Copyright Notice

Institute of Communication Networks and Computer Engineering
University of Stuttgart
Pfaffenwaldring 47, D-70569 Stuttgart, Germany
Phone: +49-711-685-68026, Fax: ++49-711-685-67983
email: mail@ikr.uni-stuttgart.de, http://www.ikr.uni-stuttgart.de

Quick-Start and XCP on a Network Processor: Implementation Issues and Performance Evaluation

Simon Hauger, Michael Scharf, Jochen Kögel, Chawapong Suriyajan
Institute of Communication Networks and Computer Engineering
University of Stuttgart, Pfaffenwaldring 47, 70569 Stuttgart, Germany
Email: simon.hauger@ikr.uni-stuttgart.de

***Abstract*—The Quick-Start extension of the Transmission Control Protocol (TCP), as well as the Explicit Control Protocol (XCP), are experimental congestion control schemes that use router feedback to overcome limitations of TCP's standard mechanisms. Both approaches require additional packet processing in every router and therefore raise the question whether, and how, this can be achieved in high-speed routers.**

This paper studies the realization complexity of the Quick-Start and XCP router functions on a network processor. We show that in both cases synchronization issues among parallel processing entities have to be considered, and that this affects the router performance. We develop and compare different synchronization mechanisms for highly parallel packet processing. Our prototype implementation on an Intel IXP network processor allows to quantify the impact on throughput and delay caused by the additional packet processing in the fast path. The measurements reveal that Quick-Start and XCP processing is feasible at multiple Gbit/s line speed, with Quick-Start being simpler to scale.

I. Introduction

Most Internet applications use the Transmission Control Protocol (TCP) for reliable, best effort transport. The TCP sending behavior is governed by the congestion control, which continuously probes the available path capacity and reduces the sending rate when packet loss occurs. However, the TCP congestion control is challenged by new broadband network technologies that result in large bandwidth-delay products. Recent research has shown that in this case a more expressive congestion feedback from routers along the path can improve link utilization, per-flow performance, and inter-flow fairness.

An existing example for router-assisted congestion control is Explicit Congestion Notification (ECN). Several recent research activities generalize this one-bit congestion occurrence feedback towards a more expressive in-band notification on path characteristics. Two proposals considered by the IETF are Quick-Start TCP [1] and the Explicit Control Protocol (XCP) [2]. Quick-Start is an experimental TCP extension that allows connection end-points to ask the routers along the path for an allowed sending rate, in particular at the beginning of data transfers. XCP is designed to completely substitute the TCP congestion control by giving routers a continuous control of the sending rate of ongoing data transfers.

Since both approaches require not only changes in end-systems, but also additional packet processing in the network, a key question is whether, and how, such mechanisms could be integrated in routers with line speeds of multiple Gbit/s. Several simulation studies such as [3] or [4] show that both approaches have the potential to overcome short-comings of TCP's congestion control. Furthermore, it has been proven that it requires only limited effort to extend the software stacks of operation systems to support Quick-Start [5], or XCP [6], respectively. However, the realization of Quick-Start and XCP functions in core routers is a rather unexplored field so far.

Network processors are increasingly used in core routers. They use specialized processor cores to achieve high-speed packet processing. Due to their programmability, they allow a flexible realization of new protocol functions, as required by Quick-Start and XCP. The Intel IXP network processor series is widely used for experimental purposes. For instance, [7] uses an Intel IXP2400 for a partial XCP implementation, and [8] for other related implementation efforts. An alternative are Field-Programmable Gate Arrays (FPGA), e. g. [9] reports an FPGA-based implementation of another router-assisted congestion control scheme. FPGAs require a more complex, low-level programming and are not further considered here.

In this paper, we study the implementation complexity and performance of Quick-Start and XCP on a network processor. We show that in both cases the needed functions lay restrictions on parallel packet processing, and we discuss solutions to circumvent these problems. To the best of our knowledge, this is the first comprehensive implementation and performance study of Quick-Start and XCP on a network processor.

The rest of this paper is structured as follows: Section II gives an overview of router-assisted congestion control schemes, in particular Quick-Start and XCP, and also introduces router realization aspects. In Section III, we present our implementation for the Intel IXP network processor series. We detail the challenges that arise from parallel processing, and discuss possible solutions. The measurements results in Section IV compare the throughput and delay performance of Quick-Start and XCP. Finally, Section V concludes the paper.

II. Router-Assisted Congestion Control

Several research proposals aim at improving the Internet congestion control by using explicit feedback from routers. This paper studies the Quick-Start TCP extension and XCP, which are described in detail in IETF documents [1], [2]. A survey of other related work can be found in [1], too.

This work was partly funded within the EIBONE project KOMMT!flexRN by the German Bundesministerium für Bildung und Forschung under contract No. 01BP566.

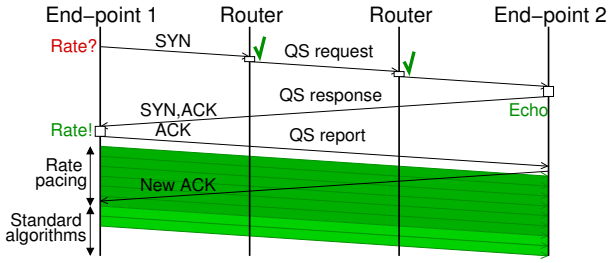

Fig. 1. Quick-Start (QS) request during the 3-way TCP handshake

A. Quick-Start TCP Extension

The standard TCP congestion control has difficulties to determine an appropriate sending rate after connection setup, or e. g. after long idle periods, since the path characteristics are unknown at this point in time. Traditionally, TCP uses the slow-start heuristic to probe the available bandwidth in these cases, but this is time-consuming in high-speed wide-area networks. The experimental Quick-Start extension addresses this issue by using feedback from the routers on the path: It allows hosts to ask for an initial sending rate, e. g., during the TCP three-way handshake. If the request is approved, avoiding the slow-start can significantly speed up data transfers, in particular in large bandwidth-delay product networks.

Fig. 1 illustrates a Quick-Start request during TCP connection establishment: The originator adds a *Quick-Start request* option to the IP header, which indicates a desired sending rate. This target rate is encoded in 15 coarse-grained steps ranging from 80 kbit/s to 1.31 Gbit/s. The routers along the path can approve, reduce, or disallow this request. If the request arrives at the destination, the granted rate is echoed back piggybacked as a TCP option (*Quick-Start response*). The originator can then detect whether all routers along the path have explicitly approved the request. If not, the default congestion control (TCP slow-start) is used to ensure backward compatibility. If the Quick-Start request is successful, the originator can immediately increase its congestion window and start to send with the approved rate, using a rate pacing mechanism. After one round-trip time the default TCP congestion control mechanisms are used for subsequent data transfers. Quick-Start does not guarantee any data rate, i. e., it is a light-weight speed-up mechanism for elastic best effort traffic only.

The support of Quick-Start requires two additional functions in routers: First, a router has to know the available bandwidth on the outgoing links and must keep track of their utilization. And second, Quick-Start requests must be processed by an admission control that decides whether to accept a Quick-Start request, and which data rate to grant. More details about the required functions can be found in [1], [3].

B. Explicit Control Protocol (XCP)

XCP is the outcome of research efforts towards a new congestion control for a Future Internet. Its design principle is to give routers a fine-grained control over sending rates of flows. This allows to achieve high link utilization, fast convergence and fairness in high bandwidth-delay product networks. Unlike Quick-Start, XCP is not a TCP extension, but specifies a new protocol layer on top of the Internet Protocol (IP). The sender puts a congestion header in *every* IP packet, which carries information about the traffic characteristics of the flow. Each router processes this XCP header, performs some mathematical calculations, and adds information whether to increase or decrease the sending rate of the flow. The receiver returns the feedback to the sender by piggybacking.

XCP routers must have knowledge of the link capacities and internal queue statistics. XCP does not require any per-flow state, but routers must periodically recalculate some local parameters to ensure fairness. Details about the algorithms, as well as some remaining open issues, are explained in [2]. The underlying control theory is introduced in [4].

C. Architecture of Routers

Both, Quick-Start and XCP, require all routers on the path to be extended beyond basic IP forwarding functionality. This is especially challenging for core routers, which have to process up to 125 million packets per second (Mpps) per port in 40 Gbit/s networks. Core routers usually have a modular architecture, where packets travel from the ingress line card through the switch fabric to the egress line card. The ingress line card performs packet classification and modification (e. g. route lookup, policing), while queue management (e. g. drop tail, active queue management, scheduling) is typically performed on the egress line card. Smaller routers, e. g. enterprise routers, typically perform all functions monolithically. For more details on router functions and architectures refer to [10], [11].

For packet throughput at speeds of several Gbit/s it is essential that time critical tasks that apply to every packet are performed very fast. These functions of the *fast path* are therefore handled by specialized hardware, namely ASICs and/or network processors. The more flexible network processors integrate several (some models > 100) specialized processing cores for pipelined and/or parallel processing and other specialized hardware units. More details on network processors can be found in [11]. Tasks that have to be performed for a small fraction of packets only, and tasks that demand for more complex processing (e. g. route calculation) are performed on the *slow path* using General Purpose Processors (GPP).

III. Implementation on a Network Processor

Realizing router-assisted congestion control with Quick-Start or XCP implies additional functionality of the routers. We identify issues that have to be solved when implementing these functions on network processors, and we describe our implementation on the Intel IXP2400.

A. Required Additional Router Functions

Router functions are divided into fast path and slow path functions. As the XCP protocol is intended to be used in all packets, the corresponding processing has to be performed in the fast path. In contrast, the Quick-Start option is only contained in few packets. As a consequence, routers might be

able to handle Quick-Start options in the slow path. However this induces additional delays and packet reordering. So, a superior solution is to process the Quick-Start option in the fast path as well. For both XCP and Quick-Start, functions that have to be executed at regular intervals independent of incoming packets are best performed in the slow path.

In the fast path, the additional protocol fields have to be processed. For Quick-Start these are located in an IP option, otherwise normally treated in the slow path. For XCP they are placed in a separate protocol above IP. XCP requires that some global counters are updated initially, according to the fields of the received packet. For both protocols, the field containing the bandwidth request has to be modified depending on global state variables, such as the total granted bandwidth for Quick-Start or the positive and negative capacity for XCP. Accordingly, some of these variables have to be changed, too.

In the slow path, per output port several global variables have to be updated periodically. For Quick-Start the spare bandwidth has to be calculated. The update interval is configurable. For XCP a total of seven global variables need to be changed every average round-trip time.

In modular routers, the utilization of the output ports is measured on the egress line cards. As a consequence, Quick-Start or XCP processing is likely to be implemented there.

B. Synchronization

Fast path hardware in high speed routers, especially in network processors, often processes several packets in parallel. Basic IP forwarding is performed independently on each packet, not leading to any synchronization problems. However, advanced protocol mechanisms such as Quick-Start or XCP violate this property of packet independence, since they require read *and* write operations to global variables, as described in the previous section. As a result, a synchronization mechanism has to be employed. Otherwise more bandwidth than being actually available would be granted to incoming requests.

An effective and well-known mechanism is to use locks, as depicted in Fig. 2a. Only the process that obtains the lock is allowed to enter the critical section and to access the shared variables. After changing the shared variables, it releases the lock so that other processes can access the shared data.

This kind of synchronization mechanism has significant impacts on throughput and delay of the system. The throughput of an unsynchronized pool of n parallel processes with a processing delay of T_d is $n \cdot T_d^{-1}$. However, if we use locking for the synchronization of a critical section of duration T_{crit}, the throughput is limited to T_{crit}^{-1}, respectively, because only one process can complete the critical section at a time. Also the processing delay increases by up to $(n-1) \cdot T_{crit}$ due to the waiting time to obtain the lock. Thus, if critical sections are longer than T_d/n, the throughput is reduced. Furthermore, increasing the parallelism of the system will not increase its throughput but will increase its delay. These effects can be later seen in Section IV, even though we minimized the length of the critical sections in our implementations.

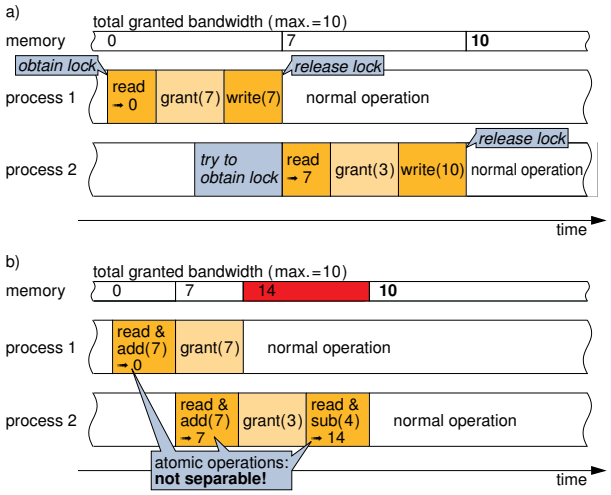

Fig. 2. a) Synchronization with locks, b) Synchr. using atomic operations

Another mechanism for accessing shared variables is the use of atomic operations. An atomic operation cannot be interrupted by another process. For instance, by using the atomic operation 'read & add', a variable is read from memory, subsequently increased by an addend and written back to memory without interrupt by concurrent processes. These atomic operations can be used for implementing Quick-Start: When processing the requested bandwidth, each process atomically adds the complete requested bandwidth to the shared variable holding the total granted bandwidth, as shown in Fig. 2b. If this exceeds the maximum allowable bandwidth, the surplus amount has to be atomically subtracted again.

Only simple arithmetic or logic operations can be performed atomically on the memory. This is why the rather sophisticated processing of the shared variables for XCP packet processing cannot be implemented efficiently with this method.

C. The Intel IXP Network Processor Family

All network processors in Intel's IXP2XXX series comprise the same type of processing cores for fast path processing, the so called microengines (MEs). However, they differ in the number of MEs as well as in clock frequency and interface bandwidth. MEs are simple RISC processors that provide eight hardware-supported threads for hiding memory latency. Exchange of data among MEs and the also embedded GPP can be performed via fast on-chip memory or external SRAM, while external DRAM is mainly used as packet buffer. On-chip and SRAM memory controllers offer atomic operations for realizing synchronization mechanisms and FIFO buffers.

The IXP communication infrastructure allows to map the processing tasks flexibly onto the MEs: distributed as a functional pipeline, as a parallel cluster or in a combined way. Sample mappings are given by the software framework [12] of the manufacturer. Furthermore, it provides building blocks for common processing tasks as well as data structure and interface definitions. For more information about the IXP network processor series refer to [11].

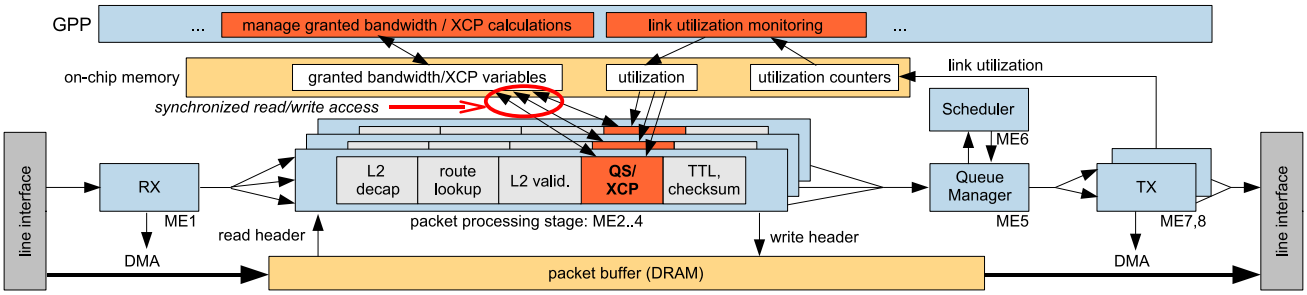

Fig. 3. Realization of Quick-Start/XCP fast path functions on Microengines (ME) and slow path functions on the GPP of an IXP2400

D. Implementation of Quick-Start and XCP

Our implementation is based on the Intel IXP2400 network processor, which contains eight MEs. It was configured as a monolithic router, i. e. ingress and egress line interfaces are directly attached. We extended the standard router application provided by the software framework [12], as shown in Fig. 3.

The standard router works as follows: The RX block transfers incoming packets to DRAM and forwards respective handles to the packet processing stage, consisting of three parallel MEs. Here, each packet header is loaded and several header classification and modification tasks are performed, before the modified headers are written back to DRAM. The handles are delivered in-order to the Queue Manager that cooperates with the Scheduler block. This block decides from which queue to forward a packet handle to the TX block, which finally transfers the packets from DRAM to the egress line interface. Slow path functions are handled in the GPP.

For implementing Quick-Start and XCP, the required fast path functions stated in Section III-A were implemented in the packet processing MEs, as shown in Fig. 3. Here also the synchronized access to the shared variables is performed. Our Quick-Start implementation can be configured to operate using synchronization either by locking or by atomic operations. For the XCP implementation only locking is possible. The slow path functions like the calculation of the egress link utilization were added to the GPP. In Table I the implementation effort for Quick-Start and XCP is compared.

Our implementation of Quick-Start is fully working and has already been used to build up simple network topologies with Quick-Start enabled Linux endpoints. To the best of our knowledge, this is the first comprehensive implementation of Quick-Start on a network processor platform. For XCP we implemented only the fast path functions so far, since these are sufficient to compare the performance of the routers. Neither implementation requires any additional (expensive) DRAM operations compared to the standard router. Neither keeps per-flow state. For further implementation details refer to [13].

TABLE I
COMPARISON OF IMPLEMENTATION EFFORT

	Quick-Start	XCP
Fast path	processing of - few packets (e. g. SYN) - IP option	processing of - all packets - new protocol fields
	modification of global variables	
	no per-flow state	
Slow path	calculation of used output bandwidth	
	periodic update of global variables	
Synchro-nization	locking/atomic operations	only locking possible
	1 variable per port	7 variables per port
	short critical section	long critical section

IV. Performance Evaluation

To evaluate the performance impact of the additional protocol processing, we measured throughput and delay of the Quick-Start and XCP routers in different scenarios.

A. Router Variants

We used three different router variants in our measurements:

- The unmodified 'normal router' without any Quick-Start or XCP support, as provided by the manufacturer's software framework [12] with only minor changes.
- The enhanced router with Quick-Start support ('QS') in both variants with synchronization by locking and by atomic operations ('AOp'). The threshold up to which Quick-Start requests were granted was set to 1 Gbit/s.
- The enhanced router with fast path XCP support. Here all packets were processed as XCP packets.

B. Measurement Methodology

We tested the routers under load by using several traffic generators: Three FPGA-based traffic generators [14] and several Linux PCs. The FPGA-based traffic sources generated Ethernet frames containing simple IP packets. These hardware traffic generators were capable to send the maximum possible packet rate on each Gigabit Ethernet line without any jitter.

Ethernet/IP packets with Quick-Start requests were generated by a C program on Linux PCs. For the Quick-Start measurements two traffic mixes were used: one with a ratio of 1% and one with a ratio of 50% Quick-Start packets. The requested rate was always set to 10.24 Mbit/s. The generated data rate is constant and pre-defined, i. e., the traffic generators do not use our Quick-Start endpoint implementation [5]. This allowed to test the routers in permanent overload.

We coded our XCP router implementation in a way that all packets are processed as XCP packets independent of their content. By this we could use simple IP packets for the

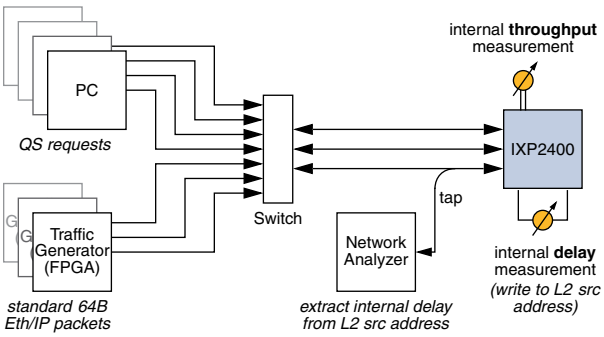

Fig. 4. Measurement setup

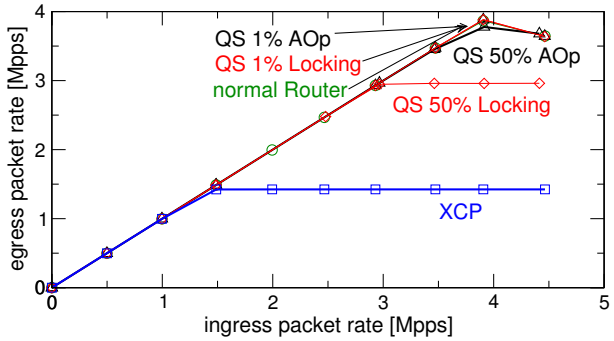

Fig. 5. Router egress packet rate as a function of the load

measurements with the XCP router, as the processing time of the XCP header fields is independent of their numeric values.

As only the packet headers are processed, we generated minimum-sized Ethernet frames of 64 Bytes length. Thus, we achieved the highest possible packet rates of up to $(1\ Gbit/s)/[(64+20)\cdot 8\ bit)] = 1.488$ million packets per second (Mpps) per Gigabit Ethernet line (taking also the 20 Bytes of inter-framing gap and preamble into account).

Our measurement setup is depicted in Fig. 4. All traffic sources were connected to a gigabit switch that switched the traffic to the three Gigabit Ethernet interfaces of our network processor board ENP-2611 from Radisys [15], which contains the Intel IXP2400 network processor running at 600 MHz.

For our evaluation we used the internal throughput measurements that were taken by the considered router applications anyway for deciding about bandwidth requests. So we only needed to output them by the GPP. In order to overcome the 1 Gbit/s limitation of one port, we summed up the throughput of all three ports as one virtual 3 Gbit/s port and synchronized the access to the shared variables, as if they were one single port. Synchronizing each port individually would obviously result in better throughput and delay values.

The delay measurements were also taken internally. At the beginning of the packet processing stage a time-stamp was put to an unused field of the internal meta-data. At the last ME before the packet is put to the output buffer, the difference to the current time was calculated and written to the Ethernet source address of the packet. This did not change the system significantly, as neither the number of memory accesses nor the size of the meta-data was increased. We recorded the outgoing packets of one Ethernet interface with a wiretapped network analyzer and extracted the measured delay values.

C. Throughput Measurement

The throughput of the different router variants in packets per second (pps) is shown in Fig. 5. The egress packet rate is plotted over the ingress packet rate. In the given setup, the unmodified 'normal' router can process packet rates of up to almost 4 Mpps. For higher ingress rates, packets are dropped.

The router enhanced with Quick-Start support has the same performance as the 'normal' router for a traffic mix with 1% Quick-Start requests, which can be assumed to be a realistic scenario. In this case, there is also no difference between the synchronization by locks and by atomic operations. However, when the ratio of Quick-Start requests is increased to 50%, synchronization by locking becomes a bottleneck, and packets are dropped for ingress rates above 3 Mpps. In contrast, the router using synchronization by atomic operations shows almost the same behavior as the unmodified router. A Quick-Start ratio of 50% should never occur in normal usage, but it might be observed e. g. in certain Denial-of-Service attacks. Our experimental measurement results reveal that the router performance is not significantly affected even in such an extreme case if synchronization by atomic operations is used.

Our router supporting XCP, however, shows a significantly lower maximum throughput of only about 1.4 Mpps. Here the access to global variables must be synchronized for 100% of the packets. Furthermore, the critical section is longer than that of the Quick-Start router. This is why the measured maximum throughput is much lower than that of the Quick-Start routers.

D. Delay Impact

The additional processing delay in the routers caused by the new protocol functions is another important performance metric. Table II lists the measured internal delays of the different router variants for a low and a high traffic load scenario. For the Quick-Start routers, both the delay of Quick-Start requests and of normal IP packets is shown.

Apparently, the Quick-Start routers do not impose any additional delay to normal IP packets. Also, the Quick-Start requests do not experience any significant additional latency in both Quick-Start router variants if their relative number is small. This indicates again that Quick-Start support can be added to routers without significant impact on performance.

TABLE II
COMPARISON OF INTERNAL DELAY

Load	33% (1.5 Mpps)	100% (4.5 Mpps)
Standard router	6.6 μs	12.2 μs
QS router – AOp (1% QS) normal packets / QS requests	6.6 μs / 7.0 μs	12.1 μs / 12.3 μs
QS router – Locking (1% QS) normal packets / QS requests	6.6 μs / 7.0 μs	12.2 μs / 12.3 μs
XCP router (XCP packets)	19.4 μs	19.8 μs

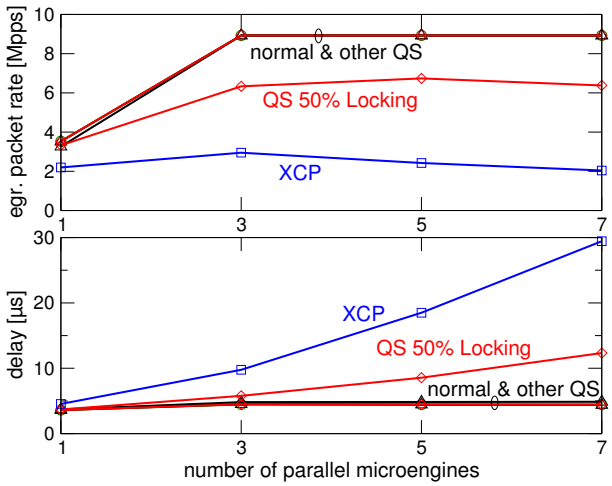

Fig. 6. Egress packet rate and delay for different degrees of parallelism

The XCP router, however, delays packets significantly longer compared to the 'normal' router. Note that the XCP router is already at its capacity limit at 33% load. Here, all packets have to wait about 23 times $((3\,\text{MEs} \cdot 8\,\text{threads}) - 1)$ the duration of the critical section before they can access the shared variables. As a consequence, synchronization becomes a major contributor to the total delay.

E. Increasing Parallelism

In Section III it was shown that using locks for synchronization may decrease the maximum possible packet rate and enlarges the internal delay of the packets. In order to further study this effect we performed simulations using the Intel IXP2800 model. This network processor has basically the same architecture as the IXP2400, but it features 16 MEs. As a result, one can use more parallel MEs to execute the IP processing stage, including the Quick-Start or XCP processing.

Fig. 6 shows both the packet rate and the internal delay as a function of the number of MEs, i. e., the degree of parallelism. 6 Gbit/s line interfaces were used in the simulations. The obtained results cannot be compared directly to our previous measurements, as the IXP2800 works with a higher clock rate.

The results of the Quick-Start router using atomic operations for accessing the shared variables are again almost identical to the results of the 'normal' router. The throughput increases from 1 to 3 MEs from about 3.5 Mpps to the maximum rate of almost 9 Mpps, since 3 or more MEs are required for packet processing at full load. The internal delay is approximately constant at 4 μs, independent of the degree of parallelism.

The Quick-Start router using locking has a comparable performance if only 1% of the packets are Quick-Start requests. With 50% Quick-Start requests the throughput is limited to only 6 Mpps with 3 parallel processors, and it does not improve by further increasing the parallelism. However the internal delay goes up approximately linearly with the number of MEs.

The XCP router shows a similar behavior. The maximum throughput is much less than the theoretical maximum of the architecture. The throughput even slightly decreases for more than 3 parallel processors. Furthermore, the internal delay gets significantly larger with increasing parallelism. This reveals that it might be difficult to implement XCP on router architectures with a high degree of parallel packet processing.

The linear increase of delay and the bounded throughput can be very well explained by the theoretical predictions from Section III-B. This shows that synchronization becomes a performance bottleneck. The more than linear increase in particular of the XCP delay can be caused by further effects, such as contention on internal buses of the network processor.

V. Conclusion

Both Quick-Start and XCP are recent proposals for new congestion control schemes using fine-grained router feedback. While the performance benefit of both schemes has been studied extensively by means of simulation, the actual implementation in routers has hardly been addressed so far. This paper shows that Quick-Start and XCP can be implemented in the fast path of network processor based routers. A key issue is the synchronization of the access to global variables from parallel processing entities. We introduce and compare different synchronization schemes to solve this problem. Our implementations of both Quick-Start and XCP on an Intel IXP2400 network processor show a comparable complexity. The measurements reveal that Quick-Start and XCP processing is feasible at multiple Gbit/s line speed. However, synchronization may limit their performance, in particular for XCP.

References

[1] S. Floyd, M. Allman, A. Jain, and P. Sarolahti, "Quick-Start for TCP and IP," IETF RFC 4782 (experimental), 2007.
[2] A. Falk, Y. Pryadkin, and D. Katabi, "Specification for the Explicit Control Protocol (XCP)," IETF Internet Draft, work in progress, 2007.
[3] P. Sarolahti, M. Allman, and S. Floyd, "Determining an appropriate sending rate over an underutilized network path," *Computer Networks*, vol. 51, no. 7, pp. 1815–1832, May 2007.
[4] D. Katabi, M. Handley, and C. Rohrs, "Congestion control for high bandwidth-delay product networks," in *Proc. ACM SIGCOMM*, 2002.
[5] M. Scharf and H. Strotbek, "Performance evaluation of Quick-Start TCP with a Linux kernel implementation," in *Proc. IFIP Networking 2008, Springer LNCS 4982*, May 2008, pp. 703–714.
[6] Y. Zhang and T. R. Henderson, "An implementation and experimental study of the Explicit Control Protocol (XCP)," in *Proc. IEEE Infocom*, Mar. 2005, pp. 1037–1048.
[7] T. Faber and E. Coe, "Congestion control with explicit feedback," Presentation at IRTF ICCRG meeting, Feb. 2007.
[8] K. Nakauchi and K. Kobayashi, "An explicit router feedback framework for high bandwidth-delay product networks," *Computer Networks*, vol. 51, no. 7, pp. 1833–1846, 2007.
[9] N. Dukkipati, G. Gibb, N. McKeown, and J. Zhu, "Building a RCP (rate control protocol) test network," in *Proc. IEEE Symposium on High-Performance Interconnects (HOTI 2007)*, Aug. 2007, pp. 91–98.
[10] J. Aweya, "On the design of IP routers Part 1: Router architectures," *Journal of Systems Architecture*, vol. 46, no. 6, pp. 483–511, Apr. 2000.
[11] D. Comer, *Network Systems Design Using Network Processors*. Pearson Education, Inc., Upper Saddle River, NJ, USA, 2006.
[12] Intel, *Intel Internet Exchange Architecture Portability Framework Developers Manual, SDK 3.5 Release*, Intel corporation, Nov. 2003.
[13] C. Suriyajan, "Design and implementation of Quick-Start and XCP router functions on a network processor," Master Thesis, University of Stuttgart, IKR, Oct. 2007.
[14] P. Schrem, "Design and realization of a generic traffic generator in VHDL," Student Thesis (in German), University of Stuttgart, IKR, 2006.
[15] Radisys Corporation. (2006) ENP-2611 datasheet. ENP-2611.pdf. [Online]. Available: http://www.radisys.com/products/datasheets/