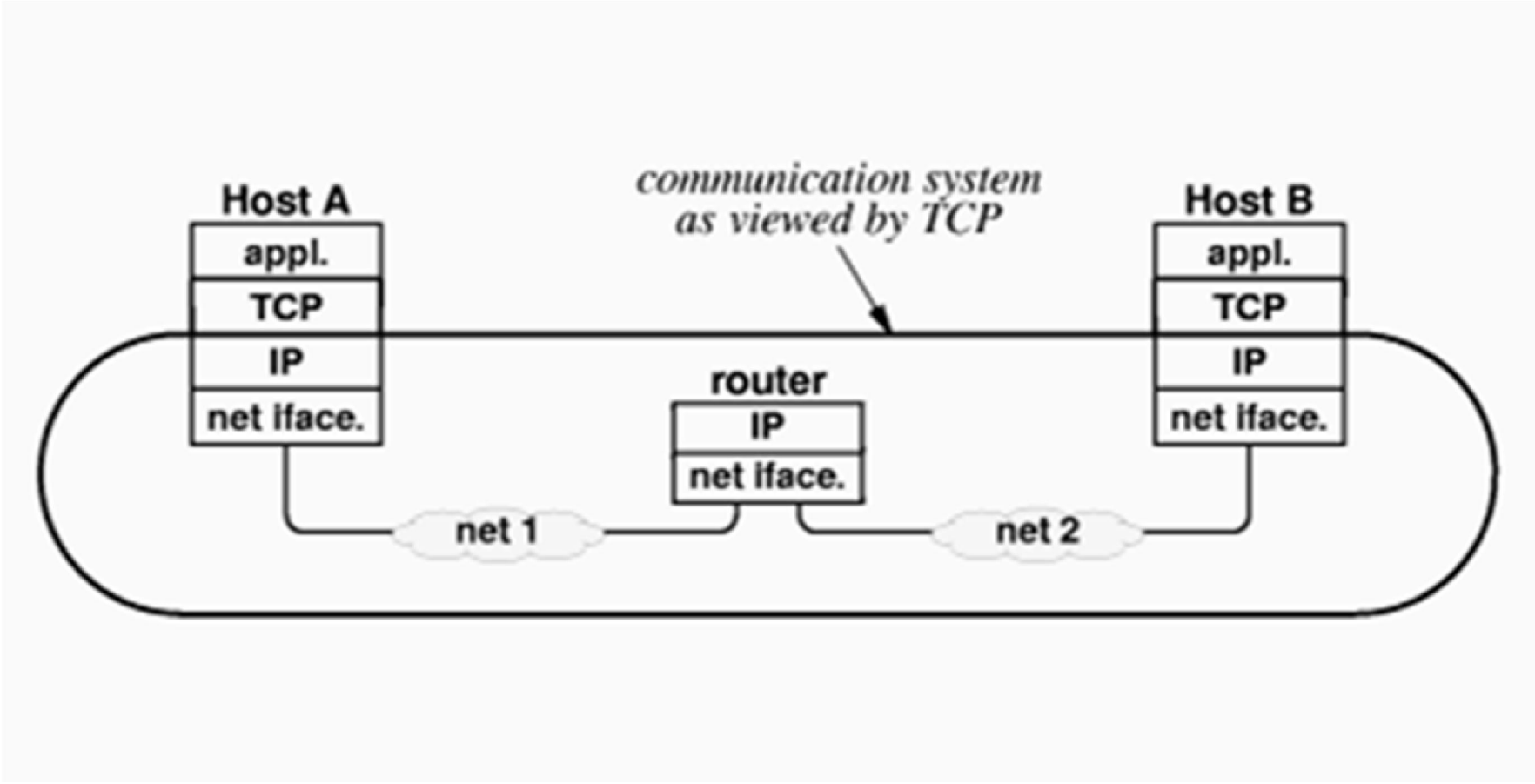
当TCP和IP运营


Host A
appl.
TCP
IP
net iface.
communication system
as viewed by TCP
router
IP
net iface.
Host B
appl.
TCP
IP
net iface.
net 1
net 2

完整的TCP传输服务提供

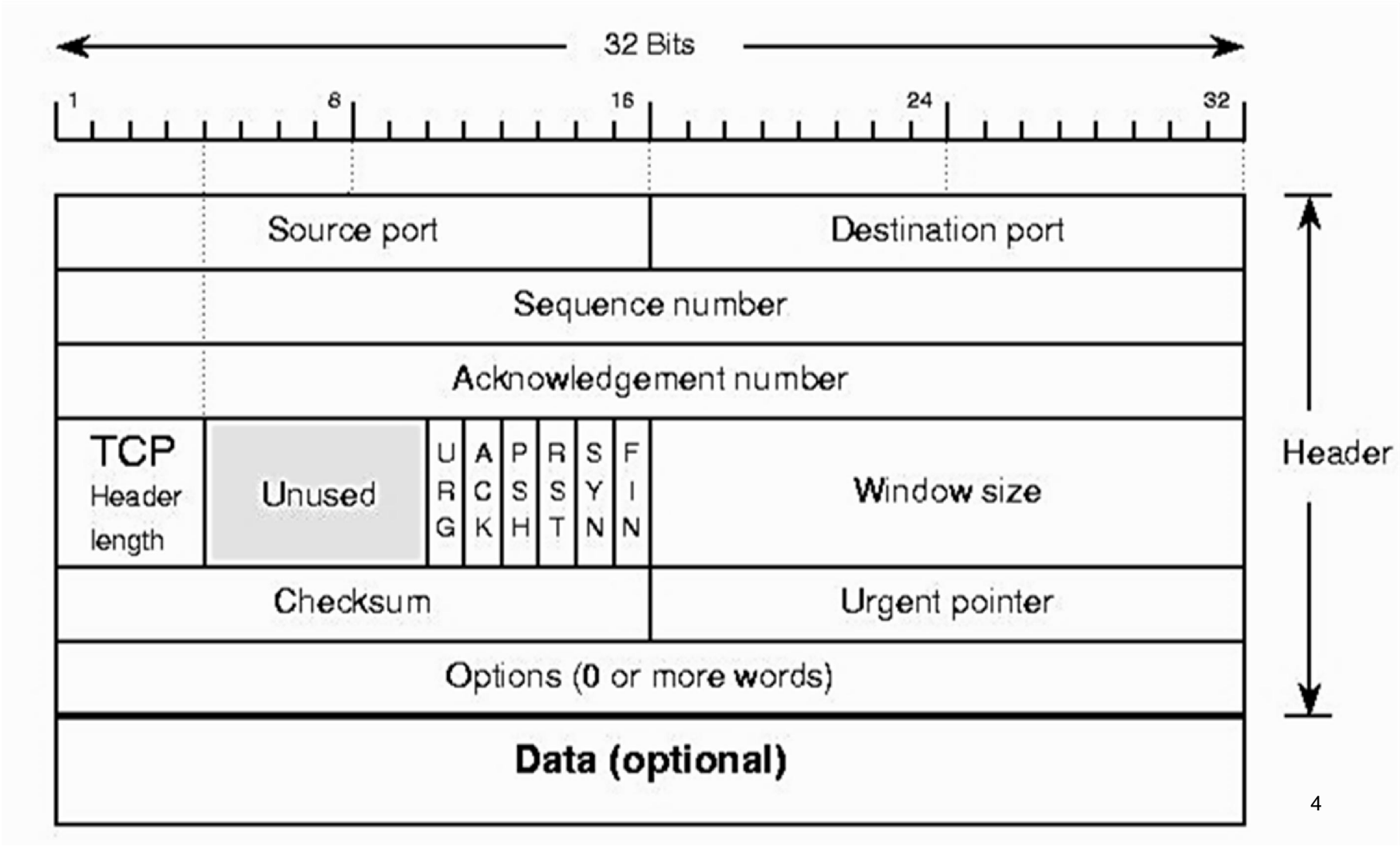
- TCP传输服务具有以下特点：

  - ***连接方向：*** 上面已经讨论
  - ***点至点的通讯方式：*** 每个TCP连接具有完全相同 二 端点。
  - ***完整的可靠性：*** TCP保证数据将被传递准确的发送，即无数据丢失或失序的
  - ***全双工通信：*** TCP连接允许数据在任一方向流动
    - TCP缓冲器传出和传入数据
    - 这允许应用程序继续执行其他代码而数据正在被传输

完整的TCP传输服务提供

- ***流接口：*** 该 资源 应用程序发送 *连续* 跨越连接的八位位组序列
  - 数据传递 *整块* 到TCP交付
  - TCP并不能保证在同样大小的块，它是由源应用程序提供传输数据。
- ***可靠的连接启动：*** TCP两种应用 同意 任何新的连接
- ***优美的连接关闭：*** 这两者都可以请求连接被关闭
  - TCP保证关闭连接之前，可靠地交付所有数据

TCP分段报头格式


32 Bits
1
8
16
24
32
Source port
Destination port
Sequence number
Acknowledgement number
TCP Header length
Unused
URG
ACK
PSH
RST
SYN
FIN
Window size
Checksum
Urgent pointer
Options (0 or more words)
Data (optional)
Header

打开一个TCP连接： 三路握手

- 打开一个TCP连接：
  - 服务器调用 *插座，绑定，* 和 *听* 正在执行 *被动* 打开
  - 客户端调用 *连* 正在执行 *活性* 打开
- 要将呼叫 *连* 导致TCP客户端实体发送一个SYN（ *同步* ）分割：
  - 这部分包含了客户端的初始 序列 它的数据数
- 服务器TCP实体必须承认收到的SYN段，它必须把自己的SYN段：
  - 这包含了数据的初始序列号（注：全双工操作）
  - 此外，该服务器的SYN包含客户端的SYN报文的ACK

三路握手 - 续。

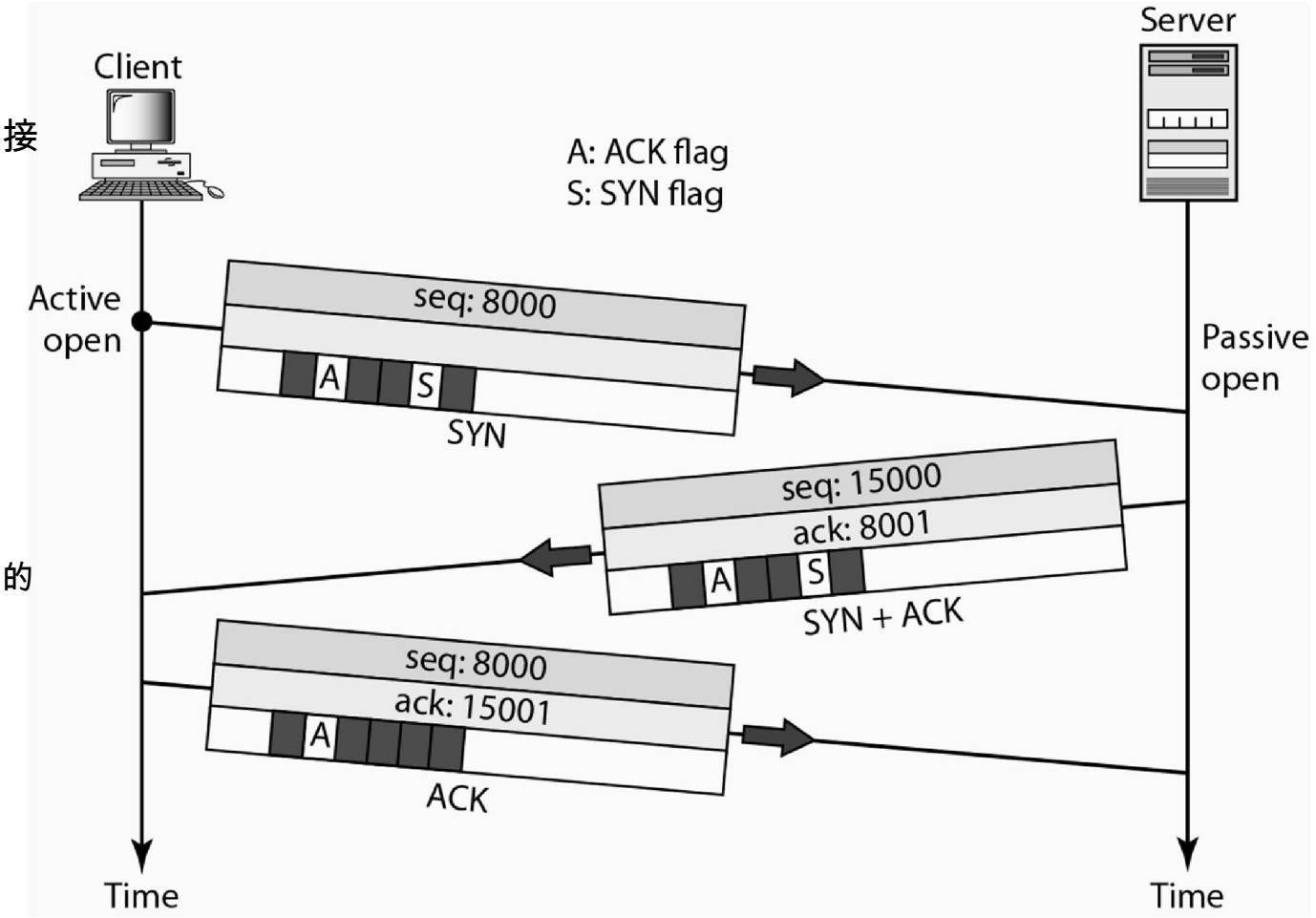
- 客户端TCP实体也必须确认收到服务器的SYN段：
  - 由于有在打开顺序中使用三个段此过程被称为 *三次握手*
- **注意在序列号：**
  - **包含在TCP确认段的序列号（ACK）是 未来预期 序列号**
  - SYN消息占用的序列号空间的1个字节
  - 对自己的ACK不占用序列号
  - 请参阅该示例启动顺序上的下一个幻灯片

例 *开盘* 三次握手

插口（ ）连接

（ ）（ 块 ）

connect（ ）的返回



插座（ ）的b

ind（ ）听（ ）

接受（ ）（ 块

）

accept（ ）返回

关闭TCP连接

- 关闭TCP连接：
  - 应用程序调用 *关* 正在执行 *活性* 关
  - 连接的另一端被认为是执行 *被动* 关
- 要将呼叫 ***关（）*** 使本地TCP实体发送一个FIN段：
  - 这意味着应用程序已完成数据发送
  - 无论是应用程序，客户端或服务器，可以调用 ***关（）***

关闭TCP连接

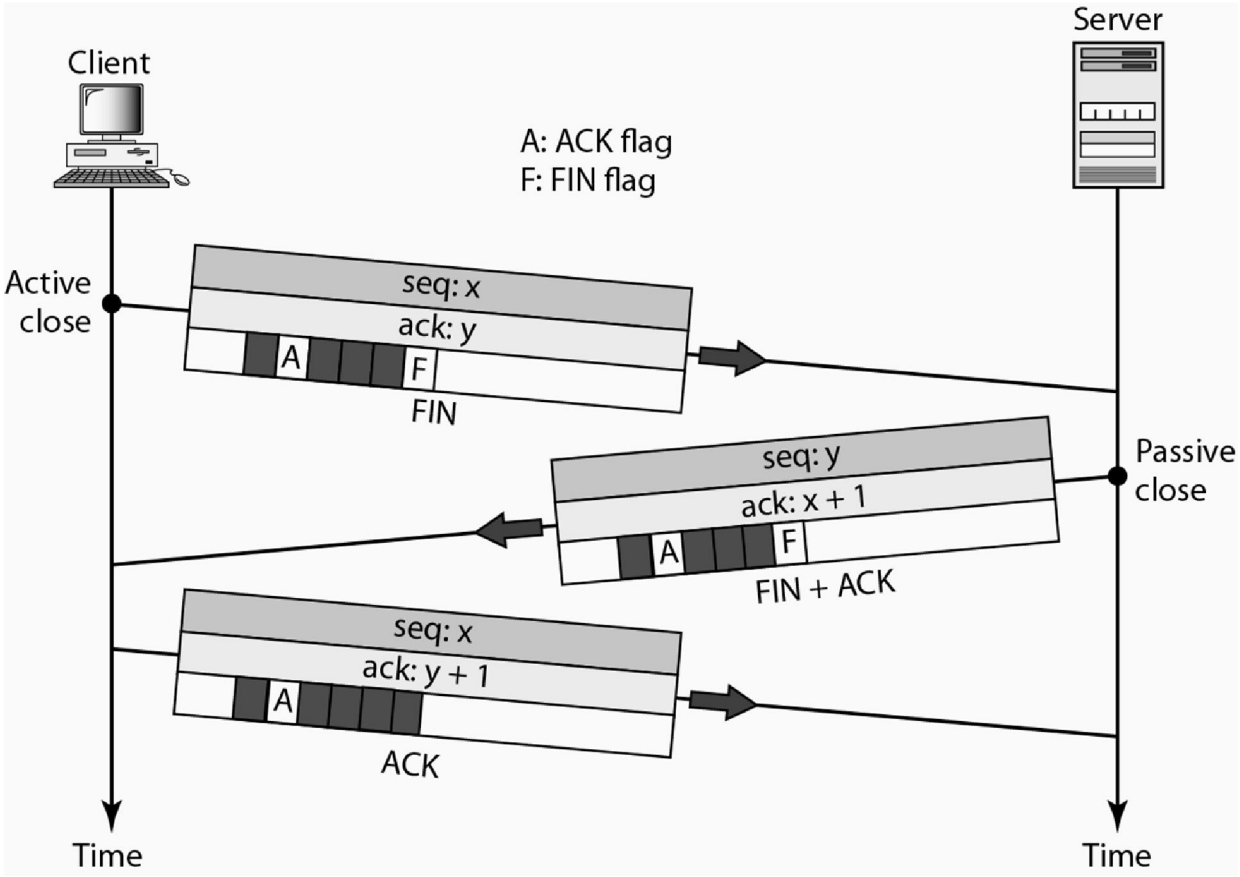
- 的FIN段由接收TCP实体应答：
  - 此FIN消息被传递到应用程序作为 *档案结尾* 和排队 后 任何剩余的数据
  - 一个FIN的接收意味着没有更多的数据将在连接上到达
- 其接收的应用程序 *档案结尾* 可以选择：
  - 离开本地插座打开，以便将数据返回到远程应用更长。这就是所谓的 *半关闭*，要么，
  - 关 它的 本地套接字通过调用 *关（）* 致使其本地TCP实体发送一个FIN。这FIN段必须由远程TCP实体被确认
- 这种选择的结果无论是 *三路* 要么 *四向* 用于关闭连接信号交换序列

关闭TCP连接

- 关闭连接需要在每个方向上，即四个区段是一个FIN和ACK 一般 需要：
  - **然而 *活性* FIN通常与数据和发送 *被动***
    FIN可以与ACK成一个单一的段被组合，即一个三次握手
- 连接也可以通过终止即终止关联的Unix进程应用程序关闭：
  - **这会导致所有开放套接字描述符 关**
  - 这导致FIN段上任何打开的TCP连接发送
- **注意在序列号：**
  - 就像SYN片段，FIN段也占据了序列号空间的1个字节
  - 对自己的ACK不占用序列号
  - 参阅例如连接终止序列上的下一个幻灯片

例 *闭幕* 用一个 *三通* 握手

关（ ）

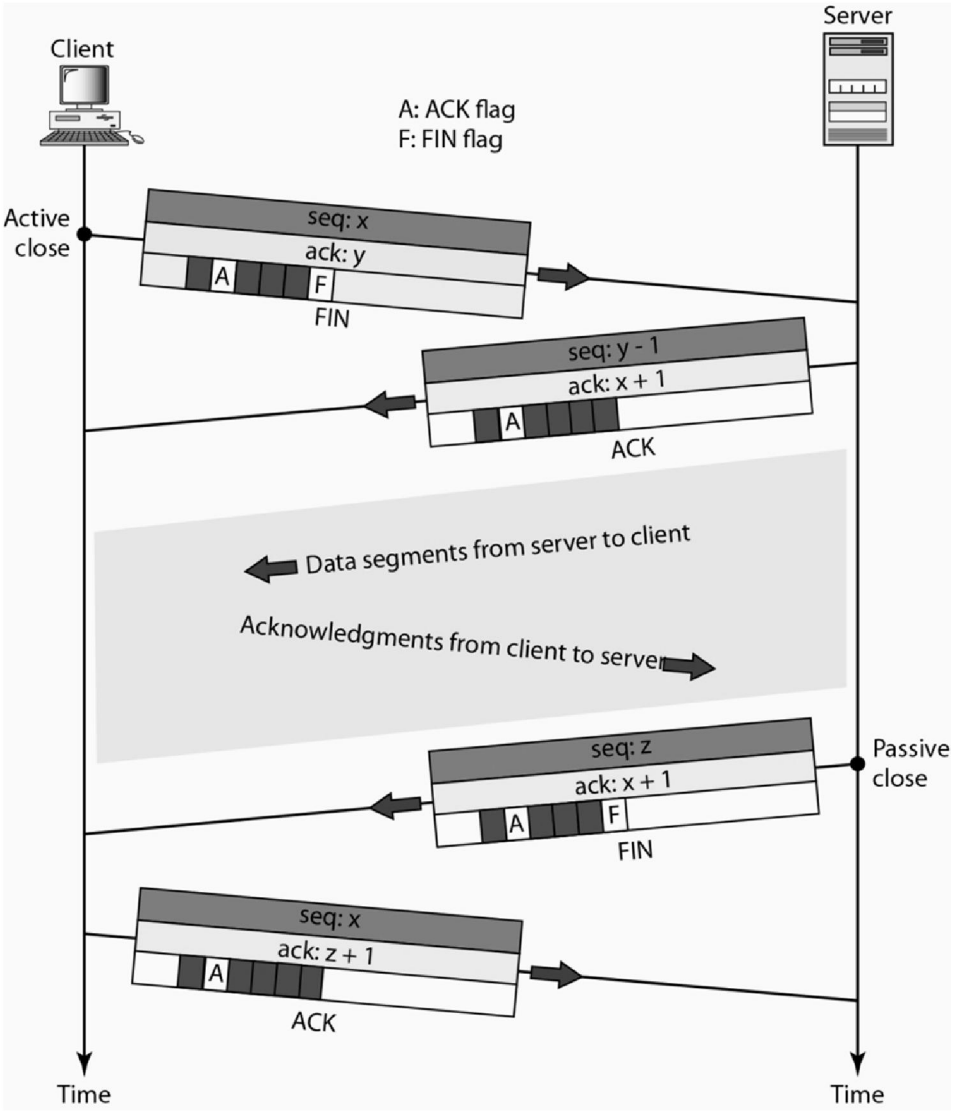


读（ ）返回0

例 *闭幕* 采用四次握手

（半关闭）

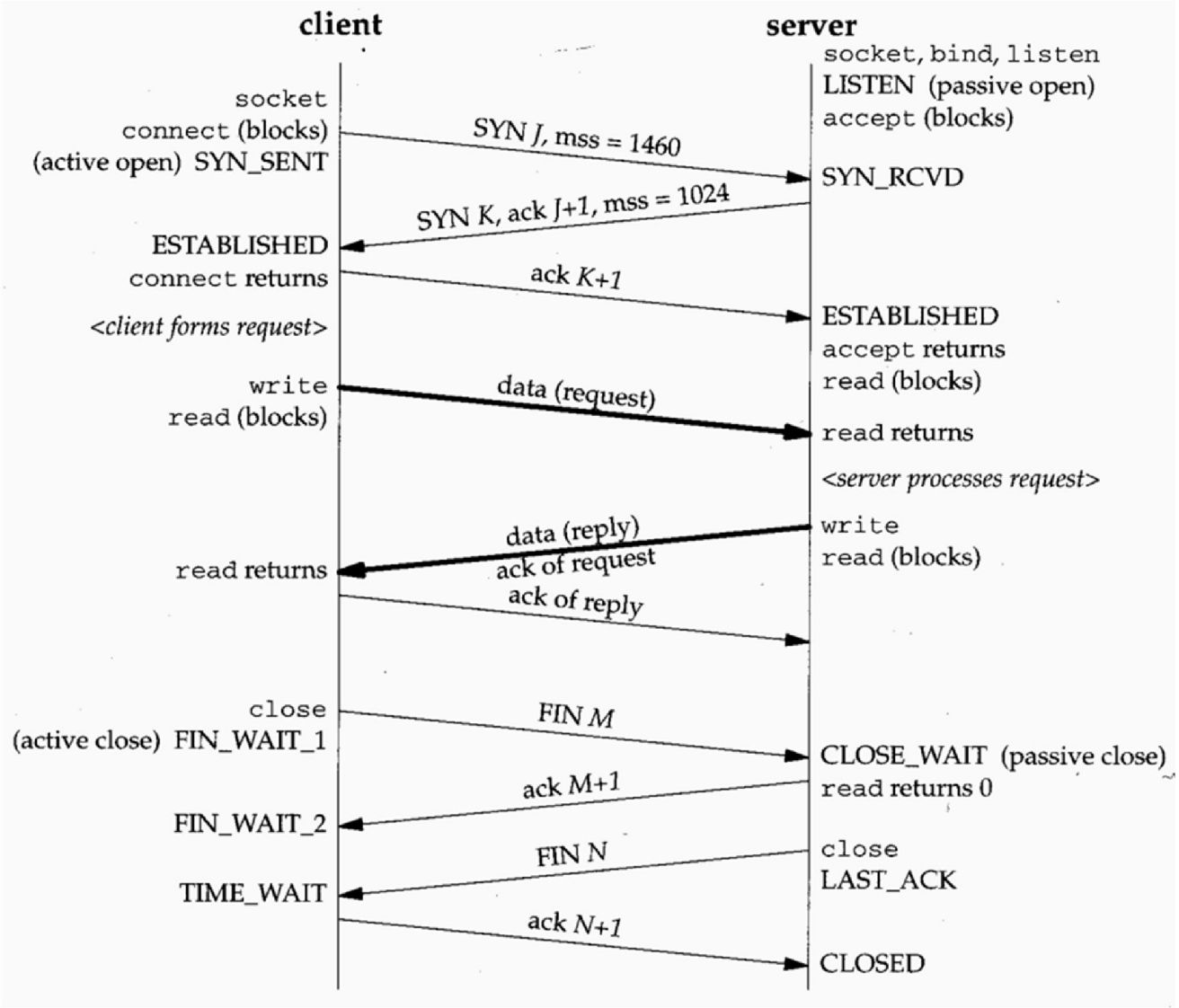
关（ ）



读（ ）返回0

关（ ）

一个完整的连接序列的示例



TCP状态转换图 - 解释

- 该TCP *状态转换图* 显示TCP的连接的操作 *编制* 和

  终止 阶段：

  - 有一个连接定义的11个不同状态
  - 客户端和服务器转换被分别示出为深色固体和暗虚线
  - 从一个状态到另一个的转换依赖于在该状态下接收的分段
    - 例如一个应用程序执行 *活性* 在一旦接收到SYN的CLOSED状态移动到SYN_SENT状态，然后到ESTABLISHED状态以ACK开放

- ESTABLISHED状态是大多数数据传输发生

TCP状态转换图 - 解释

- 连接终止：
  - 从ESTABLISHED状态有两种可能的转变：
    - 应用程序调用 *关* 即一个 *活性* 接近移动到FIN_WAIT_1状态
    - 的应用接收一个FIN（即 *被动* 接近）移动到CLOSE_WAIT状态
- 在罕见的情况下是可能的两端同时发送SYN /翅片（称为 *同时开/关*）：
  - 这个场景不是在这里探索

TCP状态转换图

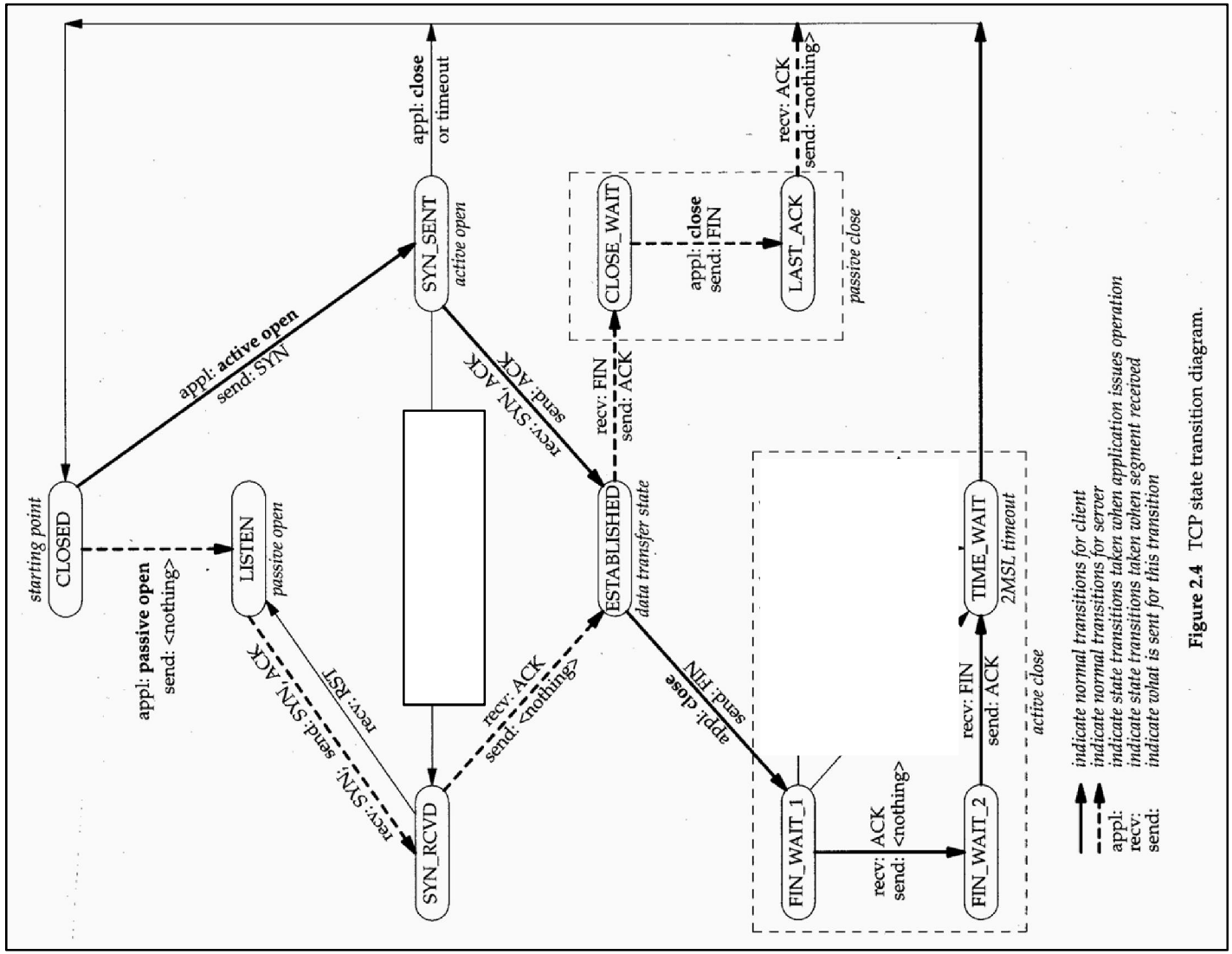


Figure 2.4 TCP state transition diagram.

TCP的TIME_WAIT状态 - 解释

- 执行该结束 *活性* 接近经历TIME_WAIT状态一段 两次 的MSL（最大段寿命）
  又名 *2MSL*

  - MSL是时间的最大量的IP *数据报* 可以住在一个网络。这被连接到TTL字段（最大值为255）
    - 将在后面覆盖

  - TCP选择为1分钟之间MSL的值。4分钟

TCP的TIME_WAIT状态 - 解释

- 有 二 原因TIME_WAIT状态：
  - 为了实现TCP的全双工连接 *终止* 可靠
    - 回想一下两个TCP的服务产品是 *全双工通信* 和 *可靠的终止*
    - 执行一个结束 *活性* 关闭保留在TIME_WAIT状态，因为它可能会重新发送最后的ACK
  - 为了让老 *重复* 段到网络中到期：
    - 包含TCP段数据报可被卷进 *路由循环* 互联网中由于 *路由错误。* 这些被称为 *丢失* 要么 *重复徘徊*
    - TCP必须处理这些重复的已连接 *转世*
    - TCP不会启动 *化身* 当前处于TIME_WAIT状态的连接
    - 这保证了所有旧 *重复* 从以前的化身已过期