The Edge of Smartness

Carey Williamson
Department of Computer Science
University of Calgary
Calgary, AB, Canada
carey@cpsc.ucalgary.ca

***Abstract*—This paper argues the need for "smart edge" devices to enhance the performance, functionality, and security of data center networks. Three examples, drawn primarily from the prior networking literature, are used to illustrate this point. The first example is the *TCP in-cast problem*, wherein highly concurrent TCP flows traverse a limited-buffer LAN switch, degrading system throughput. The second example is *redundant traffic elimination* (RTE), to economize on data movement, particularly over bandwidth-constrained Internet paths. The third example is *speed scaling*, in which metrics such as energy consumption or economic cost trump throughput or response time metrics. The paper concludes with speculative ideas about other functionality that could also reside at the "smart edge".**

***Keywords*-Data Centers, Performance, Scalability, TCP in-cast, Redundant Traffic Elimination, Speed Scaling**

I. Introduction

Data centers are a key part of an emerging IT paradigm, wherein large-scale data management is provided "in the cloud", and manipulation of data is provided using "services-oriented computing". This paradigm has been enabled by several underlying trends in IT technologies, including the growing capacity and reduced cost of storage, improved software support for highly-parallel applications, the pervasiveness of high-speed Internet access, the enhanced functionality of Web browsers, and the ubiquity of mobile data access using personal wireless devices.

This paradigm shift leads us ever-closer to the idealized world of "anytime, anywhere, anything" access to data, on any device. However, it also changes our assumptions about the underlying network, as well as the applications and protocols in use. As such, it is important to consider issues that affect the performance and scalability of the overall system, including not only the data center itself, but also the Internet protocols that affect the user-perceived performance for consumers of data center services.

The performance-related issues that arise in this context are many. One such issue relates to performance metrics. Is throughput the primary metric, as it has been traditionally for bulk data transfer and media streaming applications? Or is response time a more suitable metric, as has been the case for Web-based interaction, gaming, and instant messaging services? Or are other metrics now important? Scanning the current literature indicates that energy conservation and power management are of growing importance, particularly in large-scale data centers. Even small improvements in efficiency can produce dramatic reductions in energy use and operational cost at large-scale. A second issue relates to data filtering. It is intuitively obvious that data should be manipulated, refined, and filtered within the well-provisioned data center before being shipped across the Internet to a user, who might be using a resource-poor mobile wireless device. However, it is less clear how to do this when multiple data centers are involved in cloud-based services, for example to support Web-based mashups. Traditional approaches to object-level compression and caching may not suffice, particulary given the trends toward dynamic data, personalized services, and encrypted traffic. Other techniques for the detection and elimination of redundant traffic may make sense in this context. A third issue relates to network latency. Because data centers support many diverse network applications, some of which are data-intensive, and some of which are latency-sensitive, care is needed in how these applications are supported using Internet protocols. The basic question is "UDP or TCP?", but since most applications require reliable data delivery, TCP is the only viable choice. However, improvements to TCP may be needed to better support data center networks [4].

This combined set of performance and scalability issues provides the primary motivation for this paper, and its premise that additional functionality is required at the edge of the network. The rationale for this view is a simple variation of the well-accepted *end-to-end* design principle for the Internet, in which the core of the network is kept as simple as possible, with all of the "smarts" for intelligent operation pushed into the end systems. The new variation here is that data centers and cloud computing are blurring our notion of what an end system is. As such, putting the "smart edge" functionality close to the data center, but not in the end systems, seem like the most logical choice.

The rest of this paper elaborates upon this argument, and provides three concrete examples that support the rationale for the "smart edge". In particular, Section II describes the TCP in-cast problem, Section III discusses RTE, and Section IV describes speed scaling systems. Section V identifies other additional functionaly that may also belong in the "smart edge". Finally, Section VI concludes the paper.

II. Example 1: TCP In-Cast Problem

The Transmission Control Protocol (TCP) [18] is used as the transport-layer protocol for reliable data transfer in data center networks, just as it is on the Internet. However, data center networks typically have much higher link speeds than the general Internet, much lower loss rates, and much lower propagation delays. For example, the typical propagation delay in a data center network is 0.1 ms, while the default retransmission timeout (RTO) on the Internet is 200 ms.

In a data center network, each user request for data typically results in many servers transmitting data over the data center network concurrently, which can lead to packet losses for a limited-buffer switch. For short-lived flows, packet losses trigger TCP retransmission timeouts, which degrade the throughput of data center applications. This throughput collapse is called the *TCP-incast problem*.

The TCP in-cast problem has spurred a lot of recent research [4], [19], [22], [25]. One approach to solve the problem involves fine-grain timer resolution for TCP retransmission [25], or the use of different TCP variants [4], [19], These approaches require changes at the transport level, or within the operating system (OS) kernel. The other potential solution is to rely on switch-based mechanisms within the network [22].

In our own work [20], we explore the use of application-level flow scheduling to solve the TCP-incast problem. The research question we study is how to schedule responses to client requests to avoid data losses at a bottleneck link. In particular, we model application-level scheduling, which does not require any changes in the TCP stack or the network switches. The main result we derive is an analytical model for the achievable goodput of an application in a data center under lossless scheduling. To verify our theoretical results, we perform extensive simulations. The simulation results confirm the main theoretical results of our model. To the best of our knowledge, our work is the first application-level approach to the TCP-incast problem in data center networks. A "smart edge" device is the obvious location to coordinate the application-level scheduling.

Figure 1 shows a simple model of a data center network. A client connects to the data center via a switch, which in turn is connected to many servers. The client requests data from one or more servers, and the data transfers occur from the servers to the client, via the switch. The bottleneck link is the link from the switch to the client.

The client requests data using a large logical block size (e.g., 1 MB). Within the data center, the actual data blocks are usually striped across many servers using a smaller block size called the Server Request Unit (SRU). A typical SRU size is several kilobytes (KB) [4], though some deployments consider sizes up to 256 KB [19]. A client issuing a request for a data block sends a request packet to each server for its corresponding piece of the requested data block. The request,

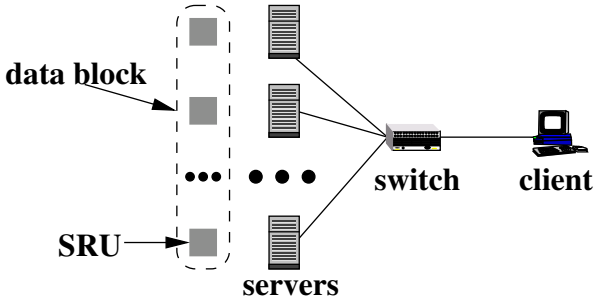


Figure 1. Data center topology

which is served through TCP, is completed only after all SRUs from the requested data block have been successfully received by the client.

Because of the high fan-in from servers to the client, the data transfer workload can overflow the buffer at the bottleneck link, leading to packet losses, and subsequent TCP retransmissions. Since the flow size for an SRU is small, the TCP congestion window is also small, which means that packet losses lead to coarse-grain TCP timeouts. If the value of RTO_{min} is 200 ms [21], and the RTT is 0.1 ms, then significant degradation of application goodput occurs [19].

To validate our model, we perform simulations using the *ns-2* network simulator. We use the network topology shown in Figure 1. The bottleneck link and each access link has 1 Gbps capacity and 0.025 ms propagation delay. Thus, the propagation RTT for each server is 0.1 ms. The buffer size of the bottleneck link is 32 KB. These parameters characterize a typical data center [19]. We set the OS timer granularity Δt to 1 ms [19].

In our simulation, we explore a simple data center scenario with a fixed-size 10 KB SRU. We vary the number of servers from 1 to 50. Using our mathematical model, we can calculate the theoretical value of the goodput g_0 for many servers:

$$g_0 = \lim_{n\to\infty} g = \lim_{n\to\infty} \frac{S}{\widetilde{T}(\lceil \frac{n}{N} \rceil - 1) + T} = \frac{S_{SRU}N}{\widetilde{T}} \quad (1)$$

We calculate that T = 1.887 ms, $\widetilde{T}$ = 2 ms, and N = 5. Substituting the calculated values into Equation (1) gives:

$$g_0 = \lim_{n\to\infty} g \approx 200 \text{ Mbps} \quad (2)$$

Figure 2 shows that the goodput indeed converges to approximately 200 Mbps, and the simulation results show reasonable qualitative and quantitative agreement with the model. Specifically, there are sudden decreases of the theoretical and simulation values of the goodput, the amplitude of which diminishes as the number of servers is increased. For example, the theoretical value goes from 204 Mbps for 15 servers to 162 Mbps for 16 servers. Such a jagged

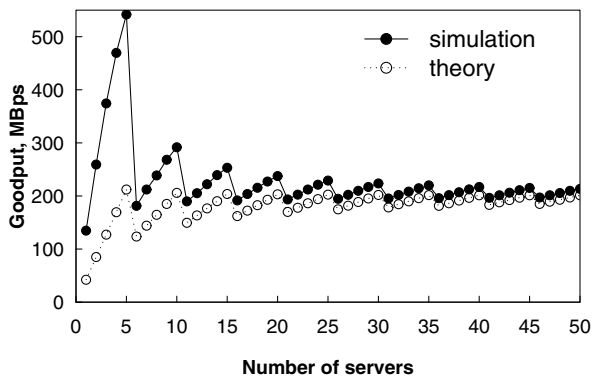


Figure 2. Effect of number of servers on goodput (fixed 10 KB SRU size)

trajectory for the goodput is due to the packetized traffic and the integer number of scheduled responses from servers, which depends on $\lceil \frac{n}{N} \rceil$. The "length" of each step in the zig-zag path is determined by the number of servers, which is $N = 5$ in this scenario.

This simple experiment illustrates the subtlety and the severity of the TCP in-cast problem. For small changes in configuration parameters, there can be throughput changes by a factor of two or more. This problem highlights the need for more careful coordination of data transfers at the edge of the data center network.

III. Example 2: Redundant Traffic Elimination

Redundant Traffic Elimination (RTE) has garnered growing research interest recently [1], [5], [6], [7], [15], [24], as Internet traffic continue to grow and evolve. The estimated redundancy in general Internet traffic is 15-50%. The redundancy arises from the law of large numbers for user behaviour, and highly-skewed popularity distributions for Internet content [9], [14]. As a result, there are many repeated transfers of highly similar content across the Internet.

Repeated transfers of similar data constitute a waste of network resources. This can be a pernicious problem for limited-bandwidth Internet access links (e.g., wireless or cellular access networks), or even for high-bandwidth links operating at or near their capacity. Redundant traffic also has an economic cost, if usage-based billing is applied.

Many techniques have been proposed for RTE, including *protocol-independent* RTE [24], which operates at the network layer. Protocol-independent RTE divides packet payload into *chunks*, and uses unique hash values to detect redundant content [24].

The RTE process can be viewed as a pipeline, as shown in Figure 3. A chunk selection algorithm has a sampling parameter that determines how many chunks are chosen, and a heuristic to determine which ones are chosen. The selected chunks are usually non-overlapping. There is a finite cache of recently-observed packets at each endpoint, usually with a First-In-First-Out (FIFO) *cache replacement policy*. When matching chunks are detected within a packet, an optional *chunk expansion* algorithm can be run at the endpoints to expand the matched region, increasing byte savings [5], [24].

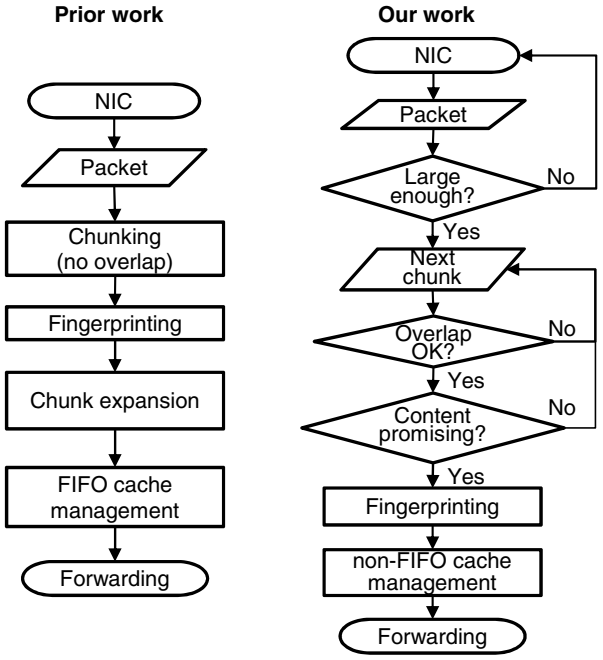


Figure 3. Processing pipeline for RTE

In our own work [17], we have augmented the RTE pipeline with additional functionality, as shown on the right hand side of Figure 3. Our new proposed techniques include size-based bypass, chunk overlap, savings-based cache management, and content-aware chunk selection. We evaluate these techniques on full-payload packet traces from university and enterprise environments, and demonstrate their effectiveness. We also study the impacts of configuration parameters, such as chunk size, sampling period, and cache management policy, on the benefits of RTE, and identify the underlying sources of redundancy in Internet traffic (see Table I).

Table I
Contribution to RTE savings by different content types

Type	Value	Description	Example
Nulls	57.1%	Consecutive null bytes	`0x00000000`
Text	16.7%	Plain text (English)	`Gnutella`
HTTP	7.3%	HTTP directives	`Content-Type:`
Mixed	6.2%	Plain text and other chars	`14pt font`
Binary	5.8%	Random characters	`0x27c46128`
HTML	3.7%	Fragment of HTML code	`<HTML> <p>`
Char+1	3.2%	Repeated text chars	`AAAAAAAz`

Our key results include the following:

- Using 64-byte rather than 32-byte chunks improves RTE by up to 21%, while reducing execution time.

- Size-based bypass reduces execution time by 2-25%, while *improving* RTE up to 4% in some cases.
- Chunk overlap can improve RTE by 9-14%.
- A savings-based cache replacement policy can improve the effectiveness of RTE by up to 12%.
- The cumulative effects of the foregoing techniques improve existing RTE savings by up to 54%.

We believe that deployment of RTE solutions at the "smart edge" can help address bottleneck bandwidth issues, either within or beyond the data center network. Having RTE solutions inside the data center network itself may in fact be the most appropriate, since the data volumes are higher, and the captive data is more likely to be unencrypted. If the network traffic leaving the data center network is encrypted (e.g., Binary), then there is relatively little potential benefit for RTE.

IV. Example 3: CPU Speed Scaling

Have you ever wished that your desktop computer had a "Go faster!" button? Something that you could push to get simulations done for your conference paper deadline, or complete annual reports when the fiscal year ends. Some humans are able to get work done more quickly when critical deadlines approach, but what about computers?

Many computers already have this functionality, although the feature is not widely known, or used particularly well. The technical term for this feature is Dynamic Voltage and Frequency Scaling (DVFS), and it allows the capabilities of the CPU (e.g., clock rate, power consumption) to be adjusted dynamically, under software control by the operating system [23]. Modern processors typically support on the order of a dozen discrete operating states, rather than just two (On/Off) or three (On/Sleep/Off). This functionality for dynamic adjustment of the processor speed[1] is often referred to as "CPU speed scaling".

The implications of speed scaling are interesting to explore even in the "simple" single processor case. One of the earliest papers on this topic was by Weiser *et al.* in 1994 [26], who used simulation to explore the tradeoffs between response time and CPU energy consumption for empirical Unix workloads. This work pre-dated the landmark YDS 1995 paper [27] on speed scaling, which provided the first formal treatment of the speed scaling problem, including several heuristic algorithms, as well as proofs that they were within a constant factor of optimal. The latter paper triggered substantial follow-on work by Albers [2], [3], Bansal [11], [12], [13], and others on improved algorithms, tighter bounds, and alternative metrics for evaluating speed scaling designs. In essence, the traditional performance metrics of throughput and response time become secondary to energy consumption (or a weighted combination of energy consumption and response time) in this context. These new metrics are especially relevant in data centers, where large-scale computing resources are managed.

One intriguing paper on this topic appeared in ACM SIGMETRICS 2010 [8]. In it, the authors argue that in speed scaling systems, tradeoffs exist between optimality, fairness, and robustness. In particular, it is possible to provide any two of these properties, but not all three. For example, Shortest Remaining Processing Time (SRPT) scheduling can be used to provide theoretically-optimal response time, assuming job demands are fully known in advance, but it will create bias against large jobs, and it will not be very robust if there are uncertainties in the workload. Conversely, Processor Sharing (PS) is fair to all jobs, and robust, but its response time can be much worse than SRPT.

In our own current work, we are exploring the role of scheduling in speed scaling systems, and in particular the tradeoffs between efficiency, fairness, and cost. Efficiency refers to how closely the system achieves optimality, such as minimizing response time, maximizing throughput, or maximizing user utility. Fairness refers to whether jobs with different characteristics (e.g., size, priority, CPU-bound, I/O-bound) are treated similarly or not. Cost currently refers to the aggregate energy consumption for the system when executing a given workload, though we ultimately envision translating this into an economic (dollar) cost model based on dynamic energy pricing.

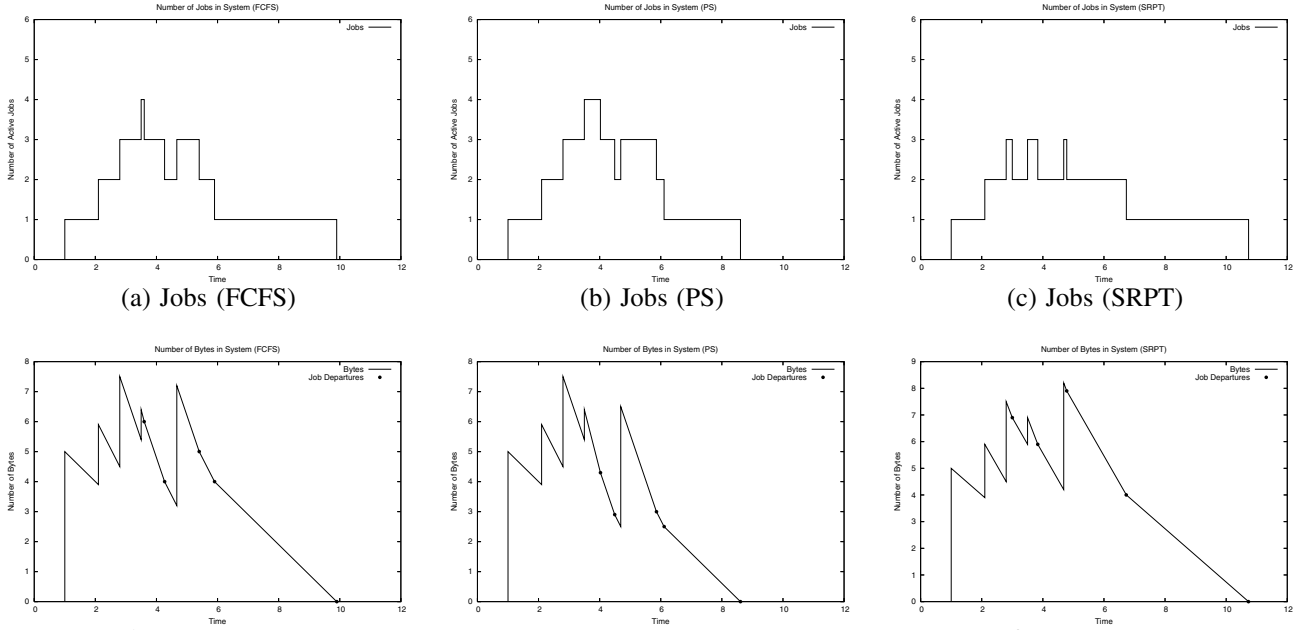
To illustrate the issues, consider a simple single processor system, initially empty, to which a small set of jobs arrive, as shown in Table 1. We have built a simple discrete-event simulation model of this system, with configurable scheduling policies, and adequate instrumentation to record job response times and execution costs.

Job ID	Arrival Time	Job Size (MB)
1	1.0	5
2	2.2	2
3	2.8	3
4	3.5	1
5	4.7	4

Figure 4 shows an example of our simulation results for the workload in Table 1. The top row of graphs shows the instantaneous number of jobs in the system, for each of three scheduling policies (FCFS, PS, and SRPT, from left to right). The second row of graphs shows the instantaneous amount of work (MB) in the system, when the CPU speed is scaled dynamically based on the number of jobs in the system.

Multiple insights emerge from studying Figure 4, and other variations of it. The first observation is that the three scheduling algorithms produce different profiles for the number of jobs in the system, and different response times, since they complete the jobs at different times and in different orders. This is an obvious property of scheduling algorithms, and indeed the reason for their existence. The

[1] A similar concept applies to network links with adjustable physical capacity, but the effects are less direct since the transmission time is only one component of the end-to-end network delay.



(a) Jobs (FCFS) (b) Jobs (PS) (c) Jobs (SRPT)

(d) Bytes (FCFS) (e) Bytes (PS) (f) Bytes (SRPT)

Figure 4. Example Results for Dynamic Speed Scaling

second observation is that speed scaling based solely on the amount of work in the system would have no effect on energy consumption, since the speed scaling decisions would be the same for each scheduling algorithm (since the byte backlog is identical for all work-conserving scheduling algorithms [16]). In other words, scheduling algorithms are only interesting if the speed scaling is based on the number of jobs in the system, as it is in Figure 4. Scaling solely based on the amount of work in the system would also create unfairness, since (for example) a single large job in the system would receive service at a higher rate than a single small job in the system. For these reasons, most speed scaling designs are based, at least in part, on the number of jobs in the system. The third observation is that the scheduling algorithms have different energy costs. For example, FCFS runs the CPU at a moderate speed for most of the simulation, while SRPT tends to run it at a lower speed (since there are fewer jobs in the system), but for a longer time. If power consumption is proportional to the square of the CPU speed (a typical modeling assumption in the literature), then scheduling algorithms such as SRPT might be advantageous. However, the final observation is that SRPT may not be optimal in all respects. While SRPT tends to complete jobs quickly, has fewer jobs in the system, and runs the CPU at a lower speed, it also means that the system needs to run longer to complete the jobs. FCFS and PS both finish sooner, in this particular example.

While this example is admittedly trivial, it does illustrate the basic tradeoffs between efficiency, fairness, and cost. These issues are interesting in the single processor case, and even more challenging for multiple processors [2].

In the data center context, we see an important role for a "smart edge" device, whose purview includes not only the data center itself (e.g., job scheduling, number of servers, speed scaling, power consumption), but also the consideration of network-related effects (e.g., TCP throughput, RTT, fairness) that impact user-perceived response time. Much work remains to be done here.

V. Additional Functionality

The three foregoing examples illustrate some of the performance and scalability issues for data center networks, and provide justification for the "smart edge" approach advocated in this paper. However, they are not the only examples. Below, I speculate on other possible functionality that might belong at the "smart edge" for data center networks:

- **Upload managers**. Many Internet users make use of *download manager* software when retrieving large files, such as media content files, from the Internet. These software packages split large digital content into smaller pieces, and download the pieces in parallel from several different hosts, often using multiple concurrent TCP connections. Download managers are useful because they reduce download time, by exploiting parallelism.

  In the content download scenario, the download manager is essentially a multiplexing point for collecting the aggregate traffic, before delivering it to the client,

typically resident on the same local host. However, the picture is slightly different in the data center scenario. In some sense, the LAN switch at the data center is an intermediate multiplexing point, which then must deliver the data to the requesting client across the Internet.

Perhaps an "upload manager" is apropos here, to operate in concert with the download manager at the client end. This would allow the client to interact more efficiently with the "smart edge", rather than with the data center itself. Splitting the control loop for end-to-end data delivery in this fashion is analogous to Indirect-TCP [10], which efficiently delivers wired Internet content to mobile clients on wireless access networks.

- **Third-party transfers**. The foregoing discussion has assumed that data centers are used in the traditional client-server paradigm. However, one can envision Web-based services, such as mashups, that require interactions between multiple data centers before the final response for the client is fully composed. In such a scenario, a suitable framework is required for server-server interaction, without the client having to request, merge, and join all the data separately. The smart edge could again play an important role here, perhaps by facilitating efficient and concurrent control of third-party data transfers, rather than iterative or recursively nested serial interactions with each data source. This topic requires further investigation.
- **Security and privacy**. Last but not least, there are many security and privacy concerns associated with data warehouses, cloud computing, and Web-based services. Is my data safe? Who has access to data? How do I prevent my data from being used in unintended ways? As data volumes, Internet accessibility, and cybercrime activities continue to grow, these questions are at the forefront for many individuals, institutions, and governmental organizations. Suitable functionality may be required in the "smart edge" to alleviate the security and privacy concerns for data center networks.

VI. Summary

In this paper, I argue the need for "smart edge" devices that can enhance the performance, functionality, and security of data center networks. Three specific examples, namely TCP in-cast, redundant traffic elimination, and CPU speed scaling are presented as case studies to illustrate the issues, and provide justification for the "smart edge" approach. Other possible roles for "smart edge" devices are also identified. In addition to the need for such devices at the data center edge of the network, there may well be a need for such devices at the client edge as well, as the complexity and variety of Internet access devices continues to grow.

Acknowledgements

The author thanks Maxim Podlesny for his work on TCP in-cast, Emir Halepovic for his detailed explorations of RTE, and Maryam Elahi for inspiring our ongoing investigations of speed scaling systems. Financial support for this work was provided by the Natural Sciences and Engineering Research Council (NSERC) in Canada, and the Informatics Circle of Research Excellence (iCORE) in the Province of Alberta.

References

[1] B. Aggarwal, A. Akella, A. Anand, A. Balachandran, P. Chitnis, C. Muthukrishnan, R. Ramjee, and G. Varghese, "EndRE: An End-System Redundancy Elimination Service for Enterprises", *Proceedings of USENIX NSDI*, San Jose, CA, pp. 419-432, April 2010.

[2] S. Albers, F. Mueller, and S. Schmelzer, "Speed Scaling on Parallel Processors", *Proceedings of ACM Symposium on Parallelism in Algorithms and Architectures* (SPAA), pp. 289-298, 2007.

[3] S. Albers, "Energy-Efficient Algorithms", *Communications of the ACM*, Vol. 53, No. 5, pp. 86-96, May 2010.

[4] M. Alizadeh, A. Greenberg, D. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data Center TCP (DCTCP)", *Proceedings of ACM SIGCOMM*, pp. 63-74, New Delhi, India, September 2010.

[5] A. Anand, A. Gupta, A. Akella, S. Seshan, and S. Shenker, "Packet Caches on Routers: The Implications of Universal Redundant Traffic Elimination", *Proceedings of ACM SIGCOMM*, Seattle, WA, pp. 219-230, August 2008.

[6] A. Anand, C. Muthukrishnan, A. Akella, and R. Ramjee, "Redundancy in Network Traffic: Findings and Implications", *Proceedings of ACM SIGMETRICS* Seattle, WA, pp. 37-48, June 2009.

[7] A. Anand, V. Sekar, and A. Akella, "SmartRE: An Architecture for Coordinated Network-wide Redundancy Elimination", *Proceedings of ACM SIGCOMM*, Barcelona, Spain, pp. 87-98, September 2009.

[8] L. Andrew, M. Lin, and A. Wierman, "Optimality, Fairness, and Robustness in Speed Scaling Designs", *Proceedings of ACM SIGMETRICS*, pp. 37-48, June 2010.

[9] M. Arlitt and C. Williamson, "Internet Web Servers: Workload Characterization and Performance Implications", *IEEE/ACM Transactions on Networking*, Vol. 5, No. 5, pp. 631-645, October 1997.

[10] A. Bakre and B. Badrinath, "I-TCP: Indirect TCP for Mobile Hosts", *Proceedings of IEEE ICDCS*, Vancouver, BC, pp. 136-143, May 1995.

[11] N. Bansal, T. Kimbrel, and K. Pruhs, "Speed Scaling to Manage Energy and Temperature", *Journal of the ACM*, Vol. 54, 2007.

[12] N. Bansal, K. Pruhs, and C. Stein, "Speed Scaling for Weighted Flow Time", *Proceedings of ACM-SIAM Symposium on Discrete Algorithms*, 2007.

[13] N. Bansal, H. Chan, and K. Pruhs, "Speed Scaling with an Arbitrary Power Function", *Proceedings of ACM-SIAM Symposium on Discrete Algorithms*, 2009.

[14] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web Caching and Zipf-like Distributions: Evidence and Implications", *Proceedings of IEEE INFOCOM*, New York, NY, March 1999.

[15] F. Douglis and A. Iyengar, "Application-specific Delta-encoding via Resemblance Detection", *Proceedings of USENIX Technical Conference*, San Antonio, TX, pp. 113-126, June 2003.

[16] M. Gong and C. Williamson, "Revisiting Unfairness in Web Server Scheduling", *Computer Networks*, Vol. 50, pp. 2183-2203, 2006.

[17] E. Halepovic, C. Williamson, and M. Ghaderi, "Enhancing Redundant Network Traffic Elimination", submitted for publication, 2010.

[18] V. Jacobson, "Congestion Avoidance and Control", *Proceedings of ACM SIGCOMM*, August 1988.

[19] A. Phanishayee, E. Krevat, V. Vasudevan, D. Anderson, G. Ganger, G. Gibson, and S. Seshan, "Measurement and Analysis of TCP Throughput Collapse in Cluster-based Storage Systems", *Proceedings of FAST*, February 2008.

[20] M. Podlesny and C. Williamson, "An Application-Level Solution to the TCP-incast Problem in Data Center Networks", submitted for publication, 2011.

[21] S. Rewaskar, J. Kaur, and F. Smith, "A Performance Study of Loss Detection/Recovery in Real-world TCP Implementations", *Proceedings of IEEE International Conference on Network Protocols*, October 2007.

[22] A. Shpiner and I. Keslassy, "A Switch-Based Approach to Throughput Collapse and Starvation in Data Centers", *Proceedings of IEEE International Workshop on Quality of Service* (IWQoS), June 2010.

[23] D. Snowdon, E. Le Sueur, S. Petters, and G. Heiser, "Koala: A Platform for OS-level Power Management", *Proceedings of ACM EuroSys*, pp. 289-302, 2009.

[24] N. Spring, and D. Wetherall, "A Protocol-independent Technique for Eliminating Redundant Network Traffic", *Proceedings of ACM SIGCOMM*, Stockholm, Sweden, pp. 87-95, August 2000.

[25] V. Vasudevan, A. Phanishayee, H. Shah, E. Krevat, D. Anderson, G. Ganger, G. Gibson, and B. Mueller, "Safe and Effective Fine-grained TCP Retransmissions for Datacenter Communication", *Proceedings of ACM SIGCOMM*, August 2009.

[26] M. Weiser, B. Welch, A. Demers, and S. Shenker, "Scheduling for Reduced CPU Energy", *Proceedings of USENIX Operating System Design and Implementation* (OSDI), 1994.

[27] F. Yao, A. Demers, and S. Shenker, "A Scheduling Model for Reduced CPU Energy", *Proceedings of ACM Foundations of Computer Systems* (FOCS), pp. 374-382, 1995.