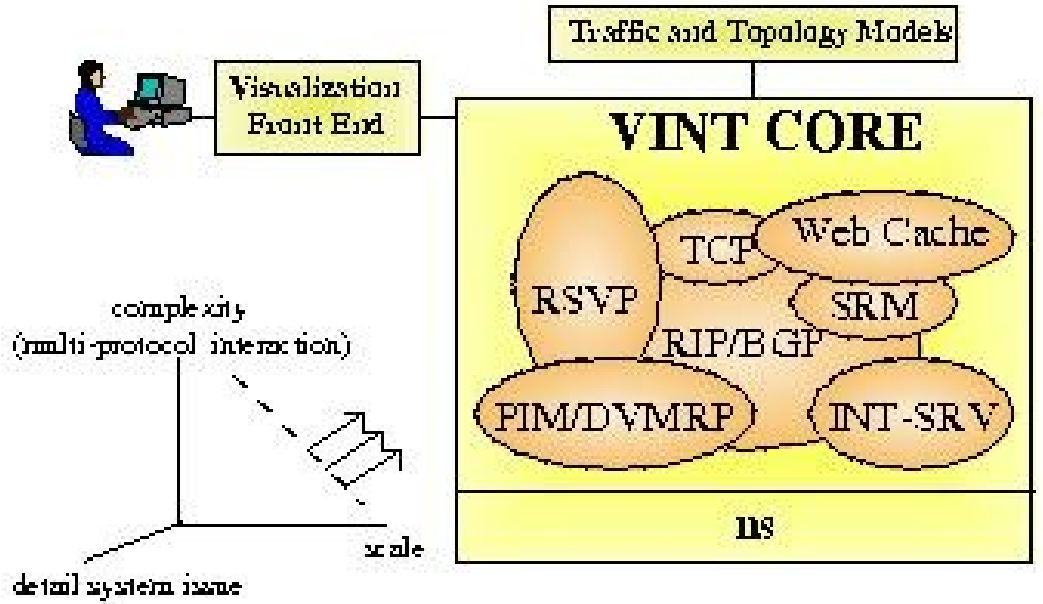
Network Simulator - ns2

Rodolfo W. L. Coutinho
rwlc@dcc.ufmg.br

Histórico

- Iniciado em 1989 como uma variante do simulador de redes REAL e atualmente está na versão 2.35 (4 de Nov. 2011).

- Mantido pelo projeto VINT. (USC/ISI, Xerox PARC, LBNL, and UCB)



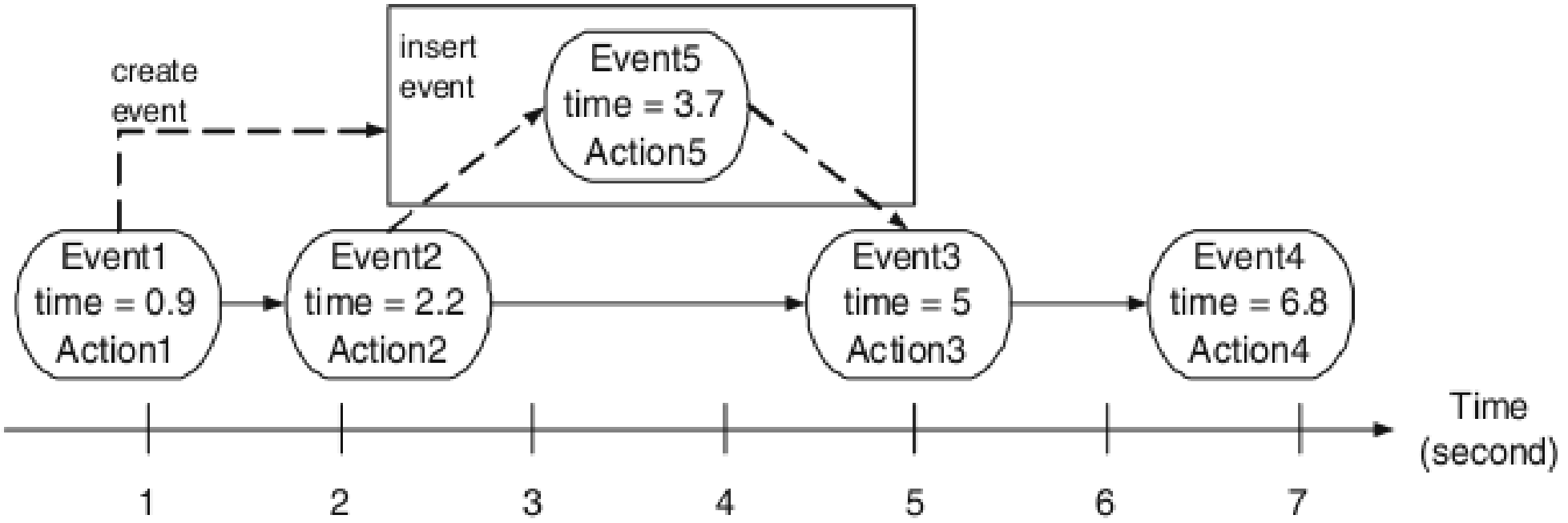
Histórico

- Livremente distribuído e open source.
  http://www.isi.edu/nsnam/ns/ns-build.html

- Nível de pacotes.

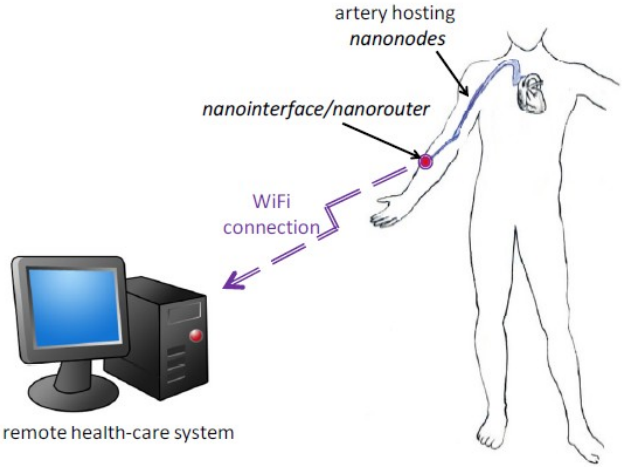
Histórico

- ns é um simulador de eventos discretos (event-driven)



Simulação de redes e protocolos

- Algumas redes que podem ser simuladas no ns2 …



Simulação de redes e protocolos

- Alguns protocolos ...
  - TCP (reno, tahoe, vegas, sack)
  - MAC (802.11, 802.3, TDMA)
  - Roteamento (DSDV, DSR, AODV, TORA)
  - RSSF (diffusion, gaf)
  - Aplicação: web, ftp, telnet, cbr

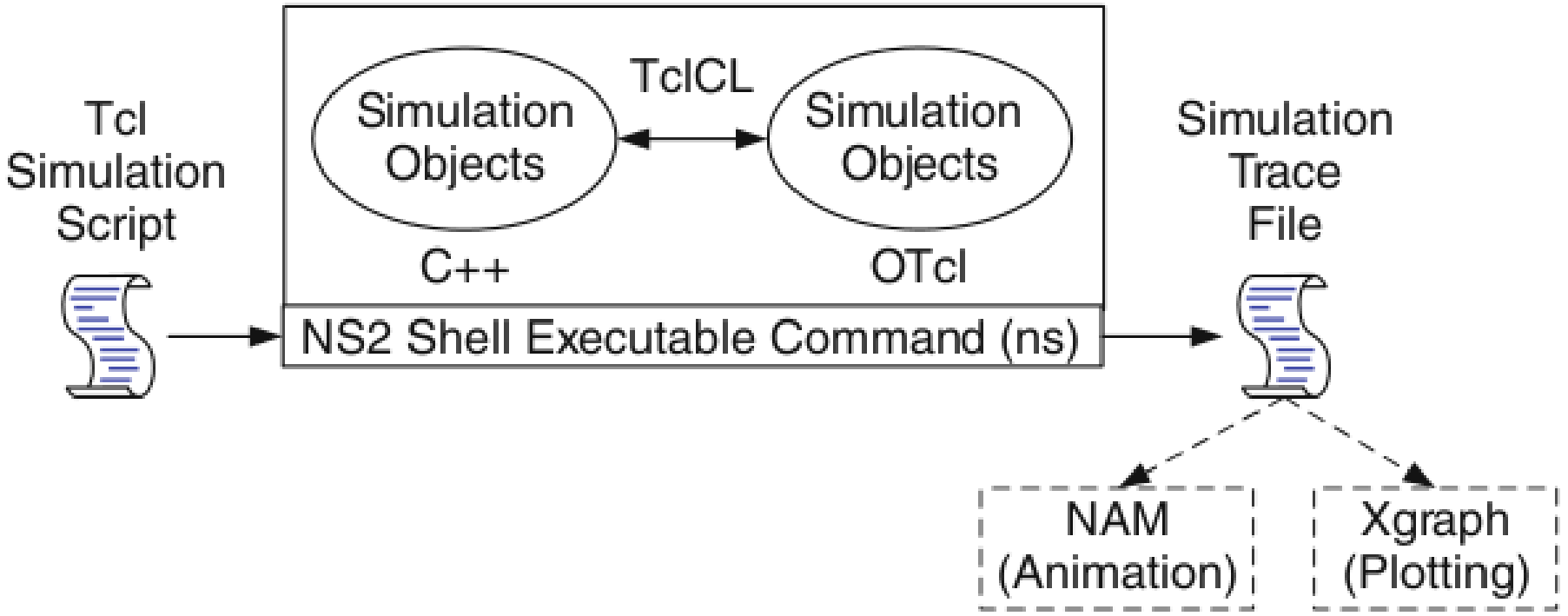
Componentes

- ns - Simulador
- NAM – Network AniMator
  - Usado para visualizar a saída do ns.
- Pré-processamento
  - Geradores de tráfego e topologias
- Pós-processamento
  - Analisadores de traces

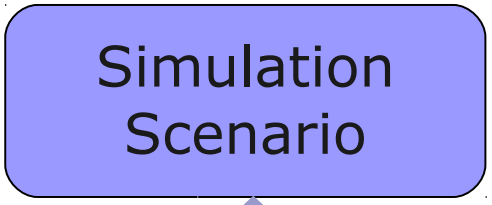
Plataformas suportadas

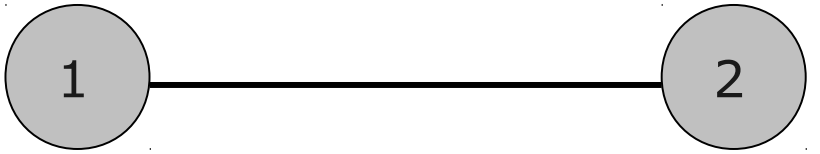
- Unix e sistemas baseados no unix
  - FreeBSD
  - Linux
  - Solaris
- Windows
  - Necessário Cygwin

Arquitetura básica

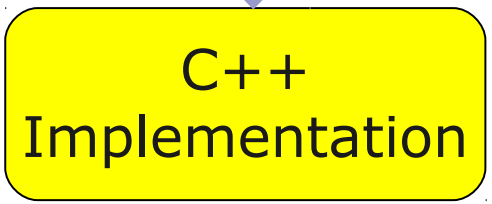


Arquitetura básica






```
set ns_ [new Simulator]
set node_(0) [$ns_ node]
set node_(1) [$ns_ node]
```




```
class MobileNode : public Node
{
        friend class PositionHandler;
public:
         MobileNode();
                 •
                 •
}
```

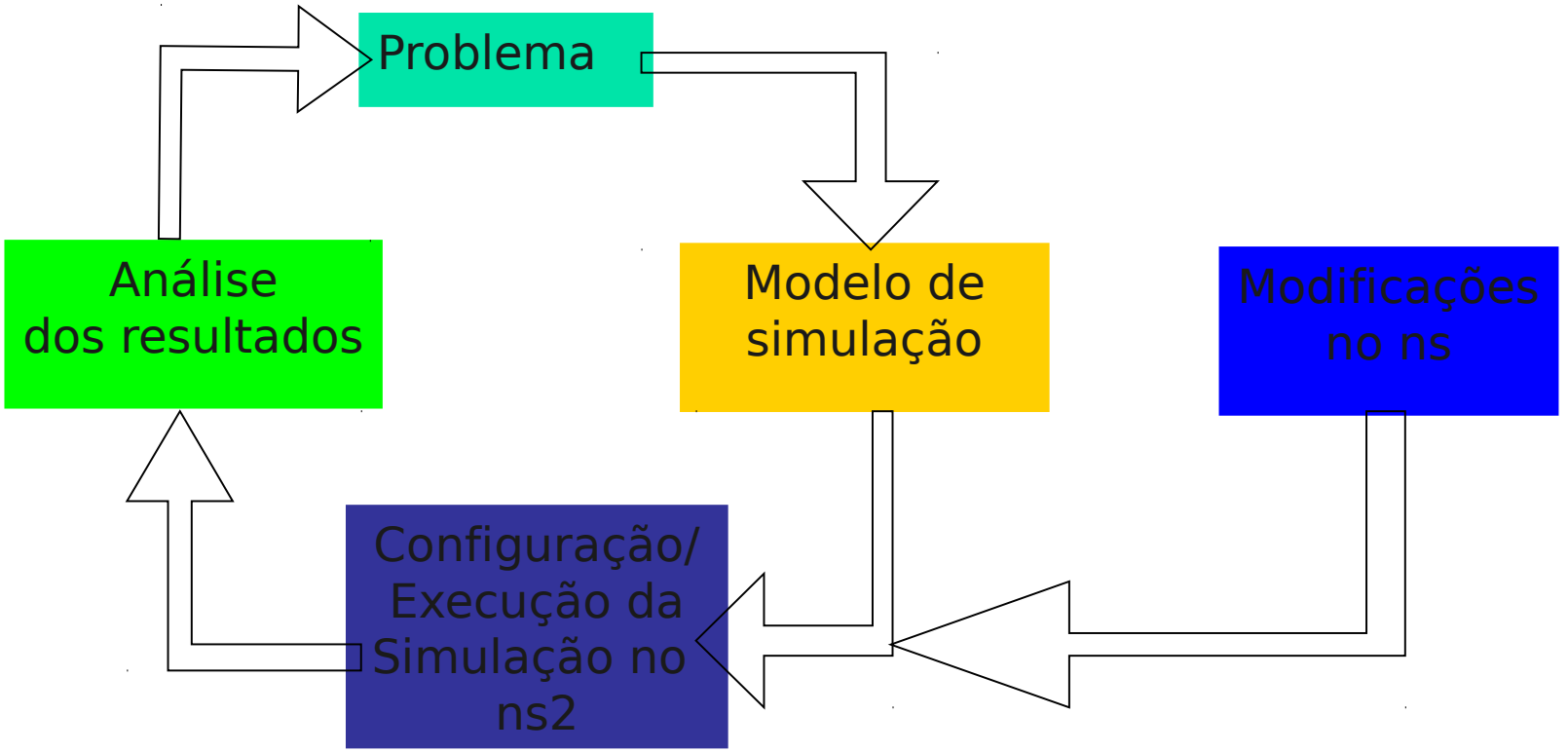
Arquitetura básica

- Por que duas linguagens?
  - C++
    - Velocidade e eficiência
  - Otcl
    - Front-end para configurar simulação
    - Tempo de iteração é mais importante (mudança do modelo e re-execução.)

Arquitetura básica

- Por que duas linguagens?
  - C++
    - Para lidar com pacotes
    - Manipulação de bytes
    - Modificar módulos existentes
  - Otcl
    - Criar e configurar a rede
    - Executar simulações com os módulos existentes

Usando o ns2


Problema
Análise
dos resultados
Modelo de
simulação
Modificações
no ns
Configuração/
Execução da
Simulação no
ns2

Flooding na rede

- Criando um protocolo de roteamento
- Flooding na rede
  - Nó sink inicia o flooding
  - Nós sensores disseminam a informação para seus vizinhos.
    - Sempre que receber uma mensagem
    - Ainda não enviou a mensagem recebida

Flooding na rede

- Criando um novo protocolo
  - Definir as mensagens e formato dos pacotes.
  - Fazer o bind do cabeçalho do pacote para o TCL.
  - Criação de timers, se necessário.
  - Criação do protocolo.
  - Bind do Agent.
  - Função command.

Módulo para redes aquáticas

- AquaSim
  - http://obinet.engr.uconn.edu/wiki/index.php/Aqua-Sim
  - ns-2.30
  - Canal acústico
  - Protocolos MAC e de roteamentos

Flooding na rede

- Instalação do Aquasim
  - ssh 150.164.7.30 -l user
  - Senha: user
  - Copiar o arquivo Aqua-Sim-1.0.tar
  - autoconf, automake, build-essential, tk-8.4, tk-8.4-dev, tcl-8.4, tcl-8.4-dev
  - Descompactar o aquivo (tar -xf)
  - ./install

Uma máquina virtual com a instalação está disponível no mesmo endereço !

Configurar variáveis de ambiente

- PATH=$PATH:~/Aqua-Sim-1.0/bin:~/Aqua-Sim-1.0/tcl8.4.13/unix:~/Aqua-Sim-1.0/tk8.4.13/unix

- LD_LIBRARY_PATH=~/Aqua-Sim-1.0/otcl-1.12:~/Aqua-Sim-1.0/lib

- TCL_LIBRARY=~/Aqua-Sim-1.0/tcl8.4.13/library

- export PATH
- export LD_LIBRARY_PATH
- export TCL_LIBRARY

Flooding na rede

- Mudanças necessárias:
  - Declaração do tipo do pacote
    - common/packet.h
  - TCL library
    - tcl/lib/ns-packet.tcl
    - tcl/lib/ns-default.tcl
    - tcl/lib/ns-lib.tcl
  - Makefile
    - OBJ_CC = \ … dir/prot.o\ …
  - touch common/packet.cc
  - make

Flooding na rede

- Criar diretório do protocolo na pasta ns-2.30
- Criar arquivos myprot.cc e myprot.h

myprot.h

```
#ifndef __myprot_h__
#define __myprot_h__

#include <limits>
#include <packet.h>
#include <agent.h>
#include <scheduler.h>
#include <classifier-port.h>
#include <address.h>
#include <trace.h>
#include <mobilenode.h>
#define HDR_MYPROT_PKT(p)     hdr_myprot_pkt::access(p)

struct hdr_myprot_pkt {
   int seq_num;
   int hops;

   static int offset_;
   inline static int& offset() { return offset_;}
   inline static hdr_myprot_pkt* access (const Packet* p) {
      return (hdr_myprot_pkt*) p->access(offset_);
   }
};

class MyProtAgent : public Agent {

   protected:
      int num_hops_;
      int next_hop_;
      int id_;
      PortClassifier* dmux_;
      Trace *logtarget;
        MobileNode *mn_;

   public:
      MyProtAgent();
      int command(int, const char*const*);
      void recv(Packet*, Handler*);
      void sendMsg();
      void sendBcast();
};

static class MyProtClass : public TclClass {
   public:
      MyProtClass () : TclClass("Agent/MYPROT") {}
      TclObject* create (int argc, const char*const* argv) {
        // assert(argc == 5);
         return (new MyProtAgent());
      }
}class_myprot;
#endif
```

Packets are used to exchange information between objects in the simulation

Agents represent endpoints where network-layer packets are constructed or consumed, and are used in the implementation of protocols at various layers.

```
#include "myprot.h"

int hdr_myprot_pkt::offset_;

MyProtAgent::MyProtAgent() : Agent(PT_MYPROT) {
   id_ = 0;
   num_hops_=numeric_limits<int>::max();
}

int MyProtAgent::command(int argc, const char*const* argv){
      if (argc == 2) {
                if (strcmp(argv[1],"send-bcast") == 0) {
                           sendBcast();
                           return TCL_OK;
                        } else if (strcmp(argv[1], "start") == 0) {
                  return TCL_OK;
               }else if (strcmp(argv[1], "print-dist") == 0) {
                  printf("Node [%d] - distance to sink [%d] - next hop[%d]\n", id_, num_hops_, next_hop_);
                    return TCL_OK;
               }
      } else if (argc == 3) {
         if (strcmp(argv[1], "port-dmux") == 0) {
            dmux_ = (PortClassifier*)TclObject::lookup(argv[2]);
            if (dmux_ == 0) {
                              fprintf (stderr, "%s: %s lookup of %s failed\n", __FILE__,argv[1], argv[2]);
                              return TCL_ERROR;
            }
            return TCL_OK;
         } else if(strcmp(argv[1], "log-target") == 0 || strcmp(argv[1], "tracetarget") == 0) {
            logtarget = (Trace*) TclObject::lookup(argv[2]);
            if(logtarget == 0)
                              return TCL_ERROR;
            return TCL_OK;

          } else if (strcmp(argv[1], "set-node") == 0 ) {
                TclObject *obj;
                if ((obj = TclObject::lookup(argv[2])) == 0) {
                        fprintf(stderr, "lookup falhou\n");
                              return TCL_ERROR;
                }

                mn_ = (MobileNode *)obj;
                id_ = mn_->address();
return TCL_OK;
        }
      }
      return Agent::command(argc, argv);
}
```

The command() function is invoked from Tcl

```
void MyProtAgent::sendBcast() {
   num_hops_ = 0;
   sendMsg();
}



void MyProtAgent::recv(Packet* p, Handler*) {
   struct hdr_cmn* ch = HDR_CMN(p);
   struct hdr_ip* ih = HDR_IP(p);
   struct hdr_myprot_pkt* pkt = HDR_MYPROT_PKT(p);

   if (pkt->hops+1 < num_hops_) {
      num_hops_ = pkt->hops +1;
      next_hop_ = ih->saddr();
      sendMsg();
   }

   Packet::free(p);

}

void MyProtAgent::sendMsg() {
  Packet *p;
  p = allocpkt();
  hdr_cmn *cmh = HDR_CMN(p);
  hdr_ip *iph = HDR_IP(p);
  hdr_myprot_pkt *myproth = HDR_MYPROT_PKT(p);

  iph->dport() = iph->sport();
  iph->daddr() = IP_BROADCAST;

  iph->saddr() = id_;

  myproth->hops = num_hops_;
  Scheduler::instance().schedule(target_, p, 0.0);
}
```

Flooding na rede

- common/packet.h

```
75 enum packet_t {
76         PT_TCP,
77         PT_UDP,
78         ...
79         // Bell Labs Traffic Trace Type (PackMime OL)
80         PT_BLTRACE,
81
82         PT_MYPROT,
83
84         // insert new packet types here
85         PT_NTYPE // This MUST be the LAST one
86 };
87
```

Flooding na rede

- common/packet.h

```
79 class p_info {
80 public:
81         p_info() {
82                 name_[PT_TCP]= "tcp";
83                 name_[PT_UDP]= "udp";
84                 ...
85
86                 // Bell Labs (PackMime OL)
87                 name_[PT_BLTRACE]="BellLabsTrace";
88
89                 name_[PT_MYPROT]="MyProt";
90
91                 name_[PT_NTYPE]= "undefined";
92         }
```

Flooding na rede

- tcl/lib/ns-packet.tcl

```
112
113 foreach prot {
114         MyProt
115 # Common:
116         Common
117         Flags
118         IP      # IP
119 # Routing Protocols:
120         NV      # NixVector classifier for stateless routing
121         rtProtoDV       # distance vector routing protocol
122         rtProtoLS       # link state routing protocol
123         SR      # source routing, dsr/hdr_sr.cc
124         Src_rt  # source routing, src_rtg/hdr_src.cc
125 # Routers:
126         LDP     # mpls/ldp.cc
127         MPLS    # MPLS, MultiProtocol Label Switching
128         Resv    # Token buckets, for reservations.
```

Flooding na rede

- tcl/lib/ns-lib.tcl

```
589 # XXX This should be moved into the node initialization procedure instead
590 # of standing here in ns-lib.tcl.
591 Simulator instproc create-wireless-node args {
592         $self instvar routingAgent_ wiredRouting_ propInstance_ llType_ \
593             macType_ ifqType_ ifqlen_ phyType_ chan antType_ \
594             energyModel_ initialEnergy_ txPower_ rxPower_ \
595             idlePower_ sleepPower_ transitionPower_ transitionTime_ \
596             topoInstance_ level1_ level2_ inerrProc_ outerrProc_ FECProc_
597
598         Simulator set IMEPFlag_ OFF
599
600         # create node instance
601         set node [eval $self create-node-instance $args]
602
603         # basestation address setting
604         if { [info exist wiredRouting_] && $wiredRouting_ == "ON" } {
605                 $node base-station [AddrParams addr2id [$node node-addr]]
606         }
607         switch -exact $routingAgent_ {
608             MyProt {
609                         set ragent [$self create-myprot-agent $node]
610             }
611             DSDV {
```

Flooding na rede

- tcl/lib/ns-lib.tcl

```
751
752 Simulator instproc create-myprot-agent { node } {
753         set ragent [new Agent/MyProtAgent [$node id]]
754         $self at 0.0 "$ragent start"
755         $node set ragent_ $ragent
756         return $ragent
757 }
758
```

Flooding na rede

- Makefile

```
156 OBJ_CC = \
157    tools/random.o tools/rng.o tools/ranvar.o common/misc.o common/timer-handler.o \
158    common/scheduler.o common/object.o common/packet.o \
159    common/ip.o routing/route.o common/connector.o common/ttl.o \
160    trace/trace.o trace/trace-ip.o \
161    classifier/classifier.o classifier/classifier-addr.o \
162    classifier/classifier-hash.o \
163    classifier/classifier-virtual.o \
164    classifier/classifier-mcast.o \
165    classifier/classifier-bst.o \
166    classifier/classifier-mpath.o mcast/replicator.o \
167    classifier/classifier-mac.o \
168    classifier/classifier-qs.o \
169    classifier/classifier-port.o src_rtg/classifier-sr.o \
170         src_rtg/sragent.o src_rtg/hdr_src.o adc/ump.o \
171    qs/qsagent.o qs/hdr_qs.o \
172    myprot/myprot.o \
173    apps/app.o apps/telnet.o tcp/tcplib-telnet.o \
174    tools/trafgen.o trace/traffictrace.o tools/pareto.o \
175    tools/expoo.o tools/cbr_traffic.o \
176    adc/tbf.o adc/resv.o adc/sa.o tcp/saack.o \
```

Flooding na rede

- touch common/packet.cc
- make

Referências

- T. Issariyakul, E. Hossain. “Introduction to network simulator NS2”. Springer, 2009
- “The NS manual”, 2011. Disponível em www.isi.edu/nsnam/ns/doc/ns_doc.pdf
- “Ns by example”. Disponível em http://nile.wpi.edu/NS/
- Marc Greis. “Tutorial for the Network Simulator ns”. Disponível em http://www.isi.edu/nsnam/ns/tutorial/
- F. Ros, P. Ruiz. “Implementing a New Manet Unicast Routing Protocol in NS2”. 2004
- L. C. Gonçalves, M. E. O. Corrêa. “Tutorial de ns2”. Disponível em http://www.midiacom.uff.br/~debora/redes1/pdf/tutorial-ns2.pdf
- http://www.nsnam.com/