End-To-End Arguments in System Design[1]

Summary

The end-to-end argument, a principle for placing functions in layered systems, shows that correctness can be ensured by end-to-end checks and retries, while low-level implementations serve mainly for performance enhancements.

Important points

1. Low-level mechanisms cannot guarantee correctness, but they can improve performance by reducing error frequency. Correctness requires end-to-end checks and retries at the application level.
   Justification: This is important because while low-level mechanisms help, they cannot cover all threats. An end-to-end checksum and retry provide correctness at minimal cost. Also, issues such as duplicate suppression and encryption follow the same principle as data correctness: they are better implemented at the application layer, since only the application has sufficient context to handle them effectively. However, if low-level functions are unreliable, end-to-end retries may loop excessively. In this case, low-level mechanisms are still necessary, not for guaranteeing correctness, but for providing a sufficiently reliable environment on which application-level optimizations can build.
2. The "true" ends vary with the application. What's useful for one case may harm another.
   Justification: This is important because it prevents misapplication of the principle. It shows that the argument is a guideline rather than a rigid rule, forcing designers to consider the application context, such as latency versus accuracy, rather than blindly enforcing reliability at one layer. It encourages nuanced engineering decisions that fit the actual end requirements instead of applying a one-size-fits-all solution.
3. History shows repeated rediscovery, indicating that end-to-end arguments is a general principle. It keeps reappearing in different domains: networking, file storage, banking audits, OS kernels, even RISC architecture.
   Justification: This is important because it demonstrates the generality of the principle: correctness belongs at the endpoints in any layered system. Historical failures, such as corrupted tape systems or network gateways, show the cost of ignoring it. It also provides a unifying way to think across fields — whether in networking, operating systems, or storage, the same principle applies. Finally, it reinforces that the end-to-end argument is not just theoretical but solves practical, recurring problems in real-world systems.

Comments/Questions

1. My past work experience has mostly been at the application layer, but I've often faced similar situations that require careful consideration. In practice, we emphasize error-handling at the right level. For instance, if we have a network module and a client module that uses it, then the network module should have its own error-handling strategy, while the client module should have its own as well. Errors that can be resolved within the network module should be contained there, and only those requiring client attention should be passed up. Another example, when designing retry mechanisms, users might have concerns about network traffic usage. Blindly retrying may not solve the real issue and could instead waste resources. It may be better simply to display an error message and let them fix it. Overall, this reflects the same philosophy: in layered systems, functions should be handled at the layers most suited to the application's context.
2. This paper's end-to-end argument resonates strongly with current challenges in AI systems. In the recent boom of AI, one question has always concerned me: how can we ensure that prompts are neither altered nor misused by any entities in the AI pipeline, such as the model provider, cloud service, or even user interface? In my opinion, we should not blindly assume that AI systems are benign. Instead, as the paper suggests, correctness and integrity must ultimately be enforced at the endpoints. This means that critical safeguards, such as encryption and decryption, should remain at the hosts, not within intermediate layers of the pipeline. For example, asymmetric cryptographic techniques, such as public-key encryption[2], could ensure that sensitive inputs remain protected, even while interacting with AI services. Consistent with the end-to-end principle, all of these operations must be performed at the endpoints, not within the network.

Citation

---

1. Saltzer, J. H., Reed, D. P., & Clark, D. D. (1984). *End-to-end arguments in system design*. ACM Transactions on Computer Systems, 2(4), 277–288. https://doi.org/10.1145/357401.357402 ↩

2. Public-key cryptography. (n.d.). *Wikipedia*. Retrieved September 1, 2025, from https://en.wikipedia.org/wiki/Public-key_cryptography ↩