1. 现代网络架构存在的问题

# 20世纪末，互联网开始兴起，虽然它采用了不同于电话网的突破性的基于TCP/IP的通信模型，但是其目的还是和电话网一样，是为了实现点对点的通信，所以因特网的先驱者们建立了以IP地址为基础的网络模型。然而发展到现在。网络的主要功能已经发生了很大变化：

# 绝大多数数据密集型产业服务迁移到网络上，比如金融、交通，网络成为了一个储藏数据的工具。

# 数字编码技术的发展，让网络传输的内容由开始的文本，发展到图片、音频、视频等等。

# 任何人都可以通过网络，很方便地获取，创造内容，这导致了每时每刻都有海量的新的内容在网络上产生。

由此可见，网络的主要功能已经由点对点通信变为了内容分发，所以原来的以IP地址为基础的网络模型也就难以适应现在的要求，带来了传输速度上的限制和安全隐患。

在现有网络架构下，我们获取内容时一般需要以下几个步骤：

1. 获得内容所在主机的IP地址
2. 和对应主机建立连接
3. 请求内容
4. 关闭连接

这就带来几个问题：

1. 要先知道内容所在主机的地址，
2. 一旦我们认为获得的主机地址是可信的，就认为主机传送的内容也是可信的。

而实际上我们仅仅是想获取某个内容，并不关心内容所在的主机，并且用主机的可信度代表内容的可信度也更容易带来安全隐患。

1. NDN网络的基本思想，架构，优势

由于现在网络架构的大规模运用，同时不同国家地区对于网络由各自的政策法律，难以将现有网络架构彻底改变为一种全新的架构。这就要求新的网络架构必须保证在一定程度上和现有的网络架构保持兼容性。

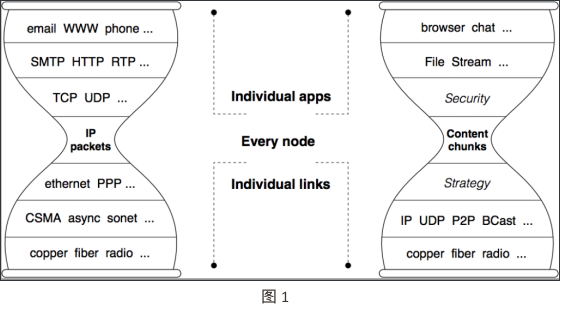


图1左边显示了现代网络的漏斗型架构，IP作为“腰”，扮演着联系上层的应用和底层的传输协议的角色。NDN保留了互联网的漏斗型架构，但是对漏斗的“腰部”进行改进，以便建立分布式网络。在目前的网络架构中，IP数据包只能记录源点和目的点的IP地址，而NDN中最关键的是解除这一限制。在NDN网络中，NDN数据包中的名字可以是任何信息，比如主机、电影、一个远程控制命令等等。这样的改进让NDN网络能充分利用现有网络的大量基础设施，并保证和现有网络架构的兼容性。

1. NDN的基本思想

网络的细腰型结构显得简洁而又健壮，中间IP层提供了最基本的功能，将上下层从逻辑上隔离开来，互不干扰，从而上下层可以不断改进，而又保证兼容性。这是网络在过去几十年爆炸性发展的关键。NDN借鉴了这一优势，将细腰型架构应用到自身。

在一开始，网络间的互相通信都是可信任的，所以没有安全意识。但是随着网络的扩大，互联网的产生，病毒、黑客开始产生。人们不得不在现有网络基础上修修补补以防御各种攻击，这难以应对日益严重的安全问题。NDN从一开始就将安全问题作为核心之一，NDN为每个命名数据提供签名，从底层提供安全保证。

现在的网络有一个End-to-End原则，意思是当要实现一个功能时，如果能只靠端到端的主机实现，那么就绝不修改中间节点，这保证了网络的健壮性。NDN吸收并改进了这个原则，底层只提供最基本的功能。（但是随着网络发展，什么是最基本的功能还有待商榷。）

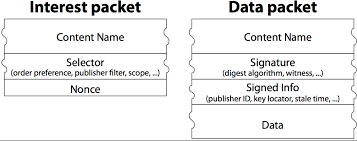
网络流量应该是能自动调节的，负载均衡对于网络的稳定性极为重要。现在的网络通过运输层协议来实现流量控制，而NDN在腰部就提供负载均衡的功能。

路由和转发的分离已经证明是必须的选择，NDN仍将保持这一结构。

1. NDN的架构

在NDN中，通信是由接收方驱动的。比如一个用户为了请求一个数据，需要发出一个Interest包（如图２左所示），在Interest包中包含着用户请求的数据的名字。网络可以通过这个名字来唯一确定一个数据（当然，也有可能不存在这样的数据）

图２



接收到Interest包的节点会记录这个Interest包是从哪个接口接收的，当节点本身存在该Interest包指定的数据时，就会根据记录向对应的接口转发该数据。如果没有，就会根据路由表转发该Interest包。详细结构如下图３所示。

为了减少网络中的流量，达到内容分发的目的，NDN定义了CS、FIB、PIT三个数据结构，如图４所示。当节点一段时间内接收到多个Interest包，且请求的是同一数据时，只有第一个Interest包会被转发。节点先通过查询FIB(Forwarding Information Base)来确定应该向哪个节点转发此请求，转发完成后，这些请求将会被记录到PIT(Pending Interest Packet)中，如下图所示，PIT中记录Interest包中指定的名字和对应接收Interest包的接口，相同名字的合并为同一项。当一个节点接收到一个数据包时，将检查PIT，若PIT中某项中的名字和数据包中的名字相同，将会向对应接口转发此数据包，并从PIT中删除此项。并且会根据自身缓存策略，决定是否缓存此数据包。如果缓存，该数据包将会被保存到CS(Content Store)中，以后节点收到对此内容的请求时，将会从CS中提取此数据包，返回给请求者。流程图如下图所示：

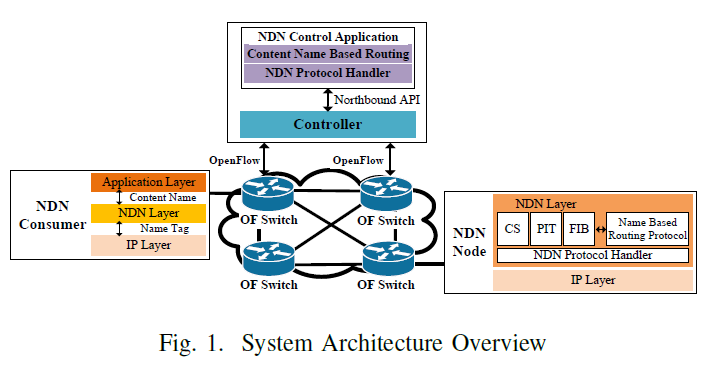
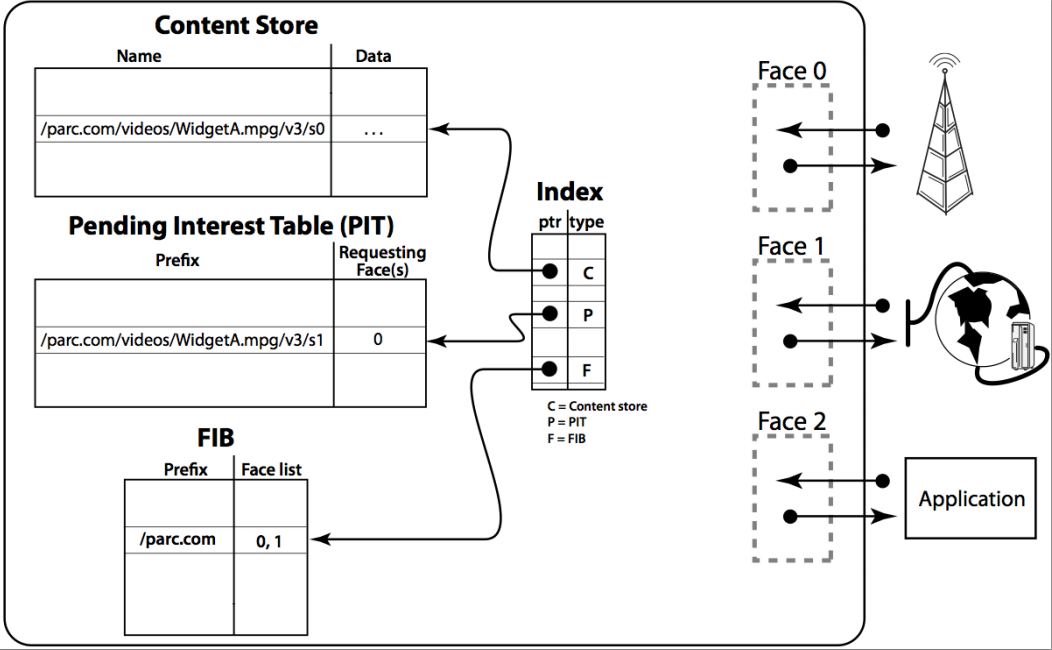


图３

Interest Packet

节点是否存在请求的内容？

CS中是否存在请求的内容？

PIT中是否存在该请求？

FIB中是否记录着转发接口？

返回对应数据包

Y

N

N

N

返回对应数据包

Y

在PIT中记录对应接口

Y

转发该Interest包，在PIT中记录

Y

N

返回不可达

Data Packet

PIT是否存在对该内容的请求记录？

Y

N

向对应接口转发该data packet

缓存决策

NDN数据包相对于网络中的节点是完全独立的，这就使得中间可以NDN缓存数据包以满足将来到达的对此数据包的请求，这又使得NDN不需要向现在的网络那样增添额外的代价，而是从底层就可以支持如内容分发、多播等功能。

1. NDN的内容命名

NDN网络中的每个内容都有一个名字(name)，不同内容的名字不同。对NDN来说，内容的名字仅仅用于路由，标志其唯一性，并不理解内容的名字包含的其他意义。这样的设定就使得每个应用可以在既定的命名规则下选择自己的命名模式以满足其特定需求。这大大提高了NDN中层次间的独立性。

NDN设计了层次化的命名规则，比如，bilibili网站发布的一个视频可以按照如下命名：

/bilibili/video/航拍中国

在这里，“/”是一个分隔符，被“/”分割开的是名字的最小组成单位，其长度，用的字符自由度极高，我们称之为“名字块”。这样的命名方式有利于命名者表示内容之间的关系，比如，航拍中国第一季第六集可以这样表示：

/bilibili/video/航拍中国/1/6

不仅如此，层次化的命名规则方便实现路由聚合，正如现在网络实现的IP路由聚合那样，可以大大减小路由表的大小。

命名规则是NDN架构中最重要的部分，对于如何定义、分配顶层名字块等问题仍待解决，但使用层次结构的命名规则，保证名字与上层的独立性仍是不变的原则。

1. NDN的安全性

　在NDN中，数据本身与安全性绑定在一起，而不是像现在的网络那样，安全性与数据独立开来，安全作为一个功能模块，用来确定数据的来源、完整性，并以此来确定数据的安全性。每条数据都与其名字一起签名，以将二者安全地绑定。数据签名是强制性的，上层无法将之修改或取消。数据的签名与数据的发布者的信息相结合，就能够确定数据来源，这样就使得数据请求者对数据的信任与数据获取方式分离，进一步保证了数据的安全性。并且，NDN还支持对细粒度数据的安全性判断，允许用户对特定上下文中特定数据段的公钥发布者的可信度进行独立判断。

1. NDN的路由转发策略
2. NDN的缓存策略
3. NDN需要解决的问题
4. NDN缓存策略
5. 基本思想

节点为每个到来的内容赋予一个缓存优先级，节点根据缓存优先级来决定是否缓存该内容。缓存优先级由基本优先级、内容大小两个参数计算得到，基本优先级由流行度、偏好度两个参数形成。节点只记录内容的基本优先级，并且每隔一段时间就更新所有内容的基本优先级，在内容到达后，节点再通过基本优先级和内容大小计算内容的缓存优先级，再通过查看缓存空间的大小和比较内容缓存优先级的大小决定是否缓存该内容。并且，流行度、偏好度、基本优先级都是基于一段某一段时间的，我们可以称之为更新周期(updata\_circle)，在每个更新周期的末尾，我们会根据这个周期内收到或者发出的Interst包的情况，并结合上个周期计算得到的的流行度、偏好度、基本优先级，来计算这个周期的流行度、偏好度、基本优先级，从而在下一个周期用这些参数决定对收到的内容的缓存与否。

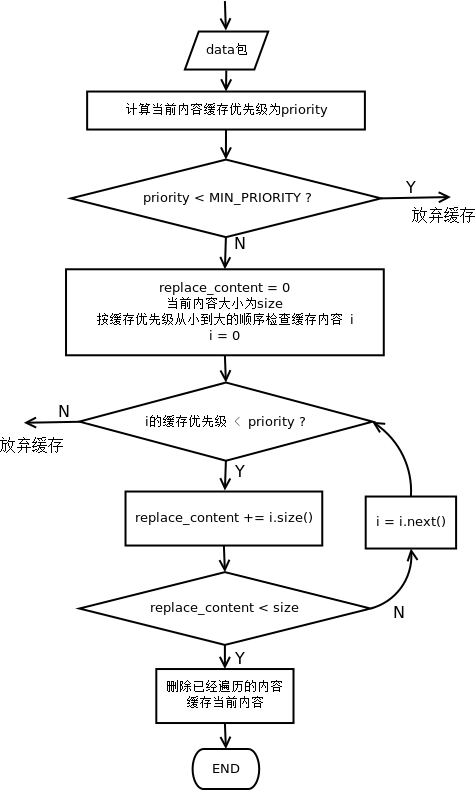
1. 参数定义
2. 流行度(popu\_rate)
3. 含义：表示从某一节点观察，某一内容在网络中的流行程度。
4. 表示方法：一段时间内节点收到的对某一内容的interest packet的数量与此节点收到的interst packet的总数的比。
5. 偏好度(pref\_rate)
6. 含义：表示节点对于某一类内容的偏爱程度
7. 表示方法：一段时间内节点发出的对某一类内容的interest packet的数量与此节点发出的interest packet的总数的比
8. 内容大小(content\_size)
9. 含义：表示数据包中内容的大小
10. 基本优先级(bprior)
11. 表示方法：

popu\_weight\*popu\_rate+pref\_weight\*pref\_rate

1. 缓存优先级(cache\_prior)
2. 含义：表示单位内容的优先级，以防止优先级较高的超大型内容占用过多的缓存空间。
3. 表示方法：

bprior / content\_szie

1. 缓存策略
2. 初始化：节点初始化时会创建 PopularityTable 和 PreferenceTable 两个数据结构，分别用于记录收到的Interest包的数量和发出的Interest包的数量，用于将来计算流行度和偏好度。另外还会创建 basic\_cache\_priority\_table 用于保存内容的基本优先级。初始这些数据结构为空，因为没有收到或者发出任何Interst包。
3. 运行：当节点收到一个来自其他节点的Interest包后，将提取该Interest包请求的数据的名字，并更新 PopularityTable 中对应的项。如果 PopularityTable 不存在对应的项，则需要先创建该项，再进行更新。当节点自身发出一个请求(Interest 包)后，将请求的内容的名字提取出来，并更新 PreferenceTable 中对应的项。如果 PreferenceTable 不存在对应的项，则需要先创建该项，再进行更新。
4. 获取缓存优先级：节点接收到一个数据包后，提取出该数据包的内容的名字 content\_name ，通过名字访问保存基本优先级的数据结构 basic\_cache\_priority\_table ，获得基本优先级 bprior 。若 basic\_cache\_priority\_table 不存在对应内容的缓存优先级，则认为该内容的缓存优先级为0。



1. 缓存决策：获取该内容的缓存优先级后，若该值小于某一设定的值（这里设为MIN\_PRIORITY），则放弃缓存该内容，并丢弃该内容。否则，检查节点缓存空间是否足够，若足够，直接缓存该内容。否则按缓存优先级从小到大的顺序检查节点中已缓存的并且缓存优先级小于该内容的内容，并设

replace\_content\_size=0

每检查一个内容，将其大小加到replace\_content\_size上，若在检查完成之前：

replace\_content\_size >= content\_size

那么删除已经检查的所有内容，并将该内容放入缓存中。若在检查完成后仍未出现上式的情况，则放弃缓存该内容。流程图如上图所示。

1. 保持：在每隔更新周期末期，节点更新一次流行度、偏好度和基本优先级。更新即将开始前已有的基本有的基本优先级为 bprior\_old ,本更新周期内的基本优先级为 bprior\_new 。节点更新这个更新周期内的流行度和偏好度，并计算出这个更新周期内的基本优先级：

bprior\_new=popu\_weight\*popu\_rate+pref\_weight\*pref\_rate

再更新节点保存的的基本优先级：

bprior=bprior\_old\*old\_prior\_weight + bprior\_new\*new\_prior\_weight

此时得到的基本优先级即可用于下一个更新周期的缓存决策。更新完成后，需要清空 PopularityTable 和 PreferenceTable 中的记录，以用于记录下一个更新周期收到和发出的Interst包的数量，计算流行度和偏好度。

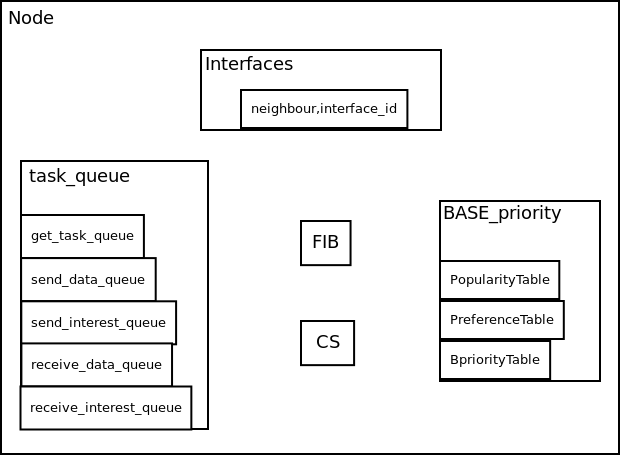
1. 仿真实验
2. 拓扑结构

实验采用的拓扑结构如下图所示：



1. 节点定义

节点定义如下图所示，task\_queue保存节点要执行的任务，get\_task\_queue负责保存节点发出Interest packet的任务，send\_interest\_queue、send\_data\_queue、receive\_interest\_queue和receive\_data\_queue分别负责保存节点转发interest packet的任务、发送data packet的任务、接收interest packet的任务和接收data packet的任务。Interfaces保存了节点所有接口的状态，比如接口是否允许连接，是否打开，邻接点是哪个节点等。其他结构的功能上文已经提到，此处不再赘述。



1. 性能评价参数
2. 运行结果
3. 分析
4. 结论