5617, Spring 2020
computer networking and communication

anduo wang, Temple University

TTLMAN 305, T 17:30-20:00

End-To-End Arguments in System Design

http://web.mit.edu/Saltzer/www/publications/endtoend/endtoend.pdf

End-To-End argument

design principle

- the placement of functions among the modules of a distributed system

End-To-End argument

design principle

- the placement of functions among the modules of a distributed system
- functions placed at lower level
  - redundant
  - of little value

moving a function upward

placing a function in a layered system closer to the application that uses the function

- one class of function placement
- sharpened by the emergence of data communication network

data communication network

for a distributed system that includes communication

- draw a modular boundary around the communication subsystem (network) and a firm interface between it and the rest of the system
- a function can be placed at?

data communication network

for a distributed system that includes communication

- draw a modular boundary around the communication subsystem (network) and a firm interface between it and the rest of the system
- a function can be placed at
  - the network subsystem
  - the client (application that uses the function)
  - the joint nature
  - redundantly

data communication network

for a distributed system that includes communication

- draw a modular boundary around the communication subsystem (network) and a firm interface between it and the rest of the system
- a f
  - t
  - t
  - t
  - r

End-To-End argument

- the function in question can completely and correctly be implemented only with the knowledge and help of the application standing at the endpoints of the communication subsystem
- providing that questioned function as a feature of the communication subsystem is impossible

example function — reliable data transfer (rdt)

from host A to host B, failures can occur at various points

- A passes (app) data to the rdt program
- A rdt program askes the network subsystem to transmit
- the network subsystem moves packets from A to B
- B communication program removes packets from the network protocol to the rdt app
- rdt app writes the received data on the disc

reliable data transfer (rdt) — 1st attempt

brute force countermeasure

- reinforce each of the steps along the way
  - using duplicates, time-out, retry, redundancy, error checking
- reduce the probability of each individual threat

rdt — alternate approach

end-to-end check and retry

- if something wrong, retry from the beginning
- when failure rare:
  - normally work on a first try, occasionally a 2nd/3rd tries

brute force countermeasure VS. end-to-end check and retry

Q: whether or not this attempt to be helpful on the part of the network is useful to the rdt app

- brute force
  - even the threat of one step (e.g., step 4) is eliminated, the rdt app must still counter the remaining threats
  - only reduce the frequency of retries
  - no effect on the inevitability of correctness of the outcome

brute force countermeasure VS. end-to-end check and retry

Q: whether or not this attempt to be helpful on the part of the network is useful to the rdt app

- brute force
  - even the threat of one step (e.g., step 4) is eliminated, the rdt app must still counter the remaining threats
  - only reduce the frequency of retries
  - no effect on the inevitability of correctness of the outcome
  - for the network to go out of its way to be extraordinarily reliable does not reduce the burden on the app …

brute force countermeasure VS. end-to-end check and retry

Q: amount of effort to put into reliable measures

- an engineering trade-off based on performance, rather than a requirement for correctness, frequently the trade-off is complex
- brute force
  - more efficient (hop-by-hop), but some app may find the cost of the enhancement not worth the result
- end to end check and retry
  - within app, simplifies the network but increases overall cost

other functions

delivery guarantees

secure transmission

duplicate message suppression

in order message delivery

delivery guarantee

lower level support may be wasting effort

- the acknowledgement really desired is an end-to-end one
  - knowing the message was delivered to the target host is not very important
  - what the app wants is whether or not the target host acted on the message
- implemented at the app level anyway, originated only by the target app
- but (still) useful within the network as a form of congestion control

delivery guarantee

lower level support may be wasting effort

- the acknowledgement really desired is an end-to-end one
  - knowing the message was delivered to the target host is not very important
  - what the app wants is whether or not the target host acted on the message
- implemented at the app level anyway, originated only by the target app
- but (still) useful within the network as a form of congestion control

secure transmission

not need for the network to provide encryption / decryption of traffic

- the network trusted to securely manage the keys?
- data still vulnerable as they pass into the target node / or fan out the target app

secure transmission

not need for the network to provide encryption / decryption of traffic

- the network trusted to securely manage the keys?
- data still vulnerable as they pass into the target node / or fan out the target app

network-level / app-level protection can be complementary

duplicate message suppression

causes

- time-out triggered failures detection and retry within the network
- originated by the app itself in its own failure / retry
  - e.g. 1 a remote user, puzzled by lack of response, initiate a new login to a time-sharing server
- e.g. 2 system crashes at one end of a multisite transaction

duplicate message suppression

causes

- time-out triggered failures detection and retry within the network
- originated by the app itself in its own failure / retry
  - e.g. 1 a remote user, puzzled by lack of response, initiate a new login to a time-sharing server
  - e.g. 2 system crashes at one end of a multisite transaction

suppression must be accomplished by the app itself, with knowledge of how to detect its own duplicates

in order message delivery

why not in network?

- messages may be sent along independent virtual circuits / paths
- messages may be originated by distributed app

identify the endpoints

end-to-end argument is a property of the specific application

- speech message system
  - scenario 1: two people in real-time conversation
  - scenario 2: voice packets stored for later listening

historical notes

the debate: datagram VS. virtual circuit
RISC
open OS

the debate

datagram? virtual circuit?

- modularity argument
  - reliable, in order, duplicate-suppressed stream of data within the network
  - favors virtual circuit

the debate

datagram? virtual circuit?

- modularity argument
  - reliable, in order, duplicate-suppressed stream of data within the network
  - favors virtual circuit
- end-to-end argument
  - centrally provided versions of those functions incomplete for some app, others will find it easier to build their own
  - favors datagram & connectionless protocol

reduced instruction set computer (RISC)

- better performance by implemented exactly the instructions needed from primitive tools
- attempt to anticipate the client's requirements for an esoteric feature will miss the target
- client will reimplement those features anyway

open OS

- against making any function a permeant fixture of lower level modules
- make functions always replaceable by an app's special version
- more flexible for apps

end to end argument and "Occam's Razor"

Occam's Razor

- do not make more assumptions than the minimum needed

end-to-end argument is a kind of "Occam's Razor"

- when it comes to choosing the functions to be provided within a subsystem
- the subsystem frequently specified before app that uses the subsystem are known
- a rational principle for organizing the subsystem