Computer Communications 32 (2009) 691–702

Contents lists available at ScienceDirect

Computer Communications

journal homepage: www.elsevier.com/locate/comcom

Transparent TCP acceleration

Sameer Ladiwala [a], Ramaswamy Ramaswamy [a], Tilman Wolf [b,*]

[a] Cisco Systems Inc., San Jose, CA, USA
[b] Department of Electrical and Computer Engineering, University of Massachusetts, 151 Holdsworth Way, Knowles 211C, Amherst, MA 01003, USA

ARTICLE INFO

Article history:
Received 26 June 2008
Received in revised form 22 November 2008
Accepted 24 November 2008
Available online 7 December 2008

Keywords:
Network processor
Transmission control protocol
Throughput

ABSTRACT

Transparent transmission control protocol (TCP) acceleration is a technique to increase TCP throughput without requiring any changes in end-system TCP implementations. By intercepting and relaying TCP connections inside the network, long end-to-end feedback control loops can be broken into several smaller control loops. This decrease in feedback delay allows accelerated TCP flows to react more quickly to packet loss and thus achieve higher throughput performance. Such TCP acceleration can be implemented on network processors, which are increasingly deployed in modern router systems. In our paper, we describe the functionality of transparent TCP acceleration in detail. Through simulation experiments, we quantify the benefits of TCP acceleration in a broad range of scenarios including flow-control bound and congestion-control bound connections. We study accelerator performance issues on an implementation based on the Intel IXP2350 network processor. Finally, we discuss a number of practical deployment issues and show that TCP acceleration can lead to higher system-wide utilization of link bandwidth.

1. Introduction

The transmission control protocol (TCP) is a commonly used transport layer protocol in the Internet. It provides reliability, flow control, and congestion control services on top of the lossy, best-effort network layer. There are a number of control mechanisms implemented in TCP that determine when a packet is considered lost, when to retransmit, and when to slow down the rate of transmission. Over the past decades, many components of the TCP protocol have been fine tuned to incrementally increase the overall TCP performance. In all these approaches, the network itself was considered an interconnect that simply moves data from one end-system to another. Only small changes in the network functionality were considered (e.g., RED, FRED, etc.) to support TCP performance.

In our work, we make use of recently developed network processor technology to implement TCP accelerators *inside* the network. We discuss how this approach can significantly improve the performance of TCP connections and how it can be incrementally deployed. The significance and potential impact of this work is considerable since the deployment of such TCP accelerators improves the performance that any network user can experience without changing anything on end-systems.

The idea of a TCP accelerator is to terminate TCP connections on a router and then relay the data to a second connection towards the end system. It is possible to have numerous TCP accelerators in a single end-to-end connection. TCP is a rate-adaptive protocol, and the performance improvement in TCP acceleration comes from reducing the delay in the feedback loop that determines the sending rate and when to retransmit packets. It is important to note that this feedback loop is always active and retransmissions are often necessary due to the TCP behavior of continuously increasing the transmission rate until packet loss occurs. As a result, even under ideal conditions (no packet loss due to link errors and no contention for bandwidth), retransmissions happen. The shorter the feedback loop is to trigger the retransmission, the higher the overall performance of the connection.

In this paper, we present the fundamental concepts of TCP acceleration, a detailed performance study, and a discussion of deployment issues. The specific contributions are:

- *TCP acceleration as a novel approach to improving TCP throughput:* We present the concepts of TCP acceleration on its context to related work. We show a detailed algorithm of how TCP acceleration is implemented on a node.
- *Simulation results of TCP acceleration performance in various networking scenarios:* We show the performance improvements gained from TCP acceleration in different settings. We show that accelerated TCP connections achieve higher bandwidth than conventional TCP connections on these bottleneck links. While some of this bandwidth gain is obtained at the cost of non-accelerated TCP connections, we show that TCP acceleration can lead to a *higher overall utilization* of bottleneck links and thus a more effective network system as a whole.

* Corresponding author. Tel.: +1 413 545 0757; fax: +1 413 545 1993.
E-mail addresses: sladiwal@cisco.com (S. Ladiwala), ramramas@cisco.com (R. Ramaswamy), wolf@ecs.umass.edu (T. Wolf).

0140-3664/$ - see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.comcom.2008.11.036

- *Prototype implementation on a network processor:* A prototype implementation on the Intel IXP2350 network processor is used to demonstrate the feasibility of a high-performance implementation of TCP acceleration.
- *Discussion of deployment issues:* For new network technology, it is imperative that it can be easily integrated with current Internet software and hardware. One major challenge is that in most realistic scenarios the end-system software (e.g., TCP/IP stack) cannot be changed. The TCP acceleration technique that we present here is *completely transparent* to end-systems. All the functionality is implemented on routers. Further, the proposed TCP acceleration technique can be *deployed incrementally*. Even a single router with TCP acceleration can improve the throughput performance of some TCP flows. We show simulation results on an Internet-like topology with realistic traffic that show improvements in system-wide utilization.

Section 2 introduces the related work on which transparent TCP acceleration is based. Section 3 describes the functionality of a TCP acceleration system in detail. The performance impact of TCP acceleration is quantified in Section 4 where we present results from simulations. Details of our prototype implementation on the Intel IXP2350 network processor are presented in Section 5. Section 6 discusses the practical aspects of a potential deployment of TCP acceleration. Finally, Section 7 summarizes and concludes this paper.

2. Related work

There has been a large body of work on improving TCP throughput, which has often focused on developing TCP improvements on the end-system. We are addressing acceleration techniques inside the network.

2.1. TCP acceleration techniques

The performance of an end-to-end transport protocol degrades in long-haul data transmission over lossy links. A recent study on overlay networks by Liu et al. [1] has shown that breaking a long end-to-end TCP connection into multiple shorter TCP "relay" connections can improve efficiency and fairness. The theoretical results from this study are the basis of our work. The main difference is that we are considering a transparent use of TCP relay nodes inside the network. This means that the end-system does not have to be configured to use overlay networks. From a practical point of view, this is an important aspect as changes in end-system software are difficult to deploy.

Other examples of work show how the use of multiple TCP connections can improve throughput. Split TCP connections are used to cope with differences in the communication media that could cause the congestion control window to close. The idea is to split the connection at the boundary in order to isolate performance issues that are related to a particular medium. This was implemented in I-TCP [2,3]. Ananth and Duchamp introduce the idea of implementing a single logical end-to-end connection as a series of cascaded TCP connections [4]. The Snoop Protocol [5] adopts a proxy-based approach to improve TCP performance. The proxy caches packets and performs local retransmissions across a wireless link by monitoring the acknowledgments to TCP packets generated by the receiver.

2.2. The end-to-end argument

It has been argued in the design of the Internet that it is desirable to provide certain functions (e.g., error detection, reliability) in an end-to-end fashion rather than implementing them repeatedly on each hop [6]. In this sense, TCP acceleration violates this end-to-end principle. The reason why we feel this alternate design view is justified is due to the performance improvement that accelerated TCP can provide. This is similar to I-TCP described above and other TCP improvements for wireless environments [7].

2.3. TCP processing on routers

In addition to theoretical work on TCP connections, there are numerous examples where some sort of TCP processing has been implemented inside the network (i.e., on a router). A commonly used technique for building application layer firewalls involves inserting a TCP proxy in the communication path of the two communicating end points. Spatscheck et al. show in [8] that TCP connection splicing improves TCP forwarding performance by a factor of two to four as compared to simple IP forwarding on the same hardware.

Layer 4 switches that provide load balancing and transport layer caching functionality often perform TCP traffic redirection. These are examples where simple TCP-level processing of packets on routers can improve the TCP forwarding performance [9,10].

2.4. TCP processing on network processors

In order to implement a TCP accelerator, it is necessary to augment routers with the ability to store packets and perform TCP-compliant processing of packets. Network processors (NPs) are ideal systems to perform this task. NPs are usually implemented as embedded multiprocessor systems-on-a-chip with considerable amounts of processing power and memory and are software programmable [11]. NPs implement packet processing functions on the input and output ports of routers. Commercial examples of network processors are the Intel IXP2350 [12], the EZchip NP-3 [13], and the LSI APP [14]. Network processors have been used in a number of scenarios to provide advanced packet processing functions ranging from simple forwarding [15] to complex functions like network measurement [16].

In the context of TCP processing, network processors have been used to offload TCP processing from high-end server systems [17,18]. Moving complex TCP processing from end-systems into specialized networking hardware reduces the system load and frees up resources. In our work, the TCP accelerator provides similar functionality as a TCP-offloading system. Instead of providing an interface between the network and a host, the TCP accelerator acts as a connector between two TCP connections inside the network. Packets are buffered and forwarded using a modified TCP state machine. Zhao et al. have implemented TCP processing on a network processor on a router in order to splice two TCP connections for load-balancing purposes [19]. Except for the initial connection setup, the processing complexity of splicing is very simple and can be performed on a per-packet basis. TCP acceleration requires more TCP processing to be performed on the network processor since packets get buffered and potentially retransmitted.

NPs are just the first step to experimenting with TCP acceleration. If TCP acceleration becomes a mainstream functionality of routers (similar to how NAT [20] and firewalling [21] have become a ubiquitous function of home gateways), then ASIC-based implementations are likely to become available and present better power and performance tradeoffs.

3. Transparent TCP acceleration

The functionality of TCP accelerator nodes is implemented on routers that can be added transparently and incrementally to the

network. In this section, we first describe the overall idea of transparent TCP acceleration from the perspective of an end-to-end connection traversing the network. Then, we view TCP acceleration from a router's point of view.

3.1. Network topology

Fig. 1(a) illustrates a conventional TCP connection where only the end-systems participate in Layer 4 processing. The network performs Layer 3 forwarding on datagrams and does not alter any of the Layer 4 segments. Fig. 1(b) illustrates how TCP acceleration nodes (denoted by 'A') change this paradigm. An accelerator node terminates TCP connections and opens a second connection to the next Layer 4 node. This allows the accelerator node to shield the TCP interactions (e.g., packet loss) from one connection to another. As a result, the feedback control loops, which implement the fundamental mechanisms of reliability, flow control, and congestion control, are smaller with lower delay. As a result, accelerated TCP can react faster and achieve higher throughput than conventional TCP.

3.2. Node architecture

Before we quantify the performance improvement from TCP Acceleration in Section 4, we discuss how an accelerator node implements this functionality.

3.2.1. Acceleration example

To illustrate the behavior of an individual TCP accelerator node, Fig. 2 shows a space–time diagram for an example connection over conventional routers and TCP accelerators. For simplicity, unidirectional traffic with 1-byte packets is assumed. The initial sequence number is assumed to be 1. As Fig. 2(a) illustrates in this example, conventional routers just forward segments without interacting on the transport layer. In contrast, the TCP accelerator node in Fig. 2(b) actively participates in the TCP connection (e.g., responds to SYN, DATA, ACK, and FIN segments). By receiving packets and acknowledging them to the sender *before* they have arrived at the receiver, the TCP accelerator effectively splits one TCP connection into two connections with shorter feedback loops. In order to be able to retransmit packets that may get lost after an acknowledgment has been sent to the sender, the accelerator node requires a buffer (shown on the side of Fig. 2). The following example shows the typical behavior of the TCP accelerator:

- *Immediate response to sender:* SYN and DATA packets are immediately buffered and acknowledged. The only exception is the first arrival of DATA 3, where no buffer space is available.
- *Local retransmission:* When packets are lost, they are locally retransmitted (e.g., DATA 3). Due to a shorter RTT for both connections, a shorter timeout can more quickly detect the packet loss.
- *Flow control back pressure:* When the connection from the accelerator node is slower than the one to it, buffer space will fill up and no additional packets can be acknowledged and stored. This will cause the sender to detect packet loss and slow down.

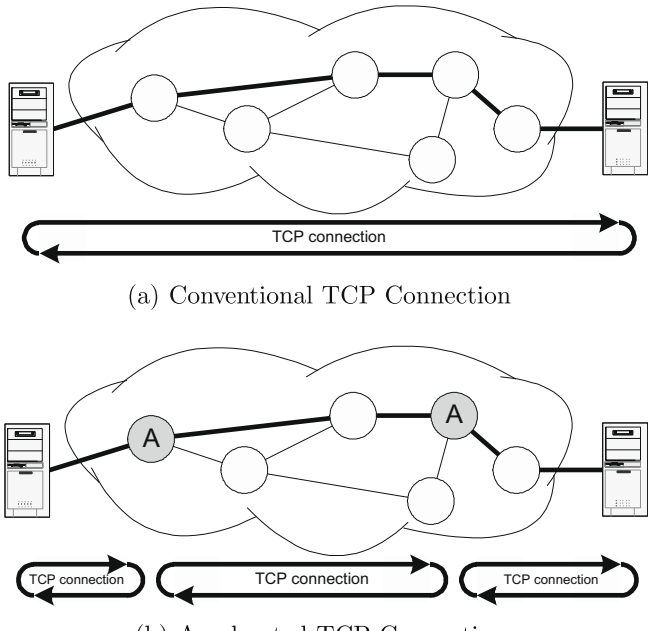


Fig. 1. Conventional and accelerated TCP connections. Systems that implement TCP functionality are marked with 'A'.

The most important observation in Fig. 2 is that the end-systems do not see any difference to a conventional TCP connection (other than packet order and performance).

3.2.2. Acceleration algorithm

The detailed interactions of a TCP accelerator node with a flow of packets from a connection are shown in Algorithm 1. The steps of the algorithm are:

- *Lines 1–2:* A packet is received and classified. The variable p represents the packet and f represents the flow to which the packet belongs.
- *Lines 3 and 35–36:* If a packet is not a TCP packet, it is forwarded without further consideration.
- *Lines 4–10:* If a packet is a SYN packet (indicating connection setup) and no flow record exists, then a flow record is established. A SYN/ACK is returned to the sender and the SYN is forwarded towards the destination ("upstream" and "downstream" respectively). Since the SYN can get lost, a timer needs to be started for that packet. If the SYN/ACK gets lost, the original sender will retransmit the SYN and cause a retransmission of the SYN/ACK.
- *Lines 11–19:* If data are received from the upstream sender and buffer space is available, then the packet is buffered and forwarded downstream. If no buffer space is available, the TCP accelerator needs to propagate back-pressure to slow down the sender. In this case, the packet is not acknowledged and perceived as a packet drop by the sender. The congestion control mechanism of the sender will slow down the sending rate until buffer space becomes available. Packets are only forwarded when the downstream connection does not have too much outstanding data (lines 15–18).
- *Lines 20–26:* If an ACK is received, then buffer space in the complementary flow (flow in the opposite direction, denoted by $\bar{f}$) can be released. This reduces the amount of outstanding data and (potentially several) packets can be transmitted from the buffer space of $\bar{f}$.
- *Lines 27–31:* If a FIN is received, connection teardown can be initiated.
- *Lines 32–33:* If a packet does not match the above criteria, it is handled as an exception.
- *Lines 39–40:* Whenever a timeout occurs, the packet that has initiated the timer is retransmitted.

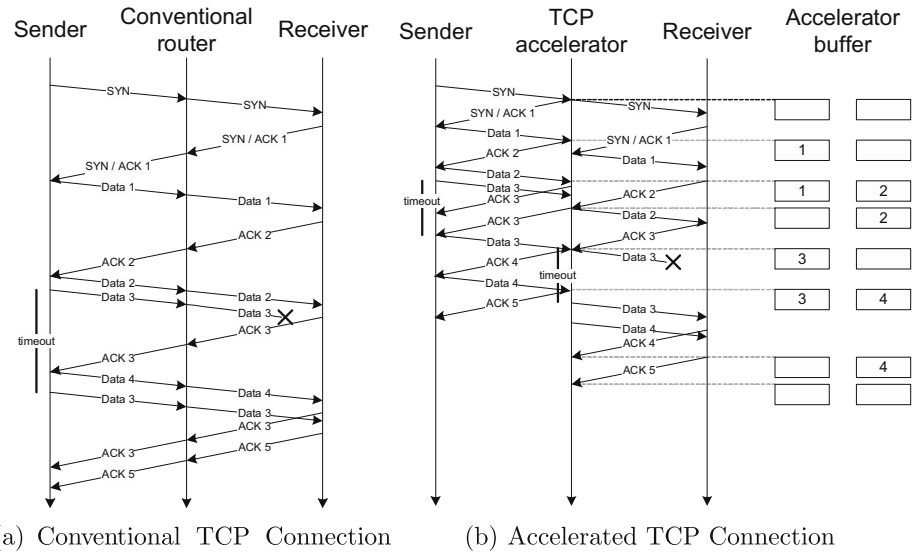


Fig. 2. Message sequence chart of an example connection comparing conventional and accelerated TCP connections.

Algorithm 1. TCP acceleration algorithm

```
1:  p = receive_packet()
2:  f = classify(p)
3:  if (is_TCP(p)) then
4:    if (is_SYN(p)) then
5:      if (!record_exists(f)) then
6:        establish_buffer_space(f)
7:      end if
8:      send(SYN/ACK, upstream)
9:      send(p, downstream)
10:     start_timer(p)
11:   else if ((is_DATA(p))&&(record_exists(f))) then
12:     if (!buffer_full(f)) then
13:       store(p, f)
14:       send(ACK, upstream)
15:       if (outstanding_ACKs(f) + size_of(p) <
          max_window(f) then
16:         send(p, downstream)
17:         start_timer(p)
18:       end if
19:     end if
20:   else if ((is_ACK(p))&&(record_exists(f̄))) then
21:     stop_timer(p̄)
22:     release(p̄, f̄)
23:     while (outstanding_ACKs(f̄)+
        size_of(next_stored_packet(f̄)) <
        max_window(f̄) do
24:       send(next_stored_packet(f̄), f̄)
25:       start_timer(next_stored_packet(f̄))
26:     end while
27:   else if ((is_FIN(p))&&(record_exists(f))) then
28:     send(FIN/ACK, upstream)
29:     send(p, downstream)
30:     start_timer(p)
31:     mark_buffer_for_removal(f)
32:   else
33:     handle_exception(p, f)
34:   end if
35: else
36:   send(p, downstream)
37: end if
38:
39: when (timeout(p))
40:   retransmit(p)
```

The flow control window size that is advertised by an accelerator node is the amount of free buffer space up to half of the total buffer space allocated to the connection (maximum 64 kB). In addition to the state maintenance as described above, a RTT estimator needs to be maintained for each flow according to the TCP specification. The timers for each transmitted packet can be implemented efficiently as described in [22]. Since each connection requires buffer space, it might not be possible to accelerate all connections traversing an accelerator node. In such a case, only a subset of connections is accelerated (not considered in Algorithm 1). This can be performed as part of the packet classification step in Line 2.

3.2.3. NP software components

Fig. 3 shows the architecture of a TCP acceleration node on a network processor. The NP implements two processing paths for packets. Packets that cannot be accelerated due to resource constraints or non-TCP protocols are forwarded without any modification. In order to identify such packets, it is necessary to have a packet classification mechanism (e.g., simple 5-tuple hash function). Packets that are accelerated require Layer 3 and Layer 4 processing, which involves IP input processing, TCP acceleration, and IP output processing. The TCP accelerator has access to a large memory to store TCP state (connection state as well as data buffers). It is important to note that packets which are processed in the TCP accelerator are not addressed to the router system that performs the acceleration. Instead, the router transparently intercepts these packets and performs the acceleration. The end systems are also unaware of this processing that is performed by the router.

3.2.4. Processing and memory resources

TCP processing requires additional computational and memory resources as compared to plain IP forwarding. The processing consists of IP input and output processing as well as TCP processing. The total processing requirements in terms of the number of processing cycles are presented in Section 5. The memory requirements are determined by the size of the TCP connection state (tens of bytes) and the TCP buffer size (tens of kilobytes). The buffer requirements for a TCP accelerator are determined by the maximum window size that is allowed on a connection. The accelerator needs to reliably buffer all packets that have not been acknowledged by the receiver plus all packets that can possible be sent by the sender. Thus, the ideal buffer size is two times the maximum window size of the connection.

Typical network processor systems are equipped with multiple processor cores (4–16) and large SRAM and DRAM memories (1–8 MB SRAM, 64–256 MB DRAM). Assuming a maximum window size of 64 kB, a buffer of 128 kB needs to be allocated for maximum efficiency. With this configuration a total of 1000 connections can be supported by 128 MB of memory. This is sufficient to support most TCP connections on low-speed edge routers.

3.3. Cross-layer protocol issues

One observation on TCP Acceleration is that it slightly changes the "delivery guarantees" of TCP. In a conventional TCP connection, receiving an ACK at the sender implies that the destination has received the acknowledged data. In the case of TCP Acceleration, this is no longer true. Instead, an ACK signifies that the next accelerator has received the acknowledged data and will try to relay it to the destination (or the next accelerator node). It may be conceivable that an accelerator node that has acknowledged a packet may go down before passing the packet to the next node. In that case, there would be a discrepancy between what the sender thinks the receiver has received and what the receiver really has received.

However, in most operating systems it is not possible (or at least not easily) to determine is data that has been placed in a TCP socket has been acknowledged. Instead, network applications use application-layer protocols to determine correct delivery of messages. Thus, this change in ACK semantics has very little practical impact.

4. TCP performance of acceleration

The key question about the performance of TCP acceleration is how much benefit can be gained for the cost of implementing an almost complete TCP/IP stack on a network processor. The theoretical foundation of the performance of accelerated TCP can be illustrated with the equation that has been derived for TCP throughput performance (i.e., bandwidth *bw*) [23]:

$$bw_{TCP} = 1.22\frac{MTU}{RTT \cdot \sqrt{loss}} \quad (1)$$

Assuming a constant maximum transfer unit (*MTU*), a larger round-trip time (*RTT*) and a higher link loss rate (*loss*) causes the throughput performance to degrade. It is clear that with accelerated TCP, we can achieve lower round-trip times due to the separation of feedback loops. The reduced RTT, however, also increases the number of packets sent on a link. If there is contention on this link, then the loss rate increases. Therefore it is not easily possible to derive an analytic result for the overall performance of accelerated TCP. Therefore, we present a number of simulation experiments that help in obtaining a quantitative understanding of the performance impacts. First, we explore TCP connections that are bound by the flow-control functionality in TCP. Then, we explore the more complex case of congestion-control bound TCP connections. These experiments were carried out on the ns-2 [24] simulator.

The *FullTcp* agent (with Reno congestion control) was modified to create a new agent that is aware of the TCP acceleration mechanism. Additionally, a new application module was developed to pull data from the TCP receive buffer and transfer it to the next accelerator node or the end system. This application is used to simulate the functionality of the TCP accelerator node.

4.1. Flow control

In the following subsections, we explore different aspects of TCP acceleration by varying accelerator node placement, varying the amounts of data transferred, and constraining available processing resources. The results shown here are a subset of a wider range of results which were presented in our previous work [25]. In each case, we compare the speedup offered by TCP acceleration to that of regular TCP:

$$speedup = \frac{bw_{accelerated\ TCP}}{bw_{regular\ TCP}} \quad (2)$$

By "regular TCP" or "conventional TCP" we mean a standard ns-2 *FullTcp* and *TCPReno* agent on the same topology. The total end-to-end delay for all experiments is 100 ms (unless stated otherwise). Unless explicitly specified, link bandwidths are 100 Mbps, the RED queuing discipline is used, and processing delay is negligible. The FTP application is used to provide data transfer. In all cases, there is no background traffic that competes with the data transfer.

4.1.1. Single relay node

First, we explore the speedup offered over regular TCP for different placements of a single accelerator node over a link. The placement of the node is varied by changing the distance (in terms of delay) of the accelerator node from the source. The results are shown in Fig. 4 and show that evenly spaced accelerator nodes are best as they cut the link delay for both sides into half. Unevenly

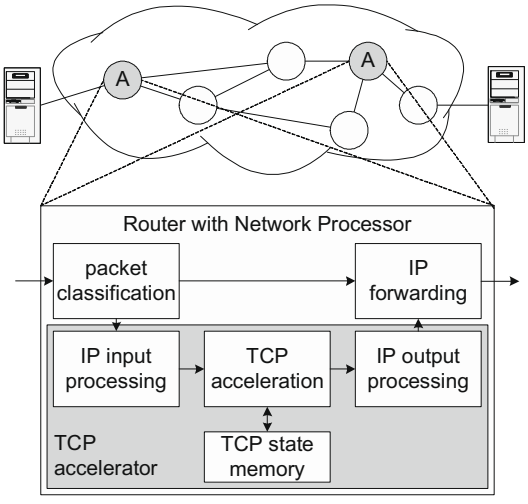


Fig. 3. System architecture of TCP acceleration node.

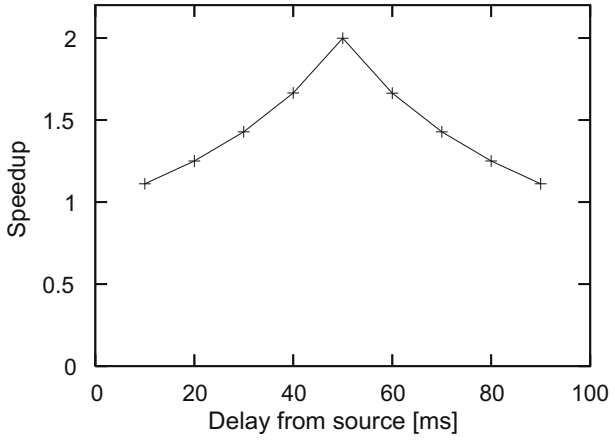


Fig. 4. Speedup of accelerated TCP over regular TCP depending of location of node. The *x*-axis shows the distance of the accelerator from the source with a total end-to-end delay of 100 ms.

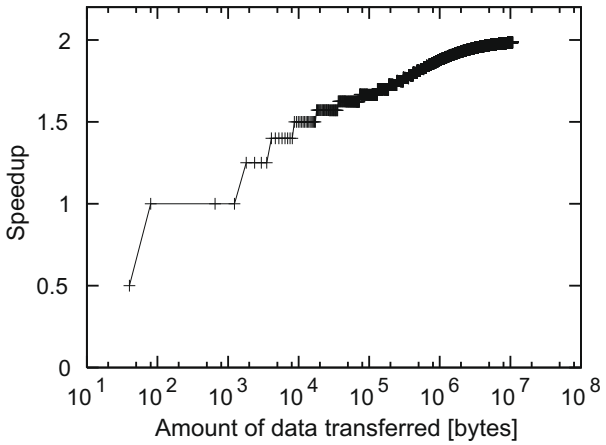


Fig. 5. Speedup of accelerated TCP over regular TCP depending on connection duration. The *x*-axis shows the amount of data that is transferred.

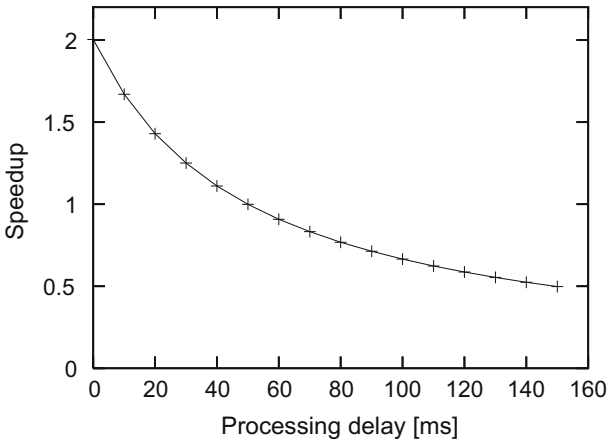


Fig. 7. Speedup of accelerated TCP over regular TCP depending on processing delay of acceleration node. The *x*-axis shows the amount of delay introduced due to acceleration processing.

split connections can still benefit from acceleration as the speedup is always greater than 1.

4.1.2. Connection duration

One question that is particularly important in the context of web transfers is the connection duration. Many web documents consist of small objects and TCP connections to download them are often very short. We obtain Fig. 5 by exploring the connection duration with a single accelerator node in the middle of the topology. There is some overhead in accelerated TCP only for the initial ACK. Transfers of a few kilobytes already achieve an overall speedup of 1.5. The maximum speedup for a single accelerator node is 2 and large file transfers converge to this value.

4.1.3. Multiple acceleration nodes

The maximum speedup with a single acceleration node is 2. The question arises as to how much speedup multiple acceleration nodes can achieve. The results in Fig. 6 assume evenly spaced acceleration nodes totaling an end-to-end delay of 100 ms. The maximum speedup increases linearly with the number of nodes. This is due to the reduced round-trip time for each connection, which allows faster acknowledgment of sent data and transmission of the next window. It should be noted that in this case the connection throughput is flow-control-limited, not congestion-control-limited. In the latter case, the speedup would be limited by the available bandwidth.

4.1.4. TCP processing delay

The previous experiments assume that the processing delay for TCP acceleration is negligible. As we have discussed in Section 3.2.4, TCP acceleration can incur a significant processing cost and it is important to see what impact this has on the achievable speedup. The results for a single accelerator node with varying processing delay are shown in Fig. 7. Even with a processing delay of 40 ms (which is very large, even for complex packet processing functionality), accelerated TCP still performs better than regular TCP. Processing delays in the lower millisecond range have hardly any impact on the overall performance.

4.1.5. Acceleration with limited memory

In addition to processing power, TCP acceleration also requires a significant amount of buffer space to be maintained per connection. In Fig. 8, we vary the maximum window size on the accelerator node and plot the speedup obtained over regular TCP. The window size is half the amount of data that must be stored on the node in the worst case. The accelerator performs best if the window size is at least as large as the maximum window size used by the sender and receiver. Smaller window sizes linearly decrease

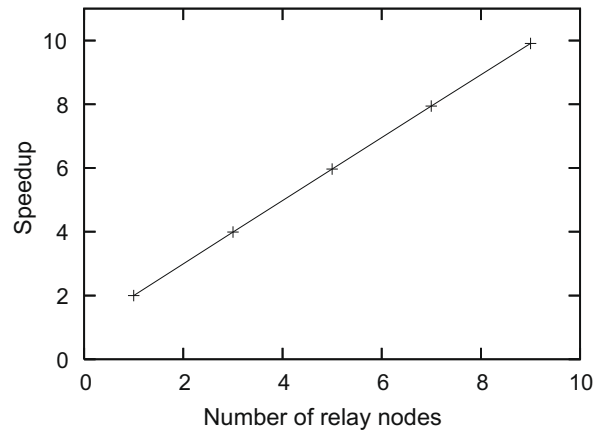


Fig. 6. Speedup of accelerated TCP over regular TCP depending on number of accelerators. The *x*-axis shows the number of evenly spaced acceleration nodes with a total end-to-end delay of 100 ms.

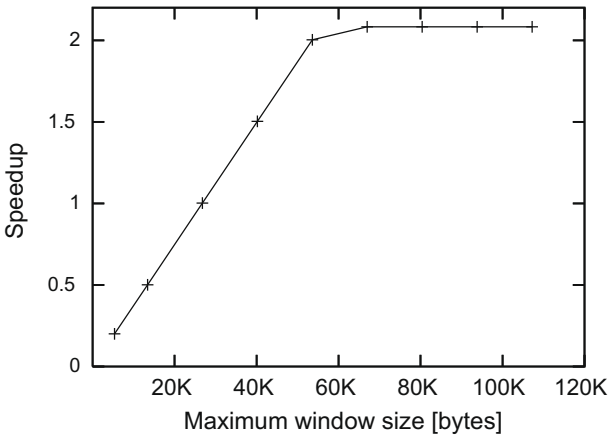


Fig. 8. Speedup of accelerated TCP over regular TCP depending on maximum window size. The *x*-axis shows the amount of data that can be stored on an acceleration node.

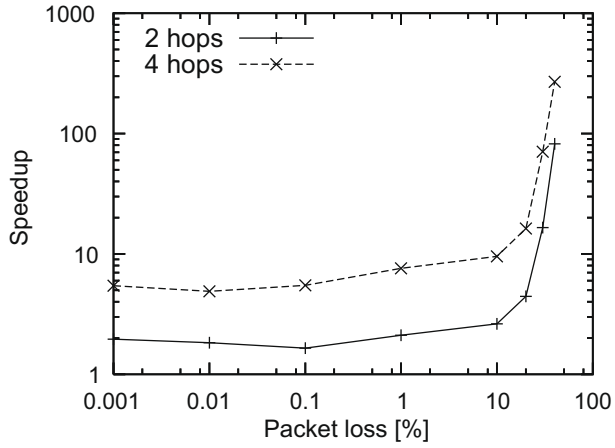


Fig. 9. Speedup of accelerated TCP over regular TCP depending on packet loss rate.

the throughput. Larger window sizes do not improve performance, because the sender cannot utilize the larger window.

4.1.6. Acceleration of lossy links

One scenario where accelerated TCP significantly outperforms regular TCP is on lossy links (e.g., wireless links). The throughput is higher because local retransmissions from the accelerator node can repair the loss locally and do not require end-to-end retransmissions. Fig. 9 shows the speedup of a single accelerator node ("2 hops") and three accelerator nodes ("4 hops") over regular TCP for different packet loss rates. For loss rates around 1–10%, the speedup on a multihop topology is close to 10-fold. For higher loss rates (which are unlikely to be encountered in a realistic network) the speedup is even higher. This is due to conventional TCP performing extremely poorly under these conditions. Thus, the relative benefit of accelerated TCP is higher.

4.2. Congestion control

The previous results show that TCP acceleration can significantly increase the throughput for flow-control limited connections. This case is applicable when there is no congestion in the network or when the connection duration is so short that the TCP window has reached the size where congestion becomes an issue.

The following set of results show that there are also benefits to using TCP acceleration for connections whose throughput is limited by congestion control. In this context, we also explore the issue of fairness between accelerated and non-accelerated TCP flows.

4.2.1. Throughput under congestion

First, we explore how accelerated TCP connections behave in comparison to conventional TCP flows. The simulation results shown in Fig. 11 were derived from an ns-2 simulation with an extremely simple topology shown in Fig. 10. The nodes adjacent to the bottleneck link provide acceleration services to some of the connections. Unless noted otherwise, 10 connections start at time $t = 0$ at the left-most node and connect to the rightmost node.

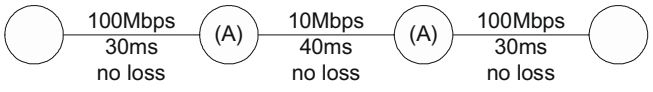
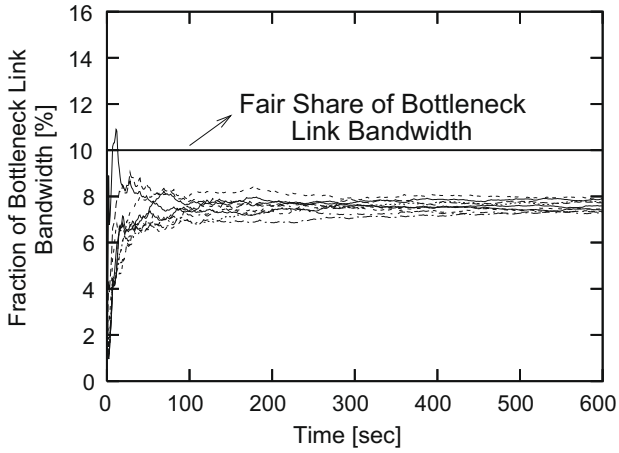
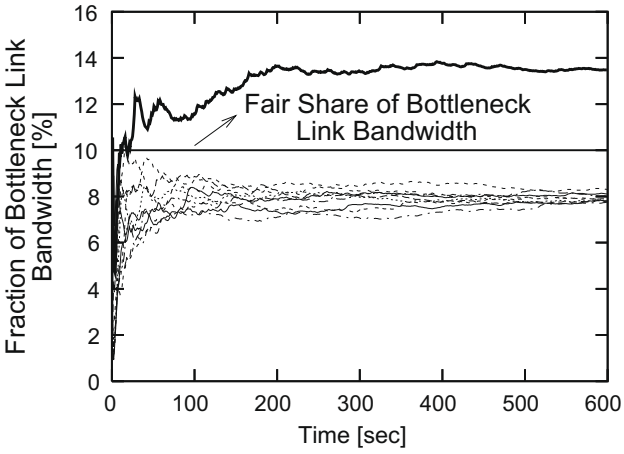


Fig. 10. Setup of ns-2 simulation for experiments with TCP congestion. Nodes denoted with "(A)" provide TCP acceleration.

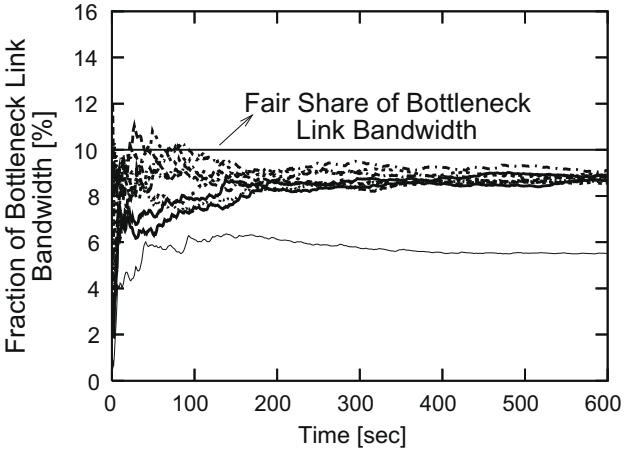
Fig. 11 shows two experiments – one with 9 conventional and 1 accelerated TCP flow, and one with 1 conventional and 9 acceler-



(a) 10 Conventional TCP Flows



(b) 9 Conventional and 1 Accelerated TCP Flow



(c) 1 Conventional and 9 Accelerated TCP Flows

Fig. 11. Throughput of accelerated TCP flows (bold) and conventional TCP flows (thin).

ated TCP flows. In Fig. 11(a), one can observe that conventional TCP connections reach steady state at an average bandwidth of just below the fair share of the link speed. The overhead for protocol headers does not allow a full use of the entire bandwidth for "useful" data. Even between identical connections (as in this experiment), there is a certain level of "unfairness". The average bandwidth that is achieved by each connection differs by around 10% even after $t = 600$ s. Nevertheless, all connections show roughly the same behavior. In Fig. 11(a), one of the conventional TCP flows is replaced with an accelerated TCP flow, where the nodes before and after the bottleneck link implement TCP acceleration. The average throughput for the accelerated flow (shown in bold) increases by approximately 60% over non-accelerated flows. This is expected as we have decreased the TCP control loop on the congested link from an RTT of 200 ms to an RTT of 80 ms. Looking at Eq. (1), we expect the performance to increase by a factor of $\frac{200\text{ ms}}{80\text{ ms}} = 2.5$ due to the decrease of the RTT. However, this improvement in RTT is counterbalanced by an increase in the loss rate due to more "pressure" on the bottleneck link. The accelerated flow increases the number of packets in the queue and thus increases the loss rate. The combination of lower RTT and higher loss rate yields a performance improvement of a factor of 1.6.

When further increasing the number of accelerated flows, we observe the behavior shown in Fig. 11(b). The performance improvement for each accelerated flow diminishes as more flows have the same "competitive advantage" and is only around a factor of 1.1. The conventional TCP flow observes a significant reduction in performance to about 0.7 times the original bandwidth. This raises the immediate question of fairness of TCP acceleration.

4.2.2. Connection startup

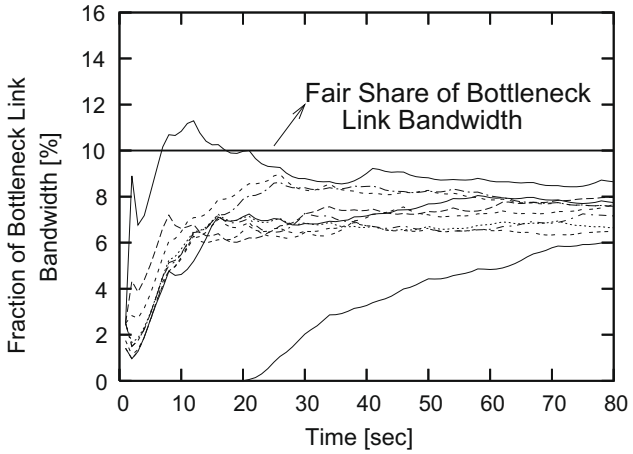
Fig. 12 shows the startup behavior of nine flows starting at time $t = 0$ and one additional flow starting at time $t = 20$ s. In Fig. 12(a), all connections are conventional TCP and it can be observed that the late flow requires a longer time to reach fair link sharing than the other flows. In Fig. 12(b), the late flow is an accelerated TCP flow that reaches the fair share of the link much more quickly than a conventional TCP flow. It eventually does reach a higher bandwidth than the fair share as shown in Fig. 11. Finally, Fig. 12(c) shows a late regular TCP flow competing with accelerated flows. In this case, the late flow requires more time to gain the same throughput as in Fig. 12(a). Nevertheless, the accelerated flows do allow the conventional TCP connections to behave roughly the same.

4.2.3. Fairness

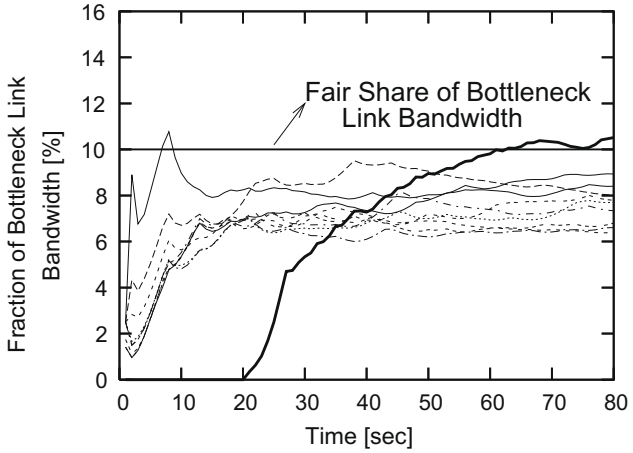
From Fig. 11(b), we see that TCP acceleration takes away bandwidth from conventional TCP flows and thus raises the question of fairness. The issue of TCP-friendly bandwidth consumption has been studied in the context of rate-controlled flows [26]. For TCP acceleration, two points of view can be taken:

- *TCP acceleration is unfair.* Clearly, as shown in Fig. 11(b), an accelerated TCP flow receives more bandwidth on the bottleneck link than a conventional TCP flow. Since in both cases the end-to-end connections have the same RTT and loss rate, both types of connections should receive the same bandwidth.
- *TCP acceleration is fair.* Fairness is only achieved when both connections have the same RTT and the resulting loss rate. Thus, in most practical cases, absolute fair sharing of the bottleneck link is not practically possible. In principle, TCP acceleration only changes the RTT of an accelerated flow. All other aspects of TCP are the same as on conventional TCP.

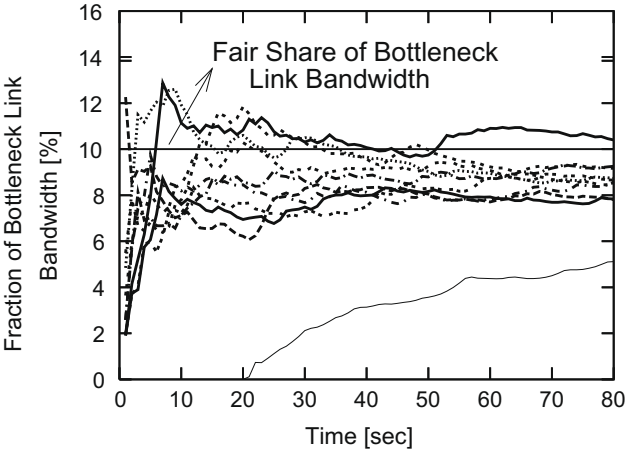
We do not attempt to judge which view is the right one. Since there is nothing fundamentally different between accelerated TCP and conventional TCP, the use of TCP accelerators can probably not be detected and discouraged easily. Therefore it can be expected that such an acceleration service will become more widely used and a necessary feature to remain competitive in the market.



(a) 10 Conventional TCP Flows



(b) 9 Conventional and 1 Accelerated TCP Flow



(c) 1 Conventional and 9 Accelerated TCP Flows

Fig. 12. Throughput of accelerated TCP flows (bold) and conventional TCP flows (thin) with delayed start.

4.2.4. Link utilization

The throughput improvements shown above address the performance seen by one particular user. Another question about the benefit of TCP acceleration is how the use of this service impacts the overall system. To explore this question, we look at the utilization of the bottleneck link. If the utilization increases with the use of more accelerated TCP connections, then we can infer that there is not only an individual benefit but also a global benefit.

In our experiments, 10 flows of conventional TCP achieve a link utilization of 76.4%. The cause for the utilization being less than 100% lies in the oscillating behavior of TCP's congestion control mechanisms. Connection back-offs in AIMD congestion control can cause buffer underflows that lead to underutilized links. The longer the RTT, the more severe these oscillations are. With even a single accelerated TCP flow (in addition to nine conventional TCP flows), we observe a link utilization of 85.6%, which illustrates the more efficient behavior of TCP connections with shorter RTTs. Additional accelerated TCP flows do not further increase the utilization.

4.3. Summary of simulation results

From the above results, we can extract a few key observations:

- Flow-control limited connections can achieve significant throughput improvements when using TCP acceleration (Figs. 4 and 6).
- Even a processing delay in the order of 10 ms (10% of the RTT) can still yield a speedup of 1.5× (Fig. 7).
- Links with high loss rates benefit extremely from TCP acceleration (Fig. 9).
- TCP acceleration provides competitive advantage to connections competing for bottleneck link bandwidth (Fig. 11).
- The utilization of bottleneck links increases with the fraction of accelerated TCP connections leading to better system-wide performance.

These results show that there are a number of benefits associated with the use of transparent TCP acceleration.

5. System implementation of TCP accelerator

To illustrate the feasibility of high-performance TCP acceleration, we have implemented a prototype system on an Intel IXP2350 network processor.

5.1. IXP2350 prototype

The design of the prototype implementation is shown in Fig. 13. Packets that need to be accelerated are processed by the TCP Processing component. This component accesses the connection state and connection buffer to determine the appropriate action (as described in Algorithm 1). The processing component can be replicated across threads on the same microengine and across multiple microengines to improve system utilization. However, our current prototype uses a single instance of TCP processing, which allows us to analyze the performance of the system without having to consider interactions across multiple instances. The prototype uses approximately 128 kB of memory per connection to buffer packets. With typical SRAM memories and the use of larger DRAM, TCP acceleration could be scaled to support thousands of flows.

5.2. Implementation results

We have evaluated the implementation of our TCP accelerator using the cycle-accurate simulator provided by Intel. The results are shown in Table 1. The processing cost for each component of the algorithm is reported in processor cycles on the 900 MHz microengines and in microseconds. The ranges for data transmission times are a result of different packet sizes. The minimum and maximum reported correspond to 64-byte and 1516-byte packets.

The results show that the delay caused by TCP acceleration processing on the IXP2350 is in the order of tens of microseconds. This is about an order of magnitude more than Layer 3 forwarding, but several orders of magnitude less than what would degrade acceleration speedup as shown in Fig. 7. This illustrates that network processors are ideal for supporting a large number of accelerated TCP flows inside the network.

6. Practical deployment

The main argument for transparency in TCP acceleration is that it allows for incremental deployment. As we have mentioned before, not needing to change the TCP/IP protocol stack in end-systems is a key aspect of implementing acceleration functions in routers. This independence from end-systems also gives hope that TCP acceleration can have practical impact in today's Internet infrastructure. It could be conceivable that internet service providers (ISPs) provide acceleration technology initially to their premium customers (which might not exceed the maximum number of accelerated connections on any given router). Also, it is possible to make acceleration available as a best-effort feature, where routers accelerate a TCP connection if buffer memory is available at connection time. This issue of practical deployment is further explored in this section.

Table 1
Processing cost for TCP accelerator on the Intel IXP2350.

Processing	Cycles	Time (μs)	Algorithm
SYN/SYN + ACK	473	0.52	Lines 4–10
DATA receive	1876–33,948	2.08–37.72	Lines 11–19
ACK transmit	772	0.86	
ACK receive	247	0.28	Lines 20–26
DATA transmit	2322–32,726	2.58–36.36	
FIN/FIN + ACK	1014	1.13	Lines 27–31
Timeout	2311–33,080	2.57–36.75	Lines 39–40

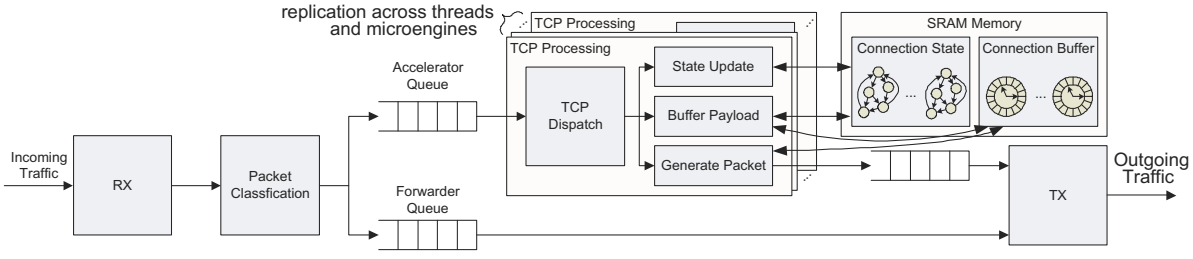


Fig. 13. Implementation of TCP acceleration on IXP2350 network processor.

6.1. Accelerator placement

There is one key constraint on the placement of a TCP accelerator. All TCP packets of an accelerated connection need to pass through the accelerator node for it to work properly. Due to the transparency of the accelerator, the end-system is not aware of the placement of the node and thus cannot "force" traffic on a certain route. It is therefore possible that packets take different routes and thus cause incorrect behavior.

Routing changes do happen in networks, but there are topological aspects that can guarantee proper operation if the TCP accelerator is placed appropriately. Many end-systems and subnetworks have only a single access link to the Internet (e.g., hosts connected through access routers or stub-domains). In such a case, there are no options for routing alternatives and traversal of that router is guaranteed. These routers are particularly suitable for TCP acceleration as they match the performance characteristics that can be achieved by an implementation on network processors.

6.2. Accelerated edge router

Fig. 4 shows that TCP acceleration provides the most benefit when the accelerator is placed in the middle of the connection. One drawback of edge routers is that they are very close to the end-system in terms of connection delay. Therefore it is often only possible to obtain minor gains by using TCP acceleration.

We have performed a measurement experiment to quantify the delay in TCP connections on the router that connects our campus to the Internet. To obtain the data, we have used an Endace GIGEMON system which contains a DAG 4.3GE [27] card. The results are shown in Fig. 14. The x-axis shows the delay on the LAN side of a connection and the y-axis shows the delay on the WAN side of a connection. In general, we can see that the WAN delay – as expected – is higher than the LAN delay for most connections. The diagonal lines indicate the 50%/50% boundary of LAN/WAN delay, the 10%/90% boundary, and the 1%/99% boundary. While there are connections that are approximately balanced in terms of round-trip time on both sides, the majority has a very short RTT on one side and a long RTT on the other side. Thus, potential performance improvements are somewhat limited if TCP acceleration is restricted to edge routers only.

This indicates that it would be desirable to push TCP acceleration deeper into the network core where delays are balanced and acceleration benefits are higher. There are several practical constraints that currently limit such a deployment. First of all, it is not uncommon to use asymmetric routes in the core of the Internet. TCP acceleration in its current form cannot be used in the presence of asymmetric routes because both directions of traffic need to be visible to the system. Further, network processor technology can only provide very simple packet processing functions at data rates of 10 Gbps and higher. Implementing the TCP acceleration function as described in our paper would be challenging at best.

Another approach to addressing the problem of limiting TCP acceleration to the network edge is to identify the flows where the placement of the edge router is balanced in RTT terms. In Fig. 14, we can see a considerable number of flows in the upper right area that are prime candidates. These show roughly a 50%/50% split in RTT and higher overall RTT. It is a question for future research as to how to identify such flows.

6.3. Realistic topology

When deploying a feature incrementally, it is important to understand when a critical mass is reached where benefits are clearly observable. This question requires the exploration of TCP acceleration on an entire network topology. We have used a simulation setup with 200 nodes and a transit stub topology with eight transit domains nodes [28]. We simulate 400 flows over a duration of 30 s. Flow sources and destinations are randomly chosen from the stub domains so that almost all (>95%) of the flows are forced to pass through at least one transit domain node. TCP acceleration is implemented for a varying number of flows on the transit domain nodes. Roughly 50% of the transit domain edges are congested. We measure the overall throughput of the entire network to judge how efficient a particular configuration operates.

Fig. 15 shows the performance increase of a network with TCP acceleration over a network with just conventional TCP. This is expressed as a speedup compared to the baseline case of 100% conventional TCP flows and 0% accelerated TCP flows. With 70% of connections using TCP acceleration, we can achieve a 1.5× speedup and with 90% of connections using TCP acceleration, we can obtain a 2× speedup. It is important to note that this does not mean that we need 70% or 90% of all routers to support TCP acceleration. It is sufficient that a connection encounters one or a few TCP accelera-

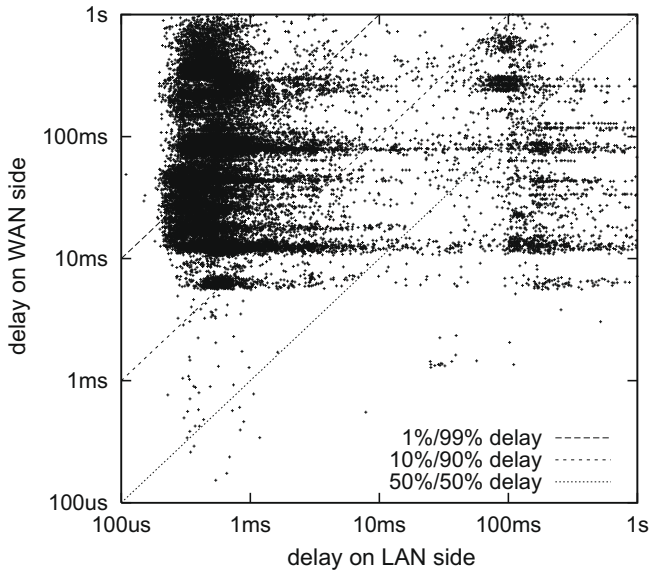


Fig. 14. Round-trip time measurement of TCP connections on UMass Internet access link. The graph separates the portion of the RTT on the LAN side of the edge router from the RTT portion on the WAN side.

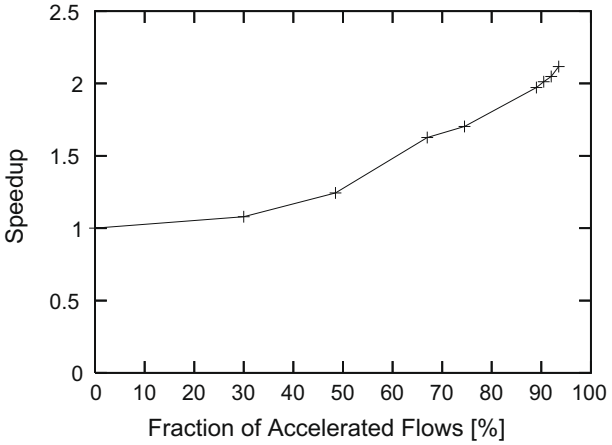


Fig. 15. Simulation results on stub domain topology.

tion nodes along its path. Thus, the required number of routers with the TCP acceleration service is much lower.

6.4. Practical deployment issues

When discussing a large deployment of TCP acceleration nodes, there are some important practical issues that need to be considered:

- TCP acceleration shifts some of the "control of the network stack" from operating system vendors to network equipment vendors. Since TCP congestion control algorithms have been limited to end-systems, their implementation has been the purview of operating system developers. Thus, the control of what functions to put into the transport layer of the protocol stack has been with them. With deployment of TCP acceleration, transport layer functionality migrates into network equipment, where its development is controlled by network equipment vendors.
- Related to the above point is the issue of how deployment of new congestion control algorithms can be achieved. With more parties being involved, this step seems to become more complex. However, TCP acceleration nodes simplify the process because changes in a few devices can impact a large fraction of traffic. Traditionally, non-transparent changes to TCP congestion control required changes in all deployed operating systems. With TCP acceleration, it is possible to simply change the congestion control algorithm that is used between acceleration nodes. Congestion control involving end-systems remains unchanged. Thus, a connection may use traditional congestion control for the first and last hop, but advanced congestion control on all hops in between.
- The use of TCP acceleration makes research on new congestion control algorithms more difficult. Since a connection may involve one or more TCP accelerators, their operation may interfere with experimental validation of new congestion control algorithms. To avoid such interference, traffic would need to be encapsulated in UDP (just for the purpose of experimentation). Once a new algorithm is developed, it can be deployed as described above.

These issues do not present insurmountable obstacles, but need to be considered in the case of a real deployment.

6.5. Technical limitations

While the overall results for TCP acceleration paint a positive picture on the potential impact of such a service, there are a number of technical limitations in the current approach:

- The requirement for both directions of traffic to traverse the same node for acceleration is a limitation in terms of where the system can be deployed. As we have shown above, there are flows that can benefit from acceleration on the network edge, but it would be desirable to push TCP acceleration towards the core to gain more throughput improvements for a larger fraction of the traffic.
- The performance of network processors plays a role in the delay that is introduced by TCP acceleration. Only connections where this delay is relatively short compared to the overall end-to-end delay should be accelerated. It is challenging to identify these connections among all other connections.
- It is possible that the accelerator acknowledges a packet that later cannot be delivered to the receiver. Most Internet protocols use application-layer semantics that can detect and recover from this condition.
- The acceleration only works on packets where the TCP header is available in cleartext. IPSec connections encrypt all packet headers and payloads beyond layer 3. Therefore TCP acceleration is not possible on such connections. The acceleration of SSL on the other hand is possible, because TCP headers are visible.
- TCP options are limited to those that are supported by the accelerator. This may restrict the features of the TCP connection that the sender and receiver can use.

Despite these open issues, we feel that the performance results that we present in this paper show an exciting potential for improving TCP performance transparently and deploying such accelerators incrementally in the current Internet.

7. Summary

In this work, we have introduced transparent TCP acceleration techniques that can speedup TCP connections without end-system support. We have discussed how such an acceleration system operates and how it can be implemented. Simulation results show that TCP acceleration can provide significant performance improvements for flow-control limited connections. Even for connections that are traversing congested links, a user can experience better performance and link utilization can be increased. We have presented an implementation of the accelerator on the Intel IXP2350 network processor and have used it to study accelerator performance. A discussion of deployment issues emphasizes the potential for practical impact in today's Internet.

Acknowledgments

This research was supported in part by a gift from Intel Research. This work was done while Sameer Ladiwala and Ramaswamy Ramaswamy were with the University of Massachusetts Amherst. The authors thank Yong Liu and Yu Gu for providing the ns-2 simulation code used in [1].

References

[1] Y. Liu, Y. Gu, H. Zhang, W. Gong, D. Towsley, Application level relay for high-bandwidth data transport, in: Proceedings of the First Workshop on Networks for Grid Applications (GridNets), San Jose, CA, 2004.
[2] A. Bakre, B. Badrinath, I-TCP: indirect TCP for mobile hosts, in: Proceedings of the of 15th International Conference on Distributed Computing Systems (ICDCS), 1995, pp. 136–143.
[3] A. Bakre, B. Badrinath, Implementation and performance evaluation of indirect TCP, IEEE Transactions on Computers (1997) 260–278.
[4] A.I. Sundararaj, D. Duchamp, Analytical characterization of the throughput of a split TCP connection, Technical Report, Department of Computer Science, Stevens Institute of Technology, 2003.
[5] H. Balakrishnan, S. Seshan, E. Amir, R.H. Katz, Improving TCP/IP performance over wireless networks, in: MobiCom'95: Proceedings of the First Annual International Conference on Mobile Computing and Networking, Berkeley, CA, 1995, pp. 2–11.
[6] D.D. Clark, The design philosophy of the DARPA Internet protocols, in: Proceedings of the ACM SIGCOMM 88, Stanford, CA, 1988, pp. 106–114.
[7] Y. Tian, K. Xu, N. Ansari, TCP in wireless environments: problems and solutions, IEEE Communications Magazine 43 (3) (2005) S27–S32.
[8] O. Spatscheck, J.S. Hansen, J.H. Hartman, L.L. Peterson, Optimizing TCP forwarder performance, IEEE/ACM Transactions on Networking (2000) 146–157.
[9] A. Cohen, S. Rangarajan, H. Slye, On the performance of TCP splicing for URL-aware redirection, in: USENIX Symposium on Internet Technologies and Systems, 1999.
[10] G. Apostolopoulos, D. Aubespin, V. Peris, P. Pradhan, D. Saha, Design, implementation and performance of a content-based switch, in: Proceedings of the IEEE INFOCOM 2000, Tel Aviv, Israel, 2000, pp. 1117–1126.
[11] T. Wolf, Challenges and applications for network-processor-based programmable routers, in: Proceedings of the IEEE Sarnoff Symposium, Princeton, NJ, 2006.
[12] Intel Corporation, Intel Second Generation Network Processor, 2005. Available from: <http://www.intel.com/design/network/products/npfamily/>.
[13] EZchip Technologies Ltd., Yokneam, Israel, NP-3 – 30-Gigabit Network Processor with Integrated Traffic Management, May 2007. Available from: <http://www.ezchip.com/>.

[14] LSI Corporation, APP3300, Family of Advanced Communication Processors, August 2007. Available from: <http://www.lsi.com/>.
[15] T. Spalink, S. Karlin, L. Peterson, Y. Gottlieb, Building a robust software-based router using network processors, in: Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP), Banff, AB, 2001, pp. 216–229.
[16] R. Ramaswamy, N. Weng, T. Wolf, A network processor based passive measurement node, in: Proceedings of the Passive and Active Measurement Workshop (PAM), Boston, MA, 2005, pp. 337–340 (extended abstract).
[17] Hewlett–Packard Company, Maximizing HP StorageWorks NAS Performance and Efficiency with TCP/IP offload engine (TOE) Accelerated Adapters, March 2003. Available from: <http://www.alacritech.com>.
[18] Teja Technologies, TejaNP Datasheet, 2003. Available from: <http://www.teja.com>.
[19] L. Zhao, Y. Luo, L. Bhuyan, R. Iyer, Design and implementation of a content-aware switch using a network processor, in: Proceedings of the 13th International Symposium on High Performance Interconnects (Hot-I05), Stanford, CA, 2005.
[20] K.B. Egevang, P. Francis, The IP network address translator (NAT), RFC 1631, Network Working Group, May 1994.
[21] J.C. Mogul, Simple and flexible datagram access controls for UNIX-based gateways, in: USENIX Conference Proceedings, Baltimore, MD, 1989, pp. 203–221.
[22] G. Varghese, A. Lauck, Hashed and hierarchical timing wheels: efficient data structures for implementing a timer facility, IEEE/ACM Transactions on Networking 5 (6) (1997) 824–834.
[23] S. Floyd, Connections with multiple congested gateways in packet-switched networks. Part 1. one-way traffic, SIGCOMM Computer Communication Review 21 (5) (1991) 30–47.
[24] LBNL, Xerox PARC, UCB, and USC/ISI, The Network Simulator – ns-2. Available from: <http://www.isi.edu/nsnam/ns/>.
[25] T. Wolf, S. You, R. Ramaswamy, Transparent TCP acceleration through network processing, in: Proceedings of the IEEE Global Communications Conference (GLOBECOM), vol. 2, St. Louis, MO, 2005, pp. 750–754.
[26] J. Mahdavi, S. Floyd, TCP-friendly unicast rate-based flow control, Technical Note, Pittsburgh Supercomputing Center. January 1997.
[27] Endace Limited, DAG4.3GE Network Monitoring Interface Card, 2005. Available from: <http://www.endace.com/>.
[28] E.W. Zegura, K. Calvert, J.M. Donahoo, A quantitative comparison of graph-based models for Internet topology, IEEE/ACM Transactions on Networking 5 (6) (1997) 770–783.