MultiEdge: An Edge-based Communication Subsystem for Scalable Commodity Servers

Sven Karlsson, Stavros Passas[†], George Kotsis[†], and Angelos Bilas[†]

Institute of Computer Science (ICS)
Foundation for Research and Technology - Hellas (FORTH)
P.O. BOX 1385, Heraklion, GR-71110, Greece
{svenka,stabat,kotsis,bilas}@ics.forth.gr

Abstract

At the core of contemporary high performance computer systems is the communication infrastructure. For this reason, there has been a lot of work on providing low-latency, high-bandwidth communication subsystems for clusters. In this paper, we introduce MultiEdge*, a connection oriented communication system designed for high-speed commodity hardware.* MultiEdge *provides support for end-to-end flow-control, ordering, and reliable transmission. It transparently supports multiple physical links within a single connection.*

We use MultiEdge *to examine the behavior of edge-based protocols using both micro-benchmarks and real-life shared memory applications. Our results show that* MultiEdge *is able to deliver about 88% of the nominal link throughput with a single 10-GBit/s link and more than 95% with multiple 1-GBit/s links. Our application results show that performing all of the communication protocol at the edge does not seem to cause any degradation in performance.*

1. Introduction

Communication infrastructure for scalable systems has recently gone through a wave of commoditization. Most scalable systems today, such as parallel systems for scientific and commercial applications rely on interconnects that plug in the I/O bus and are designed independently of the system motherboard and CPU [1, 8, 23].

However, such communication subsystems require not only the use of specialized network interface cards (NICs) but switches as well. The result is that scalable systems need to employ multiple interconnects for different purposes. Typically, such systems are already interconnected with high-end Ethernet-based networks. In addition, they require independent interconnects for different application domains. This physical partitioning of systems based on their connectivity, results in excessive system costs and management complexity.

An important problem today is to examine whether it is possible to build communication infrastructure that does not require physical partitioning of nodes based on the interconnect type. To achieve this we need to examine if different communication protocols may be provided on top of the same physical interconnect and still satisfy different application domains.

The main difference of communication subsystems traditionally used in scalable systems has been the degree of support required from the network for the communication protocol. Based on this, we can divide interconnects in two categories: core-based and edge-based.

Most cluster interconnects are core-based, e.g. Myrinet [8], Infiniband [1], and Quadrics [23]. These interconnects rely on the network core, i.e., the switches for providing FIFO ordering, flow-control, and reliable communication. On the other hand, edge-based interconnects, such as Ethernet, incorporate all "intelligence" at the networked edge, i.e., NICs and hosts, and only simple forwarding functions are required from the network core.

An emerging aspect of high-end communication subsystems that may further blur differences is the use of spatial parallelism. Spatial parallelism is a new dimension in the design of high-end interconnects that uses multiple physical paths in a decoupled manner to carry the traffic of a single, end-to-end communication channel. Multiple links are already used today in high-end communication systems for byte-level parallelism: A single data unit sliced in bytes, is transmitted over multiple physical links that are tightly

1-4244-0910-1/07/$20.00

[†]Also with the Department of Computer Science, University of Crete, P.O. Box 2208, Heraklion, GR 71409, Greece.

controlled by the sender and the receiver. However, as the number of links increases, it becomes difficult to control the links tightly and to achieve efficient byte-level parallelism.

Another approach to exploiting spatial parallelism is to transparently send full frames on top of separate links. We believe that, for technology reasons similar to multi-core CPUs, the use of spatial parallelism in this decoupled manner will be a main factor in improving communication throughput. However, such systems may exhibit increased congestion, out-of-order delivery, and impose increased processing demands at the edge of the communication subsystem.

Given the lower cost and proliferation of edge-based networks, such as 1- and 10-Gigabit Ethernet, it becomes important to examine protocol layers that can support traditional end-to-end communication semantics and spatial parallelism for serving different application domains.

In this work, we study the network traffic in high throughput, commodity edge-based networks. We also study the impact of spatial parallelism on the network traffic and system performance.

To this end we first present the design and implementation of an edge-based communication subsystem, *MultiEdge*, which uses Ethernet and provides RDMA-type operations, FIFO ordering, reliable transmission, and supports spatial parallelism. We also present a novel communication API that allows users to send data out-of-order in a single communication channel.

Then, we use *MultiEdge* to examine in detail the impact of edge-based protocols on network traffic and system performance. We present a detailed analysis of the extra traffic induced due to reordering, congestion, and spatial parallelism, both with micro-benchmarks and real applications on a a cluster of 16 dual-processor nodes. We present results over a single 1-GBit/s link, two 1-GBit/s links, and a single 10-GBit/s link.

Our micro-benchmark results show that *MultiEdge* is able to deliver about 88% of the nominal link throughput with 10-GBit/s links and more than 95% with 1-GBit/s links and that minimum latency is about 30μs. Our application results show that performing all communication protocol at the edge does not seem to impose any particular overheads for the size of the systems we examine. Also, even for applications such as FFT and Radix that induce a lot of bursty communication, there is only a small percentage of control traffic and retransmitted data. Finally, *MultiEdge* does expose the full throughput of multiple 1-GBit/s links to applications, however, the applications we examine are not able to benefit from this.

Overall, our work shows that edge-based protocols have the potential for significantly reducing the cost of scalable systems in the range of a few hundred nodes. The contributions in this paper: (i) The design of *MultiEdge* and its features for achieving high-throughput at low CPU utilization. (ii) The introduction of decoupled spatial parallelism in the communication subsystem for a single end-to-end communication channel. (iii) The detailed analysis of edge-based protocols, using *MultiEdge* on top of 1- and 10-GBit/s links using both micro-benchmarks and real applications.

The rest of this paper is organized as follows. Section 2 presents the design and implementation of *MultiEdge*. Sections 3 and 4 present our experimental setup and a detailed performance analysis of *MultiEdge* using both micro-benchmarks and real applications. We discuss related work in Section 5. Finally, we draw our conclusions in Section 6.

2. Design

In this section, we introduce the design of *MultiEdge*. We start with an overview of the design and the programming API and then continue with detailed description of the end-to-end support for flow control, ordering, spatial parallelism, and optimizations.

2.1. Overview

MultiEdge is organized in three layers: the *hardware drivers*, the kernel level *protocol layer*, and a user-level library, see Figure 1. The Ethernet hardware drivers access the Ethernet hardware directly and provide a hardware independent interface to the protocol layer. The drivers perform simple low-level access to hardware such as hardware initialization, Ethernet frame reception and transmission, and low-level interrupt processing. The protocol layer is hardware independent, implements the programming API and adds higher level functionality such as end-to-end flow control, reliable data transfer, and high-level interrupt processing.

In the current implementation of *MultiEdge*, the protocol layer is implemented as a Linux kernel character device driver and has support for Broadcom Tigon 3, Intel PRO/1000 and Myricom 10 Gigabit Ethernet hardware.

2.2. Communication Primitives

MultiEdge provides a set of point-to-point, connection-oriented communication primitives; Before any communication can occur between two nodes, a connection has to be set up between the nodes. The programming API of *MultiEdge* has a set of primitives for this purpose.

Once a connection is set up, communication is based on fully asynchronous remote memory operations. Currently, two operations are defined: remote read and remote write. Each operation can access all the virtual address space of a process executing on a remote node. Operations are initiated through a single communication primitive:

```
int RDMA_operation(connection,
 remote_virtual_address,
 local_virtual_address,
 transfer_size, operation, flags);
```

The parameters are, in order: the connection on which the operation is initiated; the virtual addresses for the operation on the remote and local nodes respectively; the size of the data in bytes; the type of operation; and a bit-field of various options that modify the behavior of operations. For example, the API provides a mechanism to deliver a notification to the remote node when selected remote memory write operations have finished. This mechanism is controlled through one of the option bits in the `flags` bit-field.

Each operation can also, when initiated, return a handle. The programmer can query the progress of each issued operation using the operation handle and a set of API primitives.

Finally, one important aspect of *MultiEdge*'s API is that although the API includes primitives for registering memory regions, receive buffers need not be pre-registered. Data is instead copied directly into the virtual address space of the receiver.

2.3. Data Transfer Path

The data transfer path for a remote memory write is outlined in Figure 1. The flow on the initiator side is as follows: i) The application invokes the `RDMA_operation` in the user level library; ii) The user-level library issues a system call to pass control to the kernel level protocol layer; iii) The protocol layer copies the data in the memory region to be written to remote memory from user level to a kernel-level, DMA-capable buffer and constructs an Ethernet header; iv) A transmit primitive is invoked on an Ethernet hardware driver and DMA is used to transport data out of memory and into the NIC.

When a frame is received, the Ethernet hardware uses DMA to copy the frame data to kernel level DMA capable buffers, marked as (1) in Figure 1. After the DMA transfer is complete, an interrupt is issued and an interrupt handler is invoked by the kernel (marker 2). The interrupt handler performs a few low-overhead tasks, such as updating hardware registers and then signals the protocol layer. All high-overhead operations are performed by a kernel level thread, which is waken up by the protocol layer (marker 3). The thread copies data from the kernel-level DMA buffers to user level (marker 4) before returning to sleep (marker 5).

2.4. Flow Control

MultiEdge uses *end-to-end* flow control to ensure reliable communication. All operations and transfers are guaranteed to complete in the presence of dropped Ethernet frames due to *transient* problems, e.g. contention, bit errors, or transient link failures. We use a sliding window flow control algorithm with a fixed size window. The flow control algorithm operates on an Ethernet frame basis. The size of the window is set at compile time.

The receive path uses positive acknowledgments to notify the sender of received frames and negative acknowledgments to report back lost or damaged frames that need to be retransmitted. *MultiEdge* uses piggy-backing to reduce the number of explicit acknowledgments. All data frames carry *positive* acknowledgement information. To further reduce the number of explicit acknowledgements, *MultiEdge* uses *delayed* acknowledgements: it will defer transmission of explicit positive or negative acknowledgements until after a number of frames have been received or dropped or a time-out occurs.

Finally, to ensure data is delivered even in corner cases, such as link failures and lost acknowledgments, the sender will retransmit the last transmitted Ethernet frame if it has not received a positive acknowledgment for that frame within a coarse-grain timeout period.

2.5. Spatial Parallelism

MultiEdge can make use of multiple network interfaces within a single communication channel. Whenever a frame needs to be transmitted, *MultiEdge* will use one of the available network interfaces based on a load-balancing policy. We currently use a round-robin policy. Thus, frames may be delivered out of order.

Frames belonging to remote memory operations that need not be ordered with respect to other operations, may be processed at the receive path as soon as they arrive without need for buffering. However, we believe that, to be meaningful, out-of-order delivery can not be done indiscriminately. To support both in-order and out-of-order delivery we extend the communication API to support the following operation flags:

- Backward fence: This remote memory operation will be performed on the destination node only after all previous operations issued by this source node to the same destination have been performed.
- Forward fence: Any subsequent operation issued by this source node to the same destination will be performed only after this operation has been performed.

These flags are orthogonal and so a single operation can specify both flags. The default behavior is to allow operations, and all their data frames, to be freely re-ordered. To specify ordering constraints, the programmer uses the `flags` bit-field of the `RDMA_operation` API call. Individual frames originating from the same, or even different,

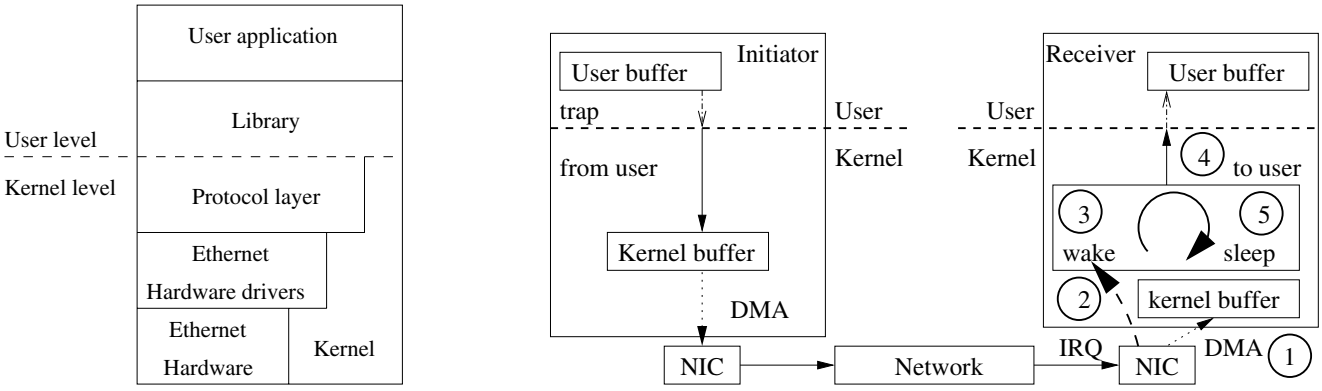


Figure 1. Overview of MultiEdge layers (left) and data transfer path (right).

operations can be re-ordered with respect to each other if only the semantics of the operation flags are upheld.

2.6. Reducing Overheads

A major concern at high network speeds is how to reduce software overheads so that the data rates can be used efficiently. In particular, interrupts induce a significant overhead in modern operating systems. The design of *MultiEdge* tries to minimize interrupts both in the send as well as the receive path in the following way: When an interrupt arrives, the interrupt handler disables subsequent interrupts and notifies the protocol layer. When the receive or send protocol path is invoked by an interrupt handler, it processes all pending interrupt related events, e.g. send frame completions or newly received frames, by polling each network interface. The protocol layer enables interrupts when there are no more interrupt related events and no protocol kernel thread is active.

3. Experimental Setup

We evaluate *MultiEdge* using a cluster of 16 nodes equipped with dual Opteron 244 1.8 GHz processors. The nodes are connected with D-Link DGS-1024T switches. The switch firmware used does not support jumbo Ethernet frames. Each node uses a Tyan S2892 motherboard with 1 GByte of main memory and two Broadcom Tigon 3, 1-GBit/s Ethernet interfaces. We use Debian Etch as operating system with a Linux version 2.6.12.6 kernel compiled with GCC 4.0.4.

We also use a smaller setup with four nodes equipped with one Myricom 10G-PCIE-8A-C card each. These four nodes are otherwise identical to the nodes in the aforementioned cluster. The four 10-GBit/s Ethernet cards are interconnected with a HP ProCurve 6400cl J8433A 10 Gigabit Ethernet switch.

We use the following system setups in our experiments:

- 1L-1G: Single 1-GBit/s link, where 16 nodes are connected via a single 1-GBit/s Ethernet switch and NICs.
- 1L-10G: Single 10-GBit/s link, where 4 nodes are connected via a single 10-GBit/s Ethernet switch and NIC.
- 2L-1G: Two 1-GBit/s links, where 16 nodes are connected via two 1-GBit/s Ethernet switches and NICs. In this setup, all frames are always delivered in-order.
- 2Lu-1G: Similar to 2L-1G, only we allow *MultiEdge* to deliver frames out of order, when there are no ordering restrictions.

We evaluate *MultiEdge* using both three micro-benchmarks as well as a realistic cluster programming environment. The three micro-benchmarks all use two nodes. They are as follows:

ping-pong transfers data using remote memory writes between two nodes in a request-reply fashion. Request and replies carry the same amount of data.

one-way performs sends of data in one direction only. The sending node performs remote memory writes back-to-back.

two-way in which both nodes performs simultaneous one way transfers and exercises the send and receive paths at the same time. To accurately reflect the amount of data transfered in this benchmark, the throughput presented for this benchmark is the sum of the throughputs of both nodes' transfers.

We also use *GeNIMA* [5], a software DSM system that provides a shared address space abstraction on top of our cluster to run real-life applications. *GeNIMA* is a page-based system that has been optimized for system area networks that support remote DMA operations. Software DSM

Application	Problem Size	Seq. Exec. Time (ms)	Footprint (MBytes)
Barnes-Spatial	128K/64K particles	2877713	120/45
FFT	2^{22} complex values	4752	200
LU	8Kx8K matrix	412096	500
Radix	32M integers	4179	120
Raytrace	Balls scene 1Kx1K	376096	210
Water-Nsquared	128K molecules	11678974	90
Water-Spatial	128K molecules	231889	80
Water spatialFL	128K mols	229586	80

Table 1. Benchmark applications.

systems impose requirements that are different and usually more demanding than those of message passing systems [17] which may be closer to the behavior of micro-benchmarks. On top of *GeNIMA* we run a set of applications from the SPLASH-2 benchmark suite [29]. The applications and problem sizes we use are listed in Table 1. The table also holds the working data set used in the experiments for each application and the execution time of a purely sequential version of each application. All applications as well as *GeNIMA* have been compiled with GCC 4.0.4 at optimization level -O2.

Previous versions of *GeNIMA* have only supported 32-bit architectures. We port *GeNIMA* to *MultiEdge* and we also extend its internal data structures to take advantage of the 64-bit address space in our experimental platform. Similarly, we port the applications to a 64-bit address space. Our modifications do not alter application behavior in any way.

For each application and system configuration we present statistics at two levels. First, we use speedup curves and execution time breakdowns to examine overheads incurred by *MultiEdge*. Then, we present network-level statistics to examine how *MultiEdge* and edge-based protocols behave with real-life workloads.

Although our nodes are equipped with two CPUs each, in our experiments we use only one CPU for the application. The second CPU is used for executing the communication protocol. We choose to perform our analysis with using a single CPU for the application because we believe that future multicore CPUs will have enough cores available to dedicate to protocol processing and thus, this setup is more representative of future trends. However, we have also performed experiments with using both CPUs for application threads, but we do not include them here for space reasons and because they do not offer any further insight with respect to the results we are presenting.

4. Performance Evaluation

Figures 2(a,b) show latency and throughput for each micro-benchmark. Latency in ping-pong reflects one-way memory to memory time for each operation. We see that minimum latency is about 30μs for 1L-10G. For one-way and two-way this measurement reflects host overhead to initiate an operation. In both cases, minimum host overhead is about 2μs and it is not affected by bi-directional traffic.

In terms of throughput, *MultiEdge* can fully utilize link throughput in the 1L-1G and 2L-1G configurations delivering a maximum throughput of about 120 MBytes/s and 240 MBytes/s with one and two links respectively in all tests.

In 1L-10G the maximum throughput is about 1100 MBytes/s of the peek 1250 MBytes/s, or about 88%. This is achieved with one-way. The reason for not reaching the maximum of 1250 MBytes/s appears to be a higher-than-expected overhead on the sender side. One part of the overhead comes from the fact that the NIC does not allow us to disable the interrupts on the send path that are used for freeing send buffers, even when our polling technique is effective. We have verified that the flow-control scheme we use does not limit the maximum throughput.

In ping-pong the maximum throughput is about 710 MBytes/s. This is partly due to the round-trip time of a single operation and partly due to additional interrupt overhead incurred with the specific NICs. In two-way, maximum throughput is about 1500 MBytes/s out of the 2500 MBytes/s combined peak throughput of the send and receive paths in each node.

Figure 2(c) shows the CPU utilization for the communication protocol. Maximum CPU utilization is plotted out of 200%, reflecting the existence of two CPUs in each node. The values are indicative as we somewhat underestimate CPU utilization as we cannot accurately measure context switch overheads.

For one and two 1-Gbit/s link configurations we see that in ping-pong CPU utilization is at most about 35% and for operations less than 16 KBytes it is always less than 20%. For one-way, CPU utilization is at most 30%. On the other hand, two-way exhibits up to 140% for small operations. However, even in this case, CPU utilization is at most 43% for transfer sizes greater than 1K.

For 1L-10G, we see that the maximum CPU utilization in ping-pong is about 75%, which corresponds to a maximum throughput of about 710 MBytes/s. In one-way, maximum CPU utilization is about 95%, corresponding to a maximum throughput of about 1100 MBytes/s. Finally, two-way has a maximum CPU utilization of about 170% as both the send and receive path are operating concurrently. This corresponds to about 750 MBytes/s in each path for a total of about 1500 MBytes/s.

We have also studied in more detail what happens at

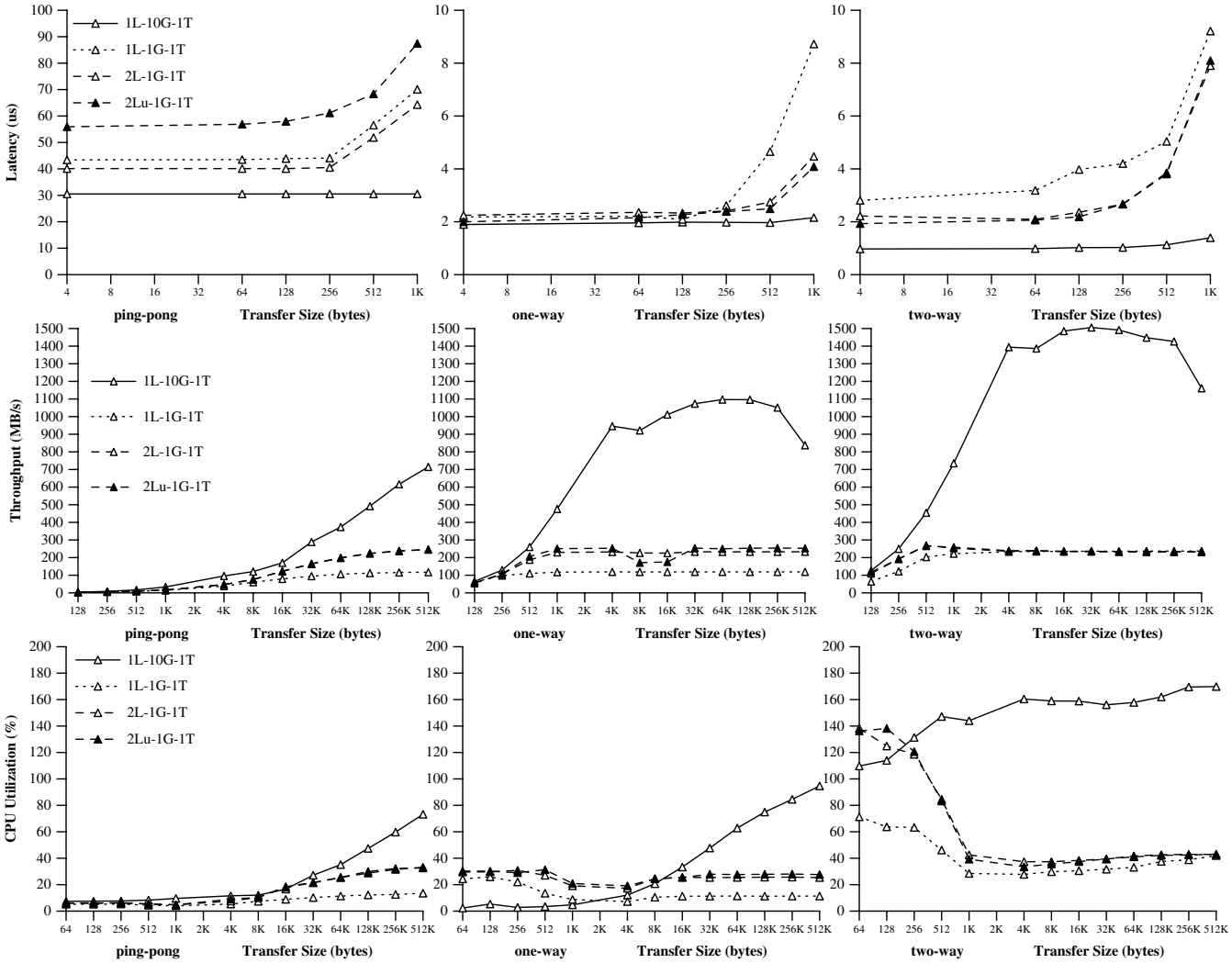


Figure 2. MultiEdge latency (μs), throughput (*MBytes/s*) and CPU utilization for each micro-benchmark.

the network itself. All variants of the benchmarks running on single link incur almost no out-of-order delivery at all. In multi-link configurations, the fraction is at most 45-50%, due to the round-robin scheduling of frames to links. Our results indicate that frames arrive out-of-order but closely spaced and thus, only a few frames may need to be buffered on average. We measure the number of extra frames in the network generated by *explicit*, positive or negative, acknowledgements and data retransmissions. We see that in all cases the percentage of extra frames is at most 5.5%. Moreover, our experiments show that the number of dropped frames is low and about 20% of the extra traffic.

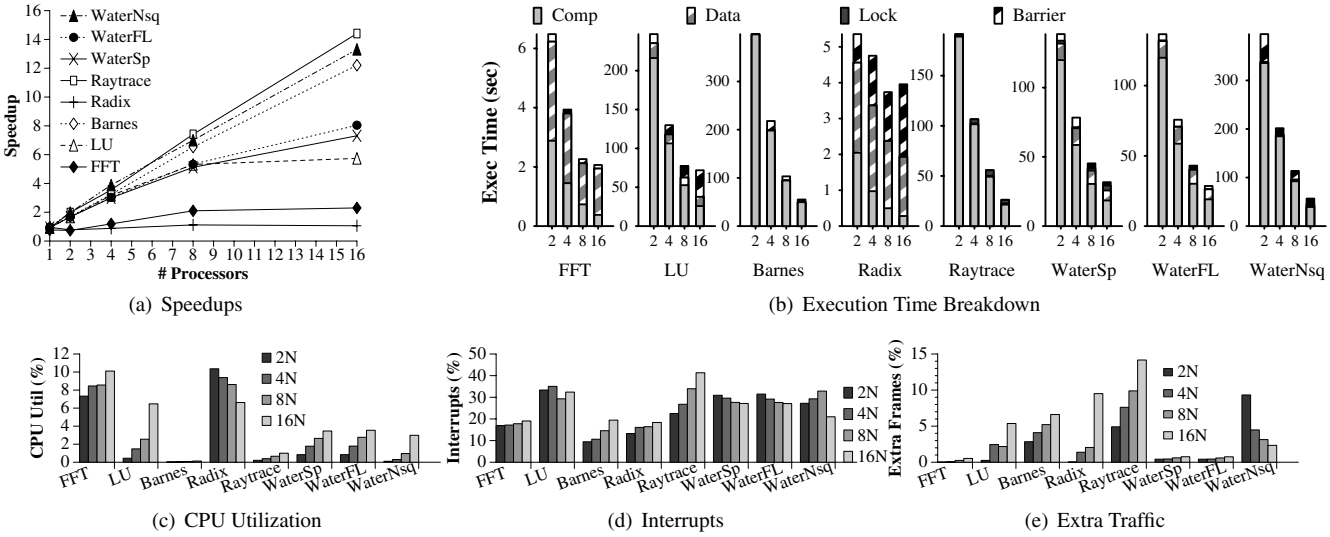
4.1. Application Results

Figure 3 shows our results for the single 1-GBit/s link setup. We can divide applications in three categories:

Barnes, Raytrace, and Water-Nsquared that scale well with speedups in the range of 13-14 and exhibit low communication and synchronization overheads.

Lu, Water-Spatial, and Water-SpatialFL that exhibit medium speedups in the range of 6-8. Although larger problem sizes may allow these applications to improve their speedups, we do not examine this here, as absolute application performance is not our main focus.

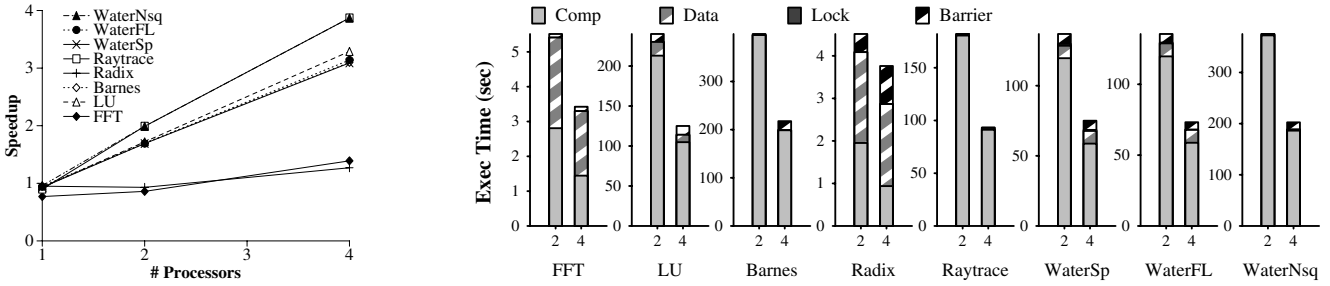
FFT and Radix exhibit poor scalability. They are both known to cause very high processor-memory traffic, which is translated to inter-node traffic. Radix has poor spatial lo-



(a) Speedups (b) Execution Time Breakdown

(c) CPU Utilization (d) Interrupts (e) Extra Traffic

Figure 3. Application statistics over a single 1-GBit/s link (1L-1G).



(a) Speedups (b) Execution Time Breakdown

Figure 4. Application statistics over a single 10-GBit/s link (1L-10G).

cality generating a high amount of traffic and false sharing. In FFT, the dominant part of the parallel overhead is remote memory fetches which account for, on the average, roughly 77% of the overhead.

Looking at the network statistics in Figures 3(c,d,e) we see that the CPU time spent in the *MultiEdge* protocol does not exceed 11% even for the most communication and synchronization intensive applications. For most applications the time spent in *MultiEdge* is up to 4%. Also, between 10-40% of the frames cause interrupts at the send and receive paths. The additional traffic induced by *MultiEdge* is at most 15% of the application traffic. Almost all of the additional traffic is due to explicit acknowledgements rather than retransmissions. Similar to the micro-benchmarks for the 1L setups, the amount of out-of-order data delivery is almost always close to zero (not shown).

In the single, 10-GBit/s link setup (Figure 4) we see that most applications, except FFT and Radix, achieve good speedups in the range of 3 to 4. This is mainly due to the small number of nodes in our setup. Compared to the 1L-1G setup (Figure 3), synchronization and data wait time improve across applications by about a factor of two on most applications. Radix and FFT still spend a significant portion of execution time in communication and barrier synchronization.

Figure 5 shows our results when using two links and all network operations are strictly ordered. We note that application speedups and execution times are similar to 1L-1G (we show only execution time breakdowns). Data wait and synchronization time improve only for Radix, which however, still exhibits, poor scalability.

Figures 5(b,c,d,e) present network-level statistics for this setup. Similar to the 1L-1G setup, we see that CPU time spent in the send and receive paths of *MultiEdge* is at most 12% for the applications that exhibit the highest communication to computation ratio, FFT and Radix, and which

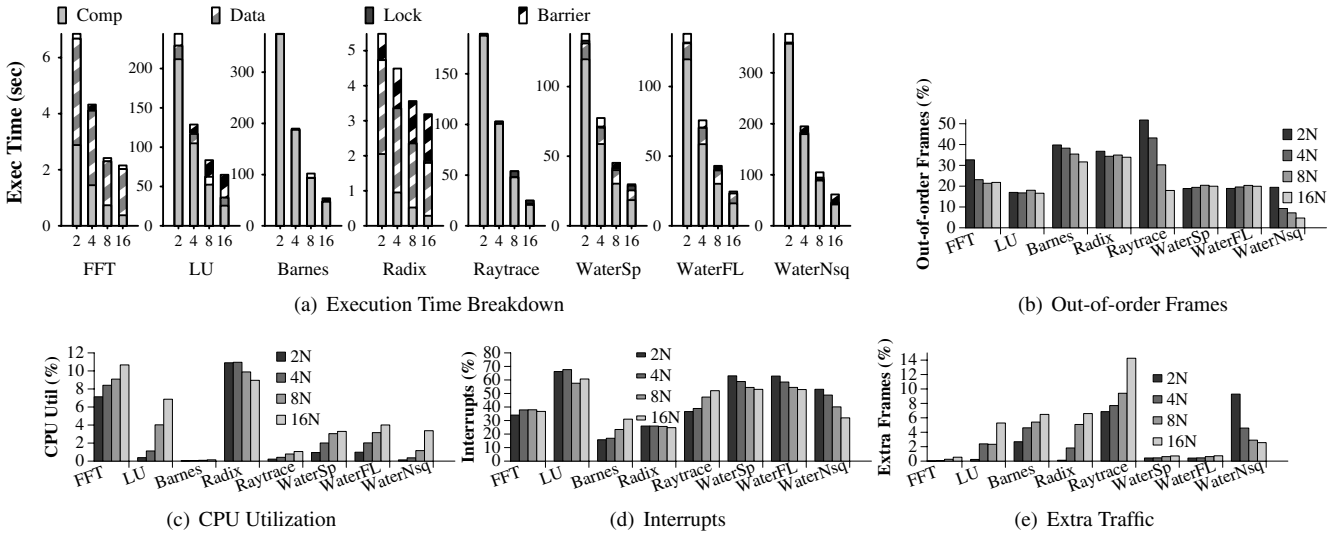


Figure 5. Application statistics over two 1-GBit/s links (2L-1G).

exhibit the poorest scalability. This indicates that communication protocol overhead is not a significant factor for most applications.

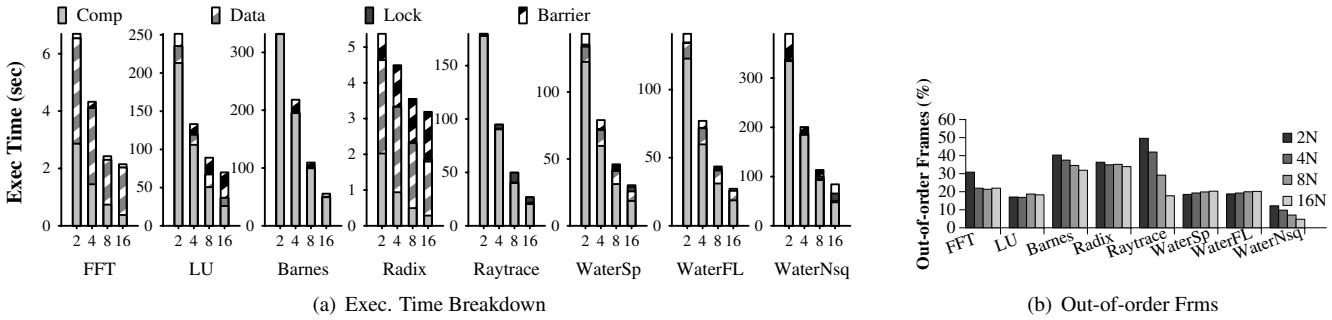
The number of frames received out of order are significantly higher than the 1L setup, which agrees with micro-benchmark results. Between 10-50% of the frames are not received in order, resulting in a re-ordering every 2-10 frames. However, the additional traffic induced by *Multi-Edge* in the form of explicit acknowledgements and data retransmissions is at most 10% of the transmitted frames in Raytrace and Water-Nsquared and at most 4% in all other applications, indicating that the additional traffic due to congestion and transient errors is very low. Finally, between 10-35% of frames in the send and receive path generate interrupts, resulting effectively in a total coalescing factor of about 3-10.

Figure 6 shows our results for two 1-GBit/s links when data is allowed to be delivered out-of-order. For these experiments, we modify the *GeNIMA* protocol using the communication API extensions and we enforce ordering only between necessary operations, rather than all operations between each pair of nodes. We see that this does not have a significant impact on application performance. However, we are interested more in the possible impact of reordering at the network protocol level. Our measurements show that the network level statistics are very close to those for ordered operations. Thus, overall, reducing the amount of ordering applications, or *GeNIMA*, impose on operations does not alter communication protocol behavior in any significant manner.

5. Related Work

There has been extensive previous work in improving base communication performance by enabling user-level communication, eliminating copies of data, and reducing host overheads and context switches [4, 11, 14, 22, 24, 27]. Also, there has been work on network interface architectures and support for high-performance cluster communication [1, 6, 7, 8, 15, 16, 23]. Finally, previous work has evaluated low-latency, high-speed interconnects in various contexts [2, 18, 19]. Our work differs from these efforts and builds on previous work in two important ways: (a) We advocate kernel-level, edge-based communication subsystems that provide high level semantics and transparency, important for commercial applications and (b) we introduce spatial parallelism and examine the impact on edge-based protocols using both micro-benchmarks and real applications.

Previous efforts that are related to our work in terms of the underlying platform include [25, 28]. The authors in [28] provide a communication protocol, UNet, on top of Fast Ethernet and ATM interconnects. Their goal is to provide high-bandwidth, low-latency communication on top of commodity interconnects. They focus on data transfers and describe how they can be performed directly from user space when the NIC provides a programmable CPU and what support is required at the kernel-level for less aggressive NICs. The authors in [25] present a user-level, zero-copy protocol design and implementation on top of 1 GBit/s Ethernet. They achieve a minimum latency of 23 μs and a maximum bandwidth of 880 MBits/s, close to our kernel-level protocol over a single 1 GBit/s link. In our work we



(a) Exec. Time Breakdown

(b) Out-of-order Frms

Figure 6. Application statistics over two 1-GBit/s links allowing out-of-order data delivery (2Lu-1G).

examine 1- and 10-GBits/s links, multiple links, and we present detailed network statistics on the impact of edge-based protocols on network traffic.

M-VIA is an implementation of the Virtual Interface Architecture (VIA) [13] over Gigabit Ethernet networks. It only supports single network interfaces. Moreover, previous work with M-VIA has only examined performance issues on 1 GBit/s Ethernet networks. The authors in [3] examine the base send/receive performance of VIA on native and Ethernet implementations. They find that 1 GBit/s Ethernet implementations of VIA have the potential of delivering higher throughput than TCP/IP-based protocols. However, native VIA implementations provide about 30-60% better throughput. The authors in [21] compare various MPI implementations in a cluster interconnected with Gigabit Ethernet. The MPI implementations rely either on TCP/IP or a VIA-type substrate for basic communication capabilities. They find that using TCP/IP imposes significant overheads and that VIA-type base communication on top of Gigabit Ethernet has significant potential for improving MPI performance.

Our multi-link approach bears similarity with *inverse multiplexing* [9, 12] that has been proposed to improve the end-to-end throughput of wide range connections. Although the concept is similar, the tradeoffs and required mechanisms in our setup, i.e., scalable systems, are very different.

Previous work has examined issues in building multirail network configurations. The authors in [10] use simulation to examine rail allocation methods in multi-stage, cluster-based networks. They find that certain allocation methods can result in significant improvements in latency and bandwidth. In contrast, we aim to examine in a real system the overheads and benefits of using multiple rails on edge-based communication subsystems. The authors in [20] examine how multiple rails can be used in Infiniband interconnects under MPI. Our work on one hand uses Ethernet as the interconnect and on the other hand is more transparent in that all higher system layers are able to take advantage of multiple rails.

Finally, the authors in [26] design and build a multi-dimensional hyper crossbar network using multiple Gigabit Ethernet interfaces. They find that their system delivers more than 90% of the peak throughput for different micro-benchmarks. To the best of our knowledge, the system they designed supports spatial parallelism but, and in contrast to our work, does not support remote memory operations.

6. Conclusions

In this paper, we examine the viability of edge-based communication protocols for building scalable servers. We examine both the level of performance they may offer, how they may take advantage of spatial parallelism in the interconnect, and the the impact they may have on network traffic and behavior.

We design and implement *MultiEdge*, a communication subsystem that support remote read and write memory operations over ordinary 1- and 10-Gigabit Ethernet interfaces, using raw Ethernet frames.

We evaluate *MultiEdge* using both micro-benchmarks and real-life shared memory applications running on top of a software distributed shared memory system. Our micro-benchmark results show that *MultiEdge* is able to deliver about 88% of the nominal link throughput with 10-GBit/s links and more than 95% with 1-GBit/s links. The minimum latency is about 30μs. Our application results show that performing all communication protocol at the edge does not seem to impose any particular overheads for the size of the systems we examine. Under bursty applications, such as FFT and Radix that induce a lot of communication, there is only a small percentage of control traffic and retransmitted data. Finally, *MultiEdge* does expose the full throughput of multiple 1-GBit/s links to applications.

Finally, we believe that future work should examine (a) larger system configurations with more nodes and communication paths that consist of multiple switches and (b) hybrid approaches that include support from the core of the

network for certain operations or require extensions of the network interfaces for offloading edge-protocol overheads. *MultiEdge* and the underlying infrastructure can help in exploring these directions in great detail.

7. Acknowledgements

We would like to thank the members of the CARV laboratory at FORTH-ICS for the useful discussions. We thankfully acknowledge the support of the European FP6-IST program through the UNIsIX project and the HiPEAC Network of Excellence.

References

[1] An infiniband technology overview. Infiniband Trade Association, http://www.infinibandta.org/ibta.

[2] S. Araki, A. Bilas, C. Dubnicki, J. Edler, K. Konishi, and J. Philbin. User-space communication: A quantitative study. In *Proc. of SC'98*, Nov. 1998.

[3] M. Baker, P. A. Farrell, H. Ong, and S. L. Scott. Via communication performance on a gigabit ethernet cluster. In *Proc. of Euro-Par 2001*.

[4] A. Basu, V. Buch, W. Vogels, and T. von Eicken. U-net: A user-level network interface for parallel and distributed computing. In *Proc. of SOSP15*, pages 40–53, Dec. 1995.

[5] A. Bilas, C. Liao, and J. P. Singh. Using network interface support to avoid asynchronous protocol processing in shared virtual memory systems. In *Proc. of ISCA26*, May 1999.

[6] M. Blumrich, C. Dubnick, E. Felten, and K. Li. Protected, user-level dma for the shrimp network interface. In *Proc. of HPCA2*, Feb. 1996.

[7] M. Blumrich, K. Li, R. Alpert, C. Dubnicki, E. Felten, and J. Sandberg. A virtual memory mapped network interface for the shrimp multicomputer. In *Proc. of ISCA21*, pages 142–153, Apr. 1994.

[8] N. Boden, D. Cohen, R. Felderman, A. Kulawik, C. Seitz, J. Seizovicm, and W. Su. Myrinet: A gigabit-per-second local-area network. *IEEE Micro*, 15(1):29–36, 1995.

[9] F. M. Chiussi, D. A. Khotimsky, and S. Krishnan. Generalized inverse multiplexing of switched atm connections. In *Proc. of GLOBECOM'98*, Nov. 1998.

[10] S. Coll, E. Frachtenberg, F. Petrini, A. Hoisie, and L. Gurvits. Using Multirail Networks in High-Performance Clusters. *Concurrency and Computation: Practice and Experience*, 15(7-8):625–651, 2003.

[11] C. Dubnicki, A. Bilas, Y. Chen, S. Damianakis, and K. Li. Vmmc-2: Efficient support for reliable, connection-oriented communication. In *Proc. of HOT Interconnects V*, 1997.

[12] J. Duncanson. Inverse multiplexing. In *IEEE Communications Magazine*, pages 34–41, Apr. 1994.

[13] D. Dunning and G. Regnier. The Virtual Interface Architecture. In *Proc. of HOT Interconnects V*, Aug. 1997.

[14] T. Eicken, D. Culler, S. Goldstein, and K. Schauser. Active messages: A mechanism for integrated communication and computation. In *Proc. of ISCA19*, May 1992.

[15] Giganet. Giganet cLAN family of products. http://www.emulex.com/products.html, 2001.

[16] R. Gillett, M. Collins, and D. Pimm. Overview of network memory channel for PCI. In *Proc. of COMPCON'96*, Feb. 1996.

[17] S. Karlsson and M. Brorsson. A comparative characterization of communication patterns in applications using mpi and shared memory on an ibm sp2. In *Proc. of CANPC'98*, Jan. 1998.

[18] J. Liu, B. Chandrasekaran, W. Yu, J. Wu, D. Buntinas, S. Kini, and D. Panda. Microbenchmark performance comparison of high-speed cluster interconnects. *IEEE Micro*, 24(1):42–51, 2004.

[19] J. Liu, B. Chandrasekaran, W. Yu, J. Wu, D. Buntinas, S. P. Kini, D. K. Panda, and P. Wyckoff. Microbenchmark performance comparison of high-speed cluster interconnects. *IEEE Micro*, 24(1):42–51, 2004.

[20] J. Liu, A. Vishnu, and D. K. Panda. Building multirail infiniband clusters: Mpi-level design and performance evaluation. In *Proc. of SC'04*, Nov. 2004.

[21] H. Ong and P. A. Farrell. Performance comparison of lam/mpi, mpich, and mvich on a linux cluster connected by a gigabit ethernet network. In *Proc. of 4th Annual Linux Showcase and Conference*, Oct. 2000.

[22] S. Pakin, V. Karamcheti, and A. Chien. Fast Messages (FM): efficient, portable communication for workstatin clusters and massively-parallel processors. *IEEE Concurrency*, 1997.

[23] F. Petrini, W. Feng, A. Hoisie, S. Coll, and F. E. The quadrics network: High-performance clustering technology. *IEEE Micro*, 22(1):46–57, 2002.

[24] L. Prylli and B. Tourancheau. BIP: a new protocol designed for high performance. In *Proc. of PC-NOW'98*, Mar 1998.

[25] P. Shivam, P. Wyckoff, and D. Panda. EMP: Zero-copy OS-bypass NIC-driven Gigabit Ethernet message passing. In *Proc. of SC'01*, Nov. 2001.

[26] S. Sumimoto, K. Ooe, K. Kumon, T. Boku, M. Sato, and A. Ukawa. A scalable communication layer for multi-dimensional hyper crossbar network using multiple gigabit ethernet. In *Proc. of ICS06*, June 2006.

[27] H. Tezuka, A. Hori, and Y. Ishikawa. PM: a high-performance communication library for multi-user parallel environments. Technical Report TR-96015, Real World Computing Partnership, 1996.

[28] M. Welsh, A. Basu, and T. von Eicken. Atm and fast ethernet network interfaces for user-level communication. *Proc. of HPCA3*, February 1997.

[29] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta. The SPLASH-2 programs: Characterization and methodological considerations. In *Proc. of ISCA22*, June 1995.