NaNET: Socket API and Protocol Stack for Process-to-Content Network Communication

Massimo Gallo, Lin Gu, Diego Perino, Matteo Varvello
Bell Labs, Alcatel-Lucent
first.last@alcatel-lucent.com

ABSTRACT

Inter-process communication (IPC) refers to the set of methods which enable data exchange among processes. When two processes are remote (connected via a network), IPC is realized through the socket and the networking protocol stack implemented at end-hosts. Today, most computer networks rely on the Internet NETtworking socket domains (INET) and the Internet Protocol (IP) suite.

Information-centric networking (ICN) is a novel networking paradigm centered around named data rather than named hosts. ICN shifts the communication principle from process-to-process towards process-to-content (PCC) by mean of a novel name-based protocol suite. We propose Named NETworking, a new socket domain and protocol stack that realize PCC à la NDN [1] while remaining compatible with current protocols and standards. Finally we implement NaNET in the Unix operating system as a set of kernel modules.

Categories and Subject Descriptors

C.2.1 [**Network Architecture and Designs**]: Network communications; C.2.2 [**Network Protocols**]: Protocol architecture

General Terms

Design, Implementation.

Keywords

NDN, API, stack, protocols.

1. PCC AND NaNET

We call NaNET socket domain and protocol stack that enables PCC à la NDN. NaNET has two main design goals: backward compatibility, and ease of integration with existing operating systems. Accordingly, we rethink current API and protocol stack to support PCC as shown in Figure 1.

To realize an incremental shift from IPC to PCC, our rationale is to extend the Unix implementation of the BSD sockets with a novel socket domain, and to deploy NDN as an overlay over existing technologies. Differently from [5], we propose a protocol stack composed of two layers, transport and network, which sits between socket API and an underlying technology that provides connectivity between two NDN nodes. NaNET supports Ethernet and IP to provide connectivity between two NDN nodes.

ICN'14, September 24–26, 2014, Paris, France.
ACM 978-1-4503-3206-4/14/09.
http://dx.doi.org/10.1145/2660164.

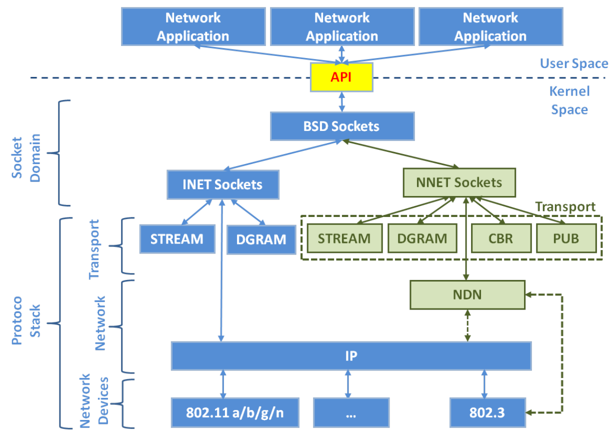

Figure 1: Comparison of TCP/IP and NaNET API and protocol stack.

Similarly to [5], we identify two key processes involved in the *process-to-content* communication model (PCC) used by NDN: the *publisher* and the *consumer*. The publisher process makes content publicly and permanently available to the network by opening one or more sockets that simply wait for incoming requests. The consumer process requests content, based on the end-user interest. Such request targets the desired content (that can be cached in the middle of the network) instead of its publisher process.

In the following we briefly describe NaNET stack and consuper/publisher API.

1.1 Socket Addressing Scheme

NaNET uses the hierarchical naming scheme proposed by NDN to address content. To avoid conversion between application data unit (ADU) and transmission unit, we follow the application layer framing design principle by Clark et. al [4]. Accordingly, an ADU corresponds to a content segment and defines the granularity for which the application can support out-of-order packets and recovery from packet losses. The publisher process is in charge of defining the proper ADU size based on application constraints.

Consumer Operation	**System Call**
Open the socket	`socket(domain,type,protocol)`
Associates a content name or prefix to open a NaNET socket.	`bind(fd,*addr,addrlen)`
Allows to verify if the desired content exists, and to retrieve its meta-data	`connect(fd,*addr,addrlen)`
Set/Get the socket options. e.g., receiver timeout, transmission rate, etc...	`setsockopt(fd,level,optname,*optval,optlen)` `getsockopt(fd,level, optname,*optval,*optlen)`
Start process for named data retrieval. The major difference between these system calls lies in the fact that `recvfrom()` and `recvmsg()` can specify the name of the content to be received, while `recv()` and `read()` do not.	`recv(fd,*buf,len,flags)` `read(fd,*buf,len)` `recvfrom(fd,*buf,buflen,flags,*addr,*addrlen)` `recvmsg(fd,*msg,flags)`
Close the socket	`close(fd)`

Table 1: Consumer operations.

Publisher Operation	**System Call**
Open the socket	`socket(domain,type,protocol)`
Associates a content name or prefix to open a NaNET socket.	`bind(fd,*addr,addrlen)`
Set/Get the socket options. e.g., receiver timeout, transmission rate, etc...	`setsockopt(fd,level,optname,*optval,optlen)` `getsockopt(fd,level, optname,*optval,*optlen)`
Inform underlying network layer that this process can serve all content items whose names fall within a content prefix, triggering forwarding table update.	`listen(fd,backlog)`
Instruct a socket to wait for incoming Interests. Once an Interest is received, it returns the packet to the publisher process that is responsible to handle it.	`accept(fd,*addr,addrlen)`
Send data back to the requester spedifying the data name.	`sendto(fd,*buf,len,flags,*addr,addrlen)` `sendmsg(fd,*msg,flags)`
Close the socket	`close(fd)`

Table 2: Publisher operations.

1.2 Socket API

NaNET is designed to be easily integrated with the current Unix implementation of the BSD socket. Indeed NaNET API mimics current socket's API, though with different functionalities and implementations.

Similarly to [6], we make the NDN communication protocol available through the socket's API by introducing new NaNET address and protocol families, AF_NANET and PF_NANET.

Available Consumer/Publisher system calls and a brief description of their meaning are listed in Tab.1,2.

1.3 Transport Layer

Transport layer has different functionalities at the consumer and producer process.

The consumer's transport layer is responsible for transfer reliability, flow/congestion control, data encapsulation and decapsulation, data multiplexing and demultiplexing. We identify three transport protocols at the consumer:

DATAGRAM: It allows the consumer to directly request an ADU without the need of either flow or congestion control. DATAGRAM is the is unreliable and the consumer process is responsible of managing retransmissions if needed.
STREAM: It allows the consumer to retrieve a sequence of bytes using a congestion control mechanism like the ones proposed in [2, 3]. STREAM is reliable but packet losses recovery can be disabled.
CBR: It allows the consumer to request a "constant bit rate" for the delivery of a byte range. As STREAM, CBR is reliable by default but packet losses recovery can be disabled.

At the publisher only one transport protocol is available, **PUB**, which is is responsible for data encapsulation and decapsulation, and multiplexing and demultiplexing. Publisher's process simply listens for incoming requests to which it replies with the requested data, if available.

Transport layer functionalities described above naturally arise when moving from IPC to PCC. NDN also introduces additional functionalities dictated by security as signature generation (producer) and verification (consumer) are needed.

1.4 Network Layer

As for the IP network layer, packet forwarding is the main operation the NDN network layer is responsible of. However, packets are forwarded based on content names rather than IP addresses. When shifting from IP address to content names, several functionalities naturally arise, such as caching and multi-path. It follows that the network layer should implement additional operations to support such functionalities. Differently from IP, data fragmentation and reassembly are delegated to underlying layers (possibly an additional messaging layer) which provides connectivity between two NDN nodes.

2. REFERENCES

[1] Named data networking. `http://named-data.net/`.
[2] G. Carofiglio, M. Gallo, L. Muscariello, and L. Papalini. Multipath congestion control in content-centric networks. In *Proc. of IEEE INFOCOM NOMEN workshop*, 2013.
[3] G. Carofiglio, M. Gallo, L. Muscariello, M. Papalini, and S. Wang. Optimal multipath congestion control and request forwarding in information-centric networks. In *Proc. of ICNP*, 2013.
[4] D. D. Clark and D. L. Tennenhouse. Architectural considerations for a new generation of protocols. In *Proc. of ACM SIGCOMM*, 1990.
[5] I. Moiseenko and L. Zhang Anand. Consumer-producer api for named data networking. *NDN Technical Report*, 2014.
[6] E. Nordstrom, D. Shue, P. Gopalan, R. Kiefer, M. Arye, S. Ko, J. Rexford, and M. J. Freedman. Serval: An end-host stack for service-centric networking. In *Proc. of USENIX NSDI*, 2012.