The libfluid OpenFlow Driver Implementation

Allan Vidal[1], Christian Esteve Rothenberg[2], Fábio Luciano Verdi[1]

[1]UFSCar [2]UNICAMP

32nd Brazilian Symposium on Computer Networks and Distributed Systems
May 5-9 2014

Outline

SDN

What is SDN?

Software-defined networking (SDN) is a set of approaches to networking aiming to improve network programmability, removing logic functionality from close hardware and putting them in the hands of developers

Why SDN?

It emerged as a way to control networks programmatically, as a solution to several management problems common to networks worldwide [1]:

- Automatic configuration
- Global view of the network
- Custom software on commodity hardware

OpenFlow and SDN

What is OpenFlow?

The OpenFlow protocol [2] was proposed as a solution to the lack of programmability in networks. It is a common language (like an API) that is used by control software and networking equipment. Complexity lies in the control agent that commands the network equipment. OpenFlow is an approach to SDN that doesn't break with existing technologies such as Ethernet and TCP/IP.

SDN and OpenFlow: overview

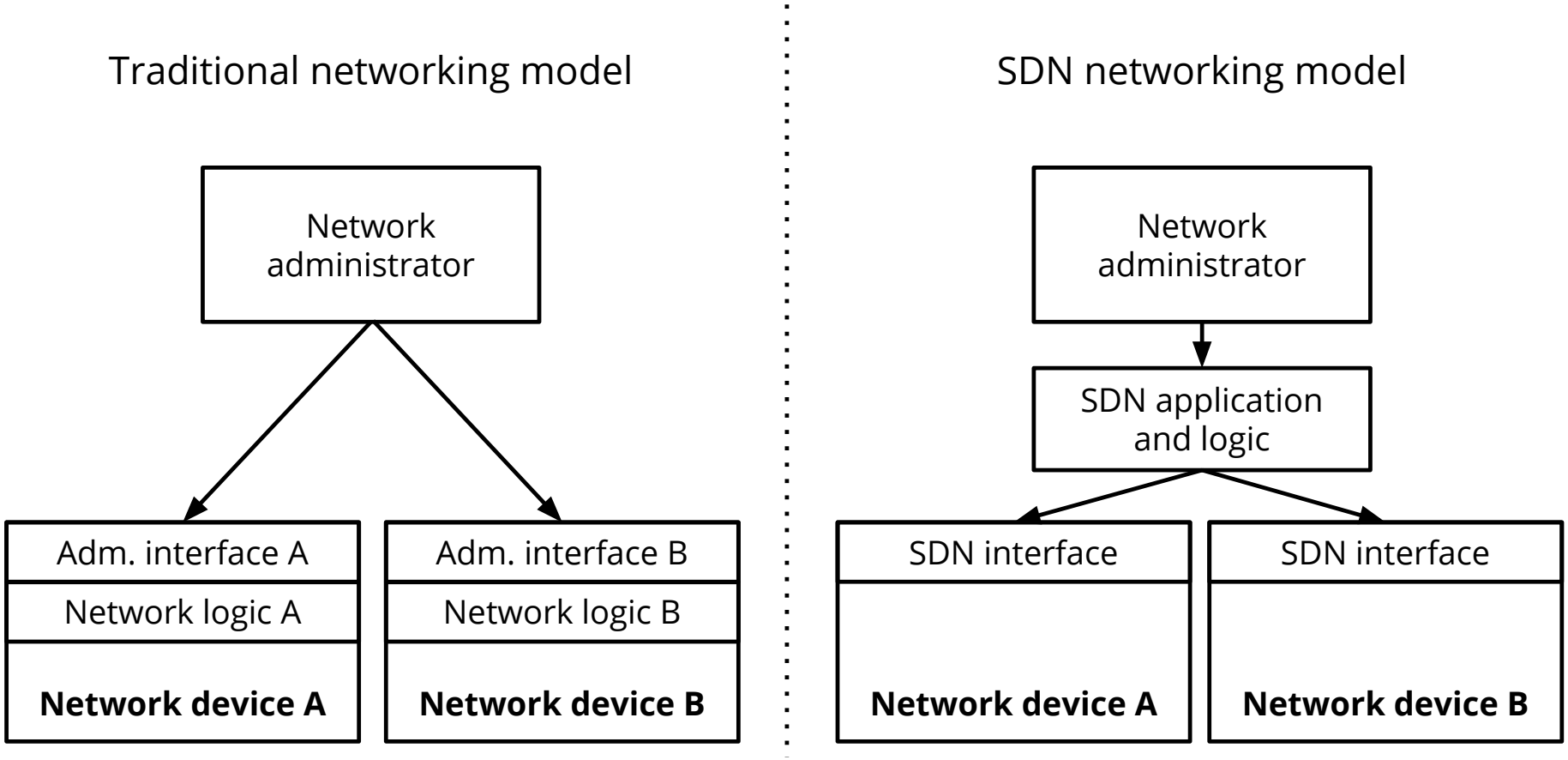

Figure: Traditional and SDN models compared

Identified issues

SDN in general

- Issue 1: a unified connectivity layer for both controllers and equipment agents
- Issue 2: minimal overhead on applications and no performance compromise
- Issue 3: requirement assumptions of controller applications

Specific to OpenFlow

- Issue 4: a single, portable and lightweight implementation
- Issue 5: protocol version agnosticism
- Issue 6: make it possible to build stand-alone applications

Our proposal

Current OpenFlow layer implementations fall short on the issues we mentioned.
By building a simple set of abstractions for the control channel and providing a clear, separate implementation of the connectivity layer, we can benefit controller, switch and application developers by making development easier.
A small set of tools on top of this implementation can further help adaptation to different use cases and network applications without sacrificing ease of use, performance or technology choices.

General architecture overview

Based on the issues and our ideas, we arrived at the following practical implementation guidelines:

- **Issue 1: a unified connectivity layer for both controllers and equipment agents**
  Implement the software as a library that can be bundled in both sides and use common OOP abstractions to reuse code.
- **Issue 2: minimal overhead on applications and no performance compromise**
  Correctly use an already existing and well-tested solution for the event loop library.
- **Issue 3: requirement assumptions of controller applications**
  Don't overuse object orientation abstractions and explore how multithreading can be leveraged to answer different requirements.

General architecture overview (cont.)

Based on the issues and our ideas, we arrived at the following practical implementation guidelines:

- **Issue 4: a single, portable and lightweight implementation**
  Implement the software as a portable library and use a third-party, portable and lightweight event loop library as the basis.
- **Issue 5: protocol version agnosticism**
  Completely separate the messaging and connectivity layers so that the connectivity layer only assumes a few basic, unchanging protocol guarantees.
- **Issue 6: make it possible to build stand-alone applications**
  Implement the software as a library, so that it can be directly linked into applications.

General architecture overview (cont.)

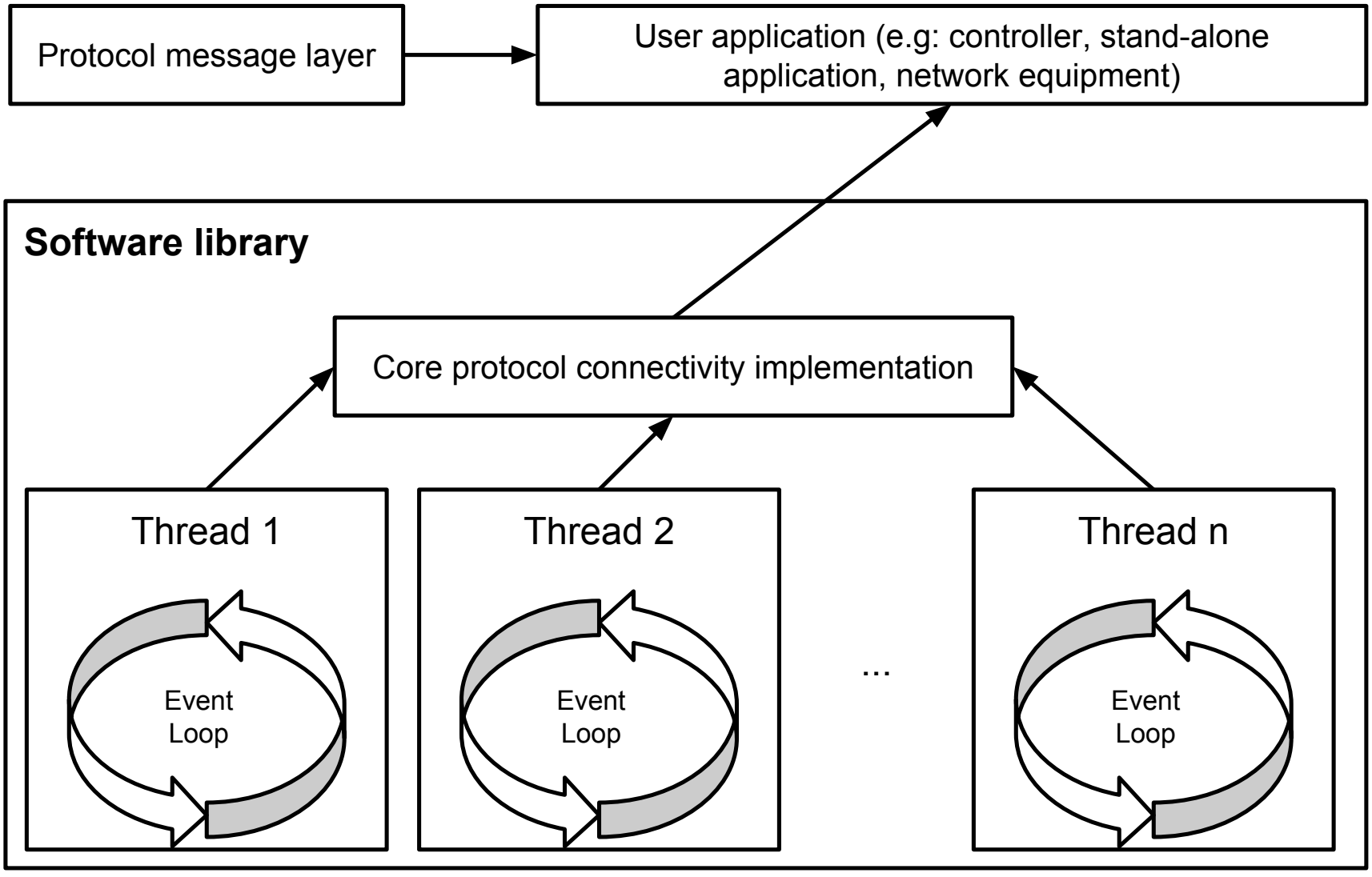

Figure: Architecture overview

The tool

- libfluid is the tool we are presenting to solve these issues.
- Built for the Open Networking Foundation competition [3] and chosen as the winner entry (9 were submitted, worldwide), being awarded a cash prize and highlighted internationally.
- This library is divided in two parts, taking the direction we outlined in our research so far: **libfluid_base** and **libfluid_msg**. Our main point of interest in this presentation is **libfluid_base**.

Architecture

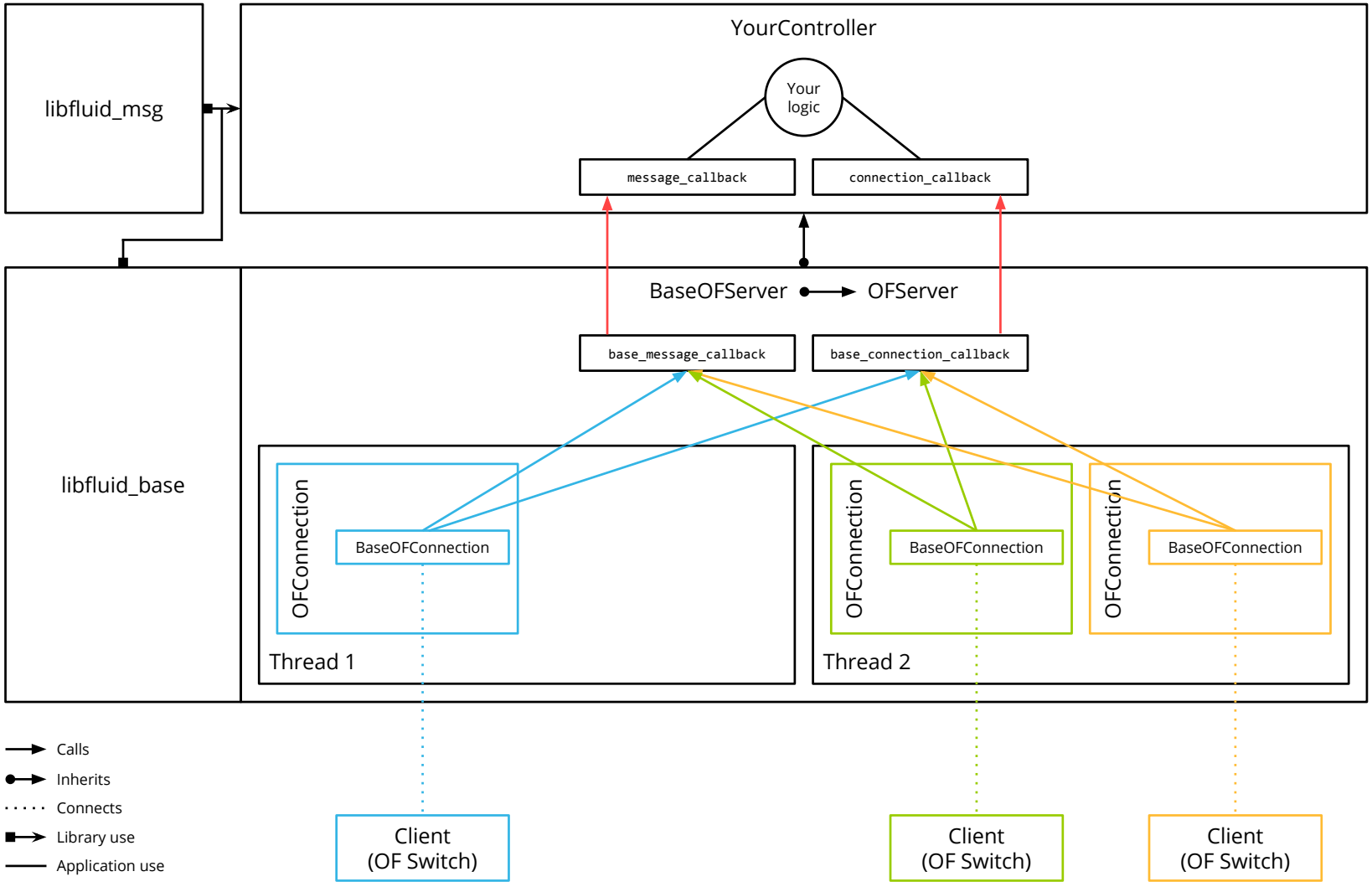

Figure: libfluid architecture

Architecture: key points

- We have already demonstrated a way of building a controller (server) and a switch (client) using the same code base, reusing the abstractions we built.
- We decided to leave the responsibility for dealing with each version of the protocol to **libfluid_msg**. **libfluid_base** is completely agnostic to OpenFlow versions and message parsing.
- A library is used for providing an event loop that runs in each thread, dealing with socket management and IO (libevent [4]).

Architecture: key points (cont.)

- A user specifies how many threads it wants to use when initializing the driver, and these threads will be responsible for listening to connection events.
- The user must implement two methods for handling major types of events: message and connection events.
- The user needs to implement the two callback methods and call a start method to have a bare-bones OpenFlow controller running.

Architecture: key points (cont.)

- The way of deploying threading is a core issue to our research, which impacts on design decisions. We will be looking into how we can take advantage of multithreading to prioritize events.
- C++ was chosen as the main language due to higher level abstractions allowed by OOP, but we are not making use of higher-level features that might pose portability problems (such as templates and exceptions).
- This current implementation is ready for use for any use you can think of: prototyping, embedding in hardware, building applications, or just having fun with OpenFlow!

Demonstration

We will be showing:

- A quick presentation on our documentation and other resources available for developers.
- A quick overview of the installation process.
- A sample dummy controller application will be shown in detail, and we will also take a glance on a learning switch application implemented using libfluid.

The code and resources are available at:
`http://opennetworkingfoundation.github.io/libfluid/`

Conclusion

- By recognizing the distinct nature of the different parts in an SDN control protocol, we have identified a few issues in current solutions and possible improvements to them.
- Our solution, though in early stages, shows promising results, being chosen as the winner in an important competition in its area.
- We want to improve it by applying a research-oriented approach to solving issues such as: an improved event model, formalization of the client-server architecture and higher-level abstractions.

References I

[1] Nick Feamster, Jennifer Rexford, and Ellen Zegura. The Road to SDN. *Queue*, 11(12):20, 2013.

[2] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. OpenFlow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 38(2):69–74, 2008.

[3] Open Networking Foundation. Open Networking Foundation. `https://www.opennetworking.org/`, 2014. [Access: 15-Feb-2014].

[4] Nick Mathewson and Niels Provos. libevent. `http://libevent.org/`, 2011. [Access: 15-Feb-2014].