Wire-Speed Total Order

Tal Anker[1] [2], Danny Dolev[2], Gregory Greenman[2], Ilya Shnaiderman[2]

[1] Radlan - a Marvell Company
tala@marvell.com

[2]The Hebrew University of Jerusalem
The School of Engineering and Computer Science
Jerusalem, Israel
{anker, dolev, gregory, ilia}@cs.huji.ac.il

Abstract

Many distributed systems may be limited in their performance by the number of transactions they are able to support per unit of time. In order to achieve fault tolerance and to boost a system's performance, active state machine replication is frequently used. It employs total ordering service to keep the state of replicas synchronized. In this paper, we present an architecture that enables a drastic increase in the number of ordered transactions in a cluster, using off-the-shelf network equipment. Performance supporting nearly one million ordered transactions per second has been achieved, which substantiates our claim.

1 Introduction

In distributed computing, developing software has been traditionally considered to be the main goal. Since most of participating components in a distributed system are software modules, it is usually assumed that the number of "transactions" such a system could generate and handle is limited mainly by the CPU resources.

A recent technological trend implies introducing hardware elements into distributed systems. Implementing parts of a distributed system in hardware immediately imposes performance requirements on its software parts. An example of a system that combines hardware and software elements is a high-capacity Storage Area Network, combining a cluster of PC's, Disk Controllers and interconnected switches that can benefit from high-speed total order. The rationale is elaborated on in greater detail in [1, 12].

This paper shows how message *ordering* can be guaranteed in a distributed setting, along with a significant increase in the number of "transactions" produced and processed. The proposed architecture uses off-the-shelf technology with minor software adaptations.

Message ordering is a fundamental building block in distributed systems. "Total Order" is one of the basic message delivery order guarantees, allowing distributed applications to use the state-machine replication model to achieve fault tolerance and data replication. Extensive analysis of algorithms providing total ordering of messages can be found in [10]. One of the most popular approaches to achieve total order implies using a sequencer that assigns order to all messages invoked. This scheme, however, is limited by the capability of the sequencer to order messages, e.g., by CPU power. The goal of the methodology presented in this paper is to achieve a hardware-based sequencer while using standard off-the-shelf network components. The specific architecture proposed uses two commodity Ethernet switches. The switches are edge devices that support legacy-layer-2 features, 802.1q VLANs and inter VLAN routing, which are connected via a Gigabit Ethernet link and a cluster of dual homed PCs (two NICs per PC) that are connected to both switches. One of the switches functions as the *virtual sequencer* for the cluster. Since the commodity switch supports wirespeed on its Gigabit link, we can achieve a near wirespeed traffic of a totally ordered stream of messages.

In this paper, we describe the architecture, its assumptions and the adjustments made to the software of the PCs. Performance results presented show that a near wirespeed traffic of totally ordered messages is now a reality. The proposed architecture can be adjusted to various high-speed networks, among them In-

1-4244-0054-6/06/$20.00

finiBand [3] and Fiber-Channel [2], which do not support multicast with ordering guarantees. In addition, our approach includes a highly efficient optimistic delivery technique which can be utilized in various environments, e.g. replicated databases, as shown in [14].

2 Contribution

In this work, the following contributions have been made:

- We proposed a new cost-effective approach that uses only off-the-shelf hardware products. The approach is not limited to CSMA/CD networks and can be applied to other networks as well.
- The approach has been implemented and evaluated within a real network.
- We managed to remove significant overhead from middleware that implements active state machine replication. It is known that replication usually provides good performance for read requests, but incurs a significant overhead on write requests [5]. We reduced the message latency and increased the throughput of the system that can now perform ordering of more than a million messages per second.

3 Model and Environment

The distributed setting is composed of a set of computing elements (PCs, CPU based controllers, etc.) residing on a LAN connected by switches. The computing elements, referred to as nodes, can be either transaction initiators (senders), or receivers, or both.

The nodes are connected via full-duplex links through commodity switches. We assume that the switches support IGMP snooping [17]. Support of traffic shaping is not mandatory, but is highly recommended. In addition, the switches can optionally support jumbo frames, IP-multicast routing and VLANs.

The communication links are reliable, with a minimal chance of packet loss. The main source of packet loss is a buffer overflow rather than a link error. In Section 6, we discuss the fault tolerance issues. We assume that the participating group of nodes is already known. Dynamic group technology can be used to deal with changes in group membership, although this case is not considered in this paper.

4 Problem Definition

The main goal of our study is to provide an efficient mechanism for total ordering of messages. Extensive classification of total order definitions and algorithms can be found at [10]. Informally, the primitive insures that messages sent to a set of processes are delivered by all these processes in the same total order.

Most algorithms attempt to guarantee the order required by a replicated database application, namely, *Uniform Total Order (UTO)* defined in [23] by the following primitives:

- **UTO1 - Uniform Agreement :** If a process (correct or not) has UTO-delivered(m), then every correct process eventually $UTO - delivers(m)$.
- **UTO2 - Termination :** If a correct process sends m, then every correct process eventually delivers m according to UTO.
- **UTO3 - Uniform Total Order :** Let m_1 and m_2 be two sent messages. It is important to note that $m_1 < m_2$ if and only if a node (correct or not) delivers m_1 before m_2. Total order ensures that the relation "$<$" is acyclic.
- **UTO4 - Integrity :** For any message m, every correct process delivers m at most once, and only if m was previously broadcasted.

Our system guarantees not only UTO in accordance with the above definition, but also FIFO for each process.

- **FIFO Order :** If m_1 was sent before m_2 by the same process, then each process delivers m_1 before m_2.

Before a UTO is agreed on, a Preliminary Order (PO) is "proposed" by each of the processes. If the PO is identical for all correct (non-faulty) processes, it is called Total Order (TO). PO and TO should either be confirmed or changed by the UTO later.

5 Implementation

As noted above, our implementation of Total Ordering follows the methodology based on a sequencer-based ordering. However, we implement this sequencer using off-the-shelf hardware which is comprised of two Ethernet switches and two Network Interface Cards (NICs) per node. For the simplicity of presentation, we assume that all the nodes are directly connected to the two switches. However, our algorithm can work in an arbitrary network topology, as long as the topology maintains a simple constraint: all the paths between the set of NICs for transmission (TX) and the set of

NICs for reception (*RX*) share (intersect in) at least one link (see Section 8 for scalability discussion).

We assume that all the network components preserve FIFO order of messages. This implies that, once a packet gets queued in some device, it will be transmitted according to its FIFO order in the queue. It is noteworthy that if QoS is not enabled on a switch, the switch technology ensures that all the frames received on a network interface of the switch and egressing via the same arbitrary outgoing link, are transmitted in the order they had arrived; i.e., they preserve the FIFO property. We verified this assumption and found that most switches indeed comply with it, the reason being that the performance of TCP depends on it. Similarly to TCP, our algorithm makes use of this feature for performance optimization, but does not require it for the algorithm correctness.

In our implementation, multicast is used in order to efficiently send messages to the nodes' group. Our goal is to cause all these messages to be received in the same order by the set of nodes that desire to get them (the receivers group). To achieve this, we dedicate a single link between the two switches on which the multicast traffic flows. Figure 1 shows the general network configuration of both the network (the switches) and the attached nodes. The methodology of the network is such that all the nodes transmit frames via a single NIC (TX NIC connected to the "left" switch in the figure) and receive multicast traffic only via the other NIC (RX NIC connected to the "right" switch in the figure). This ensures that received multicast traffic traverses the link between the switches. Since **all** multicast traffic traverses a single link, thus all the traffic is transmitted to the nodes in the same order via the second switch. As the switches and the links preserve the FIFO order, this in turn implies that all the messages are received in the same order by all the nodes.

In a general network setting, there is a chance, albeit a small one, that a message omission may occur due to an error on the link or a buffer overflow (e.g. in the NIC, OS or in the switch). In a collision-free environment (like full-duplex switched environment), a link error is very rare. In addition, buffer overflow can be controlled using a flow control mechanism. Thus, the hardware mechanism enhanced with the proposed flow control (described in the next section), ensures, with high probability, the same order for all received messages. Ways to handle message omission when faults occur are discussed in Section 6.

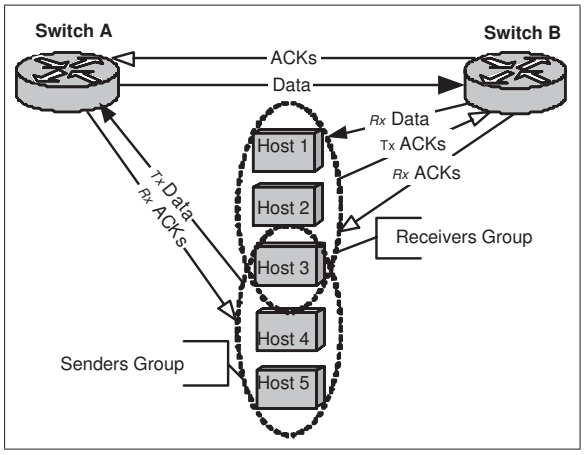

Figure 1. Architecture

5.1 Providing UTO

The preliminary ordering of the hardware configuration is not enough to ensure UTO because messages may get lost or nodes may fail. To address this issue, our protocol uses a *simple* positive acknowledgment (ACK) scheme (similar to the TCP/IP protocol) to ensure that the PO is identical at all the receivers. Each receiver node UTO-delivers (see Section 4) a message to the application only after it has collected ACKs from each receiver node in the system. In order to reduce the number of circulating auxiliary control messages in the system, the ACKs are aggregated according to a configurable threshold parameter. If the system settings are such that each sender node is also a receiver, the ACK messages can be piggybacked on regular data messages.

For the sake of reliability, the sender node needs to hold messages for some period of time. This implies that sender nodes need to collect ACK messages, even though they do not deliver messages to the application. The ACK messages are used by a flow control mechanism (termed as *local* flow control in [18]) in order to maintain the transmission window. Each sender node is allowed to send the next data message only if the number of messages which were originated *locally* and are still unacknowledged by all the receiver nodes is less than a defined threshold value (the transmission window size). Since the ACKs are aggregated, the number of messages that could be sent each time may vary.

In order to increase the performance for small messages, a variation of a Nagle algorithm [20] is used as described in Section 5.2.1. Since the main source of message losses is buffer overflow, careful tuning of the

flow control mechanism combined with ACKs aggregation can reduce the risk of losing messages. For our particular configurations, we identified the appropriate combination of the window size and the number of aggregated ACKs to achieve maximum throughput. The specific implementation of the flow control mechanism presented in this paper allows overall performance to converge with the receiving limit of the PCI bus.

5.2 Optimizations for Achieving High Performance

Various applications may be characterized by different message sizes and packet generation rates. For example, one application may be in a SAN environment in which it is reasonable to assume that the traffic can be characterized by a very large number numer of small messages (where the messages carry meta-data, i.e. a lock request). Another application can be a "Computer Supported Cooperative Work" (CSCW) CAD/CAM, in which data messages may be large. In view of these modern applications, the need to achieve high performance is obvious. Below, a description is presented of the mechanisms and techniques we have implemented and measured in order to reach that goal.

5.2.1 Packet Aggregation Algorithm

It was stated by [11] that at high loads, message packing is the most influential factor for total ordering protocols. We use an approach similar to that in the Nagle algorithm [20], in order to cope with a large number of small packets. Only the messages whose transmission is deferred by flow control are aggregated in buffers. The most reasonable size of each buffer is the size of an MTU. When the flow control mechanism shifts the sliding window by n messages, up to n "large" messages will be sent.

5.2.2 Jumbo frames

The standard frame size in Gigabit Ethernet is ∼1512 bytes. The size of the jumbo frame is ∼9000 bytes. Numerous studies show MTU size has an impact on the overall performance, such as [7], which reports increased performance for jumbo frames. The main reasons for the performance improvement include:

- lower number of interrupts (when moving the same amount of data) and
- less meta-data overhead (headers).

In order to fully benefit from the use of jumbo frames, all components of the system should be configured to support it; otherwise, fragmentation occurs. Since we control all the components in the proposed system, we avoid this problem. Performance results prove that jumbo frames allow to obtain better throughput. For example, in the configuration of two senders and three receivers we achieve a maximum throughput of 722Mb/s.

5.3 Multicast Implementation Issues

As mentioned above, every node is dual-homed, i.e. is connected to the network with two NICs. In the IP multicast architecture, a packet accepted on some interface must be received on the same interface from which the node sends unicast traffic towards the source of the multicast packet. This condition is called the Reverse-Path-Forwarding (RPF) test, which is performed in order to detect and overcome transient multicast routing loops in the Internet. However, this poses a problem for our network settings, since we intend to receive the multicast traffic from the RX NIC while we are transmitting it from the TX NIC. There are several options for overcoming this difficulty, including:

- disabling the RPF test on the particular node;
- ensuring that the source address of the multicast packets has the same subnet portion as the NIC on which it is received (i.e., the RX NIC in our case).

We used the second approach and modified the RX flow in the NIC driver, so that it spoofs the source IP address of the packet. Another issue related to the usage of IP multicast in our settings is that self-delivery of multicast packet is usually done via *internal loopback*. Packets that are sent by the local host and are supposed to be received by it, are usually delivered immediately by the operating system. We disabled this feature, so that ALL delivered packets are received via the RX NIC and thus all the packets pass through the same delivery process (thus ensuring that total order is maintained).

6 Fault-Tolerance

Failures can be caused by different sources: a switch failure, a physical link disconnection, a failure of a process and a crash of a node running the process. All these failures can be identified by failure detectors.

We implemented a loss-of-packet failure handling. The proposed algorithm contains a built-in method for identifying and retransmitting a missing packet by introducing a leader node whose order takes over when a

conflict occurs. It is noteworthy that nodes in our system do not wait for the leader's ordering in failure-free scenarios. In Technical Report [1] we elaborate on combaning our algorithm with either Virtal Synchrony [9] or Paxos [16].

An alternative approach was presented by Pedone et al. [22]. The authors define a weak ordering oracle as an oracle that orders messages that are broadcast, but is allowed to make mistakes (i.e., the broadcast messages might be delivered out of order). The paper shows that total-order broadcast can be achieved using a weak ordering oracle.

In [1] we consider fault tolerance in greater detail as well as present a solution for overcomming failures of links and/or switches.

7 Performance

This section presents the results of the experiments performed to evaluate the architecture. The following configuration was used:

1. **Five end hosts**: Pentium-III/550MHz, with 256 Mb of RAM and 32 bit 33 MHz PCI bus. Each machine was equipped also with two Intel®Pro/1000MT Gigabit Desktop Network Adapters. The machines ran Debian GNU/Linux 2.4.25.

2. **Switches**: Two Dell PowerConnect 6024 switches, populated with Gigabit Ethernet interfaces. These switches are "store and forward" switches (i.e., a packet is transmitted on an egress port only after it is fully received).

The experiments were run on an isolated cluster of machines. For each sample point on the graphs below and for each value presented in the tables, the corresponding experiment was repeated over 40 times with about 1 million messages at each repetition. We present the average values with confidence intervals of 95%. Unless otherwise specified, the packet size in the experiments was about 1500 bytes (we also experimented with small packets and with jumbo frames). The throughput was computed at the receiver side as $\frac{packet\ size \times average\ number\ of\ delivered\ packets}{test\ time}$. In order to simulate an application, we generated a number of messages at every configurable time interval. However, in most Operating Systems, and in particular in Linux 2.4, the accuracy of the timing system calls is not sufficient to induce the maximal load on the system. We therefore implemented a traffic generation scheme that sends as many messages as possible after each received ACK. Since the ACKs were aggregated, the size of the opened flow control window varied each time.

The paper hereafter refered to as Technical Report.

7.1 Theoretical bounds

It is important to observe that, regardless of the algorithm used to achieve the Total Order of messages, there are other system factors that limit the overall ordering performance. One of the bottlenecks that we encountered resulted from the PCI bus performance. In [24] it is shown that the throughput achieved by PCI bus in the direction from the memory to the NIC is about 892Mb/s for packets of 1512 bytes size and about 1 Gb/s for jumbo frames. However, a serious downfall in the PCI bus performance was detected in the opposite direction, when transferring the data from the NIC to the memory. The throughput of 665Mb/s only for packets of 1512 bytes size and 923Mb/s for jumbo frames was achieved. Thus, the throughput allowed by PCI bus imposed an upper bound on the performance of a receiver node in our experiments. There are various studies on PCI bus performance, e.g. [19], which suggest several benchmarks and techniques for tuning. It will be shown later that our solution approximates the theoretical and experimental upper bounds of PCI bus. In future work, we plan to evaluate our architecture over PCI Express whose throughput is higher and is thus to yield significantly better performance.

We first discuss the best throughput results obtained for each configuration. The latency obtained per result is presented as well. Two types of configurations were used: those where all the nodes were both senders and receivers (all-to-all configurations), and those in which the sets of senders and receivers were disjoint. It is important to note that for some configurations, such as the all-to-all configuration and the experiments with the jumbo-frames, we utilized the traffic shaping feature of the switching device, namely the one that is connected to the TX NICs. This ensured that no loss occurred on a node due to the PCI bus limitations. The value serving to limit the traffic was selected by measuring maximum achievable throughput for each setting. The main benefit of using traffic shaping is the limit it imposes on traffic bursts that were the major cause of packet drops in our experiments.

7.1.1 All-to-all Configurations

Results for all-to-all configurations and configurations with dedicated senders are discussed separately, since when a node serves as both a sender and a receiver, the CPU and PCI bus utilization patterns differ, and the node is overloaded.

Nodes Number	Throughput *Mb/s*	PO Latency *ms*	UTO Latency *ms*
3	310.5 (0.08)	4.2 (0.03)	6.5 (0.03)
4	344.4 (0.04)	4.4 (0.02)	6.8 (0.02)
5	362.5 (0.09)	4.1 (0.02)	6.7 (0.02)

Table 1. Throughput and Latency for all-to-all configuration

Table 1 presents throughput and latency measurements for all-to-all configurations, along with the corresponding confidence intervals shown in parentheses. The nodes generate traffic at the maximum rate bound by the flow control mechanism. Two different latency values are presented: PO Latency and UTO Latency. PO Latency is defined as the time that elapses between transmission of message by a sender and its delivery by the network back to the sender. UTO Latency is defined as the time elapsed between a message transmission by a sender and the time the sender receives ACKs for this message from every receiver.

The number of the nodes that participated in this experiment increases from 3 to 5. As presented in Table 1, the achieved throughput increases with the number of participating nodes. This is accounted for by the PCI bus behavior (See Section 7.1). Since each node both sends and receives data, the load on the PCI is high, and the limitation is the boundary on the total throughput that can go through the PCI bus. As the number of nodes grows, the amount of data each individual node can send decreases. When a node sends less data, the PCI bus enables it to receive more data. The nonlinearity of the increase in throughput in this experiment can be attributed to the above mentioned property of the PCI bus, where the throughput of transferring data from memory to NIC is higher than in the opposite direction.

7.1.2 Disjoint Groups of Senders and Receivers

Table 2 presents the performance results of throughput measurements for disjoint sets of nodes. We used 2-5 nodes for various combinations of groups of senders and receivers. The maximum throughput of ∼512.7Mb/s was achieved. In the trivial configuration of a single sender and a single receiver, the result is close to the rate achieved by TCP and UDP benchmarks in a point-to-point configuration, where the throughput reaches 475Mb/s and 505Mb/s, respectively. The lowest result was registered for a single sender and four receivers, the achieved throughput of 467Mb/s not falling far from the best throughput.

For a fixed number of receivers, varying the number of senders yields nearly the same throughput results. For a fixed number of senders, increasing the number of receivers decreases the throughput. The reason is that a sender has to collect a larger number of ACKs generated by a larger number of receivers. It is noteworthy that the flow control mechanism opens the transmission window only after a locally originated message is acknowledged by all the receiver nodes. Possible solutions to this problem are discussed in Section 8.

Table 3 presents the results of UTO latency measurements at the receiver's side. As can be seen, in case of a fixed number of senders, increasing the number of receivers increases the latency. The explanation is similar to that for the throughput measurement experiments: the need to collect ACKs from all the receivers. Increasing the number of senders while the number of receivers is fixed causes an increase in the UTO Latency. Our hypothesis is that this happens due to an increase in the queues both at the switches and at the hosts.

As was mentioned above, in case a node either sends or receives packets, the utilization of the PCI bus and other system components is different from the case when a node acts as both a sender and a receiver. For this reason, the results presented in this section cannot be compared with those described above.

7.2 Tradeoffs of Latency vs. Throughput

In this section, we discuss the impact of an increased load on latency. In order to study the tradeoff of Latency vs. Throughput, a traffic generation scheme different from that in the previous experiments was used. The scheme was implemented by a benchmark application that generated a configurable amount of data.

7.2.1 All-to-all Configuration

In this section, all-to-all configuration is considered. Figure 2 shows the latencies for the 5-node configuration. Obviously, the UTO latency is always larger than

	Receivers			
Senders	1	2	3	4
1	512.7 (0.47)	493.0 (0.17)	477.0 (0.34)	467.1 (0.40)
2	512.5 (0.27)	491.7 (0.67)	475.7 (0.33)	
3	510.0 (0.55)	489.6 (0.41)		
4	509.2 (0.30)			

Table 2. Throughput (Mb/s) for different configurations

	Receivers			
Senders	1	2	3	4
1	2.3 (0.003)	3.1 (0.035)	3.2 (0.045)	3.1 (0.012)
2	2.5 (0.002)	3.1 (0.025)	3.4 (0.040)	
3	3.2 (0.004)	3.6 (0.041)		
4	4.9 (0.003)			

Table 3. UTO Latency (ms) for different configurations

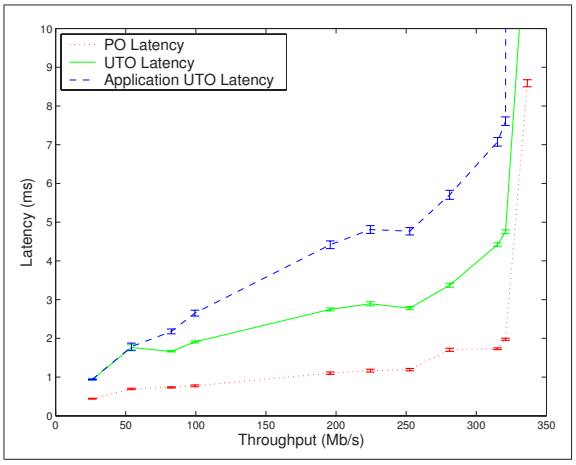

Figure 2. Latency vs. Throughput (all-to-all configuration)

the PO latency. One can see an increase in the latencies when the throughput achieves the 50Mb/s value, i.e. a point from which small transient packet backlogs were created, and then a slight increase until the throughput approaches about 250Mb/s. After this point, the latencies start increasing. The PO latency reaches the value of about $1ms$ and UTO of about $3ms$ for throughput of about 330Mb/s.

We also measured the Application UTO Latency, which is the time interval from the point when the application sent a message until it can be "UTO delivered". One can see that when throughput increases, the Application UTO Latency increases too. This happens because the Linux 2.4 kernel allows events to be scheduled with a minimal granularity of $10ms$. Thus, in order to generate a considerable load, the benchmark application has to generate an excess number of packets every $10ms$. Packets that are not allowed to be sent by the flow control mechanism are stored in a local buffer data structure. When ACKs arrive, the flow control mechanism enables sending some more packets previously stored for transmission. Packets that cannot be immediately sent increase the Application UTO Latency.

7.2.2 Large Packet Sizes

Figure 3 shows how increasing the application packet size, along with increasing the MTU size, affects the Application UTO Latency. In this experiment, we used disjoint groups of two senders and three receivers. We compared results achieved for jumbo frames with those obtained for regular Ethernet frames of MTU size. As expected, in case of jumbo frames, a larger throughput can be achieved, mainly due to the significantly reduced amount of PCI transactions.

When throughput increases, the Application UTO Latency increases, too, the reasons being the same as for the "all-to-all configuration". One can see that at lower throughput values, the jumbo frames show higher latency. This can be attributed to the fact that when

The latest versions of Linux support $1ms$ granularity

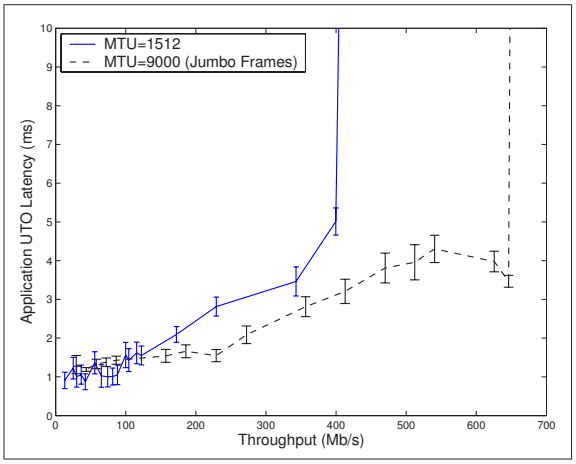

Figure 3. Latency vs. Throughput for different MTU sizes

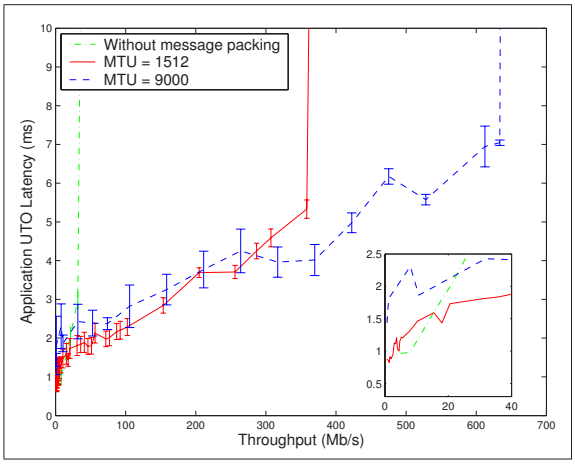

Figure 4. Packet aggregation (the low throughput area is extended)

the system is relatively free, the high transmission latency of jumbo frames dominates; in other words, the time for putting a jumbo frame on the wire is larger. As the load on the system increases, the overhead of the PCI bus and packet processing becomes the dominating factor, and using jumbo frames helps to reduce this overhead and thus to achieve the UTO faster.

7.2.3 Packet aggregation

The experiment evaluated the effect of using the packet aggregation algorithm described in 5.2.1. Figure 4 shows the performance of the system with small packets, the payload size being about 64 bytes. Two accumulating packet sizes were used, Ethernet MTU of 1500B and jumbo frame size of 9000B. In addition, the same tests were conducted without packet aggregation. Since the throughput without packet aggregation is considerably smaller, in the same figure the area corresponding to the throughput values between 0 and 40Mb/s is shown. One can see that the maximum throughput without packet aggregation is about 50Mb/s. On the other hand, using an accumulating size of 1500B increased the maximum throughput up to 400Mb/s. With accumulating size of jumbo frames, the throughput climbed as high as 630Mb/s, which is about one million small packets per second.

Comparing corresponding curves in Figures 3 and 4, one can see that packet aggregating causes a higher latency and a lower maximum achievable throughput. It could be explained by the amount of CPU resources spent on aggregating the messages.

7.3 Comparisons with previous works

There are only few papers that evaluate performance of total order algorithms over real networks. The rapid advancement of networking technology in recent years often makes the comparison irrelevant. For example, [11] presented performance evaluations of several total order algorithms. However, the measurements were carried out on a shared 10 Mb/s Ethernet network, which is 100 times slower than Gigabit Ethernet which is widely deployed today.

In the experiment described below, we compared the performance of our system with results of an algorithm based on weak ordering oracles ([22], described in Section 6), and of an algorithm based on failure detectors [6]. When carrying out the measurements for the comparative experiment, we tried to provide similar settings. All links were configured to 100Mb/s rate, the message size was 100 bytes, no message packing was used and the aggregation of ACKs limit was set to 3. The experiments in [22] were performed at 4 nodes for weak ordering oracles and at 3 nodes for the algorithm based on failure detectors. In our experiments, we used 4 nodes. Since the main parameters of the experiments under comparison coincide, while there might be differences in equipment and implementation environments, it is likely that the approximation is sufficient.

As the comparison presented in [22] shows, the maximum throughput for both algorithms was 250 messages per second. The latency of the weak ordering oracle algorithm increased from about 2.5s for the throughput

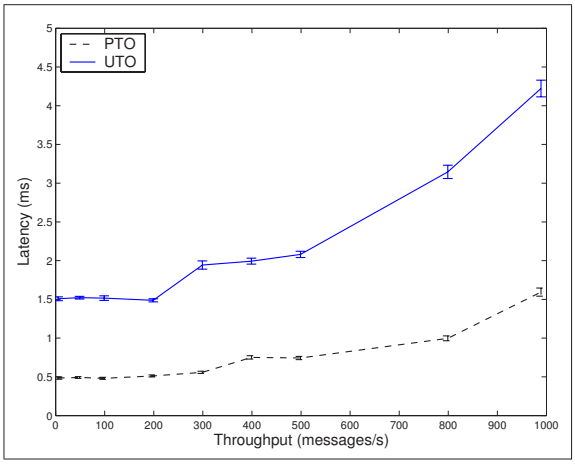

Figure 5. Number of ordered messages/s in 100Mb/s network.

of 50 messages/sec up to about $10ms$ for the throughput of 250 messages/sec. The performance of the algorithm based on failure detectors depends largely upon the timeout set for heartbeat messages. For large timeout of about $100ms$, the latency was within the range of 1.5-2ms, and for small timeout ($2ms$) the latency was within the range of 8-10ms.

Figure 5 presents the results of our experiments in 100Mb/s network and shows that the throughput of about 1000 messages/sec was achieved. The throughput of 300 messages/sec induces the PO latency of about $0.7ms$, and the UTO latency was within the range of 1.7-2.2ms. The 95%-confidence interval was also computed and found practically negligible, as one can see in the graphs. It is important to note that while for low throughput our results do not differ significantly from those achieved by Pedone et al. [22], for a high throughput we reach lower latency. The reason is that in our system, order is not distrupted even if a message m is lost, as losses happen mostly in switch A (see Figure 1). So, if m is missed by a process, there is a high probability that m is lost by all the processes, and PO order remains the same among all the processes. When m's sender discovers that m is lost, it retransmits m promptly.

Another question is whether the propagation time of a message in our two-switch topology is much higher than in a one-switch topology. Theoretically, the propagation time in a Gigabit network over a single link is $\frac{1500*8}{10^9}$=0.012ms, the speed of signal transmission over the cable is negligible, and the maximum processing time in the switch that we used is not more than 0.021ms. We performed two experimental measurements of propagation time. In the first experiment, the ping utility was used to measure the latency of 1500-size packet, and 0.05ms propagation time was obtained in both topologies. In the second experiment, we used application level ping based on UDP protocol, as opposed to the original ping utility which works on kernel level. In the application level ping, we registered 0.12ms latency in both topologies. The results show that packet processing time ($\sim$0.1ms) is much higher than message propagation time ($\sim$0.012ms). We can conclude, therefore, that the two-switch topology, without significantly increasing the latency, allows to predict message order with much higher probability!

8 Scalability

The performance evaluation presented above was carried out only for up to five nodes. This evaluation proves that the architecture can be efficient for small systems. As for larger systems, our measurements showed only a small degradation of throughput (about 0.5% per sender) when the number of senders increases. However, increasing the number of receivers decreases the throughput. In Technical Report [1], we discuss a few solutions for making the system scalable when the number of receivers increases. We also prove that the number of ports in the switches does not put any limitations on our apprach, and discuss the issue of providing support for multiple groups [1].

9 Related Work

There are a number of works which deal with the problem of Total Ordering of messages in a distributed system. A comprehensive survey of this field, covering various approaches and models, can be found in [10]. The authors distinguish ordering strategies based on where the ordering occurs, i.e. senders, receivers or sequencers. We use the network as a virtual sequencer, so our algorithm falls into the latter category.

As was noted in Section 1, the approach called "Optimistic Atomic Broadcast" was first presented in [21]. In [8], an interesting approach to achieving total order with no message loss was presented. The authors introduced buffer reservation at intermediate network bridges and hosts. The networking equipment connecting the senders and receivers was arranged in a spanning tree. The reservation was made on the paths in the spanning tree so that no message loss could occur. The ordering itself was performed using Lamport timestamps [15]. The paper assumed a different net-

work and presents only simulation results, which makes it hard to perform any comparisons.

An implementation of a Total Ordering algorithm in hardware was proposed in [13]. This work offloads the ordering mechanism into the NIC and uses CSMA/CD network as a virtual sequencer. The authors assume that a single collision domain connects all the participating nodes. Using special software and hardware, the algorithm prevents nodes that missed a message from broadcasting new messages, thus converting the network to a virtual sequencer. In our opinion, the use of a single collision domain is the main drawback of this approach, as it is known that collisions may significantly reduce the performance of system.

Another work that deals with Total Ordering and hardware is presented in [4]. In this work, a totally ordered multicast which preserves QoS guarantees is achieved. It is assumed that the network allows bandwidth reservation specified by average transmission rate and the maximum burst. The algorithm suggested in the paper preserves the latency and the bandwidth reserved for the application.

References

[1] T. Anker, G. Greenman, D. Dolev, and I. Shnayderman. Wire-Speed Total Order. Technical report, January 2006.

[2] ANSI. Fibre Channel Physical and Signaling Interface (FC-PH). X3.230-1994.

[3] I. T. Association. InfiniBand Architecture Specification. Release 1.2.

[4] Z. Bar-Joseph, I. Keidar, T. Anker, and N. Lynch. Qos preserving totally ordered multicast. In *Proc. 5th International Conference On Principles Of Distributed Systems (OPODIS)*, pages 143–162, December 2000.

[5] R. Burns. *Data Management in a Distributed File System for Storage Area Networks.* PhD thesis, March 2000. University of California, Santa Cruz.

[6] T. D. Chandra, V. Hadzilacos, and S. Toueg. The weakest failure detector for solving consensus. In M. Herlihy, editor, *Proceedings of the 11th Annual ACM Symposium on Principles of Distributed Computing (PODC'92)*, pages 147–158, Vancouver, BC, Canada, 1992. ACM Press.

[7] J. Chase, A. Gallatin, and K. Yocum. End system optimizations for high-speed TCP. *IEEE Communications Magazine*, 39(4):68–74, 2001.

[8] X. Chen, L. E. Moser, and P. M. Melliar-Smith. Reservation-based totally ordered multicasting. In *Proc. 16st Intl. Conf. on Distributed Computing Systems (ICDCS-16)*. IEEE CS, May 1996.

[9] G. V. Chockler, I. Keidar, and R. Vitenberg. Group communication specifications: a comprehensive study. *ACM Comput. Surv.*, 33(4):427–469, 2001.

[10] X. Defago, A. Schiper, and P. Urban. Total order broadcast and multicast algorithms: Taxonomy and survey. *ACM Computing Surveys*, 36(4):372–421, December 2004.

[11] R. Friedman and R. van Renesse. Packing messages as a tool for boosting the performance of total ordering protocols. In *HPDC*, pages 233–242, 1997.

[12] G. Greenman. Msc. thesis: High speed total order for san infrastructure, June 2005. The Hebrew University of Jerusalem, http://www.cs.huji.ac.il/labs/danss/.

[13] P. Jalote. Efficient ordered broadcasting in reliable csma/cd networks. In *Proc. 18st Intl. Conf. on Distributed Computing Systems (ICDCS-18)*. IEEE CS, May 1998.

[14] B. Kemme, F. Pedone, G. Alonso, and A. Schiper. Processing transactions over optimistic atomic broadcast protocols. In *Proceedings of 19th International Conference on Distributed Computing Systems (ICDCS'99)*, 1999.

[15] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7):558–565, 1978.

[16] L. Lamport. The part-time parliament. *ACM Transactions on Computer Systems*, 16(2):133–169, 1998.

[17] M. Christensen, K. Kimball, F. Solensky. Considerations for IGMP and MLD Snooping Switches. IETF draft.

[18] S. Mishra and L. Wu. An evaluation of flow control in group communication. *IEEE/ACM Trans. Netw.*, 6(5):571–587, 1998.

[19] L. Moll and M. Shand. Systems performance measurement on PCI pamette. In K. L. Pocek and J. Arnold, editors, *IEEE Symposium on FPGAs for Custom Computing Machines*, pages 125–133, Los Alamitos, CA, 1997. IEEE Computer Society Press.

[20] J. Nagle. Congestion Control in TCP/IP Internetworks. RFC 896, January 1984.

[21] F. Pedone and A. Schiper. Optimistic atomic broadcast. In *Proceedings of the 12th International Symposium on Distributed Computing*, pages 318–332. Springer-Verlag, 1998.

[22] F. Pedone, A. Schiper, P. Urban, and D. Cavin. Solving agreement problems with weak ordering oracles. In *EDCC-4: Proceedings of the 4th European Dependable Computing Conference on Dependable Computing*, pages 44–61. Springer-Verlag, 2002.

[23] P. Vicente and L. Rodrigues. An indulgent uniform total order algorithm with optimistic delivery. In *Proc. 21st IEEE Symposium on Reliable Distributed Systems*. IEEE CS, October 2002.

[24] W. Wadge. Achieving gigabit performance on programmable ethernet network interface cards. 2001. http://www.cs.um.edu.mt/ ssrg/wthesis.pdf.