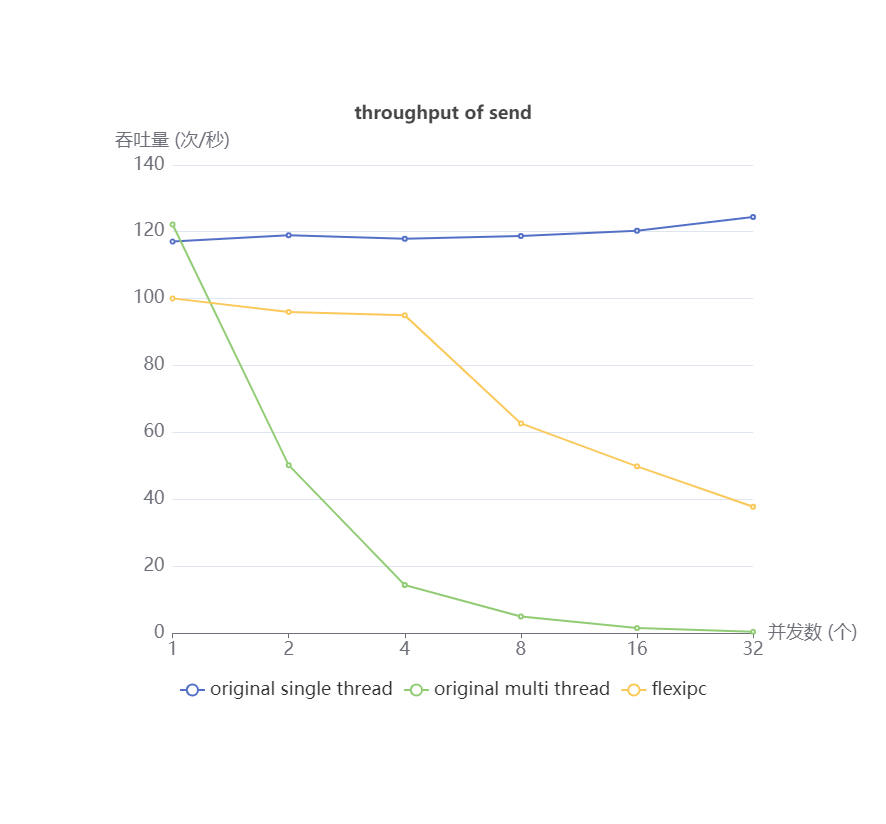
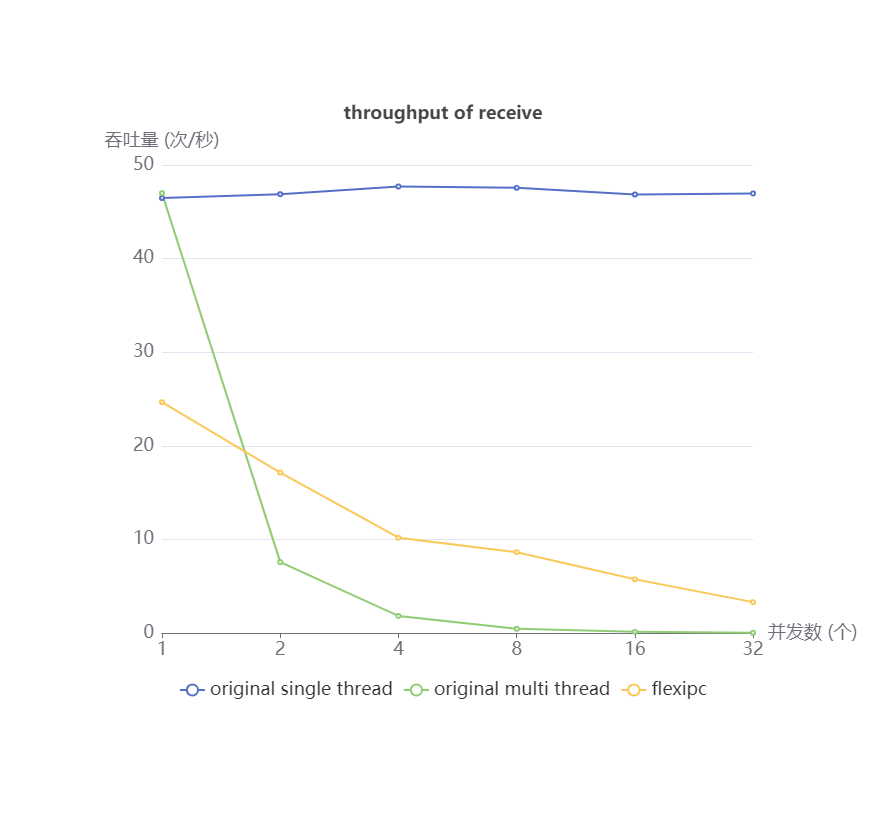


**图 1：test\_net\_client.c的吞吐量**



**图 2：send操作的吞吐量**



**图 3：receive操作的吞吐量**

上面三张图依次测量了test\_net\_client.c、send操作和receive操作的吞吐量。其中，横轴代表并发数，例如并发数为8代表同时有8个test\_client发起请求；每个client执行了20次请求循环；纵轴代表吞吐量，代表lwip在1秒钟内可以处理多少次对应的请求；每张图都测试了三种模式下的吞吐量，其中**original single thread**不使用flexipc，在test\_client端使用单线程进行请求，作为吞吐量的**参考基准**；**original multi thread**不使用flexipc，在test\_client端使用多线程进行请求；**flexipc**使用flexipc，并在test\_client端使用多协程进行请求。从上面的结果图中，可以得出以下结论：

1. flexipc + 多协程调度策略的使用，对test\_net\_client.c的吞吐量有一定的提升。从图1中可以看到，随着并发数的增加，多线程非flexipc模型下，吞吐量急剧下降；与之相反，多协程flexipc模型下，吞吐量逐渐上升。前者的原因，推测是当并发数增加，系统在线程将的调度以及用户态与内核态间的切换开销逐渐占据上风，并严重影响了lwip的处理性能；而在flexipc中，IPC请求的执行避免client线程到server线程的迁移以及随着这些迁移而带来的用户态与内核态切换的开销，同时，协程之间的调度完全是在用户态进行的，协程与协程之间的切换仅仅需要保存一些寄存器的值即可，相对于线程与线程间的切换开销大大降低，因此在多并发情况下表现更好。
2. 调度策略的引入，会影响单个操作的吞吐量。由图2和图3可知，无论是多线程调度还是多协程调度，随着并发数的增加，send操作和receive操作的吞吐量都在逐渐下降。这是因为，当调度器发现某个操作尚未完成时，会将上下文切换到另一个线程/协程执行，从而增加了单个操作的**turnaround time**（即从该操作开始被系统处理到系统处理完成的这段时间），而总的操作数目不变，因此针对该操作的吞吐量便会下降。由于线程间的切换开销远大于协程间的切换开销，多线程调度下，单个操作的吞吐量下降幅度远大于多协程模型。
3. Send操作的吞吐量大于receive操作。对比图2和图3可知，在相同的并发数下，无论是否使用flexipc，send操作的吞吐量始终大于receive操作。我的理解是，send操作属于非阻塞操作，即无需外部依赖，操作只管执行完毕即可；而receive操作为阻塞操作，其需要依赖某个操作完成后（如send操作）才能继续执行，因此receive操作的**turnaround time**总是会大于send操作，对应的吞吐量也就小于send操作。同时，从图2和图3的对比可以看到，flexipc对send操作的吞吐量的提升大于对receive操作吞吐量的提升，应该也是非阻塞操作的非依赖性和阻塞操作的依赖性的结果。