Reading4

问题回答：

1. end-to-end argument考虑的是如何划分、分割一个系统中各个层级的功能，这通常需要在reliability和performance之间作出trade-offf，来考虑如何将不同的function安排在不同的层级当中， 由于把function放在底层需要更高的性能，所谓end-to-end 原则给出的就是一些需要在application（最顶层）设计而不应该在更低的层级（network层）安排的原则。这样function都在application实现，network只管传输数据即可。

2. 三种适合使用end-to-end 原则的情况：
   数据加密：网络传输层要想传输，那么数据的加密还是需要密钥的，这样在network层进行很大的加密也没什么意义，不如在network层只进行通用加（防治用户不小心发出不该发出的信息），而在application端口进行end-to-edn加密，只有两个端口知道key

   抑制冗余的数据：由于一些application层的请求无法被network识别为重复操作，因为application层需要识别冗余，如果application层进行冗余识别的操作，那么其他的低层级的抑制重复操作就需要适当减少

   电话通信（实时）：电话需要的是实时的语音传输，这个时候轻微的数据damage没那么重要，重要的是实时性，要是为了过度保证数据的正确性而带来延迟是难以接受的，这些行为应该在端口（application）中进行，也是一种tradeoff

3. 三种不适合使用end-to-end原则的情况：
   金融业务：支付宝对数据的延迟性能要求没有语音通话高，但是对数据的准确性要求极高，这种涉及高度安全的内容以及容易被篡改的内容，需要在网络传输的过程中就进行足够的机密，在端口加密解密是不够的，在网络传输通道中如果不做处理，被窃取会造成很严重的后果

   有明确先后顺序的application：如果每一个步骤都不可或缺并且有明确安排，那么最好按照既定的顺序发送各种request和回复，单单从application层的操作是不够的，传输层也应该按照一定的顺序

   信息安全业务：如果信息是绝密的，安全级别很高，则需要精心设计network层的安全加密传输，不仅仅局限于application的加密技术

4. 我认为对于inode文件系统，把文件名也放在block中，目录存在inode中不太好，功能没有完全分离，可以考虑专门储存目录以及目录中信息的位置，（但这样读取小文件的速度会变慢），把inode和dir分开的目的是link 雨unlink更方便管理，比原有的设计更清晰，在某种程度上也是一种end-to-end的功能分离原则

5. 我认为对于OS来说，DMA应该完全取代外设之间的设备读取，不需要经过CPU的处理，甚至可以考虑存完cache直接由两个设备之间通信，这样就是所谓的end-to-end原则，省去了底层调用的麻烦。
   对于操作系统的调用，systemcall和内核的call也可以放在更高的层级，因为不同的application可能需要不同的systemcall，这对于系统来讲将会是处理速度的飞跃，但对于application的编写者就不够友好。

reading记录：

Introduction：

这篇文章谈论的是关于一种函数的放置方式，这种方式强调把application需要调用的函数上移。
在一个包含交互的计算机系统中，每个模块之间都有边界，并且有明确的接口。
函数一般被以下几种方式执行：
communication system，client（joint venture），
函数只有在communication system endpoints中才会被正确执行

end to end caretaking

假设从A向B发送文件，执行以下几个步骤：

1. A让文件系统读磁盘，并且传输到fixed size 的blocks中
2. 在A让数据交互系统执行一些协议，分包
3. data communication network把包congA传到B
4. hostB端一个communicatuoin program 移除来自protocol的packet，交给second part of file transfer application
5. file transfer program 让fs写入数据

以下是一些可能出现的问题：

1. 由于磁盘问题，读取文件可能产生错误
2. fs或者file transfer可能出错
3. 硬件的processor
4. communication可能丢包或者改变packet的数据
5. 两个host其中一个可能崩掉

提出结论：需要careful transfer，checksum

trade-off：降低很小的reliability，用高层的checksum来换取performance
原因2条：

1. 因为底层的系统可能给很多个application服务，这一个操作会给很多不需要的子系统降低efficiency
2. 因为底层系统掌握较少的信息，所以不够efficient

other examples or end-to-end argument

1. ARPANET早起 在收到一个ACK的message之前不对其他消息相应
   但是知道消息被发过去这件事其实并不是很重要，而是应该返回这个消息是否被检查通过并运行了
   另一个策略是让target host足够精致，他保证确认接受信息的时候目标host就会执行

   在一些application中，如果要求其他的action被完成之后才完成，或者可能不完成，就需要end-to-end ack
2. 数据加密
   如果data transmitt提供加密/解秘，就必须要求有密钥
   因此文件就会易损
   application仍然需要检查authentity

   因此底层就不需要加密了，加密也没用

   底层加密被用于防治用户不小心发出去了消息，因此网络层的加密变得简单，变得通用
3. 抑制冗余的数据
   Application层级的重复请求无法被抑制，因此application层需要检查冗余
   remote system的用户
   Application层无论如何都需要检查冗余（login），那么network以及底层就不再需要
4. 确保FIFOmessage的发送传递
   更高层必须确保diliver的顺序
5. 事务管理
   有些需要返回的事物不需要ack
   打电话、接听电话允许有轻微的damage，却要保证实时性，不能有太大的延迟