In this thesis, we explored whether it is possible to take advantage of current hardware trends and modify the reliable multiserver operating system design in such a way that it can become an option for building commodity operating systems, which can have both performance and reliability and to achieve it without making compromises on either side. Such a system can improve the experience of average computer users as well as reduce the cost of overprovisioning in data centers or reduce the runtime of long running complex scientific computations. We demonstrated that it is possible at least for the network stack!

We achieve our goal by exploiting the fact that current multicore processors can execute different parts of a multiserver operating system simultaneously without the need of switching between them, thus removing the biggest performance overhead of multiserver systems without changing their general structure. However, this approach is not a straightforward solution that works out of the box and our work discusses the logical and practical details of a reliable, dependable, well performing and resource-aware operating system, which we call NEWTOS.

First, we demonstrated that the communication between servers and drivers of the system is prohibitively expensive and blindly using multicore processors to run the servers on dedicated cores does not make the system perform better. We introduced pure user-space communication between pairs of cooperating processes. The microkernel is excluded from passing messages and only serves as a provider of reliable communication which allows the processes to setup the communication channels in a trustworthy way. In addition to avoiding the microkernel overhead of message passing, the new user-space communication channels are asynchronous, so the communicating processes do not need to wait for a rendezvous and can proceed with processing independent requests without blocking.
To demonstrate the benefits of the new communication mechanism and the distribution of system processes across the available cores, we reimplemented the network stack, a critical part of modern operating systems for performance and reliability. Even though we could have used another subsystem, we opted for the network stack because currently well established multigigabit links can generate load that is beyond what has ever been thought that a multiserver system could handle. In contrast, a subsystem like the storage stack is limited by the speed of the storage devices and only emerging flash-based technologies can sustain similar data transfer rates. In addition, due to the growing popularity of network attached storage (NAS), the network stack is becoming a vital part of the storage stack itself. The novel network stack of NEWTOS has dramatically raised the bar and it shows that a multiserver system, much like other commodity operating systems, is able to process data at multigigabit rates.

Moreover, while we have increased performance, we simultaneously increased reliability of the entire operating system. Using the same principle of breaking up a monolithic system into a collection of isolated servers, we decomposed the monolithic network stack into several components along the boundaries of the individual protocol layers. This simplifies crash recovery since some components have little or no state making them easy to restore. At the same time, we can use specific methods with varying performance overheads to protect components with large and frequently changing state. The key benefit of this decomposition is that a crash in an easily restorable part of the network stack does not directly affect the more complicated components.

Running some system processes on dedicated cores naturally limits the number of cores available to other system processes and, above all, to the applications. This contrasts with the common understanding of how operating systems work based on the prevalent experience of most users with monolithic systems. Although our assumption is that the number of cores of future processors will be less limiting, we investigate the benefits of running a multiserver system on cores that feature more hardware threads as compared to cores, adding threads is easier, cheaper and the system processes often need only a hardware container for their context. At the same time, we also emulate heterogeneous architectures to demonstrate that smaller and simpler cores, are sufficient for running system processes even when the system is relatively highly loaded.

The limitation of running parts of the network stack on dedicated cores is the fact that a single component cannot handle more work than what is possible to handle on a single core. Even though our network stack employs multiple cores, and thus can handle more than a single component network stack could have, the scalability is limited by the most demanding component. At the same time, we want to avoid multithreading. To scale the network stack, we devised a method of running multiple copies of the network stack. Each copy or replica works on its own as the copies are all strictly isolated and do not communicate with each other.

Running multiple independent replicas of the network stack allows the multi-
server system to scale and naturally increase its reliability as a failure in one of the replicas does not render the entire network stack unusable.

Last but not least, we demonstrated that the user-space messaging we use within the network stack to communicate between its parts is further extendable as we use it to improve communication between the operating system and the applications. Applications can use this extended mechanism to request operating system services without expensive system calls that disrupt their execution. We use the model of the user-space communication channels to build network sockets. We have presented that legacy event-driven servers can take advantage of these sockets without any code modifications and achieve better performance running in NEWTOS than in Linux.

We successfully demonstrated that it is possible to build systems that perform “on par” with commodity operating systems, while offering a huge competitive advantage in their reliability and dependability. This combination makes them unique and should make them widely acceptable in many areas of computing.