source_ id] [$dest_ id] $udpOutDatagrams_ $udpInDatagr
$qMonitor_ set-udpOutDatagrams 0
$qMonitor_ set-udpInDatagrams 0
}
$ns_ at [expr [$ns_ now] + $sampleInterval_] "$self queue-sample-timeout-udp"
}

SimpleLink instproc queue-sample-timeout-para { } {
$self instvar qMonitor_ ns_ qtrace_ sampleInterval_
$self instvar source_ dest_

        #initial for each sample timeout, caculate parameter
        $qMonitor_ new-sample-timeout

set paraAverage_ [$qMonitor_ get-paraAverage]
set paraDeviation_ [$qMonitor_ get-paraDeviation]
set paraHurst_ [$qMonitor_ get-paraHurst]
```

```
if {$qtrace_ != 0} {
puts $qtrace_ "[$ns_ now] [$source_ id] [$dest_ id] $paraAverage_ $paraDeviation_
}
$ns_ at [expr [$ns_ now] + $sampleInterval_] "$self queue-sample-timeout-para"
}


SimpleLink instproc sample-queue-size { } {
$self instvar qMonitor_ ns_ qtrace_ sampleInterval_ lastSample_

set now [$ns_ now]
set qBytesMonitor_ [$qMonitor_ get-bytes-integrator]
set qPktsMonitor_ [$qMonitor_ get-pkts-integrator]

$qBytesMonitor_ newpoint $now [$qBytesMonitor_ set lasty_]
set bsum [$qBytesMonitor_ set sum_]

$qPktsMonitor_ newpoint $now [$qPktsMonitor_ set lasty_]
set psum [$qPktsMonitor_ set sum_]

if ![info exists lastSample_] {
set lastSample_ 0
}
set dur [expr $now - $lastSample_]
if { $dur != 0 } {
set meanBytesQ [expr $bsum / $dur]
set meanPktsQ [expr $psum / $dur]
} else {
set meanBytesQ 0
set meanPktsQ 0
}
```

```
$qBytesMonitor_ set sum_ 0.0
$qPktsMonitor_ set sum_ 0.0
set lastSample_ $now

return "$meanBytesQ $meanPktsQ"
}


SimpleLink instproc dynamic {} {
$self instvar dynamics_ head_

if [info exists dynamics_] return

set dynamics_ [new DynamicLink]
# $dynamics_ target head_
# set head_ $dynamics_
$dynamics_ target [$head_ target]
$head_ target $dynamics_


$self transit-drop-trace
$self all-connectors isDynamic
}

#
# insert an "error module" after the queue
# point the em's drop-target to the drophead
#
SimpleLink instproc errormodule args {
$self instvar errmodule_ queue_ drophead_ head_
if { $args == "" } {
```

```
return $errmodule_
}

set em [lindex $args 0]
set errmodule_ $em

#set h [$queue_ target]
set h $head_
#$queue_ target $em
set head_ $em
$em target $h

$em drop-target $drophead_
}
```

APPENDIX B
EXAMPLE AND RESULT

```
----SNMP udp group example
udp_monitor.tcl

source ../lib/ns-queue.tcl
set ns [new Simulator]

$ns color 0 blue
$ns color 1 red
$ns color 2 white

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

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

set mf [open out.mon w]

set nf [open out.nam w]
$ns namtrace-all $nf

$ns simplex-link $n0 $n1 5Mb 2ms DropTail



set udp0 [new Agent/UDP]
$ns attach-agent $n0 $udp0
set cbr0 [new Application/Traffic/CBR]
```

```
$cbr0 set packet_size_ 1024
$cbr0 attach-agent $udp0
#$udp0 set packetsize_ 536

set null0 [new Agent/Null]
$ns attach-agent $n1 $null0
$ns connect $udp0 $null0

$ns at 0.0 "$cbr0 start"

$ns at 2.0 "finish"


proc finish {} {
global ns f mf nf
close $f
        close $mf
close $nf
        $ns flush-trace

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

$ns monitor-queue $n0 $n1 $mf
#$ns monitor-queue $n0 $n1 [$ns get-ns-traceall]
set link01 [$ns link $n0 $n1]
$link01 set qBytesEstimate_ 0
$link01 set qPktsEstimate_ 0
$link01 set sampleInterval_ 0.2
```

```
$ns at 1.0 "$link01 queue-sample-timeout-udp"

$ns run

---result-------
1 0 1 110 110
1.2 0 1 22 22
1.3999999999999999 0 1 22 22
1.5999999999999999 0 1 22 22
1.7999999999999998 0 1 22 22
1.9999999999999998 0 1 22 22

-----Self-Similar traffic monitor

source ../lib/ns-queue.tcl
set ns [new Simulator]

$ns color 0 blue
$ns color 1 red
$ns color 2 white

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

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

set mf [open out.mon w]

set nf [open out.nam w]
$ns namtrace-all $nf
```

```
$ns simplex-link $n0 $n1 5Mb 2ms DropTail


set udp0 [new Agent/UDP]
$ns attach-agent $n0 $udp0
set cbr0 [new Application/Traffic/CBR]
$cbr0 set packet_size_ 1024
$cbr0 attach-agent $udp0
#$udp0 set packetsize_ 536

set null0 [new Agent/Null]
$ns attach-agent $n1 $null0
$ns connect $udp0 $null0

$ns at 0.0 "$cbr0 start"

$ns at 2.0 "finish"


proc finish {} {
global ns f mf nf
close $f
        close $mf
close $nf
        $ns flush-trace

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

```
$ns monitor-queue $n0 $n1 $mf
#$ns monitor-queue $n0 $n1 [$ns get-ns-traceall]
set link01 [$ns link $n0 $n1]
$link01 set qBytesEstimate_ 0
$link01 set qPktsEstimate_ 0
$link01 set sampleInterval_ 0.05
$ns at 1.0 "$link01 queue-sample-timeout-para"

$ns run

------result-----------

1 0 1 1100.000000 0.000000 NaN
1.05 0 1 60.000000 0.000000 NaN
1.1000000000000001 0 1 40.000000 10.000000 1.000000
1.1500000000000001 0 1 60.000000 9.428090 1.414214
1.2000000000000002 0 1 60.000000 8.660254 1.732051
1.2500000000000002 0 1 60.000000 8.000000 2.000000
1.3000000000000003 0 1 40.000000 9.428090 2.121320
1.3500000000000003 0 1 60.000000 9.035079 1.897367
1.4000000000000004 0 1 60.000000 8.660254 1.732051
1.4500000000000004 0 1 60.000000 8.314794 2.138090
1.5000000000000004 0 1 40.000000 9.165151 2.400397
1.5500000000000005 0 1 60.000000 8.907235 2.041241
1.6000000000000005 0 1 60.000000 8.660254 1.732051
1.6500000000000006 0 1 60.000000 8.426501 2.190890
1.7000000000000006 0 1 40.000000 9.035079 2.529822
1.7500000000000007 0 1 60.000000 8.844333 2.110579
1.8000000000000007 0 1 60.000000 8.660254 1.732051
1.8500000000000008 0 1 40.000000 9.112902 2.711088
```

```
1.9000000000000008 0 1 60.000000 8.958064 2.356660
1.9500000000000008 0 1 60.000000 8.806948 2.151411
```