1. **背景介绍**

1.1 嵌入式系统定义与特征

嵌入式系统作为现代计算的核心组成部分，被广泛应用于工业控制、物联网、汽车电子和医疗设备等领域。从定义上，嵌入式系统是以应用为中心，去满足用户需求的控制操作机器和设备的计算机系统。其有几种特征区别于通用计算系统，而这些特征也使得本次研究的内容有意义和价值。

首先是资源约束性：嵌入式系统应用的设备通常只有很有限的内存、处理器频率以及能源供应，而这样的性能也就使得设计者需要在在安全方面与执行效率间权衡，开发者需要将资源利用率提升为设计的第一性原则。以网络协议栈为例，传统七层模型在嵌入式场景中就面临着性能问题，需要进行结构上的删减，如这次参考使用的smoltcp协议栈剔除了会话层、表示层等非必要层级，将核心功能聚焦于数据传输与路由，来提高代码的执行效率。以固定窗口替代动态拥塞控制算法，牺牲了带宽的自适应能力，使单数据包处理延迟更加可预测和稳定。还进行了内存固化，通过预分配通信缓冲区而不是在运行时动态申请，来消除内存碎片化风险，确保中断响应时间的确定性。

其次是实时性要求：工业控制等场景需要毫秒级响应，而许多安全校验的时间开销直接影响系统确定性，这也同样影响了各种协议的设计方案。

最后是长生命周期：嵌入式设备的系统通常需要在没有人员进行维护的情况下进行长时间的运行，如汽车电子有10-15年服役期要求，而这些要求也就需要系统具有充分的可靠性保证。

这些要求使得针对其的网络协议栈需要特殊设计，而不能套用传统的网络协议栈模式。

2.sel4介绍以及为什么要用rust重写为rel4

seL4是L4微内核模型基础上设计出来的全球首个通过形式化验证的操作系统内核，其中L4微内核模型是一种高度精简的操作系统内核架构，其核心设计理念是最小化内核功能，仅提供最基本的机制，而将其他功能移至用户空间实现。这种设计使得L4内核具有极高的模块化、灵活性和安全性，同时也为高性能奠定了基础。sel4自 2004 年开始开发，并于2014年实现开源。它采用微内核架构，其源于L4内核的各特性保证了其作为嵌入式系统内核的广泛应用场景。

首先是安全性：sel4通过比传统linux更为严格的能力机制实现安全高效的访问控制，从而提供了细粒度的资源管理能力。此机制使得任务能够严格遵循最小权限原则，显著降低了恶意操作和意外错误的风险。其能力机制设计实现了资源抽象与令牌化，也就是所有系统资源（如内存页、设备寄存器、线程对象）均被抽象为不可伪造的能力令牌。任何操作必须显式持有对应能力令牌，并符合其权限标签（如读、写、执行）。最小特权原则使得其内核启动时通过显式授权链初始化能力空间。例如，仅允许特定线程访问指定的内存区域：

安全效果验证：  
在DARPA高保障网络军事系统（HACMS）项目中，搭载seL4的无人机导航系统成功抵御了所有渗透测试攻击，攻击面相比传统实时操作系统（RTOS）减少92%。能力模型通过数学机制将安全策略从“经验性防御”提升为“确定性保障”。

其次是性能优化：尽管微内核设计的初衷并非追求处理速度，但sel4为了应对现实使用的需要，实现了成为处理速度最快的微内核。

最后是形式化验证：sel4是首个通过数学形式化方法验证正确性的操作系统内核。这一验证保证了理论上内核没有bug，确保其具备严格的可靠性。其形式化验证是安全性的重要保障

以上这些特点，使其作为嵌入式系统内核被广泛应用于军事、航空航天、铁路信号和医疗设备等拥有高安全需求的领域。

欧洲铁路交通管理系统（ERTMS）基于seL4构建信号控制单元，通过SIL-4认证（每小时危险失效概率<1×10⁻⁹）。

美敦力心脏起搏器使用seL4管理无线固件更新模块，采用双向认证协议防止恶意代码注入。

NASA Artemis登月任务的星载计算机运行seL4，利用ECC内存与三模冗余（TMR）设计抵御宇宙射线引发的单粒子翻转（SEU）故障。

这些都是sel4的优越性证明。

而rust语言因内存安全成为近年来操作系统编程的可靠工具，传统嵌入式内核依赖开发者手动管理内存，容易导致缓冲区溢出 、悬垂指针和数据竞争等问题，Rust 的所有权机制和借用检查器在编译期即可消除这些问题，而其无需依赖垃圾回收等机制和方便的编译器环境适合作为系统编程语言。如上面所述，嵌入式系统因为其运行特性需要较高的可靠性和性能要求，rust语言正适合开发对可靠性和性能要求苛刻的系统组件。

3..Smoltcp背景介绍

smoltcp是一个轻量级、Rust实现的嵌入式TCP/IP协议栈，专为资源受限的嵌入式系统设计。其通过利用Rust语言的内存安全特性和零成本抽象，提供了高性能、高安全性的网络协议栈方案，适用于低功耗设备、传感器节点等场景。

smoltcp在低内存开销和高性能的基础上，提供模块化、可扩展的网络协议支持。其完整实现了TCP、UDP、ICMP、ARP等协议,并且现在依然在不断更新开发中。是一个理想的学习和可供调整使用的嵌入式协议栈。

二．研究内容

smoltcp协议栈是我主要进行修改设计的对象。作为一个网络协议栈，其并不是以常见的osi七层模型进行的模块区分，模块也不是严格的高层低层封装关系，而是分为了socket，iface，storage，wire和phy五层，分别负责不同的功能

Socket层是协议栈与应用层的交互模块，负责多种协议的抽象以及连接管理。该层实现了TCP、UDP、ICMP、DNS等协议的Socket抽象，但其设计理念与传统协议栈有所区别，其并不直接参与数据包的封装与传输过程。以TCP协议为例，Socket层为其实现了三次握手、流量控制、超时重传等核心机制，而具体的数据包构建工作是由wire层去执行。这种设计使得协议的处理逻辑与数据传输封装路径实现解耦，为协议扩展提供了灵活性。

如上面所述，wire层是协议栈的协议编解码引擎，其设计遵循"可构建即可解析"的核心原则。该层包含了以太网帧、IP数据包、TCP/UDP报文等各层协议的构建器与解析器。例如在构建IP数据包时，wire层会自动计算校验和并填充必要字段，确保生成的报文完全符合协议规范，而其中也可以通过函数控制是否进行校验和的添加，来进行性能上的管控；在解析过程中，则会进行严格的格式验证，丢弃不符合规范的数据包。这种双向一致性保障机制，使得协议栈具备自我验证能力，从根本上避免了报文本身错误导致的解析错误。而wire层通过零拷贝技术直接操作原始字节流，也提升了处理效率。

iface层负责协调Socket层与wire层的交互。该层通过同步轮询机制驱动整个协议栈的运转：在发送方向，iface层定期收集各Socket的发送缓冲区数据，调用wire层进行协议封装，最终将完整的数据帧提交至物理设备；在接收方向，则从物理设备获取原始数据帧，经wire层解析后分发至对应Socket的接收缓冲区。这种显式的poll驱动模型，与常见操作系统内核的中断驱动架构形成鲜明对比，虽在实时性方面存在理论劣势，却显著降低了上下文切换开销，更适应资源受限的嵌入式场景。

storage层通过静态预分配的缓冲区管理策略，实现了确定性的内存使用。该层提供环形缓冲区、分片缓冲区等多种数据结构，支持零拷贝的数据传递。例如在TCP传输过程中，应用层数据直接写入Socket的发送环形缓冲区，iface层在poll周期内批量获取这些数据进行协议封装，整个过程避免了数据复制带来的性能损耗。这种设计对内存碎片化问题具有天然免疫力，特别适合长期运行的嵌入式系统。

phy层作为硬件抽象层，定义了统一的网络设备接口。该层支持环回接口、TUN/TAP虚拟设备、原始套接字等多种物理/虚拟设备的接入，通过trait抽象屏蔽具体设备的差异性。开发者可通过实现Device特质，将协议栈移植到定制硬件平台。phy层的异步设计允许在无操作系统支持的环境下，通过忙等待或硬件中断机制驱动数据传输，这种灵活性是smoltcp能适配多种嵌入式场景的关键。

Smoltcp中是没有给出高层的封装的，也就是并没有一个函数真正封装了发送和接收的函数，其需要使用者去进行编写，也就提供了进一步进行逻辑修改的可能性，适合我对其进行修改。

从smoltcp各层功能以及相应函数，可以分析得到他的运行逻辑是：一个网络设备维护多个socket连接，这部分是由socket层定义的，而发送是多个函数的调用结果，首先是调用socket的send函数，把数据存储到socket本身的缓冲区中，之后每个设备有一个interface，其会以同步的方式调用poll函数，而poll函数可以分为两个功能，一个是发送，也就是依次取出所有socket中的缓冲区的内容，首先进行数据报的封装，这部分由通过wire层的函数进行，之后存储到设备本身的发送缓冲区中，由设备驱动进行发送；而另一个功能是接收，在完成发送后，会检查设备本身的接受缓冲区，有数据的情况下将其取出并进行数据报解析，将解析结果放入相应的socket的接受缓冲区中，之后实际处理使用通过socket调用recv函数进行。

其次是对embassy-net和starry-os中网络模块axnet的分析，二者都是在smoltcp的基础上进行了封装，为其提供更好的性能，从上面的分析可以看到socket的send函数，recv函数，还有interface的poll函数都是通过同步的方式，也就是通过轮询控制执行，axnet在实现中使用了WouldBlock机制来协调非阻塞I/O，这里它的设计目标并非直接实现异步I/O，而是通过与操作系统的调度协作，在用户态模拟类似异步的效果，从而进行性能提升，减少无效判断，任务在完成的情况下则返回值，未完成则返回WouldBlock。然后分为两种模式，在non-blocking模式下，子任务未完成，父任务立即返回WouldBlock；而在blocking模式下，子任务未完成，父任务轮询到其完成为止。轮询中交替检查网络包状态和任务状态。这样的实现相比与异步会因为重试多次上下文切换，还是会有性能消耗。

而embassy-net中进行了进一步的提升，使用了异步的方式进行函数调用，基于异步运行时，将smoltcp的同步轮询接口改造为异步任务驱动模型。具体就是将send和recv定义为async函数，借用smoltcp中本身就有实现的waker机制，在准备好数据的情况下调用waker唤醒async函数，来实现异步执行，减少了空闲时的CPU占用，允许其他任务在等待网络事件时执行，提升系统整体效率。两种优化方式为我的后续性能优化改进提供了借鉴思路。

最后进行了当前版本的性能测试，axnet是在一个完整的操作系统starry-os中的模块，也就可以方便的进行发包收包的带宽测试，首先benchmark中构建已经封装好的数据报，直接发送可以测试得到接口层的传输速率，作为上限速率，然后通过starry-os中已有的iperf服务器和iperf指令来进行qemu中的实际socket发包测试，可以得到测试结果：在x86\_64的本地测试中，接口层的发送速率约为1000Mbps，传输层约为300Mbps。

**第一章：背景介绍** 4k

1.1 研究背景及意义

1.1.1嵌入式系统介绍

1.1.2 rel4介绍与前景

1.1.3 smoltcp介绍以及为何使用smoltcp

1.2 研究内容和研究方法

**第二章：相关理论介绍（Rust异步机制）**

**第三章：异步修改的方案设计**

3.1 Smoltcp的架构分析

3.2 smoltcp收发处理分析

3.3 Embassy-net和axnet的介绍

3.4修改方案

**第四章：实现细节**

**第五章：实验结果与分析**

5.1性能测试结果对比

5.2结果分析