Resource Optimization of Stream Processing in Layered Internet of Things

1st Anik Momtaz
Michigan State University
East Lansing, Michigan
USA
0000-0002-4739-1032

2nd Ramy Medhat
Google LLC
Waterloo, Ontario
Canada
0000-0002-9385-5862

3rd Borzoo Bonakdarpour
Michigan State University
East Lansing, Michigan
USA
0000-0003-1800-5419

***Abstract*—IoT (Internet of Things) applications often involve stream processing using multiple complex layers of processing nodes, where in each layer, data is received by the nodes, processed, and then transmitted to the nodes in subsequent layers. Such systems present a tradeoff between *reliability* and *resource usage*, including CPU power, energy, network bandwidth, memory, etc. Reducing the reliability at which a node processes inbound data in a layer can have repercussions on nodes in subsequent layers in the network that receive less reliable data, and in turn impact the reliability of the application as a whole. In this paper, we present a generalized model of streaming IoT applications as a *layered network of producers and consumers*. Our model captures trade-offs between reliability and resource usage of the system. We present an efficient algorithm using SMT constraint solvers to determine the optimal selection of processing quality for each node in the network, such that target system reliability is achieved while respecting the given resource bounds, and resource usage is minimized. In addition, we present a lightweight machine learning based solution to drastically improve our model in terms of run time. We have fully implemented our technique and report experimental results on a layered IoT network.**

I. Introduction

A. Motivation

Modern IoT devices typically involve physical components that are embedded with processing ability, software, and other hardware components that are able to connect and exchange data with other devices and systems over the Internet or other communication channels [1]. Applications of IoT often involve complicated stream processing tradeoffs. The rate of incoming data forces a tradeoff between the *infrastructure cost* (e.g., in terms of resource usage) and *quality*, as well as network *reliability*. High quality output, which could require more complex processing algorithms or a higher sampling rate, will commonly have an impact on power, energy, and bandwidth. In *layered IoT networks*, where nodes in a layer receive data from another layer, process it, and feed it into subsequent layer of nodes, the tradeoff becomes difficult to manage. This is magnified in systems with dynamically load shifts from one part of the network to another. In such systems, it is difficult to determine an optimal configuration that would cost effectively balance tradeoffs between resource usage and the quality levels. Fig. 1 demonstrates a layered network, where data flows from node v_{in} towards node v_{out}.

Most systems employ manual controls to manage tradeoffs. For instance, some IoT applications allow administrators to control the quality of malicious activity detection systems according to expected load [2], [3]. This approach often targets a specific subset of resources, and lacks the flexibility required by large dynamic systems. Another approach is to over-provision the processing infrastructure, network bandwidth, and allocated power budget such that no limits are hit [4], which is usually a very costly approach.

The main challenge in resource management and optimization stems from the fact that the nodes connected in a network often receive data, process it, and then transmit data to subsequent nodes. This leads to a quality vs. cost tradeoff among different nodes, where resources do not just depend on pairs of consecutive interacting devices, but on the interplay of the devices in the entire network. That is, reducing the processing quality of a node can have repercussions on subsequent nodes in the entire network that receive lower quality data. This means that optimization of quality versus resource usage has to be done for the entire network, and not just for pairs of nodes that are interacting with each other. On top of this challenge, it is straightforward to observe that quality and resource usage are often conflicting – a higher quality and reliability demands a higher resource usage, making optimization more difficult.

B. Our Contributions

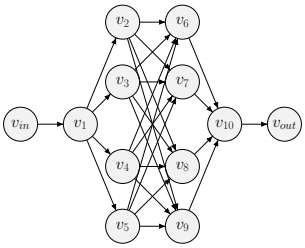


Fig. 1: Producer-consumer network.

Assuming a stream processing layered IoT network, each node in the network encounters a tradeoff between quality of processing and resource usage. In this paper, we introduce an abstract model of stream processing applications, where the processing nodes are modeled as a *network of producers-consumers*, which is a directed acyclic graph. Each node in the network is a consumer of data flowing through its incoming edges, and a producer of data flowing through its outgoing edges. Processing of data being consumed/produced by a node can be performed at variable *quality levels*. The quality level of processing dictates the amount of resources used by the node. The notions of a resource includes power, energy, RAM, disk, or network bandwidth, etc. In addition to these resources, we model *reliability* as a non-renewable resource that flows through the network and is partially depleted depending on the quality levels of the nodes it flows through. Nodes are subject to *resource constraints* and bounds that can be individual or collective. Lower quality results in higher error, which propagates through the network and can impact the quality of subsequent nodes, as well as the overall

reliability. We provide an efficient methodology to model a system, such that resource bounds are respected, and a chosen goal by the designer is optimized. This goal is accompanied by optimization objectives such as maximizing reliability or minimizing consumption of energy.

In order to solve this multi-objective optimization problem, we reduce the problem to a satisfiability modulo theory (SMT) problem. The SMT-solving technology has made tremendous progress in the past two decades [5], and we exploit its advances to solve our problem. To this end, we represent (1) the elements of the producers-consumers graph as well as the notions of data rates, quality, reliability, and resource consumption as SMT entities (e.g., variables, functions, constants, etc.), (2) the resource constraints and bounds as a set of SMT constraints, (3) the pillars of our original optimization problem as additional SMT constraints that will be checked and searched using a binary search algorithm to find the optimal answer, and (4) a machine learning based model that aims to even further optimize the problem in terms of execution time at the cost of a minimal loss in accuracy.

The SMT aspects of our technique is implemented using the SMT-solver Z3 [6] and the machine learning aspects of our technique is implemented using the machine learning toolkit Scikit-learn [7] and Keras [8] artificial neural network interface. Our model aims to optimize reliability and resource consumption trade-offs. We thoroughly explore these trade-offs through detailed synthetic experiments. We also apply our techniques on a real-world case study, where we optimize a network of embedded streaming devices, so that the network (1) delivers the best possible performance using the available resources, or it (2) uses the minimal amount of a certain resource while meeting a given performance goal.

II. Producer-Consumer Network

A. Model

A *producer-consumer network* is a directed acyclic graph (DAG) $G = (\mathbb{V}, \mathbb{E})$, where each vertex $v \in \mathbb{V}$ is a node that can be a producer, a consumer, or both, depending on its incoming/outgoing edges. A *producer* node only has outgoing edges, a *consumer* node only has incoming edges, and a producer/consumer node has both incoming and outgoing edges. Let $\mathsf{Pred}(v)$ denote the finite set of predecessor nodes from which v receives data, and $\mathsf{Succ}(v)$ denote the finite set of successor nodes which receive data from v. The set $\mathbb{E}$ of edges represented as ordered pairs of vertices such that:

$$\mathbb{E} = \Big\{(u, v) \mid v \in \mathsf{Succ}(u)\Big\}.$$

An edge from u to v represents a stream of items flowing from u to v, in which case u is a producer (potentially also a consumer) and v is a consumer. A node $v \in \mathbb{V}$, where $\mathsf{Pred}(v) = \emptyset$ is called a *source* and a node $u \in \mathbb{V}$, where $\mathsf{Succ}(u) = \emptyset$ is called a *sink*. Fig. 1 shows an example of a producer-consumer network. The network models a hierarchical monitoring system, where $v_{[1,9]}$ are producers of events that are consumed and manipulated by nodes $v_{[2,10]}$. Nodes v_{in} and v_{out} are placeholders for enforcing flow-based bounds.

A producer (resp., consumer) node $v \in \mathbb{V}$ may receive (resp., emit) data at a set of possible *input rates* $\mathsf{IRate}(v)$ (resp., *output rates* ORate_v). Let $\mathsf{Out}(u, v)$ denote the outgoing data rate from node u into node v. For example, in Fig. 1, the incoming data for v_1 is received from v_{in}, and the outgoing data is sent to v_2, v_3, v_4 and v_5. For every node $v \in \mathbb{V}$, we define $\mathsf{In}(v)$ such that,

$$\mathsf{In}(v) = \sum_{u \in \mathsf{Pred}(v)} \mathsf{Out}(u, v).$$

B. Resource Bounds

We first present the notions of *reusable* and *consumable* resources in our model. *Reusable resources* are not depleted when an item is processed. Examples of reusable resources are CPU, power, memory, network bandwidth, and quality. These resources are instantly reclaimed once an item is processed. We denote the finite set of reusable resources in the system as follows: $\mathbb{R} = \{R_1, R_2, \ldots, R_n\}$, for some $n \geq 1$. *Consumable resources* are depleted once an item is processed. Examples of consumable resources are energy, time, and reliability. For instance, once an error is encountered during the processing of an item along its path in the network, it cannot be reclaimed. We denote the finite set of consumable (depletable) resources as follows: $\mathbb{D} = \{D_1, D_2, \ldots, D_m\}$ for some $m \geq 1$.

Our model supports bounding resources on both nodes and edges. Let $G = (\mathbb{V}, \mathbb{E})$ be a producer-consumer network. A bound on a resource $res \in \mathbb{R} \cup \mathbb{D}$ for a subset of nodes $V \subseteq \mathbb{V}$ (resp., a subset of edges $E \subseteq \mathbb{E}$) is denoted by b_V^{res} (resp., b_E^{res}). We also set $b_V^{res} = \langle lb, ub \rangle$ (resp., $b_E^{res} = \langle lb, ub \rangle$) as a pair that implies the sum of resource $res \in \mathbb{R} \cup \mathbb{D}$ consumed by all nodes (resp., edges) in V (resp., E) must reside within the lower bound lb and the upper bound ub. Finally, we denote the set of all resource bounds for all resources in $\mathbb{R} \cup \mathbb{D}$ and for any subset of nodes and edges by $\mathbb{B}$. For instance, $b^{\text{PWR}}_{\{v_1, v_2, v_3\}} = \langle 0, 500 \rangle$ denotes that the total power consumption of nodes v_1, v_2 and v_3 should not exceed 500 watts.

Let μ_v^{res} (resp., μ_e^{res}) be the amount of resource $res \in \mathbb{R} \cup \mathbb{D}$ unit consumed by node $v \in \mathbb{V}$ (resp., edge $e \in \mathbb{E}$). Formally, a bound $b_V^{res} = \langle lb, ub \rangle$ on vertices $V \subseteq \mathbb{V}$ and a resource res enforces the following:

$$lb \leq \sum_{v \in V} \mu_v^{res} \leq ub.$$

Likewise, a bound $b_E^{res} = \langle lb, ub \rangle$ on edges $E \subseteq \mathbb{E}$ and a resource res enforces the following:

$$lb \leq \sum_{e \in E} \mu_e^{res} \leq ub.$$

C. Configurations

There are various configuration parameters that impact the resource usage of a node and the reliability of its output.

- *Sampling rate.* Some systems depend on sampling from continuous-time and continuous-valued signals and the amount of resources consumed by a node is proportional to the sampling rate [9]. Lower sampling rate is usually associated with reduced reliability or confidence. Hence, sampling rate is a configuration parameter that controls the tradeoff between resource usage and reliability.
- *Outgoing data rate.* The outgoing data rate of a node impacts the resource usage of subsequent nodes [9], [10]. If this data is sampled, then reliability is negatively impacted.
- *Precision.* Some algorithms support controllable precision. For instance, image processing may be accomplished with high or low precision [11]. Work in [12] demonstrates how configurable precision impacts accuracy and resource usage.

- *Algorithm alternatives.* In some systems, there are different algorithms that can be used to process data, with varying resource usage and reliability [13], [14]. For instance, data loss prevention systems employ different classifiers for malicious activity that have different processing costs [15].

To simplify our model, we abstract all the above parameters into a single *quality* symbol. Given a producer-consumer network $G = (\mathbb{V}, \mathbb{E})$, let us associate each node $v \in \mathbb{V}$ with a finite set of *quality levels*:

$$\mathbb{Q}_v = \Big\{ \mathsf{Qual}^1(v), \mathsf{Qual}^2(v), \ldots, \mathsf{Qual}^k(v) \Big\}$$

where the number of levels k can be different for each node v. A node v can use each quality level $\mathsf{Qual}^i(v)$, where $1 \leq i \leq k$ to process items that are being received at some input data rate in $\mathsf{IRate}(v)$, and being produced at some outgoing data rate in $\mathsf{ORate}(v)$. Part of our stream optimization (see Section III) is to find the best quality for the possible input/output data rates. To this end, for each node $v \in \mathbb{V}$, let

$$\vartheta_v : \mathsf{IRate}(v) \times \mathbb{Q}_v \rightarrow \mathsf{ORate}(v)$$

be a function that maps an incoming data rate and a quality level to an outgoing data rate. That is, we have:

$$\mathsf{ORate}_v = \vartheta\Big(\mathsf{IRate}(v), \mathsf{Qual}^i(v)\Big)$$

where $\mathsf{Qual}^i(v)$ is the i^{th} quality level of node v.

D. Monitor

We leverage a remote *network monitor* that is used to poll each node in the network at a fixed interval in order to keep track of available resources $res \in \mathbb{R} \cup \mathbb{D}$, as well as control the quality levels of the said node in real time. In this paper, we assume our monitor to be *fault tolerant* and *deadlock free*. Note that the monitor is a separate entity from the nodes. Therefore, the monitor's ability to optimize (and allocate) resources is not affected by the workload of the nodes.

E. Reliability

Quantifying reliability is generally a challenging task. Reliability of each node not only depends on its quality level, but on other *environmental factors* as well. For example, the reliability of a node that captures video streams may vary based on the time of the day and the surrounding lighting conditions. Let us assume each node $v \in \mathbb{V}$ is influenced by m_v number of environmental factors. We denote $\mathsf{U}_v^j \in [0, 1]$ where $1 \leq j \leq m_v$ as the j^{th} environmental factor of the node v. An environmental factor of 1 indicates the best possible reliability when other factors (as well as the quality) remain unchanged, whereas an environmental factor of 0 indicates the worst. All the intermediary values are determined by the node's architecture. In a similar manner, we denote $\mathsf{U}_v^{\mathsf{Qual}} \in [0, 1]$ as the *quality factor* of the node v. A quality factor of 1 maps to the highest quality level $\mathsf{Qual}^{\max}(v)$ supported by the implementation of the node's code. This could be a configuration where a computationally intensive algorithm is used, input data is not sampled or buffered, and numerical precision is set to the maximum supported precision. On the other hand, a quality factor of 0 maps to the lowest quality level $\mathsf{Qual}^{\min}(v)$ supported by the node. This should be a configuration below which the system becomes unusable. Quality factor for the remaining quality levels in $\mathbb{Q}_v - \{\mathsf{Qual}^{\max}(v), \mathsf{Qual}^{\min}(v)\}$ are determined by the system design. Now that we have defined the quality factor $\mathsf{U}_v^{\mathsf{Qual}}$ and the environmental factors $\mathsf{U}_v^1, \mathsf{U}_v^2, \ldots, \mathsf{U}_v^{m_v}$ for a node v, we are ready to define it's reliability $\alpha_v \in [0, 1]$ as follows:

$$\alpha_v = \frac{\mathsf{U}_v^{\mathsf{Qual}} + \mathsf{W}_v^1.\mathsf{U}_v^1 + \mathsf{W}_v^2.\mathsf{U}_v^2 + \ldots + \mathsf{W}_v^{m_v}.\mathsf{U}_v^{m_v}}{1 + \mathsf{W}_v^1 + \mathsf{W}_v^2 + \ldots + \mathsf{W}_v^{m_v}}$$

Where $\mathbb{W}_v = \{\mathsf{W}_v^1, \mathsf{W}_v^2, \ldots, \mathsf{W}_v^{m_v}\}$ are the respective weights of the environmental factors $\mathbb{U}_v = \{\mathsf{U}_v^1, \mathsf{U}_v^2, \ldots, \mathsf{U}_v^{m_v}\}$.

It is difficult to determine the discrete quality levels of a node, and then map them to numerical quality factor values. This is mainly because different nodes in a producer-consumer network carry out different tasks, and therefore, require their own methods for quality level determination. For example, the quality level of a node that is tasked with capturing and streaming video can be determined by its current video resolution. In other words, the maximum operational resolution can be considered as the highest quality and mapped to a quality factor of 1, and the minimum operational resolution can be considered as the lowest quality and mapped to a quality factor of 0. However, this method will clearly fail to determine the quality levels of a node that is tasked with detecting motion, where the polling interval rate could be a better representation of quality levels for the said node. In this paper, we do not attempt to provide methods for determining quality levels. We merely propose an abstraction that allows tweaking the system into yielding desirable results.

F. Relationship between Configurations and Resources

We now define the relationships between configurations and resources. Let CRate_{res} denote the set of possible rates of consumption of resource res. Also, let

$$\varphi_v^{res} : \mathsf{IRate}(v) \times \mathbb{Q}_v \rightarrow \mathsf{CRate}(res)$$

be a function that maps the rate of incoming data and the quality level of node $v \in \mathbb{V}$ to a possible consumption rate value in $\mathsf{CRate}(res)$. For example, for a node v with a quality levels of $\mathsf{Qual}(v)$ that is receiving data at the rate of $\mathsf{IRate}(v)$, we determine the rate at which resource PWR is consumed on the said node using $\varphi_v^{\text{PWR}}(\mathsf{IRate}(v), \mathsf{Qual}(v))$. Recall that resource res can be either reusable or non-reusable. Hence, each node defines a set of functions, in which the elements are functions φ_v^{res} for all resources $res \in \mathbb{R} \cup \mathbb{D}$ as follows:

$$\Phi_v = \Big\{ \varphi_v^{res} \mid res \in \mathbb{R} \cup \mathbb{D} - \{\text{REL}\} \Big\}$$

where REL is the reliability resource. All resources besides REL are strictly dependent on the corresponding nodes' quality levels. However, reliability of a node depends on the incoming data, as well as the node's quality level. Therefore, we exclude it from Φ_v, since it is subject to change even for the same node depending on the incoming data.

We incorporate this notion of reliability to model systems where error is compound, i.e., receiving erroneous data may impact the reliability of produced data differently even at the same quality level, and under the same environmental factors. This behavior is common in precision based quality levels, where rounding error is compounded as more mathematical operations are performed on a data path. Thus, we introduce the following recursive function ψ_v to determine the *compounded reliability* of node $v \in \mathbb{V}$ as follows:

	ORate			PWR ($Watt$)			TIME (s)		
	q^1	q^2	q^3	q^1	q^2	q^3	q^1	q^2	q^3
v_1	100	75	50	80	65	40	10	13.3	20
v_2	90	60	30	75	55	35	11.1	16.6	33.3
v_3	100	70	40	80	65	40	10	14.3	25
v_4	120	90	60	85	70	40	8.3	11.1	16.6
v_5	110	80	70	85	75	36	10.3	15.1	21.1

TABLE I: Nodes $v_{[1,5]}$ resource usage.

	PWR ($Watt$)			TIME (s)			α_v		
	q^1	q^2	q^3	q^1	q^2	q^3	q^1	q^2	q^3
v_6	85	70	55	16	20.1	25	100	90	82
v_7	80	65	50	18.2	18.2	38.3	100	92	84
v_8	88	55	50	15	17.7	29	100	88	79
v_9	90	70	55	13	14.5	19	100	93	83
v_{10}	100	80	60	17	22	45	–	–	–

TABLE II: Nodes $v_{[6,10]}$ resource usage.

$$\psi_v(\mathsf{Qual}(v)) =$$

$$\begin{cases} \mathsf{comp}(\mathsf{Qual}(v), \mathbb{U}_v, \mathbb{W}_v, \\ \quad \{\psi_u(\mathsf{Qual}(u)) \mid u \in \mathsf{Pred}(v)\}) \;\; \text{if} \;\; \mathsf{Pred}(v) \neq \emptyset \\ \alpha_v \;\; \text{if} \;\; \mathsf{Pred}(v) = \emptyset \end{cases}$$

where α_v is the reliability of v when it is a source node (by system design) and comp denotes a function that computes the reliability of a node given its quality, environmental factors and their weights, and the reliability of its predecessors. For instance, comp could be instantiated with a function that computes the average (or maximum) reliability of all predecessors times the quality level of the node.

For example, we introduce the characteristics of the nodes in the network of Fig. 1. Table I lists the production rate (ORate) for resources power (PWR) and response time (TIME) of nodes $v_{[1,5]}$ at different quality levels. We abbreviate the quality level Qual^i as q_i The quality level for nodes $v_{[1,5]}$ is designated by the sampling rate. Thus, q^1 which is the highest quality level has the highest rate of outgoing items, versus the lowest quality level q^3.

G. Revised Definitions

Based on the definitions introduced in the previous subsections, we now redefine a node as follows:

$$v = \Big\langle \mathsf{Pred}(v), \mathsf{Succ}(v), \mathsf{Qual}(v), \mathsf{ORate}(v), \Phi_v, \psi_v \Big\rangle$$

Thus, the node now includes a set of quality levels (i.e., $\mathsf{Qual}(v)$), a function that determines the rate of outgoing data (i.e., $\mathsf{ORate}(v)$), a set of functions that determine resource usage (i.e., Φ_v), and a function that determines the reliability of the node's output (i.e., ψ_v).

Finally, we redefine the graph as follows:

$$G = \Big\langle \mathbb{V}, \mathbb{E}, \mathbb{R}, \mathbb{D}, \mathbb{B} \Big\rangle$$

Thus, the graph now defines a set $\mathbb{D}$ of consumable resources, a set $\mathbb{R}$ of reusable resources, and a set $\mathbb{B}$ of bounds on resources.

III. Formal Statement of the Problem

First, observe that in the model proposed in Section II, quality levels may affect the following:

1) Node reliability ψ_v, which is a function of the quality level, environmental factors and their weights, and the incoming reliability values of all predecessors.
2) Resource consumption φ_v^{res}, which is a function of the quality level and the incoming data rate.
3) Production rate $\mathsf{ORate}(v)$, which is a function of the quality level and the incoming data rate.

The majority of the stream processing systems benefit greatly from knowing how to answer one or both of these two questions; (1) how the usage of available resources can be optimized to reach maximum reliability, and (2) how to minimize available resource usage while ensuring reliability is maintained above a target threshold. Thus, roughly speaking, our *multi-objective* problem statement for this paper is as follows. Given (1) a producer consumer network on which a set of bounds is defined, and (2) a *target reliability* for all consumer-only nodes,

- *Quality Maximization.* Our first objective is to identify a single quality level for every node, such that the reliability for consumer-only nodes, is maximized, while satisfying all bounds. For example, maximizing the efficiency of each device in a network of smart home devices while not exceeding the specified renewable resources like power, CPU, bandwidth etc.
- *Resource Usage Minimization.* Our second objective is to minimize consumption of a given resource for all nodes while achieving a target reliability. For example, minimizing the power usage of a producer-consumer network, that does not demand maximum reliability from its nodes.

Formally, our optimization problem is as follows:

Problem Statement

Given a producer-consumer network $G = \langle \mathbb{V}, \mathbb{E}, \mathbb{R}, \mathbb{D}, \mathbb{B} \rangle$ and a resource $res \in \mathbb{R} \cup \mathbb{D}$, identify quality levels $\mathsf{Qual}(u)$ for all $u \in \mathbb{V}$ such that:

$$\forall v \in \{v' \mid \mathsf{Succ}(v') = \emptyset\}. \max \Big(\psi_v\big(\mathsf{Qual}(v)\big)\Big)$$

and

$$\forall v \in \mathbb{V}. \min \Big(\varphi_v^{res}\big(\mathsf{IRate}(v), \mathsf{Qual}(v)\big)\Big)$$

In the next section, we will present our solution to solve the above optimization problem.

IV. SMT-based Solution

In this section, we present our solution to solve the multi-objective optimization problem presented in Section III. Our solution is based on a reduction to solving the satisfiability problem for SMT. Practically, we utilize an SMT-solver in order to optimize reliability and resource consumption tradeoffs. Each SMT instance is described in terms of (1) SMT entities (e.g., variables, functions, constants) and (2) SMT constraints (e.g., Boolean conditions over first-order predicates).

A. SMT Entities

We now introduce the entities used to represent the components of our producer-consumer network $G = (\mathbb{V}, \mathbb{E})$. In some SMT entity definitions, we use *free variables* that the SMT solver can manipulate to provide a satisfaction verdict.

Nodes. In our SMT encoding, we represent the set of nodes $\mathbb{V}$ as a set of integers $\{1, 2, \cdots, |\mathbb{V}|\}$, where each element represents a node in $\mathbb{V}$.

Edges. We store the information of edges in $\mathbb{E}$ in the form of a $|\mathbb{V}| \times |\mathbb{V}|$ Boolean array edge such that:

$$\bigwedge_{i=1}^{|\mathbb{V}|}\bigwedge_{j=1}^{|\mathbb{V}|} \mathsf{edge}[i][j] = \begin{cases} \texttt{true} & \text{if} \quad (i,j) \in \mathbb{E} \\ \texttt{false} & \text{if} \quad (i,j) \notin \mathbb{E} \end{cases}$$

where $\mathsf{edge}[i][j]$ implies there exists an edge from node v_i to node v_j in G.

Successor Nodes. We encode the function Succ as an SMT function succ that maps a node to a set of successor nodes as follows:

$$\bigwedge_{i=1}^{|\mathbb{V}|} \mathsf{succ}(i) = \Big\{ j \mid (i,j) \in \mathbb{E} \Big\}$$

Predecessor Nodes. We encode the function Pred as an SMT function pred that maps a node to a set of predecessor nodes as follows:

$$\bigwedge_{i=1}^{|\mathbb{V}|} \mathsf{pred}(i) = \{ j \mid (j,i) \in \mathbb{E} \}$$

Node Resource Consumption. We define the function rcon that maps a resource and a node to a free variable that denotes the resource consumption of the said node as follows:

$$\bigwedge_{res \in \mathbb{R} \cup \mathbb{D}} \bigwedge_{i=1}^{|\mathbb{V}|} \mathsf{rcon}(res, i) = \mu_i^{res}$$

Edge Resource Consumption. We define the function rcoe that maps a resource and an edge to a free variable that denotes the resource consumption of the said edge as follows:

$$\bigwedge_{res \in \mathbb{R} \cup \mathbb{D}} \bigwedge_{(i,j) \in \mathbb{E}} \mathsf{rcoe}(res, (i,j)) = \mu_{(i,j)}^{res}$$

Resource Inflow. We define the function iflo that maps a resource and a node to a free variable that denotes the inflow of resource to the said node as follows:

$$\bigwedge_{res \in \mathbb{R} \cup \mathbb{D}} \bigwedge_{i=1}^{|\mathbb{V}|} \mathsf{iflo}(res, i) = \zeta_i^{res},$$

where ζ_v^{res} denotes the inflow of resource res into node v

Resource Outflow. We define the function oflo that maps a resource and a node to a free variable that denotes the outflow of resource from the said node as follows:

$$\bigwedge_{res \in \mathbb{R} \cup \mathbb{D}} \bigwedge_{i=1}^{|\mathbb{V}|} \mathsf{oflo}(res, i) = \xi_i^{res},$$

where ξ_v^{res} denotes the outflow of resource res from node v.

Edge Flow. We define the function eflo that maps a resource and an edge to a free variable that denotes the amount of flow going through the said edge as follows:

$$\bigwedge_{res \in \mathbb{R} \cup \mathbb{D}} \bigwedge_{(i,j) \in \mathbb{E}} \mathsf{eflo}(res, (i,j)) = \nu_{(i,j)}^{res}$$

where $\nu_{(i,j)}^{res}$ denotes the flow of resource res through edge (i,j).

Quality. We encode the function Qual as an SMT function qual that maps a node to all of its abstracted quality levels in disjunction (see Section II) as follows:

$$\bigwedge_{i=1}^{|\mathbb{V}|} \big(\mathsf{qual}(i) = \bigvee_{j=1}^{|\mathbb{Q}|} \mathsf{Qual}^j(v_i)\big)$$

Reliability. We encode the function ψ as an SMT function rel that maps the quality of a node to its reliability as follows:

$$\bigwedge_{i=1}^{|\mathbb{V}|} \mathsf{rel}(\mathsf{qual}(i)) =$$

$$\begin{cases} \mathsf{eval}(\mathsf{qual}(i), \mathbb{U}_v, \mathbb{W}_v, \\ \qquad \{\mathsf{rel}(\mathsf{qual}(j)) \mid j \in \mathsf{pred}(i)\}) & \text{if} \quad \mathsf{pred}(i) \neq \emptyset \\ \alpha_i & \text{if} \quad \mathsf{pred}(i) = \emptyset \end{cases}$$

Note that when $\mathsf{pred}(i) = \emptyset$, node i is the source (producer-only) node in the producer-consumer network, and therefore, its reliability, α_i is known.

B. SMT Constraints

We now introduce the constraints that address our problem statement using the SMT entities we defined in the previous section.

Resource Inflow Constraint. For resources $res \in \mathbb{R} \cup \mathbb{D}$, the amount of a resource flowing into a node depends on the amount of flow carried over all its incoming edges. Traditionally, the inflow is the sum of all flows on incoming edges. That is,

$$\bigwedge_{res \in \mathbb{R} \cup \mathbb{D}} \bigwedge_{i=1}^{|\mathbb{V}|} \zeta_i^{res} = \sum \big\{ \mathsf{eflo}(res, (i,j)) \mid (i,j) \in \{(v', v) \mid v' \in \mathsf{pred}(v)\} \big\}$$

For instance, power is a resource that can be summed over incoming edges.

Resource Outflow Constraint. For resources in $res \in \mathbb{R} \cup \mathbb{D}$, the amount of a resource flowing out from a node is traditionally equal to the amount of the resource flowing in, which is the conservation of flow principle. This models renewable resources efficiently, yet does not capture non-renewable resources. We generalize the resource outflow constraint using function Ξ:

$$\bigwedge_{res \in \mathbb{R} \cup \mathbb{D}} \bigwedge_{i=1}^{|\mathbb{V}|} \mathsf{oflo}(res, i) = \Xi_i^r \left(\mathsf{iflo}(res, i), \mathsf{rcon}(r, i)\right)$$

For instance, power is a renewable resource, and thus,

$$\bigwedge_{i=1}^{|\mathbb{V}|} \Xi_i^{\text{PWR}} \left(\mathsf{iflo}(\text{PWR}, i), \mathsf{rcon}(\text{PWR}, i)\right) = \mathsf{iflo}(\text{PWR}, i)$$

However, energy is depletable, and therefore,

$$\bigwedge_{i=1}^{|\mathbb{V}|} \Xi_i^{\text{EGY}}\left(\mathsf{iflo}(\text{EGY}, i), \mathsf{rcon}(\text{EGY}, i)\right) = \mathsf{iflo}(\text{EGY}, i) - \mathsf{rcon}(\text{EGY}, i)$$

Resource Bound Constraint on Nodes. For all resources $res \in \mathbb{R} \cup \mathbb{D}$, we enforce the given upper bound and lower bound on nodes as follows:

$$\bigwedge_{res \in \mathbb{R} \cup \mathbb{D}} \Big(lb \leq \sum_{i=1}^{|\mathbb{V}|} \big\{ \mathsf{rcon}(res, i) \big\} \leq ub \Big)$$

Resource Bound Constraint on Edges. We use bounds on edges to control the distribution of resources $res \in \mathbb{R} \cup \mathbb{D}$ across outgoing edges of a node. We identify two main methods of assigning flows to outgoing edges: *broadcast* and *distribution* resources.

a) Broadcast: In this case, nodes broadcast their outflow to outgoing edges. Reliability is broadcast, since all outgoing edges of a node carry data with the same reliability level that the node produces. We can enforce resources $res \in \mathbb{R} \cup \mathbb{D}$ to be broadcast using the following constraint:

$$\bigwedge_{res \in \mathbb{R} \cup \mathbb{D}} \bigwedge_{(i,j) \in \mathbb{E}} \Big(\mathsf{oflo}(res, i) \leq \mathsf{rcoe}(res, (i, j)) \leq \mathsf{oflo}(res, i) \Big)$$

Upon simplification, we have:

$$\bigwedge_{res \in \mathbb{R} \cup \mathbb{D}} \bigwedge_{(i,j) \in \mathbb{E}} \mathsf{rcoe}(res, (i, j)) = \mathsf{oflo}(res, i)$$

b) Distribution: In this case, the outflow is distributed across all outgoing edges. For instance, in a multiple consumer setting any one of a set of receiving nodes can process items. In this case, the outgoing data flow of the producer is distributed among all consumer nodes. The objective here is to determine the fraction of data flowing to each consumer such that resource bounds are respected and the usage of a specific resource is optimized. We can enforce resources $res \in \mathbb{R} \cup \mathbb{D}$ to be distributed using the following constraint:

$$\bigwedge_{res \in \mathbb{R} \cup \mathbb{D}} \bigwedge_{i=1}^{|\mathbb{V}|} \big(\mathsf{oflo}(res, i) \leq \sum_{j \in \mathsf{succ}(i)} \big\{ \mathsf{rcoe}(res, (i, j)) \big\} \leq \mathsf{oflo}(res, i) \big)$$

Upon simplification, we have:

$$\bigwedge_{res \in \mathbb{R} \cup \mathbb{D}} \bigwedge_{i=1}^{|\mathbb{V}|} \sum_{j \in \mathsf{succ}(i)} \big\{ \mathsf{rcoe}(res, (i, j)) \big\} = \mathsf{oflo}(res, i)$$

Data Flow Constraint. Data outflow can be either broadcast or distributed. We encode the function Out as an SMT function out that maps an edge to outdoing data rate of that edge.

c) Broadcast: In case the data outflow is broadcast, we enforce the following constraint on all edges:

$$\bigwedge_{(i,j) \in \mathbb{E}} \mathsf{out}((i, j)) = \mathsf{ORate}(v_i)$$

d) Distribution: In case the data outflow is distributed, we enforce the following constraint on all edges:

$$\bigwedge_{i=1}^{|\mathbb{V}|} \sum_{j \in \mathsf{succ}(i)} \big\{ \mathsf{out}((i, j)) \big\} = \mathsf{ORate}(v_i)$$

Reliability Maximization Constraint. Finally, let C denote the conjunction of all the above constraints. The constraint for maximizing reliability on sink (consumer only) nodes is as follows:

$$C \wedge \big(\bigwedge_{i \in \{j \mid \mathsf{succ}(i) = \emptyset\}} \max(\mathsf{rel}(\mathsf{qual}(i))) \big)$$

Resource Optimization Constraint. If we want to minimize the total consumption of some $res \in \mathbb{R} \cup \mathbb{D}$ across all nodes, while ensuring the reliability on sink (consumer only) nodes remain above a given threshold α, then we enforce the following constraint instead:

$$C \wedge \Big(\bigwedge_{i \in \{j \mid \mathsf{succ}(i) = \emptyset\}} \mathsf{rel}(\mathsf{qual}(i)) \geq \alpha \Big) \wedge \min\Big(\sum_{i=1}^{|\mathbb{V}|} \big\{ \mathsf{rcon}(res, i) \big\} \Big)$$

C. Solver Optimization

Solving the Reliability Maximizing Constraint and the Resource Optimization Constraint both require a significant amount of computation power and time (see Fig. 2a). This is mostly because C is a conjunction of a large set of constraints, coupled with the fact that it is a minimization or maximization problem. This means, there is only one solution for which the value of the object is maximized or minimized, which in turn means that our SMT solver having to explore a large search space.

To this end, we employ some optimization techniques in order to reduce run time for the SMT solver. In this subsection, we show one such technique and report the improvement it shows in terms of run time over the naive method.

Binary Probing First, let A_α be an SMT constraint such that,

$$A_\alpha = C \wedge \big(\bigwedge_{i \in \{j \mid \mathsf{succ}(i) = \emptyset\}} \mathsf{rel}(\mathsf{qual}(i)) \geq \alpha \big)$$

We say $A_\alpha \models G$ iff A_α is satisfied for α, otherwise $A_\alpha \not\models G$. Let solve be a function that, given G and A_α, returns some value β such that $\alpha \leq \beta \leq 1$ and $A_\alpha \models G$, otherwise $\beta = -1$. Formally,

$$\mathsf{solve}(A_\alpha, G) \begin{cases} \beta & \text{if} \quad A_\alpha \models G \wedge \beta \in [\alpha, 1] \\ -1 & \text{otherwise} \end{cases}$$

Algorithm 1: Best Reliability Estimation Algorithm

Data: Producer-consumer network G, SMT constraint A_α, Target reliability α, Error margin ϵ
Result: Estimated Best Reliability α'

$\alpha_{min} \leftarrow 0$
$\alpha_{max} \leftarrow 1$
$pivot \leftarrow .5$
$\alpha' \leftarrow 0$

$found \leftarrow$ `false`
while $\neg found$ **do**
  if $|\alpha_{max} - \alpha_{min}| \leq \epsilon$ **then**
    $found \leftarrow$ `true`
  $\beta \leftarrow \mathsf{solve}(A_{pivot}, G)$
  if $\beta \neq -1$ **then**
    if $\beta > \alpha'$ **then**
      $\alpha' \leftarrow \beta$
    $\alpha_{min} \leftarrow \lfloor pivot \rfloor$
  else
    $\alpha_{max} \leftarrow \lfloor pivot \rfloor$
  if $found$ **then**
    break
  $pivot \leftarrow (\alpha_{min} + \alpha_{max})/2$
return α'

Now, using a *Binary Probing* technique described in Algorithm 1, we can invoke our SMT solver in a pattern similar to the traditional binary search, and find an estimated α', that is sufficiently close, that is, within the error margin ϵ of the *real* best reliability. Using a similar technique to this, we can also find the estimated minimum of any $res \in \mathbb{R} \cup \mathbb{D}$. It should be mentioned that even in worst case, after just five SMT invocations, the error from binary probing will only be $\approx 3.125\%$, which can be reduced even further at the cost of more SMT invocations.

V. Machine Learning-based Optimization

On top of our SMT-based solution described in Section IV, we employ a machine learning-based optimization technique to further improve our solution in terms of run time at the cost of negligible (details in the next section) accuracy.

A. Artificial Neural Network

We first create an *Artificial Neural Network* [16] (ANN) where neurons in the output layer denote the resources that need to be optimized, and the neurons in the input layer denote the remaining resources. For example, when solving the Quality Maximization problem, each neuron in the output layer represents each quality level of each node $v \in \mathbb{V}$, that is, the number of neurons in the output layer is $l_o = |\mathbb{V}|$. Each neuron in the input layer represents remaining resources like power, CPU, memory, bandwidth etc., that is $l_i = |\mathbb{R} \cup \mathbb{D} - \{\text{QUAL}\}|$, where QUAL $\in \mathbb{R}$ is the quality resource. For determining the number of neurons in the hidden layer, we chose the method proposed by the authors in [17], that is, the number of neurons in the hidden layer is,

$$l_h = \left(\frac{2}{3} \times |\mathbb{R} \cup \mathbb{D} - \{\text{QUAL}\}|\right) + |\mathbb{V}|$$

As an example, let us consider the producer-consumer network in Fig. 1. If we want to maximize the quality of the network with respect to power (PWR), CPU (CPU), memory (MEM) and bandwidth (BW), then the corresponding ANN should be of the form where $l_o = 10$, $l_i = 4$, and $l_h = 13$.

B. Training Dataset

Our training dataset for the ANN is generated using the SMT-based solution detailed in Section IV. For example, in order to generate the training data-set for the Quality Maximization problem, we find the best qualities for all nodes for resources with random values, and populate the data-set with the results. This allows us to carry out machine learning process in an *unsupervised* manner.

During training, while the traditional approach is to split the dataset into two subsets (i.e. training dataset and testing dataset), for smaller datasets, this may introduce biased estimates [18]. As our model must be applicable to both small and large datasets, in order to reduce statistical bias, we employ *k-fold cross validation* [19] to train our ANN.

Using k-fold cross validation ensures that each group is used as the testing dataset once, and as the training dataset $k-1$ times. There various ways of selecting the value of k. However, in our work, we assign $k = 10$, as this is shown to generally yield minimal statistical bias [20], [21]. Note that for generating our dataset, we normalize the sample values to avoid unwanted weights. However, when we report the experimental results, we use the actual values for ease of comparison and understandability.

C. Model Accuracy

We determine the accuracy of our trained model by directly comparing its results with the results from SMT-based solution. For the Quality Maximization problem, we define the accuracy, acc_q of the model as follows:

$$acc_q = 1 - \frac{\sum_{v \in \mathbb{V}} |SMT_{q_v} - ML_{q_v}|}{\sum_{v \in \mathbb{V}} |\mathbb{Q}_v|}$$

Where the quality level reported by the SMT-based solution is the $SMT^{th}_{q_v}$ quality level of node v, the quality level reported by the machine learning model is the $ML^{th}_{q_v}$ quality level of node v, and $\mathbb{Q}_v$ is the set of quality levels of node v. For the Resource Minimization problem, we define the accuracy acc_r of the model as follows:

$$acc_r = \frac{\sum_{res \in \mathbb{R} \cup \mathbb{D}} \frac{|SMT_{res_v} - ML_{res_v}|}{MAX_{res_v} - MIN_{res_v}}}{|\mathbb{R} \cup \mathbb{D}|}$$

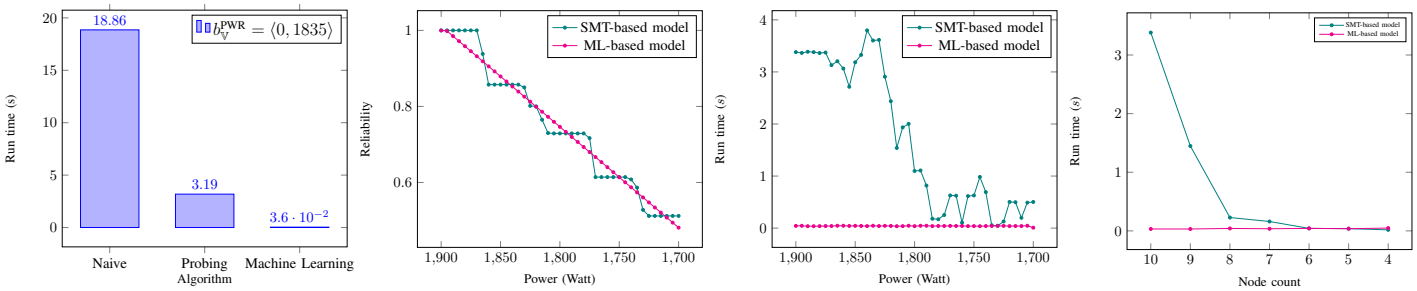
Where SMT_{res_v} is the value of resource res of node v reported by the SMT-based solution, ML_{res_v} is the value of resource res of node v reported by the machine learning model, and MAX_{res_v} (resp., MIN_{res_v}) denotes the upper bound (resp., lower bound) of resource res observed in the dataset. Note that, $0 \leq \{acc_q, acc_m\} \leq 1$, where 0 indicates the worst accuracy, and 1 indicates the best accuracy.

VI. Case Studies and Evaluation

In this section, we evaluate our technique for resource optimization using synthetic data generated from a *simulated* layered network of nodes, as well as real world data collected from a network of embedded devices, where the nodes in the network are Raspberry Pi devices tasked with specific streaming objectives.

A. Synthetic Experiments

In this subsection, we introduce our synthetic experiments to demonstrate how our proposed model can be used to optimize various resources.



(a) Naive vs. optimized vs. ML. (b) Reliability vs. power. (c) Run time vs. power. (d) Run time vs. node availability.

Fig. 2: Synthetic experiment results.

	q_1	q_2	q_3	q_4	q_5
v_{in}	-	-	-	-	-
v_1	200	195	190	185	180
v_2	185	180	175	170	165
v_3	190	185	180	175	170
v_4	195	190	185	180	175
v_5	180	175	170	165	160
v_6	195	190	185	180	175
v_7	185	180	175	170	165
v_8	180	175	170	165	160
v_9	190	185	180	175	170
v_{10}	200	195	190	185	180
v_{out}	-	-	-	-	-

Fig. 3: Node power consumption for different quality levels.

1) Experimental Setup: We construct our producer-consumer network using 10 nodes, $\mathbb{V} = \{v_1, v_2, \cdots, v_{10}\}$, with $v_{[1,9]}$ being the producer nodes, and $v_{[2,10]}$ being the consumer nodes. We add two placeholder nodes to the network, v_{in} and v_{out}, along with two edges (v_{in}, v_1) and (v_{10}, v_{out}). Fig. 1 shows our producer-consumer network. We use edge (v_{in}, v_1) to regulate bounds on resources, and we use node v_{out} to compute network reliability.

2) Resource Bounds: In this experiment, we consider the two resources power (PWR) and reliability (REL). Where PWR $\in \mathbb{R}$ and REL $\in \mathbb{D}$. We assign three possible quality levels to each node. Table 3 shows different power consumption values for all possible quality levels in each node.

3) Machine Learning Setup: For our dataset, we generate 500 data samples using our SMT-based solution. each sample contains a randomly selected PWR value, and the optimized quality levels for the 10 nodes in Fig. 1. We train our ANN with this dataset for 10 epochs (full iterations) using the k-fold cross validation method described in Section V.

4) Experimental Results: We run a variety of experiments on the setup described above and report our findings below.

Naive Model vs. Optimized Model. vs. Machine Learning Model First we run the model to find the best possible reliability, given bounds on other resources in the producer-consumer network. We want to observe the improvement in run time between a model that performs regular constraint solving, and a model that performs constraint solving using the binary probing technique shown in Algorithm 1. To this end, we assign the PWR resource $b_{\mathbb{V}}^{\text{PWR}}$ to $\langle 0, 1835 \rangle$ and run our solvers. As shown in Fig. 2a, using the naive (brute force) technique, we get a best reliability value of 0.855 within a run time of 18.855 seconds, whereas using the binary probing technique, we get a best reliability value of 0.857 within a run time of 3.185 seconds. Using our machine learning model, we get a best reliability value of 0.839 in under 0.036 seconds.

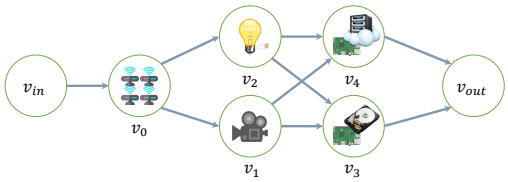


Fig. 4: A Multi-Layer Network of Raspberry Pi Devices

Reliability vs. Power. We now observe the tradeoff between reliability and power. We start off by assigning the PWR resource $b_{\mathbb{V}}^{\text{PWR}}$ to $\langle 0, 1900 \rangle$. We can observe in Fig. 2b the burndown of reliability as we tighten the power bound by 5 watts on each iteration. We stop at $\langle 0, 1700 \rangle$ when the network is no longer functional due to the given power being lower than the minimal requirement. In this scenario, our machine learning-based model report a more uniform reliability drop than our SMT-based model. Fig. 2c demonstrates the run time measurements for the same experiment. As the available power is reduced, the search space for the SMT solver gets smaller as well due to having to check for fewer valid configurations. Which is why a gradual decline in run time can be observed for the SMT-based model. However, the machine learning-based model reports the results through inference, and therefore show very little variation in run time.

Run time vs. Node availability. In this experiment, we observe the effect of node availability, and by extension overall reliability on run time. To this end, we assign the PWR resource $b_{\mathbb{V}}^{\text{PWR}}$ to $\langle 0, 1835 \rangle$ and reduce reliability of the nodes v_5, v_2, v_3, v_8, v_7 and v_9 to 0 one node at a time in the given order. Fig. 2d shows the run time of this experiment. As expected, as more nodes become inactive, the overall run time for the solver decreases for the SMT-based model. However, similar to the previous observation, for the ML-based model, the run time remains steady. Note that removing any more nodes from the network will render the network inactive, as the solver will fail to find any valid path from v_{in} to v_{out}.

B. Case Study

We now introduce our case study on a real world layered network, where each node is tasked with a streaming or processing job, and is operated with a Raspberry Pi device.

1) Experimental Setup: We construct our layered network with five nodes as shown in Fig. 4, where v_0, v_1 and v_2 are producer nodes, and v_1, v_2, v_3 and v_4 are consumer nodes in $\mathbb{V}$. Just like before, we add two place holder nodes v_{in} and

v_{out} such that, v_{in} has an outgoing edge to v_0, and v_{out} has two incoming edges from v_3 and v_4. Below we explain the streaming tasks of each device, along with the resources and quality levels.

Motion Sensor Node (v_0). This node is comprised of four motion sensors that are able to detect objects and movement. When any one of these motion sensors are activated, v_0 sends an activation signal to its subsequent nodes. Table IIIa shows resource consumption for node v_0 under different quality levels. In this case, the quality levels simply indicate the number of active motion sensors. Motion sensors are fairly reliable under normal operational circumstances, which is why v_0 has high reliability as long as at least one sensor is active.

Camera Node (v_1). This node is tasked with operating a 5MP camera at varying resolution/bitrate, and is activated upon receiving an activation signal from v_0. The resolution/bitrate of the captured stream is changed at runtime, which allows us to map the different resolution/bitrate modes to the quality level of node v_1. The node consumes two other resources: bandwidth and power. At the highest quality level, this node uses ≈ 2.5 Mbit/s and ≈ 566 mAh. Table IIIb shows resource consumption for node v_1 under different quality levels, and Fig. 5a shows the tradeoff between stream resolution vs. power consumption and bandwidth usage. Note that change stream resolution/bitrate is not always proportional to bandwidth. This is due to video compression standards (e.g H.264).

Illuminance Detection Node (v_2). This node is connected to an illuminance detector, and a smart light bulb with adjustable brightness. As both of these devices are powered and operated by node v_2 with minimal data transfer and delay, we consider them to be a part of node v_2 itself. Similar to v_1, this node is activated upon receiving an activation signal from v_0. Depending on how dark or bright the general area that is being streamed by the camera on node v_1 is, node v_2 adjusts the brightness of the smart bulb accordingly. We keep the illuminance detector some distance away from the network, in order to prevent light flickering due to feedback loop. Table IIIc shows resource consumption for node v_1 under different quality levels.

Local Storage Node (v_3). This node compresses, checksum verifies, and stores the video stream received through edge (v_1, v_3) into a secure external hard disk drive. Table IIId shows resource consumption for v_1 under different quality levels.

Cloud Storage Node (v_4). Similar to node v_3, node v_4 is also tasked with storing the video stream received through edge (v_1, v_3). However, instead of storing it locally, the stream is uploaded directly to a cloud storage. Offloading the compression and checksum verification task to the cloud allows node v_4 to consume minimal power, at the cost of additional bandwidth. Table IIIe shows resource consumption for node v_4 under different quality levels. Note that the bandwidth requirement for node v_4 is double the amount in comparison to that of v_3 at the same quality level. This is due to the fact that when v_4 receives a video stream from node v_1, it uploads the same stream to the cloud, effectively doubling the required bandwidth. Furthermore, we assume that while maintaining similar qualities and stream, node v_4 generally more reliable than node v_3 for being able to store and backup data in the cloud as shown in Fig. 5b.

2) Machine Learning Setup: For our machine learning dataset, similar to our synthetic experiments, we generate 500 data samples using our SMT-based solution. each sample contains a randomly selected PWR value, and the optimized quality levels for the 5 nodes in the Pi network shown in Fig. 4. We train our ANN with this dataset for 10 epochs (full iterations) using the k-fold cross validation method described in Section V.

3) Experimental Results: We run a variety of experiments on the setup described above and report our findings below.

Reliability vs. Power. We now observe the tradeoff between reliability and power in our multi-layer network of nodes. We start off by assigning the PWR resource $b_{\mathbb{V}}^{\text{PWR}}$ to $\langle 0, 1410 \rangle$ as shown in Fig. 5c. From this point we measure the best obtainable reliability and tighten the bound by 5 mAh on each iteration in a similar manner as our synthetic experiment. Fig. 5d shows the run time for the same experiment. The run time is low at the beginning due to the system having adequate power flow to operate all nodes at a near maximum reliability, and therefore having very few SMT constraints to solve. Observe that our machine learning-based model exhibits similar behavior as seen during our synthetic experiments. The reliability drop is a steady decline, whereas the run time does not very to a great degree. However, for both models, this changes when $b_{\mathbb{V}}^{\text{PWR}}$ is $\langle 0, 1410 \rangle$ and onward, as the available power is no longer sufficient for all nodes to operate at maximum quality and reliability.

We now observe the changes in quality levels for which the burndown in reliability has occurred. From Table IIIf, we can see that the model gradually lowered the quality levels of v_0 to q_4 first due to the small difference in reliability between each quality level. Afterwards, the quality level of v_3 (local node) is lowered instead of v_4 (cloud node) due to v_3 consuming more power than v_4. Once the quality level of v_3 has reached the lowest point, lowering the available power further finally caused the model to lower the quality level of v_4. Finally, below 900 mAh, the available power was not sufficient for keeping the nodes running, even at the lowest quality levels, and therefore, the network was shutdown.

We run the same experiment again with available bandwidth as the tightening resource. In this case, we observe the exact opposite behavior, where the quality level of v_4 (cloud node) was lowered first, and then v_3 (local node). This is due to the fact that v_4 requires double the bandwidth when compared to v_3, as shown in Table IIIe.

VII. Related Work

The work in this paper is inspired by related work in managing resource tradeoffs. One stream of work in CPS is concerned with security related tradeoffs, where security comes at a cost of energy or performance. The work in [22] proposes a method to determine when to inject cryptographic checks without interfering with control tasks. The approach in the paper tries to maximize security checks while maintaining a predefined level of control quality. Similarly, the work in [23] proposes a feedback scheduling technique for maintaining network quality of service (QoS) in wireless sensor networks. Both fall under soft real-time constraints, where essentially security is traded off with deadline adherence. The work in [24] manages the energy security tradeoff in a distributed cyber-physical system. This paper generalizes this problem into a multi-resource multi-node optimization.

	Active Sensors	Reliability	Power Usage
q^5	0	0	450
q^4	1	0.94	455
q^3	2	0.96	460
q^2	3	0.98	465
q^1	4	1	470

(a) Quality levels and resource usage of v_0.

	Resolution	Bandwidth	Power Usage
q^6	256 x 144	15	446
q^5	426 x 240	35	476
q^4	640 x 360	50	488
q^3	854 x 480	125	500
q^2	1280 x 720	275	512
q^1	1920 x 1080	2500	566

(b) Quality levels and resource usage of v_1.

	Illuminance	Brightness	Power Usage
q_{11}	120000	0	472
q_{10}	108000	10	479
q_9	96000	20	486
q_8	84000	30	493
q_7	72000	40	500
q_6	60000	50	507
q_5	48000	60	514
q_4	36000	70	521
q_3	24000	80	528
q_2	12000	90	535
q_1	0	100	542

(c) Quality levels and resource usage of v_2.

	Resolution	Bandwidth	Power Usage
q^6	256 x 144	15	458
q^5	426 x 240	35	464
q^4	640 x 360	50	470
q^3	854 x 480	125	476
q^2	1280 x 720	275	482
q^1	1920 x 1080	2500	488

(d) Quality levels and resource usage of v_3.

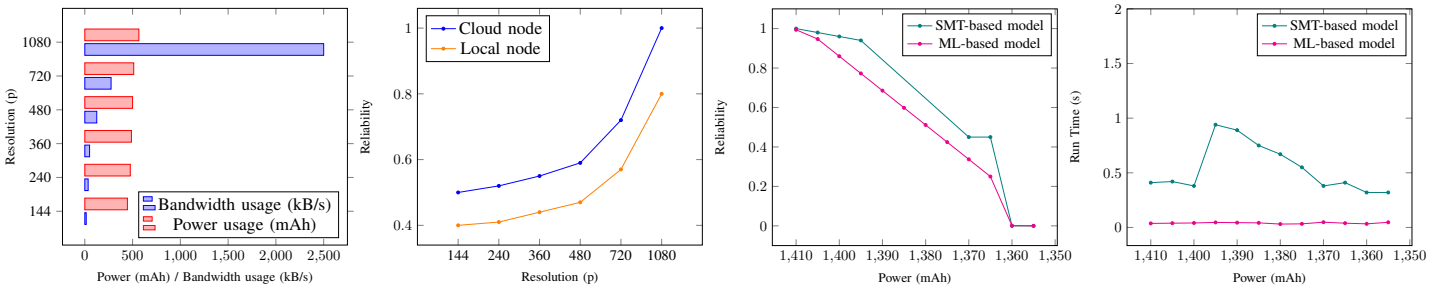
	Resolution	Bandwidth	Power Usage
q^6	256 x 144	30	452
q^5	426 x 240	70	452
q^4	640 x 360	100	452
q^3	854 x 480	250	452
q^2	1280 x 720	550	452
q^1	1920 x 1080	5000	452

(e) Quality levels and resource usage of v_4.

	MOTION	CAM	LIGHT	LOCAL	CLOUD
1410	q_1	q_1	q_1	q_1	q_1
1405	q_2	q_1	q_1	q_1	q_1
1400	q_3	q_1	q_1	q_1	q_1
1395	q_4	q_1	q_1	q_1	q_1
1390	q_4	q_1	q_1	q_2	q_1
1385	q_4	q_1	q_1	q_3	q_1
1380	q_4	q_1	q_1	q_4	q_1
1375	q_4	q_1	q_1	q_5	q_1
1370	q_4	q_1	q_1	q_6	q_1
1365	q_4	q_1	q_1	q_6	q_1
1360	q_4	q_1	q_1	q_6	q_3
1355	q_4	q_1	q_1	q_6	q_6

(f) Quality level changes due to available power.

TABLE III: Quality level tables for different nodes.



(a) Resolution vs. power. (b) Local vs. cloud processing. (c) Reliability vs. power. (d) Run time vs. power.

Fig. 5: Case study results.

There is a line of work in the parallel and distributed processing domain on distributing power resources efficiently. The work in [25] proposes a method to bound the energy consumption of an MPI program using a linear programming model that knows the execution time of jobs on machines and the effect of changing the frequency on their speedup. This work has been extended in [26] to propose a scalable method to determine individual task power bounds in a distributed setting given a global power bound. The work in [27] addresses the problem of energy consumption in a producer consumer network using learning mechanisms to reduce the energy consumption of the overall system.

Many researchers targeted the problem of finding optimal energy savings without impacting performance. The work in [28] studies the trade-off between energy and delay for a wide set of applications. The paper also studies metrics to use to predict memory or communication bottlenecks. The work in [29], [30] attempted to tackle the problem on a single processor. The work mainly proposes an alternative DVFS strategy that maintains the same performance at reduced energy consumption. The work in [31] discusses the effectiveness of using control theory in power management. The series of papers [32], [33], [34] constructs an integer linear programming (ILP) model to determine the minimum energy consumption that a program can consume on a single processor. Then, the authors propose a heuristic to approximate the ILP, and couples this with an analytical model for energy consumption prediction.

VIII. Conclusion and Future Work

In this paper, we presented a generalized model of streaming network of IoT applications as a *network of producers and consumers*. Our model captures tradeoffs between the *quality* of the output and *resource usage*. We formulated these tradeoffs as a *multi-objective optimization* problem that aims at minimizing the resource usage while maximizing the reliability (and quality) of devices (or tasks) in a network. To solve the aforementioned optimization problem, we presented an efficient algorithm based on constraint solving using SMT-solvers to determine the optimal selection of processing quality for each node in the network such that resource bounds are respected and error is minimized. We further improve this work by incorporating machine learning and dramatically speed up the resource optimization speed. This is an important problem as IoT network applications often involve stream processing which comprises a complex network of processing nodes where data is received, processed, and then transmitted to subsequent nodes. We have fully implemented our technique and report experimental results on a network of IoT devices.

As for future work, an immediate extension of our technique is to model networks with feedback loops that are not necessarily acyclic. Another interesting direction is to apply our technique in the area of runtime monitoring of distributed systems. A runtime monitor is essentially the consumer of events produced by different units in a system and there is the obvious tradeoff between reliability of monitors and their runtime overhead as well as network communication.

REFERENCES

[1] J. Yick, B. Mukherjee, and D. Ghosal, "Wireless sensor network survey," *Computer networks*, vol. 52, no. 12, pp. 2292–2330, 2008.

[2] V. Mittal, S. Gupta, and T. Choudhury, "Comparative analysis of authentication and access control protocols against malicious attacks in wireless sensor networks," in *Smart computing and informatics*. Springer, 2018, pp. 255–262.

[3] Y. Zhou, Y. Zhang, and Y. Fang, "Access control in wireless sensor networks," *Ad Hoc Networks*, vol. 5, no. 1, pp. 3–13, 2007.

[4] B. Bhuyan, H. K. D. Sarma, N. Sarma, A. Kar, R. Mall *et al.*, "Quality of service (qos) provisions in wireless sensor networks and related challenges," *Wireless Sensor Network*, vol. 2, no. 11, p. 861, 2010.

[5] L. Liu, W. Kong, T. Ando, H. Yatsu, and A. Fukuda, "A survey of acceleration techniques for smt-based bounded model checking," in *2013 international conference on computer sciences and applications*. IEEE, 2013, pp. 554–559.

[6] L. M. de Moura and N. Bjørner, "Z3: An efficient SMT solver," in *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, 2008, pp. 337–340.

[7] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg *et al.*, "Scikit-learn: Machine learning in python," *the Journal of machine Learning research*, vol. 12, pp. 2825–2830, 2011.

[8] N. Ketkar, "Introduction to keras," in *Deep learning with Python*. Springer, 2017, pp. 97–111.

[9] D. Brunelli and C. Caione, "Sparse recovery optimization in wireless sensor networks with a sub-nyquist sampling rate," *Sensors*, vol. 15, no. 7, pp. 16 654–16 673, 2015.

[10] A. H. Sodhro, L. Chen, A. Sekhari, Y. Ouzrout, and W. Wu, "Energy efficiency comparison between data rate control and transmission power control algorithms for wireless body sensor networks," *International Journal of Distributed Sensor Networks*, vol. 14, no. 1, p. 1550147717750030, 2018.

[11] I. Mehmood, A. Ullah, K. Muhammad, D.-J. Deng, W. Meng, F. Al-Turjman, M. Sajjad, and V. H. C. de Albuquerque, "Efficient image recognition and retrieval on iot-assisted energy-constrained platforms from big data repositories," *IEEE Internet of Things Journal*, vol. 6, no. 6, pp. 9246–9255, 2019.

[12] R. Medhat, S. Funk, and B. Rountree, "Scalable performance bounding under multiple constrained renewable resources," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 16, 2017.

[13] P. Kuila and P. K. Jana, "A novel differential evolution based clustering algorithm for wireless sensor networks," *Applied soft computing*, vol. 25, pp. 414–425, 2014.

[14] K. K. Lim, J. Park, and J. G. Shon, "Differential data processing technique to improve the performance of wireless sensor networks," *The Journal of Supercomputing*, vol. 75, no. 8, pp. 4489–4504, 2019.

[15] R. Mogull and L. Securosis, "Understanding and selecting a data loss prevention solution," *Technicalreport, SANS Institute*, 2007.

[16] O. I. Abiodun, A. Jantan, A. E. Omolara, K. V. Dada, N. A. Mohamed, and H. Arshad, "State-of-the-art in artificial neural network applications: A survey," *Heliyon*, vol. 4, no. 11, p. e00938, 2018.

[17] S. Xu and L. Chen, "A novel approach for determining the optimal number of hidden layer neurons for fnn's and its application in data mining," 2008.

[18] S. Geman, E. Bienenstock, and R. Doursat, "Neural networks and the bias/variance dilemma," *Neural computation*, vol. 4, no. 1, pp. 1–58, 1992.

[19] T.-T. Wong and P.-Y. Yeh, "Reliable accuracy estimates from k-fold cross validation," *IEEE Transactions on Knowledge and Data Engineering*, vol. 32, no. 8, pp. 1586–1594, 2019.

[20] J. Gareth, W. Daniela, H. Trevor, and T. Robert, *An introduction to statistical learning: with applications in R*. Spinger, 2013.

[21] M. Kuhn, K. Johnson *et al.*, *Applied predictive modeling*. Springer, 2013, vol. 26.

[22] V. Lesi, I. Jovanov, and M. Pajic, "Security-aware scheduling of embedded control tasks," *ACM Trans. Embed. Comput. Syst.*, vol. 16, no. 5s, pp. 188:1–188:21, Sep. 2017. [Online]. Available: http://doi.acm.org/10.1145/3126518

[23] F. Xia, L. Ma, J. Dong, and Y. Sun, "Network qos management in cyber-physical systems," in *Embedded Software and Systems Symposia, 2008. ICESS Symposia'08. International Conference on*. IEEE, 2008, pp. 302–307.

[24] A.-D. Vu, R. Medhat, and B. Bonakdarpour, "Managing the security-energy tradeoff in distributed cyber-physical systems," in *Proceedings of the 10th ACM/IEEE International Conference on Cyber-Physical Systems*, 2019, pp. 118–128.

[25] B. Rountree, D. K. Lowenthal, S. Funk, V. W. Freeh, B. R. De Supinski, and M. Schulz, "Bounding energy consumption in large-scale MPI programs," in *Supercomputing, 2007. SC'07. Proceedings of the 2007 ACM/IEEE Conference on*. IEEE, 2007, pp. 1–9.

[26] R. Medhat, S. Funk, and B. Rountree, "Scalable performance bounding under multiple constrained renewable resources," in *Proceedings of the 5th International Workshop on Energy Efficient Supercomputing*, 2017, pp. 1–8.

[27] R. Medhat, B. Bonakdarpour, and S. Fischmeister, "Energy-efficient multiple producer-consumer," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 3, pp. 560–574, 2018.

[28] V. W. Freeh, D. K. Lowenthal, F. Pan, N. Kappiah, R. Springer, B. L. Rountree, and M. E. Femal, "Analyzing the energy-time trade-off in high-performance computing applications," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 18, no. 6, pp. 835–848, 2007.

[29] C.-H. Hsu and U. Kremer, "The design, implementation, and evaluation of a compiler algorithm for cpu energy reduction," in *ACM SIGPLAN Notices*, vol. 38, no. 5. ACM, 2003, pp. 38–48.

[30] J. R. Lorch and A. J. Smith, "Improving dynamic voltage scaling algorithms with pace," in *ACM SIGMETRICS Performance Evaluation Review*, vol. 29, no. 1. ACM, 2001, pp. 50–61.

[31] Q. Wu, P. Juang, M. Martonosi, L.-S. Peh, and D. W. Clark, "Formal control techniques for power-performance management," *IEEE Micro*, no. 5, pp. 52–62, 2005.

[32] F. Xie, M. Martonosi, and S. Malik, "Compile-time dynamic voltage scaling settings: Opportunities and limits," in *ACM SIGPLAN Notices*, vol. 38, no. 5. ACM, 2003, pp. 49–62.

[33] ——, "Intraprogram dynamic voltage scaling: Bounding opportunities with analytic modeling," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 1, no. 3, pp. 323–367, 2004.

[34] ——, "Bounds on power savings using runtime dynamic voltage scaling: an exact algorithm and a linear-time heuristic approximation," in *Proceedings of the 2005 international symposium on Low power electronics and design*. ACM, 2005, pp. 287–292.