

---

# Commentaries on "Active Networking and End-To-End Arguments"

In 1984, Saltzer, Reed, and Clark identified "end-to-end arguments," a set of design principles related to the organization and placement of functions within a system. Among other applications, these general principles have been helpful in the design of modern layered protocols. In the context of active networks, however, which take the non-traditional view of a programmable network infrastructure, the interpretation and application of end-to-end arguments is an open question.

This article is a collection of short commentaries by recognized networking experts offering their perspectives on the relation between active networks and "end-to-end arguments." We hope their insights will serve as a stimulus for thoughtful debate.

The first commentary, by Bhattacharjee, Calvert, and Zegura, frames the question and makes a case arguing that active networking does not conflict with end-to-end argument principles. The authors are leading a research group at Georgia Institute of Technology working on the CANEs (Composable Active Network Elements) project, which focuses on service composition and applications for active networking. The authors also participate in the DARPA active network com-

munity's efforts to define network/node and service composition architectures.

The second commentary, by Partridge, Strayer, Schwartz, and Jackson, claims that end-to-end arguments discourage active networking in the internet layer, but encourage active networking at all other layers except perhaps the transport layer. The BBN Technologies (part of GTE Corp.) team recently completed a DARPA project called Smart Packets, where concisely encoded active packets are used for network management and diagnostics in IP. Readers may know Craig Partridge as the former editor-in-chief of *IEEE Network*, and Tim Strayer as the program chair for the 1998 IEEE Local Computer Networks Conference.

The prerogative of the "last word" is deferred to Reed, Saltzer, and Clark, who reexamine their original end-to-end arguments specifically within the context of active networks. The authors, affiliated with Techburst and the MIT Laboratory for Computer Science, are distinguished experts with extensive records of research accomplishments in computer design and computer networks.

*Thomas M. Chen and Alden W. Jackson  
Special Issue Guest Editors*

---

## **Samrat Bhattacharjee, Kenneth L. Calvert, and Ellen W. Zegura** Georgia Institute of Technology

Discussions of the implementation of various functions in a communication network often include appeals to some form of "end-to-end argument." What is typically called "the end-to-end argument" is actually a set of architectural principles that guide the placement of functions within a distributed system [1]. Such principles are often construed to preclude the implementation of any kind of higher-level function within a network. As such, it might seem that active networks are the antithesis of end-to-end arguments. We claim, however, that the "activation" of networks is a natural extension of these well-accepted design principles [5].

For the purposes of this discussion, "active networking" refers to the placement of user-controllable computing and other resources in the communication network, where they can be utilized by applications that need those capabilities [2]. An active network supports a user-network interface allowing the nodes of the network to be *programmed* by the application (user) to provide a desired functionality, such as routing. The level of programmability might range from a Turing-complete programming language to a set of predefined, user-selectable functions whose behavior can be controlled through parameters. The important point is that with active networks, *the network service can be tailored to the user's requirements*. It is this aspect of

active networks that relates to end-to-end arguments in general.

An end-to-end argument "provides a rationale for moving a function upward in a layered system closer to the application that uses the function" [1]. According to this principle, a computer network, as part of the "lower layers" of a distributed system, should avoid attempting to provide functions that can be better implemented in the end systems, especially if some applications might not benefit from such functions at all. The canonical example of such a function is reliable transfer. The network can go to great lengths to protect against and recover from losses in the network, but an application that requires reliability will generally have to protect against other sources of error; in that case the network's efforts are redundant. On the other hand, applications that do not need reliability would still have to pay for it (e.g., through reduced performance).

We identify the following principles as the key end-to-end arguments applying to the placement of functionality in networks:

- Some services require the knowledge and help of the end-system-resident application or user to implement, and thus *cannot* be implemented entirely within the network.
- If not all applications will make use of a service, it should be implemented in such a way that only those applications using it have to pay the price of supporting it in the network.
- The amount of support for any given end-to-end service in the network is an engineering trade-off between the performance seen by the application and the cost of implementing the support. We claim that these principles do not rule out support for higher-level functionality within the net-

work. Rather, they require that the interface to such functionality be carefully designed; that costs and benefits of such support be calculated; and that the engineering trade-off be carefully evaluated.

A fundamental premise of our argument is the following: *Some services can best be supported or enhanced using information that is only available inside the network.* In other words, the network may have information not available to the application, and the timely use of that information can significantly enhance the service seen by the application. Examples of information that is first (or only) available to the nodes of the network include:

- The time and place where congestion occurs [3]
  - Global patterns of access to objects retrieved over the network (e.g., Web pages); in particular, the location of "hot spots," or points in the network where requests for objects are highly correlated in time and space [4]
  - The location of packet losses within multicast distribution trees
- On the other hand, applications may have information that is needed by the network in order to fully optimize performance. Examples of this type of information include:
- The existence of dependencies among application data units (e.g., some are useless if others are not received)
  - Variations in importance of data units, including whether to retransmit if lost
  - Whether or not it is acceptable to service a request using cached data

Thus, to optimize performance it is desirable to combine application and network information.

Classically, an end-to-end argument views the network as a monolithic entity that provides a single type or quality of service to all users. For example, the debate about reliable service assumed the network would support a single paradigm for all users: either reliable or best-effort transport. Active networks allow users to increase the likelihood that the service offered by the network will be useful to them, by providing an interface which supports multiple (or programmable) services.

There are costs associated with such a flexible interface, and they affect all of the network's users whether they take advantage of active network support or not. The (monetary) cost of *providing* the interface, although likely to be significant, is paid once and can be amortized over all users for a period of time. The performance cost of *using* the interface should vary with the application; this is the essence of a good end-to-end argument. For example, applications that need only a vanilla best-effort datagram delivery service should not

suffer reduced performance because of the increased flexibility of the interface. On the other hand, any performance penalty for customizing network behavior (e.g., signaling overhead, or taking packets off the "fast path") must be more than offset by improved end-to-end performance delivered to the ultimate users. These performance costs will be determined ultimately by the primitives and composition mechanisms provided by the active network architecture.

Active networks provide end-to-end system designers with more choices for function placement. Successful system design still requires making correct choices, which depend on the details of the particular problem being solved. The key task in active network design, then, is to identify the cases in which the performance gains and enhanced capabilities justify the cost incurred in deploying and using an active networking architecture.

### References

- [1] H. Saltzer, D. Reed and D. Clark, "End-to-end arguments in system design," *ACM Trans. Comp. Sys.*, vol. 2, no. 4, 1984.
- [2] D. Tennenhouse *et al.*, "A survey of active network research," *IEEE Commun. Mag.*, vol. 35, no. 1, 1997.
- [3] S. Bhattacharjee, K. Calvert and E. Zegura, "An architecture for active networking," *Proc. High Perf. Networking '97*, 1997.
- [4] S. Bhattacharjee, K. Calvert, and E. Zegura, "Self-organizing wide-area network caches," *IEEE INFOCOM '98*, 1998.
- [5] S. Bhattacharjee, K. Calvert, and E. Zegura, "Active networking and the end-to-end argument," *Proc. Int'l. Conf. Network Protocols*, 1997.

### Biographies

SAMRAT BHATTACHARJEE (bobby@cc.gatech.edu) received B.S. degrees in mathematics and computer science from Georgia College in 1994. He is currently a Ph.D. candidate in the Networking and Telecommunications Group in the College of Computing, Georgia Institute of Technology. His research interests are in networking, distributed systems, and operating systems.

KENNETH L. CALVERT [M] (calvert@cc.gatech.edu) is a senior research scientist in the College of Computing at Georgia Tech. His research deals with the design and implementation of communication protocols and services, with an emphasis on reducing the cost of deploying new application-specific services. Current interests include active networks and development of an upper-layer architecture for the Internet. He holds a Ph.D. from the University of Texas at Austin. He is a member of ACM.

ELLEN W. ZEGURA [M] (ewz@cc.gatech.edu) received a B.S. degree in computer science and electrical engineering (1987), an M.S. degree in computer science (1990), and a D.Sc. degree in computer science (1993), all from Washington University, St. Louis, Missouri. She is currently an assistant professor in the College of Computing at Georgia Institute of Technology. Her research interests are in wide-area internet networking with emphasis on interaction between applications and network services. Current work includes active networking, application-aware network services, and anycasting. She is a member of ACM.

---

**Craig Partridge, Tim Strayer,  
Beverly Schwartz, and Alden W. Jackson**  
BBN Technologies

Active networking is a new communications paradigm for data networks. While some uses of active networking have shown promise, the general utility of active networking is still (widely) questioned. In this essay, we seek to evaluate active networking in the context of the end-to-end argument. In particular, in what parts of the networking stack does the end-to-end argument tell us that active networking is useful, and where does the end-to-end argument say that active networking is a bad idea? We argue that active networking could have a place in every network layer — except the internet layer — where the end-to-end argument strongly suggests active networking is a bad idea.

The end-to-end argument [1] is a system design principle

intended to help determine where to place services in a subsystem. The argument states that a function or service should be implemented in a subsystem only if the service can be completely implemented in that subsystem or if, by partially implementing the service in the subsystem, the subsystem substantially improves the performance of other parts of the system. The classic example of such a trade-off is the use of checksums for data transfer. If an application is to ensure that a data transfer successfully took place, it must checksum the delivered data at both ends and compare the result. However, we can rely on a less expensive checksum if lower layers, especially the media access layer, do a cyclic redundancy check (CRC) to protect against layer-specific errors.

The end-to-end argument has often been generalized to a layered environment: a communications layer should not try to offer services that only a higher layer can implement completely, unless the incompletely offered service is a clearly useful performance enhancement.

Active networking is a research program that seeks to place a program in a packet and then have that program executed at some or all of the network nodes the packet traverses [2, 3]. The exact function of the program varies depending on which active networking researcher one talks to. For example, the program may be run at each node to determine how the packet should be routed; or the program may be used to augment the packet service, for example, by setting up a special forwarding service that subsequent packets may use.

Both practices can be used to implement the same service. For instance, consider flow setup. Each packet in the flow could carry a program which implements the flow's forwarding choices for its packet; or the first packet in the flow could deposit a program in each router in the flow's path, and subsequent packets could simply call the program to be forwarded according to the flow's choices.

Our evaluation is done in pieces. We separate the data processing stack (the five layers of the Internet stack on the left of Table 1) from the management subsystem and examine each in turn, asking the following two questions:

- Does adding a programmable environment at layer  $N$  (or to the management subsystem) enhance performance of the layers above  $N$  (or the management subsystem and the layers it manages)?
- Does adding a programmable environment at layer  $N$  allow us to eliminate the need to support functions at layers above  $N$  (i.e., taking the end-to-end argument in reverse, can we show the function is now implementable without the higher layers' involvement)?

The easiest layer is the application layer. Active networking has been a feature of the application layer for several years. The most notable example is Java, where an applet is exported to a remote machine which runs the applet until it is completed or terminated by the user. But other work, such as CORBA, Linda, and Remote Evaluation, has also demonstrated the tremendous value of integrating programs with application data at the application layer. Fundamentally, the triumph of moving programs is to move the program to its distributed data, rather than bring the data to the program. In response to the two thought questions, we can comfortably say that active networking enhances performance for higher layers (in this case, the user), and may relieve the user of some burdens such as trying to figure out which system supports the graphics routines needed to display their data (since we can now export those routines to the user's system).

The transport layer is a more confusing case. The transport layer would clearly benefit from programmability. Programmers often bemoan the limited set of communications paradigms that current transport protocols offer. And there is a design theory, known as Application Layer Framing (ALF) [4], which argues that applications are better placed than transport protocols to determine how their data should be packetized and transmitted over the network. So active networking could allow the transport layer to enhance the performance of the application layer.

However, as we better understand the problems of sharing a network's capacity fairly among transport protocol users, it has become clear that we need to constrain transport protocol to certain behavioral norms [5-7]. In particular, transport protocols' reactions to congestion need to meet certain constraints. How to impose behavioral norms on a program is an open question, and until it is solved, adding programmability

User	Operator
Application	Management
Transport	
Internet	Subsystem
Subnetwork	
Data link	

■ Table 1. *The Internet stack.*

to the transport protocol potentially violates the end-to-end argument. In particular, while a programmable transport layer could enhance higher-layer performance, it could also require new functionality at higher and lower layers to ensure that buggy or malicious transport-layer programs do not violate transmission rules. In short, while we answered yes to our first thought question, we must ruefully answer no to the second.

The most serious conflict between the end-to-end argument and active networking comes in the internet layer. The purpose of the internet layer is to achieve universal connectivity and communication between an arbitrary number of heterogeneous devices. The difficulty is that it is hard to find a way that active networking could enhance this service, while easy to find ways that active networking harms this service.

When a packet's path is affected by some code carried in the packet (or, worse, in someone else's packet), the chances that the packet will reach the destination is reduced considerably. Programs are buggy, and there is still no effective way to prove the correctness of a nontrivial program. When the delivery of a packet depends on code execution at each node in the route, the packet is at the mercy of poorly implemented, damaged, or out-of-version execution environments. Since the code can use any of thousands of variables as conditionals in determining a delivery path, every communications path is therefore unique. If your packet's program doesn't work, only you have the necessary knowledge to debug your mix of programs and data to figure out why. At the same time, because the range of actions a packet can take has been increased, the damage that a buggy program — say, one that copies packets at each node — can inflict has been magnified and made harder for network operators to stop.

Another way to think about the internet layer is that active networking at the internet layer adds a great deal of complexity to a very simple process. Van Jacobson once described the process of forwarding a packet as "Very simple. A router has only three choices when presented with a packet. It can transmit the packet. It can delay (queue) the packet. Or it can throw the packet away" [8]. All active networking can do is increase our flexibility to choose among these options, and increase the risk that we choose incorrectly.

Oddly enough, as we go further down the networking stack — to the subnet and link layers — active networking looks like a winner again. For example, suppose when a modem dialed into a modem bank, the first step was to download the code to run the appropriate signaling protocol. No need to figure out if the modem and modem bank both run the same type of 56 kb/s signaling; the modem would automatically run whichever type the modem bank used. The modem industry already operates in this fashion, albeit more crudely, by shipping code to update modems to new standards.

One can envisage even more sophisticated systems in which an all-purpose PCMCIA card with a four-wire connector can plug in, download the appropriate medium access control (MAC) layer protocol, such as 100 Mb/s Ethernet (either BaseT or AnyLAN), a modem protocol, or FDDI, and proceed to transmit and receive packets. The interface could effectively shield higher software from worrying about what network it is running over and eliminate the need for much configuration software. (Note that putting code in every packet doesn't work well here — we do not want devices booting a new MAC layer to receive each packet.) Looking to our two questions, adding programmability to the subnet and link layers has the potential to both enhance performance (reducing

the user's need to update hardware) and eliminate some higher-layer functions.

Auxiliary to the network stack is the network management system. Here programmability is also a tremendous win. In today's network management systems, management centers frequently poll devices for data, and then examine the data to detect problems. This practice is very inefficient; in most cases, the polled data indicates there are no problems — indeed, usually not even a change in conditions — and we have used valuable network bandwidth to learn that nothing has changed. A better approach is to occasionally send a program to each device, and only have it return information about changes. An even better approach might be to distribute a program to each device that watches for certain misbehaviors, and reports back if it sees them. Today's approach to network management was chosen because of fears in the late 1980s that programmable systems would be too complex. Now that we better understand programmable systems, there seems no good reason not to make network management programmable. Programmability enhances performance and allows us to eliminate needless work by sending the program to the data it needs to observe.

One of the interesting results of this exercise is that it indicates yet again that the internet layer is special. All other layers could benefit, at least somewhat, from putting programs in packets. But the simplicity that ensures interconnectivity, the fundamental feature of the internet layer, resists the complexity programmability brings.

#### References

- [1] J. H. Saltzer, D. P. Reed, and D. D. Clark, "End-to-end arguments in system design," *ACM Trans. Comp. Sys.*, vol. 2, no. 4, Nov. 1984.

- [2] D. L. Tennenhouse and D. J. Wetherall, "Towards an active network architecture," *ACM Comp. Commun. Rev.*, vol. 26, no. 2, Apr. 1996.
- [3] D. L. Tennenhouse et al., "A survey of active network Research," *IEEE Commun. Mag.*, vol. 35, no. 1, Jan 1997, pp. 80–86.
- [4] D. D. Clark and D. L. Tennenhouse, "Architectural considerations for a new generation of protocols," *Proc. ACM SIGCOMM '90*, Sept. 1990, pp. 200–8.
- [5] V. Jacobson, "Congestion avoidance and control," *Proc. ACM SIGCOMM '88*, Aug. 1988, pp. 314–329.
- [6] R. Jain and K. K. Ramakrishnan, "A Binary Feedback Scheme for Congestion Avoidance in Computer Networks with a Connectionless Network Layer," *Proc. ACM SIGCOMM '88*, Aug. 1988, pp. 303–13.
- [7] S. Floyd and V. Jacobson, "Random Early Detection Gateways for Congestion Avoidance," *IEEE/ACM Trans. Networking*, vol. 1, no. 4, Aug. 1998, pp. 397–413.
- [8] V. Jacobson, personal communication, c. 1990.

#### Biographies

CRAIG PARTRIDGE [SM] (craig@bbn.com) is a chief scientist at BBN Technologies, a part of GTE Corporation, where he does research on gigabit and terabit networking technologies. He is the former Editor-in-Chief of *IEEE Network* and *ACM Computer Communication Review*. He is also a part-time professor at Stanford University and received his Ph.D. from Harvard University.

TIM STRAYER [M] (strayer@bbn.com) is a senior scientist at BBN Technologies, a part of GTE Corporation, where he does research on next-generation network protocols, active networks, routing, mobility, and satellite communication. He has written over 25 journal and conference papers and several book chapters, and co-authored the 1992 Addison-Wesley book *The Xpress Transfer Protocol*.

BEVERLY SCHWARTZ (bschwartz@bbn.com) is an engineer at BBN Technologies, a part of GTE Corporation, where she works on applying Active Networking technology to network management. She has an M.Sc. in computer science from Harvard and a B.S.E.E. from Tufts.

ALDEN W. JACKSON [M] (awjacks@bbn.com) is a senior scientist at BBN Technologies, a part of GTE Corporation, where he does research on active networking, network management, and network security technologies. He received his Ph.D. degree in electrical engineering from the University of Delaware. He is on the technical editorial board of *IEEE Network*.

---

### David P. Reed, TechBurst Jerome H. Saltzer and David D. Clark, MIT Laboratory for Computer Science

Some 20 years have elapsed since we identified and named end-to-end arguments [1], a class of system design principles that organize and guide the placement of function within a system. These arguments and the underlying principles have now been invoked in many contexts, becoming part of the vocabulary of network protocol and operating system designers. Like other general design principles, end-to-end arguments impose a structure on the design space, rather than solving the design problem. This structure provides a basis for discussion and analysis of trade-offs, and suggests a strong rationale to justify design choices.

This note comments on a current design controversy that can be framed partly in terms of end-to-end arguments. One form of active networking [2], a novel category of communication network architectures, comprises attaching programs to data packets, with the intent that those programs be executed at points within the network. The purpose is to better match the behavior of the network to the requirements of the application. The question being raised is whether or not this idea directly violates an end-to-end argument.

On one hand, programmability appears to contradict the end-to-end principle that a function or service should be carried out within a network layer only if it is needed by all clients of that layer, and it can be completely implemented in that layer.<sup>1</sup> On the other hand, programmability may allow a network client to implement precisely the service it needs, an outcome that is consonant with end-to-end arguments. Contradiction and consonance aside, because programmability enables such a wide range

of possibilities, applying end-to-end arguments in a general, yet definitive, way may be impossible. Instead, the specifics of each particular active networking idea would benefit from evaluation in light of the end-to-end principle.

#### *Programmability's Effect on Design-Time Function Placement*

End-to-end arguments address design more than implementation and implementation more than execution; that is, they suggest who should provide the code, not on which box it should run. Moving functions and services upward in a layered design, closer to the application(s) that use them, increases the flexibility and autonomy of the application designer to apply those functions and services to the specific needs of the application. With that view, programmability in a lower layer can be seen as a means to defer design choices upward in the layering, closer to the application, and later in time, even

---

<sup>1</sup> There are some situations where applying an end-to-end argument is counterproductive. One category is cost-related. For example, many-to-many communications can be done by forwarding packets among multi-cast servers located outside the network, but it appears to be much more effective to add some support at the lowest layers of the network. Another category is where for some reason the endpoints are not in a position to cooperate. For example, version 5 of the Kerberos authentication system pulls inside the application programming interface the function of replay prevention, because experience with version 4 showed that few application programmers understood how to do that critical function correctly. The interesting thing is how few examples have turned up in 20 years of experience with systems like the Internet.

though the resulting functions may actually take place deep inside the network.

At the same time, one can raise a concern that a general programming interface can lead to complex and unpredictable interactions among independently designed applications and independently acting users. Part of the context of an end-to-end argument is the idea that a lower layer of a system should support the widest possible variety of services and functions, to permit applications that cannot be anticipated. That is, minimize the lower-layer function, get out of the way, and let the higher layer do its thing. But this flexibility actually implies that end-to-end arguments have not one, but two complementary goals:

- Higher-level layers, more specific to an application, are free (and thus expected) to organize lower-level network resources to achieve application-specific design goals efficiently (application autonomy).
- Lower-level layers, which support many independent applications, should provide only resources of broad utility across applications, while providing to applications a usable means for effective sharing of resources and resolution of resource conflicts (network transparency).

While making lower layers more active or programmable is likely to enhance application autonomy, the risk is that programmable lower layers may reduce network transparency. The reason is that a key element of transparency is some ability to predict how the network will behave.

Since lower-level network resources are shared among many different users with different applications, the complexity of potential interactions among independent users rises with the complexity of the behaviors that the users or applications can request. For example, when the lower layer offers a simple store-and-forward packet transport service, interactions take the form of end-to-end delay that can be modeled by relatively straightforward queuing models. Adding priority mechanisms (to limit the impact of congestion) that are fixed at design time adds modest complexity to models that predict the behavior of the system. But relatively simple programming capabilities, such as allowing packets to change their priority dynamically within the network, may create behaviors that are intractable to model, in the same way that the simple rules of cellular automata such as Conway's Game of Life[3] can lead to remarkably complex behavior.

To maintain the largest degree of network transparency, then, the end-to-end principle requires that the semantics of any active features be carefully constrained so that interactions among different users of a shared lower level can be predicted by a designer who is using the services and functions of that active layer. Lack of predictability thus becomes a cost for all users, including those that do not use the programmability features. Getting the semantics of active enhancements right is a major challenge, and wrong active enhancements are likely to be worse than none at all, since everyone helps pay the cost of something that is used by only a few but reduces transparency for everyone else.

Thus, even though active network ideas are not ruled out by end-to-end arguments, we have not seen practical high-impact examples of a sufficiently simple, flexible, and transparent programming semantics suitable for use in lower levels of networks. Until such examples are developed in detail, the existence of active networks that meet the end-to-end criteria should perhaps be classified as theoretical.

### *Keep It Simple, Stupid*

End-to-end arguments arose from our work on secure operating system kernels in the Multics project [4, 5], and our work

on end-to-end transport protocols in LANs and the Internet experiment [6]. Similar thinking by John Cocke and his colleagues on the role of compilers in simplifying processor architecture led to the RISC approach [7] to processor architecture, which also suggests moving function from lower layers to more application-specific layers. In the past 20 years, systems designers have only begun to explore and demonstrate the profound implications of this kind of architectural approach on design in large-scale systems.

An end-to-end argument is similar to the argument for RISC: it serves to remind us that building complex function into a network implicitly optimizes the network for one set of uses while substantially increasing the cost of a set of potentially valuable uses that may be unknown or unpredictable at design time. A case in point: had the original Internet design been optimized for telephony-style virtual circuits (as were its contemporaries SNA and TYMNET), it would never have enabled the experimentation that led to protocols that could support the World Wide Web, or the flexible interconnect that has led to the flowering of a million independent Internet service providers (ISPs). Preserving low-cost options to innovate outside the network, while keeping the core network services and functions simple and cheap, has been shown to have very substantial value. In this way, an end-to-end argument does not oppose active networks per se, but instead strongly suggests that enthusiasm for the benefits of optimizing current application needs by making the network more complex may be misplaced.

This idea of "stupid operating systems," "stupid networks," and "stupid processors" hasn't yet had its full run.<sup>2</sup> The telephone company still seems to think that all users want the illusion of a copper pair from the user's house to some ISP point of presence in another city. In its Internet access approach, the cable company places the real network one step closer, but neither architecture is really prepared for the household with three computers and two network access providers. Politicians want both the cable company and the ISP to filter packets for things children shouldn't see, and the FBI asks them to make copies of specific data streams to simplify wiretapping (these may be examples of noncooperating endpoints). Political arguments aside, even if one accepts these requirements, the corresponding implementation proposals are sometimes stunning in the way they fail to consider end-to-end arguments.

### *Take It Case by Case*

It is important to keep in mind that end-to-end arguments are one of several important organizing principles for systems design. While there will be situations where other principles or goals have greater weight, an end-to-end argument can facilitate the design conversation that leads to a more flexible and scalable architecture. We suggest that, as researchers investigate various ideas that are lumped into the research area called active networking, it is important to consider and understand the specific applications of end-to-end arguments to the designs under consideration. Those active interfaces that survive close inspection under this light should benefit from such analysis.

---

<sup>2</sup> David Isenberg of AT&T has recently reinvented end-to-end arguments in an entertaining paper, "The Rise of the Stupid Network," available at <http://www.computer-telephony.com/ct/att.html>, which criticizes the telephone industry's concept of "intelligent network" based on ideas very similar to those in the original end-to-end arguments paper.

---

## References

- [1] J. H. Saltzer, D. P. Reed, and D. D. Clark, "End-to-end arguments in system design," *ACM Trans. Comp. Sys.*, vol. 2, no. 4, Nov. 1984, pp. 277-88. An earlier version appeared in *2nd Int'l. Conf. Dist. Comp. Sys.*, Apr., 1981, pp. 509-12.
- [2] D. L. Tennenhouse *et al.*, "A survey of active network research," *IEEE Commun. Mag.*, vol. 35, no. 1, Jan. 1997, pp. 80-86.
- [3] E. R. Berlekamp, J. H. Conway, and R. K. Guy, *Winning Ways for your Mathematical Plays, Vol. 2: Games in Particular*, Academic, 1982, pp. 817-49.
- [4] M. D. Schroeder, D. D. Clark, and J. H. Saltzer, "The Multics kernel design project," *6th ACM Symp. Op. Sys. Principles*, *ACM Op. Sys. Rev.*, vol. 11, no. 5, Nov. 1977, pp. 43-56.
- [5] D. P. Reed, "Processor Multiplexing in a Layered Operating System," S.M. and E.E. thesis, MIT Dept. of EECS, 1976; available as MIT Laboratory for Computer Science tech. rep. LCS-TR-164, July 1976.
- [6] D. D. Clark, K. T. Pogran, and D. P. Reed, "An introduction to local area networks," *Proc. IEEE*, vol. 66, no. 11, Nov. 1978, pp. 1497-1516.
- [7] G. Radin, "The 801 Minicomputer," *Proc. 1st ACM Symp. Programming Languages and Op. Sys.*, in *Comp. Architecture News*, vol. 10, no. 2, Mar. 1982, pp. 39-47.

## Biographies

DAVID P. REED [M] (dpreed@reed.com) received B.S., S.M., E.E., and Ph.D. degrees from the Department of Electrical Engineering and Computer Science at the Massachusetts Institute of Technology in 1973, 1976, 1977, and 1978, respectively. He was assistant professor of computer science and engineering at MIT from 1978 to 1984, vice president/chief scientist at Software Arts, Inc. from 1983 to 1985, vice president/chief scientist at Lotus Development Corp. from 1985 to 1992, and a senior scientist at Interval Research Corp. from 1992 to 1996. He is now an independent consultant, investor, and researcher. In his early research career he made significant contributions to operating system security, the design and philosophy of the Internet protocol suite, and architectures for coordination of activities in loosely coupled, large-scale distributed systems. During his commercial career, he led the design and implementation of a number of commercial personal computer software products that support personal productivity and collaboration. His current interests include communications architectures that reach out to remote, mobile, and wearable digital devices, business economic models that explain the value of investment in high-uncertainty, long-term R&D, and systems architectures that support open, constructive collaboration among humans. He is a member of ACM and Sigma Xi.

JEROME H. SALTZER [F] (Saltzer@mit.edu) received the degrees of S.B. in 1961, S.M. in 1963, and Sc.D. in 1966, from the Massachusetts Institute of Technology, all in electrical engineering. Since 1966, he has been a faculty member of the Department of Electrical Engineering and Computer Science at MIT, where he helped formulate the undergraduate curriculum in Computer Science, and developed the core subject on the engineering of computer systems. At the MIT Laboratory for Computer Science he developed an early word-processing system, participated in the refinement of the Compatible Time-Sharing System (CTSS), and was involved in all aspects of the design and implementation of the Multiplexed Information and Computing Service (Multics). More recently, his research activities have involved the design of a token-passing ring local area network, networking of personal computers, and designing the electronic library of the future. From 1984 through 1988 he was technical director of MIT Project Athena, a system of networked engineering workstations for undergraduate education. Throughout this work, he has had a particular interest in the impact of computer systems on society, especially on privacy. In September 1995, he retired from the full-time faculty. He continues to write and teach about computer systems part-time from his MIT office. He is a member of the National Academy of Engineering, a Fellow of the AAAS, a member of ACM, Sigma Xi, Eta Kappa Nu, and Tau Beta Pi, a former member of the Computer Science and Telecommunications Board of the National Research Council, and a member of the Mayor's Cable Advisory Board for the City of Newton, Massachusetts.

DAVID CLARK [F] (ddc@lcs.mit.edu) graduated from Swarthmore College in 1966, and received his Ph.D. from MIT in 1973. He has worked since then at the MIT Laboratory for Computer Science, where he is currently a senior research scientist in charge of the Advanced Network Architecture group. After receiving his Ph.D., he worked on the early stages of the ARPAnet and on the development of token ring local area network technology. Since the mid-1970s he has been involved in the development of the Internet; from 1981 to 1989 he acted as chief protocol architect in this development, and chaired the Internet Activities Board. His current research area is protocols and architectures for very large and very-high-speed networks. Recent activities include extensions to the Internet to support real-time traffic, explicit allocation of service, pricing, and new network technologies. He was a major author of two studies by the Computer Science and Telecommunications Board of the National Research Council on information infrastructure. He is a member of the ACM and the National Academy of Engineering. He is chairman of the Computer Science and Telecommunications Board of the National Research Council. He received the ACM Sigcomm award, the IEEE award in International Communications, and the IEEE Hamming Award for his work on the Internet.