

The Language Network Code—A Brief Overview

Sebastian Fischmeister
Department of Electrical and Computer Engineering
University of Waterloo
Waterloo, Canada
sfischme@uwaterloo.ca

1 History

The general goal of the work on Network Code is to allow building adaptive and verifiable distributed real-time systems. The language has been originally drafted in 2004 [4, 8] and has been subsequently improved as the work progressed from a pure software implementation in the kernel driver [6] to a hardware solution [5].

2 The Language

Network Code is a domain-specific language for programming communication schedules and arbitration mechanisms for real-time communication. Network Code programs of a certain structure remain verifiable [6], analyzeable [1], and composable [2]. Furthermore, Network Code and its runtime can be seen as a programmable communication layer [7] and can mimic concepts from other protocols by incorporating communication-round segments that use tokens, round-robin, or priorities. As such it allows developers to rapidly program dynamic communication behavior tailored to the application needs without compromising safety.

The Network Code language consists of nine instructions which control timing, data flow, control flow, and error handling. These instructions can describe arbitrary TDMA schedules [4]. Each node executes different programs, because the actions of the sender differ from the actions of a receiver for each slot. For example a single slot is characterized by: the sender, the slot length, and the set of receivers. The sender’s program creates the message and then transmits it. The receiver’s program receives the message and stores it in a buffer to be used by the application. The slot size can vary each time and it is defined by the time between the sender starting the transmission (executing `send()`) and the receiver processing message (executing `receive()`).

In the following, we provide two brief examples to demonstrate how Network Code works. Most of the instructions and parameters are intuitive, and parameters, which are unimportant for this introduction, are masked with the symbol ‘.’. For detailed descriptions, we direct the interested reader to the language report [3].

As an example for virtual circuit-switched communication consider the following programs. Note that for sake of simplicity, we assume that both nodes start at the same time and there is no clock skew; also `wait()` is a composite instruction used for instructive purposes.

<i>Sender:</i> 0 L0: create (msg_a, A) send (1, msg_a, .) 2 future (10, L0) halt ()	<i>Receiver:</i> 0 wait (9) L1: receive (1, A) 2 future (10, L1) halt ()
---	--

The sender first creates a packet from variable *A* using the alias `msg_a`. Then, it sends the message on channel 1, and sets up an alarm in ten time units to continue at label L0. It then halts execution (the `halt()` instruction) and waits for the alarm to resume operation. The receiver first waits nine time units for the first delivery of a message and then receives a this from channel 1 into the local variable *A* every ten time units.

As an example for packet-oriented communication, consider the following programs using the same assumptions as before:

<hr/> <p><i>Node 1:</i></p> <pre> 0 L0: mode(soft) wait(4) 2 mode(hard) future(11, L0) 4 halt()</pre> <hr/>	<hr/> <p><i>Node 2:</i></p> <pre> 0 L1: wait(5) mode(soft) 2 wait(4) mode(hard) 4 future(1, L1) halt()</pre> <hr/>
---	--

The instruction `mode()` controls the mode of operation of the run-time system. In the soft mode, the system offers best-effort communication, in the hard mode it provides guaranteed communication, and the init mode is used for setting up the system. The system guards access to the network through temporal isolation. Node 1 gets exclusive access to the medium during the first four time units, and Node 2 for time five to nine. While they have exclusive access, both nodes transmit messages from a general output queue. Messages are automatically received through the transceiver and best-effort-traffic messages are logically separated from guaranteed-traffic messages.

Note that Network Code also supports raw communication. In the previous example, only one node was in the soft mode at a time. If several nodes are in the soft mode, all of them might concurrently access the network.

3 Future Work & References

Network Code has been applied in a Medical Device Plug&Play case study demoed at HIMSS’08 in Orlando. Ongoing work on the application side is to port the existing FPGA core to the NetFPGA platform and provide tool support for this new platform. Network Code also has been successfully generated from a high-level specification [8]. Ongoing work in this direction aims to generate programs for particular domains such as networked control systems.

4 Acknowledgements

Several people actively participated in the research, development, and test of Network Code and its formal representation in tree schedule. The alphabetical list includes: Madhukar Anand, Gregor König, Insup Lee, Kevin Perry, Oleg Sokolsky, and Robert Trausmuth.

References

- [1] M. Anand, S. Fischmeister, and I. Lee. An Analysis Framework for Network-Code Programs. In *Proc. of the 6th Annual ACM Conference on Embedded Software (EmSoft)*, pages 122–131, Seoul, South Korea, October 2006.
- [2] M. Anand, S. Fischmeister, and I. Lee. Composition Techniques for Tree Communication Schedules. In *Proc. of the 19th Euromicro Conference on Real-Time Systems (ECRTS)*, pages 235–246, Pisa, Italy, July 2007.
- [3] S. Fischmeister et al. Network Code Language Specification. Technical report, University of Pennsylvania, 2007. Manual & specification.
- [4] S. Fischmeister. Multi-Dimensional Schedules for Media-Access Control in Time-Triggered Communication. In *Proc. of the IEEE Symposium on Computers and Communications (ISCC)*, pages 664–670, La Manga del Mar Menor, Spain, June 2005. IEEE Press.
- [5] S. Fischmeister, I. Lee, and R. Trausmuth. Hardware Acceleration for Verifiable, Adaptive Real-Time Communication, 2008. submitted to ETFA.
- [6] S. Fischmeister, O. Sokolsky, and I. Lee. A Verifiable Language for Programming Communication Schedules. *IEEE Transactions on Computers*, 56(11):1505–1519, November 2007.
- [7] S. Fischmeister and R. Trausmuth. A Programmable Arbitration Layer For Adaptive Real-Time Systems. In *Proc. of the Intl. Workshop on Adaptive and Reconfigurable Embedded Systems (APRES)*, pages 27–31, 2008.
- [8] Gregor König. Using Interpreters for Scheduling Network Communication in Distributed Real-Time Systems. Master’s thesis, Salzburg University, Jakob-Haringer-Str. 2, 5020 Salzburg, Austria, March 2005.