ns-3-nsc

Using real-world network stacks in discrete-event network simulation

Florian Westphal

21st June 2008


```
   1024D/F260502D <fw@strlen.de>
1C81 1AD5 EA8F 3047 7555
E8EE 5E2F DA6C F260 502D
```

Agenda

- Introduction to ns-3 and simulation
- Design and model
- creating simulations
- Introduction to NSC
- NSCs design
- ns-3-nsc archi^W plumbing
- TODO and Wishlist

ns-3

- network simulator for internet systems
- successor to ns-2
- development started in 2006, work in progress
- written in C++
- GPL v2 license

ns-3 and ns-2

ns-2:

- consists of a C++ core and an OTcl interpreter "frontend"
- network topology is set up using OTcl script

ns-3:

- builds a shared library, `libns3.so` (there is no "`ns3`" program)
- simulations are written in C++
- python bindings are being worked on

ns-3 simulation

- programmatic description of network:
  - create node objects
  - create network topology
  - assign ip addresses, etc.
- simulation is run by the ns-3 scheduler
- simulation time moves from event-to-event

Outputs: pcap traces (per interface); customizeable trace namespace to get particular events (eg. ipv4 tx on interface i_1)

ns3 layers

- Applications: socket API, OnOff Application, Sink, . . .
- Transport Layer: TCP, UDP
- Network Layer: IPv4, Routing (static), . . .
- Link Layer: PPP, CSMA, WiFi, . . .
- Physical Layer: Loss, Delay, . . .

ns-3 nomenclature/concepts

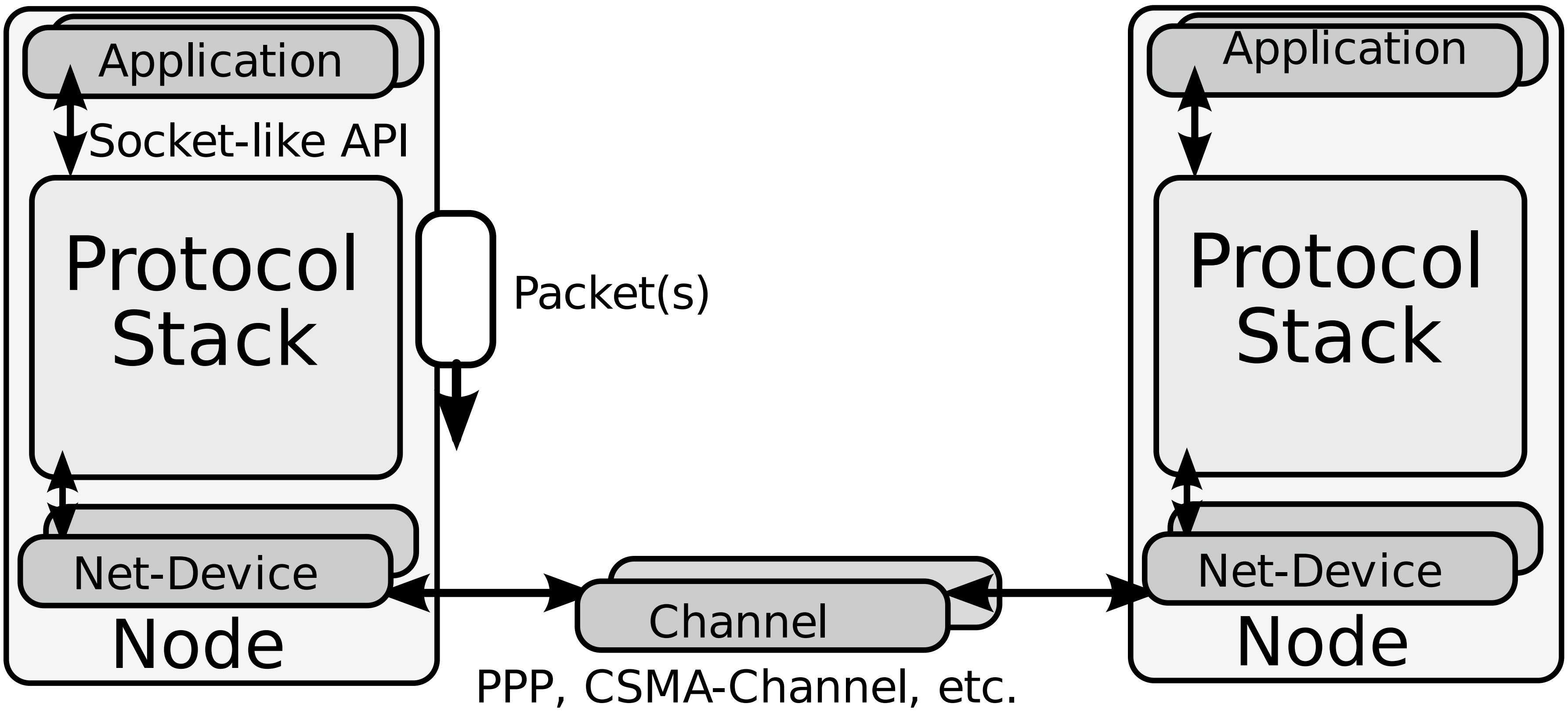
- A simulated network has **Node**s ("computers")
- Each node uses a particular protocol stack, e.g. TCP/IP
- Node can run applications that talk to network via socket-like interface
- Nodes have one or more **NetDevice**s ("network interfaces")
- network interfaces are connected to particular **Channel**s, e.g. PPP, CSMA, . . .
- Data is exchanged using **Packet**s

The Packet class

- Abstraction for network packets
- contains byte buffer with a serialized representation of packet
- methods to add/remove protocol headers and data
- can add metadata/“tags” to a packet

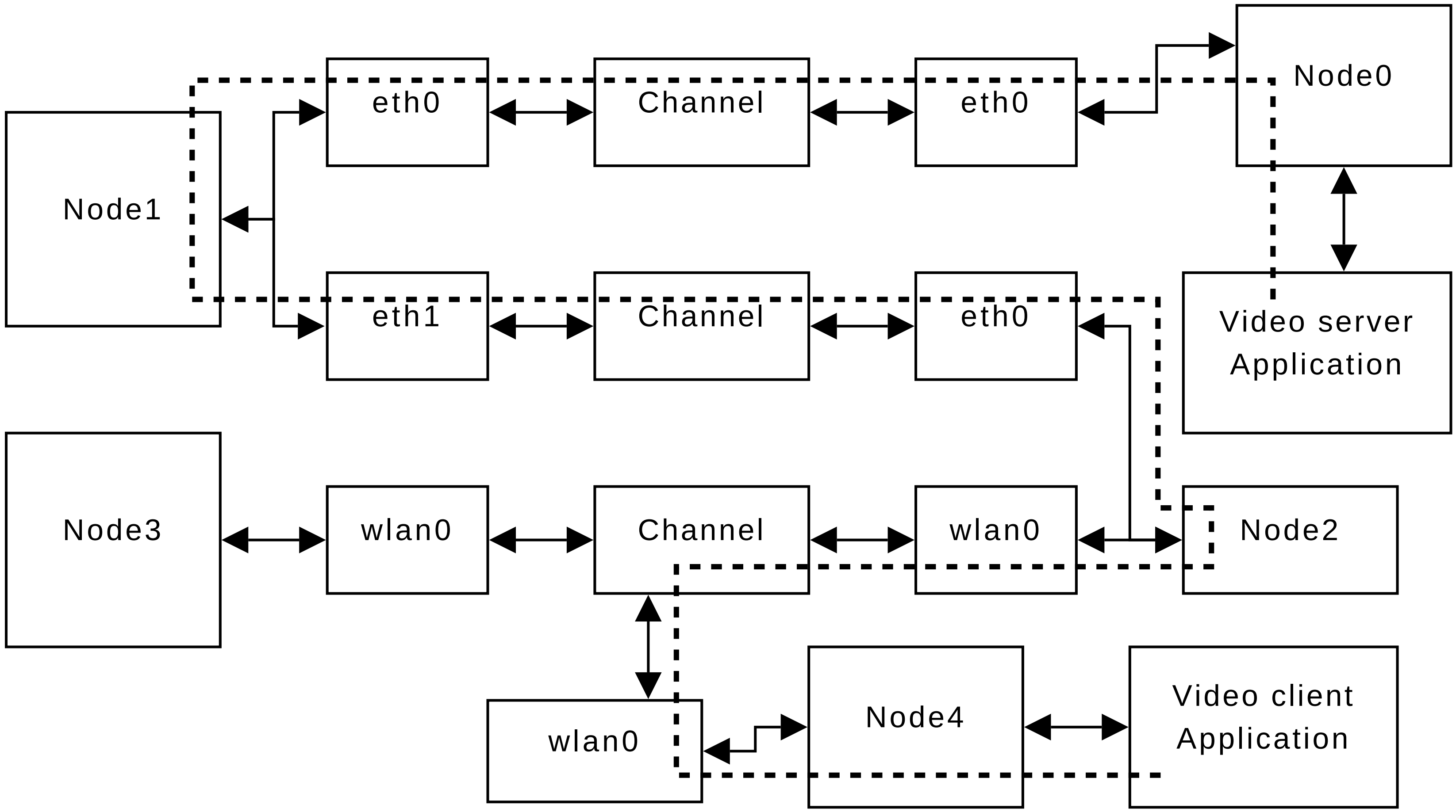
All Headers are also classes, with Serialize/Deserialize methods

ns-3 model



Source: ns-3 tutorial slides, by Tom Henderson[1]

a high level topology structure



(taken from paper by Mathieu Lacage, used with kind permission)

describing a topology – 2 nodes, ethernet

```
int main (...) {
  NodeContainer c0;
  c0.Create (2);
```

Then, create a csma channel: `CsmaHelper csma;`. Can set properties, e.g. Delay and DataRate:

```
csma.SetChannelParameter ("DataRate", StringValue ("10Mbps"));
csma.SetChannelParameter ("Delay", TimeValue (MicroSeconds(500)));
```

Next, connect the nodes in the container to this channel: `NetDeviceContainer dev0 = csma.Install (c0);`

setting IP addresses

```
InternetStackHelper internet;
internet.Install (c);
Ipv4AddressHelper ipv4;
ipv4.SetBase("192.168.0.0", "255.255.255.0");
ipv4.Assign(dev0);
```

and a routing method: `GlobalRouteManager::PopulateRoutingTables();`
`Assign()` automatically sets up proper IP addresses within the subnet.

Setting applications

Set up a TCP discard service:

```
PacketSinkHelper sink("ns3::TcpSocketFactory",
       InetSocketAddress(Ipv4Address::GetAny (), Port));
```

. . . install it on node 2 and start it 'now'

```
ApplicationContainer apps = sink.Install (c.Get (1));
apps.Start (Seconds (0));
```

Creating an application

Now all we need is to create an application... ns-3 offers a socket-like interface.

```
Ptr<Socket> localSocket =
  Socket::CreateSocket(c.Get (0), TcpSocketFactory::GetTypeId());
Simulator::ScheduleNow(&StartFlow, localSocket, bytecount,
                       Ipv4Address("192.168.0.2"), Port);
```

This creates a new local socket and calls a method "StartFlow" at the start of the simulation.

StartFlow

```
void StartFlow(Ptr<Socket> localSocket, uint32_t nBytes,
            Ipv4Address servAddress, uint16_t servPort) {
localSocket->Connect(InetSocketAddress (servAddress, servPort));

localSocket->SetConnectCallback (MakeCallback (&CloseConnection),
                                Callback<void, Ptr<Socket> > ());
uint32_t sent;
while (sent < nBytes) {
 [..]
 ret = localSocket->Send (data, curSize);
 [..]
```

Starting the simulation

Need to call `Simulator::Run ();` to run the simulation. Can call `CsmaHelper::EnablePcapAll ("tcp-large-transfer");` before that to get pcap-dumps of all interfaces.

```
$ /usr/sbin/tcpdump -n -r tcp-large-transfer-1-0.pcap
reading from file tcp-large-transfer-1-0.pcap,
             link-type EN10MB (Ethernet)
02:00:00.000044 arp who-has 192.168.0.2
                   (ff:ff:ff:ff:ff:ff)tell 192.168.0.1
[..]
```

NSC – network simulation cradle

- developed by Sam Jansen at WAND, 1st release 2005
- essentially provides wrapper/glue to run a kernel TCP/IP stack in user space
- . . . and in a network simulator
- runs FreeBSD 5.3, OpenBSD 3.5 and Linux 2.6.18 network stacks
- NSC is fairly independent of the actual network simulator
- $\rightarrow$ want to use NSC with ns-3

TCP Stack $\leftrightarrow$ NSC $\leftrightarrow$ Network Simulator

ns-3-nsc goal...

- want to hide all NSC details, if possible
- want to use recent real-world network stacks in simulation
- for the time being, hooks into `InternetStackHelper`:

```
internet.SetNscTcp("Linux");
```

NSC – architecture

- Main Problem: A stack is usually only for a single host
  - need separate stacks for each node in the simulation
  - Nowadays, one could try to re-use e.g. Linux Network Namespaces (OS Virtualization) $\Rightarrow$ Stack specific, not all support namespaces
- NSC solution: (Mostly) Automated source code transformation using the "globalizer"
- each stack is compiled into a shared library (e.g. `liblinux2.6.18.so`)

The globalizer

- The globalizer is a program to transform C code
- No simple search & replace (macros, typedefs, variables on stack, . . .)
- Reads (preprocessed) C code from stdin and writes result to stdout
- Also reads a list of global symbols to be replaced, e.g. `global_var` `function/static_var`
- Outputs the transformed program code "`long global_var` → `long global_var[NUM_STACKS]`" "`global_var = x` → `global_var[get_sta`
- Globalizer also handles cases like `*global_var = &global_2`

Cradle

- the cradle provides the neccessary infrastructure for each network stack
- fake ethernet driver
- lots of support code (e.g. Linux:)
  - `copy_from_user` $\rightarrow$ memcpy
  - `register_filesystem`, inode handling, . . .
  - kernel start up routine ("`do_initcalls`")
- cradle also defines API to map syscalls and set up the stack
- defines timer interrupt method that is called by the simulator periodically (e.g. every 10 ms)

NSC API

NSC provides its API in the form of classes

- INetStack class:
  - low level network stack functions: eth driver Input, sysct, . . .
  - also has method to create new Sockets (INetStreamSocket).

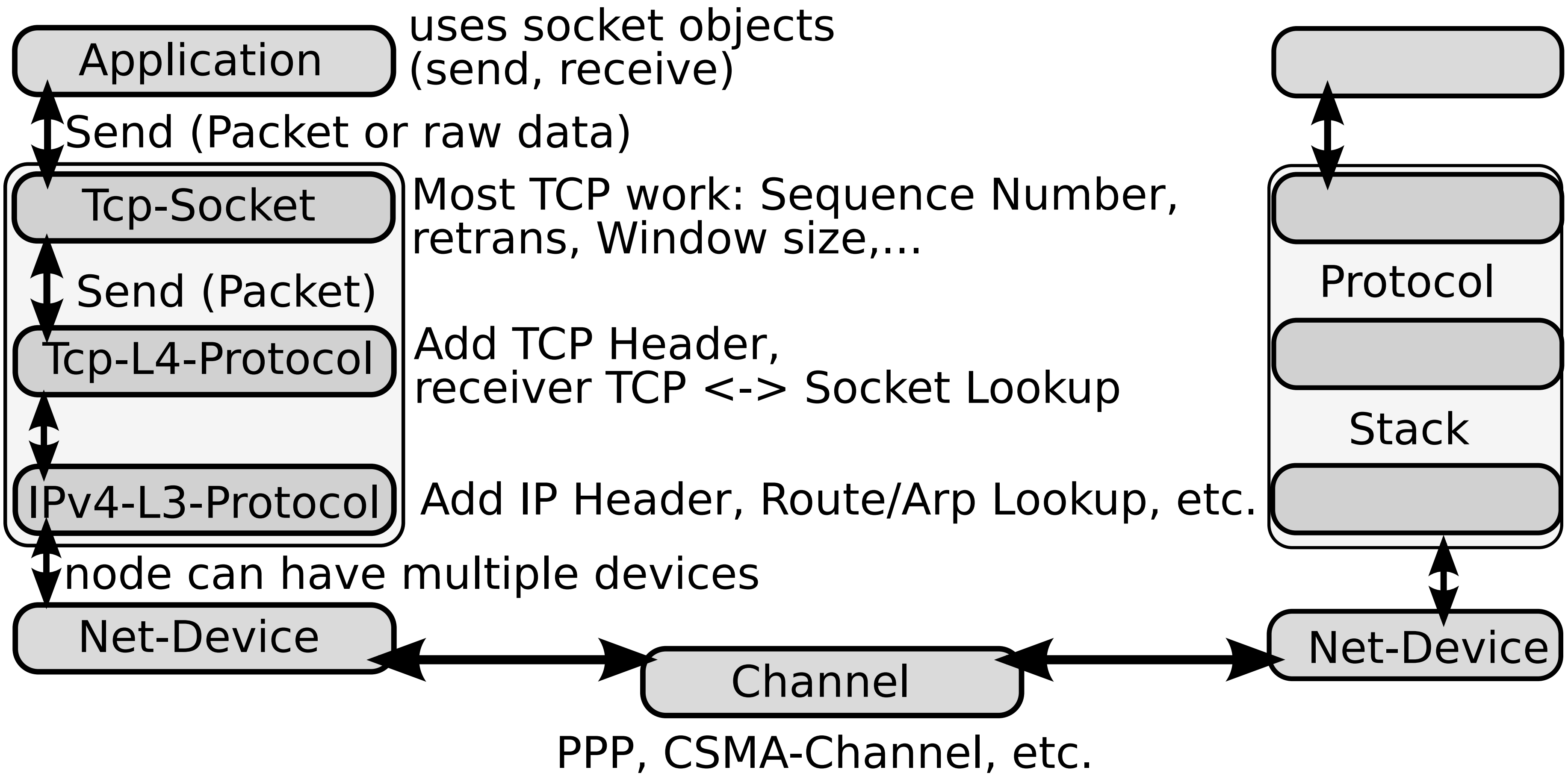
- INetStreamSocket
  - a connection endpoint ("file descriptor"), belongs to a stack instance
  - methods to change state of endpoint: e.g. connect, accept . . .
  - methods to read/write data

nsc callbacks

- ISendCallback
  - called when NSC sends a packet out to the network
  - called by NSCs ethernet driver
  - This hook is used to re-inject packet into the simulator

- IInterruptCallback
  - called by NSC whenever something "interesting" happens
  - the Linux NSC glue calls this when the stack calls `__wake_up`
  - the simulator can then check for state change on the connection endpoints

ns-3 TCP model

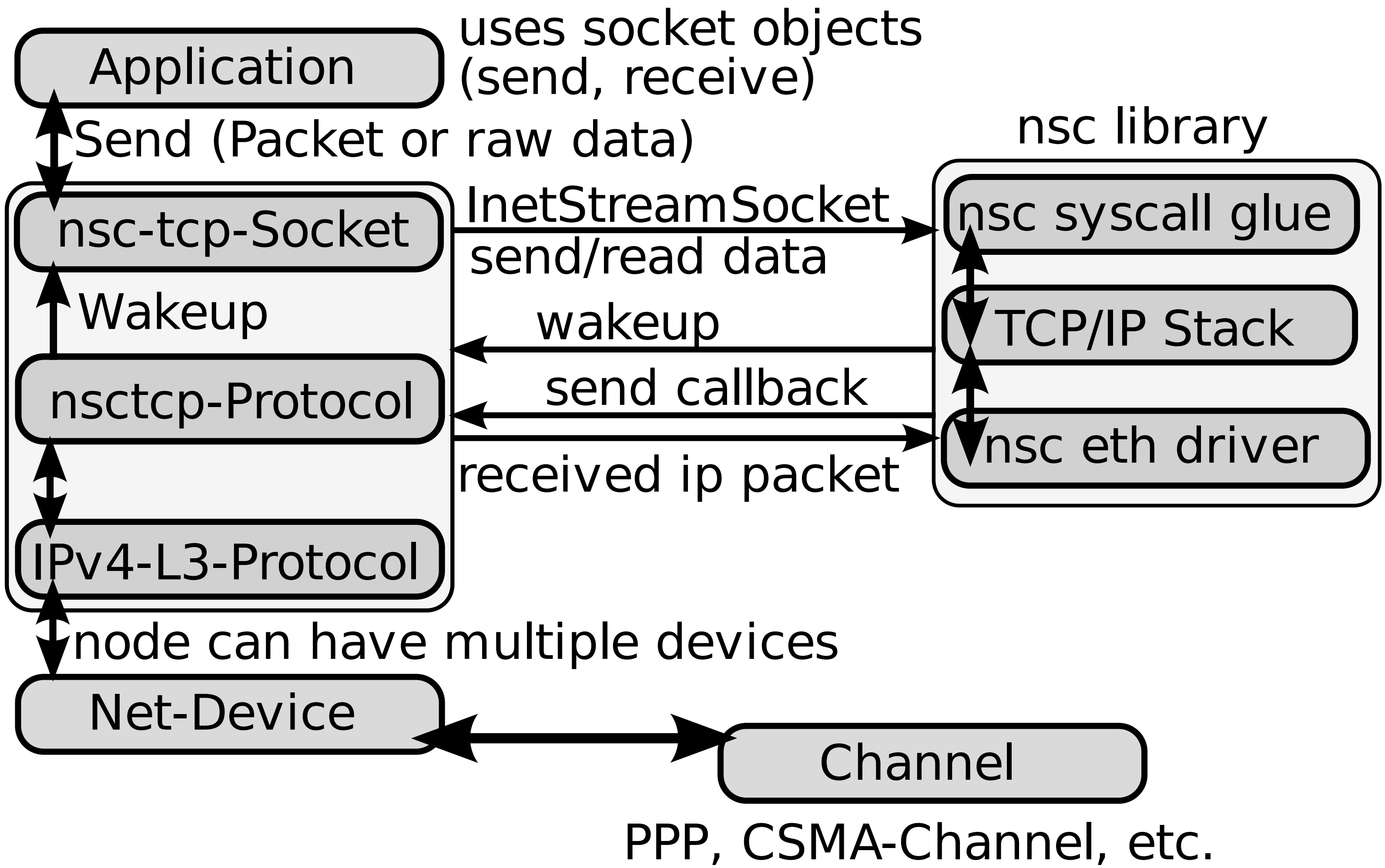


Integration work...

most of ns3-nsc-glue resides in two classes:

- NscTcpL4Protocol (wraps the actual stack)
  - maps the stack (shared library) using `dlopen()`
  - deals with in/and output to the NSC ethernet driver
- NscTcpSocketImpl (wraps a INetStreamSocket)
  - deals with socket in/output
  - does not push any data to NscTcpL4Protocol directly

ns-3-nsc model



Current state...

- simple scenarios work well
- can observe e.g. SACK with Linux hosts
- can run mixed setups, i.e. Linux $\leftrightarrow$ FreeBSD
- there are bugs, e.g. segfault if NSC receives packet (from netdevice) before a socket is created
- NSC has to be extended to provide information needed by ns-3 (eg peer IP address)

ns-3-nsc Wishlist...

- ns-3 is still in the alpha stage, e.g. ns-3-nsc needs to queue data in SYN_SENT state
- impossible to identify errors, because errno values are stack dependant
  - need some kind of nsc_errno to map to corresponding stack errno
- no setsockopt interface in ns-3-nsc (NSC supports this)
- same for sysctl

Sources/Literature/Acknowledgements

1 Tom Henderson. ns-3 tutorial slides, simultools 08 conference. `http://www.nsnam.org/tutorials/simutools08/ns-3-tutorial-slides.pdf`

- NSC pages: `http://www.wand.net.nz/~stj2/nsc/`
- ns3 project homepage: `http://www.nsnam.org`

Thanks to: Mathieu Lacage, Tom Henderson, Sam Jansen & The ns-3 team.