**0822-读书笔记**

3.2 Design goals

通过互联网传送数据时,较大的ADU必须分段,因为网络数据包大小受MTU限制。TCP / IP和NDN处理数据分段有两个主要差异。首先,因为TCP将所有应用程序数据视为字节流,所以TCP分段将忽略ADU的边界,因此ADU只能在分段重新组装后进行识别 (Figure1)。NDN数据包携带独立ADU的名字或ADU段,因此这些数据包可以直接匹配应用程序的数据单元。

第二,应用程序可以在数据传输时保证不同程度的观察和控制。在图1所示的简单例子中,如果使用TCP / IP发送几个连续的ADU，若网络上的某一个部分在运输过程中丢失,所有的后续ADU,即使它们已经到达目的地,也将阻塞交付给应用程序。这是著名的head-of-line(HOL)阻塞问题。另一方面,如果使用NDN，面临相同的部分损失问题时,所有成功收到ADU可以立即交付给应用程序而无需等待恢复丢失的部分。

*3.2.1 Goals for the consumer abstraction*

**为了确定消费者抽象的设计目标,我们做一个初始的假设,一般来说,独立的应用程序希望根据自己的优先级进行ADU抓取。因此我们描述设计目标为应用程序可能希望在处理ADU时获得何种支持。鉴于我们仍然在对这个新消费者/生产者的API进行测试,当前的设计目标可能随着时间的推移，当我们对应用程序的需求有更深的理解时进行进一步修订。生产者抽象也是如此。**

**目前，我们相信新的消费者抽象模型应该支持以下应用程序模式。**

1. **ADU顺序抓取,并且允许在必要时丢失流上的任意ADU。这点可支持处理实时流媒体的应用程序。**
2. **并行抓取ADU以加速内容传输。这可以应用在web download 和torrent方面。**
3. **获取独立的动态ADU。物联网等应用程序需要这点。**
   1. Producer context

producer context用于在常见的前缀下发布数据(图4)。它是通过调用给定的名称前缀参数的producer()原语进行初始化。与TCP / IP的服务器端socket不同, producer context在没有连接到网络,没有任何的Interest的情况下，准备发布数据。除了需求驱动的情况,我们的consumer / producer 模型不要求consumer和producer同时被“连接”。因此，数据发布行为可以在任何时间发生,包括生产者断开连接的情况。在我们的Simple-Video 示例中,生产者按照自己的节奏发送数据,并且提前于消费者的获取。

一个应用程序进程调用produce() operation开始数据发送（传送名称后缀和应用程序帧(ADU)内容）。在Simple-Video 例子中，后缀名是帧数。在一般情况下,名称后缀参数允许应用程序开发人员重用相同的producer context，以发布任意的名称子树数据。在Simple-Video 例子中，一个context用于发送视频帧,另一个用于音频帧。生产者名称树中的context定位如图4所示。

当以下操作皆完成时，produce() operation结束

1. 应用程序帧(ADU)已经分段为一组数量适当的数据包。
2. 段号附加到每个包的名字上。
3. 每个包都保证安全(如signed)。
4. 递交数据到send缓存区，并且离开context。

虽然一些生产者应用程序可能想写永久存储的数据包（NDNFS or Repo-NG [9, 11]. ），默认情况下,段是暂时存储在send缓冲区（内存存储数据包）的。

context的发送缓冲区不同于socket发送缓冲区。具体有两个方面：首先,socket发送缓冲区用来转传unacknowledged segments 。而producer context的发送缓冲区是作为传入的Interest包所指示的data包的临时缓存。换句话说,发送缓冲区软化了数据生产和获取之间的时间异步。第二,在socket中,数据包ACK后去除,而在一个context中,数据包的驱逐基于内存环境，例如,当应用程序调用producer()的时候，缓存已经写满，并且替换方式是FIFO。

为了获得Interest所指引的data, producer context.必须attach到本地NFD。这可以通过调用attach()操作。到达的Interest进入接收缓冲区,并且等待轮次，直到它们与发送缓冲区的数据包匹配。如果Interest通过name和Interest选择器匹配data包成功, 就表示这个Interest在发送缓冲区中被满足了。如果没有找到匹配的数据包,那么需要告知应用程序这个Interest。

在某些情况下,对于特定producer context， Interest的到达速度可能太高了以致于不能尽快处理。在其他情况下,请求的数据不能在Interest的生存周期内产生。代替消费者盲目超时的方式,应用程序可以使用nack()操作给Interest一个NACK来使其满足(4.1节),这样消费者可以用最直接的方式来处理这种情况。

3.4 Consumer context

,