Network failure mostly arises from improper transport protocol implementations. A few algorithms are investigated to deal with this problem; some of them follow a “conservation of packets” principle which states that a new packet cannot be put into the network unless an old packet leaves. This principle fails under three conditions: connections not reaching equilibrium, injection of new packets before old packets leaving, resource limits causing equilibrium unreachable.

In networks, sender's use of acks acts as a “clock” in data packets moving. Though this ensures network stability, it makes it hard to start, as data flowing requires acks to clock out packets whilst getting acks indicates data flowing. Thus, a slow-start algorithm is developed to gradually increase the amount of data in-transit. It sets congestion window, cmwd, to one packet in beginning and increment on each ack for new data; it also sends the minimum of the receiver's window on sending. This algorithm does not affect performance as in fact the window opens very quickly; meanwhile, it ensures fast data sourcing of connection.

Sender may wrongly inject new packets due to inaccurate estimation of round trip time. One mistake for this can be not estimating the variation of the round trip time. In addition, when load is more than 30%, connection will automatically retransmit packets which are just delayed in transmission; this waste bandwidth and make network congestion worse. Another mistake lies in backoff after retransmit. Currently only exponential backoff works well under a network of unknown settings, which is a linear system.

Packets get lost either because of being damage in-transit, which is rare case, or network congestion. To solve the latter problem, a “congestion avoidance” strategy is introduced; it requires the network to signal congestion occurring to transport endpoints, and endpoints change utilization accordingly. For the first step, network congestion signal is created whenever packet loss and timeout occur. For the second, a congested network has queue length increasing exponentially and window size decreasing multiplicatively under source control; but under no congestion, the network may be wasting bandwidth as a connection may be using less than its fair share, which means that the net is easy to be driven to saturation but hard to recover. Therefore, small, constant changes to the window size may be the best policy to endpoints. The algorithm for this sets cwnd to half the current window size and increment by 1/cwnd on each ack.

Since both above-mentioned algorithms manipulate the congestion window, confusion easily arises when they are implemented together.

Fair sharing of network capacity can only be done on gateways which has sufficient information required. Congestion detection algorithm aims to signal end nodes appropriately early. If congestion is detected early, small resizing of senders' windows can amend it; otherwise massive adjustments are needed. To solve this, round-trip-time prediction is used as a cheap approach against second-order effects in traffic.

End-to-end argument is a design principle guiding which layers of a distributed computer system a module should be placed in; it appears clearer when data communication network develops as a computer system component. In this case, end-to-end argument opposes low-level function implementations.

“Careful file transfer” involves five steps in transmitting a file: at host A the file transfer program

Comment [S1]: Content: A
Organization: A
Writing: A

Grading Criteria
Content: Does your summary contain the most important information presented in the paper? Is the content correct, or does it contain technical errors?
Organization: Do you have good ordering of the sections/subsections? Are the sections linked together well or do they appear abruptly? Does the paper flow? Did you motivated/introduce/define items appropriately before using them?
Writing: Did you use words and grammar correctly. Does the writing have a clear and easy to understand style?

reads the file from disk via file system; the data communication system transmit it; packets move from A to B along network link; at host B the data communication program receives the packet and hands it to the corresponding file transfer application; data is written on the disk. Corresponding to the above steps different threats are present: (1)hardware fault; (2)mistake in buffering and copying of data; (3)transient error in hardware processor; (4)unreliable network link; (5)crash of hosts midway.

To counter the threats, the application can use some error detection techniques; but this cannot remove threat 2 totally as it is hard to write correct programs and furthermore it is not necessarily the file transfer application programmers' job. Also, if threats occur rarely, such countermeasures are costly.

An alternative way is “end-to-end check and retry”, such as implementing a checksum to minimize undetected error. Then if failures are rare, first try would work normally; two or more tries indicate that some parts of the system are broken.

The communication system guarantees reliable data transmission by providing selective redundancy; but this only removes threat 4. When file transfer programs provide its own retries, this guarantee has no effect on correctness of the outcome, which is already assured by the retries. Therefore, end-to-end argument states that the lower level data communication cannot reduce burden of application layer to ensure reliability.

However, it is not always the case that lower levels are totally useless in ensuring reliability. For highly unreliable networks, some implementations of reliability checks in lower levels would be helpful in reducing great number of retries for file transfer application; but these implementations have performance costs as it uses bandwidth and causes delay. In fact, the performance tradeoff may be complex.

To consider performance issue, note that: placing early-checking function in higher level simplifies the process but increase costs as each application then must provide its own reliability enhancement; placing it in lower communication level may be efficient but some applications not in need of this function may find it costly.

This argument can be applied to delivery acknowledgement; possible strategies are making the application know whether the target host acted on the message received and letting the target host guarantee that the message is acted upon by the target application. It also applies to data encryption; the communication subsystem need not provide automatic encryption of all traffic. it also applies to duplicate message suppression, which insists that this mechanism can be omitted from the lower level if it is already implemented in application. It also applies to order to message delivery, which states that an independent mechanism at higher level must control the ordering of actions.

In conclusion, end-to-end argument acts as a rational criterion for assigning functions to layers.