FireDeX: a Prioritized IoT Data Exchange Middleware for Emergency Response

Kyle E Benson, Georgios Bouloukakis, Casey Grant, Valérie Issarny, Sharad Mehrotra, Ioannis Moscholios, Nalini Venkatasubramanian

FireDeX: a Prioritized IoT Data Exchange Middleware for Emergency Response

Kyle E. Benson[1], Georgios Bouloukakis[1,3], Casey Grant[2], Valérie Issarny[3], Sharad Mehrotra[1], Ioannis Moscholios[4], Nalini Venkatasubramanian[1]
{kebenson,gboulouk}@ics.uci.edu,cgrant@nfpa.org,valerie.issarny@inria.fr,sharad@ics.uci.edu
idm@uop.gr,nalini@uci.edu
[1]Donald Bren School of Information and Computer Sciences, University of California, Irvine, USA
[2]National Fire Protection Association, Quincy, USA
[3]MiMove Team, Inria Paris, France
[4]Dept. of Informatics & Telecommunications, University of Peloponnese, Tripolis, Greece

ABSTRACT

Real-time event detection and targeted decision making for emerging mission-critical applications, e.g. smart fire fighting, requires systems that extract and process relevant data from connected IoT devices in the environment. In this paper, we propose FireDeX, a cross-layer middleware that facilitates timely and effective exchange of data for coordinating emergency response activities. FireDeX adopts a publish-subscribe data exchange paradigm with brokers at the network edge to manage prioritized delivery of mission-critical data from IoT sources to relevant subscribers. It incorporates parameters at the application, network, and middleware layers into a data exchange service that accurately estimates end-to-end performance metrics (e.g. delays, success rates). We design an extensible queueing theoretic model that abstracts these cross-layer interactions as a network of queues, thereby making it amenable for rapid analysis. We propose novel algorithms that utilize results of this analysis to tune data exchange configurations (event priorities and dropping policies) while meeting situational awareness requirements and resource constraints. FireDeX leverages Software-Defined Networking (SDN) methodologies to enforce these configurations in the IoT network infrastructure. We evaluate its performance through simulated experiments in a smart building fire response scenario. Our results demonstrate significant improvement to mission-critical data delivery under a variety of conditions. Our application-aware prioritization algorithm improves the value of exchanged information by 36% when compared with no prioritization; the addition of our network-aware drop rate policies improves this performance by 42% over priorities only and by 94% over no prioritization.

KEYWORDS

Publish/Subscribe Middleware, Event Prioritization, Utility Functions, Emergency Response, Queueing Networks, SDN

Middleware '18, December 10–14, 2018, Rennes, France

ACM ISBN 978-1-4503-5702-9/18/12.
https://doi.org/10.1145/3274808.3274830

ACM Reference Format:
Kyle E. Benson, Georgios Bouloukakis, Casey Grant, Valérie Issarny, Sharad Mehrotra, Ioannis Moscholios, Nalini Venkatasubramanian. 2018. FireDeX: a Prioritized IoT Data Exchange Middleware for Emergency Response. In *Proceedings of 19th International Middleware Conference (Middleware '18).* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3274808.3274830

1 INTRODUCTION

As we move further into a future full of connected devices, the Internet of Things (IoT) promises to revolutionize societal-scale operations and influence our daily lives. It integrates pervasive sensing/actuation, dynamic data analytics, and communications. Domains such as transportation, home automation, healthcare, and emergency response are becoming increasingly IoT-enabled; this provides data-driven insights to improve situational awareness. This is particularly useful in mission critical applications, e.g. to enable effective and timely emergency response. Recent smart city efforts such as the SmartAmerica Challenge and Global City Teams Challenge have showcased the integration of IoT into a variety of community settings and application domains [8, 31, 60].

A distributed ***data exchange*** solution that manages the flow of relevant data to/from devices, systems and individuals (data producers and consumers) is a critical centerpiece of IoT deployments. In this paper, we adopt a ***publish/subscribe*** (pub/sub) model for IoT data exchange based on our previous experiences with such systems and the popular use of pub/sub (e.g. MQTT[42]) in IoT implementations. In mission critical emergency scenarios, IoT devices can forward raw sensor data to interested recipients (e.g first responders and emergency management agencies) through a data exchange broker to help coordinate the response effort. We consider IoT sensors as publishers, all manner of data as events, and interested entities (i.e. human stakeholders or other IoT devices and services) as subscribers. Data exchange brokers route information to actuators (e.g. alarms), data analytics services that detect new events, or to a local logging database for post-incident analysis and forensics. Key challenges arise when enabling timely data exchange to a diverse set of recipients including: managing heterogeneous information with varying size, format, relevance, urgency, etc.; seamless integration of new IoT data sources with pre-existing sources and information on the fly; supporting reliable and timely communication over constrained networks (e.g. due to lossy channels and failed components).

To overcome these challenges, we propose **FireDeX**, an integration middleware that unifies (a) smartspace IoT data and infrastructure (e.g. a Smart Building Management System (BMS)) with (b) programmable network infrastructure (e.g. through Software-Defined Networking (SDN)) and (c) domain specific applications (e.g. smart fire fighting systems and apps brought onsite by emergency responders). FireDeX aims to support timely and reliable delivery of the most critical data to relevant subscribers despite challenging network conditions. It utilizes the cross-layer (i.e. application and network-aware) approach for IoT data exchange shown in Fig. 1 and described here. It leverages edge computing (i.e. data exchange brokers at the network edge) and SDN to bridge critical application requirements with network state. This approach facilitates platform and device independent adaptation of IoT communication at the middleware layer. Using SDN, FireDeX configures the underlying physical network to prioritize messages according to the requirements (i.e. subscriptions with utility functions that quantify situational awareness) and underlying network resource constraints (e.g. bandwidth, error). Through the use of priority queues and carefully tuned packet drop rates (i.e. for bandwidth allocation), it ensures timely delivery of the most important data possible. To this end, we construct an extensible queueing theoretic model to abstract the cross-layer flow of data. We present our formal analysis that our proposed novel algorithms leverage to configure the data exchange. These algorithms manage active subscriptions by separately assigning priorities to each and then allocating bandwidth to them. Our simulation-based experiments validate our model/analysis, demonstrate the FireDeX approach's efficacy, and compare its core algorithms with alternatives.

The FireDeX middleware combines several novel capabilities and design features, notably the following key contributions:

- Applying a cross-layer approach (application, middleware, networking) to prioritizing mission critical IoT data exchange during a fire response scenario in an IoT-enhanced smart building with SDN-enabled edge infrastructure. (§2)
- Formulating an extensible formal model of these three layers based on the unified framework of queueing theory. This model includes our new multi-class priority queueing model. We use it here to represent an SDN switch, but it is generally suitable for use in other queueing networks. (§3)
- Leveraging the above queueing model to explore the configuration parameter space and derive novel algorithms that prioritize IoT events and tune notification delivery/delay. FireDeX leverages SDN to configure the underlying physical network with priority queueing disciplines and carefully tuned packet drop rates (i.e. for bandwidth allocation). (§4)

2 THE FIREDEX APPROACH

To motivate the need for prioritized IoT data exchange, the challenges involved, and our proposed approach, we begin with an IoT-enhanced structural fire scenario.

2.1 A Driving Scenario: Fire Fighting with IoT

During a fire, an occupant or automated system activates an emergency dispatch process. A local fire department(s) responds by sending a team of fire fighters (FFs). An Incident Commander (IC) coordinates the effort from an Incident Command Post (ICP) set up onsite. To effectively manage the dynamic response and minimize casualties, injuries, and property damage, the IC requires up-to-date situational awareness information. Today the IC still derives much of this information from non-digital sources (e.g. human-initiated reports via voice and/or FF radio, paper records) and collates it in an ad-hoc manner to guide operations. However, smartspaces equipped with IoT devices enable access to live data feeds that can generate actionable information in real-time through proper filtering, prioritizing, and analysis. Such a data-driven approach to improving FF outcomes fuses sensor data producers with consumers (e.g. analytics) and actions (e.g. device actuations, alerts). This improves the efficacy of the classical fire response workflow, thereby driving the emergence of *Smart Fire Fighting (SFF)* technologies.

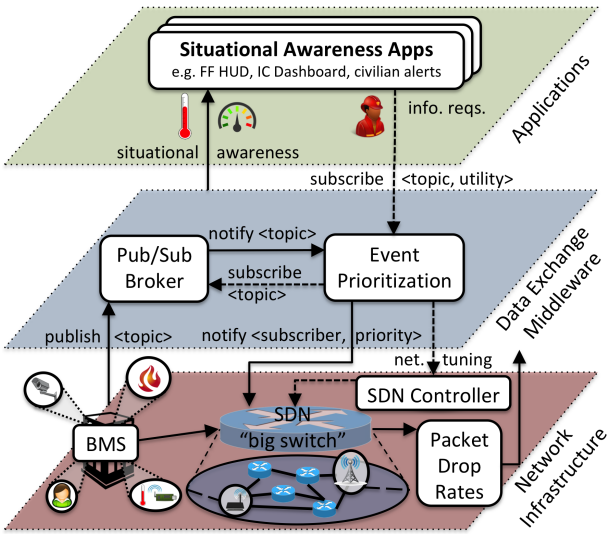

Figure 1: The FireDeX cross-layer middleware.

In 2015, the US National Institute of Standards and Technology (NIST) and National Fire Protection Association (NFPA) published the Research Roadmap for Smart Fire Fighting (Res. Roadmap) [28] that "establishes the scientific and technical basis" of SFF. This interdisciplinary collaboration brought together members of industry, government, the fire science practice, and academia to identify technologies and key challenges in SFF.

Maintaining up-to-date situational awareness for SFF requires the integration and enrichment of static and dynamic data from buildings and IoT infrastructure. Static information such as building floor plans, inspection histories, and presence of hazardous material can be gathered apriori. Dynamic information published by IoT devices (in the building and brought by FFs) must be delivered to relevant subscribers (e.g. analytics services) and combined with contextual knowledge to generate situational awareness. Different data types vary in importance to responders depending on the situation (e.g. "smoke" > "water pressure" when initially sizing up the event and vice versa during active fire suppression).

Stakeholders (e.g. IC, FFs, residents) request data and visualize it using e.g. dashboards, heads-up displays, mobile alerts. The IC assesses building occupancy for coordinating evacuations, tracks the locations and biometrics of FFs to ensure their safety and effectiveness, and detects environmental hazards within the building

such as high temperature or smoke levels. A key challenge for SFF is delivering mission-critical data for "timely, targeted decision making" in an unreliable, partially available, and congested network environment [28]. Given the heterogeneous value of events and limited resources for delivering notifications, we believe event prioritization is necessary in such mission-critical settings. To this end, we propose the FireDeX data exchange middleware to manage the flow of situational awareness information. It assigns and enforces event priorities according to the requirements and capabilities of three middleware layers: application, data exchange, network.

2.2 Background on IoT Data Exchange

Several research challenges for mission-critical IoT data exchange arise from the above driving SFF scenario.

Heterogeneity of devices and information: Heterogeneous IoT devices in buildings/structures (e.g. sensors, cameras) produce data that varies in size, frequency (periodic samples vs. asynchronous alerts), type, and importance to individual subscribers [6, 47, 62, 63]. In existing structures, this complexity is handled by a BMS that locally manages devices and data. Recent work on smart buildings includes ontologies [4] and protocols to support building automation [17], techniques to preserve privacy [40], enabling energy efficiency [58], programmable building operating system services [19], context aware IoT management via SDN [33], and accurate positioning for location based apps [18].

To manage both scale and heterogeneity, we design FireDeX as an edge middleware that leverages existing IoT infrastructure and services managed by a third-party BMS. Its pub/sub approach integrates capabilities of new devices/tools brought on scene by responders, while separating operational and ownership concerns. Note that external entities often lack the knowledge, access, and expertise to reconfigure local devices. This might conflict with existing configurations customized by building IoT administrators.

Managing smart spaces at scale in real time: Tuning data collection parameters at each device (e.g sampling rate, resolution) for individual subscribers is not always viable, especially as the number and diversity of devices scales. FireDeX's edge broker approach naturally supports scalability; geographically-dispersed subscribers interconnect through a distributed network of data exchange brokers. This network may be hierarchical: top-level brokers running in cloud data centers serve a large region with many local brokers running in edge data centers to serve a smaller local area (e.g. one or a few buildings, a campus). FireDeX can leverage work in large-scale pub/sub systems [7, 47, 63] as well as research into managing data exchange configurations according to data processing workload characteristics [38, 55]. Other related research proposes similar centralized control of IoT data exchange [32, 61].

Managing unreliable IoT networks: Communications are often constrained in crisis scenarios. FireDeX leverages SDN to manage networking for IoT deployments by offloading network configuration tasks from constrained devices and network hardware.

SDN APIs (e.g. OpenFlow [39], P4 [13]) provide a unified view of and control over the underlying network infrastructure. An SDN controller (e.g. ONOS [10]) observes the underlying network by querying switches for various statistics e.g. packets sent/received, loss rates, etc. SDN provides a variety of abstractions to represent the underlying physical network. These include authorized access to directly manage physical switches, control over virtual (software-based) switches [43] (e.g. running alongside the broker), network virtualization [12] to reserve "slices" of the physical infrastructure, etc. Recent research into SDN-enabled 5G cellular architectures [54] supports the potential for such interfaces that connect emergency responder devices to the building's internal network.

SDN-based approaches have been used in general IoT networks to: configure QoS-enabled routing [36, 45]; differentiate pub/sub subscriptions at the network level and prioritize them separately to provide bounded queueing delays [57]; manage wireless IoT sensor networks [21]; for meet real-time data flow processing delays through prioritization [3]. Recent research [9, 11, 53] used SDN to enable high performance pub/sub through network-level multicast. FireDeX's extensible design can easily integrate such techniques.

Research on Network Utility Maximization (NUM) [59] aims to tune the underlying network according to application-level requirements. NUM configures a network (e.g. assigns bandwidth) to serve nodes in a manner that maximizes utility functions that capture a user's degree of satisfaction with the network's performance. Few prior researchers have investigated discrete priority classes, which we leverage in our approach, within the context of NUM. The authors of [41] propose assigning more bandwidth to users (i.e. via weighting their requests higher) based on their requested priority levels. Similarly, [48] manages IoT devices to maximize utility by allocating bandwidth and offloading processing, but the authors do not use SDN or consider the data exchange middleware and prioritized application requirements. Our cross-layer approach and consideration of utility functions sets apart our work from most related SDN research referenced above.

Modeling cross-layer data exchange interactions: To analyze IoT data exchange performance we must consider all three layers' characteristics and their effects on each other. However, existing efforts typically focus on each layer in isolation. Therefore, we model cross-layer interactions by composing and extending previous work at each layer through the unified framework of *queueing theory* [29, 49]. Queueing Petri Nets (QPNs) enable accurate performance prediction in pub/sub systems [34, 46]. Alternatively, [14, 15] model and analyze the performance of pub/sub and middleware protocols using Queueing Networks (QNs). While QPNs have an advantage over QNs in representing parallelism, QNs provide convenient primitives to construct well-formed performance models for efficient analysis [56]. Furthermore, QNs have been extensively applied to model network infrastructure performance [5, 24, 25, 27] and more recently SDN infrastructure [22, 51, 52].

2.3 Enabling Event Prioritization

We now overview how the FireDeX middleware addresses the above challenges. We frame our discussions in terms of the three layers depicted in Fig. 1: mission-critical applications, abstractions representing the physical network infrastructure, and the data exchange middleware that bridges these two to manage the overall system configuration and flow of information. As shown in Fig. 2, FireDeX integrates other middleware technologies: data APIs for interfacing with IoT data (i.e. through the BMS), a local pub/sub broker, a thin client middleware running on each subscribing IoT device,

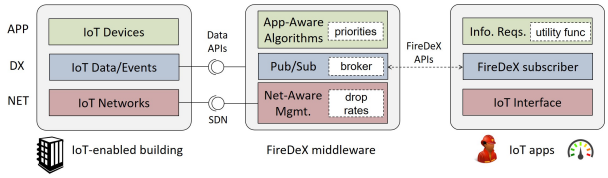

Figure 2: FireDeX middleware architecture

and SDN APIs for managing local network infrastructure. It implements the proposed algorithms and provides middleware APIs for our data prioritization and network management approach. To ensure delivery of the most important events despite network resource constraints (e.g. failures, poor signal strength, limited bandwidth), it **prioritizes events** and **allocates available network bandwidth** according to application requirements.

Application layer: FireDeX subscriber devices run a client middleware to establish broker connections, retrieve a list of event topics, subscribe to relevant ones, and report data exchange/network channel statistics. Because different data vary by importance, we propose prioritizing events according to their relative importance to the emergency response effort. To configure this, subscribers register **utility functions** with their FireDeX subscriptions. These functions capture a quantified measure of value for varying rates of event delivery performance. Our proposed algorithms consider these utility functions when configuring the data exchange and network to maximize users' utility (i.e. situational awareness).

Data exchange layer: FireDeX prioritizes subscriptions according to their subscriber-specified utility functions. It leverages the queueing theoretic analysis we present in §3 to estimate system performance under a given configuration. This analysis drives the algorithms presented in §4 that assign discrete **priority classes** and allocate available network bandwidth. FireDeX connects subscriber clients and the BMS data APIs with the pub/sub broker, which performs the actual routing of events.

While some existing data exchange implementations and protocols support priorities, configuring them requires specific APIs [44]. Furthermore, many popular options (e.g. the MQTT [42] protocol and associated broker implementations) do not support priorities and so require equal treatment of all events transmitted to the same subscriber. To decouple FireDeX from the underlying data exchange broker, which may be specific to the site's BMS, we do not employ application-layer (i.e. in-broker) prioritization. Rather, we propose enforcing priorities at the network layer through unified APIs provided by SDN. This approach accounts for both application-level requirements (e.g. utility functions) and network-level state information (e.g. available bandwidth) without mandating (or extensively modifying) specific broker technologies. Hence, FireDeX essentially extends the data exchange broker/protocol with network and application-aware prioritization.

Network layer: FireDeX manages network infrastructure through APIs provided by an SDN controller that likely runs alongside the BMS (i.e. at the edge). It gathers network state information to derive resource constraints. It combines these with the subscribers' information requirements to drive its management algorithms. The authors of [61] previously advocated a similar approach of centrally gathering a global view of a pub/sub system's state to simplify its management. They refer to this central control approach as *SDN-like* because it separates the pub/sub control and data plane. They

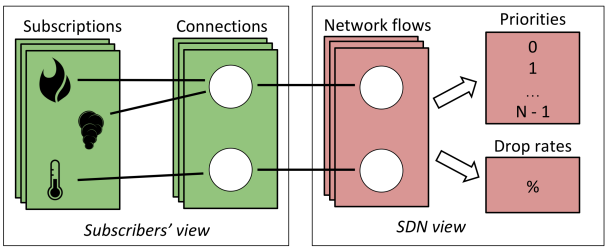

Figure 3: FireDeX differentially prioritizes subscriptions at the SDN layer using multiple connections per subscriber.

further propose integrating SDN with the data exchange middleware, which this centralization cleanly enables. We advocate for this approach in IoT settings where offloading device management and data processing from constrained devices typically leads to centralized (i.e. cloud-centric) designs. For simplicity of discussion, we consider the ***big switch*** model shown in Fig. 1 that abstracts the entire local physical network into a single virtual SDN switch. This provides a simplified single-network view of the whole distributed system that may span multiple physical heterogeneous networks (e.g. building Wi-Fi and local cellular) and different locations.

To enforce event priorities at the network layer, FireDeX leverages SDN APIs. It configures priority queueing disciplines for packets matching the different subscriptions. However, for the network to distinguish the data exchange-layer concept of subscriptions, we must first translate it to a network-level concept. As shown in Fig.3, we accomplish this through the SDN concept of **network flows**. SDN switches match incoming packets of a particular network flow according header information. For example, OpenFlow considers OSI Layer 2-4 fields: IP/MAC address, UDP/TCP port, VLAN, etc. To differentiate subscriptions as belonging to different network flows, a FireDeX subscriber maintains multiple **network connections** with the pub/sub broker (e.g. over different Layer 4 port numbers). This may represent different applications running on the same device and/or one application opening multiple connections. The latter case enables the network to distinguish and manage individual groups of subscriptions based on their assigned connection. The data exchange middleware layer dictates this assignment of (possibly multiple) subscriptions to one network connection and its corresponding unique network flow. Subscribers initiate multiple connections and then register each subscription to avoid directly configuring the underlying data exchange broker. FireDeX also assigns each network flow a priority level by considering subscriber requirements. It configures the SDN switches to forward packets matching these network flows through the proper priority queue.

To manage available network resources, FireDeX also allocates bandwidth to each network flow. It applies **preemptive packet drop rates** that consider the utility of each network flow's subscriptions. We propose dropping lower-priority packets before switch buffers fill up to prevent high delays and dropping of higher-priority packets. §4.3 discusses this concept further and proposes our optimization-based algorithm for setting these drop rates. This algorithm is partly inspired by the aforementioned research in Network Utility Maximization (NUM). However, in FireDeX subscribers actually define the utility functions according to their information needs, and so they indirectly cooperatively control the assignment of bandwidth. Furthermore, our proposal leverages discrete priority

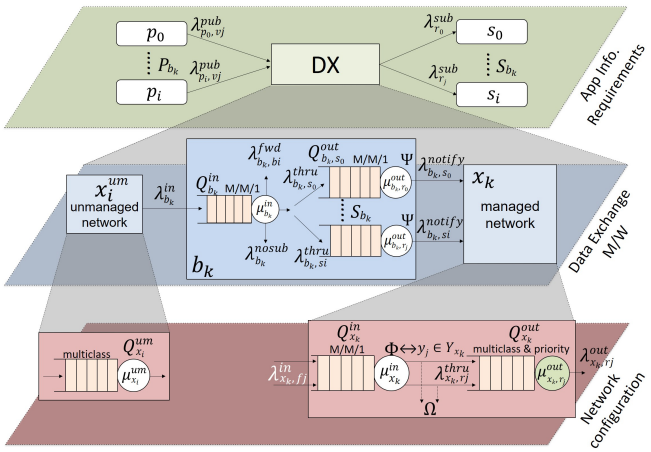

Figure 4: FireDeX queueing network model.

classes to drive priority queueing disciplines and calculates the best priority assignments rather than assuming them as a given input.

3 FIREDEX FORMAL MODEL

From the above scenario, we formulate a generalized model for prioritized data exchange in mission-critical settings. FireDeX combines queueing theoretic approaches from both the middleware and network layers to construct the representative and extensible 3-layer queueing network shown in Fig. 4. The data exchange middleware bridges the network infrastructure and application layers to enable a novel cross-layer end-to-end performance model. We derive this analytical model to estimate a particular configuration's expected performance.

3.1 Queueing Network Performance Modeling

Refer to Table 1 for the notations used throughout this section.

3.1.1 ***Application Modeling.*** Let $V_{p_i} \subseteq V$ be the set of topics each p_i publishes to e.g. "smoke". Let $\lambda^{pub}_{p_i,v_j}$ be the publication rate of events with topic v_j published by p_i per unit time.

Assumption 1. *$\lambda^{pub}_{p_i,v_j}$ is based on a Poisson process.*

We define each subscription as a tuple $r_j = (s_i, v_j, U_{r_j})$ where utility function U_{r_j} quantifies the information value for subscriber s_i receiving events with topic v_j. Let $R_{s_i} = \{r_j \in R : s_i \in r_j\}$ be the set of prioritized information requests (i.e. subscriptions) for each subscriber s_i. Let $\lambda^{sub}_{r_j}$ be the incoming rate of events matching subscription r_j received per unit time by subscriber s_i.

Let Ξ_{r_j} be the *success rate* of delivering events matching subscription $r_j = (s_i, v_j, U_{r_j})$ to subscriber s_i. By Assumption 1, we can estimate Ξ_{r_j} (i.e. by summing Poisson process rates to produce the rate of an aggregate Poisson process) as:

$$\mathrm{E}\left[\Xi_{r_j}\right] = \frac{\lambda^{sub}_{r_j}}{\sum_{p_i \in P} \lambda^{pub}_{p_i,v_j}}$$

Let Δ_{r_j} be the *response time*: the average end-to-end delay of events matching subscription $r_j = (s_i, v_j, U_{r_j})$ from the moment they are published until s_i receives them. Below we calculate this metric, which includes event processing times, network delays, etc.

3.1.2 ***Data Exchange Modeling.*** The data exchange layer represents a network of broker nodes B. We assume that each publisher/subscriber connects with a single broker that we refer to as its *home broker*: b_{p_i} is the broker that p_i publishes to and b_{s_i} is the broker that subscriber s_i receives events from. Furthermore, we define the set of publishers connected with b_k as $P_{b_k} = \{p_i \in P : b_k = b_{p_i}\}$, the set of subscribers connected with b_k as $S_{b_k} = \{s_i \in S : b_k = b_{s_i}\}$, and the set of subscriptions handled by b_k as $R_{b_k} = \cup_{s_i \in S_{b_k}} R_{s_i}$.

A broker b_k forwards events with rate $\lambda^{fwd}_{b_k,b_i}$ to another broker $b_i \in B$ for eventual consumption by one of the latter's subscribers. As depicted in Fig. 4, we model each broker b_k using a single inbound M/M/1 queue $Q^{in}_{b_k}$ and multiple outbound M/M/1 queues $Q^{out}_{b_k,s_i}$. By Assumption 1 and the exponentially distributed service rate of $Q^{in}_{b_i}, \forall b_i \in B - \{b_k\}$, we know that $\lambda^{fwd}_{b_k,b_i}$ is Poisson. Hence, we can define the arrival rate of events at $Q^{in}_{b_k}$ as the sum of all (post-network transformation) event publication/forwarding rates over all publishers/brokers:

$$\lambda^{in}_{b_k} = \sum_{p_i \in P_{b_k}} \sum_{v_j \in V_{p_i}} \Gamma\left(\lambda^{pub}_{p_i,v_j}, p_i, b_k\right) + \sum_{b_i \in B, b_i \neq b_k} \Gamma\left(\lambda^{fwd}_{b_i,b_k}, b_i, b_k\right)$$

Note that Γ, which we define in §3.1.3, represents network-layer traffic shaping due to error rates, administrative policies, etc.

Forwarding, replication, or dropping of events based on current subscriptions occurs at the exit of $Q^{in}_{b_k}$. Let $\mu^{in}_{b_k}$ be $Q^{in}_{b_k}$'s service rate for analyzing an incoming event and determining where to forward it (e.g. based on a topic routing tree). We assume $\mu^{in}_{b_k}$ is constant (or averaged) across all topics and independent of current subscriptions. Events not matching subscriptions R_{b_k} are dropped with rate $\lambda^{nosub}_{b_k}$.

For each of broker b_k's subscribers $s_i \in S_{b_k}$, it forwards events matching a subscription $r_j \in R_{s_i}$ to $Q^{out}_{b_k,s_i}$ with rate $\lambda^{thru}_{b_k,s_i}$ for transmission to s_i. Recall that each broker maintains multiple connections (network flows) with each subscriber. Let $\mu^{out}_{b_k,r_j}$ be the service rate at $Q^{out}_{b_k,s_i}$ that captures the time it takes to map an event matching subscription r_j to the corresponding connection of s_i. It forwards these publications into the network layer with rate $\lambda^{notify}_{b_k,r_j}$. Hence, we calculate the per-subscriber forwarding rate as:

$$\lambda^{notify}_{b_k,s_i} = \sum_{r_j \in R_{s_i}} \lambda^{notify}_{b_k,r_j}$$

FireDeX Configuration Parameters: The data exchange layer also represents the FireDeX configuration service. FireDeX associates each subscription with one of the *network flows* $f_j \in F$ in order to manage subscription traffic in a network-aware manner. Recall from §2.3 that network flows represent multiple connections between a subscriber and its home broker. We define the set of network flows for a particular subscriber s_i as $F_{s_i} \subseteq F$. Additionally FireDeX defines a set of unique *priority classes* $y_j \in Y$. It assigns each network flow to a priority class for managing network traffic in an application-aware manner. Note that y_j has higher priority than y_k for $j < k$, i.e. $y_0 = 0$ is the highest priority.

To configure the end-to-end data exchange interactions across all 3 layers, FireDeX employs the following functions:

Table 1: Notations of the parameters in our cross-layer data exchange model

Application Layer	
Notation	**Description**
$v_j \in V$	event topics
$s_i \in S$	subscribers
$r_j \in R$	subscriptions
$p_i \in P$	publishers
$\lambda^{pub}_{p_i,v_j}$	publication rate
$\lambda^{sub}_{r_j}$	r_j's delivery rate
Ξ_{r_j}	r_j's success rate
Δ_{r_j}	r_j's end-to-end response time

Data Exchange Layer	
Notation	**Description**
$b_k \in B$	brokers
$\lambda^{notify}_{b_k,r_j}$	r_j's notification rate
$\Psi : R \mapsto F$	network flow for a subscription
$\Phi : F \mapsto Y$	priority for a network flow
$\Omega : F \mapsto [0,1]$	packet drop rate for a network flow

Network Layer	
Notation	**Description**
$x_k \in X$	SDN switches
$h_j \in H, H = P \cup S \cup B$	network hosts
$w_{x_k,h_j} \in W, w_{x_k,h_j} \in \mathbb{N}$	bandwidth between x_k and h_j
$G_{v_j} \in \mathbb{Z}_{>0}$	serialized packet size for topic v_j
$z_{h_j,h_i} \in Z, z_{h_j,h_i} \in [0,1]$	packet *error rate*
$\Gamma : \mathbb{N} \times H \times H \mapsto \mathbb{N}$	transforms event departure to arrival rates (e.g. packet errors)
$f_j \in F$	network flows
$y_j \in Y$	unique *priority classes*

$\Psi : R \mapsto F$ is the function mapping subscriptions (i.e. events matching them) to the corresponding subscribers' network flows. Note that we denote $\Psi(s_i, v_j) = \Psi(r_j)$ as the network flow for subscription $r_j = (s_i, v_j, U_{r_j})$ and so $\Psi : S \times V \mapsto F$. As described in §2.3, this mapping allows the SDN data plane to distinguish packets containing events from each other based on their topics.

$\Phi : F \mapsto Y$ is the function mapping network flows to priority classes. This defines which priority class (i.e. priority queue) the SDN infrastructure uses for a packet transmitted on network flow f_j. This packet contains event(s) matching subscriber s_i's subscription r_j where $f_j = \Psi(r_j)$. Hence $\Phi \circ \Psi(r_j)$ is subscription r_j's priority.

$\Omega : F \mapsto [0,1]$ is the function mapping network flows to preemptive packet drop probabilities. By dropping some packets on a network flow, FireDeX more accurately tunes the data exchange configuration than through priority assignment alone. Somewhat akin to network traffic policing, this technique lowers the bandwidth usage of a network flow so that the aggregate bandwidth needs of all flows does not exceed that available. By dropping packets in the lower-priority flows, this prevents switch buffers from filling up and dropping higher-priority packets.

3.1.3 ***Network Modeling.*** Publications forwarded to the network layer are encapsulated in packets for transmission by the SDN infrastructure. To simplify the analysis used in our queueing model, we leverage the following:

Assumption 2. *The data exchange and applications encapsulate each event in a single packet for transmission through the network.*

Let X be the set of *SDN switches* that connect with the various hosts H. A host h_j may have multiple physical network interfaces/connections to one or more switches and packets between two hosts may traverse multiple routes. However, SDN abstractions support the following assumption that simplifies our analysis:

Assumption 3. *We consider multiple switches/routes between two hosts as aggregated into a single virtual SDN switch/link that captures the underlying physical network topology and characteristics.*

By Assumption 3, we need only to model a single *big switch* serving a publisher or subscriber. Hence, we refer to x_{s_i} as the *FireDeX-managed SDN switch* that controls traffic between b_{s_i} and s_i. We refer to x_{p_i} as the *unmanaged SDN switch* that exposes the network characteristics (defined below) of the network channel between b_{p_i} and p_i. Note that FireDeX does not manage the latter switch because this might conflict with deployment-specific IoT device configurations. To model multiple hosts sharing the same network medium (e.g. a wireless channel), we apply Assumption 3 and model such a channel as one switch serving multiple hosts. We therefore define the set of subscribers served by switch x_k as $S_{x_k} = \{s_i \in S : x_{s_i} = x_k\}$, all of their subscriptions as $R_{x_k} = \{\cup_{s_i \in S_{x_k}} R_{s_i}\}$, and and all of their network flows as F_{x_k}. Similarly, let $P_{x_k} = \{p_i \in P : x_{p_i} = x_k\}$ be the set of publishers served by x_k.

Let $Q^{um}_{x_i}$ be the queue modeling the *unmanaged switch* x_i that encompasses a *publisher-broker* or *broker-broker* link. By Assumption 2, we have the packet arrival rate for publications and forwarded events at switch x_i as $\lambda^{pub}_{p_i,v_j}$ and $\lambda^{fwd}_{b_i,b_k,v_j}$ respectively. We model $Q^{um}_{x_i}$ as a multi-class queue, which enables us to calculate the average transmission delay of a packet ($\Delta^{tx}_{r_j}$) based on its size. Each class corresponds to the topic of an event encapsulated within a packet. Hence, we define the expected *serialized size* (e.g. in bytes) of a packet that, by Assumption 2, contains a single event published to topic v_j as $G_{v_j} \in \mathbb{Z}_{>0}$. By Assumption 3, we have w_{x_k,h_j} as the bottleneck bandwidth available between two hosts (i.e. from the switch x_k serving them to the destination host h_j). Therefore, we can define a per-topic packet transmission rate as:

$$\mu^{um}_{x_i,v_j} = \frac{w_{x_i,b_k}}{G_{v_j}}$$

This enables calculating the average *transmission delay* $\Delta^{um}_{x_i}$ of packets in $Q^{um}_{x_i}$. We apply Γ to packets departing the switch queue $Q^{um}_{x_i}$ in order to transform event departure rates from a host h_j to event arrival rates at the destination host h_i. To simplify our analysis, we leave retransmission of packets for future work and instead consider only packet error rates. Let $z_{h_j,h_i} \in [0,1]$ be this packet error rate that, by Assumption 3, allows us to model packet drops at the single switch between these hosts. We have the arrival rate of publications (on topic v_j from publisher p_i at broker b_k) as:

$$\Gamma\left(\lambda^{pub}_{p_i,v_j}, p_i, b_k\right) = \left(1 - z_{p_i,b_k}\right) \lambda^{pub}_{p_i,v_j}$$

We define the transformed arrival rate of events forwarded from broker b_i to b_k similarly.

We model each *managed SDN switch* encompassing a *broker-subscriber* link as two different queues: *1)* an M/M/1 queue $Q^{in}_{x_k}$ that feeds into *2)* our newly-proposed queueing model: a non-preemptive priority and multi-class queue $Q^{out}_{x_k}$. By Assumption 2, we therefore have the arrival rate at switch x_k of event-encapsulating packets within a network flow f_j as $\lambda^{in}_{x_k,f_j}$.

$Q^{in}_{x_k}$ processes each incoming packet by matching its header contents to a corresponding network flow f_j and determining the assigned priority (i.e. $\Phi(f_j)$). Let $\mu^{in}_{x_k}$ be the service rate at $Q^{in}_{x_k}$ that captures the time required to perform this matching (e.g. an SDN switch TCAM lookup), assign the given priority, and route the packet to the appropriate output port. Note that this might actually capture delays from forwarding packets along a multi-switch route.

Before enqueueing the packet at the correct output port, the switch first applies the dropping policy to each flow according to the FireDeX-computed function $\Omega(f_j)$. Because our model does not consider further packet drops in $Q^{in}_{x_k}$ or before $Q^{out}_{x_k}$, we have the per-subscription arrival rate at $Q^{out}_{x_k}$ as:

$$\lambda^{thru}_{x_k,r_j} = \left(1 - \Omega \circ \Psi(r_j)\right) \lambda^{notify}_{b_k,r_j} \tag{1}$$

Multi-class priority queue $Q^{out}_{x_k}$ separates the departure rates of each packet according to its serialized size and the switch's available bandwidth. Note that the assigned priority class affects the response time but not the departure rates of these packets. By Assumption 2, we therefore have the service (i.e. transmission) rate of packets encapsulating events that match subscription $r_j = (s_i, v_j, U_{r_j})$ from SDN switch x_k to subscriber s_i as:

$$\mu^{out}_{x_k,r_j} = \frac{w_{x_k,s_i}}{G_{v_j}}$$

We have the departure rate from $Q^{out}_{x_k}$ as: $\lambda^{out}_{x_k,r_j} = \lambda^{thru}_{x_k,r_j}$. We then apply Assumption 2 and Γ to packets departing switch queue $Q^{out}_{x_k}$. Considering packet error rates, we have the arrival rate of events at subscriber s_i matching subscription $r_j = (s_i, v_j, U_{r_j})$ as:

$$\Gamma\left(\lambda^{out}_{x_k,r_j}, b_{s_i}, s_i\right) = \lambda^{sub}_{r_j} = \left(1 - z_{b_{s_i},s_i}\right) \lambda^{out}_{x_k,r_j}$$

3.2 End-to-end Analytical Model

We now leverage the above queueing network to derive theoretical performance results. This analysis, the accuracy of which we validate in §5.2, enables FireDeX to tune the data exchange performance characteristics of end-to-end event response time and delivery success rate. To calculate Δ_{r_j}, the end-to-end response time of events for subscription r_j, we calculate the propagation and queueing delays at each layer. Note that the queueing delay in our model captures the real-world processing and network transmission delays.

To simplify our analysis, we exploit the local nature of our target scenario and consider only a single broker (b_k) in the remainder of this section. Future work will explore relaxing this assumption and extending this analysis to include the more general scenario of a distributed broker network enabled by our queueing network model above. By the above assumption, we calculate the per-subscription end-to-end response time metric as:

$$\Delta_{r_j} = \mathrm{E}\left[\Delta^{prop}_{p_i,b_k} + \Delta^{um}_{x_{p_i}}\right] + \Delta_{b_k} + \Delta^{prop}_{b_k,s_i} + \Delta_{x_{s_i}} \tag{2}$$

where $\Delta^{prop}_{b_k,h_j \in H}$ is the *propagation delay* (i.e. physical network latency) between the broker and another host h_j (b_k or s_i). $\Delta^{um}_{x_{p_i}}$ and $\Delta_{x_{s_i}}$ are the transmission delays of packets passing through switches x_{p_i} and x_{s_i} respectively. Δ_{b_k} is the processing delay of events passing through b_k.

We must estimate the heterogeneous propagation delays for this subscription's events from each possible publisher on topic v_j i.e. $\{p_i \in P_{b_k} : v_j \in V_{p_i}\}$. By our single broker assumption, we have this as the expected delay from any such publisher to broker b_k. In the same manner, we estimate the queueing delay at the intermediate switch x_{p_i}. Therefore, we have:

$$\mathrm{E}\left[\Delta^{prop}_{p_i,b_k} + \Delta^{um}_{x_{p_i}}\right] = \sum_{\{p_i \in P_{b_k} : v_j \in V_{p_i}\}} \frac{\Delta^{prop}_{p_i,b_k} + \Delta^{um}_{x_{p_i}}}{|\{p_i \in P_{b_k} : v_j \in V_{p_i}\}|}$$

The average response time of (2) includes queueing delays at each layer of FireDeX. Based on the queueing network representing FireDeX (see Fig. 4), we identify the type of each queueing model and their arrival/processing/transmission rates.

At the data exchange layer we use M/M/1 queues. Based on standard solutions for M/M/1 queues [35], we have the time that an event remains in the system (i.e. queueing time + service time; also called average delay) given by:

$$\Delta_{Q_{mm1}}(\mu, \lambda) = \frac{1}{(\mu - \lambda)} \tag{3}$$

At the network layer, we use three different types of queueing models: *i)* the M/M/1 queue ($Q^{in}_{x_k}$); *ii)* the multi-class queue ($Q^{um}_{x_i}$) and *iii)* the non-preemptive priority and multi-class queue ($Q^{out}_{x_k}$). As already pointed out, we model the transmission of packets inside the unmanaged switch queue ($Q^{um}_{x_i}$) using a multi-class queue (each class corresponds to the topic of an event encapsulated within a packet). Based on standard solutions [35], the average delay for a particular subscription r_k is given by:

$$\Delta_{Q_{mcl}}(\mu, \lambda, r_k) = \frac{1}{\mu_{r_k} - \mu_{r_k} \sum_{r_j \in R} \lambda_{r_j}/\mu_{r_j}} \tag{4}$$

where $\lambda = \{\lambda_{r_j} : r_j \in R\}$ and $\mu = \{\mu_{r_j} : r_j \in R\}$.

Finally, the SDN switch is modeled using the non-preemptive priority and multi-class queue (Q_{x_k}). Hence, the average delay of packets for r_k assigned with y_j is given by:

$$\Delta_{Q_{mclpr}}(\mu, \lambda, r_k, y_j) = \frac{L_{r_k,y_j}(\lambda, \mu)}{\lambda_{r_k}} \tag{5}$$

where $\lambda = \{\lambda_{r_j} : r_j \in R\}$, $\mu = \{\mu_{r_j} : r_j \in R\}$ and L_{r_k,y_j} is the number of events matching subscription r_k with assigned priority y_j (where $\Phi \circ \Psi(r_k) = y_j$) in the system (queue + server) of Q_{mclpr}. We omit the proof of (5) due to space constraints, but the analysis is similar to that of Section 3.4.2 in [26].

By relying on the above analytical models, we calculate the average delay of events for any subscription r_j at each node and layer of the FireDeX queueing network according to (2).

Data Exchange: at this layer the average delay at b_k (Δ_{b_k}) is given by calculating the queueing delay of events matching r_j at both inbound ($Q^{in}_{b_k}$) and outbound ($Q^{out}_{b_k,s_i}$) queues – i.e., $\Delta_{b_k} = \Delta_{Q^{in}_{b_k}} + \Delta_{Q^{out}_{b_k,s_i}}$. Both queues are of M/M/1 type. For $Q^{in}_{b_k}$, the incoming rate of events is $\lambda^{in}_{b_k}$ and its service rate is $\mu^{in}_{b_k}$; for $Q^{out}_{b_k,s_i}$ the incoming rate of events is $\lambda^{thru}_{b_k,s_i}$ and the service rate is $\mu^{out}_{b_k,r_j}$. Hence, we apply (3) to determine:

$$\Delta_{b_k} = \Delta_{Q_{mm1}}\left(\mu^{in}_{b_k}, \lambda^{in}_{b_k}\right) + \Delta_{Q_{mm1}}\left(\mu^{out}_{b_k,r_j}, \lambda^{thru}_{b_k,s_i}\right) \tag{6}$$

Network: at this layer the average delay ($\Delta^{um}_{x_i}$) at the unmanaged switch x_i (*publishers-broker* link) is given by calculating the queueing delay of packets matching r_k at the multi-class $Q^{um}_{x_i}$ queue. Hence, using the analytical model of (4) such a delay is given by:

$$\Delta^{um}_{x_i} = \Delta_{Q_{mcl}}\left(\{\mu^{um}_{x_i,v_j} : v_j \in V\}, \{\lambda^{pub}_{p_i,v_j} : p_i \in P_{x_i}, v_j \in V_{p_i}\}, r_k\right)$$

At the SDN switch x_k (*broker-subscribers* link) the average delay (Δ_{x_k}) is given by estimating the queueing delay for packets matching r_j at both inbound ($Q^{in}_{x_k}$) and outbound ($Q^{out}_{x_k}$) queues – i.e., $\Delta_{x_k} = \Delta_{Q^{in}_{x_k}} + \Delta_{Q^{out}_{x_k}}$. In the M/M/1 queue $Q^{in}_{x_k}$ packets arrive at a per-flow rate $\lambda^{in}_{x_k,f_j}$ and are served with rate $\mu^{in}_{x_k}$. Hence, by applying (3), $\Delta_{Q^{in}_{x_k}} = \Delta_{Q_{mm1}}\left(\mu^{in}_{x_k}, \lambda^{in}_{x_k,f_j}\right)$.

The outbound queue ($Q^{out}_{x_k}$), a multi-class and non-preemptive priority queue, has a per-subscription packet arrival rate $\lambda^{thru}_{x_k,r_j}$. Its service rates $\mu^{out}_{x_k,r_j}$ capture the specific event/packet size of the corresponding $r_k = (s_i, v_j, U_{r_j})$. Hence, we apply (5) to find:

$$\Delta_{Q^{out}_{x_k}} = \Delta_{Q_{mclpr}}\Big(\{\mu^{out}_{x_k,r_j} : r_j \in R_{x_k}\}, \{\lambda^{thru}_{x_k,r_j} : r_j \in R_{x_k}\}, r_k, \Phi \circ \Psi(r_k)\Big) \tag{7}$$

4 DATA EXCHANGE CONFIGURATION ALGORITHMS

The core algorithms of FireDeX leverage the above analytical model to configure the SDN-enabled data exchange. Considering current system state and information requirements, they assign priorities and preemptive drop rates to subscriptions (i.e. via $\Phi \circ \Psi$, Ω) in order to maximize subscriber-defined *utility functions*.

4.1 Utility Functions

To capture the relative value of information for different subscriptions, we propose using *utility functions*. Subscribers include a utility function with their subscriptions. They depend on the rate of successful event delivery Ξ_{r_j}. The overall utility for a subscriber depends on each of its subscriptions' utilities and is defined as:

$$U_{s_i} = \sum_{r_j \in R_{s_i}} U_{r_j}(\Xi_{r_j})$$

Let $\widehat{U_{r_j}}$ be a subscription's maximum achievable utility: delivering the maximum number of events under ideal network conditions (i.e. no loss, minimal latency, no other traffic to contend with).

To further capture the relative value of information between each subscriber, we consider an overall utility of all subscribing first responders. Each subscriber may define different utility functions to capture the fact that each of their needs vary (e.g. the IC may require more situational awareness than individual FFs). We define the overall utility of the configuration for all subscribers as a sum over each individual subscriber's utility:

$$U = \sum_{s_i \in S} U_{s_i}$$

To model heterogeneous information requirements in our experiments, we generate different utility functions for each subscription. We define the base utility function as:

$$U_{r_j}(\Xi_{r_j}) = \alpha_{r_j} \log(1 + \Xi_{r_j}) \tag{8}$$

Where the utility weight α_{r_j} is varied for each subscription.

4.2 Priority Assignment Algorithm

FireDeX leverages the above quantified utility metrics to assign priorities for each data flow in a manner that aims to maximize the overall utility. We decouple the assignment of priorities from that of drop rates for two reasons. Prioritization ensures the most important events get through *first*, but it does not necessarily provide guarantees about *how much* data is delivered. Hence, we first assign the priorities and then optimally set the preemptive drop rates to tune bandwidth usage for the network flows in each priority class. Second, this decoupling allows us to explore different policies in these two spaces independently.

Because the assignment of discrete priorities to maximize utility is non-trivial, we propose a heuristic to approximate a solution. It first ranks subscriptions according their maximum utility $\widehat{U_{r_j}}$ scaled by the corresponding required bandwidth. This metric essentially measures *information value per unit bandwidth* and lets FireDeX consider that some high-value subscriptions may consume a lot of network resources. We define this metric as:

$$\frac{\widehat{U_{r_j}}}{G_{v_j}\lambda^{notify}_{b_k,r_j}} \tag{9}$$

To approximate a solution to the priority-assignment problem, we propose the following greedy approach for each subscriber s_i:

(1) Sort the subscriptions $r_j \in R_{s_i}$ by (9)
(2) Split this list into $|F_{s_i}|$ sub-lists of approximately equal size
(3) Assign $\Psi(r_j) = F_{s_i}(k)$ for each $r_j \in$ sub-list number k
(4) Split the list of flows F_{s_i} into approximately $|Y|$ sub-lists of approximately equal size
(5) Assign $\Phi(f_j) = y_k$ for each $f_j \in$ sub-list number k

Note that this splitting up of lists handles unequally-sized splits (e.g. $|F_{s_i}| > |Y|$) by preferring higher priorities first.

This priority assignment ensures delivery of the highest-priority events if possible. However, an overloaded system will fill switch buffers and lead to high delay and loss of lower-priority events. Hence, we apply preemptive drop rates to avoid such a case.

4.3 Ensuring Queue Stability via Preemptive Drop Rates

Given the above priority assignment, FireDeX further fine-tunes the subscriptions' successful notification rate Ξ_{r_j}. Based on requested subscription utility functions and current network state (e.g. bandwidth constraints), it applies a preemptive packet dropping policy. This improves overall utility of the system's configuration by essentially allocating available bandwidth to the network flows. Crucially, this bandwidth allocation also ensures *queue stability* throughout the network. That is, if packets arrive at the switches' inbound queues too quickly, the forwarding queues will grow in size until the buffers fill up and packets are dropped. To prevent the dropping of high-value events, FireDeX preemptively drops lower-priority packets. The algorithms presented here determine with what probability packets of each network flow should be dropped ($\Omega(f_j)$) to improve situational awareness while ensuring queue stability.

Not only does ensuring queue stability improve system performance, it also satisfies conditions necessary for our analytical model's results to prove accurate. Let $\rho_Q = \frac{\lambda}{\mu}$ be the server utilization (i.e. probability that the server is busy) of the corresponding queue (e.g. $Q^{out}_{x_k}$). By [26] the system remains *unsaturated* (i.e. queue stability is ensured) when $\rho_Q < 1$. For FireDeX's M/M/1 queues (i.e., $Q^{in}_{b_k}$, $Q^{out}_{b_k,s_i}$, $Q^{in}_{x_k}$) we define: $\rho_{Q^{in}_{b_k}} = \frac{\lambda^{in}_{b_k}}{\mu^{in}_{b_k}}$, $\rho_{Q^{out}_{b_k,s_i}} = \frac{\lambda^{thru}_{b_k,s_i}}{\mu^{out}_{b_k,r_j}}$ and $\rho_{Q^{in}_{x_k}} = \frac{\lambda^{in}_{x_k,f_j}}{\mu^{in}_{x_k}}$. FireDeX's multi-class queues $Q^{um}_{x_i}$ and $Q^{out}_{x_k}$ have per-topic and per-subscription arrival and service rates respectively. We have their server utilization as:

$$\rho_{Q^{um}_{x_i}} = \sum_{P_{x_i}} \sum_{v_j \in V_{p_i}} \frac{\lambda^{pub}_{p_i,v_j}}{\mu^{um}_{x_i,v_j}} \tag{10}$$

$$\rho_{Q^{out}_{x_k}} = \sum_{r_j \in R_{x_k}} \frac{\lambda^{thru}_{x_k,r_j}}{\mu^{out}_{x_k,r_j}} \tag{11}$$

To improve successful delivery rate while ensuring queue stability, we propose several algorithms of increasing sophistication below. Note that these algorithmic formulations currently only consider the outbound queue of the SDN switches for this constraint as tuning the drop rates only affects $\rho_{Q^{out}_{x_k}}$. Also recall that this queue captures the bottleneck bandwidth of the network route from broker to subscriber. Future work will explore simultaneously balancing the load across data exchange brokers to also ensure stability of their queues within our model.

Each algorithm makes use of a parameter $\tilde{\rho}$ in tuning the system's tolerance to approaching, but never exceeding, the queue saturation point of $\rho_{Q^{out}_{x_k}} = 1$. Clearly, to satisfy the strict inequality $\rho_{Q^{out}_{x_k}} < 1$ we must have $\tilde{\rho} > 0$. Setting $\tilde{\rho}$ even higher provides ample buffer within the SDN switch queues for resilience to temporary notification rate spikes that might otherwise lead to queue saturation. Even if this condition is just barely satisfied (e.g. $\tilde{\rho} = 10^{-10}$), queues will still grow quite large and thereby cause high delay.

Therefore, the following drop rate policies set Ω such that:

$$\rho_{Q^{out}_{x_k}} = 1 - \tilde{\rho} \tag{12}$$

Flat drop rates: this simple naive policy sets all drop rates equal to satisfy Eq. (12) by solving Eq. (11) for a parameter β such that:

$$\Omega(f_j) = \beta \tag{13}$$

Linear drop rates: this more information value-aware policy sets the drop rates for each network flow according to its assigned priority level. It solves Eq. (11) for a parameter β that satisfies Eq. (12) with drop rates set to:

$$\Omega(f_j) = \beta\Phi(f_j) \tag{14}$$

Exponential drop rates: similar to *Linear*, this policy sets drop rates according to priority level. It solves Eq. (11) for a parameter β that satisfies (12) with drop rates set to:

$$\Omega(f_j) = 1 - \beta^{-\Phi(f_j)} \tag{15}$$

Optimized drop rates: the following convex optimization formulation assigns drop rates to maximize overall utility. Given the previously-assigned priorities as input, FireDeX assigns drop rates by solving the following convex optimization problem:

$$\begin{aligned} \text{maximize} \quad & U \\ \text{subject to} \quad & \Omega(f_j) \in [0,1], \forall f_j \in F \\ & \rho_{Q^{out}_{x_k}} \leq 1 - \tilde{\rho},\ \forall x_k \in X \end{aligned} \tag{16}$$

Note that the second constraint ensures available bandwidth constraints are met (i.e. queue stability) according to the $\tilde{\rho}$ parameter.

As long as the chosen utility functions are concave (e.g. logarithm) within the feasible domain of assigned drop rates, then (16) can be expressed as a convex optimization problem and efficiently solved. Hence, we define such a utility function, such as that given in (8), that takes as input our analytical model for determining $\lambda^{sub}_{r_j}$. Because this is an affine function over the drop rates, we can optimally solve for drop rates that maximize the overall system utility. We used CVXPY [1, 20] to implement this approach in the FireDeX middleware.

5 EXPERIMENTAL RESULTS

FireDeX uses the analytical model given in §3.2 to estimate end-to-end response times and success rates for event notifications to interested subscribers. We validate this analysis by using and extending an open source queueing simulator to represent our proposed system. We compare the subscribers' end-to-end response times given by the analytical model with those given by the simulation. Note that we omit the trivial results for validating success rates. In order to improve the figures' legibility, we did not include error bars in our plots as the simulation results' confidence intervals are very small (less than two orders of magnitude from the corresponding mean values presented in the plots). We further validate the model's accuracy under greater numbers of subscribers.

After validating our model, we then use it in combination with the simulator to evaluate the FireDeX approach for a given configuration. In particular, we compare our approach's efficacy with that of an unprioritized system and evaluate the trade-off between response times and success rates. We use our proposed priority-assignment algorithm, which we call *bandwidth-adjusted-prio*, and the *exponential* drop rate policy. Subsequently, we utilize the analytical model only to compare different algorithms' ability to maximize the overall value of information captured.

Table 2: Default parameters for our experimental configurations.

		#topics ($\|V\|$)	pub rate ($\lambda^{pub}_{p_i,v_j}$)	event size (G_{v_j})	#subscriptions ($\|R_{s_i}\|$)	utility weight (α_{r_j})
DX params	**Telemetry data**	140	Exp($\frac{1}{6}$) ∈ [4,7]	Exp($\frac{1}{110}$) ∈ [90,500]	70	Exp($\frac{1}{0.5}$) ∈ [0.01,2]
	Async events	60	Exp($\frac{1}{4}$) ∈ [3,5]	Exp($\frac{1}{800}$) ∈ [500,1100]	42	Exp(1) ∈ [0.1,4]

	#subscribers ($\|S\|$)	#publishers ($\|P\|$)	#flows ($\|F_{s_i}\|$)	#priorities ($\|Y\|$)	bandwidth (w_{s_i})	ρ tolerance ($\tilde{\rho}$)
Net params	10	160	9	9	80 Mbps	0.1

5.1 Experimental Setup

We developed a Python-based framework that models the real-world scenario given in §2.1 and provides input data for our simulations. We configure it to consider two classes of topics that represent events: sensor telemetry readings published periodically and asynchronously-published notifications that indicate real-world phenomena detected from analysis of raw sensor readings. This framework leverages the probability distributions and parameters given in Table 2 to generate random configurations for each publisher, subscriber, the broker, and the network. For example, it selects the publication rate and packet size of events from the given distributions. Note that we bound these values to maintain more realistic parameters by redrawing a new one when it lies outside the given range. Note that the actual topics published and subscribed to are chosen uniformly at random from those available.

The model presented in §3 generically captures a very wide range of scenarios and system configurations. To reduce the number of variables we explore in our experiments, we only simulate a single (i.e. last-hop) SDN switch between the broker and subscriber. Recall that this represents the bottleneck bandwidth and transmission delays. Also note that propagation delay and error rates are typically modeled as constant values. Hence, we ignore them for these experiments to focus instead on the variable delays our model aims to capture. Furthermore, we consider only 9 priority classes due to practical limitations of many existing network traffic and data exchange management systems. For example, Linux TC [37] and AMQP 0.9.1 [2] only support 8 and 10 priority queues respectively.

Queueing network simulator: after generating these configuration parameters for a single instance of a scenario, the above framework feeds them into a simulator to drive its pseudo-random number generators. That is, these parameters correspond to the expected values of the probability distributions from which the simulator draws the actual individual publications' arrival times and packet sizes. Note that we use exponential distributions in order to maintain our assumption of Poisson arrival/service rates. This simulator extends JINQS [23], a Java simulation library for multiclass queueing networks. JINQS provides a suite of primitives that allow developers to rapidly build simulations for a wide range of queueing networks. We leverage this power and extend JINQS to: *i)* represent the queueing network introduced in Fig. 4; *ii)* implement our new multi-class and non-preemptive priority queueing model; *iii)* simulate pub/sub interactions using a set of configuration parameters provided by our Python-based framework. To evaluate FireDeX, we generate parameters and run the simulator 10 times for each configuration and then average across these. Each run generates approximately 6,500,000 publications to accurately calculate per-subscription response times and success rates.

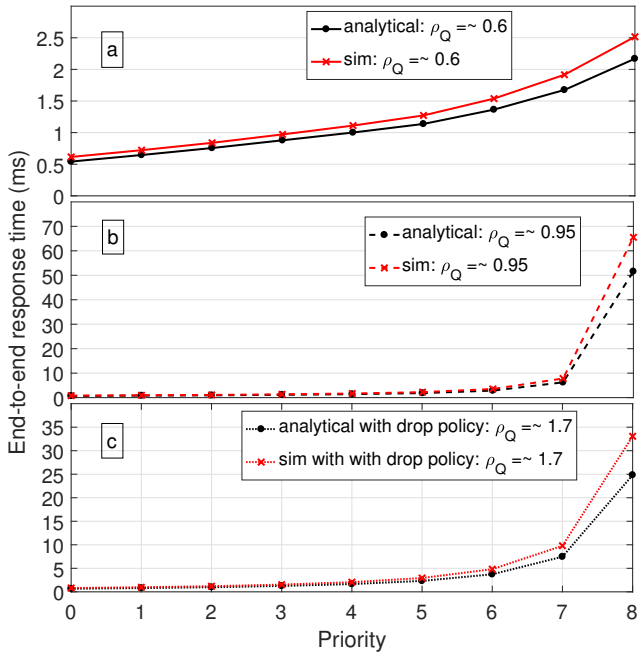

Figure 5: Analytical vs. simulation end-to-end response times for varying traffic loads ($\rho_{Q^{out}_{x_k}}$).

5.2 Validating our Queueing Network Model

To prove the accuracy of the theoretical analysis we developed in §3.2, we now compare its estimated performance metrics with those calculated from the aforementioned simulator.

5.2.1 Varying traffic loads. Recall that the SDN switch's outbound queue (shown in Fig. 4) captures the bottleneck bandwidth of the network route from broker to subscriber. FireDeX uses the corresponding server utilization ($\rho_{Q^{out}_{x_k}}$) to decide the bandwidth tuning by assigning drop rates. Therefore, we parameterize the simulated queueing network to vary the system's network traffic load: *a)* medium-load conditions ($\rho_{Q^{out}_{x_k}} = 0.6$); *b)* high-load conditions (i.e. close to saturation – $\rho_{Q^{out}_{x_k}} = 0.95$); *c)* overloaded conditions (i.e. saturated – $\rho_{Q^{out}_{x_k}} = 1.7$). Note that the saturated case (3rd) corresponds to the default parameters in Table 2. To achieve the medium-load (1st) and high-load (2nd) cases, we set the number of subscriptions for each topic class respectively: *i)* 21,15; and *ii)* 42,24.

Fig. 5 shows the results of these experiments according to assigned priority class and averaged across all topics, subscribers, etc. Comparing the curves of both the simulated measurements and the analytical results obtained by Eq. (2) reveal our model's high accuracy. We notice small differences for events with lower priority levels. In particular, note priority level 8's differences: 0.35 ms in

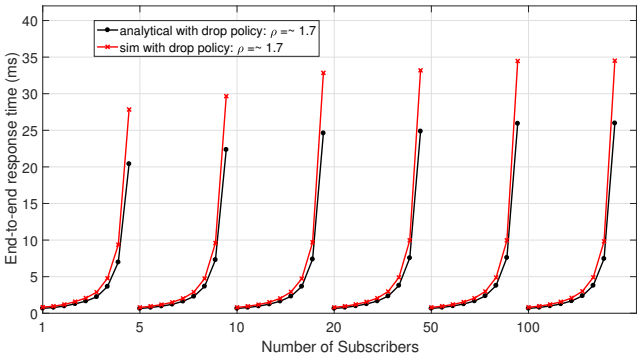

Figure 6: Analytical vs. simulation end-to-end response times for varying numbers of subscribers.

Fig. 5a, 13.98 ms in Fig. 5b and 8.24 ms in Fig. 5c. Because the system approaches saturation in Figs. 5b and 5c, we deem these results acceptable. In Fig. 5c, FireDeX uses our drop policy mechanism to drop packets at the SDN switch and return the system to below saturation (i.e. $\rho_{Q_{x_k}^{out}} = 0.9$ by using $\tilde{\rho} = 0.1$).

5.2.2 Scaling up number of subscribers. We now validate our analytical model's accuracy under varying numbers of subscribers: $|S| = 1, 10, 20, 50, 100$. To maintain the same degree of system saturation (i.e. $\rho_{Q_{x_k}^{out}} = 1.7$), we increase the bandwidth proportional to the number of subscribers: $w_{x_k,s_i} = 8Mbps$. We keep all other parameters according to Table 2. According to these parameters, we measure the simulated mean response times and plot them vs. those calculated using (2) in Fig. 6. Note the curve for each number of subscribers that shows response time increasing with the priority class. From this comparison, we see that the absolute deviation between the two curves does not exceed 10 ms across all priority levels. Therefore, our model remains accurate even with higher numbers of subscribers.

5.3 Evaluating the FireDeX Approach

We now compare our approach's efficacy with that of an unprioritized system and a system without preemptive packet drops.

We apply a buffer capacity of k packets for the simulator's SDN switch outbound queue. This models a real-world switch dropping packets when the buffer fills up. It drops the incoming packet if its priority class is less than or equal to the lowest priority class of those in the buffer. Otherwise, it evicts lower-priority packets to make space in the buffer. We set $k = 2000$ based on reported buffer sizes of various real-world SDN switches[16]. Additionally, we configure this queue in 3 different ways:

i) No priority assignment or drop policy features (i.e. a simple switch that treats all packets identically and only drops incoming ones when its buffer has filled up)
ii) Priority assignment only (i.e. no drop rates)
iii) Both priorities and drop rates (i.e. the complete FireDeX approach)

These experiments use the parameters given in Table 2. Figs. 7 and 8 show the success rates and end-to-end response times, respectively. Configuration (i) results in a 58% average success rate and 0.9 sec average response time regardless of assigned priority.

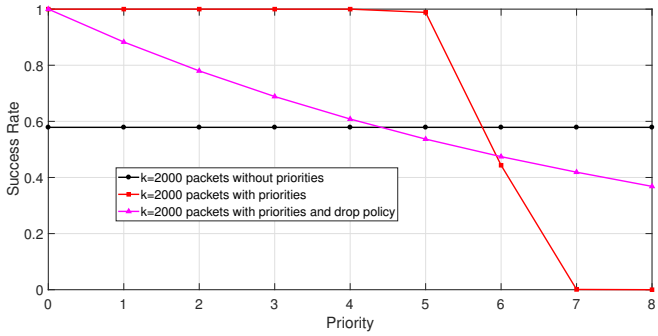

Figure 7: Comparing success rates for no priorities (i.e. single switch buffer), priorities only, and an added drop policy.

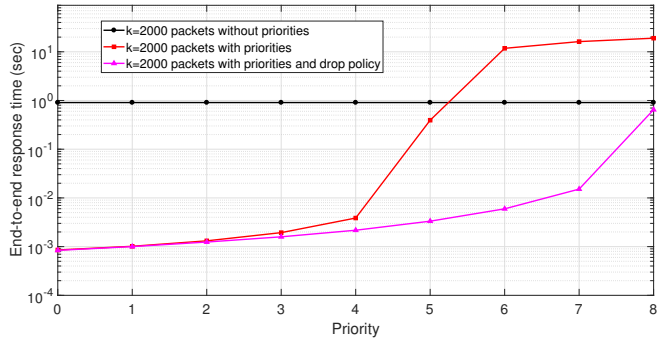

Figure 8: Comparing response times for no priorities (i.e. single switch buffer), priorities only, and an added drop policy.

The configuration (ii) experiments used the algorithm we proposed in §4.2, which we call *bw-adjusted-prio*, for assigning priorities to each network flow (i.e. to their contained subscriptions and associated packets). The results demonstrate that priority assignment significantly improves both response times and success rates for higher priority subscriptions. In particular, subscriptions with priorities 0-4 have a response time less than 4 ms and 100% success rate. However, the success rate of lower priority subscriptions suddenly decreases while the response time increases to the order of seconds. For instance, those with priority 6 have a 45% success rate and 11 sec. response time. Additionally, subscriptions with priorities 7,8 have very low success rates (almost all packets dropped), while those events successfully delivered have a high response time of 20 sec.

The results for configuration (iii) demonstrate how applying drop rates further improves response time to the order of milliseconds. Specifically, priority 0-6 subscriptions have a response time under 6 ms, whereas those with priority 8 have a response time of 647 ms. The most important subscriptions (i.e. priority 0) have 100% success rate. The FireDeX *exponential* drop rate policy smoothly decreases the success rate proportional to the priority level. This demonstrates our approach to controlling the success rate based on a subscriber's available bandwidth in order to achieve lower response times. Next, we compare the level of overall utility achieved using the various priority assignment and drop rate algorithms that base their configurations on the subscriptions' utility functions.

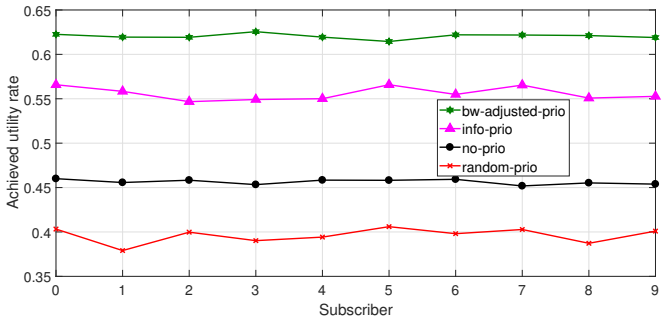

Figure 9: Comparing priority-assignment algorithms with other naive approaches.

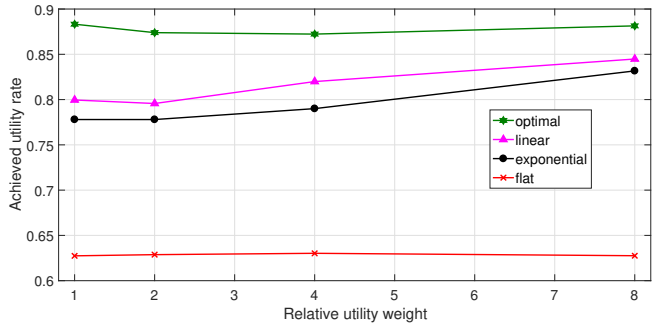

Figure 10: Comparing drop rate policies by varying the utility of one subscription class relative to the other.

5.4 Comparing Prioritization & Drop Rate Algorithms for Situational Awareness

We now compare different algorithms' ability to maximize the value of information captured for a given configuration. We measure this as the *achieved utility rate*: the ratio of a subscription's max utility ($\widehat{U_{r_j}}$) to achieved utility averaged over all subscriptions.

Fig. 9 compares our proposed priority-assignment algorithm (named *bw-adjusted-prio*) with: no priorities (i.e. a single queue; named *no-prio*), randomly-assigned priorities (i.e. to show the poor performance resulting from inaccurate priority assignment; named *random-prio*), and a naive version of the proposed greedy algorithm that, when ranking subscriptions, considers only max utility without scaling by the bandwidth requirement (named *info-prio*). This figure shows how leveraging network awareness when assigning priorities improves the achieved utility rate by 12% vs. the naive version and 36% vs. no prioritization. Note that we do not assign drop rates for the priority algorithms comparison. Instead, we configure the simulator to drop packets once the buffers fill up in order to compare the priority-assignment algorithms only. With drop rates, FireDeX improves the value of exchanged data by 42% vs. prioritization only and 94% vs. no prioritization.

We then compare the four drop rate-assignment algorithms outlined in §4.3 in conjunction with the best-performing *bw-adjusted-prio* algorithm. To demonstrate FireDeX's ability to improve situational awareness for heterogeneous data and information requirements, our experiments varied the utilities of each topic class relative to the other. We increase the random variable distribution parameters used to generate the utility weights (α_{r_j}) of one topic class relative to the other (i.e. telemetry data vs. the higher-utility higher-bandwidth lower-publication frequency asynchronous events). Fig. 10's x-axis shows the factor we scale up the asynchronous event class's α_{r_j} by as compared with the Table 2 defaults. These results demonstrate the optimization-based algorithm's superiority in capturing the most overall utility (i.e. it maximizes situational awareness) given particular network conditions.

6 CONCLUDING REMARKS

In this paper we presented FireDeX: an extensible middleware for managing mission-critical IoT data exchange. Our proposed SDN-enabled 3-layer approach bridges application-specified information requirements, generic data exchange capabilities, and physical network characteristics. Its algorithms assign priorities to subscriptions and tune their bandwidth allocation (i.e. via packet drop rates) to maximize overall situational awareness. Our experimental results showed that this approach greatly improves the performance in terms of information value captured as well as end-to-end delays.

The cross-layer queueing theoretic model serves as a sound theoretical framework underpinning the FireDeX middleware and enables the analysis used to drive its algorithms. Its modular design supports composition of alternative queueing models. Hence it lays the groundwork for many potential extensions and alterations, some of which we will address in future work. For example, we aim to relax some of the Assumptions in §3: considering multiple switches and physical network routes in managing subscriber data flows; considering non-Poisson arrival and service rates by using e.g. G/G/1 queues; converting larger events into many packets (or many events into one packet) by applying the queueing theoretic concept of *batch arrivals* [50]; configuring an entire broker network rather than just the BMS's local broker.

To study performance issues arising from a real-world implementation, we are currently developing a Python-based middleware prototype. It incorporates the algorithm implementations with RESTful services for managing an MQTT-SN[30] data exchange broker and clients through interactions with an SDN controller. We are also developing a SFF situational awareness dashboard. Upon completion of this prototype, we will deploy it in a smart building on the UCI campus and integrate it with our existing IoT testbed privacy-preserving smart building system [8, 40]. This will enable studies into: how closely our analysis models a real-world system; managing dynamic conditions such as subscriber churn; considering SDN overhead (e.g. flow table space required, delay for configuration changes and statistics collection); conducting IoT-enhanced demonstration drills with fire fighters.

ACKNOWLEDGMENTS

This work was supported by: NSF award CNS 1450768, NIST Award # 70NANB17H285, DARPA agreement # FA8750-16-2-0021, the Inria@SiliconValley International Lab, and the research associate team MINES. The authors of this paper also thank the authors of the Res. Roadmap for their valuable contributions to identifying key technological and operational research challenges for SFF.

REFERENCES

[1] Steven Diamond Akshay Agrawal, Robin Verschueren and Stephen Boyd. 2018. A Rewriting System for Convex Optimization Problems. *Journal of Control and Decision* 5, 1 (2018), 42–60.

[2] "AMQP Working Group 0-9-1". 2008. http://www.amqp.org/specification/0-9-1/amqp-org-download.

[3] Namwon An, Taejin Ha, Kyung-Joon Park, and Hyuk Lim. 2016. Dynamic priority-adjustment for real-time flows in software-defined networks. In *Telecommunications Network Strategy and Planning Symposium (Networks), 2016 17th International*. IEEE, 144–149.

[4] Bharathan Balaji et al. 2016. Brick: Towards a Unified Metadata Schema For Buildings. In *BuildSys*. http://doi.acm.org/10.1145/2993422.2993577

[5] Cory C Beard and Victor S Frost. 2004. Prioritization of emergency network traffic using ticket servers: A performance analysis. *Simulation* (2004).

[6] S. Behnel, L. Fiege, and G. Muhl. 2006. On quality-of-service and publish-subscribe. In *ICDCS Workshops*. IEEE.

[7] P. Bellavista, A. Corradi, and A. Reale. 2014. Quality of Service in Wide Scale Publish-Subscribe Systems. *IEEE Communications Surveys & Tutorials* (2014).

[8] K. Benson, C. Fracchia, G. Wang, Q. Zhu, S. Almomen, J. Cohn, L. DâĂŹarcy, D. Hoffman, M. Makai, J. Stamatakis, and N. Venkatasubramanian. 2015. SCALE: Safe community awareness and alerting leveraging the internet of things. *Communications Magazine, IEEE* (Dec 2015).

[9] K. E. Benson, G. Wang, N. Venkatasubramanian, and Y. Kim. 2018. Ride: A Resilient IoT Data Exchange Middleware Leveraging SDN and Edge Cloud Resources. In *2018 IEEE/ACM Third International Conference on Internet-of-Things Design and Implementation (IoTDI)*. 72–83. https://doi.org/10.1109/IoTDI.2018.00017

[10] Pankaj Berde et al. 2014. ONOS. In *Proceedings of HotSDN '14*. http://dl.acm.org/citation.cfm?doid=2620728.2620744

[11] Sukanya Bhowmik, Muhammad Adnan Tariq, Boris Koldehofe, Frank Durr, Thomas Kohler, and Kurt Rothermel. 2017. High performance publish/subscribe middleware in software-defined networks. *IEEE/ACM Transactions on Networking (TON)* 25, 3 (2017), 1501–1516.

[12] Andreas Blenk, Arsany Basta, Martin Reisslein, and Wolfgang Kellerer. 2015. Survey on Network Virtualization Hypervisors for Software Defined Networking. *CoRR* abs/1506.07275 (2015). arXiv:1506.07275 http://arxiv.org/abs/1506.07275

[13] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, and David Walker. 2014. P4: Programming Protocol-independent Packet Processors. *SIGCOMM Comput. Commun. Rev.* 44, 3 (July 2014), 87–95. https://doi.org/10.1145/2656877.2656890

[14] Georgios Bouloukakis, Nikolaos Georgantas, Ajay Kattepur, and Valérie Issarny. L'Aquila, Italy, Apr. 2017. Timeliness Evaluation of Intermittent Mobile Connectivity over Pub/Sub Systems. In *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering*. https://doi.org/10.1145/3030207.3030220

[15] Georgios Bouloukakis, Ioannis Moscholios, Nikolaos Georgantas, and Valérie Issarny. Paris, France, May 2017. Performance Modeling of the Middleware Overlay Infrastructure of Mobile Things. In *IEEE International Conference on Communications*. https://doi.org/10.1109/ICC.2017.7997451

[16] "Buffer requirements". 2008. https://people.ucsc.edu/~warner/buffer.html.

[17] S.T. Bushby, H. M. Newman, and M. A. Applebaum. 1999. *NISTIR 6392 GSA Guide to Specifying Interoperable Building Automation and Control Systems Using ANSI/ASHRAE Standard 135-1995, BACnet*. National Institute Of Standards and Technology.

[18] Kaifei Chen, Siyuan He, Beidi Chen, John Kolb, Randy H. Katz, and David E. Culler. 2015. BearLoc: A Composable Distributed Framework for Indoor Localization Systems. In *Workshop on IoT-Sys*. https://doi.org/10.1145/2753476.2753478

[19] Stephen Dawson-Haggerty, Andrew Krioukov, Jay Taneja, Sagar Karandikar, Gabe Fierro, Nikita Kitaev, and David Culler. 2013. BOSS: Building Operating System Services. In *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation (nsdi'13)*. USENIX Association, Berkeley, CA, USA, 443–458. http://dl.acm.org/citation.cfm?id=2482626.2482669

[20] Steven Diamond and Stephen Boyd. 2016. CVXPY: A Python-Embedded Modeling Language for Convex Optimization. *Journal of Machine Learning Research* 17, 83 (2016), 1–5.

[21] A. El-Mougy, M. Ibnkahla, and L. Hegazy. 2015. Software-defined wireless network architectures for the Internet-of-Things. In *2015 IEEE 40th Local Computer Networks Conference Workshops (LCN Workshops)*. 804–811. https://doi.org/10.1109/LCNW.2015.7365931

[22] G. Faraci, A. Lombardo, and G. Schembra. 2017. A building block to model an SDN/NFV network. In *2017 IEEE International Conference on Communications (ICC)*. 1–7. https://doi.org/10.1109/ICC.2017.7997430

[23] Tony Field. 2006. JINQS: An extensible library for simulating multiclass queueing networks, v1. 0 user guide.

[24] Chuan Heng Foh, Yu Zhang, Zefeng Ni, Jianfei Cai, and King Ngi Ngan. 2007. Optimized cross-layer design for scalable video transmission over the IEEE 802.11 e networks. *IEEE Transactions on Circuits and Systems for Video Technology* (2007).

[25] Phillipa Gill, Zongpeng Li, Anirban Mahanti, Jingxiang Luo, and Carey Williamson. 2008. Network information flow in network of queues. In *MASCOTS 2008*. IEEE.

[26] Donald Gross, John Shortle, James Thompson, and Carl Harris. 2008. *Fundamentals of queueing theory*. John Wiley & Sons, 4th edition.

[27] Hassan Halabian, Ioannis Lambadaris, and Chung-Horng Lung. 2010. Network capacity region of multi-queue multi-server queueing system with time varying connectivities. In *ISIT*. IEEE.

[28] A. Hamins, C. Grant, N. Bryner, A. Jones, and G. Koepke. 2015. *NIST Special Publication 1191 Research Roadmap for Smart Fire Fighting*. National Institute Of Standards and Technology.

[29] Fei He, Luciano Baresi, Carlo Ghezzi, and Paola Spoletini. Tallinn, Estonia, June 2007. Formal analysis of publish-subscribe systems by probabilistic timed automata. In *International Conference on Formal Techniques for Networked and Distributed Systems*.

[30] IBM 2013. *MQTT For Sensor Networks (MQTT-SN)*. IBM.

[31] Masugi Inoue, Yasunori Owada, Kiyoshi Hamaguti, and Ryu Miura. 2014. Nerve Net: A Regional-Area Network for Resilient Local Information Sharing and Communications. In *Proceedings of the 2014 Second International Symposium on Computing and Networking (CANDAR '14)*. IEEE Computer Society, Washington, DC, USA, 3–6. https://doi.org/10.1109/CANDAR.2014.83

[32] Yaser Jararweh, Mahmoud Al-Ayyoub, Ala' Darabseh, Elhadj Benkhelifa, Mladen Vouk, and Andy Rindos. 2015. SDIoT: a software defined based internet of things framework. *Journal of Ambient Intelligence and Humanized Computing* 6, 4 (01 Aug 2015), 453–461. https://doi.org/10.1007/s12652-015-0290-y

[33] Pradeeban Kathiravelu, Leila Sharifi, and Luís Veiga. 2015. Cassowary: Middleware Platform for Context-Aware Smart Buildings with Software-Defined Sensor Networks. In *Proceedings of the 2Nd Workshop on Middleware for Context-Aware Applications in the IoT (M4IoT 2015)*. ACM, New York, NY, USA, 1–6. https://doi.org/10.1145/2836127.2836132

[34] Samuel Kounev, Kai Sachs, Jean Bacon, and Alejandro Buchmann. Orlando, FL, USA, May 2008. A methodology for performance modeling of distributed event-based systems. In *11th IEEE International Symposium on Object Oriented Real-Time Distributed Computing (ISORC)*.

[35] E. Lazowska, J. Zahorjan, S. Graham, and K. Sevcik. 1984. *Quantitative system performance: computer system analysis using queueing network models*. Prentice-Hall, Inc.

[36] Chienhung Lin, Kuochen Wang, and Guocin Deng. 2017. A QoS-aware routing in SDN hybrid networks. *Procedia Computer Science* 110 (2017), 242–249.

[37] "Linux TC". 2001. http://lartc.org/manpages/tc.txt.

[38] Ruben Mayer, Boris Koldehofe, and Kurt Rothermel. 2015. Predictable low-latency event detection with parallel complex event processing. *IEEE Internet of Things Journal* (2015), 274–286.

[39] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. 2008. OpenFlow: Enabling Innovation in Campus Networks. *SIGCOMM Comput. Commun. Rev.* 38, 2 (March 2008), 69–74. https://doi.org/10.1145/1355734.1355746

[40] S. Mehrotra, A. Kobsa, N. Venkatasubramanian, and S. R. Rajagopalan. 2016. TIPPERS: A privacy cognizant IoT environment. In *PerCom Workshops*. https://doi.org/10.1109/PERCOMW.2016.7457158

[41] H. A. Nguyen, T. V. Nguyen, and D. Choi. 2009. How to Maximize User Satisfaction Degree in Multi-service IP Networks. In *2009 First Asian Conference on Intelligent Information and Database Systems*. 471–476. https://doi.org/10.1109/ACIIDS.2009.16

[42] OASIS 2014. *MQTT Version 3.1.1*. OASIS.

[43] Ben Pfaff, Justin Pettit, Teemu Koponen, Ethan J. Jackson, Andy Zhou, Jarno Rajahalme, Jesse Gross, Alex Wang, Jonathan Stringer, Pravin Shelar, Keith Amidon, and Martín Casado. 2015. The Design and Implementation of Open vSwitch. In *NSDI*. http://dl.acm.org/citation.cfm?id=2789770.2789779

[44] Pivotal, "RabbitMQ". 2018. https://www.rabbitmq.com/.

[45] Zhijing Qin, Grit Denker, Carlo Giannelli, Paolo Bellavista, and Nalini Venkatasubramanian. 2014. A software defined networking architecture for the internet-of-things. In *IEEE NOMS*.

[46] K. Sachs, S. Kounev, and A. Buchmann. 2013. Performance modeling and analysis of message-oriented event-driven systems. *Software & Systems Modeling* (2013).

[47] Pooya Salehi, Kaiwen Zhang, and Hans-Arno Jacobsen. 2017. PopSub: Improving resource utilization in distributed content-based publish/subscribe systems. In *ACM DEBS*.

[48] Farzad Samie, Vasileios Tsoutsouras, Sotirios Xydis, Lars Bauer, Dimitrios Soudris, and Jörg Henkel. 2016. Distributed QoS Management for Internet of Things Under Resource Constraints. In *Proceedings of the Eleventh IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES '16)*. ACM, Article 9, 10 pages. https://doi.org/10.1145/2968456.2974005

[49] Arnd Schröter, Gero Mühl, Samuel Kounev, Helge Parzyjegla, and Jan Richling. 2010. Stochastic performance analysis and capacity planning of publish/subscribe systems. In *DEBS*. ACM, 258–269.

[50] DN Shanbhag. 1966. On infinite server queues with batch arrivals. *Journal of Applied Probability* 3, 1 (1966), 274–279.

[51] Deepak Singh, Bryan Ng, Yuan-Cheng Lai, Ying-Dar Lin, and Winston KG Seah. 2017. Modelling Software-Defined Networking: Switch Design with Finite Buffer and Priority Queueing. In *LCN*. IEEE.

[52] Keshav Sood, Shui Yu, and Yong Xiang. 2016. Performance analysis of software-defined network switch using $M/Geo/1$ model. *IEEE Communications Letters* (2016), 2522–2525.

[53] Muhammad Adnan Tariq, Boris Koldehofe, Sukanya Bhowmik, and Kurt Rothermel. 2014. PLEROMA: A SDN-based high performance publish/subscribe middleware. In *Proceedings of the 15th International Middleware Conference*. ACM, 217–228.

[54] Sahrish Khan Tayyaba and Munam Ali Shah. 2018. Resource allocation in SDN based 5G cellular networks. *Peer-to-Peer Networking and Applications* (2018).

[55] Jörg Thalheim, Antonio Rodrigues, Istemi Ekin Akkus, Pramod Bhatotia, Ruichuan Chen, Bimal Viswanath, Lei Jiao, and Christof Fetzer. 2017. Sieve: actionable insights from monitored metrics in distributed systems. In *Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference*. ACM.

[56] Mary Vernon, John Zahorjan, and Edward D Lazowska. 1986. *A comparison of performance Petri nets and queueing network models*. University of Wisconsin-Madison, Computer Sciences Department.

[57] Yali Wang, Yang Zhang, and Junliang Chen. 2017. Pursuing Differentiated Services in a SDN-Based IoT-Oriented Pub/Sub System. (06 2017), 906–909.

[58] Thomas Weng, Anthony Nwokafor, and Yuvraj Agarwal. 2013. BuildingDepot 2.0: An Integrated Management System for Building Analysis and Control. In *ACM BuildSys'13*. http://doi.acm.org/10.1145/2528282.2528285

[59] Yung Yi and Mung Chiang. 2008. Stochastic network utility maximisation - A tribute to Kelly's paper published in this journal a decade ago. 19 (06 2008), 421–442.

[60] Justyna Zander, Pieter J. Mosterman, Taskin Padir, Yan Wan, and Shengli Fu. 2015. Cyber-physical Systems can Make Emergency Response Smart. *Procedia Engineering* 107 (2015), 312 – 318. https://doi.org/10.1016/j.proeng.2015.06.086 Humanitarian Technology: Science, Systems and Global Impact 2015, HumTech2015.

[61] Kaiwen Zhang and Hans-Arno Jacobsen. 2013. SDN-like: The Next Generation of Pub/Sub. *CoRR* abs/1308.0056 (2013). http://arxiv.org/abs/1308.0056

[62] Kaiwen Zhang, Vinod Muthusamy, Mohammad Sadoghi, and Hans-Arno Jacobsen. 2017. Subscription covering for relevance-based filtering in content-based publish/subscribe systems. In *IEEE ICDCS*.

[63] K. Zhang, M. Sadoghi, V. Muthusamy, and H.-A. Jacobsen. 2017. Efficient covering for top-k filtering in content-based publish/subscribe systems. In *ACM/IFIP/USENIX Middleware Conference*.