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. Certain functions can only be correctly implemented at the endpoints. Some correctness properties (e.g., reliable file transfer, security, data integrity) cannot be fully cover inside the network or lower layers, they must be checked at the application level. Lower-layer mechanisms are for performance and reliability, not full correctness.
   Justification: This is important because intermediate layers can only provide partial assurance, while the application itself determines what counts as “correct.” For instance, in e-commerce, the network might confirm that a payment request reached the server, but only the application can confirm that the purchase was recorded and inventory updated. Without these checks at the ends, users may see failures despite flawless network delivery. That said, if the underlying network were too unreliable, these safeguards would be overwhelmed, so low-level mechanisms remain useful for improving performance and baseline reliability.
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. In real-time voice calls example, it shows that the argument is a guideline rather than a rigid rule. It encourages nuanced engineering decisions that fit the actual end requirements instead of applying a one-size-fits-all solution.
3. “Occam’s razor”: End-to-end arguments caution against hardwiring functions into lower layers; instead, some functions should remain replaceable at the application level, keeping systems modular, flexible, and adaptable to diverse requirements.
   Justification:
   This is important because it enables layered systems, such as networks and operating systems, to support many different applications (e.g., browsers, email, video conferencing) without requiring core modifications each time. Building complexity into the network or lower layers locks the system into a particular use case, increasing the burden on applications that do not need those features. By keeping the lower layers simple and pushing functionality to the endpoints, systems remain more general and adaptable. For example, the Linux kernel is used by countless systems; if it were optimized only for certain purpose, like streaming, other applications might suffer.

Comments/Questions

1. My past work experience has mostly been at the application layer, but I’ve often encountered situations that resonate the end-to-end argument. In practice, we emphasize handling errors at the appropriate level. For instance, if we have a network module (a relatively low-level component) and a client module that uses it (a higher-level component), the network module should resolve the errors it can internally, while only throwing those that require application-level attentions. Similarly, when designing retry mechanisms, it is not always best to retry blindly at the client side. Users may be concerned about network usage, and repeated retries may waste resources without addressing the real issue. In some cases, simply displaying an error message and letting the user take corrective action, like checking Wi-Fi, is more effective. These examples reflect the same principle as the paper: correctness relies at the "true" ends, while lower-level components should focus on performance and efficiency improvements.
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, encryption and decryption 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 ↩