International Journal of Networking and Computing – www.ijnc.org, ISSN 2185-2847
Volume 15, Number 1, pages 51–63, January 2025

Application of Network Calculus Models to Heterogeneous Streaming Applications

Clayton J. Faber
SimpleRose
St. Louis, MO, 63101, USA

Roger D. Chamberlain
Dept. of Computer Science and Engineering
Washington University in St. Louis
St. Louis, MO, 63130, USA

Received: June 22, 2024
Revised: October 17, 2024
Accepted: November 26, 2024
Communicated by Masahiro Shibata

Abstract

Network calculus has seen extensive use in the performance modeling of communications systems. Here, we apply network calculus techniques to the modeling of streaming data applications running on heterogeneous computing platforms. We quantitatively compare the performance predictions from network calculus with predictions from a discrete-event simulation model and a previously presented queuing theory model for two different applications.

Keywords: Performance modeling, Network calculus, Streaming data systems

1 Introduction

In streaming data applications it can be difficult to reason about performance prior to deployment. Moreover, the application might need to be expanded to transform data into the appropriate input format or handle metadata processing. If one chooses to accelerate stages using heterogeneous hardware, data movement also becomes a concern when data is migrated to a new memory domain or the data movement takes place between physically separate networked compute resources.

When reasoning about performance in streaming applications it is often helpful to utilize analytical models to gain insights into how to spend time and development resources prior to a full deployment test. Queuing models have a long history for this purpose [3]. One can readily apply queuing theory models to these streaming applications. Padmanabhan et al. [25] utilized queuing models to reason about streaming applications on heterogeneous architectures. Faber et al. [12] present a queuing model for the BLAST biosequence alignment application [2] deployed on heterogeneous execution platforms that we will use as one of our illustrative applications below. These models, based on $M/M/1$ queuing networks, utilize isolated measurements of individual compute stages and flow analysis to identify bottlenecks and identify where developers can focus their attention for performance improvements. While these models can provide steady-state mean flow analysis, similar to most queuing models, without further experimentation and measurement effort to characterize both arrival and service distributions, it can be difficult to determine bounds on time

and buffering requirements. Furthermore, queueing models can also have difficulty capturing the effects of data bundling between stages, and how the delay of waiting for jobs effects the overall throughput. To answer these shortcomings we turn to network calculus.

Network calculus is an analytic modeling technique that applies system theory concepts to computer communication networks utilizing min-plus algebra [10, 11, 21]. Although the technique was originally designed with network elements in mind, in this paper we propose additions to the network calculus modeling technique to reason about streaming data applications in a heterogeneous environment where the movement of data between memory domains and across network links is of vital importance. With modifications to support computational elements, in addition to network elements, deterministic network calculus models can give insights into delay due to data aggregation at nodes (packetization), end-to-end delay, and bounds on data flow through the overall application or through individual subsets of stages in the stream. Furthermore, the models retain the desirable property of being derived from measurements taken in isolation without a full deployment, similar to the aforementioned queuing theory models.

Here, we propose the use of network calculus modeling techniques to analyze the performance of streaming data computations. We include in the network calculus models elements of both data communication (which is typical of network calculus) and data computation (which is new). We compare the proposed network calculus models to a discrete-event simulation designed to mimic a collection of computation nodes with multiple stages in a streaming data application. In the example applications we model two types of communication links, traditional network links and PCIe buses, in addition to compute components executing individual stages of a streaming data application. Through both modeled results and simulated results we show the utility of network calculus when considering performance in streaming data applications.

2 Background and Related Work

Network calculus is a modeling approach, similar to queuing theory, that is designed to analyze systems that utilize queues and has historically been primarily used to analyze bounds and model performance in networking systems. It relies on the min-plus and max-plus algebras, which define a different set of operators compared to normal algebra. In min-plus algebra, addition is replaced by the infimum operator and multiplication is replaced with addition. Similarly in max-plus algebra, addition is replaced by the supremum and, once again, multiplication is replaced with addition. These two algebras are used in conjunction with the convolution operator to reason about data as it traverses a system.

In network calculus, data are modeled by a cumulative function with respect to time to represent the flow in and out of systems. Systems are modeled in a similar fashion with curves representing guarantees on flow into and out of the system, known as arrival and service curves, respectively.

Consider a data flow, in units of bits, $r(t)$, arriving at a system and let $\alpha(t)$ be a wide-sense increasing function with $\alpha(0) = 0$. The flow is constrained by $\alpha(t)$ and is an arrival curve if and only if for any $0 \leq s \leq t$:

$$r(t) - r(s) \leq \alpha(t - s).$$

Following a similar logic the system offers a service guarantee for an output flow $r^*(t)$. Allow $\beta(t)$ to be a wide-sense increasing function and $\beta(0) = 0$. $\beta(t)$ is a service curve given to the flow $r(t)$ with an output curve $r^*(t)$, defined by:

$$r^*(t) \geq \inf_{s \leq t} \{r(s) + \beta(t - s)\}.$$

Alternatively, this can be written as the min-plus convolution:

$$r^*(t) \geq r(t) \otimes \beta(t).$$

Furthermore, we can define an upper-bound on service provided defined as:

$$r^*(t) \leq r(t) \otimes \gamma(t),$$

where $\gamma(t)$ is the maximum (i.e., best case) service curve.

When utilizing a network calculus model it is up to the designer to use appropriate equations to represent arrival and service curves. For arrival curves it is common to model the data flow using an affine curve known as the leaky bucket arrival curve:

$$\alpha(t) = \begin{cases} R_\alpha \cdot t + b & \text{if } t > 0 \\ 0 & \text{otherwise.} \end{cases}$$

Here, R_α represents the rate of arrival and b is a burst, that is, how much data can be sent instantaneously. When considering the service curves, these are commonly represented as rate latency functions with an associated rate, R_β, and delay, T, associated with them:

$$\beta(t) = \begin{cases} R_\beta \cdot (t - T) & \text{if } t > T \\ 0 & \text{otherwise.} \end{cases}$$

By utilizing these models we can reason about bounds on a specific node such as the backlog generated by the flow entering the node, the delay data will experience at a given node, and what is the upper-bound output flow of the node. Figure 1 displays a data over time plot of a leaky-bucket arrival curve and two rate latency functions representing both a maximum and normal service curve adapted from [21]. Also in this figure horizontal and vertical lines are included that are meant to represent maximum virtual delay and backlog respectively. Finally from these two lines we can derive an output flow bound, $\alpha^*(t)$, which, along with the delay and backlog, will be further expanded upon below.

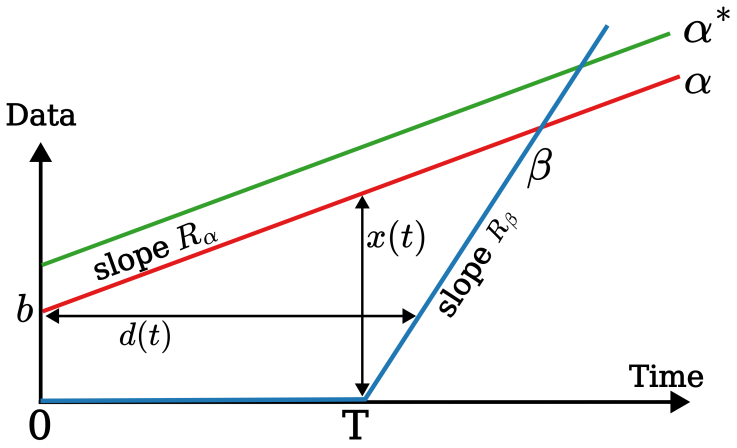


Figure 1: Plot of a Leaky Bucket Arrival Curve, α, and a Rate-Latency Service Curve, β, showing the relation of the Backlog, $x(t)$, Virtual Delay, $d(t)$, and Output Flow, α^*, bounds. Adapted from [21].

The use of network calculus is widespread in networking systems [4, 5, 20, 28]. These applications are mostly concerned with extensions to other models, such as network firewalls [33] and job scheduling [23]. Network calculus also has two sub-branches; one that deals with probabilistic systems, that behave in a stochastic manner, called stochastic network calculus [17], and the other dealing with hard real-time deadlines, known as real-time network calculus [30]. In this particular work we use the standard, deterministic, network calculus and this is, as far as we know, the first application of these models to streaming computations that specifically target heterogeneous architectures.

There are of course other examples of modeling for heterogeneous architectures that use other types of models. Faber et al. [12] apply a queuing network model to perform flow analysis, estimating roofline performance for the overall application. In that study, the actual performance was almost 30% lower than what the roofline model predicted. While this model is indeed useful it would be

beneficial to have a more holistic view of the system along with a roofline, which the model presented here aims to do. When considering queuing theory models it is important to point out that both network calculus and queuing theory as mathematical models are designed to reason about queuing systems and there has been work to explain how one represents network calculus ideas in a queuing theory space [16, 26].

3 Network Calculus Modeling

As mentioned prior we want to use network calculus to reason about bounds on a given streaming application that utilizes heterogeneous architectures, however some additional assumptions and modifications to the standard model must be made in order to utilize it properly. Firstly, network calculus in its original inception deals with continuous data flows that are bit-by-bit, however in the modern era a majority of network equipment work on a per-packet scheme and are similar to jobs flowing through a streaming application. This packetization does indeed have an effect on some of the properties that network calculus models [21] and needs to be accounted for in our final model as well. These adjustments come in the form modifying the arrival and service curves with a variable that describes the size of the maximum packet l_{max}. Consider a flow $r(t)$ and a packetizer, P^L, the packetized version of the arrival, service, and maximum service curves are [32]:

$$P^L(r(t)) \leq \alpha(t) + l_{max}1_{t>0}$$

$$\beta'(t) = [\beta(t) - l_{max}]^+$$

$$\gamma'(t) = \gamma(t).$$

where $1_{t>0}$ is 0 for $t \leq 0$ and 1 for $t > 0$.

With these adjustments we can now talk about important bounds previously mentioned and shown in Figure 1, virtual delay and backlog. The virtual delay, $d(t)$, is a measure of the maximum amount of time it takes for a system to output the same amount of data sent to the system. For a leaky bucket arrival curve α and a rate latency service curve β the virtual delay is given by:

$$d(t) \leq T + \frac{b}{R_\beta},$$

where T is the delay in the expression for β and b is the burst size in the expression for α. The backlog bound, $x(t)$, is a bound on the maximum amount of data that resides in the server before output is sent, and is calculated as the maximum deviation between α and β. In this example it is calculated as:

$$x(t) \leq b + R_\alpha \cdot T.$$

Finally, we can make an estimation of the output bound of a system, $\alpha^*(t)$. This is known as the output flow bound. It is found by calculating both a min-plus convolution and a min-plus de-convolution utilizing the arrival curve of the node and both the maximum and normal service curves:

$$\alpha^* = (\alpha \otimes \gamma) \oslash \beta.$$

While these bounds are beneficial to have, it is important to know that these bounds assume that $R_\alpha \leq R_\beta$. If $R_\alpha > R_\beta$ it is noted in [21] that the bounds are infinite, which is the same result predicted by queuing theory if the arrival rate is greater than the service rate, resulting in an infinite bound on the queue. Taking this into account, there are three particular scenarios that we are interested in: when $R_\alpha < R_\beta$ or standard operation, when $R_\alpha = R_\beta$, and finally when $R_\alpha > R_\beta$. While the bounds are indeed infinite for backlog and virtual delay over the long run, we hypothesize that we can use values given by the model to understand estimates on required queue size for individual nodes as a job traverses a system implementing a streaming data application.

One important aspect of targeting heterogeneous architectures is the need to gather enough data to make dispatching a job worthwhile. The inherent overheads associated with initiating a computation on an attached accelerator, for example, can motivate the aggregation of a minimum

data volume at the input to the accelerator prior to dispatching the job to the accelerator. We call this metric the job ratio. To reflect this in the service curve representations, we have made a modification to how initial delay is calculated at these nodes. For a node n that collects data of size b_n prior to initiation, where b_n is larger than the burst rate of the previous node ($b_n > b^*_{n-1}$), the latency at node n is:

$$T_n^{tot} = T_{n-1}^{tot} + \frac{b_n}{R_{\alpha_{n-1}}} + T_n.$$

Intuitively, total latency is the summation of initial delay of the previous nodes, T_{n-1}^{tot}, the time to collect a job from the previous node, $b_n/R_{\alpha_{n-1}}$, and finally the initial delay of the current node, T_n.

4 Modeling of Biosequence Alignment Application BLAST

Actual streaming data applications are often modeled as a chain of nodes interconnected into a directed acyclic graph. The model nodes in the chain might represent computation and/or communications (as described in [6]), especially given that data movement in a heterogeneous environment can be critical for performance. We believe that network calculus is well suited for capturing this type of data movement and can be a viable tool for measuring the effects of data channels in streaming environments.

4.1 BLAST

The stages of our BLAST implementation mirror the stages of the NCBI BLASTN computation pipeline, shown in Figure 2, and it is built using the Mercator framework on a GPU [9]. The DNA database to be searched, represented in FASTA format, is first converted to two bits per DNA base. This is a pre-processing step, `fa_2bit`, from the Data Integration Benchmark Suite (DIBS) [8] that is implemented on an FPGA [13]. In the next computational stage, `seed match`, each byte-aligned 8-mer (8-base word) of the database is checked to see whether it appears in a hash table (stored in GPU DRAM) constructed from all 8-mers of the query sequence. If the 8-mer at database position p does appear in the table, a third stage, `seed enumeration`, accesses the table to enumerate all positions q at which it appears, generating one or more 8-mer matches (p, q). These matches are passed to the fourth stage, `small extension`, which attempts to extend each match to the left and right by up to 3 bases. If a match (p, q) can be extended to a total length of at least 11, it is passed on to the final stage, `ungapped extension`, which extends the match to the left and right, this time allowing scoring of both matches and mismatches. Our implementation limits ungapped extension to at most a fixed-size window (currently 128 bases) centered on the initial seed match. Only seed matches whose highest-scoring ungapped extension score above a specified threshold are returned for further processing. Our implementation does not presently perform gapped extension [1], but for BLASTN, that stage takes negligible time compared to the rest of the pipeline [18] and would be implemented on the host processor.

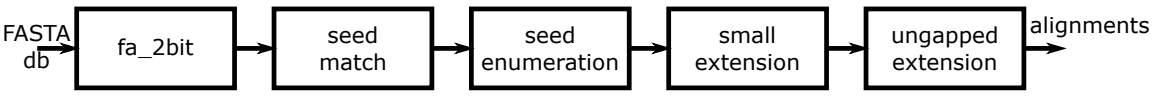


Figure 2: BLAST application.

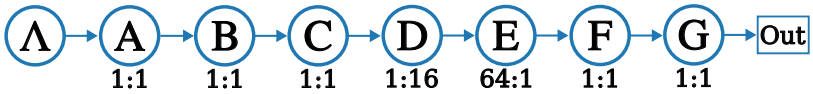
Most stages of BLASTN act as filters over either database positions (seed matching) or matches (small and ungapped extension). Their task is to eliminate inputs that should not proceed to the next stage. Seed matching in particular is a highly effective filter, eliminating the vast majority of input 8-mers, for query lengths much less than 2^{16} bases. Seed enumeration, in contrast, may produce multiple matches per input position if the same 8-mer occurs at several places in the query. Except for highly repetitive query sequences, this stage produces on average 1-2 matches per input position.

All stages of BLASTN produce a variable number of outputs per input, and most produce zero outputs for the majority of their inputs. On a SIMD processor such as a GPU, executing all stages of BLASTN independently in each thread will result in many threads discarding their inputs and becoming idle early in the computation, resulting in many wasted cycles. The Mercator system therefore inserts queues between each stage to collect and redistribute work among threads before executing the next stage. These queues have limited size, so each stage may need to be executed multiple times; scheduling execution of stages is performed so as to maximize GPU thread occupancy and minimize overhead [27].

4.2 Network Calculus Model

Given a set of N nodes representing stages of a heterogeneous streaming application, we can create network calculus maximum and normal service curves to represent the guarantees on service at each node. Along with the actual compute nodes we can also create service curves to represent guarantees on data movement. These nodes can be concatenated together to find the overall service curve of the full system. Going further, we can create models for intermediate systems by finding service curves for a subset of contiguous nodes.

To test this model we use the BLAST application [2] described in Faber et al. [12]. Figure 3 illustrates the setup of this application with nodes representing stages of the streaming data application.



Node	Function	Node	Function
Λ	Data source	D	Network link
A	PCIe link	E	PCIe link
B	FPGA computation	F	GPU computation
C	PCIe link	G	PCIe link

Figure 3: Data flow diagram for BLAST. Nodes represent computations or communications, and the job ratio is shown below each node (i.e., the ratio of input data block size to output data block size). Node D decomposes large data blocks from the FPGA for delivery over the network, and Node E composes even larger data blocks for delivery to the GPU.

We take these nodes as described and model their execution time in a discrete-event simulator facilitated by the SimPy library [29] in Python3. Each node is given a maximum and minimum execution time, a data packet size to consume, and data packet size to emit when the execution time has completed. Discrete events in the simulation model include arrival of a data packet at a node, initiation of execution of that data packet when the node becomes free (completes its execution of previous data packets), and departure of the data packet from the node (once execution has completed for that data packet). The time chosen for execution is chosen from a uniform random distribution using the minimum and maximum times as bounds. A comprehensive description of the discrete-event simulation techniques used is provided in [34], which primarily uses the simulation to model graph neural network applications and also includes a simulation of the example application of Section 5 below.

Following the approach of Timcheck and Buhler [31], we normalize the data volumes at each stage referred to the input, as some stages have a natural lossless data compression. Below we report all of the following: network calculus predictions on bounds, the results of the original $M/M/1$ queuing theory model and empirical measured performance (from [12]), and our simulated system performance.

The predictions from our network calculus model and discrete-event simulation are depicted in Figure 4. The service curve, represented by $\beta(t)$, corresponds to the lower bound of predicted performance. The arrival curve, represented by $\alpha(t)$, corresponds to an upper bound on performance. The output flow bound, represented by $\alpha^*(t)$, is a loose upper bound. The simulated data output is shown by the stairstep curve that stays between the two bounds.

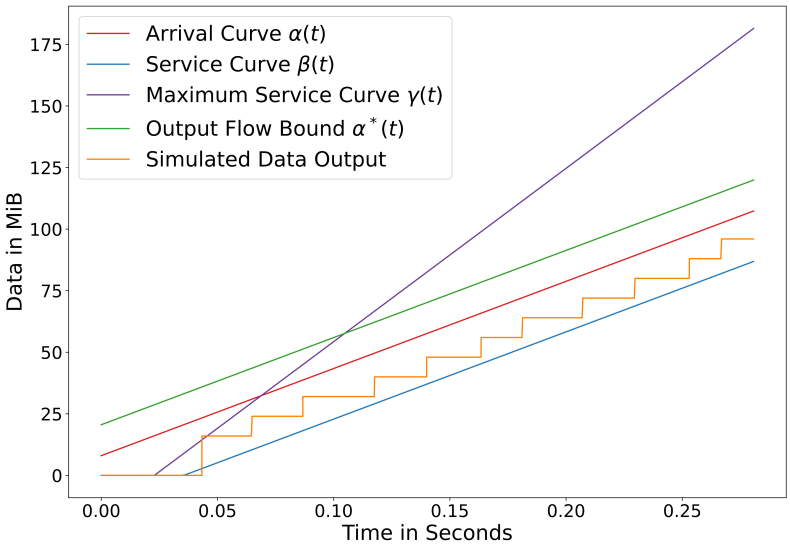


Figure 4: Network calculus model results for BLAST application.

Throughput predictions from the various models and experiments are presented in Table 1. As is apparent, the network calculus throughput predictions align well with both the discrete-event simulation results and the empirical results reported in [12].

Table 1: BLAST streaming data application throughput.

Source	Value
Network calculus upper bound	704 MiB/s
Network calculus lower bound	350 MiB/s
Discrete-event simulation model	353 MiB/s
Queuing theory prediction [12]	500 MiB/s
Measured throughput [12]	355 MiB/s

Note that network calculus does not presume to make a nominal data throughput prediction. Rather, it provides upper and lower bounds on the throughput. In this case, both the discrete-event simulation throughput prediction and the measured throughput are just slightly above the network calculus predicted lower bound.

While these throughput results are clearly of interest, they haven't yet demonstrated the power of network calculus, since they are merely confirming the conclusions from previous models. Additional information we can glean from the network calculus model include the following:

1. The maximum virtual delay, d, through the system is modeled to be 46.9 ms.
2. The maximum data occupancy resident in the system (or backlog bound), x, is modeled as 20.6 MiB.

Points (1) and (2) above are corroborated by the discrete-event simulation model. For the maximum virtual delay, the longest observed delay in the simulator is 46.4 ms and the shortest delay being

40.7 ms. The maximum amount of data in system backlog accounting for all nodes and queues was observed to be 20.1 KiB. These values are within the bounds modeled using network calculus.

Further capabilities of the network calculus models include the ability to analyze any desired subset of the streaming application separate from the rest of the application. For example, the contributions of the data occupancy bounds that are due to each node in Figure 3 can be determined analytically, which can assist a developer in allocating buffers.

5 Bump in the Wire Streaming Algorithms

Utilizing network calculus we can model other data streaming applications that utilize other heterogeneous technology (and validate the model predictions using our simulation tool). One that is of particular interest to us is the utilization of what is known as "bump in the wire" communication [7]. Figure 5 shows the traditional interconnect for an FPGA accelerator, and Figure 6 shows the bump in the wire configuration.

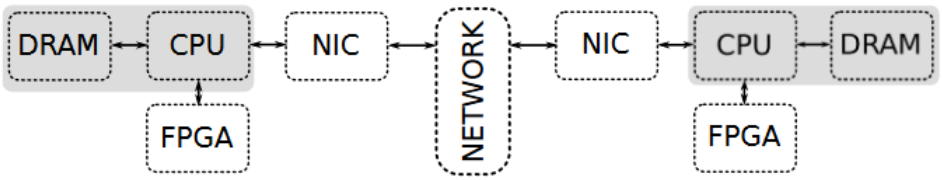


Figure 5: Traditional FPGA accelerator [19].

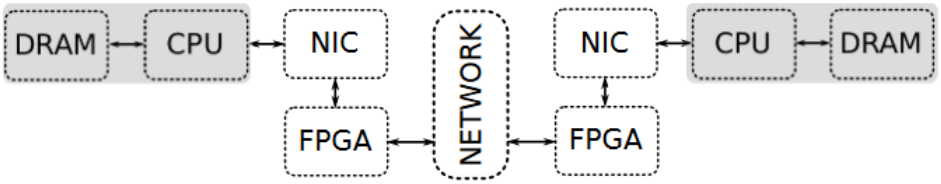


Figure 6: Bump in the wire FPGA accelerator [19].

This style of deployment particularly relies on a network connection to pass data to a new node without having to be moved out of the heterogeneous memory pool back to CPU host memory. There are many heterogeneous architectures that implement bump in the wire tech that are primarily deployed in FPGAs, utilizing custom compute alongside network communication. In this scenario instead of a specific streaming algorithm we want to look at adding functionality to a network connection, tasks usually not associated with a specialized algorithm but still desireable in many implementations. Tasks like security and/or compression are often afterthoughts when considering an application development, frequently to be considered essential when it comes to deployment. These algorithms, depending on their implementation, can be considered a type of streaming data application by compressing/encrypting data blocks in chunks and then decompressing/decrypting at the destination. Two FPGAs can be used in conjunction as a source and destination though a network to offload the entirety of this computation from the endpoint CPUs freeing them up for other processes.

Figures 7 and 8 show the source end flow graphs of the scenario described above, with Figure 7 indicating the data flow if the FPGA were installed in the system in the traditional way and Figure 8 indicating the data flow in a bump in the wire configuration [34]. Note that the benefit of the bump in the wire configuration is that data no longer need to flow across the PCIe bus to move from the FPGA to the network.

The FPGA manufacturer Xilinx maintains a set of of multi-purpose libraries with HLS implementations of various algorithms in the form of function primitives or fully implemented kernels.

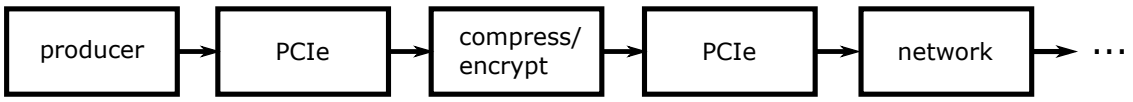


Figure 7: Example flow graph for FPGA accelerated compression/encryption using a traditional FPGA interconnection.

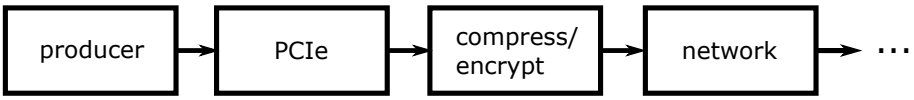


Figure 8: Example flow graph for FPGA accelerated compression/encryption using a bump in the wire configuration.

They are designed to get any developer up and running with implementations of algorithms ranging from image analysis, data analytics, and graph problems to name a few. Within the Vitis libraries are implementations of various data compression and cryptography libraries that we wish to investigate for a bump in the wire implementation. Within the two categories one can find a plethora of various compression and cryptography methods. Here, we have decided to target the LZ4 compression and AES cryptography algorithms which are ubiquitous in their respective spaces. The AES algorithm already exists as a streaming algorithm; no matter the size of the data target, it is broken into 128 bit blocks and each one is encrypted/decrypted in order. In the case of LZ4 textual compression/decompression, a target file or stream of data may need to be chunked and then run through the kernel in order for it to be considered streaming. In the Vitis libraries implementation, a streaming version of the LZ4 algorithm is implemented utilizing stream channels so data can be passed from one kernel to the next in a FIFO. It is important to note, of course, that the effectiveness of compression is dependent on the amount of repeated patterns in the target data and chunked data may reduce similarity for the overall dataset which in turn will reduce the effectiveness of compression.

In this application the amount of compression that the data will experience will effect how much data a downstream node will see until it is decompressed. To account for this in a network calculus model we will want to again normalize data in terms of the input but we want to make note of the possible compression ratio achieved by LZ4. Service curves after compression will then take two forms: one that considers the worst case scenario, a compression ratio of 1.0, and the other being the largest observed compression ratio. As these compression ratios effect how much data is truly going though the individual nodes, the lower bound service curve corresponds to a compression ratio of 1.0 and the maximum service curve will correspond to the maximum compression ratio. In addition, because the data is normalized to the input data volume, the throughput reported by the maximum service curve would be the baseline measured maximum service curve multiplied by the compression ratio which is then removed from downstream maximum service curves after decompression.

The target platform for this application is the Open Cloud Testbed (OCT) [14, 22] which deploys machines equipped with network capable Xilinx Alveo U280 FPGA cards, which can be targeted by Vitis implementations. Similar to the queuing theory model, we will test each stage in isolation and measure performance in isolation. Compression is implemented via a streaming LZ4 kernel and encryption is provided by a 256-bit CBC AES kernel, both available in the Vitis libraries. Finally the third kernel, the network communication kernel, is a demo implementation of a TCP stack and CMAC kernels that facilitate network communication between two FPGA cards [15, 24]. While Figure 8 illustrates the notion of the bump in the wire configuration, Figure 9 shows the flow graph of the application we actually model. The measured throughput for each stage is shown in Table 2.

The resulting simulation and network calculus model performance can be seen in Figure 10. Like in the previous model we combine all stages of the pipeline to create a single node for our network calculus model to determine latency and backlog bounds. Here we have removed the maximum service curve $\gamma(t)$ from the plot as it skews the overall graph and is indicative of the maximum observed throughput and also the maximum observed compression. Again we see the simulation

Figure 9: Actual flow graph for FPGA accelerated compression/encryption using the bump in the wire configuration.

Table 2: Listing of functions and their associated throughputs. The compression rates listed here are normalized with respect to their observed compression ratios: 2.2× Average, 1.0× Minimum, and 5.3× Maximum.

	Throughput					
Function	Average		Minimum		Maximum	
Compress	2662	MiB/s	1181	MiB/s	6386	MiB/s
Encrypt	68	MiB/s	56	MiB/s	75	MiB/s
Network	10	GiB/s	10	GiB/s	10	GiB/s
Decrypt	90	MiB/s	77	MiB/s	113	MiB/s
Decompress	1495	MiB/s	1426	MiB/s	1543	MiB/s
PCIe link	11	GiB/s	11	GiB/s	11	GiB/s

curve is below the potential maximum output bound for this system.

The quantitative predictions are shown in Table 3. As was the case for the BLAST application, the network calculus lower bound is just below the predicted performance of the discrete-event simulation. For both applications, the queueing theory predictions are somewhat optimistic (i.e., over-predict throughput), in part because they represent a simplified view of the system. For example, they model Markovian behaviour at each stage of the pipeline, a limitation that is not present in either the network calculus model or the discrete-event simulation model.

As was the case for BLAST, network calculus provides additional information that can be cleaned from the model, such as the following:

1. The maximum virtual delay, d, through the system is modeled to be 38 μs.

2. The maximum data occupancy resident in the entire system, x, (or backlog bound) is modeled as 3 KiB.

Points (1) and (2) above are corroborated by the discrete-event simulation model. For the maximum virtual delay, the longest observed delay in the simulator is 36.7 μs and the shortest delay being 25.7 μs. The maximum amount of data in system backlog accounting for all nodes and queues was observed to be 2 KiB. These values are within the bounds modeled using network calculus.

One important thing to note about the simulation is that it is not modeling the breaking of individual chunks of the encrypted AES output and sending them through the network node in individual packets. For ease of simulation we instead assume that data will be gathered at maximum in 1 KiB normalized chunks and then sent over the network. Another simulation shortfall is the lack of a structure to simulate overlapping stream channels which would be utilized in an FPGA deployment to transport data to downstream kernels. Furthermore we would like to corroborate these simulated results with an actual deployment. In the future we would like to show the power of network calculus as a tool to help make decisions about deployment when considering applications that have an arrival rate greater than what can be provided by the service of the system and when arrival rates need to be changed to accommodate queues that are at risk of overflowing.

6 Conclusions and Future Work

This paper has presented the use of network calculus models to bound the performance of streaming data applications executing on heterogeneous execution platforms. The models provide both upper

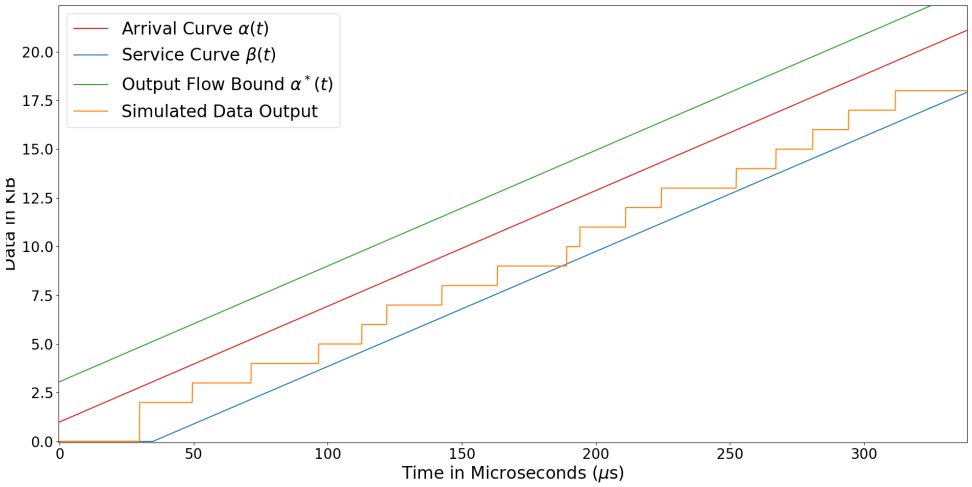


Figure 10: Network calculus model for our bump in the wire application.

Table 3: Bump in the wire streaming data application throughput.

Source	Value
Network calculus upper bound	313 MiB/s
Network calculus lower bound	59 MiB/s
Discrete-event simulation model [34]	61 MiB/s
Queueing theory prediction	151 MiB/s

bounds and lower bounds for throughput as well as latency and data volume. For a pair of example streaming applications, the bounds are tight enough to be helpful in understanding the performance implications of candidate design changes, and appropriately bracket a discrete-event simulation model of the same applications.

In future work, we would like to explore the use of these models on a wider set of streaming applications and validate the models over a wider range of empirical experiments. Further expanding the model we want to measure and quantify the effects of relaxing the constraint of $R_\alpha \leq R_\beta$ and the adoption of variable rate servers for arrival curves. By relaxing this first constraint we can begin to explore the possibilities of utilizing network calculus to guide the sizing and allocation of buffers within the application. Furthermore, utilizing variable rate arrival curves can introduce the concept of back pressure into the model, further increasing its utility.

Acknowledgments

This research was supported by the National Science Foundation under grant CNS-1763503 and a gift from BECS Technology, Inc.

References

[1] S. F. Altschul, T. L. Madden, A. A. Schäffer, J. Zhang, Z. Zhang, W. Miller, and D. J. Lipman. Gapped BLAST and PSI-BLAST: A new generation of protein database search programs. *Nucleic Acids Research*, 25:3389–402, 1997.

[2] S.F. Altschul, W. Gish, W. Miller, E.W. Myers, and D.J. Lipman. Basic local alignment search tool. *Journal of Molecular Biology*, 215(3):403–410, 1990.

[3] B. Avi-Itzhak and D. P. Heyman. Approximate Queuing Models for Multiprogramming Computer Systems. *Operations Research*, 21(6):1212–1230, December 1973.

[4] S. Azodolmolky et al. An analytical model for software defined networking: A network calculus-based approach. In *Proc. of Global Comm. Conf.*, pages 1397–1402. IEEE, 2013.

[5] Mohamed Bakhouya, S. Suboh, Jaafar Gaber, and T. El-Ghazawi. Analytical modeling and evaluation of on-chip interconnects using network calculus. In *Proc. of 3rd ACM/IEEE Int'l Symp. on Networks-on-Chip*, pages 74–79. IEEE, 2009.

[6] J.C. Beard and R.D. Chamberlain. Analysis of a simple approach to modeling performance for streaming data applications. In *Proc. of Int'l Symp. on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, pages 345–349. IEEE, August 2013.

[7] Christophe Bobda, Joel Mandebi Mbongue, Paul Chow, Mohammad Ewais, Naif Tarafdar, Juan Camilo Vega, Ken Eguro, Dirk Koch, Suranga Handagala, Miriam Leeser, Martin Herbordt, et al. The future of FPGA acceleration in datacenters and the cloud. *ACM Transactions on Reconfigurable Technology and Systems*, 15(3), 2022.

[8] Anthony M. Cabrera, Clayton J. Faber, Kyle Cepeda, Robert Derber, Cooper Epstein, Jason Zheng, Ron K. Cytron, and Roger D. Chamberlain. DIBS: A data integration benchmark suite. In *Proc. of ACM/SPIE Int'l Conf. on Performance Engineering Companion*, pages 25–28. ACM, April 2018.

[9] Stephen V. Cole and Jeremy Buhler. MERCATOR: a GPGPU framework for irregular streaming applications. In *Proc. of 15th Int'l Conf. on High Performance Computing and Simulation*, pages 727–736, July 2017.

[10] R.L. Cruz. A calculus for network delay. I. Network elements in isolation. *IEEE Trans. Inf. Theory*, 37(1):114–131, 1991.

[11] R.L. Cruz. A calculus for network delay. II. Network analysis. *IEEE Trans. Inf. Theory*, 37(1):132–141, 1991.

[12] C.J. Faber, T. Plano, S. Kodali, Z. Xiao, A. Dwaraki, J.D. Buhler, R.D. Chamberlain, and A.M. Cabrera. Platform agnostic streaming data application performance models. In *Proc. of IEEE/ACM Workshop on Redefining Scalability for Diversely Heterogeneous Architectures*, November 2021.

[13] Clayton J. Faber, Anthony M. Cabrera, Oronde Booker, Gabe Maayan, and Roger D. Chamberlain. Data integration tasks on heterogeneous systems using OpenCL. In *Proc. of 7th International Workshop on OpenCL (IWOCL)*, May 2019.

[14] Suranga Handagala, Martin Herbordt, and Miriam Leeser. OCT: The open cloud FPGA testbed. In *Proc. of 31st International Conference on Field Programmable Logic and Applications (FPL)*, 2021.

[15] Z. He, D. Korolija, and G. Alonso. Easynet: 100 Gbps network for HLS. In *Proc. of 31st International Conference on Field-Programmable Logic and Applications (FPL)*, pages 197–203. IEEE Computer Society, September 2021.

[16] Yuming Jiang. Network calculus and queueing theory: Two sides of one coin. In *Proc. of Int'l Conf. on Perf. Eval. Meth. and Tools*, 2010.

[17] Yuming Jiang and Yong Liu. *Stochastic Network Calculus*. Springer, 2008.

[18] Praveen Krishnamurthy, Jeremy Buhler, Roger Chamberlain, Mark Franklin, Kwame Gyang, Arpith Jacob, and Joseph Lancaster. Biosequence similarity search on the Mercury system. *Journal of VLSI Signal Processing Systems for Signal, Image, and Video Technology*, 49(1):101–121, 2007.

[19] Joshua Lant, Javier Navaridas, Mikel Luján, and John Goodacre. Toward FPGA-based HPC: Advancing interconnect technologies. *IEEE Micro*, 40(1):25–34, 2019.

[20] J.-Y. Le Boudec. Application of network calculus to guaranteed service networks. *IEEE Trans. Inf. Theory*, 44(3):1087–1096, 1998.

[21] J.-Y. Le Boudec and P. Thiran. *Network Calculus*. Springer, 2003.

[22] Miriam Leeser, Suranga Handagala, and Michael Zink. FPGAs in the cloud. *Computing in Science & Engineering*, 23(6):72–76, 2021.

[23] Meng Li, Guchuan Zhu, and Yvon Savaria. Delay bound analysis for heterogeneous multicore systems using network calculus. In *Proc. of 13th Conf. on Industrial Electronics and Applications*, pages 1825–1830. IEEE, 2018.

[24] Suranga Mahesh. TCP-network-demo. `https://github.com/OCT-FPGA/tcp-network-demo/`, 2022. Accessed Dec. 2023.

[25] Shobana Padmanabhan, Yixin Chen, and Roger D. Chamberlain. Optimal design-space exploration of streaming applications. In *Proc. of IEEE Int'l Conf. on Application-specific Systems, Architectures and Processors*, pages 227–230, September 2011.

[26] Krishna Pandit, J Schmittt, and Ralf Steinmetz. Network calculus meets queueing theory - a simulation based approach to bounded queues. In *Proc. of 12th Int'l Workshop on Quality of Service*, pages 114–120, 2004.

[27] Tom Plano and Jeremy Buhler. Scheduling irregular dataflow pipelines on SIMD architectures. In *Proc. of 6th Wkshp. on Programming Models for SIMD/Vector Processing*, pages 1:1–1:9, February 2020.

[28] Jens B. Schmitt and Utz Roedig. Sensor network calculus–a framework for worst case analysis. In *Proc. of Int'l Conf. on Distributed Computing in Sensor Systems*, pages 141–154. Springer, 2005.

[29] Team SimPy. SimPy: Discrete event simulation for Python. `https://simpy.readthedocs.io`, 2023. Accessed Aug. 2023.

[30] Lothar Thiele, Samarjit Chakraborty, and Martin Naedele. Real-time calculus for scheduling hard real-time systems. In *Proc. of Int'l Symp. on Circuits and Systems*, volume 4, pages 101–104. IEEE, 2000.

[31] Stephen Timcheck and Jeremy Buhler. Reducing queuing impact in streaming applications with irregular dataflow. *Parallel Computing*, 109:102863, March 2022.

[32] Amaury Van Bemten and Wolfgang Kellerer. Network calculus: A comprehensive guide. Technical Report 201603, Technical Univ. of Munich, 2016.

[33] Zenan Wang, Jiao Zhang, and Tao Huang. Determining delay bounds for a chain of virtual network functions using network calculus. *IEEE Communications Letters*, 25(8):2550–2553, 2021.

[34] Chenfeng Zhao, Clayton J Faber, Roger D Chamberlain, and Xuan Zhang. HLPerf: Demystifying the performance of HLS-based graph neural networks with dataflow architectures. *ACM Transactions on Reconfigurable Technology and Systems*, 2024.