Component Framework Technology for Flexible Protocol Stacks

Sam Michiels, Pierre Verbaeten

Abstract— **The context of this paper and the corresponding challenges are formed by network services and their requirements to the underlying protocol stack. The relevance of this research is confirmed by three recent trends in network services and their execution environment: growing reliance of businesses and individuals on network connectivity, highly dynamic network characteristics, and a wide range of connected client device types. This paper proposes DiPS+ (Distrinet Protocol Stack+), a sophisticated software architecture and component framework to support the development of protocol stacks that are easily customizable to application- and environment-specific requirements.**

Keywords— **Protocol stack, component framework, software architecture**

I. Protocol stacks

The main goal of a protocol stack is to enable communication between devices (also referred to as nodes) that are attached to a network. Doing so, a protocol stack hides network transfer details from services running on top of it. The protocol stack may, for instance, be responsible for delivering data correctly at the receiver, even if data can get lost during transfer (see also Figure 1).

The major tasks of a protocol stack are conceptually separated in protocol layers stacked on top of each other (which explains its name). Each layer in a protocol stack relies on the services offered by the layer underneath. Peer protocol layers can send information to each other by attaching a protocol-specific header (and/or trailer) to the data that is transferred. Each protocol may add such a header to a down going packet, resulting in a chain of headers attached to a data packet.

II. Context

The context of this paper and the challenges it addresses are formed by network services and their requirements to the underlying system software, i.c. protocol stacks. Network services include web page access on the Internet, e-mail hosting, offering the latest news facts, information retrieval, etc. Next to the Internet, other types of network environments are subject to network service requirements too: local area networks, in-home networks, or wireless ad-hoc networks, which are created dynamically when network nodes approach each other.

The relevance of this research is confirmed by three trends in network services and their execution environment. First of all, network (Internet) services have become critical and indispensable both for driving large businesses as well as for personal productivity. This growing reliance on network connectivity emphasizes the need for robust and flexible software, also at the operating system level.

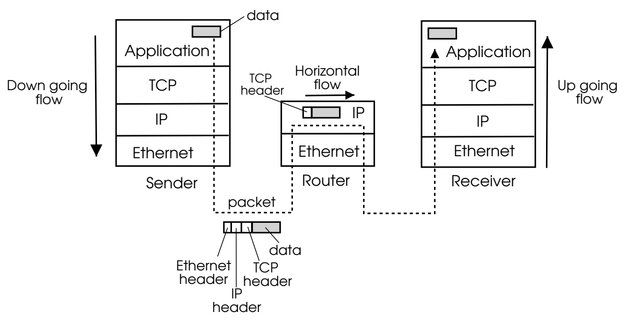

Fig. 1. Sending a TCP/IP packet via an intermediate router, which strips the Ethernet and the IP header, selects the next node to deliver the packet to (i.c. the receiver), and forwards the packet via the network and datalink layer, which both attach a new header to the packet.

Secondly, the Internet has become a highly dynamic environment with respect to the available network bandwidth, the number of parallel requests to process, the type of services required and also with respect to the service quality provided. It becomes more and more difficult to develop a single type of protocol stack that can be fine-tuned to all these dimensions of variability (e.g. by setting some parameters) [2]. Moreover, even if the system were customizable by setting parameters, network variances are often short-term, which obsoletes manual intervention. By consequence, it would be more appropriate to develop a flexible protocol stack that is able to detect various changes in the environment and adapt itself to handle them.

Thirdly, client devices on the network can be of any type as opposed to the traditional personal computer. Already a wide variety of networked client devices can be found, such as personal digital assistants (PDAs), mobile telephones, or vehicles equipped with wireless communication channels. Yet, these types of clients typically have less processing resources available than, for instance, a personal computer. By consequence, it is important to be able to customize the system – for instance by providing no more functionality than required by the services running on a particular device. In addition, the available resources should be utilized as efficient as possible, by minimizing overhead and by scheduling processing resources intelligently.

Department of Computer Science, K.U.Leuven, Leuven, Belgium. E-mail: sam.michiels@cs.kuleuven.ac.be .

These three trends – growing reliance on network connectivity, highly dynamic environment characteristics, and a wide range of client devices – reveal a common challenge for protocol stack software: offering support for flexibility at design-, development-, and execution-time, tailored to application- and/or network-specific characteristics.

III. Towards flexible protocol stacks

The need to develop flexible protocol stacks originates from two main shortcomings in traditional protocol stack designs. First of all, protocol stacks are often designed as a composition of coarse-grained blocks of code (e.g. protocol layers) with ambiguous responsibilities. Its monolithic character makes it very difficult to customize a protocol stack (e.g. by changing specific functionality inside a particular layer). Also, testing the behavior of a protocol layer is difficult if the internal code is not sub-divided into independent modules. A second major shortcoming is that the behavior of a protocol stack is often fixed at design-time and fine-tuned to handle a pre-determined load, parallelism, service type or provided service quality. Unfortunately, it is very difficult for protocol stack software to handle major deviations from the average behavior.

It is our belief that **an appropriate design method and an elegant software architecture** are required to manage the complexity inherent to protocol stack software, and to deal with dynamic network behavior. Although state-of-the-art software engineering principles (such as object- and component-oriented programming or open implementation [1]) provide guidelines to develop flexible software, all too often these guidelines are neglected during development [3]. By consequence, the resulting code is often inflexible for adaptations, which increases the risk for errors if code is adapted anyway [9]. Protocol stack developers therefore should have at their disposal an infrastructure that enforces such flexible software design.

We propose DiPS+ [4], a sophisticated software architecture and component framework to support developing flexible protocol stacks that are easily customizable to application- and environment-specific requirements. This flexibility has been achieved by enforcing a strict separation of concerns in the DiPS+ component framework. The framework distinguishes not only management from data processing, but also functionality from concurrency, and functionality from component interaction.

The DiPS+ component framework plays a central role in the protocol stack development process. It does not only offer programmers support in developing protocol stacks, it also lays the foundation for a robust and lightweight development process and a balanced software life-cycle. DiPS+ addresses three core research tracks in protocol stack development:

• **Self-management** has been addressed by providing a monitoring and management system (DMonA [5]), which is customizable by application-specific adaptability strategies. In addition, the self-management plane has been conceived as an orthogonal extension to the DiPS+ data plane as it is attached to a component's entry and exit points. In this way, DiPS+ units can be reused, whether or not DMonA is present.

• **Testing** involves testing functional correctness from day one of the software development process. Providing a powerful test framework (DiPSUnit [7], [8]) confirms the trend towards agile software engineering methodologies, which are highly iterative and presuppose change of both the design and the required functionality. This in turn requires the underlying design of production software to be open to testing.

• **Framework optimization** compensates for the overhead that is inherent to the flexibility the DiPS+ component framework offers, without necessarily throwing all flexibility overboard [6]. This sense for real-life concerns is crucial to protocol stack development.

IV. Conclusions

The strength of DiPS+ is that these three research tracks are not treated as individual solutions, but share a well-defined software architecture and component framework underneath. The DiPS+ component framework does not only offer programmers support in developing protocol stacks, but also lays the foundation for a robust and lightweight development process and a balanced software life-cycle.

Acknowledgments

We would like to thank the members of the networking task force in the DistriNet research group, and the IWT for financially supporting a major part of this research.

References

[1] G. Kiczales, J. Lamping, C. V. Lopes, C. Maeda, A. Mendhekar, and G. C. Murphy. Open implementation design guidelines. In *Proceedings of the 19th International Conference on Software Engineering (ICSE'97)*, pages 481–490, Boston, MA, USA, 1997. ACM Press, New York, NY, USA.

[2] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek. The click modular router. *ACM Transactions on Computer Systems*, 18(3):263–297, Aug. 2000.

[3] K. J. Lieberherr and I. M. Holland. Assuring good style for object-oriented programs. *IEEE Software*, 6(5):38–48, Sept. 1989.

[4] S. Michiels. *Component Framework Technology for Adaptable and Manageable Protocol Stacks.* PhD thesis, K.U.Leuven, Dept. of Computer Science, Leuven, Belgium, Nov. 2003. To be defended.

[5] S. Michiels, L. Desmet, N. Janssens, T. Mahieu, and P. Verbaeten. Self-adapting concurrency: The DMonA architecture. In D. Garlan, J. Kramer, and A. Wolf, editors, *Proceedings of the First Workshop on Self-Healing Systems (WOSS'02)*, pages 43–48, Charleston, SC, USA, 2002. ACM SIGSOFT, ACM press.

[6] S. Michiels, L. Desmet, N. Janssens, T. Mahieu, and P. Verbaeten. Dips framework optimization. Technical report, K.U.Leuven, Dept. of Computer Science, Leuven, Belgium, Jan. 2003.

[7] S. Michiels, D. Walravens, N. Janssens, and P. Verbaeten. DiPS: Filling the Gap between System Software and Testing.

[8] S. Michiels, D. Walravens, N. Janssens, and P. Verbaeten. DiPSUnit: A JUnit Extension for the DiPS Framework. In *Proceedings of Third International Conference on eXtreme Programming and Agile Processes in Software Engineering (XP2002)*, Alghero, Italy, May 2002.

[9] Y. Yokote. The apertos reflective operating system: the concept and its implementation. In *Conference Proceedings on Object-oriented programming systems, languages, and applications (OOPSLA)*, pages 414–434. ACM Press, New York, NY, USA, 1992.