A critical review of “End-to-end arguments in system design”

Tim Moors
moors@ieee.org
Polytechnic University
5 Metrotech Center
Brooklyn, NY 11201, USA

***Abstract*- The end-to-end arguments raised by Saltzer, Reed and Clark in the early 1980s are amongst the most influential of all communication protocol design guides. However, they have recently been challenged by the advent of firewalls, caches, active networks, NAT, multicasting and network QOS. This paper reviews the end-to-end arguments, highlighting their subtleties, and provides additional arguments for and against end-to-end implementations. It shows the importance of trust as a criterion for deciding whether to implement a function locally or end-to-end, and how end-to-end implementations can help robustness, scalability, ease of deployment, and the provision of appropriate service. It focuses on the performance implications of end-to-end or localized functionality, and argues against end-to-end congestion control of the form used by TCP.**

I. Introduction

The paper “End-to-end arguments in system design” [1] (henceforth called “The Paper”) has had a profound impact since it was published in 1984. For example, the literature contains numerous comments saying that the end-to-end arguments are “one of the most widely applied rules of system design” [2] and one of few general architectural principles of the Internet [3]. Yet the very success of The Paper has often led to people accepting the principle dogmatically, or applying it without considering the attendant subtleties. Furthermore, new networking products and architectures such as firewalls, caches and Network Address Translators, active networks, multicasting and network Quality Of Service all challenge the end-to-end arguments. At the same time, the recent success of peer-to-peer networking exemplifies the benefits of end-to-end implementations.

This paper reviews The Paper, and highlights new arguments for (and against) end-to-end implementations that have become pronounced in the 20 years since The Paper was published. The authors of The Paper have themselves revisited the original principle in a modern context, evaluating active networking in terms of end-to-end arguments [4], one author (Reed) has written on how end-to-end arguments remain pertinent today [5], and another (Clark) has written about their role in the context of the changing requirements of the Internet [6].

This paper first describes the primary end-to-end argument and the “careful file transfer” case study, as presented in The Paper. It then considers the performance implications of end-to-end implementations, and describes additional end-to-end arguments. Finally, it considers how responsibility and trust affect the applicability of the end-to-end arguments, and evaluates the suitability of end-to-end implementations of error control, security, routing and congestion control.

II. The end-to-end arguments

The “end-to-end arguments” guide the placement of functions in a communication network. Certain functions can be implemented in both end systems and the network, e.g. error control, security, and routing. The end-to-end arguments suggest that those functions would be better implemented in the endpoints. While The Paper presents multiple end-to-end arguments, it emphasizes one relating to correctness of function. “[T]he end-to-end argument” states that certain functions “can completely and correctly be implemented only with the knowledge and help of the application standing at the endpoints of the communication system. Therefore, providing [such] function[s completely] as a feature of the communication system itself is not possible. (Sometimes an incomplete version of the function provided by the communication system may be useful as a performance enhancement.)” [1, p. 278]. (We will return to the other end-to-end arguments in § E.)

A. An example

The Paper includes a case study of “careful file transfer” as an example of the application of end-to-end argument. As illustrated in Fig. 1, the “careful file transfer” involves transferring a file from the disk on a source computer to the disk on a destination computer. The file may become corrupted at various points on the end-to-end path, e.g. on the communication channel (❶), in intermediaries such as the router (❷), or during disk access (❸). For examples of the causes of errors, the reader should refer to an error-control text such as Lin and Costello [7] for discussion of link errors, and to Stone and Partridge [8] for a discussion of router and end-system errors. The authors argue that only a check made at the endpoints (i.e. from information stored on the disks) can

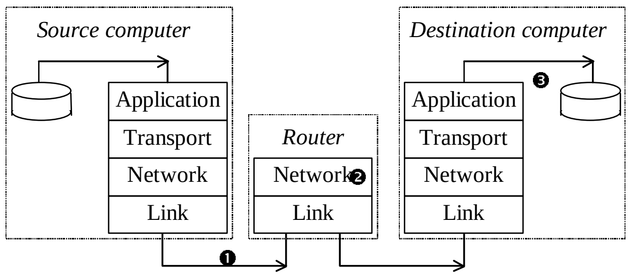

Fig. 1: Errors can occur at different points (❶, ❷, and ❸) on the end-to-end path.

"completely and correctly" ensure that no error has been introduced.

B. *Defining correctness*

It is important to scrutinize what is meant by "complete and correct" implementation of a function. The service of secrecy can be implemented completely and correctly by the applications in Fig. 1 using one-time-pads, in that the destination application can decrypt the file being transferred, but at no time do entities at other points in the network (e.g. ❶, ❷, and ❸) have sufficient information to decrypt the file. On the other hand, there is no way to provide similar absolute guarantees for integrity: To enhance integrity, the source adds to the payload certain information (e.g. a CRC) that produces redundancy within the transmitted information. The destination then checks that the redundancy remains in the received information. It is always possible, albeit perhaps improbable, that a modification can change the transmitted information to a new value that contains the redundancy of the form expected by the destination. Thus, we argue that while it makes sense to discuss integrity being provided over the *complete path* that the information traverses, it is not possible to completely ensure integrity due to the probabilistic nature of integrity checks.

An extreme form of lack of correctness is total failure. The Internet architecture was influenced by military desire for robust operation when network elements become unavailable [9], which led to displacing functions that relied on state information for proper operation to the end systems. This is based on the "fate-sharing model [that] suggests that it is acceptable to lose the state information associated with an entity if, at the same time, the entity itself is lost" [9]. In a sense, fate can be viewed as being an extreme form of correctness: An end that does not trust the fate of an intermediate system may also not trust the ability of the intermediate system to perform other functions correctly.

C. *Carefully identifying the ends*

To apply the end-to-end arguments, it is not surprising that one must identify the communication endpoints. The ends are the points where the information is *originally* generated or *ultimately* consumed. The Paper points out that in interactive speech, the ends are not the telephones, but rather the people. This highlights an important, but subtle point: that for payload information the end is usually the uppermost entity in the system, such as a human user.

Returning to the careful file transfer example, The Paper suggests that file transfer applications are the endpoints, and that after the destination has stored the file to disk, it should read the file off disk and calculate an integrity check, which it returns to the source application, which decides whether the transfer was "correct". This use of application-layer checks is also supported by a recent study of the sources of data transfer errors [8]. However, it is a popular belief (e.g. [2, p. 289][10]) that the *transport* layer (e.g. TCP) is the endpoint for applications such as file transfers. The transport layer merely resides in the end-*system*. The transport layer checks will not protect against errors introduced as the information is written to disk. (Note that while natural noise and potential design errors are omnipresent, physical security measures (e.g. electromagnetic shielding) can isolate a system from security threats. Thus, if the user trusts the end system then the transport layer of the end system may constitute a *security* endpoint.) *According to the end-to-end arguments, applications, not transport layers, should check integrity.*

So why do most applications entrust transport layers such as TCP to provide reliable transfer? There are two reasons for this: the first reason is the high *cost* of the application performing the error checking, and the second reason is the low *benefit* from this activity.

The first reason is that application writers do not wish to be burdened with reliable transfer considerations. Reliable transfer protocols are complicated, and the endpoint may not have the *capacity* to implement the service (e.g. dumb terminals such as telephones), although progress in semiconductor technology outlined by Moore's Law is circumventing this. Alternatively, the endpoint designer may not *understand* how to implement the complicated function correctly, although this can be avoided with the Protocol Organs technique of making protocol functions available through a library [11]. Reliable transfer protocols also involve significant overhead in order to check the information that has been delivered [12], e.g. disk access at the destination of the careful file transfer example is doubled by the requirement of reading the file off the disk to check it. Furthermore, if the ultimate endpoint is a human, the system designers generally do not want to make the human user solely responsible for the menial task of checking integrity. Finally, for the application to check for errors within the computer, it must also account for the possibility of itself, and not the data, being in error. Such totally self-checking systems are non-trivial [13].

The second reason is that for *most* applications, the error rate within the end-system is negligible. For example, when applications store or move information within the computing system, they generally assume that the system will not alter the information. Low-level hardware is often designed with this assumption in mind, e.g. providing error control to protect memory from soft DRAM errors. It is only when the information has an appreciable chance of being corrupted, e.g. when traversing an unreliable network, that the application seeks integrity checks. The crux of the issue is how does the application know whether its information will experience appreciable errors? It is relatively easy for the application to know about the reliability of its local system, but more difficult for it to know about the reliability of intermediaries that its data must pass through as it crosses a network. Thus, *the decision to implement reliable transfer in the transport layer is not justified on the basis of end-to-end arguments, but rather on the basis of trust.* We will return to this issue of trust in § III.

It is important that the application be able to disable integrity checking by the transport layer (and this is not possible with the most popular reliable transport protocol, TCP). This is because while *most* applications can neglect the chance of errors in the local system, *some* applications will be concerned about errors in the local system (e.g. when writing to disk), and may implement their own, truly end-to-end checks. By the end-to-end argument, such end-to-end checks render lower-layer checks redundant and useful (or detrimental) only in terms of performance. Also, interactive applications implicitly provide positive acknowledgements (though not error recovery) of receipt of requests by sending the corresponding reply. Again, for such applications, transport layer mechanisms for providing reliable transfer may only impede performance.

D. Performance

The Paper notes that local implementations of functions may enhance performance above that achievable using end-to-end implementations alone. In this section, we first consider how local implementations may enhance performance, and then consider how they may degrade performance.

Performance benefits of localized implementations

We will first consider the function of error control, and then extend to other functions such as multicasting and QOS.

Localized error detection reduces the network load by reducing the distance that erroneous packets will propagate through the network. Localized error recovery also reduces the network load by reducing the distance that retransmitted packets must propagate. Localized error recovery can also reduce the delay in delivering the packet with integrity, because of these reduced distances that the erroneous packet and retransmission must propagate, and because the propagation delay (as used to dimension retransmission timers) across a hop is likely to be smaller and less variable than the end-to-end propagation delay. (Smaller propagation delays also benefit localized flow control and congestion control.)

If routing is performed locally within the network, as is common, then terminals may not know which links their traffic will traverse. The efficiency of retransmission-based error control depends on the bit error rate of the link, and can be improved by choosing a segment size that is appropriate for the bit error rate (e.g. small segments when the BER is high). However, this requires segmentation within the network, and separate error control within the network. Thus, again, localized error control can offer improved efficiency.

Multicasting can be implemented end-to-end (e.g. by a source sending separate copies of the information to each destination), or locally (e.g. by the network copying the information as it branches towards the destinations), or a combination of both. Localized multicast branching can reduce the transmission load on the source, and the network near the source. Similarly, if nodes in the multicast tree leading from the source root to destination leaves can aggregate flow control or acknowledgement information from the destinations, or even retransmit information as needed, then they can reduce the load on and near the source. Caching provides similar benefits to localized multicasting in that both techniques expand information at points within the network, reducing the distance that it must travel. Content Distribution Networks (CDNs), most notably that run by Akamai, also improve performance by the source pushing information into caches within the network, rather than delivering information end-to-end.

All of the performance improvements described above result from localized implementations improving the *efficiency* with which communication resources are used. This distinction between efficiency and performance is important when we consider the implementation of "Quality of Service", in particular the assurance of delay requirements. Traffic tends to experience appreciable delay through a network when it passes through points of congestion, where traffic is buffered until it can be forwarded (rather than being discarded). The localized technique for ensuring delay requirements (exemplified by ATM and the Integrated Services protocols of the Internet, e.g. RSVP) is for endpoints to reserve resources within the network before communication, so nodes within the network know which traffic should be served first to meet its delay requirements, and which traffic can be delayed. This technique is necessary when the network can be overloaded by delay-constrained traffic, since excess traffic of this type must be "blocked". The end-to-end technique for ensuring delay requirements (exemplified by the Differentiated Services protocols of the Internet) is for endpoints to label their traffic to indicate its delay requirements, and for nodes within the network to use this information to decide the service order. This technique cannot prevent delay-constrained traffic from congesting the network, but when this traffic occupies only a fraction of the network capacity (as is common in modern fixed infrastructure networks which are dominated by data rather than voice and video traffic), then it can assure the delay-constrained traffic priority over the other traffic. That is, the localized approach is suitable when efficiency is needed because the load of delay-constrained traffic approaches the network capacity, whereas the end-to-end approach is suitable when this is not the case.

If communicating endpoints are not available simultaneously, then communication may progress faster if the endpoints can communicate with an intermediary, rather than directly communicate end-to-end. For example, sending email allows asynchronous communication, avoiding the problems of telephone tag. Generally, a client sends email to their server using a reliable transfer protocol such as TCP, their server then forwards the email to the destination's server (again using TCP), and the destination client eventually transfers the mail from their server (again using TCP) in order to read it. The servers may still drop the mail, so an end-to-end acknowledgement (e.g. between humans) may still be needed, but the localized error control between servers allows the mail to be transferred rapidly even when the endpoints are not available simultaneously.

The final performance benefit of localized implementations stems from sharing and economies of scale. An extreme example of this is the function of multiplexing and switching: A network could be constructed as a broadcast medium, and with only the endpoints involved in multiplexing. However, such a network would be inefficient since unicast traffic would be distributed to many endpoints that had no need to receive it. It is more efficient (in terms of communication capacity) for intermediaries (routers) to participate in the switching and route traffic only to the endpoints that need it.

The economies of scale from sharing also extend into end-systems. For example, multiple applications (endpoints) sharing a security function will not need to duplicate functions such as key management and random number generation. The paper gives another example: An end-to-end implementation of reliable transfer "may increase overall cost, since … each application must now provide its own reliability enhancement" [1, p. 281]. The end-to-end argument against sharing is that it is fairer if the user pays (§ E).

Performance benefits of end-to-end implementations

Having considered the performance benefits of localized implementations, we now consider the performance benefits of end-to-end implementations.

The principle performance benefit of end-to-end implementations is that such implementations tend to reduce the amount of processing required in the network, allowing the network to operate at higher speed when processing is the bottleneck (as is currently common with optical transmission technology). While the end-to-end arguments do not *require* that the network offer limited functionality and be simple [14], simple or "stupid" networks [15] are often a consequence of applying end-to-end arguments. Simple networks are more *scalable* than complicated networks (since there is less processing to extend as the network expands), and this is an important contributor to the recent success of peer-to-peer networking (e.g. Gnutella and Napster). However, it is important to note that some techniques for improving scalability such as NAT (for address scaling), and caching require *more* processing within the network, and are contrary to the end-to-end arguments.

A second performance benefit of simple networks (which follow end-to-end arguments) are that they are easier to design and change, and this short design turnaround time allows them to track improvements in implementation technologies. The network will also not duplicate a function (and the costs of implementing that function) that is implemented end-to-end for reasons of correctness, or because the endpoint design was unaware of the function being available in the network (e.g. because the endpoint was designed to be portable, or because the network description was too complicated).

Finally, end-to-end functions need only be encountered once (at the endpoints), whereas localized functions may be encountered multiple times, e.g. once for each hop that the traffic takes through the network. This repeated processing can also degrade performance. For example, when bridges operate in a store-and-forward mode, only forwarding packets that were received without errors, then each packet will be delayed by at least its transmission time in each bridge. If instead, the bridges allow erroneous packets to pass, then they can start forwarding them as soon as they enter the bridge, and the decision to discard (and the consequent delay) will only be made at the destination endpoint. Several bridges offer adaptive forwarding to merge the benefits of end-to-end and local implementation: They check the integrity of incoming frames, and when frames have a low error rate, they forward frames directly, in a cut-through manner, without buffering, leading to low delay, whereas if the error rate increases, they store-and-forward the frames, preventing erroneous frames from propagating.

Given that end-to-end implementations can both benefit and hinder performance, it is not possible to generalize the performance implications of end-to-end implementations.

E. Additional end-to-end arguments

While The Paper often refers to the singular "end-to-end argument" concerning correctness of function, and this is the most famous end-to-end argument, there are also other end-to-end arguments. This section reviews these arguments, relating to appropriate service, network transparency, ease of deployment, and decentralism.

A corollary of the correctness argument is that if a function is implemented end-to-end for reasons of correctness, then any local implementation may be redundant because it has already been implemented. A second end-to-end argument is that local implementations may be redundant because certain applications never need the function to be implemented, *anywhere*. This end-to-end argument states that a function or service should be carried out within a network layer only if all clients of that layer need it. (The authors of The Paper include this aspect in their definition of "*the* end-to-end principle" (italics added) in [4].) Generally, the end-system has better knowledge than the network of what type of service it needs, and has more information to provide the service. For example, this argues against the inclusion of error control (either error protection, detection or correction) within the network, since applications such as uncompressed voice do not need high integrity, and are sensitive to the delays that error recovery can introduce. This also argues that endpoints should implement compression since they know the type of information being transferred, and so can apply an appropriate compression technique.

Another expression of this end-to-end argument is that end-to-end implementations lead to a "user pays" system. In contrast, functions provided in the network add a cost that is borne by all users, irrespective of whether they use the function. A counterargument is that the total price paid by all users may be lower than the user-pays system, since the implementations can be shared as described in § D.

A corollary of this argument is that end-to-end implementations lead to a network that is more *flexible*, since it does not incorporate features that are required by specific applications. For example, loading coils were introduced into telephony cabling to improve the quality of voice calls, but now interfere with the provision of data services over that cabling [16]. Features designed into the network leave as a legacy not just themselves, but also other applications that come to be designed to assume that the feature will exist in the network [15], and this installed base of applications creates inertia that makes it difficult to change the network. As the authors of The Paper later wrote [4], the key to network flexibility "is the idea that a lower layer of a system should support the widest possible variety of services and functions, so as to permit applications that cannot be anticipated." The difficulty arises in determining *which* network functions support the widest variety of services and functions when the future services and functions are unknown. The end-to-end arguments lead to a minimalist approach, making the network flexible by virtue of the fact that it doesn't contain any functionality that might interfere with new services. However, new services, such as active networking, might need new functionality within the network. When the authors of The Paper revisited the end-to-end arguments in the context of active networks [4], they pointed out the tradeoff between flexibility, and another end-to-end argument: that end-to-end implementations allow the network to be simpler, and more transparent, and hence easier to describe, model, and predict. This is particularly important when networks scale, as the number and complexity of potential interactions between endpoints rises.

Allowing endpoints to implement services that are appropriate to their needs (e.g. web browser plug-ins) also aids the deployment of new services, since it can proceed without the protracted process of changing the network. Consequently, services implemented at endpoints in the Internet (e.g. streaming media) have progressed much faster than both those implemented in the core of the Internet (e.g. the Multicast Backbone) and new services introduced by the telephone industry. Endpoint implementations allow a service to be trialed by a minority of people who like to be on the cutting edge of technology, or have particular need for the service, and then for the service to spread from this installed base. In contrast, in order for a service to be installed within the network, it needs to already have an appreciable customer base; something that hard to produce because of network externalities (also known as Metcalfe's law) [17]. Implementing services within the network relies on the foresight of the network provider, whereas the fate of services implemented at endpoints is more market driven.

Finally, the end-to-end arguments are popular with advocates of decentralism (e.g. see [18]), since both views question the ability of a central control to predict what will be important in the future.

III. Responsibility and trust

In this section, we consider how responsibility and trust influence the end-to-end arguments. We start by considering the function of reliable transfer, and then extend to the other functions of security, routing, and congestion control.

Section II.B explained why integrity checks can never be complete, and § II.C discussed how applications are often satisfied with transport layer error control, even though it is not truly end-to-end, because they trust the end-system. In the case of integrity checks, the endpoint has an interest in the application of the service, and so takes responsibility for ensuring that the service meets its requirements, supplementing the network's service if necessary. If the endpoint could trust the service of the network, then it would not need to supplement the network's service. Trust of the network is not a characteristic of current Internet protocols, since they originated in a military environment [9] and grew in a research and development environment, where network components often fail, are compromised, or perform erratically. However, the Internet is now becoming a mature operational commercial environment, and this need for end-to-end checks may be weakened. Furthermore, while end-to-end checks may help detect that *a* component is misbehaving, they often lack the selectivity of localized checks needed to determine *which* component is misbehaving and should be replaced.

Security is another service where the endpoint may be interested in the service, and so may implement that service. However, there are also cases in which the endpoint is not *responsible* for ensuring security. For example, military systems often seek the performance offered by the technological advances in "Commercial Off The Shelf" end-systems, but also need assurances that these COTS systems will not divulge secret information. A common approach is to implement encryption at the link level (e.g. see [19]) so that the endpoints can use COTS technology that need not be proven to be trustworthy, while still maintaining security. Thus, the value of end-to-end implementation of security depends on which entity is *responsible* for security.

Routing is another function that may be implemented either at the endpoints ("source routing") or within the network ("transparent routing"). It is instructive to consider why source routing (which the authors of The Paper supported [20]) has now generally fallen from favor in preference for transparent routing. The main reason for route computation within the network is that we now consider the network to be *responsible* for routing information to its destination(s). A second reason is that routing within the network may be more responsive to link failures or congestion that is localized within the network, and may be better able to provide a hierarchy to handle the complexity of the Internet.

The final function that we will consider is congestion control. Congestion can occur in networks when the offered load exceeds the capacity of the network. Congestion control involves predicting or detecting congestion, and responding by reducing the offered load. In today's Internet, congestion control is primarily implemented in end-systems: Most traffic is carried by TCP, which employs a Slow Start algorithm [21] to try to avoid congestion, uses the rate of acknowledgement return to estimate the permissible transmission rate, and interprets packet loss as indicating congestion that requires that the source throttle its transmissions. The only network support is some Random Early Discard devices that reinforce TCP's behavior by signaling the onset of congestion by discarding packets. However, congestion control is not amenable to end-to-end implementation for the following reasons: First, like routing, congestion is a phenomenon of the network, and since multiple endpoints share the network, it is *the network* that is responsible for isolating endpoints that offer excessive traffic so that they do not interfere with the ability of *the network* to provide its service to other endpoints. Second, it is naive in today's commercial Internet to expect endpoints to act altruistically, sacrificing the performance that they receive from the network in order to help the network limit congestion. The end-to-end arguments that enable the success of peer-to-peer applications also allow the rapid proliferation of applications that do not behave in a "TCP friendly" manner. It is cavalier to allow the commercially valuable Internet to be susceptible to such risks. The requirement that the transport layer implement congestion control also prevents the use of active networking to make transport layers configurable [22]. Summarizing these first two reasons: even though the network is *responsible* for controlling congestion, it has no reason to *trust* that endpoints will cooperate in controlling congestion.

A third argument against endpoint implementation of congestion control is that it is inappropriate for certain networks, leading to an unnecessary performance penalty. For example, Slow Start unnecessarily impedes sources that are transmitting on optical circuits (which don't congest), Media Access Control protocols already provide congestion control for traffic that is local to a LAN, and the assumption that packet loss indicates congestion is invalid for wireless networks in which appreciable loss may also occur due to noise. Fourth, the transport layer lacks the innate ability to detect that congestion is imminent; it can only detect the possible presence of congestion, e.g. through observing packet loss. Schemes such as RED may signal imminent congestion, but they do so by unnecessarily discarding traffic for which the network has already spent resources partially delivering. Fifth, endpoints that implement congestion control separately must independently re-learn the network state, leading to excessively cautious behavior. Finally, while the endpoint may know *how* it would like to adapt to congestion, it is the network that knows when and where adaptation is needed [23], and should be responsible for ensuring that adaptation occurs.

Thus, congestion control is one function that is not well suited to end-to-end implementation.

IV. Conclusion

The end-to-end arguments are a valuable guide for placing functionality in a communication system. End-to-end implementations are supported by the need for correctness of implementation, their ability to ensure appropriate service, and to facilitate network transparency, ease of deployment, and decentralism. Care must be taken in identifying the endpoints, and end-to-end implementations can have a mixed impact on performance and scalability. To determine if the end-to-end arguments are applicable to a certain service, it is important to consider what entity is *responsible* for ensuring that service, and the extent to which that entity can *trust* other entities to maintain that service. The end-to-end arguments are insufficiently compelling to outweigh other criteria for certain functions such as routing and congestion control. So we must conclude by quoting The Paper: "A great deal of information about system implementation is needed to make this choice [of end-to-end or local implementation] intelligently" [1, p. 282].

References

[1] J. Saltzer, D. Reed and D. Clark: "End-to-end arguments in system design", *ACM Trans. Comp. Sys.*, 2(4):277-88, Nov. 1984

[2] L. Peterson and B. Davie: *Computer Networks: A Systems Approach*. Morgan Kaufmann, 1996

[3] B. Carpenter: "Architectural principles of the Internet", IETF, RFC 1958, Jun. 1996

[4] D. Reed, J. Saltzer and D. Clark: "Active networking and end-to-end arguments", *IEEE Net. Mag.*, 12(3):69-71, May/Jun. 1998

[5] D. Reed: "The end of the end-to-end argument", http://www.reed.com/Papers/endofendtoend.html, 2000

[6] D. Clark and M. S. Blumenthal: "Rethinking the design of the Internet: The end to end arguments vs. the brave new world"; Workshop on The Policy Implications of End-to-End, Dec. 2000

[7] S. Lin and D. Costello, Jr.: *Error Control Coding: Fundamentals and Applications*, Prentice-Hall, 1983

[8] J. Stone and C. Partridge: "When the CRC and TCP checksum disagree", *Proc. SIGCOMM*, 2000

[9] D. Clark: "The design philosophy of the DARPA Internet protocols", *Proc. SIGCOMM '88*, pp. 106-14, Aug. 1988

[10] V. Jacobson: "Compressing TCP/IP headers for low-speed serial links", IETF, RFC 1144, Feb. 1990

[11] T. Moors: "Protocol Organs: Modularity should reflect function, not timing", *Proc. OPENARCH 98*, pp. 91-100, Apr. 1998

[12] B. Lampson: "Hints for computer system design", *ACM Operating Systems Review*, 15(5):33-48, Oct. 1983

[13] D. Pradhan: *Fault-Tolerant Computer System Design*; Prentice Hall, 1996

[14] D. Reed: email to the end-to-end mailing list, May 23, 2001

[15] D. Isenberg: "The rise of the stupid network"; *Computer Telephony*, Aug. 1997, pp. 16-26.

[16] R. Lucky: "When is dumb smart?", *IEEE Spectrum*, 34(11):21, Nov. 1997

[17] S. Liebowitz and S. Margolis: "Network externality", *The New Palgraves Dictionary of Economics and the Law*, MacMillan, 1998

[18] N. Negroponte: "The future of phone companies", *Wired*, 4(9), Sep. 1996

[19] M. Anderson, C. North, J. Griffin, R. Milner, J. Yesberg, K. Yiu: "Starlight: Interactive link"; *Proc. 12th Annual Comp. Security Applications Conf.*, pp. 55-63; 1996

[20] J. H. Saltzer, D. P. Reed, D. D. Clark: "Source routing for campus-wide Internet transport", *Proc. IFIP WG 6.4 Int'l Workshop on Local Networks*, pp. 1-23, Aug. 1980

[21] V. Jacobson: "Congestion avoidance and control"; *Proc. SIGCOMM '88*; pp. 314-29, Aug.1988

[22] C. Partridge, W. Strayer, B. Schwartz, and A. Jackson, "Commentaries on `Active Networking and End-to-End Arguments'", *IEEE Network*, 12(3), May/June 1998.

[23] S. Bhattacharjee, K. Calvert and E. Zegura: "Active networking and the end-to-end argument", *Proc. Int'l Conf. on Network Protocols*, 1997