**Model Source Application Header File**

/\*

Author: Arif Husen

File: ibps.h

Written: 15/05/2016

Modifed: 15/05/2016

\*/

#include "timer-handler.h"

#include "packet.h"

#include "app.h"

#include "udp-ms.h"

#include <vector>

#include <string>

#include <sstream>

// 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

double last\_ts;

};

class MsApp;

// Sender uses this timer to

// schedule next app data packet transmission time

class SendTimer : public TimerHandler {

public:

SendTimer(MsApp\* t) : TimerHandler(), t\_(t) {}

inline virtual void expire(Event\*);

protected:

MsApp\* t\_;

};

// Reciver uses this timer to schedule

// next ack packet transmission time

class AckTimer : public TimerHandler {

public:

AckTimer(MsApp\* t) : TimerHandler(), t\_(t) {}

inline virtual void expire(Event\*);

protected:

MsApp\* t\_;

};

// Mulitmedia Application Class Definition

class MsApp : public Application {

public:

MsApp();

void send\_ms\_pkt(); // called by SendTimer:expire (Sender)

void send\_ack\_pkt(); // called by AckTimer:expire (Receiver)

//int p = 1;

int profile[6][25];

int BP[6][25];

int PS[6][25];

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\_ms \*mh\_buf); // (Sender)

void adjust\_scale(void); // (Receiver)

void account\_recv\_pkt(const hdr\_ms \*mh\_buf); // (Receiver)

void init\_recv\_pkt\_accounting(); // (Receiver)

double getRate(int bp);

int load\_ti();

int load\_bp();

int load\_ps();

int get\_pktSize(int bp,int prof);

int get\_bp();

double rate[5]; // Transmission rates associated to scale values

double interval\_; // Application data packet transmission interval

// level of the node sending the packets, could be 1 = access, 2 = aggregation, 3= core

int pktsize\_; // Application data packet size

//TclObject\*\* test\_;

int trafprof\_;

int nodelevel\_;

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

//double TFI[5];

double BPS[5];

int USERS\_;

double SRATE\_;

pkt\_accounting p\_accnt;

SendTimer snd\_timer\_; // SendTimer

AckTimer ack\_timer\_; // AckTimer

};

**Model Source Implementation File**

/\*

Author: Arif Husen , VU, MSCS Thesis Research

File: ibps.h

Written: 15/05/2016

Modifed: 15/05/2016

\*/

#include "random.h"

#include "ms-app.h"

#include <iostream>

#include "tcl.h"

#include "tclcl.h"

#include <string>

#include <sstream>

#include <fstream>

#include <stdlib.h>

// Modelsource OTcl linkage class

static class MsAppClass : public TclClass {

public:

MsAppClass() : TclClass("Application/MsApp") {}

TclObject\* create(int, const char\*const\*) {

return (new MsApp);

}

} class\_app\_ms;

// When snd\_timer\_ expires call Modelsource:send\_mm\_pkt()

void SendTimer::expire(Event\*)

{

t\_->send\_ms\_pkt();

}

// When ack\_timer\_ expires call Modelsource:send\_ack\_pkt()

void AckTimer::expire(Event\*)

{

t\_->send\_ack\_pkt();

}

// Constructor (also initialize instances of timers)

MsApp::MsApp() : running\_(0), snd\_timer\_(this), ack\_timer\_(this)

{

//bind("pktsize\_", &pktsize\_);

bind("trafprof\_", &trafprof\_);

bind("nodelevel\_",&nodelevel\_);

bind\_bool("random\_", &random\_);

load\_ti();

load\_bp();

load\_ps();

pktsize\_ = get\_pktSize(get\_bp(),trafprof\_); //initial packet size is bp = 0

}

double MsApp::getRate(int bp) {

double r = 0;

//double BP6[5] = {0,2,4,6,8};

//double profile4[5] = {};

//double profile4[5] = {};

//double profile4[5] = {};

switch (trafprof\_) {

case 0:

rate[0] = profile[0][0]\*1000\*1000;

rate[1] = profile[0][1]\*1000\*1000;

rate[2] = profile[0][2]\*1000\*1000;

rate[3] = profile[0][3]\*1000\*1000;

rate[4] = profile[0][4]\*1000\*1000;

rate[5] = profile[0][5]\*1000\*1000;

BPS[0] = BP[0][0];

BPS[1] = BP[0][1];

BPS[2] = BP[0][2];

BPS[3] = BP[0][3];

BPS[4] = BP[0][4];

r = profile[0][bp]\*1000\*1000;

//cout << (int)trafprof\_ << endl;

break;

case 1:

rate[0] = profile[1][0]\*1000\*1000;

rate[1] = profile[1][1]\*1000\*1000;

rate[2] = profile[1][2]\*1000\*1000;

rate[3] = profile[1][3]\*1000\*1000;

rate[4] = profile[1][4]\*1000\*1000;

rate[5] = profile[1][5]\*1000\*1000;

BPS[0] = BP[1][0];

BPS[1] = BP[1][1];

BPS[2] = BP[1][2];

BPS[3] = BP[1][3];

BPS[4] = BP[1][4];

r = profile[1][bp]\*1000\*1000;

//cout << (int)trafprof\_ << " -" << rate[0] << "- " << rate[1] << endl;

break;

case 2:

rate[0] = profile[2][0]\*1000\*1000;

rate[1] = profile[2][1]\*1000\*1000;

rate[2] = profile[2][2]\*1000\*1000;

rate[3] = profile[2][3]\*1000\*1000;

rate[4] = profile[2][4]\*1000\*1000;

rate[5] = profile[2][5]\*1000\*1000;

BPS[0] = BP[2][0];

BPS[1] = BP[2][1];

BPS[2] = BP[2][2];

BPS[3] = BP[2][3];

BPS[4] = BP[2][4];

r = profile[2][bp]\*1000\*1000;

//cout << (int)trafprof\_ << endl;

break;

case 3:

rate[0] = profile[3][0]\*1000\*1000;

rate[1] = profile[3][1]\*1000\*1000;

rate[2] = profile[3][2]\*1000\*1000;

rate[3] = profile[3][3]\*1000\*1000;

rate[4] = profile[3][4]\*1000\*1000;

rate[5] = profile[3][5]\*1000\*1000;

BPS[0] = BP[3][0];

BPS[1] = BP[3][1];

BPS[2] = BP[3][2];

BPS[3] = BP[3][3];

BPS[4] = BP[3][4];

r = profile[3][bp]\*1000\*1000;

//cout << (int)trafprof\_ << endl;

break;

case 4:

rate[0] = profile[4][0]\*1000\*1000;

rate[1] = profile[4][1]\*1000\*1000;

rate[2] = profile[4][2]\*1000\*1000;

rate[3] = profile[4][3]\*1000\*1000;

rate[4] = profile[4][4]\*1000\*1000;

rate[5] = profile[4][5]\*1000\*1000;

BPS[0] = BP[4][0];

BPS[1] = BP[4][1];

BPS[2] = BP[4][2];

BPS[3] = BP[4][3];

BPS[4] = BP[4][4];

r = profile[4][bp]\*1000\*1000;

//cout << (int)trafprof\_ << endl;

break;

case 5:

rate[0] = profile[5][0]\*1000\*1000;

rate[1] = profile[5][1]\*1000\*1000;

rate[2] = profile[5][2]\*1000\*1000;

rate[3] = profile[5][3]\*1000\*1000;

rate[4] = profile[5][4]\*1000\*1000;

rate[5] = profile[5][5]\*1000\*1000;

BPS[0] = BP[5][0];

BPS[1] = BP[5][1];

BPS[2] = BP[5][2];

BPS[3] = BP[5][3];

BPS[4] = BP[5][4];

r = profile[5][bp]\*1000\*1000;

//cout << (int)trafprof\_ << endl;

break;

}

return r;

}

// OTcl command interpreter

int MsApp::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

tcl.resultf(" the supportMS = " , agent\_->supportMS());

if(agent\_->supportMS()) {

agent\_->enableMS();

}

else {

tcl.resultf("agent \"%s\" does not support Model Source Application", argv[2]);

return(TCL\_ERROR);

}

agent\_->attachApp(this);

return(TCL\_OK);

}

}

return (Application::command(argc, argv));

}

void MsApp::init()

{

scale\_ = 0; // Start at minimum rate

seq\_ = 0; // MM sequence number (start from 0)

pktsize\_ = get\_pktSize(get\_bp(),trafprof\_); // get initial packet size to bp = 0

interval\_ = (double)(pktsize\_ << 3)/(double)getRate(scale\_);

}

void MsApp::start()

{

init();

running\_ = 1;

send\_ms\_pkt();

}

void MsApp::stop()

{

running\_ = 0;

}

// Send application data packet

void MsApp::send\_ms\_pkt()

{

hdr\_ms 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 = get\_pktSize(get\_bp(),trafprof\_); // 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(get\_pktSize(get\_bp(),trafprof\_), (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 MsApp::next\_snd\_time()

{

// Recompute interval in case rate or size chages

int bp = get\_bp();

pktsize\_ = get\_pktSize(bp,trafprof\_);

interval\_ = (double)(pktsize\_ << 3)/(double)getRate(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 MsApp::recv\_msg(int nbytes, const char \*msg)

{

if(msg) {

hdr\_ms\* mh\_buf = (hdr\_ms\*) msg;

if(mh\_buf->ack == 1) {

// If received packet is ACK packet

//cout << "Ack Received " << endl;

set\_scale(mh\_buf);

}

else {

// If received packet is MM packet

account\_recv\_pkt(mh\_buf);

//cout << "seq no is " << mh\_buf->ack << endl;

if(mh\_buf->seq == 0) send\_ack\_pkt();

}

}

}

// Sender sets its scale to what reciver notifies

void MsApp::set\_scale(const hdr\_ms \*mh\_buf)

{

scale\_ = mh\_buf->scale;

//cout << "new scale set ...." << scale\_ << " rate is " << getRate(scale\_)<< endl;

}

void MsApp::account\_recv\_pkt(const hdr\_ms \*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;

p\_accnt.last\_ts = mh\_buf->time;

}

void MsApp::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 MsApp::send\_ack\_pkt(void)

{

double local\_time = Scheduler::instance().clock();

adjust\_scale();

// send ack message

hdr\_ms 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);

//cout << " ACK Send " << p\_accnt.rtt << endl;

// schedul next ACK time

ack\_timer\_.resched(p\_accnt.rtt);

}

void MsApp::adjust\_scale(void)

{

if(p\_accnt.recv\_pkts > 0) {

if (p\_accnt.last\_ts > BPS[0] && p\_accnt.last\_ts <= BPS[1]) { p\_accnt.last\_scale = 0;}

if (p\_accnt.last\_ts > BPS[1] && p\_accnt.last\_ts <= BPS[2]) { p\_accnt.last\_scale = 1;}

if (p\_accnt.last\_ts > BPS[2] && p\_accnt.last\_ts <= BPS[3]) { p\_accnt.last\_scale = 2;}

if (p\_accnt.last\_ts > BPS[3] && p\_accnt.last\_ts <= BPS[4]) { p\_accnt.last\_scale = 3;}

if (p\_accnt.last\_ts > BPS[4]) { p\_accnt.last\_scale = 4;}

}

p\_accnt.recv\_pkts = 0;

p\_accnt.lost\_pkts = 0;

}

int MsApp::load\_ti(){

int status = 0;

ifstream infile;

infile.open ("/home/arifhusen/ns2/ns-2.35/tcl/msprofiles/ti.dat", ifstream::in);

string str;

string tix;

int line = 0;

if (infile.is\_open()) {

while (getline(infile, str))

{

//cout << str << endl;

int i=0;

stringstream ssin(str);

while(ssin.good() && i <= 24){

ssin >> tix;

int b = atoi(tix.c\_str());

profile[line][i] = b; // PS[prfile][ti]

i++;

}

line++;

}

infile.close();

} else {

cout << "Error opening file ti.dat";

}

return status;

}

int MsApp::load\_bp(){

int status = 0;

ifstream infile;

infile.open ("/home/arifhusen/ns2/ns-2.35/tcl/msprofiles/bp.dat", ifstream::in);

string str;

string bpx;

int line = 0;

if (infile.is\_open()) {

while (getline(infile, str))

{

// cout << str << endl;

int i=0;

stringstream ssin(str);

while(ssin.good() && i <= 24){

ssin >> bpx;

int b = atoi(bpx.c\_str());

BP[line][i] = b; // PS[prfile][bp start interval]

i++;

}

line++;

}

infile.close();

} else {

cout << "Error opening file ti.dat";

}

return status;

}

int MsApp::load\_ps(){

int status = 0;

ifstream infile;

infile.open ("/home/arifhusen/ns2/ns-2.35/tcl/msprofiles/ps.dat", ifstream::in);

string str;

string psx;

int line = 0;

if (infile.is\_open()) {

while (getline(infile, str))

{

// cout << str << endl;

int i=0;

stringstream ssin(str);

while(ssin.good() && i <= 24){

ssin >> psx;

int b = atoi(psx.c\_str());

PS[line][i] = b; // PS[prfile][bp start interval]

i++;

}

line++;

}

infile.close();

} else {

cout << "Error opening file ti.dat";

}

return status;

}

int MsApp::get\_pktSize(int bp, int prof) {

int pktsize = PS[trafprof\_][bp];

//cout << pktsize << endl;

return pktsize;

}

int MsApp::get\_bp(){

// this function will determin the current time and return the corresponding bp interval no defined in the bp.dat

double local\_time = Scheduler::instance().clock();

int bp;

//cout << sizeof(BP) <<endl;

for (int i = 0; i <= 24; i++) {

int last\_entry = 24;

if (local\_time > BP[trafprof\_][last\_entry]){ // return the last bp

bp = last\_entry;

//cout << local\_time << ":" << BP[trafprof\_][i] << endl;

} else {

if (local\_time >= BP[trafprof\_][i] && local\_time <= BP[trafprof\_][i+1])

{

bp = i;

//cout << local\_time << ":" << BP[trafprof\_][i] << endl;

}

}

}

//cout << local\_time << ":" << bp << endl;

return bp;

}

**TIPS Header File**

/\*

Author: Arif Husen , VU, MSCS Thesis Research

File: ibps.h

Written: 15/05/2016

Modifed: 15/05/2016

\*/

#include <string.h>

#include "queue.h"

#include "address.h"

#include "config.h"

class Ibps : public Queue {

public:

Ibps();

protected:

void enque(Packet\*);

Packet\* deque();

Packet\* ddq(int dq\_t);

//int eenq(Packet \*p , int enqto , int bp);

int get\_enqto(int qSize);

PacketQueue \*q0\_; // First FIFO queue

PacketQueue \*q1\_; // First FIFO queue

PacketQueue \*q2\_; // Second FIFO queue

PacketQueue \*q3\_; // First FIFO queue

PacketQueue \*q4\_; // Second FIFO queue

//PacketQueue \*q5\_;

//PacketQueue \*q6\_;

int oprof3[4];

int oprof4[4];

int KF[5][5][5]; //int qsizes[level][qno][BP]

int qsizes[5][5][5]; //int qsizes[level][qno][BP]

int cfid;

int \*a3;

int \*a4;

int deq\_turn\_; // 1 for First queue 2 for Second

//int getQ\_size(int BP , int qno);

int load\_kfactors();

int load\_qsizes();

//int getTIK\_max(int max , int BP);

//int next\_max(int max , int BP);

int\* msort(int BP , int oprof[4]);

//int get\_PIndex(int prof , double val);

int get\_BP();

int getFid(int node, int dnode);

int dn\_nodes\_;

int load\_ti();

int deqFactor;

int qu1;

int qu2;

int qu3;

int qu4;

int load\_bp();

//int KF[3][5][4]; // KF[level][BP][queueno]

int profile0[5];

int profile1[5];

int profile2[5];

int profile3[5];

int profile4[5];

int profile5[5];

int BP0[5];

int BP1[5];

int BP2[5];

int BP3[5];

int BP4[5];

int BP5[5];

};

**TIPS Implementation File**

//

// Author: Arif Husen , MS Computer Sciences, Virtual University , Pakistan

// File: ibps.cc

// Written: 01/05/2016 (for ns-2.1b4a)

// Modifed: 01/05/2016 (for ns-2.1b8a)

//

#include "ibps.h"

#include <iostream>

#include <algorithm>

#include <packet.h>

#include <sstream>

#include <fstream>

#include <stdlib.h>

#include <cmath>

static class IbpsClass : public TclClass {

public:

IbpsClass() : TclClass("Queue/Ibps") {}

TclObject\* create(int, const char\*const\*) {

return (new Ibps);

}

} class\_ibps\_round\_robin;

Ibps::Ibps() {

q0\_ = new PacketQueue;

q1\_ = new PacketQueue;

q2\_ = new PacketQueue;

q3\_ = new PacketQueue;

q4\_ = new PacketQueue;

//q6\_ = new PacketQueue;

pq\_ = q0\_;

bind("dn\_nodes\_", &dn\_nodes\_);

deq\_turn\_ = 1;

qu1 = 0;

qu2 = 0;

qu3 = 0;

qu4 = 0;

load\_ti();

load\_bp();

load\_kfactors();

load\_qsizes();

//cout << qsizes[dn\_nodes\_][1][2] <<endl;

}

void Ibps::enque(Packet\* p)

{

hdr\_ip\* iph = HDR\_IP(p);

int bp = get\_BP();

int enqto;

int n = 1;

//int fid;

//cout << qsizes[dn\_nodes\_][bp][1] << endl;

//cout << "hello" << endl;

////////// Handling the access nodes////////////////////////////////////////////////////

if (dn\_nodes\_ == 1) { // there is no downstream node , so just enqueue the packets in q1

q1\_->enque(p);

if (q1\_->length() > qsizes[dn\_nodes\_][1][bp]) {

q1\_->remove(p);

drop(p);

}

}

////////// Handling the Core Nodes nodes////////////////////////////////////////////////////

if (dn\_nodes\_ == 3) { // there is no downstream node , so just enqueue the packets in q1

switch (iph->fid\_) {

case 1 ... 4:

//cout << "hello" << endl;

enqto = get\_enqto(qsizes[dn\_nodes\_][1][bp]);

if (enqto != 0) {

switch (enqto) {

case 1:

q1\_->enque(p);

if (q1\_->length() > qsizes[dn\_nodes\_][1][bp]) {

q1\_->remove(p);

drop(p);

}

break;

case 2:

q2\_->enque(p);

if (q2\_->length() > qsizes[dn\_nodes\_][2][bp]) {

q2\_->remove(p);

drop(p);

}

break;

case 3:

q3\_->enque(p);

if (q3\_->length() > qsizes[dn\_nodes\_][3][bp]) {

q3\_->remove(p);

drop(p);

}

break;

}

}

break;

case 5:

enqto = get\_enqto(qsizes[dn\_nodes\_][2][bp]);

if (enqto != 0) {

switch (enqto) {

case 1:

q1\_->enque(p);

if (q1\_->length() > qsizes[dn\_nodes\_][1][bp]) {

q1\_->remove(p);

drop(p);

}

break;

case 2:

q2\_->enque(p);

if (q2\_->length() > qsizes[dn\_nodes\_][2][bp]) {

q2\_->remove(p);

drop(p);

}

break;

case 3:

q3\_->enque(p);

if (q3\_->length() > qsizes[dn\_nodes\_][3][bp]) {

q3\_->remove(p);

drop(p);

}

break;

}

}

break;

case 6:

enqto = get\_enqto(qsizes[dn\_nodes\_][3][bp]);

if (enqto != 0) {

// eenq(p,enqto,bp);

switch (enqto) {

case 1:

q1\_->enque(p);

if (q1\_->length() > qsizes[dn\_nodes\_][1][bp]) {

q1\_->remove(p);

drop(p);

}

break;

case 2:

q2\_->enque(p);

if (q2\_->length() > qsizes[dn\_nodes\_][2][bp]) {

q2\_->remove(p);

drop(p);

}

break;

case 3:

q3\_->enque(p);

if (q3\_->length() > qsizes[dn\_nodes\_][3][bp]) {

q3\_->remove(p);

drop(p);

}

break;

}

}

break;

}

}

if (dn\_nodes\_ == 4) {

switch (iph->fid\_) {

case 1:

enqto = get\_enqto(qsizes[dn\_nodes\_][1][bp]);

if (enqto != 0) {

//eenq(p,enqto,bp);

switch (enqto) {

case 1:

q1\_->enque(p);

if (q1\_->length() > qsizes[dn\_nodes\_][1][bp]) {

q1\_->remove(p);

drop(p);

}

break;

case 2:

q2\_->enque(p);

if (q2\_->length() > qsizes[dn\_nodes\_][2][bp]) {

q2\_->remove(p);

drop(p);

}

break;

case 3:

q3\_->enque(p);

if (q3\_->length() > qsizes[dn\_nodes\_][3][bp]) {

q3\_->remove(p);

drop(p);

}

break;

case 4:

q4\_->enque(p);

if (q4\_->length() > qsizes[dn\_nodes\_][4][bp]) {

q4\_->remove(p);

drop(p);

}

break;

}

}

break;

case 2:

enqto = get\_enqto(qsizes[dn\_nodes\_][2][bp]);

if (enqto != 0) {

//eenq(p,enqto,bp);

switch (enqto) {

case 1:

q1\_->enque(p);

if (q1\_->length() > qsizes[dn\_nodes\_][1][bp]) {

q1\_->remove(p);

drop(p);

}

break;

case 2:

q2\_->enque(p);

if (q2\_->length() > qsizes[dn\_nodes\_][2][bp]) {

q2\_->remove(p);

drop(p);

}

break;

case 3:

q3\_->enque(p);

if (q3\_->length() > qsizes[dn\_nodes\_][3][bp]) {

q3\_->remove(p);

drop(p);

}

break;

case 4:

q4\_->enque(p);

if (q4\_->length() > qsizes[dn\_nodes\_][4][bp]) {

q4\_->remove(p);

drop(p);

}

break;

}

}

break;

case 3:

enqto = get\_enqto(qsizes[dn\_nodes\_][3][bp]);

if (enqto != 0) {

//eenq(p,enqto,bp);

switch (enqto) {

case 1:

q1\_->enque(p);

if (q1\_->length() > qsizes[dn\_nodes\_][1][bp]) {

q1\_->remove(p);

drop(p);

}

break;

case 2:

q2\_->enque(p);

if (q2\_->length() > qsizes[dn\_nodes\_][2][bp]) {

q2\_->remove(p);

drop(p);

}

break;

case 3:

q3\_->enque(p);

if (q3\_->length() > qsizes[dn\_nodes\_][3][bp]) {

q3\_->remove(p);

drop(p);

}

break;

case 4:

q4\_->enque(p);

if (q4\_->length() > qsizes[dn\_nodes\_][4][bp]) {

q4\_->remove(p);

drop(p);

}

break;

}

}

break;

case 4:

enqto = get\_enqto(qsizes[dn\_nodes\_][4][bp]);

if (enqto != 0) {

// eenq(p,enqto,bp);

switch (enqto) {

case 1:

q1\_->enque(p);

if (q1\_->length() > qsizes[dn\_nodes\_][1][bp]) {

q1\_->remove(p);

drop(p);

}

break;

case 2:

q2\_->enque(p);

if (q2\_->length() > qsizes[dn\_nodes\_][2][bp]) {

q2\_->remove(p);

drop(p);

}

break;

case 3:

q3\_->enque(p);

if (q3\_->length() > qsizes[dn\_nodes\_][3][bp]) {

q3\_->remove(p);

drop(p);

}

break;

case 4:

q4\_->enque(p);

if (q4\_->length() > qsizes[dn\_nodes\_][4][bp]) {

q4\_->remove(p);

drop(p);

}

break;

}

}

break;

} // end of switch

} // end f if

}

int Ibps::load\_qsizes() {

int status = 1;

int sMulti = 1;

int k;

if (dn\_nodes\_ == 1) {

for (int q = 0; q<=4; q++) {

for (int bp = 0; bp <=4; bp++) {

k = sMulti \* KF[dn\_nodes\_][q][bp];

if (q == 0) {qsizes[dn\_nodes\_][q][bp] = 0;} //becasue we are not using the q = 0

if (q == 1) {qsizes[dn\_nodes\_][q][bp] = k+profile0[bp];}

if (q == 2) {qsizes[dn\_nodes\_][q][bp] =0;}

if (q == 3) {qsizes[dn\_nodes\_][q][bp] = 0;}

if (q == 4) {qsizes[dn\_nodes\_][q][bp] = 0;} // because we are not using the q = 4

} //end of bp loop

} //end of q loop

}

if (dn\_nodes\_ == 3){

for (int q = 0; q<=4; q++) {

for (int bp = 0; bp<=4; bp++) {

k = sMulti \* KF[dn\_nodes\_][q][bp];

if (q == 0) {qsizes[dn\_nodes\_][q][bp] = 0;} //becasue we are not using the q = 0

if (q == 1) {qsizes[dn\_nodes\_][q][bp] = k+(profile0[bp]+profile1[bp]+profile2[bp]+profile3[bp]);}

if (q == 2) {qsizes[dn\_nodes\_][q][bp] = k+profile4[bp];}

if (q == 3) {qsizes[dn\_nodes\_][q][bp] = k+profile5[bp];}

if (q == 4) {qsizes[dn\_nodes\_][q][bp] = 0;} // because we are not using the q = 4

} //end of bp loop

} //end of q loop

}

if (dn\_nodes\_ == 4){

for (int q = 0; q<=4; q++) {

for (int bp = 0; bp <=4; bp++) {

k = sMulti \* KF[dn\_nodes\_][q][bp];

if (q == 0) {qsizes[dn\_nodes\_][q][bp] = 0;} //becasue we are not using the q = 0

if (q == 1) {qsizes[dn\_nodes\_][q][bp] = k+profile0[bp];}

if (q == 2) {qsizes[dn\_nodes\_][q][bp] = k+profile1[bp];}

if (q == 3) {qsizes[dn\_nodes\_][q][bp] = k+profile2[bp];}

if (q == 4) {qsizes[dn\_nodes\_][q][bp] = k+profile3[bp];} // because we are not using the q = 4

} //end of bp loop

} //end of q loop

}

return status;

}

Packet\* Ibps::deque(){

Packet \*p;

int BP;

BP = get\_BP();

//-------------------------- Handle All Control Messages------------------------------------------

if (pq\_->length() > 0 ) {

pq\_->deque();

}

//-------------------------- Handle Access Nodes------------------------------------------

if (dn\_nodes\_ == 1) {

p = ddq(1);

}

//-------------------------- Handle Aggregation Nodes------------------------------------------

if (dn\_nodes\_ == 4){

switch (deq\_turn\_) {

case 1:

p = ddq(1);

qu1++;

if(p == 0) {

p = ddq(2);

qu2++;

deq\_turn\_ = 3;

} else {

//cout << "q2 len....." << q2\_->length() <<":" << p << endl;

if (((KF[dn\_nodes\_][1][BP]-qu1) <= 0)) {deq\_turn\_ = 2; qu1 = 0;} else { deq\_turn\_ = 1;}

//cout << deq\_turn\_ <<endl;

//deq\_turn\_ = 1;

}

break;

case 2:

//cout << "case 2" <<endl;

p = ddq(2);

qu2++;

if(p == 0) {

p = ddq(3);

qu3++;

deq\_turn\_ = 4;

} else {

//deq\_turn\_ = 2;

if ((KF[dn\_nodes\_][2][BP]-qu2) <= 0) {deq\_turn\_ = 3; qu2= 0;} else { deq\_turn\_ = 2;}

}

break;

case 3:

//cout << "case 3" <<endl;

p = ddq(3);

qu3++;

if(p == 0) {

p = ddq(4);

qu4++;

deq\_turn\_ = 1;

} else {

//deq\_turn\_ = 3;

if ((KF[dn\_nodes\_][3][BP]-qu3) <= 0) {deq\_turn\_ = 4; qu3= 0;} else { deq\_turn\_ = 3;}

}

break;

case 4:

//cout << "case 4" <<endl;

p = ddq(4);

qu4++;

if(p == 0) {

p = ddq(1);

qu1++;

deq\_turn\_ = 2;

} else {

//deq\_turn\_ = 4;

if ((KF[dn\_nodes\_][4][BP]-qu4) <= 0) {deq\_turn\_ = 1;qu4= 0;} else { deq\_turn\_ = 4;}

}

break;

}

}

//-------------------------- handle core nodes------------------------------------------

if (dn\_nodes\_ == 3) {

switch (deq\_turn\_) {

case 1:

p = ddq(1);

qu1++;

if(p == 0) {

p = ddq(2);

qu2++;

deq\_turn\_ = 3;

} else {

if ((KF[dn\_nodes\_][1][BP] - qu1) <= 0) {deq\_turn\_ = 2; qu1 = 0;} else { deq\_turn\_ = 1;}

}

break;

case 2:

p = ddq(2);

qu2++;

if(p == 0) {

p = ddq(3);

qu3++;

deq\_turn\_ = 1;

} else {

if ((KF[dn\_nodes\_][2][BP]-qu2) <= 0) {deq\_turn\_ = 3; qu2 = 0;} else { deq\_turn\_ = 2;}

}

break;

case 3:

p = ddq(3);

qu3++;

if(p == 0) {

p = ddq(1);

qu1++;

deq\_turn\_ = 2;

} else {

if ((KF[dn\_nodes\_][3][BP]-qu3) <= 0) {deq\_turn\_ = 1; qu3 = 0;} else { deq\_turn\_ = 3;}

}

break;

}

}

return (p);

}

Packet\* Ibps::ddq(int dq\_t) {

Packet\* p;

int bp = get\_BP();

//int k = 2;

int cqsize; // current queue size

if (dn\_nodes\_ == 1) {

p = q1\_->deque();

}

if (dn\_nodes\_ == 4) {

switch(dq\_t) {

case 1:

cqsize = q1\_->length();

if (cqsize > KF[dn\_nodes\_][1][bp]) {

for (int i=1; i <= KF[dn\_nodes\_][1][bp]; i++) {

p = q1\_->deque();

}

} else {p = q1\_->deque();}

break;

case 2:

cqsize = q2\_->length();

if (cqsize > KF[dn\_nodes\_][2][bp]) {

for (int i=1; i <= KF[dn\_nodes\_][2][bp]; i++) {

p = q2\_->deque();

}

}else { p = q2\_->deque();}

break;

case 3:

cqsize = q3\_->length();

if (cqsize > KF[dn\_nodes\_][3][bp]) {

for (int i=1; i <= KF[dn\_nodes\_][3][bp]; i++) {

p = q3\_->deque();

}

}{ p = q3\_->deque();}

break;

case 4:

cqsize = q4\_->length();

if (cqsize > KF[dn\_nodes\_][4][bp]) {

for (int i=1; i <= KF[dn\_nodes\_][4][bp]; i++) {

p = q4\_->deque();

}

}{ p = q4\_->deque();}

break;

}

}

if (dn\_nodes\_ == 3) {

switch(dq\_t) {

case 1:

cqsize = q1\_->length();

if (cqsize > KF[dn\_nodes\_][1][bp]) {

for (int i=1; i <=KF[dn\_nodes\_][1][bp]; i++) {

p = q1\_->deque();

}

} else { p = q1\_->deque(); }

break;

case 2:

cqsize = q2\_->length();

if (cqsize > KF[dn\_nodes\_][2][bp]) {

for (int i=1; i <= KF[dn\_nodes\_][2][bp]; i++) {

p = q2\_->deque();

}

}else { p = q2\_->deque();}

break;

case 3:

cqsize = q3\_->length();

if (cqsize > KF[dn\_nodes\_][3][bp]) {

for (int i=1; i <= KF[dn\_nodes\_][3][bp]; i++) {

p = q3\_->deque();

}

}{ p = q3\_->deque();}

break;

}

}

return (p);

}

int\* Ibps::msort(int BP , int oprof[4]){

int TI\_q1;

int TI\_q2;

int TI\_q3;

int TI\_q4;

if (dn\_nodes\_ == 1 || dn\_nodes\_ == 4){

// get the values of respected BP

TI\_q1 = profile0[BP];

TI\_q2 = profile1[BP];

TI\_q3 = profile2[BP];

TI\_q4 = profile3[BP];

int q\_serving\_order[4]; // TI values of all queus

q\_serving\_order[0] = TI\_q1;

q\_serving\_order[1] = TI\_q2;

q\_serving\_order[2] = TI\_q3;

q\_serving\_order[3] = TI\_q4;

std::sort(q\_serving\_order, q\_serving\_order + 4,std::greater<int>()); // sort the queue values

//Now get the queue numbers in above order

int q1 = 0;

int q2 = 0;

int q3 = 0;

int q4 = 0;

// determine the first queu position ( Assume that all TIs are integer and different from each other)

if (profile0[BP] == q\_serving\_order[0]) { q1 = 1;}

if (profile1[BP] == q\_serving\_order[0]) { q1 = 2;}

if (profile2[BP] == q\_serving\_order[0]) { q1 = 3;}

if (profile3[BP] == q\_serving\_order[0]) { q1 = 4;}

if (profile0[BP] == q\_serving\_order[1]) { q2 = 1;}

if (profile1[BP] == q\_serving\_order[1]) { q2 = 2;}

if (profile2[BP] == q\_serving\_order[1]) { q2 = 3;}

if (profile3[BP] == q\_serving\_order[1]) { q2 = 4;}

if (profile0[BP] == q\_serving\_order[2]) { q3 = 1;}

if (profile1[BP] == q\_serving\_order[2]) { q3 = 2;}

if (profile2[BP] == q\_serving\_order[2]) { q3 = 3;}

if (profile3[BP] == q\_serving\_order[2]) { q3 = 4;}

if (profile0[BP] == q\_serving\_order[3]) { q4 = 1;}

if (profile1[BP] == q\_serving\_order[3]) { q4 = 2;}

if (profile2[BP] == q\_serving\_order[3]) { q4 = 3;}

if (profile3[BP] == q\_serving\_order[3]) { q4 = 4;}

// prepare arrary to return

oprof[0] = q1;

oprof[1] = q2;

oprof[2] = q3;

oprof[3] = q4;

}

if (dn\_nodes\_ == 3){

// get the values of respected BP

TI\_q1 = profile0[BP]+profile1[BP]+profile2[BP]+profile3[BP];

TI\_q2 = profile4[BP];

TI\_q3 = profile5[BP];

TI\_q4 = 0;

int q\_serving\_order[4]; // TI values of all queus

q\_serving\_order[0] = TI\_q1;

q\_serving\_order[1] = TI\_q2;

q\_serving\_order[2] = TI\_q3;

q\_serving\_order[3] = TI\_q4;

sort(q\_serving\_order, q\_serving\_order + 4,greater<int>()); // sort the queue values

//Now get the queue numbers in above order

int q1 = 0;

int q2 = 0;

int q3 = 0;

int q4 = 0;

// determine the first queu position ( Assume that all TIs are integer and different from each other)

int agg\_ti\_prof = profile0[BP]+profile1[BP]+profile2[BP]+profile3[BP];

if (agg\_ti\_prof == q\_serving\_order[0]) { q1 = 1;}

if (profile4[BP] == q\_serving\_order[0]) { q1 = 2;}

if (profile5[BP] == q\_serving\_order[0]) { q1 = 3;}

//if (profile3[BP] == q\_serving\_order[0]) { q1 = 4;}

if (agg\_ti\_prof == q\_serving\_order[1]) { q2 = 1;}

if (profile4[BP] == q\_serving\_order[1]) { q2 = 2;}

if (profile5[BP] == q\_serving\_order[1]) { q2 = 3;}

//if (profile3[BP] == q\_serving\_order[1]) { q2 = 4;}

if (agg\_ti\_prof == q\_serving\_order[2]) { q3 = 1;}

if (profile4[BP] == q\_serving\_order[2]) { q3 = 2;}

if (profile5[BP] == q\_serving\_order[2]) { q3 = 3;}

//if (profile3[BP] == q\_serving\_order[2]) { q3 = 4;}

if (agg\_ti\_prof== q\_serving\_order[3]) { q4 = 1;}

if (profile4[BP] == q\_serving\_order[3]) { q4 = 2;}

if (profile5[BP] == q\_serving\_order[3]) { q4 = 3;}

//if (profile3[BP] == q\_serving\_order[3]) { q4 = 4;}

// prepare arrary to return

oprof[0] = q1;

oprof[1] = q2;

oprof[2] = q3;

oprof[3] = 0;

}

return oprof;

}

int Ibps::get\_BP() {

double ts = Scheduler::instance().clock();

int BP;

if (ts >= 0 && ts <= 2) { BP = 0; }

if (ts > 2 && ts <= 4) { BP = 1; }

if (ts > 4 && ts <= 6) { BP = 2; }

if (ts > 6 && ts <= 8) { BP = 3; }

if (ts > 8) { BP = 4; }

return BP;

}

int Ibps::load\_ti(){

//double ts = Scheduler::instance().clock();

//cout << ts << endl;

int status = 0;

ifstream infile;

infile.open ("/home/arifhusen/ns2/ns-2.35/tcl/msprofiles/ti.dat", ifstream::in);

string str;

string tix;

int line = 0;

if (infile.is\_open()) {

while (getline(infile, str))

{

//cout << str << endl;

int i=0;

stringstream ssin(str);

while(ssin.good() && i <= 5){

if (line == 0) { ssin >> tix; int b = atoi(tix.c\_str()); profile0[i] = b; }

if (line == 1) { ssin >> tix; int b = atoi(tix.c\_str()); profile1[i] = b; }

if (line == 2) { ssin >> tix; int b = atoi(tix.c\_str()); profile2[i] = b; }

if (line == 3) { ssin >> tix; int b = atoi(tix.c\_str()); profile3[i] = b; }

if (line == 4) { ssin >> tix; int b = atoi(tix.c\_str()); profile4[i] = b; }

if (line == 5) { ssin >> tix; int b = atoi(tix.c\_str()); profile5[i] = b; }

//cout << i <<endl;

i++;

}

line++;

}

infile.close();

} else {

cout << "Error opening file ti.dat";

}

return status;

}

int Ibps::load\_bp(){

int status = 0;

ifstream infile;

infile.open ("/home/arifhusen/ns2/ns-2.35/tcl/msprofiles/bp.dat", ifstream::in);

string str;

string bpx;

int line = 0;

if (infile.is\_open()) {

while (getline(infile, str))

{

// cout << str << endl;

int i=0;

stringstream ssin(str);

while(ssin.good() && i < 6){

if (line == 0) { ssin >> bpx; int b = atoi(bpx.c\_str()); BP0[i] = b; }

if (line == 1) { ssin >> bpx; int b = atoi(bpx.c\_str()); BP1[i] = b; }

if (line == 2) { ssin >> bpx; int b = atoi(bpx.c\_str()); BP2[i] = b; }

if (line == 3) { ssin >> bpx; int b = atoi(bpx.c\_str()); BP3[i] = b; }

if (line == 4) { ssin >> bpx; int b = atoi(bpx.c\_str()); BP4[i] = b; }

// cout << BP0[i] <<endl;

i++;

}

line++;

}

infile.close();

} else {

cout << "Error opening file ti.dat";

}

return status;

}

int Ibps::get\_enqto(int qSize) {

int bp = get\_BP();

int enqto = 0;

if (dn\_nodes\_ == 3) {

//a3 = msort(bp,oprof3); // this will return the requested profile in descending order , so that we assing the values

enqto = 1;

}

if (dn\_nodes\_ == 3) {

a3 = msort(bp,oprof3); // this will return the requested profile in descending order , so that we assing the values

for (int i = 0; i<=4; i++){

if (qSize == qsizes[dn\_nodes\_][a3[i]][bp]) { enqto = i+1;}

}

}

if (dn\_nodes\_ == 4) {

a4 = msort(bp,oprof4); // this will return the requested profile in descending order , so that we assing the values

for (int i = 0; i<=4; i++){

if (qSize == qsizes[dn\_nodes\_][a4[i]][bp]) { enqto = i+1;}

}

}

//cout << enqto << endl;

return enqto;

}

int Ibps::load\_kfactors() {

int status = 1;

int MinQ\_no;

int mnqno;

int min\_ti;

//int KF[3][5][5]; //int qsizes[level][qno][BP]

int kfmul = 1;

// for level one all queus have the factor 1

if (dn\_nodes\_ == 1) {

for (int q = 0; q<=4; q++) {

for (int bp = 0; bp <=4; bp++) {

KF[dn\_nodes\_][q][bp] = 1;

}

}

//kf = 1;

}

if (dn\_nodes\_ == 4) {

int oprf4[4];

int \*b4;

int min\_q\_inBP4[5];

//int MinQ\_no;

//int min\_ti;

b4 = msort(0,oprf4); mnqno = b4[2]; min\_q\_inBP4[0] = mnqno;

b4 = msort(1,oprf4); mnqno = b4[2]; min\_q\_inBP4[1] = mnqno;

b4 = msort(2,oprf4); mnqno = b4[2]; min\_q\_inBP4[2] = mnqno;

b4 = msort(3,oprf4); mnqno = b4[2]; min\_q\_inBP4[3] = mnqno;

b4 = msort(4,oprf4); mnqno = b4[2]; min\_q\_inBP4[4] = mnqno;

for (int q =0; q<=4; q++) {

for (int bp = 0; bp <=4; bp++) {

//int oprf4[4];

//int \*b4;

MinQ\_no = min\_q\_inBP4[bp];

if (MinQ\_no == 1) { min\_ti = profile0[bp]; }

if (MinQ\_no == 2) { min\_ti = profile1[bp]; }

if (MinQ\_no == 3) { min\_ti = profile2[bp]; }

if (MinQ\_no == 4) { min\_ti = profile3[bp]; }

if (min\_ti == 0) { min\_ti = 1;}

if (q == 0) {KF[dn\_nodes\_][q][bp] = kfmul+1; }

if (q == 1 && profile0[bp] > 0) {KF[dn\_nodes\_][q][bp] = kfmul+((profile0[bp] / min\_ti)+1); }

if (q == 2 && profile1[bp] > 0) {KF[dn\_nodes\_][q][bp] = kfmul+((profile1[bp] / min\_ti)+1);}

if (q == 3 && profile2[bp] > 0) {KF[dn\_nodes\_][q][bp] = kfmul+((profile2[bp] / min\_ti)+1); }

if (q == 4 && profile3[bp] > 0) {KF[dn\_nodes\_][q][bp] = kfmul+((profile3[bp] / min\_ti)+1); }

//KF[dn\_nodes\_][q][bp] = 1;

}

}

}

if (dn\_nodes\_ == 3) {

//minq[bp] = qno;

int oprf3[4];

int \*b3;

int min\_q\_inBP3[5];

//int MinQ\_no;

//int min\_ti;

b3 = msort(0,oprf3); mnqno = b3[2]; min\_q\_inBP3[0] = mnqno;

b3 = msort(1,oprf3); mnqno = b3[2]; min\_q\_inBP3[1] = mnqno;

b3 = msort(2,oprf3); mnqno = b3[2]; min\_q\_inBP3[2] = mnqno;

b3 = msort(3,oprf3); mnqno = b3[2]; min\_q\_inBP3[3] = mnqno;

b3 = msort(4,oprf3); mnqno = b3[2]; min\_q\_inBP3[4] = mnqno;

for (int q = 0; q<=4; q++) {

//cout << q <<endl;

for (int bp = 0; bp <=4; bp++) {

MinQ\_no = min\_q\_inBP3[bp];

if (MinQ\_no == 1) { min\_ti = (profile0[bp]+profile1[bp]+profile2[bp]+profile3[bp]); }

if (MinQ\_no == 2) { min\_ti = profile4[bp]; }

if (MinQ\_no == 3) { min\_ti = profile5[bp]; }

if (min\_ti == 0) { min\_ti = 1;}

if (q == 0) {KF[dn\_nodes\_][q][bp] = 0;}

if (q == 1) {KF[dn\_nodes\_][q][bp] = kfmul+(((profile0[bp]+profile1[bp]+profile2[bp]+profile3[bp])/min\_ti)+1); }

if (q == 2 && profile4[bp] > 0) {KF[dn\_nodes\_][q][bp] = kfmul+(((profile4[bp]) / min\_ti)+1);}

if (q == 3 && profile4[bp] > 0) {KF[dn\_nodes\_][q][bp] = kfmul+(((profile5[bp]) / min\_ti)+1);}

}

}

}

return status;

}