Do Formal Methodists have Bell-Shaped Heads?

(Invited Paper)

Timothy G. Griffin
Computer Laboratory
University of Cambridge
Timothy.Griffin@cl.cam.ac.uk

Abstract

Data networking has not been kind to formal methods. The Internet's birth gave rise to an intense culture war between the bell-heads and the net-heads, which the net-heads have largely won. In this area formal methodists have long been seen as the humourless enforcers of the defeated bell-heads. The result: formal methods are not a part of the mainstream of data networking and are largely relegated to the the thankless task of reverse engineering (security-related protocols are perhaps the rare exception). If we want to move beyond this situation we must build tools that enhance the ability to engage in the Internet culture — tools that encourage community-based development of open-source systems and embrace the open-ended exploration of design spaces that are only partially understood.

1 State the Problem Before Describing the Solution?

The title of this section, sans question mark, is the same as a one-page put-down of the culture of computer networking written by Leslie Lamport in 1978 [7]. The idea is simple and seductive. Start with a specification of the problem that is independent of any solution! Then describe your solution and show that it solves the problem. What could be more simple? Motherhood and apple pie when it comes to technologies like fly-by-wire, software on deep space probes, control systems for nuclear power plants, and so on.

But did this make sense in 1978 for the early pioneers of the Internet? Does it make sense today now that the Internet has grown from lab experiment to ubiquitous infrastructure? I don't think so. The reasons are obvious. The traditional (formal) approach to protocol development is to start with a specification [5]. But we can only specify a problem when we understand what we are doing. Part of the thrill of systems-related networking in the past decades has been in exploring virgin territory opened up by Moore's law and related exponential curves in bandwidth and memory. In addition, the exploration has not been conducted in a top-down manner, but rather in a grass-roots bottom-up way. And the active community has never agreed on exactly where it was going then or now.

When I was hired at Bell Laboratories in 1991 research in Internet-related technologies was essentially banned. The networking professionals (bell-heads) knew that circuits where the only realistic technology and that the "junk" from EE and operating systems (net-heads) would never work. Bell Labs didn't change this policy until about 1997, by which time it was too late (see Day's interesting book [4]). Sadly, the situation may be a bit worse in Europe than elsewhere. Many EU countries set up national telecom institutes during the 1970's, which had the unfortunate consequence of isolating communications engineers from computer scientists. Elsewhere communications has been absorbed into computer science!

In networking, formal methods has historically been associated with the losing side of these culture wars.

2 Satisfied with Reverse-Engineering?

If Internet protocols are not developed from formal specifications it may still be worthwhile to *reverse engineer* existing protocols. I've done a lot of reverse engineering with routing protocols. I think it can be very fruitful, especially if we can come away with general principles that have applicability beyond protocol-specific details. This last point is very important — your results must be interesting to a larger community because you can't expect the networking community to care. By the time you have figured something out, they will have moved on to a new set of problems.

3 Change of Thinking?

Given the cultural constraints, perhaps reverse engineering is the best we can do. Attempts have been made to lower the cost of formal methods, see the work on *lightweight formal methods* [1, 6]. I think that it is a very worthwhile effort, but one that is useful in *all* areas where formal methods are applied. Is there an approach that fully embraces the community-based open-ended nature of Internet-related systems work?

I'll suggest one answer — domain-specific languages (DSLs). It seems to me that DSLs allow us to raise the level of abstraction in which problems are solved. I will discuss only three examples — the Statecall Policy Language (SPL) [8], Ynot [3], and Idris [2].

I've had fun discussing this topic with Anil Madhavapeddy, Jon Crowcroft, and Boon Thau Loo. I hope this short talk will stimulate further discussion at the ATE workshop.

References

[1] S. Agerholm and P. G. Larsen. A lightweight approach to formal methods. *Proceedings of the International Workshop on Current Trends in Applied Formal Methods*, 1998.

[2] E. Brady. Idris - systems programming meets full dependent types. In *PLPV 2011*, 2011.

[3] A. Chlipala, G. Malecha, G. Morrisett, A. Shinnar, and R. Wisnesky. Effective interactive proofs for higher-order imperative programs. In *Proceedings of ICFP'09*, 2009.

[4] J. Day. *Patterns in Network Architectures : A return to fundamentals.* Prentice Hall, 2008.

[5] M. G. Gouda. *Elements of Network Protocol Design.* Wiley, 1998.

[6] D.Jackson. Alloy: A lightweight object modelling notation. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 11(2):256–290, 2002.

[7] L. Lamport. State the problem before describing the solution. *ACM SIGSOFT Software Engineering Notes*, 3(1):26, 1978.

[8] A. Madhavapeddy. Combining static model checking with dynamic enforcement using the statecall policy language. In *International Conference on Formal Engineering Methods (ICFEM)*, 2009.