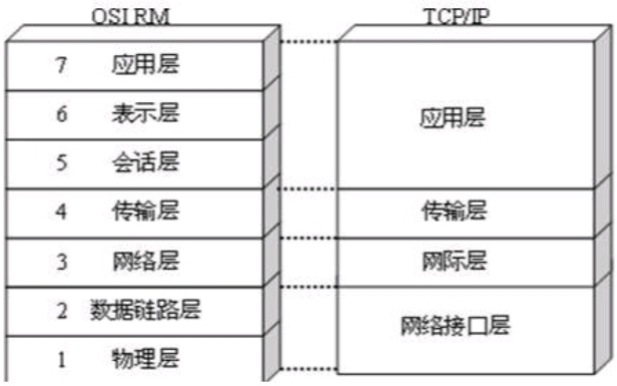
1.OSI/RM 参考模型没有成功的原因。

(1)OSI 的专家们缺乏实际经验，他们在完成 OSI 标准时缺乏商业驱动力。

(2)OSI 的协议实现起来过分复杂，而且运行效率很低。

(3)OSI 标准的制定周期太长，因而使得按 OSI 标准生产的设备无法及时进入市场，且 OSI 协议出现的晚，出现的时候已经有很好的 TCP/IP 协议。

(4)OSI 的层次划分不太合理，有些功能在多个层次中重复出现。



2．传输媒体的分类及每种类别中典型的传输媒体。

传输媒体是通信网络中发送方和接收方之间的物理通路。

传输媒体分为导向型传输媒体（电磁波沿着固体媒介传播）和非导向型传输媒体（传输信号沿自由空间传播）

(1）导向型传输媒体:

双绞线（分为屏蔽双绞线 STP 和非屏蔽双绞线 UTP)：由两根采用一定规则并排绞合，相互绝缘的铜导线组成。特点：便宜

同轴电缆：由导体铜质芯片、绝缘层、网状编制屏蔽层和塑料外层构成，分为 50Ω同轴电缆（基带同轴电缆）和 75Ω同轴电缆（宽带同轴电缆）特点：抗干扰性优于双绞线，传输距离更远，价格比双绞线贵

光纤：利用光脉冲进行通信，由纤芯（实心）和包层构成，分为单模光纤和多模光纤。特点：传输损耗小，抗雷电和电磁干扰性能好，体积小重量轻。

(2）非导向型传输媒体:

无线电波：较强穿透力，可传远距离，广泛用于通信领域。

微波：通信频率较高、频段范围宽，数据率很高，包括地面微博接力通信和卫星通信

红外线、激光：把要传输的信号分别转换为各自的信号格式，即红外信号和激光信号，再在空间中传播。

3．信息论中的三个基本问题及其内涵。

三个最基本的问题:

码元的传输速率是否有限制?

如何提高信息的传输速率?

信息传播过程是否存在局限性?

问题 1：码元传输速率有上限

奈氏准则：任何信道中，码元传输的速率是有上限的，传输速率超过此上限就会出现严重的码间串扰问题，就会识别不了了。

理想低通信道最高码元的传输速率：2W Baud

理想带通信道的最高码元传输速率：WBaud

信道带宽：W Baud

问题 2：提高信息传输速率

$$C = W\log_2(1+S/N)(bit/s)$$

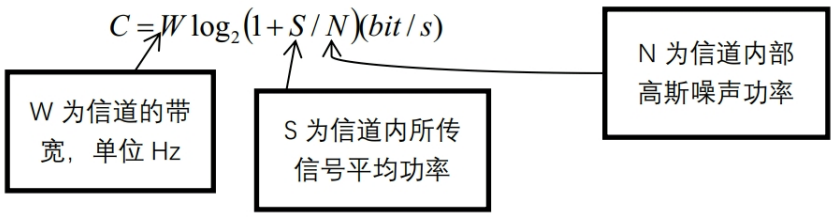
(1）增加信道带宽

(2）提高信道内所传信号的平均功率

(3）降低信道内部的噪声功率

问题 3：信息传输上限

香农公式-信道的极限传输速率：



|

表明：信道的带宽或者信道中的信噪比越大，信息的极限传输速率就越高，但是也是有上限的。实际上信息传输速率比香农的极限传输速率低不少。

注意：对于信道带宽已经确定的的信道，如果信噪比也不能再提高了，码元的传输速率也到达了上限值，我们还可以通过编码的方式让每个码元携带更多比特的信息量来增加传输速率。

4．脉码调制（PCM）的一般原理。

原理：脉冲编码调制就是把一个时间连续，取值连续的模拟信号变换成时间离散，取值离散的数字信号后在信道中传输。脉冲编码调制就是对模拟信号先抽样，再对样值幅度量化，编码的过程。

步骤：

主要包括采样、量化和编码三步。简单说就是模 / 数转换

采样：模拟信号是电平连续变化的信号。采样是一定的时间间隔，将模拟信号的电平幅度取出作为测量幅度的样本。

量化：将样本幅度按量化级取值的过程，量化后的为离散的量级值，已经不是连续值了。

编码：用相应位数的二进制代码表示量化采样值。

5.CDMA 的通信原理。

（1）可以感性的理解为：我只接受我所期望的信号类型，并且能够通过一定的方式将多个信号源分离出来。

(2) (Code Division Multiple Access）即码分多址，是一种信道复用技术，它允许每个用户在同一时刻同一信道上使用同一频带进行通信。

（3）原理：通信系统中，用户传输信息所用的信号不是靠频率不同或时隙不同来区分，而是用不同的编码序列来区分，即靠信号的不同波形来区分。

如果从频域或时域来观察，多个 CDMA 信号是互相重叠的。接收机在多个 CDMA 信号中选出其中使用预定码型的信号。其它使用不同码型的信号，因为和接收机本地产生的码型不同，而不能被解调。它们的存在类似于在信道中引入了噪声和干扰，通常称之为多址干扰。

6．信道监听策略的类型及内涵。

①1-persistent(1-持续型)：一直不停地检测，一旦检测到有空闲立马发送

②non-persistent（非持续型）：不定时检测，有空闲就发

③P-persistent(P-持续型)：以概率 P 检测信道是否空闲，如果不空闲，以 q(1-P)的概率确定下次检测时间。

7．网络层设计的四个基本问题。

①存储转发的分组交换方式有哪些。

②究竟给传输层提供什么类型的服务，是否应该把网络层和运输层独立。

③在设计提供服务时，服务与路由器的数目、类型、拓扑有没有关联。

④网络地址是否应该使用统一的规范

1．网络层设计的第一个基本问题是网络拓扑结构，也就是如何把网络中的节点连接起来。

2．网络层设计的第二个基本问题是网络地址，也就是如何把网络中的节点分配不同的地址，以便节点之间进行通信。

3．网络层设计的第三个基本问题是路由，也就是如何把数据从一个节点发送到另一个节点。

4．网络层设计的第四个基本问题是流量控制和拥塞控制，也就是如何确保网络中的数据流不会因为网络拥塞而受到影响。

8．路由算法的分类。

分类：静态与动态、单路径与多路径、平坦与分层、主机智能与路由器智能、域内与域间、链接状态与距离向量

（1）静态：不能根据网络流量和拓扑结构的变化来调整自身的路由表，必须由管理员手工设定的。路由改变随时间变化比较慢。

（2）动态：定期对路线进行更新，响应连接成本是变化的。代表：距离向量路由选择算法(RIP 协议），OSPF

（3）单路径和多路径：一些复杂的路由协议支持到同一目的的多条路径。与单路径算法不同，多路径算法允许数据在多条线路上复用，提供更好的吞吐量和可靠性。

9．路由算法设计的挑战性问题

(1）规模可扩展问题

多种因素导致核心网络的路由表超线性的快速增加

(2）性能可扩展问题

当前域间路由系统存在收敛缓慢，故障后难以快速恢复等问题

(3）安全可扩展问题

路由协议容易受到攻击，存在安全问题

(4）服务可扩展问题

不能提供保证服务质量的路由服务

(5)功能可扩展问题

组播等功能无法在网络上大规模应用

10．通过冗余实现可靠性的分类和方法。

可靠模型：冗余（空间冗余、时间冗余）

①空间冗余：独立备份副本一正向纠错（FEC）代码，需要巨大的开销

②时间冗余：重新发送如果数据包丢失 / 错误-利用响应时间换取可靠性

(1） 空间冗余（向前纠错码：需要巨大开销）

(2）时间冗余（停止等待协议、超时重发）机制

(3）包校验和：保证数据不错

(4)ACK/NAK：保证包不丢、不乱、不多

1．数据冗余：将同一数据多次存储在不同的位置，以防止数据丢失。

2．资源冗余：在系统中额外提供一些备用资源，以应对系统故障。

3．半冗余：在某些情况下，部分资源可以在需要时动态切换，以保证系统的可靠性。

4．全冗余：在系统中额外提供所有资源的备份，以应对任何故障。

5．双重冗余：在系统中同时使用数据冗余和资源冗余，以保证系统的可靠性。

6．负载平衡：在系统中使用资源冗余和负载平衡技术，以保证系统的可靠性。

7．集群技术：使用集群技术对系统进行冗余配置，以保证系统的可靠性。

11．流量控制的分类与一般原理。

流量控制原理：接收方来控制发送方的流量，让发送方的发送速率不要太快，既要让接收方来得及接收，也不要使网络发生拥塞。

流量控制的分类：

（1）利用滑动窗口来实现流量控制，发送方根据双方协商的窗口尺寸发送数据。每发送完一个帧就停止发送，等待对方的确认，在收到确认之后，根据确认的信息改变窗口尺寸，再发送下一帧。

（2）漏斗协议，发送方自己控制流量，主要目的是控制数据注入到网络的速率，平滑网络上的突发流量。

（3）令牌桶：用来控制发送到网络上的数据的数目，并允许突发数据的发送。

12．拥塞控制的静态解决方案。

1、增加缓存空间　　--缓冲区预分配法

2、提高处理机的处理速率　--分组丢弃法

3、增加链路容量或者更换链路　　---定额控制法

拥塞控制发生在网络层、数据链路层和传输层

1．固定的带宽限制：通过限制网络的带宽使用，避免拥塞的发生。

2．固定的排队策略：通过使用固定的排队策略，例如先进先出，以避免拥塞的发生。

3．固定的路由策略：通过使用固定的路由策略，例如最短路径，以避免拥塞的发生。

4．资源分配：通过明确分配网络中的资源，例如带宽，以避免拥塞的发生。

1.固定窗口大小：这种方法通过设置固定的窗口大小，以限制网络中的数据流量，从而避免网络拥塞。

13．拥塞控制的原理与算法。

拥塞控制的原理是：当网络中的数据流量超过网络的容量时，就会出现拥塞，从而导致数据丢失或延迟。为了避免这种情况，需要采用拥塞控制算法来检测和控制网络中的数据流量，以确保数据能够按照最优的路径发送。

常用的拥塞控制算法包括：

拥塞控制算法分为静态和动态两种：

1．静态拥塞控制：是指不随网络状况变化而变化的控制方法，常见的静态解决方案有固定的带宽限制，固定的排队策略，固定的路由策略和资源分配。

2．动态拥塞控制：是指随着网络状况变化而动态变化的控制方法，常见的动态拥塞控制算法有慢启动算法，拥塞避免算法，快速恢复算法等。

慢启动算法：这种算法通过逐步增加窗口大小，以检测网络的容量，并有效地控制拥塞。

拥塞避免算法：这种算法通过逐步减小窗口大小，以检测网络的容量，并有效地控制拥塞。

快速重传算法：这种算法通过重新发送丢失的数据，以确保数据能够准确地接收。

14．一般的网络攻击手段。

1、欺骗类攻击：欺骗类攻击的手段主要包括口令欺骗、IP 地址欺骗、ARP 欺骗、DNS 欺骗与源路由欺骗等。

2、拒绝服务 / 分布式拒绝服务类攻击：拒绝服务攻击与分布式拒绝服务攻击的手段 z 主要包括资源消耗型、修改配置型、物理破坏型与服务利用型等

3、信息收集类攻击：信息收集类攻击的手段主要包括扫描攻击、体系结构探测攻击、利用服务攻击等

4、漏洞类攻击：漏洞类攻击的手段包括网络协议类、操作系统类、应用软件类与数据库类。

15.网络安全的目标及其内涵。

基本目标

保密性：信息机密性：确保网络中的敏感信息不被未经授权的第三方获取。这包括个人隐私信息、企业商业机密、国家机密等。

隐私性：个人仅可以控制和影响与之相关的信息，确保个人隐私不被非法泄露或滥用。

完整性：确保网络中的数据在传输、存储和处理过程中不被篡改、删除、伪造、乱序、重放或插入。这有助于维护数据的真实性和可靠性。

可用性：确保网络服务能够连续可靠地运行，不被恶意行为所干扰。这包括确保授权用户能够按需访问和使用网络资源和数据。

可控性：仅允许实体以明确定义的方式对访问权限内的资源进行访问。这有助于防止未经授权的访问和操作，确保网络资源的合法使用。

可审查性：要求网络中的操作和活动能够被追踪和审查，以便于发现和处理潜在的安全问题。这有助于确保通信的所有参与者都不能否认曾经完成的操作，维护网络环境的公平和诚信。

网络安全的内涵

保护关键信息基础设施：关键信息基础设施是国家网络安全的重要组成部分，其安全稳定直接关系到国家安全、社会公共利益以及公民、法人和其他组织的合法权益。因此，保护关键信息基础设施是网络安全的重要任务之一。

维护国家数据安全与隐私保护：国家采取了一系列措施来维护数据安全和隐私保护，包括建立健全数据安全治理体系法规、制定数据安全标准、落实各部门职责、建立数据安全风险评估机制、实施数据安全监测预警等。

防范网络攻击与威胁：网络攻击和威胁是网络安全面临的主要挑战之一。为了防范这些攻击和威胁，国家采取了多种措施，包括提高网络安全意识、加强密码管理、谨慎使用公共网络等。同时，还积极推动网络安全技术

和标准的提升，以应对不断变化的网络威胁。

促进网络安全技术与标准的提升：网络安全技术的发展和标准的制定对于提高网络安全水平具有重要意义。国家加大了对网络安全技术研发的投入，鼓励企业加强自主创新，提高技术水平和核心竞争力。同时，还积极参与国际网络安全标准和规范的制定和推广，提升我国在国际网络安全领域的话语权和影响力。

完善网络安全法律法规与政策支持：法律法规和政策支持是保障网络安全的重要基础。国家制定了一系列网络安全相关的法律法规和政策文件，为网络安全的保护提供了明确的法律依据和制度保障。同时，还通过政策支持来推动网络安全的保护和发展，如建立和完善网络安全标准体系、推进网络安全社会化服务体系建设等。

16.IGMP 协议的不同版本及其主要区别

IGMPv1：定义了基本的组成员查询和报告过程，主要基于查询和响应机制来完成对组播组成员的管理。

没有定义离开组播组的报文，路由器基于成员组的超时机制发现离线的组成员。

支持普遍组查询报文和成员关系报告报文，但不支持特定组查询报文和成员离开报文。

IGMPv2：在 IGMPv1 的基础上增加了查询器选举和组成员离开的机制。

支持普遍组查询报文、成员报告报文和成员离开报文。

增加了对特定组的查询，以及最大响应时间字段，以动态地调整主机对组查询报文的响应时间。

支持特定组查询报文，避免了属于其他组播组成员的主机发送响应报文。

IGMPv3：支持源特定多播（Source-Specific Multicast，SSM），允许主机指定它想要接收的多播源，进一步提升了多播流量的控制与管理。

没有定义专门的成员离开报文，成员离开通过特定类型的报告报文来传达。

支持普遍组查询报文、特定组查询报文和特定源组查询报文（Group-and-Source-Specific Query）。

一个成员报告报文可以携带多个组播组信息，而之前的版本一个成员报告只能携带一个组播组。

可以直接应用于 SSM 模型，而 IGMPv1 和 IGMPv2 则需要 IGMP SSM Mapping 技术的支持。

IGMP 协议的三个版本在报文类型、特点和功能等方面存在显著差异

17.无线网络的主要类型

按覆盖范围分类:

个人区域网络(PAN):覆盖范围一般在 5 到 10 米范围内。

无线局域网(WLAN):覆盖范围通常小于 100 米。

城域网(MAN):覆盖范围通常大于 100 米，覆盖一个城市或城镇的多个建筑。

广域网(WAN):覆盖范围通常大于 1 千米，可以跨越国家、大陆甚至整个地球。

局域网(LAN):由一组设备组成，如个人计算机、服务器等，最常见的类型是以太网。

校园区域网络（CAN，Campus Area Network）:覆盖范围通常在校园区域网络中，将两个或者多个局域网连接在一起。

存储区域网络（SAN):一种特殊的高速网络，用于存储大量数据并提供对它们的访问。

18.WSNs 的主要安全威胁

WSNs（无线传感器网络）的主要安全威胁包括多个方面:

1、物理层面的安全威胁

传感节点物理操纵：由于传感节点通常部署在无人维护、不可控制的环境中，攻击者可以轻易捕捉到传感节点，他们可以分析出传感节点所存储的程序代码、路由协议及密钥等机密信息，并加载恶意程序代码到传感节点中，从而控制部分网络。

节点资源受限：传感节点的内存资源有限，使得在传感器网络中实现大多数节点间端到端安全不切实际。攻击者可以利用这一特点，通过操纵节点发送虚假路由消息，影响整个网络的路由拓扑。

2、网络层面的安全威胁

窃听攻击：由于无线电传播的广播性质，无线空中接口是开放的，使得攻击者可以在通信网路覆盖范围内窃听合法用户之间的数据传输。

DoS 攻击：DoS 攻击（拒绝服务攻击）旨在破坏网络的可用性。在 WSNs 中，DoS 攻击可能通过恶意节点产生故意干扰，以破坏合法用户之间的数据通信。

MITM 攻击： 这种攻击方式可以窃取敏感信息或注入恶意数据。

网络注入攻击：网络注入攻击旨在通过注入伪造的网络重新配置命令来干扰网络设备的操作。

3、数据层面的安全威胁

数据泄露：攻击者可以通过窃听、加入伪造的非法节点等方式获取 WSNs 中的敏感信息。

数据篡改：攻击者可以修改存储在传感节点中的信息或代码，从而伪造或伪装成合法节点加入到传感网络中。

4、其他安全威胁

密钥泄露：传感节点中的密钥信息容易被攻击者获取，从而威胁整个网络的安全性。

节点身份伪造：攻击者可以通过获取存储在传感节点中的密钥、代码等信息，伪造或伪装成合法节点加入到传感网络中。这将严重威胁网络的完整性和可用性。

19.物联网的层次架构及每层的主要功能

物联网的体系结构主要包括五个层次：感知层、网络层、中间层、应用层和业务层。每个层次的功能不同，它们共同构成了物联网的完整体系。

感知层：是物联网的最底层，主要负责采集来自各种传感器的数据，采用多种技术进行数据处理，如图像识别、语音识别等，将处理后的数据传输到网络层。

网络层：是建立在传感器网络之上的，将传感器采集的数据集合起来传输，并与外界网络进行互联。常用的技术包括低功耗无线通信技术、蓝牙、ZigBee 等。

中间层：是物联网的架构核心，它是数据处理和数据传输的枢纽。中间层主要分为数据处理层和数据传输层。数据处理层主要对采集的数据进行数据清洗、挖掘等操作；数据传输层将经过处理的数据发送到应用层。

应用层：是物联网的用户界面，也是实现物联网应用的关键。应用层主要包括云计算、移动应用、Web 应用等多种形式，用户可以通过应用层实现对物联网设备的控制和数据的查询。

业务层：是物联网最高层，它掌握着整个物联网的规划、设计和管理。它通过对物联网设备的调度和资源分配，实现物联网的安全、可靠和高效运行。

20.传感网络路由协议的主要类型

（1）能量感知路由协议、基于查询的路由协议、地理位置协议、可靠的路由协议。

（2）能量感知路由协议：从数据传输的能量消耗出发，讨论最少能量消耗和最长网络生存期等问题。

（3）基于查询的路由协议：主要用于需要不断查询传感器结点采集的数据，通过减少通信流量来节省能量，即数据融合技术与路由协议的设计相结合。

（4）地理位置协议：主要应用于需要知道目的结点的精确或大致地理位置的问题中，把结点的位置信息作为路由选择的依据，从而完成结点的路由选择功能，并且降低维护路由协议的能耗。

（5）可靠的路由协议：应用在对可靠性和实时性等方面有特别要求的问题中。

21.用源代码进行 NS-3 编译、安装与测试的基本过程。

1．安装库

```
sudo apt-get update
sudo apt install gcc g++ python
sudo apt install gcc g++ python python-dev
sudo apt install mercurial
```

2．编译安装

```
mkdir NS3       # 创建 NS3 目录
cd NS3
wget wget https://www.nsnam.org/releases/ns-allinone-3.35tar.bz2   #下载安装包
tar xvf ns-allinone-3.35.tar.bz2  # 解压
cd ns-allinone-3.35/
./build.py      #  看到'build' finished successfully... 就表示安装成功
cd ns-3.35/
./waf distclean    # 清除整个 build 目录
./waf configure--enable-examples--enable-tests     # 开启例子及帮助
./wafbuild
```

3．测试

```
./test.py    # 看到一堆 PASS 就证明安装成功
```

.test.py-c core　# 看到一堆 PASS 就证明安装成功

22．使用 NS-3 进行网络仿真的基本流程。

1）选择或开发相应的模块：根据实际仿真对象和仿真场景选择相应的仿真模块：如是有线局域网络（CSMA）还是无线局域网络（Wi-Fi）等。

2）编写网络仿真脚本：C++或者 Python

编写脚本过程：

（1）生成节点（如网卡、应用程序、协议栈等）

（2）安装网络设备（如 CSMA、WiFi）

（3）安装协议栈：ns-3 一般是 TCP/IP 协议栈

（4）安装应用层协议：依据选择的传输层协议选择相应的应用层协议

（5）其他配置（如节点是否要移动，是否要能量管理）

（6）启动仿真

3）仿真结果分析。一种是网络场景，另外一种是网络数据。

4）依据仿真结果调整网络配置参数或修改源代码

23.常用的 NS-3 仿真结果分析工具及其主要作用。

1.TcpDump

TcpDump(dump the traffic on a network）是 Linux 中强大的网络数据采集分析工具只要，根据使用者的定义对网络上的数据分组进行截获的分组分析工具。TcpDump 可以将网络中传送的数据分组的"头"完全截获下来提供分析。它支持针对网络层，协议，主机，网络或端口的过滤，并提供 and,or,not 等逻辑语句来帮助去掉无用的信息。

2.Wireshark

与 TcpDump 相比，Wireshark 界面比较友好，功能也很强大。Wireshark 是一个图形用户界面的网络分组封装分析软件，可以获取网络分组封装，并尽可能地显示最为详细的网络分组封装资料。

1.Gnuplot：这种工具可以用于绘制图表，以帮助用户更好地理解仿真结果。

2.Flowmonitor：这种工具可以用于跟踪数据流，以检测网络的性能。

3. Trace-analizer：这种工具可以用于分析数据包轨迹，以确定网络中的拥塞状况。

4.NAM：这种工具可以用于生成动画，以帮助用户更好地理解网络中的物理连接。

24.NS-3 中常用的核心概念（网络术语）及其含义。

1）结点 Node（ns3 将基本的计算设备抽象为 Node（具体表现为 C++的 Node 类））

在因特网术语中，连接到网络的计算设备称为主机或终端系统。由于 ns-3 是网络模拟器，而不是特定的 Internet 模拟器，为此我们避开术语"主机"，因为它与 Internet 及其协议密切相关。因此，我们选用了一个来源于图论，在其他网络模拟器中亦广泛使用的术语：节点。

ns-3 中基本计算设备被抽象为节点。节点由用 C++编写的 Node 类来描述。Node 类提供了用于管理计算设备的各种方法。

2）应用程序 Application（网络的数据的发起方和接收方，其为运行在 Node 内部的用户软件）

计算机软件通常可分为两大类：系统软件和应用软件。系统软件根据某种计算模型组织各种计算机资源，如内存，处理器周期，磁盘，网络等。用户通常运行应用程序获取并使用由系统软件控制的资源来实现某个目标。

在 ns-3 中，需要被仿真的用户程序被抽象为应用。这个抽象在 C++中由类 Application 表示。Application 类提供了在模拟中管理我们的用户级应用程序版本的表示的方法。

3）信道 Channel（数据传输的通道（可以视为通信的子网或者网络））

在现实世界中，人们可以将计算机连接到网络，数据在这些网络中流动的媒体通常称为信道。在 ns-3 的模拟世界中，可以把节点连接到代表数据交换信道的对象上。这里基本的通信子网这一抽象概念称为信道，并由 C++类 Channel 表示。

Channel 类提供了管理通信子网对象和把节点连接至信道的各种方法。信道类同样可以由开发者以面向对象的方法自定义。

4）网络设备 Net Device

如果想把一台计算机连接到网络上，必须在计算机上安装有网卡。一张网卡如果缺少控制硬件的软件驱动是不能工作的。在 Unix/Linux 系统中，外围硬件被划为"设备"。设备通过驱动程序来控制，而网卡通过网卡驱动程序来控制。在 Unix/Linux 系统中，网卡被称为像 eth0 这样的名字。

在 NS3 中，网络设备这一抽象概念相当于硬件设备和软件驱动的总和。NS3 仿真环境中，网络设备相当于安装在节点上，使得节点通过信道和其他节点通信。

5）拓扑助手 Topology Helpers（简化网络拓扑配置，将那些机械的重复性的配置操作用 Helper 实现）

在大型模拟网络中，需要在 Node、NetDevices 和 Channels 之间安排许多连接。由于将 NetDevices 连接到节点、将 NetDevices 连接到通道、分配 IP 地址等都是 ns-3 中的常见任务，因此我们提供了所谓的拓扑辅助工具，以使其尽可能简单。ns-3 提供拓扑辅助对象，将这些许多不同的操作组合成一个易于使用的模型，以方便使用。

{1.Node：计算设备的抽象

2.NodeContainer：保存 Node 节点的容器，用于保存和跟踪节点，即快速访问某节点，进而对节点进行操作

3.NodeList:创建节点会加入到 NodeList 中进行管理

4.Channel：实现了通讯子网的框架，具体通讯方式的实现是由子类完成的

5.NetDevice：几乎每一种连接的网络 Channel 有相应的 NetDevice。

6.NetDeviceContainer：存储 NetDevice，用于快速访问

7. Application：安装在网络节点上的应用程序

8. ApplicationContainer：存储 Node 中的 Application，用于快速访问。}

25.对 NS-3 仿真脚本进行调整（Tweaking）的三种方式。

1．命令行：这种方式可以通过命令行编辑器进行调整或跟踪，以更改仿真脚本的参数。

2．图形用户界面：这种方式可以通过图形用户界面调整或跟踪，以更改仿真脚本的参数。

3．脚本编辑器：这种方式可以通过脚本编辑器进行调整或跟踪，以更改仿真脚本的参数。

26.Tracing 的优点：

1、提高可观测性

Tracing 能够收集关于系统执行的详细数据，包括底层内核事件（如调度程序上下文切换、线程唤醒、系统调用等）和应用程序代码的执行情况（如运行的函数、每次调用的耗时、函数调用的参数等）。这些数据使得开发人员能够更深入地了解系统的运行状态，从而提高系统的可观测性。

2、结构化日志记录

Tracing 提供了一种结构化的日志记录方式，与传统的打印语句调试方式相比，它更具条理性和可读性。结构化的日志记录有助于开发人员更快地定位问题，因为相关信息被组织得更加清晰。

3、 支持异步操作跟踪

在复杂的异步应用程序中，传统的调试方式往往效率低下，难以掌握整个执行流程。

Tracing 技术能够追踪异步操作的执行过程，并收集关键的上下文信息，从而帮助开发人员更好地理解异步任务的执行过程。

4、 性能优化

Tracing 是性能打点的一种优雅实现，能够以瀑布流或火焰图的形式呈现系统的耗时分布，使开发人员能够直观地看到性能瓶颈所在。

通过分析 Tracing 数据，开发人员可以识别出哪些部分消耗了最多的资源（如 CPU、内存、网络等），从而有针对性地进行优化。

5、灵活配置与低开销

Tracing 框架（如 Perfetto）提供了灵活的配置选项，允许开发人员根据需要调整收集的数据类型、收集方式以及跟踪的详细程度。

同时，现代 Tracing 框架足够快，可以在纳秒级完成测量，而不会显著影响程序的执行速度，从而确保了低开销的监控。

6、易于集成与可视化

Tracing 技术通常易于集成到现有的开发环境和工具链中，如 Chrome 开发者工具、Flutter 等。

27.Tracing System 中两种不同跟踪方式的特性。

基于 NS3 的 Tracing 方式：NS3 的 tracing 系统建立在独立的 tracing 发送端和接收端概念上，有统一的机制来连接发送端和接收端。Trace 发送端可以在仿真过程中产生信号事件，并提供有关数据访问通道。

输出格式：NS3 提供两种 tracing 机制，即 ASCII 码 tracing 和 pcap 级别的 tracing。分布式系统中的链路追踪方式：链路追踪（Tracing）是一种用于分析分布式系统中请求处理流程的方法，其特性如下：

1.组件与概念：

Span：表示一个独立的工作单元。

Trace：表示一个请求在分布式系统中的传播路径，由多个 Span 组成，对应一个请求的生命周期。

Annotation（注解）：用于记录事件的发生时间和相关信息，如请求的开始、结束、异常等。

TraceTree（跟踪树）：以树状结构展示 Trace 中各个 Span 之间的关系，便于分析请求在系统中的传播路径。

2.数据处理与展示：

数据采集：需要对系统中的各个组件（如前端、后端、数据库等）进行数据采集，可以通过手动埋点、字节码增强、日志解析等方式实现。

数据存储：链路追踪产生的数据量较大，需要选择合适的存储方案，如关系型数据库、时序数据库、分布式日志存储等。

数据处理：对采集到的数据进行处理，提取出有用的信息，如请求耗时、异常信息等，可以采用实时计算和离线计算两种方式。

数据展示：将处理后的数据以可视化的方式展示给用户，便于用户分析系统性能和定位故障，常见的可视化工具包括 Grafana、Kibana 等。

3.技术选型与部署：根据实际业务需求和现有系统架构，选择合适的链路追踪技术，如 Zipkin、Jaeger、SkyWalking 等开源系统。将选定的链路追踪系统与现有系统集成，实现对各个组件的数据采集，并根据系统架构选择合适的部署方式。

4.性能与安全性：链路追踪会对系统性能产生一定影响，需要针对具体场景进行性能优化，如采用异步上报、采样、压缩等方式。

5.确保链路追踪系统的安全性和合规性，防止数据泄露和滥用，如进行敏感数据脱敏处理、设置合理的权限控制等。

28.NS-3 中的 Log 消息的 7 种级别

NS-3 中的 Log 消息有 7 种级别，按照详细程度从低到高分别是：

1. LOG_ERROR：记录错误信息，使用宏 `NS_LOG_ERROR`。
2. LOG_WARN：记录警告信息，使用宏 `NS_LOG_WARN`。
3. LOG_DEBUG：记录相对罕见、特别的调试消息，使用宏 `NS_LOG_DEBUG`。
4. LOG_INFO：记录关于程序进程的信息消息，使用宏 `NS_LOG_INFO`。
5. LOG_FUNCTION：记录调用的每个函数，使用宏 `NS_LOG_FUNCTION` 和 `NS_LOG_FUNCTION_NOARGS`。
6. LOG_LOGIC：记录描述函数内逻辑流的消息，使用宏 `NS_LOG_LOGIC`。
7. LOG_ALL：记录上述所有级别的信息，没有特定的宏与之对应。

此外，还有一个特殊的日志级别：NS_LOG_UNCOND：无条件地记录关联的消息，无论日志级别或组件选择如何，都会显示，使用宏 `NS_LOG_UNCOND`。

这些日志级别可以帮助用户根据不同的需求和场景来控制日志的输出。

29.虚拟机技术与容器技术特性对比。

虚拟机技术与容器技术是当前虚拟化领域的两大主流技术，它们在实现方式、性能、资源利用、可移植性、安全性等方面各有特点。以下是对这两种技术特性的详细对比

1.实现方式：

虚拟机是通过软件仿真生成的独立计算机环境，能够运行操作系统和应用程序，类似于物理计算机。每个虚拟机可以安装自己的操作系统，运行独立的应用程序。容器是一种轻量级的虚拟化技术，它通过操作系统级的虚拟化，将应用程序及其依赖环境打包在一起，确保应用程序可以在任何环境中一致运行。容器使用宿主操作系统的内核，而不是虚拟化整个操作系统。

2.性能与资源利用：

虚拟机需要仿真完整的硬件环境，并运行一个完整的操作系统，因此资源利用率相对较低，性能开销较大。虚拟机监控器在分配和管理资源时会引入额外的开销，导致资源利用率相对较低。虚拟机的启动时间较长，因为它们需要加载和启动一个完整的操作系统。容器不需要虚拟化整个操作系统，资源利用率高，启动速度快。容器通过直接利用宿主操作系统的资源，降低了资源消耗和延迟，从而提升了整体性能。容器的启动时间通常非常短，接近即时，因为容器不需要启动完整的操作系统。

3.可移植性与隔离性：

虚拟机提供了完整的隔离性，因为每个虚拟机都运行在独立的虚拟机监控器上，并拥有自己的操作系统内核。虚拟机可以运行不同操作系统的应用程序，具有高度的可移植性。容器虽然提供了一定程度的隔离，但与虚拟机相比，隔离性较弱。容器共享宿主操作系统的内核，因此如果宿主操作系统存在漏洞，容器可能受到威胁。容器可以在任何支持容器运行时的环境中运行，无论是本地开发环境还是云端生产环境，因此也具有一定的可移植性。

4.安全性：

虚拟机之间的隔离性更好，因此虚拟机通常更安全。虚拟机可以提供各种安全方案，如针对不同用户登录的不同虚拟机等。由于容器共享宿主操作系统的内核，因此安全性相对较弱。如果宿主操作系统存在漏洞，容器可能受到威胁。容器需要额外的安全措施来确保安全性，如使用安全沙箱、限制容器权限等。

5.管理复杂性：

虚拟机管理可能需要更多的手动干预，因为每个虚拟机都有自己的操作系统和应用程序。

虚拟机技术已经发展多年，拥有成熟的管理工具和生态系统，支持企业级应用的部署和管理。容器通常更容易管理，因为它们可以轻松扩展和更新。容器与持续集成和持续部署（CI/CD）工具集成紧密，使得开发团队能够更容易地实现自动化构建、测试和部署流程。

30.什么是 Kubernetes 中的 Service？它的作用是什么？

Kubernetes 中的 Service 是一种资源对象，它定义了一组 Pod 的逻辑集合和访问它们的规则。Service 为这些 Pods 提供了一个稳定的访问入口，使得其他应用程序或客户端可以通过这个入口来访问这组 Pods 提供的服务。

Service 在 Kubernetes 中是一个抽象层，它代表了一组运行中的 Pods，并为它们提供了一个统一的访问地址。这个地址通常是一个虚拟 IP 地址（ClusterIP）和一个 DNS 名称。通过 Service，客户端可以访问到一组具有相同标签（label）的 Pods，而无需关心这些 Pods 的具体数量、IP 地址或运行状态。

Service 的作用

服务发现：Service 为 Pods 提供了一个稳定的访问入口，即虚拟 IP 地址和 DNS 名称。客户端可以通过这个入口来发现和访问 Pods 提供的服务，而无需关心 Pods 的具体位置或状态。

负载均衡：Service 可以将来自客户端的请求分发到后端的多个 Pods 上，实现负载均衡。这有助于提高应用的可扩展性和性能，因为请求可以被均匀地分发到多个 Pods 上进行处理。

动态更新：当 Pods 的数量或状态发生变化时（如 Pod 的添加、删除或故障），Service 会自动更新其后端 Pods 的列表。这意味着客户端无需关心 Pods 的具体变化，只需要通过 Service 的虚拟 IP 地址或 DNS 名称来访问服务即可。

跨命名空间访问：Service 可以轻松地将不同命名空间中的 Pods 组合到一个逻辑服务中，从而实现跨命名空间的访问。

网络策略：通过配置 Service 的网络策略，可以限制访问 Service 的客户端的 IP 范围或来源等条件，增强了安全性。

31.Kubernetes 中的 Pod 是什么？它的作用是什么？

Pod 代表了在 Kubernetes 集群上运行的一个进程，它是 Kubernetes 调度的基本单位。一个 Pod 可以包含一个或多个容器，这些容器在同一个环境中共同工作，共享网络和存储资源。Pod 中的容器紧密协作，形成一个完整的功能单元。

Pod 的作用封装应用程序实例：

Pod 是应用程序的逻辑单元，可以封装单个应用程序容器，也可以将多个容器组合在一起。通过封装应用，Pod 提供了对容器生命周期和资源的统一管理方式。

资源共享：Pod 中的所有容器共享相同的网络命名空间，拥有相同的 IP 地址和端口空间。它们可以通过 localhost 直接相互通信，简化了容器之间的网络配置。Pod 还可以定义持久存储卷（Volume），并将其挂载到

Pod 中的容器上，多个容器可以共享这些卷，用于存储数据或共享状态。

应用高可用性：Pod 的生命周期是短暂的，可以随时被终止、重新调度或替换。当某个 Pod 失败时，Kubernetes 控制器（如 ReplicaSet、Deployment 等）可以自动创建新的 Pod 实例，以确保应用程序的高可用性。

简化可伸缩性：Pod 可以配合 Kubernetes 的控制器实现自动扩展，根据需求变化自动创建和关闭副本 Pod。

32.什么是 Kubernetes？它的主要特点是什么？

Kubernetes（通常简称为 K8s）是一个开源的容器编排平台，用于自动化部署、扩展和管理容器化应用程序。Kubernetes 旨在提供一个平台，以便在私有云、公有云或混合云环境中运行分布式系统。Kubernetes 是一个强大的容器编排工具，它允许用户以声明式的方式定义和运行容器化应用程序。通过 Kubernetes，用户可以自动化地部署、扩展和管理容器，从而简化应用程序的部署和管理过程，提高应用程序的可靠性和可扩展性

Kubernetes 的主要特点

容器编排：Kubernetes 能够自动化地部署、扩展和运行容器。

服务发现和负载均衡：通过 Service 对象，Kubernetes 可以自动发现新部署的服务，并通过内置的负载均衡器分配网络流量，确保应用程序的稳定性和性能。

自动化滚动更新和回滚：Kubernetes 支持应用程序的无缝更新。

自我修复：Kubernetes 能够监控容器的健康状态，并在容器出现故障时自动重启或替换容器。这种自我修复能力有助于确保应用程序的高可用性和可靠性。

存储编排：Kubernetes 提供了灵活的存储管理功能。它可以自动挂载所选的存储系统，无论是本地存储、公有云提供商的存储服务还是网络存储系统，从而满足应用程序的存储需求。

跨主机和跨云平台：这使得用户可以在不同的环境中部署和管理容器化应用程序，提高了应用程序的灵活性和可扩展性。

丰富的生态系统：围绕 Kubernetes 已经形成了众多工具、插件和解决方案，这些工具可以帮助用户更好地管理和优化容器化应用程序。

高可用性：通过多个副本、自动扩展和故障转移等机制，Kubernetes 可以确保应用程序在出现单点故障时仍然能够正常运行。

安全性：Kubernetes 提供了多种安全功能，如网络策略、身份认证和授权等。

33.描述 KubeEdge 的核心组件及其作用。

Edged：在边缘节点上运行并管理容器化应用程序的代理。

EdgeHub：Web 套接字客户端，负责与云端服务进行交互以进行边缘计算。这包括将云端资源更新同步到边缘，并将边缘侧主机和设备状态变更报告给云端。

CloudHub：Web 套接字服务器，负责在云端缓存信息、监视变更，并向 EdgeHub 端发送消息。

EdgeController：Kubernetes 的扩展控制器，用于管理边缘节点和 pod 的元数据，以便可以将数据定位到对应的边缘节点。

EventBus：一个与 MQTT 服务器（mosquitto）进行交互的 MQTT 客户端，为其他组件提供发布和订阅功能。

DeviceTwin：负责存储设备状态并将设备状态同步到云端。它还为应用程序提供查询接口。

MetaManager：Edged 端和 Edgehub 端之间的消息处理器。它还负责将元数据存储到轻量级数据库（SQLite）或从轻量级数据库（SQLite）检索元数据。