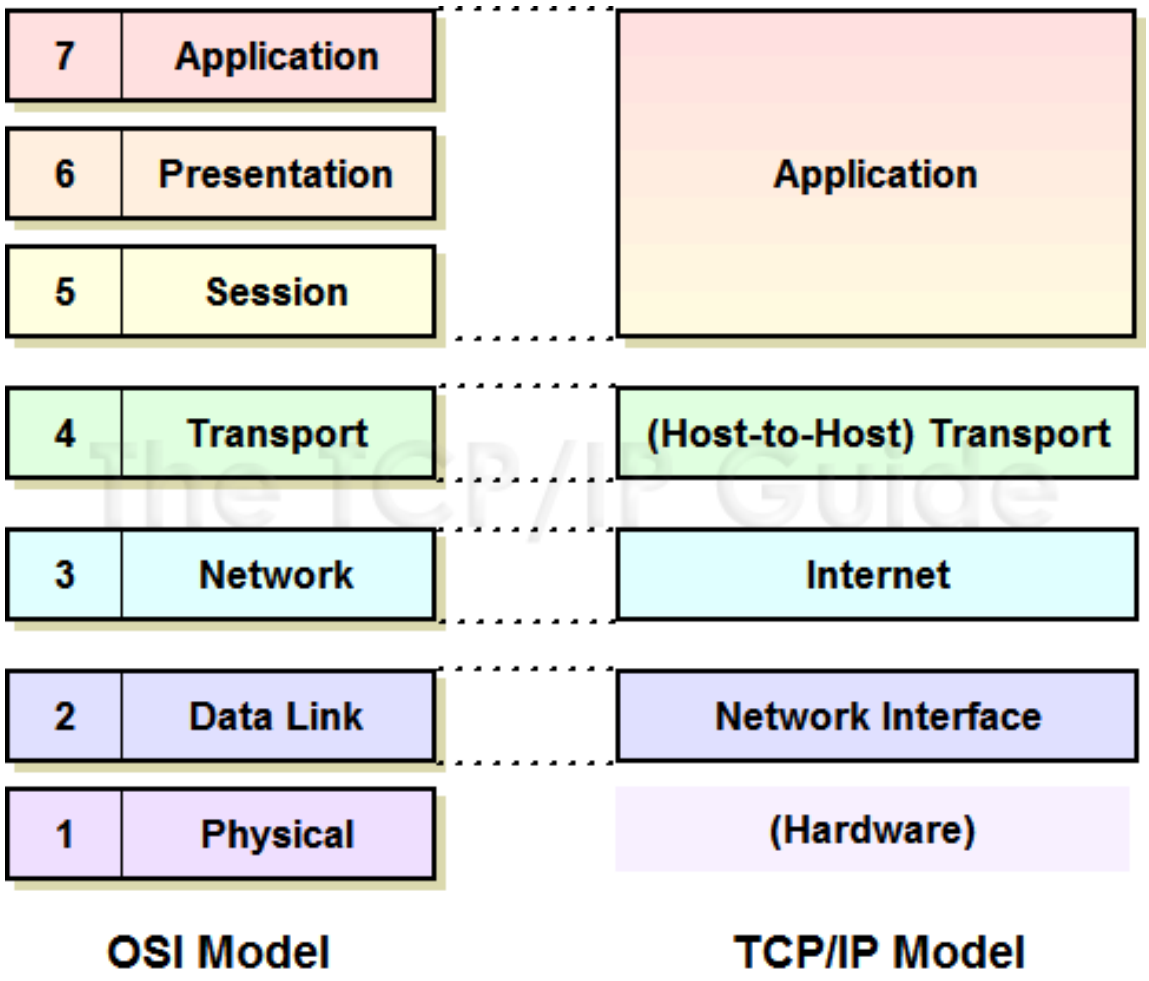
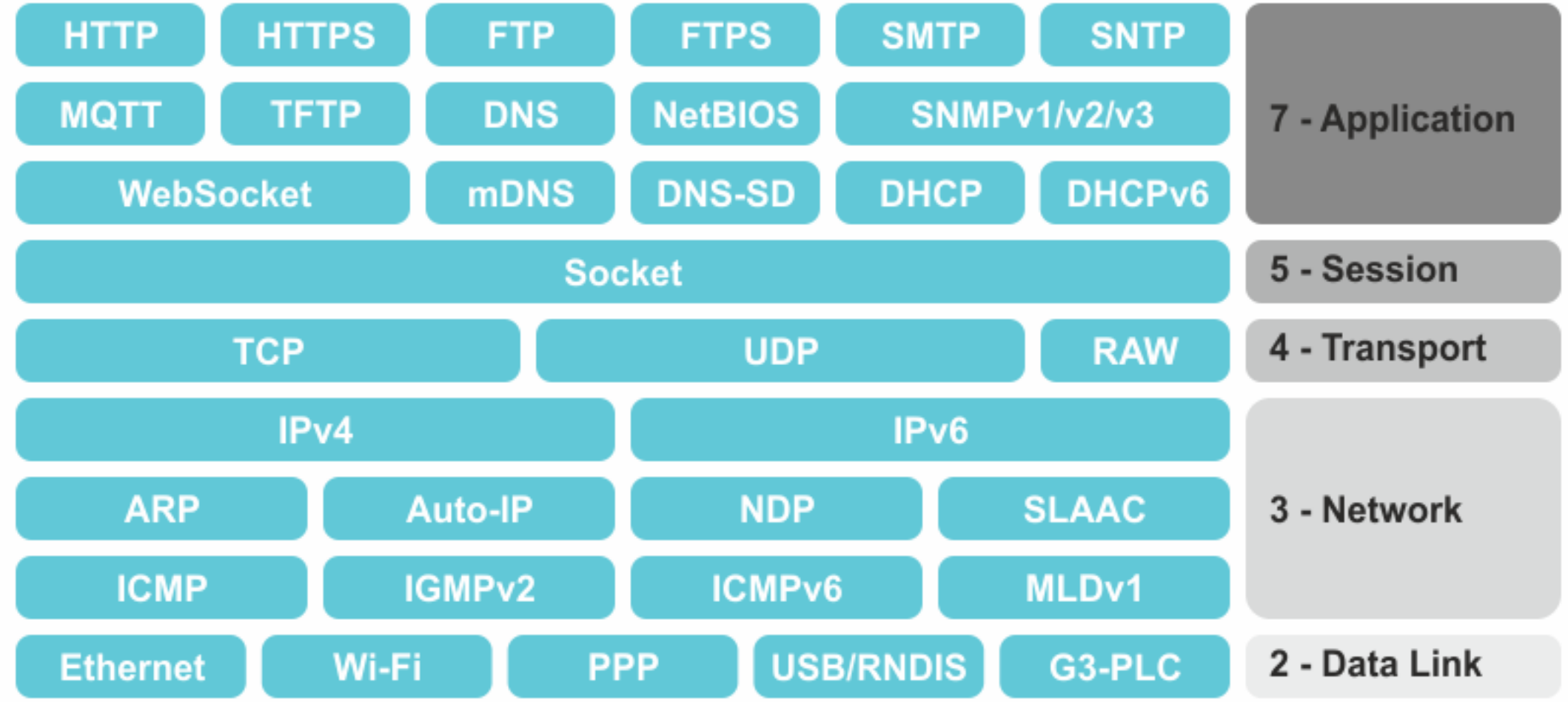
# Um guia sobre o NS3

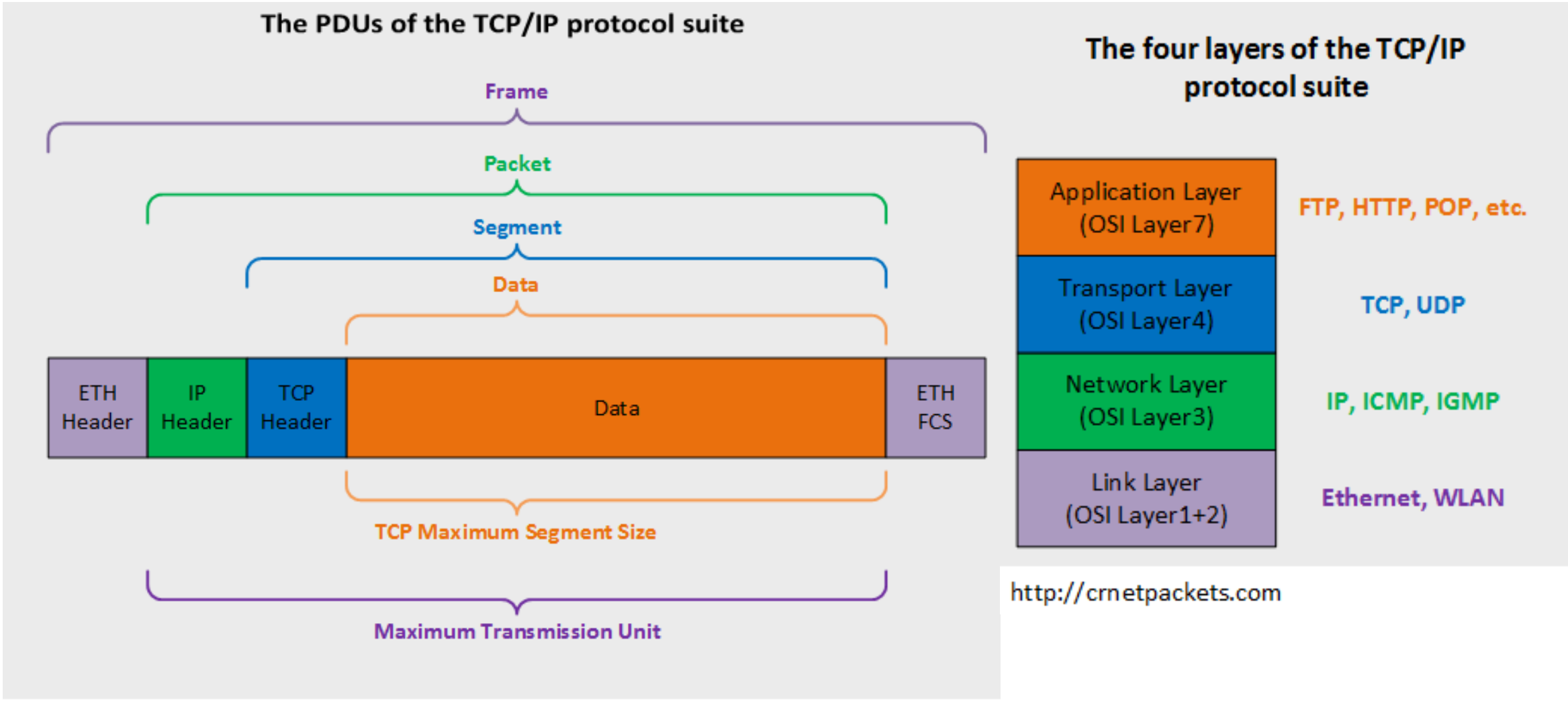
# Revisão: Pilha de rede



# Revisão: Protocolos e aplicações típicas das camadas



# Revisão: Arquitetura de rede



# NS3

- Lançado em 2008
- Simulador de eventos discretos
- Open-source
- Suporte a scripts em C++ e Python
- Foco em redes sem fio
- https://www.nsnam.org/

# NS3: Documentação

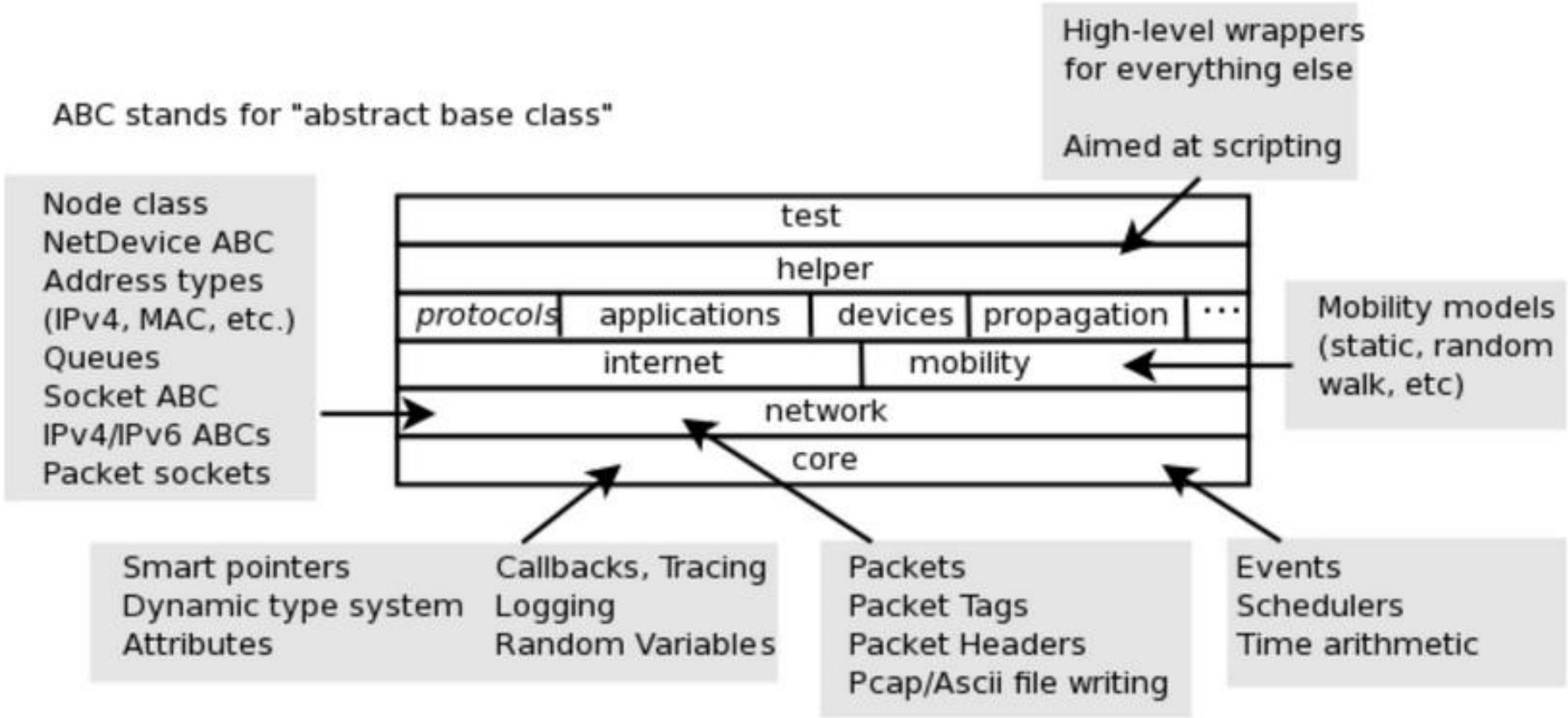
- Tutoriais
- Manuais
- Biblioteca com descrição dos modelos
- Doxygen com documentação do código
- Disponível em: https://www.nsnam.org/documentation/

- Durante a programação, provavelmente vão precisar procurar algo nos códigos fonte, mas veremos isso adiante...

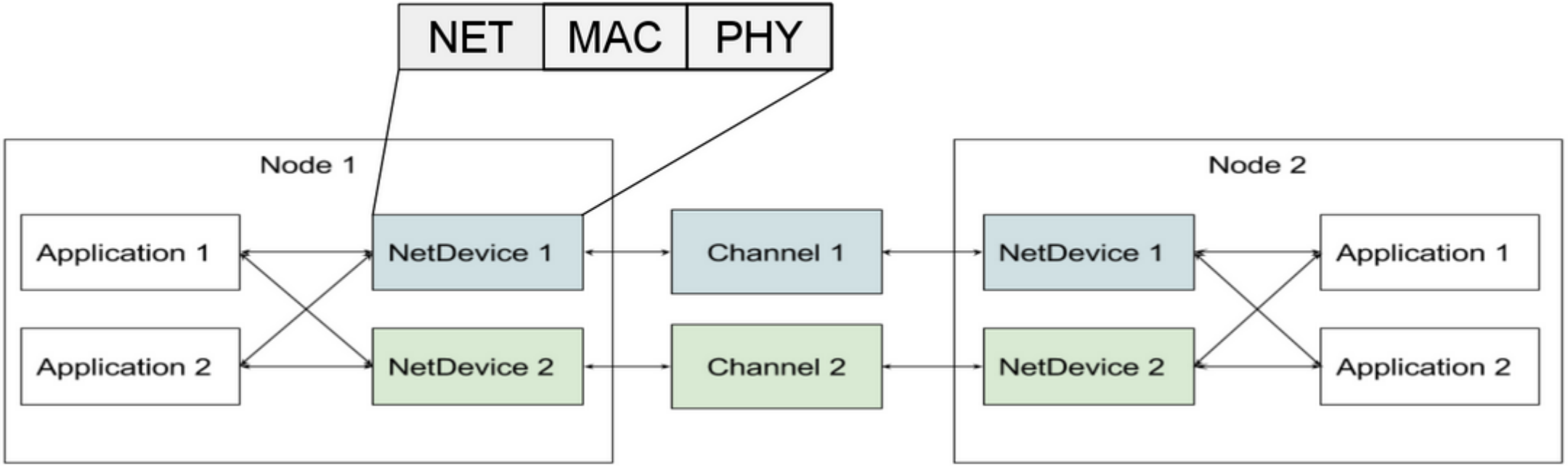
# Estrutura NS3: Módulos

- aodv
- applications
- bridge
- config-store
- core
- csma-layout
- csma
- dsdv
- emu
- energy
- flow-monitor
- internet
- lte
- mesh
- mobility
- mpi
- netanim
- network
- nix-vector-routing
- olsr
- point-to-point-layout
- point-to-point
- propagation
- spectrum
- stats
- tap-bridge
- template
- test
- tools
- topology-read
- uan
- virtual-net-device
- visualizer
- wifi
- wimax

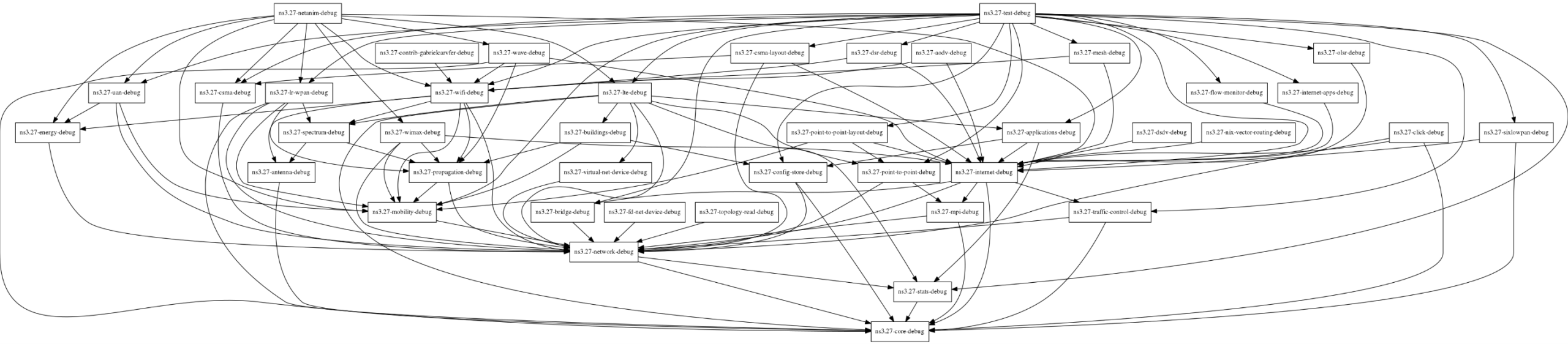
# Estrutura NS3: Estruturação dos módulos



# Estrutura NS3: Estruturação da simulação

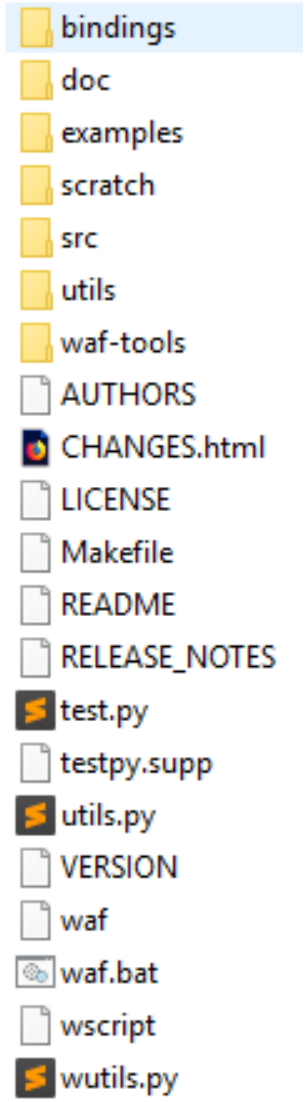


# Estrutura NS3: Dependência entre módulos



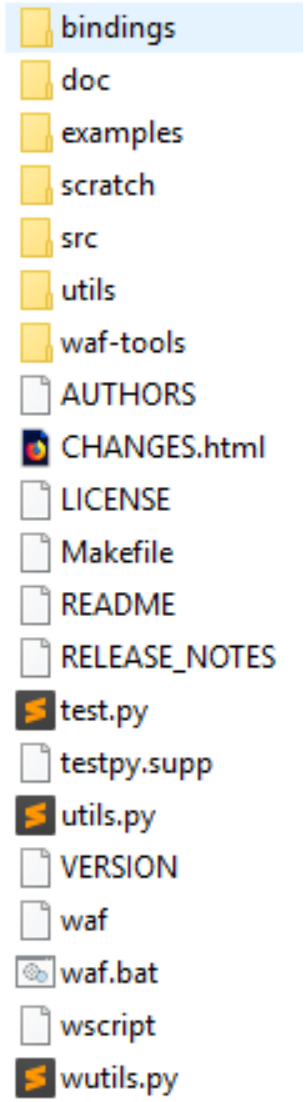
# Estrutura NS3: Estruturação em pastas

- Bindings - Interface python
- Doc - Documentação
- Exemplos
- Scratch - Pasta default para simulação (tudo que estiver dentro dela é compilado automaticamente pelo Waf)
- Src – Módulos (modelos, protocolos, dispositivos, etc)
- Utils – Ferramentas (cobertura de código, benchmark, etc)
- Waf-tools – Ferramentas para sistema de build Waf
- Waf – Sistema de build esotérico orientado à objetos



# Estrutura NS3: Estruturação em pastas

- Bindings - Interface python
- Doc - Documentação
- Exemplos
- Scratch - Pasta default para simulação (tudo que estiver dentro dela é compilado automaticamente pelo Waf)
- Src – Módulos (modelos, protocolos, dispositivos, etc)
- Utils – Ferramentas (cobertura de código, benchmark, etc)
- Waf-tools – Ferramentas para sistema de build Waf
- Waf – Sistema de build esotérico orientado à objetos



# Como instalar o NS3: Dependências

- Sistema Unix-like (Linux, BSD, Mac OS X)

  Windows Subsystem for Linux ainda falha em alguns testes

- Para simular e depurar simulação:
  1. gcc
  2. g++
  3. python
  4. tcpdump
  5. gdb

# Como instalar o NS3: método oficial

- Baixe o NS3 em
  - https://www.nsnam.org/release/ns-allinone-3.27.tar.bz2
- Descompacte
- Configure e monte
  - ./waf --build-profile=debug --enable-examples --enable-tests configure
- Simule
  - ./waf --run hello-simulator

# Como instalar o NS3: com cmake

- Baixe o NS3 com
  - Git clone –b NS3.27 https://github.com/Gabrielcarvfer/NS3.git
- Configure e monte
  - Cd NS3 && mkdir cmake && cd cmake
  - Cmake ../
  - Make && cd ..
- Simule
  - ./build/bin/hello-simulator

# Como simular: método oficial

- Crie um script “sim.cc” ou “sim.py” na pasta scratch
- Execute ./waf scratch/sim

# Como simular: com cmake (demorado)

- Na pasta do NS3 com Cmake (que ainda não tem suporte python)
- Crie um script “sim.cc” na pasta scratch
- Adicione um novo alvo no arquivo scratch/CMakeLists.txt para sim
- Volte para a pasta raiz
- Execute
  - Cd cmake && cmake ../ && make && cd .. && ./build/bin/sim

# Como simular: com cmake (rápido)

- Não tem suporte python (ainda)
- No mesma pasta com o NS3 com Cmake, execute:
  - Git clone https://github.com/Gabrielcarvfer/NS3-CMake-project-example
  - Entre na pasta
  - Crie uma pasta cmake
  - Modifique source.cpp com seu script
  - Execute seu script
- Ou execute o seguinte
  - Git clone https://github.com/Gabrielcarvfer/NS3-CMake-project-example && cd NS3-Cmake-Project-example && mkdir cmake && cd cmake && cmake ../ && make && cd .. && ./bin/NS3-CMake-project-example.exe

# Primeira simulação: estrutura básica

- Inclusão de cabeçalhos
- Usar namespace
- Script da simulação contido numa função main
  - Configuração de opções de parâmetros por linhas de comando
  - Configuração da simulação
  - Execução da simulação
  - Liberação de recursos alocados

# Primeira simulação: cabeçalhos

- Como visto, NS3 é organizado em módulos
- Cada módulo tem cabeçalhos que incluem todos os outros cabeçalhos do módulo correspondente
- Exemplo: wifi-module.h

# Primeira simulação: namespace

- Se preferir evitar usar ns3::alguma coisa, adicione o namespace ns3

```
int main ()
{
  ns3::Simulator::Run ();
  ns3::Simulator::Destroy ();
  return 0;
}
```

# Primeira simulação: namespace

- Se preferir evitar usar ns3::alguma coisa, adicione o namespace ns3

```
using namespace ns3;
int main ()
{
  Simulator::Run ();
  Simulator::Destroy ();
  return 0;
}
```

# Primeira simulação: logs

- Cada componente que suporte logs pode ter eles capturados se configurados para tal

- Exemplo: capturar log de um servidor UDP

  LogComponentEnable(“UdpEchoServerApplication”, LOG_LEVEL_INFO)

# Primeira simulação: logs

- Níveis de log:

  NS_LOG_ERROR — Log error messages;
  NS_LOG_WARN — Log warning messages;
  NS_LOG_DEBUG — Log relatively rare, ad-hoc debugging messages;
  NS_LOG_INFO — Log informational messages about program progress;
  NS_LOG_FUNCTION — Log a message describing each function called;
  NS_LOG_LOGIC – Log messages describing logical flow within a function;
  NS_LOG_ALL — Log everything.

# Primeira simulação: criando nós de rede

- A classe NodeContainer permite criar e manipular nós facilmente

- Um exemplo de uso:

```
NodeContainer nodes;
nodes.Create (2);
```

# Primeira simulação: criando nós ponto-a-ponto

- É possível configurar nós como ponto a ponto através do Helper PointToPointHelper
- Exemplo de uso:

```
PointToPointHelper pointToPoint;
pointToPoint.SetDeviceAttribute(“DataRate”, StringValue(“5Mbps”)); //Configura throughput
pointToPoint.SetChannelAttribute(“Delay”, StringValue(“2ms”)); //Configura atraso

NetDeviceContainer devices; //Estrutura para dispositivos de rede conectados
devices = pointToPoint.Install(nodes); //Cria estruturas para interface de rede dos nós
```

# Primeira simulação: criando pilha TCP/IP

- Até agora, temos 2 nós de redes ligados fisicamente, porém sem protocolo definido. Podemos instalar a pilha TCP/IP
- Exemplo:

```
InternetStackHelper stack;
stack.install (nodes);  //Instala a pilha TCP IP nos nós de rede
Ipv4AddressHelper address;

//Configura rede IP com endereço base 10.1.1.0 e máscara 255.255.255.0
Address.SetBase(“10.1.1.0”, “255.255.255.0”);

//Interfaces de rede recebem endereço IP
Ipv4InterfaceContainer interfaces = address.Assign(devices);
```

# Primeira simulação: aplicações

- Com a rede ponto-a-ponto conectada, usando protocolo TCP/IP, podemos instalar aplicações nos nós

# Primeira simulação: aplicações

- Exemplo de instalação do servidor UDP Echo (que recebe uma mensagem e a devolve a quem enviou)

```
//Servidor de UDP echo escuta porta 9
UdpEchoServerHelper echoServer(9);

//Instala aplicação UdpEchoServer para nó 1
ApplicationContainer serverApp = echoServer.Install(nodes.Get(1));

//Servidor ficará ativo desde o tempo 0s da simulação até 10s
serverApp.Start(Seconds(0.0));
serverApp.Stop(Seconds(10.0));
```

# Primeira simulação: aplicações

- Exemplo:

```
//Cliente de UDP echo se conecta ao servidor no endereço do nó 1 com porta 9
UdpEchoClientHelper echoClient (interfaces.GetAddress(1), 9);

//Configuramos o cliente
echoClient.SetAttribute("MaxPackets", UintegerValue(10)); //Número de pacotes para enviar
echoClient.SetAttribute("Interval", TimeValue(Seconds(1.0))); //Intervalo entre pacotes
echoClient.SetAttribute("PacketSize", UintegerValue(1024)); //Tamanho dos pacotes

//Instala aplicação UdpEchoClient para nó 0
ApplicationContainer clientApp = echoClient.Install(nodes.Get(0));

//Cliente ficará ativo desde o tempo 0s da simulação até 10s
clientApp.Start(Seconds(0.0));
clientApp.Stop(Seconds(10.0));
```

# Primeira simulação: Tracing

- Tracing permite coletar dados do tráfego de rede para posterior análise
- Pode ser em Texto ou Pcap
- Exemplo de uso:

```
AsciiTraceHelper asciiTrace;
pointToPoint.EnableAsciiAll(asciiTrace.CreateFileStream(“firstSim.tr”);
pointToPoint.EnablePcapAll(“firstSim”);
```

# Primeira simulação: demonstração

# Simulador

- A simulação do NS3 é baseada em eventos, que ocorrem num dado instante de tempo ou com certa frequência, e callbacks aos elementos responsáveis pelo evento para que processem sua lógica

# Simulador: Ponteiros

- Objetos do NS3 podem ser acessados através do template de ponteiro Ptr<T>, onde T é a classe alvo.

- Exemplo de criação e manipulação de ponteiros

```
Ptr<ApWifiMac> n = CreateObject<ApWifiMac>();

Template <typename T>
Ptr<Object> CreateObject <x>()
{
        ObjectFactory factory;
        Const std::string typeId = typeid(T).name();
        //”ns3::ApWifiMac
        factory.SetTypeId (typeId);
        Ptr<Object> node = factory.Create<Object>;
        return node;
}
```

# Simulador: Objetos

- No NS3, tudo deriva da classe Object, que implementa funcionalidades básicas (acesso de atributos, nome do tipo, ponteiros, etc)

# Simulador: Atributos

- São usados tanto em objetos quanto para guardar configurações do simulador (ver ConfigStore)
- Proveem uma interface genérica de acesso aos dados dos objetos: nome, descrição, valor padrão, funções de acesso e de validação

- Exemplo

```
TypeId ApWifiMac::GetTypeId (void) {
      static TypeId tid = TypeId ("ns3::ApWifiMac")
        .SetParent<RegularWifiMac> ()
        .SetGroupName ("Wifi")
        .AddConstructor<ApWifiMac> ()
        .AddAttribute ("BeaconInterval",
                "Delay between two beacons",
                TimeValue (MicroSeconds (102400)),
                MakeTimeAccessor (
              &ApWifiMac::GetBeaconInterval,
                    &ApWifiMac::SetBeaconInterval),
                MakeTimeChecker ())
[...]
```

# Simulador: Eventos

- Eventos são agendados com uma das seguinte funções
  - Simulator::Schedule(Seconds(x), &func, param1, ..., paramN-1);
  - Simulator::ScheduleNow(&func, param1, ..., paramN-1);
  - Simulator::Schedule (Time const &time, MEM mem_ptr, OBJ obj);

- A classe time faz conversão do valor de entrada (double) para milisegundos, segundos, minutos, horas, dias, etc

# Simulador: Callbacks

- Ponteiros para função

```
int func (int param)
{
    return param++;
}

int (*callbackFunc)(int param) = func;

int res = callbackFunc (1); // res = 2
// ou
int res = (*callbackFunc) (1); // res = 2
```

# Simulador: ConfigPath

- O NS3 é estruturado de maneira a permitir que sejam acessados objetos ou atributos através de um caminho
- Exemplo:

```
//Suponha um nó 1, que se movimenta com um modelo derivado
// de Mobility Model e uma função CourseChange que determina
// a mudança de curso do nó
// Podemos mudar a função que determina a mudança de curso com
Config::Connect("/NodeList/1/$ns3::MobilityModel/CourseChange",
        MakeCallback(&CourseChange));
```

# Helpers: encapsulamento de modelos

- Existem diversos modelos dentro de cada módulo
- No módulo WiFi, por exemplo, existem modelos para estações (STA) e access points (AP)
- WifiMacHelper permite que criemos e configuremos todos estes modelos MAC com uma interface genérica
- Exemplo:

```
WifiMacHelper mac;
mac.SetType(“ns3::StaWifiMac”); // Usado para configurar estações
//ou
mac.SetType(“ns3::ApWifiMac”); // Usado para configurar access points
```

# Helpers: encapsulamento de tracing

- Assim como existem helpers para modelos/componentes, existem helpers para tracing, como o AsciiTraceHelper, que abre um arquivo do tipo trace com nome indicado para coleta de informações.

# Coletando dados: PCAP

- Os dados podem ser coletados em formato PCAP (padrão do Wireshark)
- Rastreia todos pacotes enviados pelo componente rastreado
- Exemplo:

```
//phy pode ser qualquer outro componente
phy.EnablePcapAll (“trace.pcap”, true);
```

# Coletando dados: tr (padrão NS-3)

- O tipo padrão de coleta de dados do NS3 é trace (.tr)
- Resume todos os caminhos e operações efetuadas do componente rastreado
- Exemplo:

```
AsciiTraceHelper asciiTrace;
//phy pode ser qualquer outro componente
phy.EnableAsciiAll(asciiTrace.CreateFileStream("trace.tr"));
```

# Coletando dados: Log

- Logs são impressos no terminal no qual a simulação foi executada
- É possível salvar os logs redirecionando a saída para um arquivo
- Exemplo:

  ./sim > out.log

# Coletando dados: Netanim

- NetAnim permite ver dados da simulação de maneira gráfica
- Para coletar as informações, é necessário adicionar os cabeçalhos

```
#include <ns3/netanim-module.h>
#include <ns3/bs-net-device.h>
#include <ns3/csma-module.h>
#include <ns3/uan-module.h>
```

- Ao final do arquivo da simulação, antes de iniciar o simulador, faz-se a configuração para o NetAnim

```
BaseStationNetDevice b;
SubscriberStationNetDevice s;
CsmaNetDevice c;
UanNetDevice u;
AnimationInterface anim(outputFolder+"anim2.xml");
anim.SetMaxPktsPerTraceFile(0xFFFFFFFF); anim.EnablePacketMetadata(true);
```

# Coletando dados: trace source e trace sink

# Processando dados: PCAP

- Wireshark é seu amigo
- Demonstração

# Processando dados: tr (padrão NS-3)

- O arquivo pode ser processado com processadores de texto, como awk, ou ferramentas específicas, como TraceMetrics

- Demonstração (do arquivo)

# Processando dados: Log

- Processadores de texto como awk

# Processando dados: Netanim

- Dados de saída do Netanim podem ser abertos e visualizados no NetAnim, que acompanha o NS3
- Evite as versões distribuídas com SO, porque geralmente tem problemas com versões mais recentes do NS3
- Demonstração