ANÁLISIS COMPARATIVO DEL DESEMPEÑO ENTRE UNA RED OPS DISTRIBUIDA Y UNA RED OPS SDON

Daniel Fernando Benavides Quirá
Yeferson Alexander Quinayás Joaqui

Universidad del Cauca

Facultad de Ingeniería Electrónica y Telecomunicaciones
Departamento de Telecomunicaciones
Grupo I+D Nuevas Tecnologías en Telecomunicaciones - GNTT
Popayán, 2019

ANÁLISIS COMPARATIVO DEL DESEMPEÑO ENTRE UNA RED OPS DISTRIBUIDA Y UNA RED OPS SDON

ANEXOS A, B

Daniel Fernando Benavides Quirá
Yeferson Alexander Quinayás Joaqui

Director: MsC. Virginia Solarte Muñoz

Universidad del Cauca

Facultad de Ingeniería Electrónica y Telecomunicaciones
Departamento de Telecomunicaciones
Grupo I+D Nuevas Tecnologías en Telecomunicaciones - GNTT
Popayán, 2019

ANEXO A. ALGORITMOS

A.1 Algoritmo para la generación y envío de paquetes.

Se generan paquetes por intervalos de tiempo, a los cuales se les asigna atributos como su dirección origen, dirección de destino y longitud. La dirección de destino se obtiene aleatoriamente mediante una función uniforme, encargada de escoger un número entre cero y el número de nodos de la red. Además, la prioridad (o tipo de servicio) se obtiene aleatoriamente de igual manera con tres valores posibles (0 ,1 y 2).

```
if (msg == generatePacket)
   {
     // Manejo y envio del paquete
       int destAddress = destAddresses[intuniform(0, destAddresses.size()-1)];
       //int destAddress = 1;

       char pkname[40];
       sprintf(pkname,"to-%d-#%ld", destAddress, pkCounter++);
       EV << "generating packet " << pkname << endl;

       Packet *pkt = new Packet(pkname);
       pkt->setByteLength(packetLengthBytes->longValue());
       pkt->setSrcAddr(myAddress);
       pkt->setDestAddr(destAddress);

       // Se establece la prioridad
       int type = intuniform(0,2);
       pkt->setPriority(type);

       send(pkt,"out");
       scheduleAt(simTime() + sendIATime->doubleValue(), generatePacket);
       if (ev.isGUI()) bubble("Generating packet...");
   }
```

A.2 Algoritmo para la obtención de reglas por dirección de destino y prioridad de los paquetes.

Este algoritmo se encarga de obtener el archivo correspondiente a las reglas y comprara cada línea usando la clase "**Classifier_Rules**", la cual verifica si la dirección y prioridad del paquete son compatibles con las reglas consignadas en el archivo.

```
 if (numOuts != 0){

      //Almacena la información proveniente de Clasifier_Rules
      rules = (Classifier_Rules*)calloc(numOuts,sizeof(Classifier_Rules)); //se usa calloc
para reservar memoria, de forma dinámica
      //lee una línea cada vez y crea la regla asociada para cada cola.
      char *line = (char*)calloc(1500,sizeof(char)); //Debido a esto, el máximo número
de caracteres por línea es 1500.
      int i=0;

      //abre el archivo Rules.dat y lo almacena en rulesFile
      const char *rulesFile = par("rules");

      //Si rules está vacio, se muestra un mensaje de error y para la simulación.
      if(strlen(rulesFile) == 0){
         opp_error("El archivo de reglas no se encuentra definido");
      }
      // abre el archivo Rules.Dat en *ruleFile
      FILE *ruleFile = fopen(rulesFile,"r"); // r = leer el archivo... w=escribir...d=Borrar

      if(ruleFile != NULL){
         //fgets lee hasta que se encuentra un carácter \n
        while(fgets(line,1500,ruleFile) != NULL){
          if(strcmp(line,"\n") != 0 && line[0] != '#'){ //Ignora los comentarios (líneas que
empiezan con #)
             //toma "line" como parámetro de entrada
             //para realizar la comparación de los parámetros
             rules[i] = Classifier_Rules(line);
            i++;
          }
        }
      }
      else{
         opp_error("No es posible abrir el archivo de reglas");
      }

      fclose(ruleFile);
      // i debe ser igual a numQueues. Si no lo es, seguramente hay un error.
      if(!(i == numOuts)){
         printf("(Classifier_Rules) Aviso: El despachador de reglas no coincide con las
colas de los modulos.\n");
```

```
      }
      free(line);
   }
}
```

A.3 Algoritmo para la clasificación de paquetes según las reglas.

Este algoritmo se encarga de verificar que las reglas y características del paquete coincidan, si es así se retorne un valor de tipo booleano con verdadero para que el paquete sea enviado a la primera compuerta donde cumple con los mismos.

```
while(token != NULL){
       if(strcmp(token,"destAddr") == 0){ //Si la regla tiene la dirección de destino?

        //El siguiente token es una dirección de destino, si no lo es, genera un error.
        token = strtok(NULL," \n");
        if(token != NULL){
          destAddr = atoi(token);
          isSet[0] = true;
        }
        else{
         throw runtime_error("No se puede parsear el valor de la direccion de destino en
la regla");
        }
      }
      else if(strcmp(token,"priority") == 0){ //La regla tiene un tipo de servicio?
        token = strtok(NULL," \n");
        if(token != NULL){
           priority = atoi(token);
             isSet[1] = true;
        }
        else{
           throw runtime_error("No es posible parsear el valor de la prioridad en la
regla");
        }
      }
      token = strtok(NULL," ");
   }
}
```

A.4 Algoritmo para desencapsulamiento de los paquetes.

Este algoritmo es el encargado de recibir el inicio del paquete almacenándolo en una lista, una vez recibido el "**endCP**" se busca en la lista el identificador que le corresponde, abstrayendo el paquete de la lista y desencapsulándolo.

```
while(!iter_list.end()){
    item = (ScheduleBurst*)iter_list++;
    if(item->getIdBurst() == bId){
      burst = check_and_cast <Burst *> (item->decapsulate());
      listSize -= burst->getBitLength();
      delete burstList.remove(item);
      delete burst;
      numElems--;
      numElemsVector.record(numElems);
      return;
    }
  }
```

A.5 Algoritmo para programar el envío del CP.

Este algoritmo verifica que el paquete haya llegado desde el modulo "**Packager**", se encarga de encontrar el canal mediante el horizonte más cercano con la función "**findNearestHorizon**".

```
 if(!(msg->isSelfMessage())) // Recibe un paquete desde BurstAssembler (Burstifier)
   {
    //Emite el mensaje del paquete
    Burst *rfg = check_and_cast<Burst*>(msg);

    burstTam.record(rfg->getBitLength());
    burstRecv++;

    int wl = 0;
    int pos=0;

    wl=findNearestHorizon(); //retorna la posición del horizonte más apto para enviar el
paquete actual

    char pkname[40];
    sprintf(pkname,"----Wavelengt--%d",wl);
```

```
EV << "info" << pkname << endl;

//Inserta el id del paquete
rfg->setSendId(getId()); //retorna el identificador del módulo

BurstSenderInfo *myInfo = new BurstSenderInfo();

myInfo->setIdBurstifier(rfg->getIdBurstifier()); //retorna el índice del vector correspondiente al módulo
myInfo->setNumSeq(rfg->getSecNum()); //Contador de paquetes...burstCounter este sec num sirve cuando los paquetes llegan desordenados, con este indicador, se puede ordenar cuando lleguen al receptor
myInfo->setAssignedLambda(wl);

BurstifierInfo *lInfo = (BurstifierInfo*) rfg->removeControlInfo();
myInfo->setLabel(lInfo->getLabel()); //se define en el .ini una etiqueta de destino para cada burstifier
delete lInfo;

//Crea el automensaje "schedule the CP_ini send"
cMessage *ctrMsg= new cMessage("Sched");

//Entra aquí si el paquete puede ser enviado solo cuando el canal wl se establezca como libre. (habrá mucho tiempo para enviar el CP y esperar el tiempo de offset máximo)
if(horizon[wl] - rfg->getMaxOffset() >= simTime()) //el = Significa que se puede programar un paquete al mismo tiempo que el valor del horizonte
 {
  //Se almacena en ScheduledBurst
  pos = scheduleBursts.insertBurst(rfg, horizon[wl]);

  if(pos == -1)
  { //scheduledBurst está lleno. Se decarta este paquete
    delete msg;
    delete ctrMsg;
    burstDroppedByQueue++;
    return;
  }

 //Inserta la posición del paquete en la cola de scheduledBurst
 myInfo->setBurstId(pos);
```

```
    //Se llenan los campos de control (este mensaje viajará a través de todos los estados del sender)
    ctrMsg->setControlInfo(myInfo);
    ctrMsg->setKind(OPS_PROGRAMAR_CP); //Se Establece en el paso 1: Schedule CP

    //Programar envío de CP
    scheduleAt(horizon[wl] - rfg->getMaxOffset(), ctrMsg);

    //Actualizar el valor del horizonte
    horizon[wl] = horizon[wl] + (rfg->getBitLength()/dataRate) + guardTime;

    //Se Registra el valor del horizonte
    horizonVec[wl]->record(horizon[wl]);
    }

   else //Se puede mandar el CP inmediatamente de esta forma el paquete será enviado dentro del offset máximo
    {
     //Se almacena en ScheduledBurst
     pos = scheduleBursts.insertBurst(rfg, simTime() + rfg->getMaxOffset());
     char name[40];
     sprintf(name, "--pos %d", pos);
     EV<< "info"<<name<< endl;

     if(pos == -1)
       { // la cola de ScheduleBursts queue está llena. Descartar
        delete msg;
        delete ctrMsg;
        burstDroppedByQueue++;
        return;
       }

     //Insertar la posición del paquete en la cola scheduledBurst
     myInfo->setBurstId(pos);

     // LLenar los campos del mensaje de control (Este mensaje pasará a traves de todos los estados del sender)
     ctrMsg->setControlInfo(myInfo);
     ctrMsg->setKind(OPS_PROGRAMAR_CP);

     //Programar envío de CP en este momento
```

```
    scheduleAt(simTime(), ctrMsg);

    //Actualizar el valor del horizonte
    horizon[wl] = simTime() + rfg->getMaxOffset() + (rfg->getBitLength()/dataRate) +
guardTime;

    //Registrar el valor del horizonte
    horizonVec[wl]->record(horizon[wl]);
    }
   }
```

A.6 Algoritmo para encontrar el horizonte más cercano.

Este algoritmo realiza un barrido del vector "**horizon**" (el cual contiene los tiempos en los cuales se encontraran libres los canales), y devuelve la posición del menor tiempo que encuentra.

```
int Sender::findNearestHorizon()
{
   int min = 0;
     int i;
     for(i=0;i<numLambdas;i++)
     {
       if(horizon[min] > horizon[i]) //devuelve el menor tiempo que encuentra
          min = i;
     }
     return min;
}
```

A.7 Algoritmo para la generación y envío del paquete.

Este algoritmo se encarga de generar el CP, enviarlo y programar el envío del paquete de datos sobre el canal obtenido mediante el algoritmo explicado en A.6.

```
   // El canal de control se encuentra ocupado transmitiendo otro CP?  ***-->CP
   if(control_is_busy)
    {
    //Si es así, colocar el CP actual en la cola (waitingCP)
    waitingCP.insert(msg);
```

```
    }
  #============================
  // Primer paso: Enviar el ini del mensaje del CP  ***-->CP
  //============================
  else
   {
   //Si no lo es, tomar el canal de control
    control_is_busy = true;

    //Crear el mensaje CP
    BurstControlPacket *CP = new BurstControlPacket("iniCP");

    //Inicializar todos los campos del CP
    CP->setKind(1); //kind 1 = initial CP
    CP->setColorBurst(colour[info->getAssignedLambda()]); //wl

    //Incluye tiempo de llegada del paquete relativo: la diferencia entre la llegada de iniCP e iniBurst
    //Como iniCP será enviado en ste instate, este tiempo es: sendTime(burst) - current simTime
    CP->setDeltaArriveBurst (scheduleBursts.retrieveSendTime(info->getBurstId()) - simTime());

    //Llenar todos los campos del CP
    CP->setIdBurstifier(info->getIdBurstifier()); //índice o identificador del módulo burstifier al que corresponden
    CP->setNumSeq(info->getNumSeq()); //Contador de paquetes
    CP->setIdSend(getId()); //índice o identificador del módulo, en este caso sender
    CP->setLabel(info->getLabel()); //se establece en el .ini y es una etiqueta para cada burstifier
    CP->setBurstSize (scheduleBursts.retrieveBurstSize(info->getBurstId()));
    CP->setByteLength (CPSize);
    CP->setNumSaltos(numSaltos); //se fija el valor de numero de saltos con el contenido de la variable numSaltos
#============================
  // Envio del ini CP
#============================
  //Enviar el CP al canal de control (el último)
    send(CP,"out",numLambdas);

    //Retransmitir el mensaje recibido (ctlMsg = *msg here)
    msg->setKind(OPS_PROGRAMAR_FIN_CP);
```

```
      //Programar el envío de Schedule endCP
      int CPSizeInBits = CPSize*8;
      scheduleAt(simTime()+(CPSizeInBits/dataRate),msg);
      }
     }
#==============================
   // Segundo paso: Enviar el fin del mensaje del CP y programar el envío del paquete.
***-->CP
   #==============================
    else if(msg->getKind() == OPS_PROGRAMAR_FIN_CP)
     {
      info = check_and_cast<BurstSenderInfo *>(msg->getControlInfo());
      BurstControlPacket *CP = new BurstControlPacket("endCP"); //mensaje //creado
del CP

      //llenar el mensaje CP con info
      CP->setKind(2); //kind 2 = end CP
      CP->setIdBurstifier(info->getIdBurstifier());     //índice o identificador del módulo
burstifier al que corresponden
      CP->setNumSeq(info->getNumSeq());             //contador de paquetes
#==============================
      // Envio del end CP
#==============================
      //Envia endCP al canal de control
      send(CP,"out",numLambdas);

      //Retransmitir el mensaje recibido (ctlMsg = *msg here)
      msg->setKind(OPS_PROGRAMAR_RAFAGA);

      //Programar envío de BurstIni
      scheduleAt(scheduleBursts.retrieveSendTime(info->getBurstId()),msg);

      control_is_busy = false; //El canal de control está libre ahora...

      //Escoger un CP si hay alguno disponible
      if(!waitingCP.empty())
        {
         //Colocar el CP fuera de la cola y programarlo
         cMessage *CP_ini = (cMessage*)waitingCP.pop();
         scheduleAt(simTime(),CP_ini);
        }
```

```
     }
    #==============================
    // Tercer paso: Enviar el inicio del paquete óptico  ***-->BURST

   else if(msg->getKind() == OPS_PROGRAMAR_RAFAGA)
    {
     info = check_and_cast<BurstSenderInfo *>(msg->getControlInfo());
     Burst *burst= scheduleBursts.retrieveBurst(info->getBurstId());

     //Llenar los campos del paquete
     burst->setName("iniBurst");
     burst->setKind(1);               //kind 1= send burst

#==============================
   // Envio del ini Burst
#==============================
     send(burst,"out",info->getAssignedLambda());

#==============================
   // Actualiza el contador de envíos
#==============================
     burstSend++;  burstSendCore++;

   //Retransmitir el mensaje recibido (ctlMsg = *msg here)
     msg->setKind(OPS_PROGRAMAR_FIN_RAFAGA);
   //Se programa el final del paquete
     scheduleAt(simTime()+(burst->getBitLength()/dataRate),msg);
   }
#==============================
   // Cuarto y último paso: Enviar el final del paquete    ***-->BURST
   #==============================
   else if(msg->getKind() == OPS_PROGRAMAR_FIN_RAFAGA)
    {
     info = check_and_cast<BurstSenderInfo *>(msg->removeControlInfo());
     Burst *burst = new Burst("endBurst");

     //Se llenan los campos del paquete
     burst->setKind(2);                        //kind 2= end burst
     burst->setIdBurstifier(info->getIdBurstifier()); //índice o identificador del módulo
burstifier al que corresponden
     burst->setSecNum(info->getNumSeq());          //contador de paquetes
```

```
#============================
    // Envio del end Burst
   #============================
    send(burst,"out",info->getAssignedLambda());

    //Borrar paquete de la lista programada
    scheduleBursts.removeBurst(info->getBurstId());

    //Remover la información de control y los mensajes contenedores
    delete msg;
    delete info;
  } }
```

A.8 Algoritmo para la elección de la compuerta de salida del paquete en el OXC.

```
if(msg->getKind() == 1){ //Inicio del paquete.

      take(msg);//toma posesión
      Burst *recvBurst = check_and_cast < Burst*> (msg);
      numSaltos = recvBurst->getNumSaltos() + 1;
      recvBurst->setNumSaltos(numSaltos);
  }
  //Algoritmo sencillo: verifica si la compuerta de entrada; tiene una conexión programada y manda el mensaje a la compuerta de salida asignada.
  cGate *gate = msg->getArrivalGate();

  if(schedulingTable[gate->getIndex()] == -1) delete msg; //Compuerta de salida no asiganada. Se descarta el paquete
  else
    sendDelayed(msg,OXCDelay,"out",schedulingTable[gate->getIndex()]);
```

A.9 Algoritmo para el desensamble del paquete.

```
  if(dynamic_cast< Burst *> (msg) == NULL){ delete msg; return; } //El paquete de control del paquete no debería pasar en este punto

  if(msg->getKind() == 1){ //Inicio del paquete. Coloca este mensaje en la cola

    take(msg); //toma posesión
```

```
    Burst *recvBurst = check_and_cast < Burst*> (msg);
    receivedBursts.push_back(recvBurst);
    //Adiciona el paquete en el contador de paquetes recibidos
    recvBursts++;

    listSize++;
    VlistSize.record(listSize);

  }
  else if(msg->getKind() == 2){ //Final del paquete. Se busca el inicio del mensaje y se desemnsambla.
    Burst *recvBurst = check_and_cast < Burst*> (msg);
    int bId,nSeq; // valores de ID del paquete

    bId = recvBurst->getIdBurstifier();
    nSeq = recvBurst->getSecNum();

    list<Burst*>::iterator i;
    Burst* actElem;
    //Busca desde el comienzo porque el paquete que se está buscando es probablemente una de la últimas
    for(i = receivedBursts.begin(); i != receivedBursts.end(); i++){
      actElem = *i;
      if((actElem->getIdBurstifier() == bId) && (actElem->getSecNum() == nSeq)) break; //Burst found!
    }
    if(i != receivedBursts.end()){ //Si el iterador superior no llega hasta el final...significa que el paquete fue encontrado.
      Burst *burstIni= check_and_cast< Burst* > (*i);
    cMessage *tempPack;
      while(burstIni->hasMessages()){ //Libera los paquetes hasta que la cola de burstIni queue esté vacia
        tempPack = burstIni->retrieveMessage();
        //TODO: Envía el paquete a un buffer intermedio por lo tanto, todos los paquetes no serían liberados a la red eléctrica al mismo tiempo.
        send(tempPack,"out");
      }
      //Limpieza
      delete msg;
      delete burstIni;
      i = receivedBursts.erase(i);
```

```
        listSize--;
        VlistSize.record(listSize);
      }
      else{
        printf("<OPS_BurstDissasembler><t=%s> Error!! burst with id=(%d,%d) not found!\n",simTime().str().c_str(),bId,nSeq);
        delete msg;
      }
    }
```

A.10 Algoritmo del Nodo de Control Centralizado SDON

Este algoritmo muestra el comportamiento del nodo centralizado cuando se solicitan recursos y enrutamiento.

```
#==============================
     // Recepción del CP que solicita los recursos.
#==============================

NodeFcentral *item = new NodeFcentral();
   if(inType==1){

       if(inCprocess==0){

          if(miDireccion==inCenternode){

             if(miDireccion==inSour){return item->dup();}

             else if(miDireccion!=inSour){
             item->setNcprocess(1);
             item->setNcdest(inSour);
             return item->dup();
             }
          }
          else if(miDireccion!=inCenternode){
                    item->setNcdest(inCenternode);
                    return item->dup();
          }

       else if (inCprocess==1){
```

```
        if(miDireccion==inSour){
            //llega al destino
            return item->dup();
        }
        else if (miDireccion!=inSour){
            item->setNcdest(inSour);
            return item->dup();
        }
    }
  #==============================
      // Se retorna el CP hacia el nodo de origen
  #==============================
```

A.11 Algoritmo para envío de Paquetes en SDON

Éste algoritmo realiza el envío de paquetes cuando el nodo centralizado ha realizado la asignación de recursos, es decir cuando llega el paquete de control al planificador de envío de origen.

```
#==============================
    // Se recibe el CP y se busca en la cola de espera el paquete al que corresponda.
 #==============================

list<Burst*>::iterator iter_list;
Burst *item;
for(iter_list = cpendingMsgs.begin();iter_list != cpendingMsgs.end(); iter_list++){
item = *iter_list;
if(item->getSecCre() == tNumSeq){//IF CRG
        Burst *rfg = item->dup();
        rfg->setKind(PRUEBA);
        rfg->setDestrfg(tendLabel);
        cpendingMsgs.remove(item);
        delete item;
        scheduleAt(simTime(), rfg);
        return;

}
```

ANEXO B. RESULTADOS DE SIMULACIÓN

B.1 Resultados de la probabilidad de bloqueo al variar el tráfico en las redes en un tiempo de simulación de 0.075 s.

Tabla B.1 Probabilidad de bloqueo vs Tráfico.

	OPS Distribuida				OPS SDON			
	8 slots (100 GHz)		16 slots (200 GHz)		8 slots (100 GHz)		16 slots (200 GHz)	
TRÁFICO(E)	1.25 Gbps	2.5Gbps	1.25 Gbps	2.5Gbps	1.25 Gbps	2.5Gbps	1.25 Gbps	2.5Gbps
1	0,1544	0,0273	0,000216	0,0000058	0,0737	0,003990	0,000042	0
0,9	0,1344	0,0221	0,000113	0,0000041	0,0559	0,003376	0,000036	0
0,8	0,0886	0,0115	0,000046	0,0000023	0,0336	0,001303	0,000016	0
0,7	0,0758	0,0093	0,000022	0	0,0283	0,001185	0	0
0,6	0,0496	0,0050	0,000016	0	0,0188	0,000600	0	0
0,5	0,0426	0,0042	0	0	0,0143	0,000462	0	0
0,4	0,0284	0,0023	0	0	0,0092	0,000210	0	0
0,3	0,0247	0,0021	0	0	0,0074	0,000188	0	0
0,2	0,0163	0,0011	0	0	0,0048	0,000111	0	0
0,1	0,0145	0,0011	0	0	0,0042	0,000100	0	0

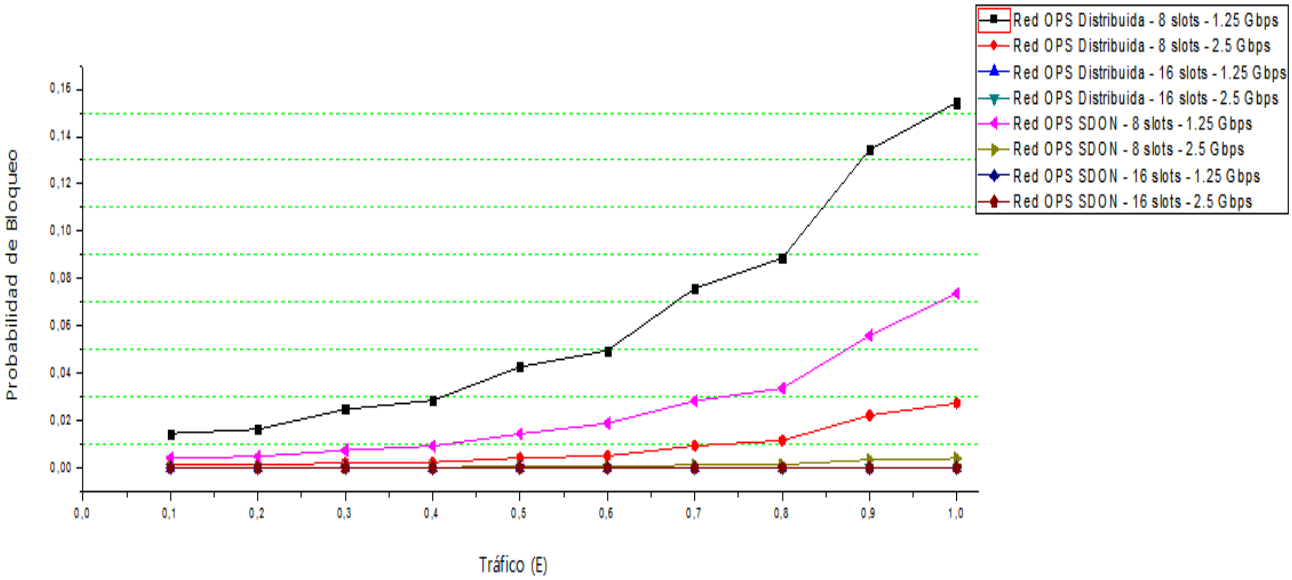

Figura B.1 Probabilidad de bloqueo vs Tráfico, 8 configuraciones [1].

B.2 Resultados Del retardo extremo a extremo al variar el tráfico en las redes en un tiempo de simulación de 0.075 s.

Tabla B.2 Retardo Extremo a Extremo vs Tráfico, OPS Distribuida.

	OPS Distribuida			
	8 slots (100 GHz)		16 slots (200 GHz)	
TRÁFICO(E)	1.25 Gbps	2.5Gbps	1.25 Gbps	2.5Gbps
1	0,00907630	0,009069015	0,009068842	0,00906866688
0,9	0,00907430	0,009068973	0,009068809	0,00906866586
0,8	0,00907171	0,009068886	0,009068746	0,00906866422
0,7	0,00907128	0,009068866	0,009068733	0,00906866399
0,6	0,00907047	0,009068815	0,009068706	0,00906866334
0,5	0,00907027	0,009068803	0,009068699	0,00906866321
0,4	0,00906990	0,009068775	0,009068686	0,00906866287
0,3	0,00906979	0,009068768	0,009068683	0,00906866284
0,2	0,00906955	0,009068747	0,009068677	0,00906866269
0,1	0,00906949	0,009068742	0,009068675	0,00906866268

Tabla B.3 Retardo Extremo a Extremo vs Tráfico, OPS SDON.

	OPS SDON			
	8 slots (100 GHz)		16 slots (200 GHz)	
TRÁFICO(E)	1.25 Gbps	2.5Gbps	1.25 Gbps	2.5Gbps
1	0,00898200	0,008970146	0,008970188	0,00896979604
0,9	0,00897566	0,008970100	0,008970152	0,00896979480
0,8	0,00897303	0,008970014	0,008970085	0,00896979195
0,7	0,00897262	0,008969994	0,008970069	0,00896979159
0,6	0,00897185	0,008969943	0,008970039	0,00896979073
0,5	0,00897160	0,008969931	0,008970032	0,00896979052
0,4	0,00897123	0,008969902	0,008970016	0,00896978961
0,3	0,00897110	0,008969895	0,008970013	0,00896978957
0,2	0,00897088	0,008969874	0,008970006	0,00896978953
0,1	0,00897082	0,008969869	0,008970003	0,00896978947

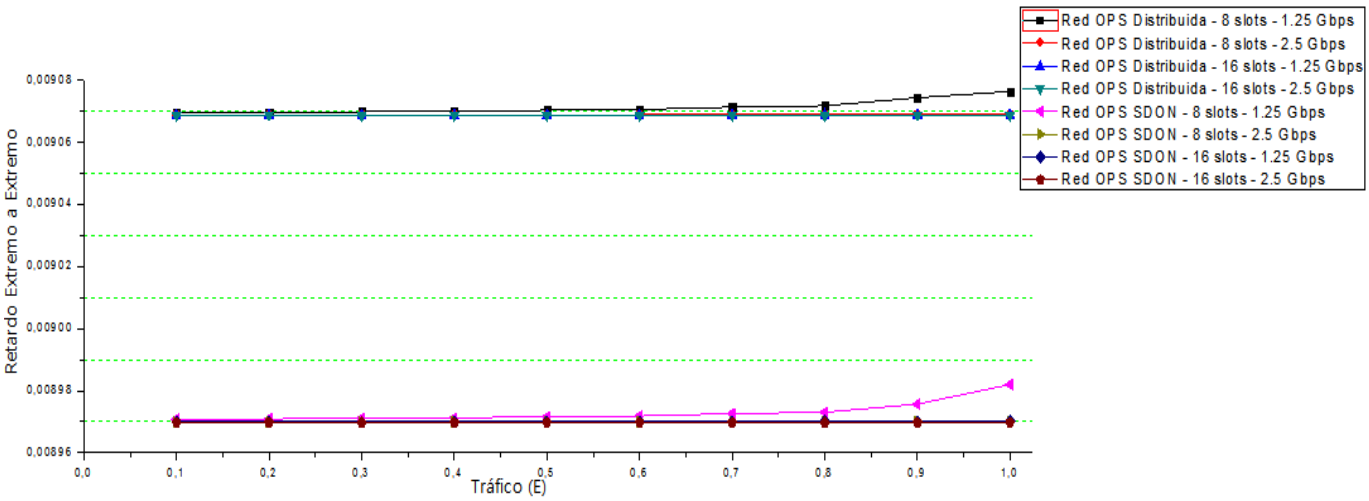

Figura B.2 Retardo Extremo a Extremo vs Tráfico, 8 configuraciones [1].

B.3 Paquetes Generados, Recibidos y Perdidos en las redes en los distintos 5 tipos de carga en un tiempo de simulación de 0.075 s.

Tabla B.4 Paquetes Generados en las dos redes.

RED	# Slots	Velocidad	PAQUETES GENERADOS POR CARGA				
			Muy Alta	Alta	Media	Baja	Muy Baja
OPS Distribuida	8	1.25 Gbps	505699	423355	363869	319119	284392
		2.5 Gbps	505788	423394	363889	319133	284403
	16	1.25 Gbps	505797	423398	363891	319135	284404
		2.5 Gbps	505798	423399	363891	319135	284404
OPS SDON	8	1.25 Gbps	495658	423428	356832	311035	284389
		2.5 Gbps	495222	423387	363886	319130	284399
	16	1.25 Gbps	505791	423388	363887	319130	284399
		2.5 Gbps	505792	423390	363887	319130	284400

Tabla B.5 Paquetes Recibidos en las dos redes.

RED	# Slots	Velocidad	PAQUETES RECIBIDOS POR CARGA				
			Muy Alta	Alta	Media	Baja	Muy Baja
OPS Distribuida	8	1.25 Gbps	432541	388487	347072	310623	280012
		2.5 Gbps	493247	418980	362218	318425	284090
	16	1.25 Gbps	505713	423383	363888	319135	284404
		2.5 Gbps	505795	423398	363891	319135	284404
OPS SDON	8	1.25 Gbps	463683	410296	350953	308409	283110
		2.5 Gbps	493460	422860	363692	319066	284369
	16	1.25 Gbps	505771	423385	363887	319130	284399
		2.5 Gbps	505792	423390	363887	319130	284400

Tabla B.6 Paquetes Perdidos en las dos redes.

RED	# Slots	Velocidad	PAQUETES PERDIDOS POR CARGA				
			Muy Alta	Alta	Media	Baja	Muy Baja
OPS Distribuida	8	1.25 Gbps	73158	34868	16797	8496	4380
		2.5 Gbps	12542	4414	1671	709	314
	16	1.25 Gbps	84	15	3	0	0
		2.5 Gbps	3	1	0	0	0
OPS SDON	8	1.25 Gbps	31975	13132	5879	2626	1280
		2.5 Gbps	1762	527	194	64	30
	16	1.25 Gbps	20	4	0	0	0
		2.5 Gbps	0	0	0	0	0

REFERENCIAS

[1] Y. Quinayas y D. Benavides, "*Análisis comparativo del desempeño entre una red OPS DISTRIBUIDA Y UNA RED OPS SDON*", Tesis de pregrado, Universidad del Cauca, Popayán, Colombia, 2019.