NETWORK PERFORMANCE ANALYSIS AND MANAGEMENT FOR CYBER-PHYSICAL SYSTEMS AND THEIR APPLICATIONS

By

William A. Emfinger

Dissertation

Submitted to the Faculty of the

Graduate School of Vanderbilt University

in partial fulfillment of the requirements

for the degree of

DOCTOR OF PHILOSOPHY

in

Electrical Engineering

October, 2015

Nashville, Tennessee

Approved: Date:

_____ _____

_____ _____

_____ _____

_____ _____

_____ _____

The DARPA System F6, AFRL RESOS, and AFRL SURE Projects supported this work. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not reflect the views of DARPA or AFRL.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

1.1 Motivation

Cyber-Physical Systems (CPS) are defined as co-engineered interacting networks of physical and computational components. These classes of systems consist of computers running sensing and actuation control software to interact closely with a physical system in a physical environment. In these systems, the computer controls some or multiple aspects of the physical system; this control aspect tightly couples the software with the computer's physical environment. Because of this tightly coupled nature, the software not only affects the computer and its surrounding environment, but also is affected by the surrounding environment. Many different types of systems fit this CPS description: e.g. autonomous vehicles including Unmanned Aerial Vehicles, Unmanned Underwater Vehicles, autonomous cars, satellites, extra-terrestrial rovers, and embedded or wireless sensors/actuators.

Consider a quad-copter and its flight control software. The flight control software must monitor the orientation and positioning sensors of the quadcopter, estimate the state based on the sensor data, and then calculate control outputs to drive the system to a goal state. Such a system deals with many constraints, ranging from weight, power, and size constraints, to processor speed, memory capacity, and wireless operational range constraints. Despite these constraints, the flight control software must complete the sense-estimate-calculate-actuate loop within enough time to ensure that the quadcopter's state does not transition into an unstable state. Such an unstable state might be the quadcopter flipping over or crashing into an obstacle. The dynamics of the control system, i.e. the physics of the quadcopter's motion and its control input response, define the bounds on the time the flight control software can spend in each iteration of the control loop. If the sensing, or state estimation tasks of the control loop take too long to complete, the state estimated

may no longer be accurate and thus the control outputs may send the quadcopter into an unstable state or the quadcopter may have transitioned into an unstable state.

As these types of CPS are being scaled-up, they are becoming more distributed in nature. The systems mentioned above could scale up to unmanned swarms of search and rescue drones, for instance, or large sensor/actuator networks for power distribution and control. Because each subsystem can directly and indirectly affect the others, all subsystems must communicate their states to each other, closing the control loop through the network.

An example of such scaling up is the recent research into developing fractionated spacecraft[39]. A fractionated spacecraft is a cluster of satellites cooperating, communicating and running distributed applications in service of the mission goals. Such a cluster design replaces the traditional monolithic satellites which are more expensive to develop, deploy, repair, and are more difficult to upgrade with new functionality. Because of this trend towards cooperating distribution of system resources, the network facilitating the cooperation and communications becomes a critical resource to the system. Whereas the single satellite's internal communications bus (direct physical connection system internal to a single satellite which allows sensing and actuation controlled by the computer) was ignored in the previous example, the wireless communications network enabling the satellite cluster cannot be ignored when analyzing the properties of the system. Because the satellites are expensive to deploy, impossible to repair, and must last for a long time to satisfy both budgetary constraints and mission goals, the application developers and system integrators for the cluster must ensure that the software on the cluster does not compromise its ability to meet the mission goals. For instance, the same orbit maintenance described previously now necessitates the use of the cluster's network. For a satellite to activate its thruster to maintain or modify its orbit, it must first ensure that such an action will not cause a collision with another of the satellites in the cluster. Therefore, every satellite must

know the state of every satellite in the system, and any thruster activation must be a coordinated action to ensure the safety and continued operation of the cluster. All of this state distribution and coordination occurs over the wireless network between the satellites, which (1) has limited resources, (2) is shared between all applications on all the satellites, and (3) varies as a function of time throughout the orbits of the satellites according to the orbital mechanics defining the system. The equations of motion for the satellites define the orbital paths taken by the satellites[1]. These paths are, for our purposes, circular or elliptical orbits, where each satellite in the cluster has the same orbital period and speed. Since these orbits have the same period, the distances between the satellites will vary periodically as a function of time. Because the satellites use a wireless network where the latency and bandwidth are directly proportional to distance, the wireless network capacity of the satellites will vary proportionally with respect to time. This final point about the equations of motion and their effect on network capacity is especially important, since it highlights how the physics of the system directly and drastically affects system resources and performance. Again, we must ensure the timing requirements of the control loops are met, except now those timing properties are directly related to the network resources, e.g. the end-to-end latency of traffic on the network links, the bandwidth of the links, and the buffer space available to the applications on each satellite.

The system provides network resources to applications and users of the system as a service. The quality of this service as seen by the users of the system is defined as the Quality of Service (QoS) of the network and is the overall performance of the network as seen by its users. The specific aspects of QoS which we focus on are the network bandwidth, end-to-end network latency, and availability of the network resources. For critical systems such as those described above which may be quite difficult to repair or replace, such requirements must be analyzed at design-time and verified to ensure that they are met. In any distributed CPS, the network performance of the system is affected by the physical environment of the system as it affects the network.

For systems using wired networks, the delay caused by the networks can affect the performance of the control systems. Further, during periods of high network load, the network performance as seen by the application traffic will degrade, which can increase the latency and buffer space required by the applications. Therefore, even in wired networks, analyzing the affect of applications' network traffic on each other is important for understanding the quality of the network service as seen by the applications. For systems whose physical network layer is made up of wireless connections, the physical environment has an even larger effect on the network resources and availability. Environmental interference or obstruction leading to multi-path self-interference or signal degradation can combine with the distance-based signal-to-noise ratio loss due to the nature of wireless media. Such effects can induce hysteresis or instability in control systems through loss of data on the network, or increased variability in the response-time of the control loops. Because the network performance of such a system is so tightly coupled with the physics governing the system, the physical dynamics must be taken into account when predicting the run-time characteristics of the network. Additionally, such resource constrained systems which are expensive to develop and deploy must maximize their return on investment through the hosting of payload applications (e.g. for scientific research), while ensuring that the resource requirements are not exceeded. This design-time analysis of time-varying resources and their constraints is paramount to ensuring a stable system, where we define stability here as 1) all applications have finite bounded network resource requirements, 2) All application's network resource requirements can be satisfied by the system. Such stability means that applications' data will be serviced by the system without loss and within the time required by the application. For networked distributed control systems, this definition of stability with respect to the network is required for meeting stability requirements of the control system itself.

Incorporating the physical dynamics into the model of the system network resources addresses only half of the problem, however. To facilitate accurate, meaningful resource

constraint analysis, the application developers are expected to model and describe the resource and timing constraints of their applications. As stated above, many of these systems have long-term missions, for which simple, static minimum/maximum resource and timing requirements lead to inefficient, underutilized, over-specified systems. To increase the fidelity of the application resource utilization model with respect to the actual application's resource usage, the time-varying nature of the application's network utilization should be modeled. In this way, tighter bounds on performance characteristics and resource utilization can be achieved. Tighter bounds on application performance and resource utilization allow system integrators to increase overall system resource utilization to maximize the mission-specific or scientific return of the system while still ensuring all applications receive their required services.

In addition to the design-time modeling and analysis which facilitates the calculation of performance guarantees about such critical CPS, the run-time systems require monitoring and management of resources and their utilization to prevent faulty or malicious applications from causing resource over-utilization and possibly making the system unstable or completely bringing the system down. Often this resource management is simply enforcing a static cap on resource utilization for each application. For such trivial resource management, often the operating system or other platform infrastructure is used to enforce these bounds on the applications' resources, e.g. open file descriptor limits or maximum buffer size limits being enforced in the operating system kernel. However, higher fidelity design-time models which more precisely capture the behavior and resource requirements of the applications can allow more sophisticated, time-varying resource monitoring and management.

Another type of adaptive resource management falls under the class of self-adaptive systems, which are capable of self-management at run-time. Using recent developments in autonomic computing, systems can use the sensors at their disposal to monitor their available resources as well as their environment, estimate the current state of the system,

and use the available system actions to transition into a new state[28]. A relevant example for such an adaptive system would be to eschew the design-time network modeling and analysis of what at run-time would be a relatively static system in favor of an adaptive network which manages the network resources for the applications based on the available resources the system has. Such a design has the benefits of possibly better utilization of system resources and better resilience to unplanned or unforeseen system events or states, but has the drawback of difficult design-time analysis. Currently, analyzing these adaptive systems at design time to derive guarantees about system behavior, resource availability, or performance is quite difficult and in many cases infeasible.

1.2 Challenges

The systems described above face many challenges for network performance prediction, as might be required by mission- or safety-critical application developers. Furthermore, an application which consumes more resources than specified at design-time, either through malicious or faulty code, can send these CPS into an unstable state by starving critical control processes of resources. Control systems can be forced into unstable system states, for example by high network latency exceeding the timing requirements of the control loops. Many systems may have a looser definition of stability, i.e. defined by the application with respect to that application, but we will consider loss of data or exceeding the latency requirements to be unstable behavior. In this section we outline the main challenges facing application developers and system integrators pertaining to network performance prediction and management and we separate these two classes of challenges into Design-Time Analysis challenges and Run-Time Management challenges.

1.2.1 Design-Time Network Performance Analysis of Distributed CPS Applications

A principal challenge of system design is the performance analysis of a system, its resources, and its applications at design-time. Such analysis and prediction is critical for

remotely managed systems and allows system integrators to provide guarantees to application developers about the services provided by the system. However, for complex distributed cyber-physical systems such design-time analysis is challenging. Such analysis may require capturing the behavior of the system and its applications in models that can then be composed and analyzed. Ensuring that the models properly capture the relevant characteristics of the run-time system is a challenge by itself, and is compounded by the challenge of composing the models for analysis. Such challenges for design-time network performance analysis are

- Modeling the interaction of the system with the physical world is difficult, esp. with respect to how the interaction directly or indirectly affects system resources and performance.
- Models of application network utilization can be imprecise and difficult to derive without a running system
- Application models may not represent actual application traffic on the network due to implementation details such as transport protocol selection (e.g. UDP vs TCP), which may alter the required bandwidth or buffering latency.
- Developing distributed applications for such systems is difficult, and should be done in a way that is amenable to modeling, analysis, and verification.
- Infrastructural code which handles low level system functions or network communications may obscure the application's network behavior from the application developer, making modeling of the application's network requirements difficult
- Network resources are becoming more critical to distributed CPS, but existing tools and techniques for design-time analysis of network resource utilization and performance do not support robust, precise analysis of such time-varying constraints

- For resource constrained systems, no processor or memory resources should be wasted, but without accurate and precise design-time analysis, systems must conservatively over-approximate network resource requirements.

- For application/system data flows in the network which require tight and/or real-time guarantees on temporal properties, design-time analysis is critical.

- Most systems route network traffic for nodes which cannot directly communicate. These routes may be defined at design-time and remain constant for the duration of the system, or may be unknown at design-time, changing dynamically during the run-time of the system. Dynamic routing is difficult to analyze for precise performance prediction because the the routes used by traffic may be unknown at design-time.

1.2.2 Run-Time Network Resource Monitoring and Management

Given specifications for system network resources and application network resource requirements, the system must ensure that no application either purposefully or inadvertently exceeds its allowed resource limits and starves other applications or critical system processes of those precious resources. Such resource management is crucial for ensuring system stability and proper service quality to applications and end-users. For systems with highly time-varying application load, system resource availability, or both, static limits under-utilize the system's resources. For such systems, higher fidelity resource management is needed to maximize the utilization of the system's resources. Further, these higher fidelity system and application network resource models pave the way for more accurate and robust failure or network attack (e.g. Denial of Service) detection which in turn can provide higher system stability. Challenges towards the development of such run-time network resource management are

- Available network resources at run-time should not be wasted if applications can use

them, but allowing run-time management is difficult because the behavior is difficult to analyze at design-time for performance analysis and prediction.

- Anomalies caused by applications attempting (due to either faults or attacks) to use more network resources than they originally specified should be detected and mitigated; the detection of coordinated attacks, e.g. distributed denial of service (DDoS), requires more sophisticated detection and mitigation techniques

- Systems are becoming more adaptive in nature and reacting to events at run-time (essentially data-dependent traffic); this adaptability is hard to provide performance metrics or guarantees for

1.3 Organization

The rest of this thesis is organized as follows

- Chapter II describes the related work in network analysis and management of distributed applications

- Chapter III describes design-time network performance analysis and prediction for CPS applications

- Chapter IV describes run-time network performance monitoring and management for CPS applications

- Chapter V concludes the thesis and describes possible extensions to the work in the future

- Appendix A lists the publications so far

CHAPTER II

RELATED WORK

2.1 Part 1: Design-Time Network Analysis and Performance Prediction

Networking systems have been developed for over half a century and the analysis of processing networks and communications networks began even earlier. As computing power has increased, the field of network performance analysis at design-time has evolved into two main paradigms: (1) network performance testing of the applications and system to be deployed to determine performance and pitfalls, and (2) analytical models and techniques to provide application network performance guarantees based on those models. The first paradigm generally involves either arbitrarily precise network simulation, or network emulation, or sub-scale experiments on the actual system. The second paradigm focuses on formal models and methods for composing and analyzing those models to derive performance predictions.

2.1.1 Performance Analysis Through Network Simulation/Emulation

Since computing networks are so prevalent, many tools exist to analyze system network behavior, either through simulation or mathematical analysis, which both attempt to determine one or more system properties based on one or more models of the system. One method of system simulation is discrete event simulation[47], in which all relevant events in the system are captured in the model and stepped through sequentially with the state of the model changing only at the simulated time steps. The resources of the system (e.g. buffer space) are simulated together with the entities in the system (e.g. the bits of the network traffic) through operations on the entities as they traverse the model and its resources. OMNET++[51] is a discrete event network simulator which simulates the network traffic as it passes through the network layers. The INETMANET[50] framework, built on top

of OMNET++, supports the simulation of network traffic over dynamic wireless links for gathering performance data about applications on the network.

NS-2[38] is a widely-used single-threaded discrete event simulator which allows both the simulation and emulation of both wired and wireless networks. Because of performance and scalability issues, however, the simulator is not well suited for scaling to large network simulation/emulation. Additionally, because of its design as a single-threaded discrete event simulator, it cannot fully simulate highly parallel distributed systems accurately. Furthermore, NS-2 has simulation accuracy issues (e.g. altering event ordering or timing) which plague any simulator used for emulation (i.e. connecting a simulator to a system to emulate the subsystem it is simulating). [20] gives a good study of the accuracy of NS-2 simulation with a testbed and finds that for constant bit-rate (CBR) traffic the simulation is accurate with respect to the behavior of the real system testbed, but for other types of traffic (e.g. FTP traffic), the simulation did not accurately model the dynamic behavior of FTP traffic.

NS-3[44] is a more recent rewrite of the NS-2 simulator designed to increase the realism of the network simulation by adding the ability to directly incorporate the actual code which implements the network protocols in the network software stack. Additionally, NS-3 has been extended to support distributed simulation of networks among multiple simulator machines. However, despite NS-3 performing better, it still produces the same results as NS-2[2] for certain protocols which may be inaccurate or unrealistic. Additionally, NS-3 modeling and simulation of the physical layer in networking systems has been shown to be incomplete or non-existent[43] which prevents the analysis of frame construction or reception effects on higher layers. [43] points out that without modeling such physical layer mechanisms, other aspects of the simulation model, such as the signal to interference noise ratios, are not valid because they are based on unrealistic assumptions which can only be removed by incorporating lower layer effects.

Despite the wide-spread use of these simulation toolsuites, it is clear that they are not

a viable candidate for providing both accurate and precise design-time guarantees about network performance and resource utilization.

Instead of simulating the network software stack and the physical network, another option is to directly emulate the network by shaping the traffic between the actual nodes of the system to directly apply the appropriate delay and enforce the proper bandwidth on each link of the network. Often this is done through the use of flow control tools either on routing node(s) or on capable network infrastructure devices, e.g. a smart switch. Dummynet[45][7] is a tool for network emulation when used on routing nodes in a system, utilizing the underlying network traffic shaping and policing tools available in Linux. Dummynet allows the configuration of routing tables, packet drop rates, link bandwidths, and link delays to conform the traffic passing through it to the supplied network configuration. OpenFlow[8] is an alternative for network emulation which instead uses a compatible smart switch to shape the network traffic and enforce the proper network topology and characteristics for all traffic in the network at a lower network layer without requiring the use of a separate traffic shaping node.

For the types of systems we have described in Chapter I, typically these types of simulation and testbed emulation are used to analyze the performance of the applications and the system. Unfortunately simulation and emulation based performance analysis techniques are unable to provide the guarantees required by application developers and system integrators.

2.1.2 Analytical Approaches to Network Analysis

2.1.2.1 Queuing Theory

Queuing Theory[27][19] is a probabilistic approach to the analysis of processing or communications networks, and has been applied to many types of systems including telecommunications, processing, and distribution systems. A queuing system can be described using notation of the form $A/B/S/\Delta/E$, introduced by [27], with the semantics:

- A: Type of arrival process, e.g. M for Poisson Arrival Process
- B: Request service time statistics, e.g. D for Deterministic service time
- S: Number of servers
- Δ: Queue length
- E: Number of producers

Queuing Theory allows the analysis of the mean number of requests (N) in the queue and the mean buffering delay (T) experienced by objects traversing the queue. Little's Theorem provides the relation between the two : $N = \lambda T$[35], where λ is the mean arrival rate into the queue. However, this theorem assumes (1) that the service policy is independent of service time and (2) the service policy is work conserving. Assumption (1) may be violated for policy-based routing and servicing which tries to provide guaranteed QoS to applications, and assumption (2) is violated by wireless networks, in which nodes with very limited connectivity or dropouts in connectivity are not able to service the data in the buffers despite the existence of the data in the buffers and applications continuing to produce data.

For the types of systems we have described in Chapter I, probabilistic analysis techniques like Queuing Theory make providing the requisite performance and resource guarantees difficult or impossible because of the stochastic nature of the models of network traffic[10]. Because of the need for these strict guarantees, other deterministic formal models for the analysis of communications and processing systems have been developed.

2.1.2.2 Network Calculus and Min-Plus Calculus

Network Calculus[10][9][33] is a theory for deterministic queuing systems which provides the ability to determine worst-case buffer requirements and application buffering delay at design-time by applying the techniques of (min,+) calculus to queuing theory. We

will describe the foundation of (min,+) calculus before covering the techniques of Network Calculus. [33], Chapter 3, gives an excellent overview of both min-plus and max-plus calculus, on which Network Calculus is based. An abbreviated explanation of the concepts of these two related dioids (additive inverses need not exist) follows.

Min-plus calculus, $(\mathbb{R} \cup \{+\infty\}, \wedge, +)$, deals with *wide-sense increasing functions* :

$$\mathcal{F} = \{f : \mathbb{R}^+ \to \mathbb{R}^+, \forall s \leq t : f(s) \leq f(t), f(0) = 0\} \tag{1}$$

which represent functions whose slopes are always ≥ 0. Intuitively this makes sense for modeling network traffic, as data can only ever by sent or not sent by the network, therefore the cumulative amount of data sent by the network as a function of time can only ever increase or stagnate. A wide-sense increasing function can further be classified as a sub-additive function if

$$\forall s,t : f(s+t) \leq f(s) + f(t) \tag{2}$$

Note that if a function is concave with $f(0) = 0$, it is sub-additive, e.g. $y = \sqrt{x}$. Sub-additivity of functions is required to be able to define meaningful constraints for network calculus, though realistically modeled systems (in Network Calculus) will always have sub-additive functions to describe their network characteristics (e.g. data serviced or data produced). This sub-additivity comes from the semantics of the modeling; since the models describe maximum data production or minimum service as functions of *time-windows*, maximum data production over a longer time window must inherently encompass the maximum data production of shorter time-windows. Some examples of wide-sense increasing functions which are of use in Network Calculus are shown in Figure 1.

The main operations of min-plus calculus are the convolution and deconvolution operations, which act on sub-additive functions. Convolution is a function of the form:

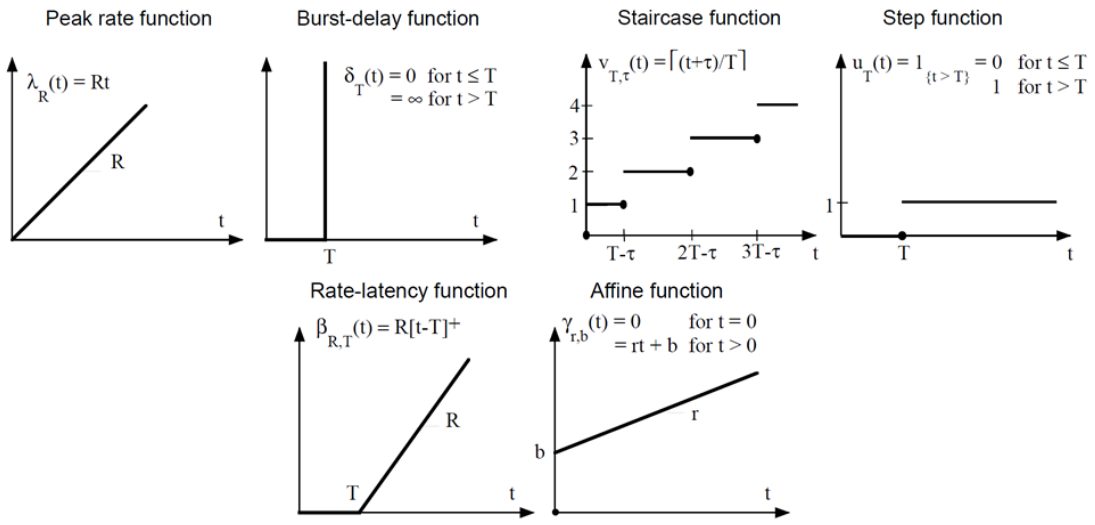


Figure 1: Example wide sense increasing functions, reprinted from [33].

$$(f \otimes g)(t) \equiv inf_{\{0 \leq s \leq t\}} \{f(t-s) + g(s)\} \tag{3}$$

Note that if the functions f, g are concave, this convolution simplifies into the computation of the minimum:

$$(f \otimes g)(t) = min(f, g) \tag{4}$$

Convolution in min-plus calculus has the properties of

1. Closure: $(f \otimes g)(t) \in \mathcal{F}$,
2. Associativity: $\forall f, g, h \in \mathcal{F}, (f \otimes g) \otimes h = f \otimes (g \otimes h)$,
3. Commutativity: $\forall f, g \in \mathcal{F}, f \otimes g = g \otimes f$, and
4. Distributivity w.r.t. $\wedge$: $\forall f, g, h \in \mathcal{F}, f \otimes (g \wedge h) = (f \otimes g) \wedge (f \otimes h)$

Similarly, deconvolution is a function of the form:

$$(f \oslash g)(t) \equiv sup_{\{0 \leq u\}} \{f(t+u) - g(u)\} \tag{5}$$

Note that $\oslash$ is not closed in $\mathcal{F}$ because $(f \oslash g)(t)$ is not necessarily 0 for $t \leq 0$.

Network Calculus focuses on abstracting the network traffic and the computing nodes as *arrival curves* and traffic shaping *service curves*. The arrival curves and service curves model the amount of data generated or serviced as functions of time window size and are bounded by maximum and minimum arrival and service curves. By abstracting the network flows and traffic shapers as arrival curves and service curves, respectively, (min,+) calculus can be used to compose models of system behavior and calculate performance characteristics of the integration of the application and the network.

Given an arrival function $R(t)$ for the data flow describing the number of bits seen on the flow during the time interval $[0,t)$, the arrival curve α constrains the flow if and only if

$$\forall s \leq t : R(t) - R(s) \leq \alpha(t-s) \tag{6}$$

This relation is shown in Figure 2. Intuitively the arrival curve representation transforms the data production from a function of time, described by $R(t)$, into a function of time-interval, described by $\alpha(t)$, for which $R \leq R \otimes \alpha$.

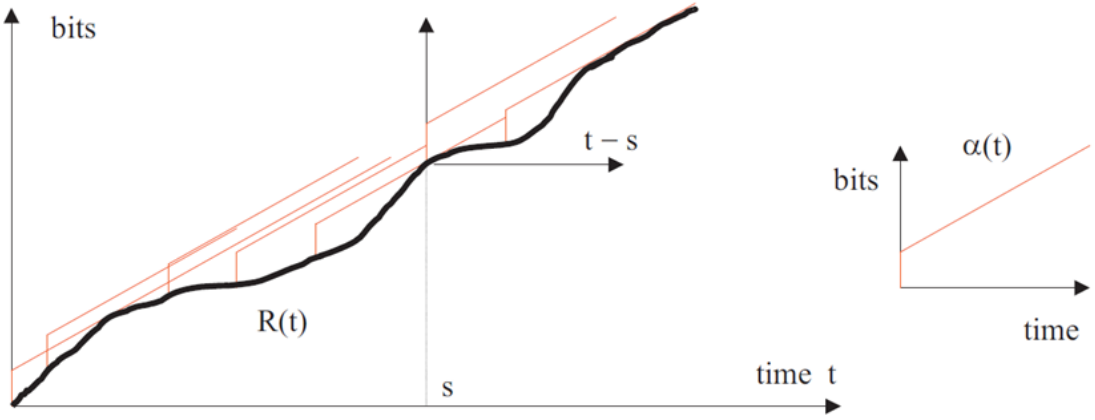


Figure 2: Illustrative example representing maximum arrival curves ($\alpha(t)$) for data flows ($R(t)$), reprinted from [33].

Similarly, service curves transform the output data flow $R^*(t)$ into a minimum service curve β according to the relation:

$$R(t) - R^*(t_0) \leq \beta(t - t_0), \forall t \geq 0 \; \exists \; t_0 \geq 0, t_0 \leq t \tag{7}$$

or more compactly $R^* \geq R \otimes \beta$. This relation is shown in Figure 3.

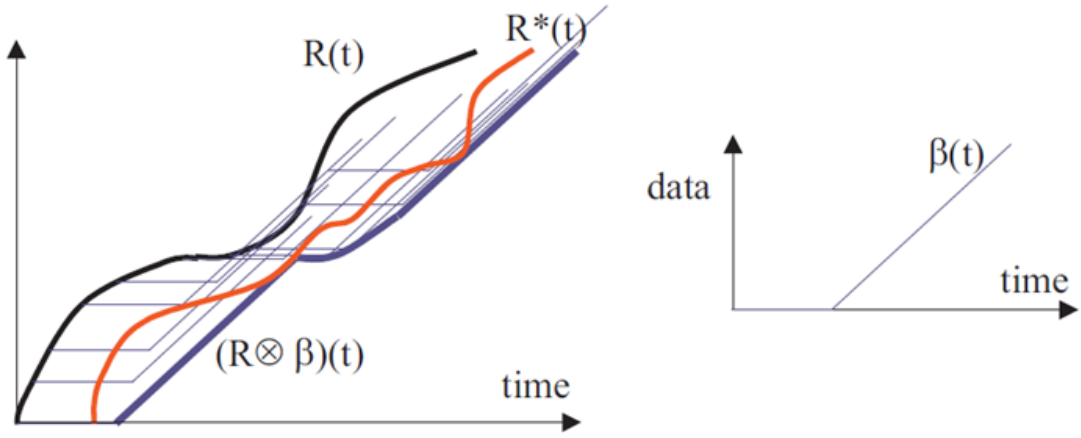


Figure 3: Illustrative example representing minimum service curves ($\beta(t)$) for output data flows ($R^*(t)$), reprinted from [33].

From the input arrival curve α into a node providing service curve β, we can use Network Calculus to compute the output flow from the node and a few performance bounds governing the buffering delay and buffer requirements for the node. The output flow from the node is constrained by the arrival curve $\alpha^* = \alpha \oslash \beta$. Given the arrival curve and service curve for a node or system, we can compute the backlog and delay bounds, see Figure 4; the backlog bound is given by:

$$R(t) - R^*(t) \leq sup_{\{s \geq 0\}} \{\alpha(s) - \beta(s)\} \tag{8}$$

and the delay bound is given by:

$$h(\alpha,\beta) = sup_{\{s\geq 0\}}[inf\{T : T \geq 0 \; and \; \alpha(s) \leq \beta(s+T)\}] \tag{9}$$

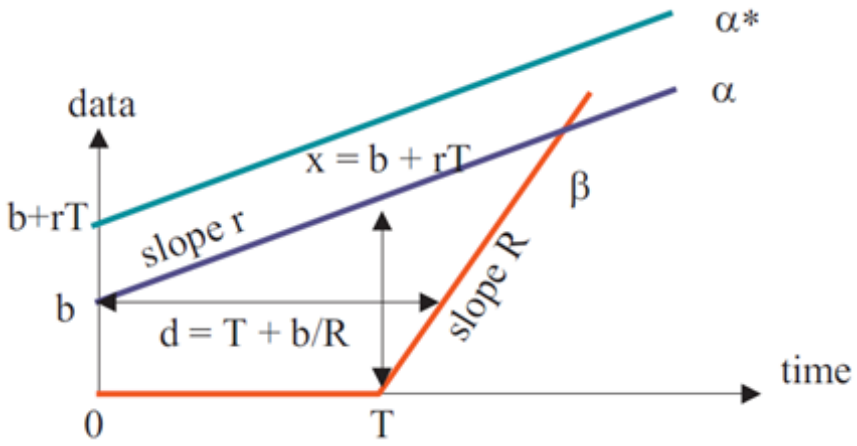


Figure 4: Illustrative example representing the backlog and delay bounds calculated from input arrival curves and node service curves, reprinted from [33].

These bounds provide the requisite information needed to make design-time guarantees about *worst-case* application performance on the network, given that both the application traffic profile and the system's network performance are deterministic.

To enable compositional system analysis, Network Calculus allows for the concatenation of nodes, Figure 5, such that a flow traversing nodes N_1 and N_2 in sequence, where each node provides FIFO service curve $\beta_{i=1,2}$, the concatenation of the two nodes offers a service curve $\beta_1 \otimes \beta_2$ to the flow. A major advantage of this approach is the ability to "Pay Bursts Only Once" (PBOO), which is the property that the delay and buffer bounds are tighter when derived from the concatenation of the system than they would have been if they were calculated iteratively. Again, note that this advantage is not applicable to non-FIFO systems[33].

However, the performance bounds calculated by Network Calculus are still worst-case performance based. For instance, there is a temporal disconnect between the arrival/service curves and the actual performance of the application or the system. This disconnect

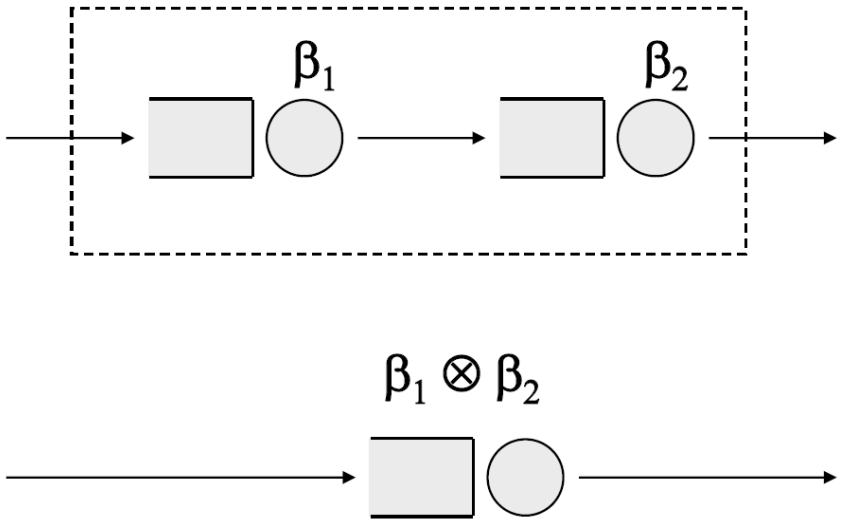


Figure 5: Illustrative example representing the concatenation of two nodes providing separate services into a single node providing an aggregate service, reprinted from [33]

leads to analysis results that may still over-approximate the required buffer size or application delay on the network. The cause of this over-approximation comes from the use of *time windows*. Because Network Calculus is focused on maximum data produced and minimum data serviced as functions of time window size, the time-varying nature of the data production or service is lost. Despite an application producing a Bulk Data Transfer (BDT) during a period of high network resource availability, Network Calculus compares that BDT to all windows of time throughout the service time of the system. As such, an expected drop in service during a different period of time will inadvertently negatively affect the application's predicted performance as analyzed by Network Calculus.

2.1.2.3 Real-Time Calculus

Real-Time Calculus[49] builds from Network Calculus, Max-Plus Linear System Theory, and real-time scheduling to analyze systems which provide computational or communications services. Unlike Network Calculus, Real-Time Calculus (RTC) is designed to analyze the impact of real-time scheduling and priority assignment in task service systems.

The use of (max,+)-calculus in RTC allows specification and analysis not of only the arrival and service curves described above for Network Calculus, but of upper and lower arrival curves ($\alpha^u(\Delta)$ and $\alpha^l(\Delta)$) and upper and lower service curves ($\beta^u(\Delta)$ and $\beta^l(\Delta)$). These curves represent the minimum and maximum computation requested and computation serviced, respectively. An overview of RTC is given in Figure 6.

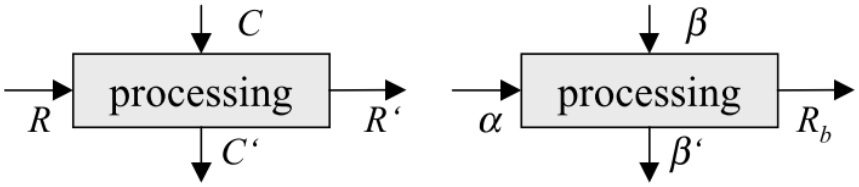


Figure 6: Overview of Real-Time Calculus' request, computation, and capacity models. $R(t)$ is the request function that represents the amount of computation that has been requested up to time t, with associated minimum request curve, α. $R'(t)$ is the total amount of computation delivered up to time t, with associated delivered computation bound $R_b(t)$. C and C' are the capacity function and remaining capacity functions which describe the total processing capacity under full load and the remaining processing capacity, respectively. C and C' are bounded by the delivery curve β and the remaining delivery curve β', reprinted from [49].

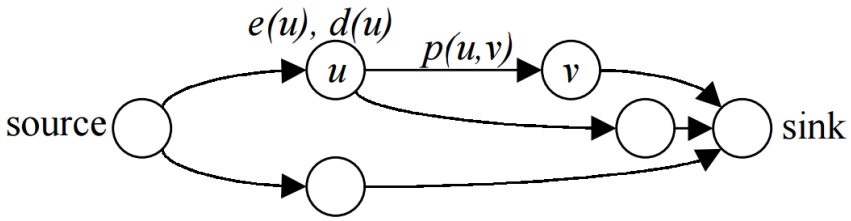


Figure 7: An example task graph for Real-Time Calculus, with conditional branches; reprinted from [49]

RTC allows for the analysis of task scheduling systems by computing the request curve for a task model which is represented as a directed acyclic graph (DAG), the task graph $G(T)$. An example task graph is shown in Figure 7. The graph's vertices represent subtasks and each have their own associated required computation time $e(u)$, and relative deadline

$d(u)$ specifying that the task must be completed $d(u)$ units of time after its triggering. Two vertices in $G(T)$ may be connected by a directed edge (u,v) which has an associated parameter $p(u,v)$ which specifies the minimum time that must elapse after the triggering of u before v can be triggered. RTC develops from this specification the minimum computation request curve α_r and the maximum computation demand curve α_d. Finally, the schedulability of a task T_i is determined by the relation:

$$\beta'(\Delta) \geq \alpha_d^i(\Delta) \quad \forall\Delta \tag{10}$$

which, if satisfied, guarantees that task T_i will meet all of its deadlines for a static priority scheduler where tasks are ordered with decreasing priority. Note that the remaining delivery curve $\beta'(\Delta)$ is the capacity offered to task T_i after all tasks $T_{1\leq j<i}$ have been processed. Similarly to Network Calculus, RTC provides analytical techniques for the computation of performance metrics such as computation backlog bounds:

$$\text{backlog} \leq sup_{\{t\geq 0\}}\{\alpha^u(t) - \beta^l(t)\} \tag{11}$$

which is equivalent to the network backlog bound derived in Network Calculus.

[18] compares different analytical methods for network performance analysis, namely Real-Time Calculus (RTC), probabilistic queuing models, parallel computation models, and protocol offload models. The authors explain the current state of system evaluation, which is based predominantly on quantitative evaluation through simulation, but make the point that such simulation techniques should be used sparingly since *only a finite state of initial states, environment behaviors, and execution traces can be considered by system simulators*. The system which the authors model for their comparison is the case of Network Interface Cards (NICs) connected to a Local Area Network (LAN), for which they derive analytical bounds on the buffer requirements as they are affected by the input/output (I/O) subsystems, the network traffic, the NIC itself, and the memory controllers. They

point out that most researchers are still using Queuing Theory, in stochastic scenarios where the network traffic is modeled as random distributions of data. Because RTC allows more precise descriptions of application traffic, it can be more beneficial for providing analysis of buffer requirements and delay experienced in the system. RTC's ability to allow such specifications comes from its roots in Network Calculus and Max-Plus Linear System theory.

2.1.2.4 Stochastic Network Calculus

These deterministic constraints can be relaxed so that the deterministic arrival and service curves are instead replaced by stochastic processes, causing the bounds on the performance to be probabilistic as well[6]. As described previously, these probabilistic performance bounds may not be precise enough to provide the types of guarantees required by certain classes of mission- or safety-critical systems.

[52] provides a good description, system model, and analysis for stochastic Network Calculus applied to wireless networks. In their work, they show the ability of network calculus to remove the need to make as many assumptions about the arrival or service processes (*e.g.* exponential service distribution) to allow general arrival and service processes. They apply stochastic network calculus to analyze backlog and delay bounds in 802.11 based multi-access systems. They formally derive bounds for the backlog and delay in the network and then compare these analytical results to bounds generated from network simulation using ns-2. From this comparison, they conclude that the derived bounds are too loose and in fact get looser the closer the system gets to saturation. They further conclude that this looseness is a direct result of stochastic network calculus itself, and claim that it requires further improvements. It is important to remember what was stated previously by [20]: the simulation does not accurately model the dynamic behavior of real traffic, so the results from [52] may too be inaccurate.

Because Network Calculus deals with either deterministic worst-case application performance on a static network or stochastic application performance on a dynamic network, system designers and application designers under-utilize the network resources of systems which require strict design-time guarantees about application performance.

2.1.2.5 Extensions to Network Calculus

When analyzing any complex system, the fidelity of the analysis results with respect to the actual system relies heavily on the level of detail of the models of the system's components and subsystems. [22] covered the effects that different levels of detail have on analysis complexity and accuracy. Importantly, they point out the requirement to not only be correct, but also be applicable, *i.e.* analysis results should be both accurate with respect to the system being modeled, but should also be relevant for the analysis and development of real systems. Additionally, they point out that not all systems require highly detailed modeling for the analysis results to be correct, since some systems and applications are insensitive to lower level details.

There are many efforts to make analytical techniques more representative of actual systems in order to increase the fidelity of the analysis results with respect to the run-time system. The researchers in [34] recognize the need to analyze not only the overall throughput of a network, but also the end-to-end delay experienced by information flows in the network. Furthermore, they derive an analytical model of Wireless Network Coding[26], a technique for combining packets together for improving network throughput in wireless networks using broadcast techniques. They show that by developing a model of the way the MAC layer works in the network and how the information flows are combined and disseminated, they can get tighter performance bounds and even derive methods for increasing performance in the network by altering the scheduling parameters of the packet flows. Analyzing multiple performance parameters, in this case the network throughput and

end-to-end delay, is a key element for analyzing and providing quality of service (QoS) to applications.

The authors in [4] also incorporate more precise models of the network to derive tighter performance bounds using Network Calculus. They show that by modeling the packetization that occurs in the network using a *packet operator* to transform arrival flows into packet flows, they can analytically derive tighter service curves than would be found from traditional Network Calculus. Clearly, there exists a desire from application developers and system integrators to derive both accurate and precise design-time performance parameters for the system and its applications.

Similarly, in [15], the authors describe how to accurately model the SpaceWire network standard which has been developed for satellites in the European Space Agency (ESA). Their network must be shared by both real-time (critical) and non real-time (non-critical) traffic, but the system developers require design-time guarantees about the temporal characteristics of all critical/real-time messages on the network. Their work focuses on accurately representing the SpaceWire network, its (static) routing protocol, and the service profiles of its routers including the aspects of their flow control algorithms. Building on previous work, they explain the need, for resource-constrained real-time systems, to accurately model the network traffic in order to derive a model of the network which is not too pessimistic. They derive accurate Network Calculus/Real-Time Calculus (RTC) based models of the wormhole switches present in the network and show the fidelity of their analytical tools compared with the industrial simulation tools developed for SpaceWire networks. Using a Network Calculus-based model, they are able to achieve analytical results that are the same order of magnitude as the simulation results for the critical traffic delay characteristics, but are less precise for the non-critical traffic in the network. One important point they make that extends to all types of systems when comparing analysis and simulation techniques is this: *worst-case delays can be extremely rare events which are hard to observe or recreate in simulations, but can be derived from analytical results.*[15]

Another approach to increasing the fidelity of the analysis is to model the Time Division Multiple Access (TDMA) medium channel access protocol using Network Calculus to derive performance metrics[46]. TDMA service curves are modeled such that the medium's transmit capacity is available to the node only during the node's designated slot. During all other slots of the TDMA period, the medium's capacity is unavailable to the node and therefore the transmit capability of the node is zero. As such, simple TDMA service curves can be described using simply a slot length, a slot bandwidth, and a TDMA period.

[29] analyzes the performance of TDMA with respect to the queue size for different probabilistic traffic models, and shows how the G/D/1 model with application-based probability distributions can be used to generate closed-form solutions for analyzing arbitrary traffic on a TDMA network.

Another aspect of system design which has been gaining momentum is the development of self-adaptive systems which provide "self-*" properties such as self-management. These types of systems are typically not used in CPS control applications or other systems which require real-time guarantees about timing or resource properties of the system. The main reason for their absence from these types of systems and applications is the lack of available, accurate modeling and analysis techniques which properly capture the behavior of the applications in a way that allows the derivation of performance guarantees. The authors in [12] describe both the need for this type of analysis for these systems and describe the overview of how the analysis would work, based on concepts from Network Calculus. Their main point is that currently such types of analysis tools do not exist for these systems, which makes developing the systems difficult with respect to these types of design parameters. They propose developing a formalized standardization for the self-adaptive behavior, which they present as a state-space with available control actions based on the sensor data in the system.

2.2 Part 2: Run-Time Network Monitoring and Management

In addition to design-time modeling and analysis, CPS system designers and integrators must ensure system stability during run-time by enforcing resource limitations on the applications to ensure no faulty or malicious code starves the system or other applications of network resources. Such enforcement is the management of the network resource for the system. Many different approaches exist to handle this type of management, generally falling into one of two categories: (1) static management or (2) dynamic management. Static management of system resources is based around enforcement of fixed resource allocations which were decided at design-time or deployment time. Such management generally is associated with high-criticality systems which must be guaranteed. Dynamic management of resources entails updating the resource allotments of each application based on currently available system resources and application load, and generally is in the class of adaptive management or adaptive systems (also called autonomic systems). In this work, we will address only static management of resources.

Static management of network resources generally, but not necessarily, means applications are given a fixed quantity of resources for the lifetime of the system. The part of the system which enforces these resource allotments however, may vary depending on the design of the system. The enforcement may happen in the network layer, in the operating system kernel, or in some cases in the middleware facilitating the network communications for the applications. We deem any enforcement happening in the kernel or in a lower layer to be *infrastructural management* (since all applications on the system must use this infrastructure and are therefore managed by it). We deem any resource management happening between the kernel and the application as *middleware management*, since different applications deployed on the system may use different middleware stacks and therefore may be managed differently.

2.2.1 Infrastructural Approaches for Network Management

[19], Chapter 3 has a good overview of both IntServ and DiffServ. DiffServ, for Differentiated Services, is designed for the provisioning of network resources to provide Quality of Service (QoS) to applications on the network but is unable to provide strict real-time guarantees about packet loss, delay, and bandwidth availability. Instead, DiffServ was designed to scale well for large systems while still providing probabilistic guarantees. IntServ, for Integrated Services, was designed to provide strict real-time guarantees about the QoS experienced by a flow on the network. Unlike DiffServ, which does not maintain any state information in the routers along network flow paths, IntServ uses a resource reservation protocol (RSVP) with explicit setup of flows for deterministically allocating bandwidth and buffer space for a flow in each router along the flow's path. While such an explicit out-of-band QoS reservation protocol enables similarly explicit resource availability and performance guarantees, the trade-off comes in the ability of the system to scale to many nodes and many flows. DiffServ's scalability comes from both the lack of explicitly maintaining per-flow state in the routers, by assigning traffic to a set of predefined classes, as well as using QoS assignment mechanisms which are built into the flow's messages, e.g. the DiffServ Code Point (DSCP) built into the Type of Service (ToS) byte in IPv4 headers and the Traffic Class byte of IPv6 headers.

Both IntServ and DiffServ were originally designed for wired networks, but [37] has worked on the required modifications to make them suitable for wireless networks, which have network connectivity and link characteristics which have more variance as a function of time. The combination of low bandwidth, high loss, and node mobility require extensions to the QoS parameters and control options available to the application provided by the QoS infrastructures. One such proposed extension is the concept of loss profiles, which govern whether an application prefers dropping data in a bursty manner (as might be preferred by audio applications) versus a distributed manner (as might be preferred by video applications). Similarly, since link bandwidth is typically much lower than in wired

networks, IntServ/RSVP's refresh messages (used to determine network changes) should be sent with a lower frequency to provide as much network bandwidth as possible to application traffic. In the same way, DiffServ requires modifications to support signaling information about link state and node location to overcome DiffServ's static provisioning scheme in the adaptation from wired to wireless networks.

A system's network infrastructure may provide multiple different QoS provisioning implementations, such as both DiffServ and IntServ. In this case, the applications can select which QoS provisioning to use. Similarly, large networks may be grouped into subnets which each internally use different QoS provisioning schemes. The boundaries between these subnets requires QoS mapping for flows which cross these boundaries. Such mapping between QoS implementations and configurations is complex and makes providing guarantees about QoS for large complex networks challenging.

Because both IntServ and DiffServ were designed for providing QoS to generic traffic for large networks including the internet, they were not designed to be able to provide performance guarantees to application developers. As such, their design and implementation function more coherently in a system which has unknown applications and application load. However, the classes of systems we focus on require more precise guarantees about performance and have the benefit of more precise design-time knowledge of applications and application load on the system.

Flexible QoS Model for Mobile Ad-hoc Networks (MANETs), FQMM[53], attempts to address the issue of run-time QoS management and adaptation to changing environmental conditions affecting the network. Recognizing that both environmental and application behavior need to be taken into account for QoS management, they argue that two methods for providing QoS in the internet, IntServ[5] and DiffServ[3], are not sufficient for dynamic mobile networks. While IntServ's scalability problem will not affect dynamic mobile networks in the near future, they argue that the connection maintenance required by the Resource ReSerVation Protocol (RSVP) renders IntServ impractical. DiffServ, on the

other hand, might be able to provide long-term QoS to applications under the varying network conditions, but is not feasibly able to provide the kind of short-term QoS required by real-time applications. Furthermore, DiffServ does not handle node mobility and external disturbances from the environment well as it was originally designed for relatively fixed (topologically) networks.

To combat the issues in both IntServ and DiffServ, FQMM is designed to handle QoS for MANETs. FQMM focuses on allowing for fine-grained provisioning of node resources and allowing node mobility through dynamically reassigning the roles of each of the nodes in the network. The provisioning of the resources for flows in the network borrows ideas from both IntServ and DiffServ by combining the per-flow granularity of IntServ for high-priority flows while lower-priority flows are provisioned on a class basis as in DiffServ. This differentiation between traffic classes and priority flows better utilizes the system resources to achieve the necessary performance for high priority flows which may need real-time performance. To provide traffic shaping they constrain the bandwidth of flows to traffic profiles, which govern the latency and bandwidth available to the flow. To combat the time-varying nature of the network, they instead define these traffic profiles as a percentage of the available network bandwidth. This type of percentage-based flow constraint limits the adaptability of the network traffic however, as certain higher-priority real-time flows may have a minimum amount of bandwidth required that cannot be met with a percentile constraint on effective link bandwidth. FQMM also addresses routing control to provide better run-time QoS to applications on the system.

2.2.2 Middleware Based Approaches for Network Management

For system and application level adaptation to changing system resources, two main approaches, namely fixed reservation of flows and run-time adaptation, provide benefits for performance or resilience. These two approaches cannot be used alone however, as fixed reservation of flows based on design-time network analysis causes low resource utilization

and run-time adaptation is generally not prepared for excessive congestion or other disturbances. GARA [16] combines these two paradigms to provide more graceful degradation and higher resource utilization at the system and application level. GARA uses priority based flow reservation which can be altered at run-time by both the application and by third parties on behalf of the application. This type of reservation scheme allows applications to monitor and react to changes in network capacity, while still attempting to ensure that high-priority flows can traverse the network. Furthermore, this type of reservation scheme is more amenable to dynamic flows which may only be active during a portion of time that the system is active. Statically defined slots reserved at design-time cause wasted resources by these applications whose flow is reserved but not used the entire time.

Finally, there do exist different protocols and communications paradigms which support run-time control of application network traffic, such as the Quality of Service (QoS) control mechanisms present in many implementations of OMG's Data Distribution Service (DDS) standard[42][41]. However, often the mechanisms available for controlling the QoS parameters of a given data stream are complex, interacting mechanisms which may be difficult for the application developer to understand and therefore are also not amenable to modeling and analysis at design time[23]. Furthermore, the developers may not be provided with or have control over lower level implementation details such as the selected transport layer protocol, which may affect the available QoS or may not be fully supported by the infrastructure. Additionally, many of the available interaction paradigms either do not support design time QoS analysis with run-time monitoring and control or the supported QoS analysis and control interfaces are only informally specified.

CHAPTER III

DESIGN-TIME NETWORK PERFORMANCE ANALYSIS OF DISTRIBUTED CPS APPLICATIONS

In this chapter, we describe research results related to the challenge of accurately predicting network QoS for systems which may require strict guarantees about performance and resource utilization.

Analyzing application and system network Quality of Service (QoS) requires either design-time models and analysis techniques or experimental measurements from an application and system testbed. For high- or mixed-criticality software and systems, typically experimental measurements are used but often these can be incomplete or quite costly to generate. Instead, a design-time modeling paradigm for networked applications and systems can provide developers and system integrators the information to accurately predict the system and application network QoS.

Some wireless mobile CPS networks, such as the network between a cluster of satellites orbiting Earth, vary periodically with respect to time, *e.g.* according to the cluster's orbital period. An example of such periodic variation in satellite network capacity is shown in Figure 8. For such networks, the physical dynamics of the nodes in the cluster are well understood and predictable, therefore the network dynamics can be fairly predictable as well. For such predictable or periodic dynamic networks, the use of worst-case network performance for analysis and constraint verification wastes the network resources over much of the life-cycle of the system. Integrating the physical dynamics of the network into the modeling and analysis tools improves the performance of the systems without degrading its reliability.

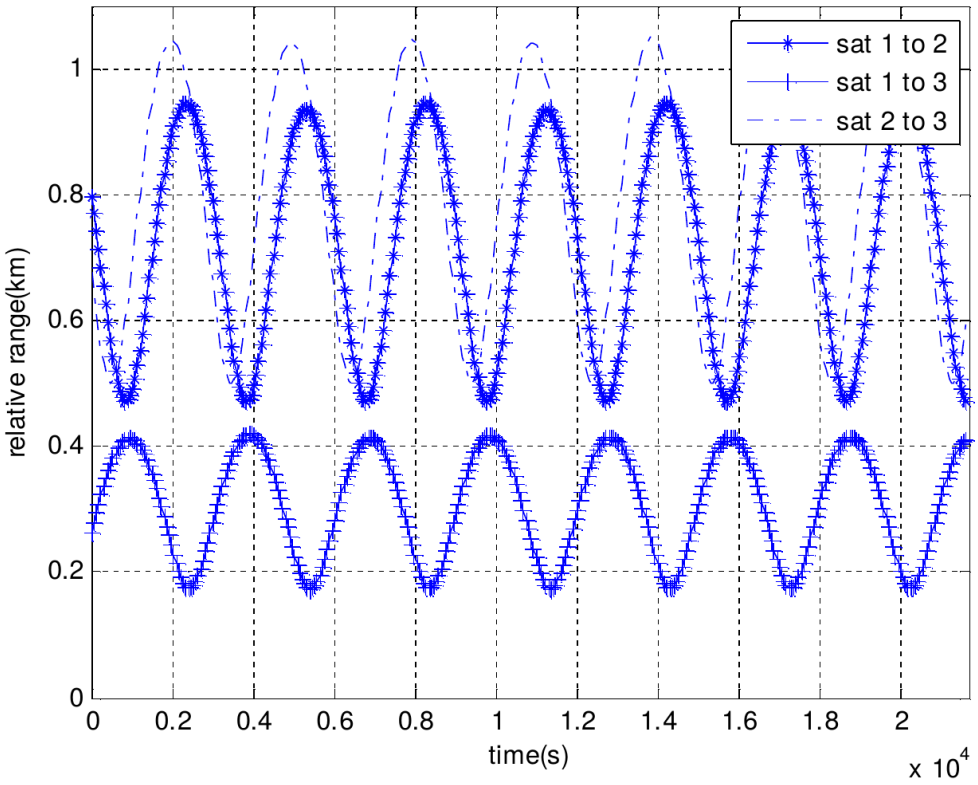


Figure 8: Distance as a function of time (pairwise) between satellites in a cluster orbiting Earth. The network capacity varies for each network link between two satellites inversely proportional to the distance between them. Reprinted from [48].

3.1 Precise Modeling of Application Network Traffic and System Network Resources for Performance Prediction

To precisely predict network performance for applications distributed in a mobile CPS, we need precise models of both the application network resource requirements and the system's network resource capacity. These predictions, if precise and reliable enough, can serve as guarantees for application developers and system integrators about the network QoS that the system will provide and the network resources the application will receive.

3.1.1 Problem

Current design tools do not incorporate the physical dynamics of the network for analysis of network constraints on the applications. For systems with known models of system

dynamics, the system's dynamics should be incorporated into the modeling tool and should integrate with the other models of the system, e.g. the system's network models. Because of the diversity of CPS, IoT systems, and other networked embedded systems in general, modeling and analysis tools targeted towards these systems must support a wide range of configurations, architectures, standards, and interfaces. The same compatibility is required in network modeling and analysis frameworks. Because many of these systems may support different types of network communications hardware, often using multiple types of network interface hardware within the same system, the models of the network must be able to express network resources in a way that can capture these differences. Because of this diversity and the modeling semantics of the commonly used paradigms (e.g. Real-Time Calculus), developing very precise predictions is difficult, since the models are not very precise with respect to the actual behavior of the applications on the system.

3.1.2 Mathematical Formalism

To model the network capability of the system and the application traffic patterns, we have developed a network modeling paradigm similar to Network Calculus' traffic arrival curves and traffic shaper service curves. This paradigm is called **P**recise **N**etwork **P**erformance **P**rediction (PNP^2).

Similarly to Network Calculus' arrival curves and service curves, our network profiles model how the system's network performance or application traffic generation changes with respect to time. Whereas Network Calculus' modeling transforms application data profiles and network service profiles into max/min curves for data received/serviced vs. length of time-window, our models take a simpler approach which models exactly the data generated by the application and the data which could be sent through the network, allowing our performance metrics to be more precise. Specifically, the bandwidth that the network provides on a given communication link is specified as a periodic time series of scalar bandwidth values. Here, bandwidth is defined as data rate, i.e. bits per second, over some

averaging interval. This bandwidth profile can then be time-integrated to determine the maximum amount of data throughput the network link could provide over a given time. The bandwidth profile for the application traffic similarly can be time-integrated to determine the amount of data that the application attempts to send on the network link as a function of time.

Having time-integrated the bandwidth profiles to obtain data vs. time profiles that the application requires and that the system provides, we can use a special type of convolution ($\otimes$), *(min,+)-calculus convolution*, on these two profiles to obtain the transmitted link data profile as a function of discrete time. The convolution we define on these profiles borrows concepts from the min-plus calculus used in Network Calculus, but does not use a sliding-window and instead takes the transformed minimum of the profiles. For a given application data generation profile, $r[t]$, and a given system link capacity profile $p[t]$, where $t \in \mathbb{N}$, the link transmitted data profile $l[t]$ is given by the convolution Equation 12. The difference $(p[t-1]-l[t-1])$ represents the difference between the amount of data that has been transmitted on the link $(l[t-1])$ and the data that the link could have transmitted at full utilization $(p[t-1])$. As demonstrated by the convolution equation, $\forall t : l[t] \leq r[t]$, which is the relation that, without lower-layer reliable transport, the link cannot transmit more application data for the application than the application requests. Note that there will be packetization and communication header overhead which will be transmitted with application data. The overhead can be determined at design time and can therefore be accounted for in the application profile.

$$\begin{aligned} y = l[t] &= (r \otimes p)[t] \\ &= min(r[t], p[t] - (p[t-1] - l[t-1])) \end{aligned} \tag{12}$$

$$\text{buffer} = sup\{r[t] - l[t] : t \in \mathbb{N}\} \tag{13}$$

$$\text{delay} = sup\{l^{-1}[y] - r^{-1}[y] : y \in \mathbb{N}\} \tag{14}$$

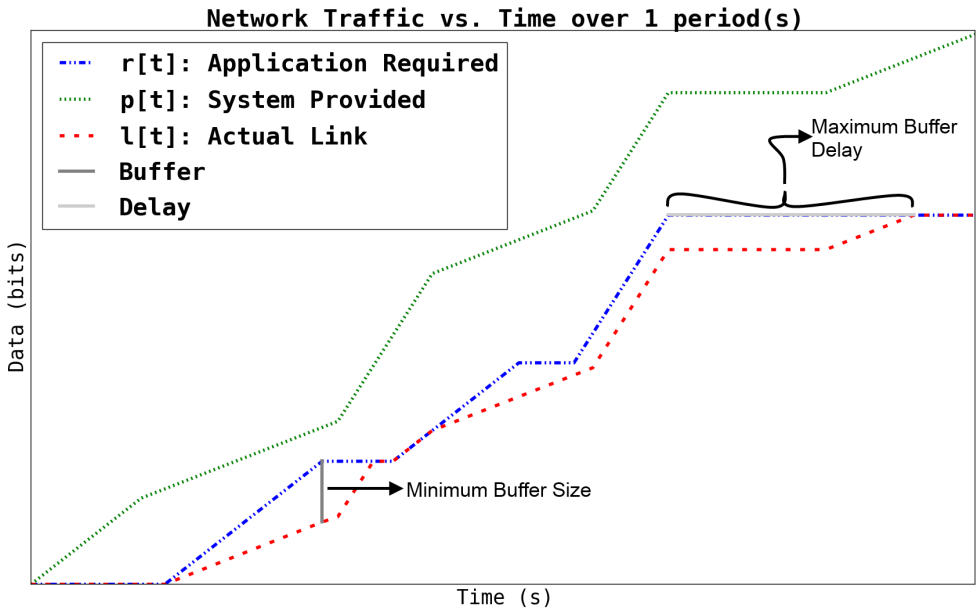


Figure 9: Representative example demonstrating convolution of the data vs. time profiles that the application requires and that the system provides. The resultant data vs. time profile describes the data that the system actually output onto the link.

Equation 13 and Equation 14 calculate, using $l[t]$, $r[t]$, and the inverse function $l^{-1}[y]$, the minimum buffer size required for the application and the maximum buffering delay experienced by application data, respectively. Figure 9 depicts the convolution operation and shows a schematic representation of the maximum buffer delay and the minimum buffer size. As can be seen in the figure, the maximum horizontal distance between the required profile and the link profile is equal to the maximum buffer delay, while the maximum vertical distance is equal to the minimum buffer size required for loss-less transmission of data on the link.

We developed the PNP^2 paradigm to precisely model system network resources as they vary with respect to time. Similarly, the application network resource requirements can be modeled very precisely as functions of time. These models can be more precise than the models developed for Network Calculus since they are explicitly functions of time, instead of functions of time-windows. Taking the example from earlier, a bulk data transfer (BDT) initiated from the satellite cluster to a ground station is scheduled for when the satellite cluster is within range of the ground station and therefore has a good downlink to the ground. With our paradigm, this coupling between the network traffic flow and the network resource availability can be captured explicitly in the model. Under the Network Calculus modeling semantics however, such a BDT would negatively affect the predicted required network buffer size and network buffering latency since that BDT data transmission window (i.e. the window of time it sends the data on the link) would be compared against the minimum service provided by the network to the ground station (which would be zero during the parts of the orbit when the ground stations are not within range of the cluster). Such a comparison drastically increases the predicted buffer space required and predicted buffering latency incurred by the BDT data.

This network modeling and analysis technique we developed has been reported in [14], which describes the network resource models, their composition, and the calculation of performance metrics such as buffer requirements and buffering delay.

3.1.3 Accuracy and Precision

When developing a new analysis technique to predict application network performance, verification that the results of the analysis are accurate is paramount. Experimental results are required not only to judge whether or not the theory is sound, but also to allow application developers and system integrators to judge the applicability of the analysis to their systems and applications. If the error in the predicted performance metrics is too high, the analysis results will cease to be useful.

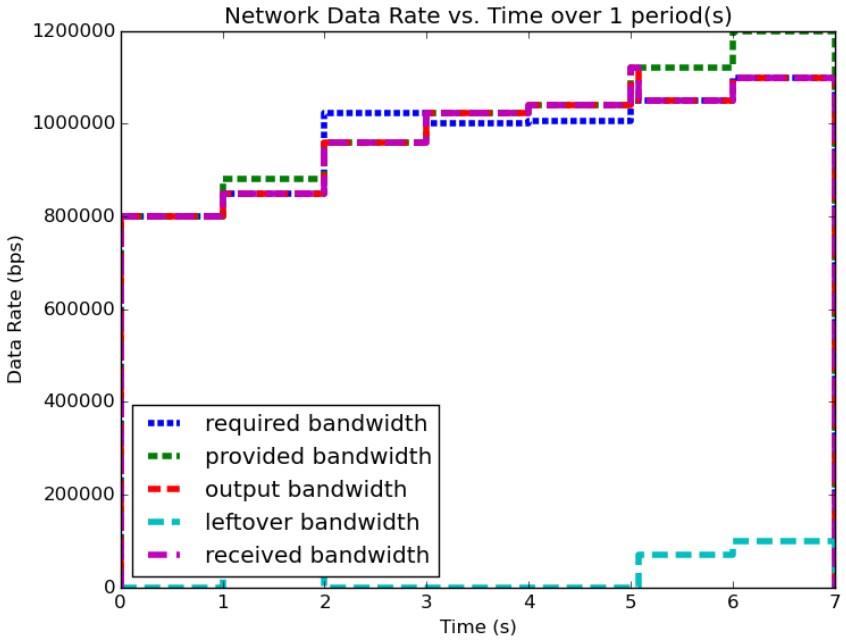
To verify the validity of these operations and metrics, we developed network measurement code which produced data for the network according to the supplied profile. This code executed on a private network testbed of nodes connected to each other through a gigabit Ethernet switch. We emulated the network link profile using traffic shaping in the Linux kernel's traffic control (TC)[32], which can be configured to control bandwidth, latency, and packet loss. For these experiments, we configured the traffic shaping to control the data rate of the application data on the network interface according to the system provided network profile.

On this testbed we ran application network traffic producer code which produces network traffic according to the supplied application profile. The profiles for the application and system are shown in Figure 10. This traffic producer code measured the delay, throughput, and buffer requirements of the traffic that was produced. By collecting these measurements over the course of multiple tests, we measured the differences between the predicted and measured buffer size and delay. The accuracy of our prediction was reported in [14] and is shown in Table 2.

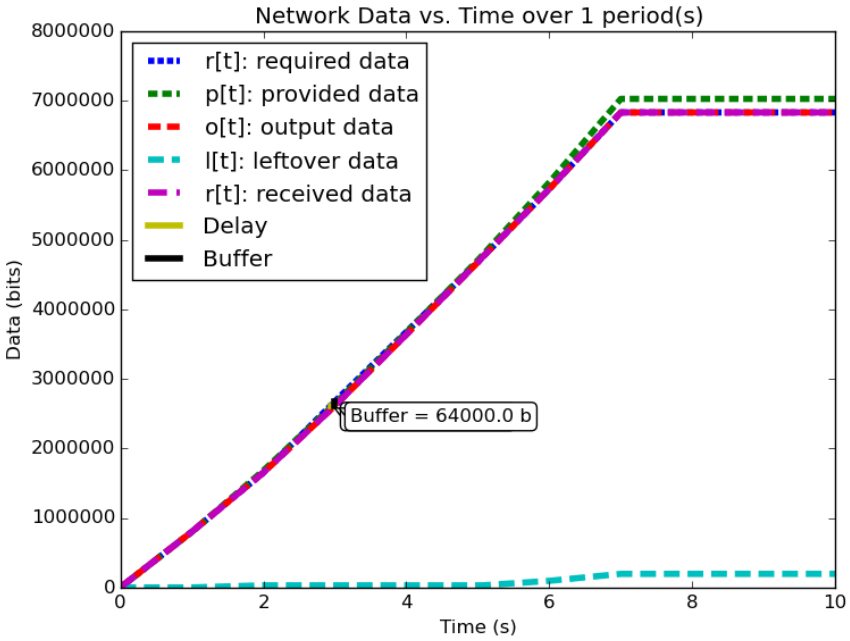
Table 1: Network utilization calculations and measured results using UDP over IPv6.

	Predicted	Measured (μ, σ)
Buffer Delay (s)	0.0625	(0.06003 , 0.00029)
Time of Delay (s)	3.0	(2.90547 , 0.00025)
Buffer Size (bytes)	8000	(7722.59 , 36.94)

As can be seen in the table, the predictions made by our analysis techniques are tight, conservative bounds on the actual performance of the application in the experimental system. Both the predicted delay and the predicted buffer size required for the application were less than 10 percent different from the actual system's required buffer size and delay.



(a) System Data Rate vs. Time



(b) System Data Analyzed with PNP^2

Figure 10: System and application profiles used for experimental validation of PNP^2. The Analysis using PNP^2 is shown on the right.

3.1.4 Assumptions Involved

As with any type of system modeling and analysis paradigm, it is important to remain aware of the types of systems the modeling/analysis is applicable to, the requirements imposed on the system by the model, and any edge cases or scenarios where the analysis or modeling paradigm breaks down.

The major assumption that we make with this type of system modeling and analysis is that we *can* know at design time what the system network capacity and the application data production will be as a (possibly periodic) function of time. Of course, this assumption is unrealistic for heavily data-dependent systems, but by performing some code analysis and/or doing some controlled experiments, models of the applications' behavior can be developed that can be analyzed.

Another key assumption and thus requirement of our modeling and analysis framework is a system-wide synchronized clock which all nodes use. By this we mean that if two nodes produce data for a third node at time $t = 3$ seconds, they will produce their data onto their respective network links at exactly the same time. This is required for the composition of profiles as they traverse the network and are routed through nodes. This assumption restricts the types of systems for which our analysis can be most useful, but is not a critical hindrance, as many such critical systems, e.g. satellite constellations or UAVs have GPS synchronized clocks, which provide such a foundation.

Another restriction with our modeling paradigm is that data-dependent flows cannot be accurately represented, since we have no way of modeling data-dependence. A related assumption is processing power and the ability of the software to adhere to the profiles: we assume the applications are able to accurately and precisely follow their data production profiles, regardless of the number of other components on their hardware node. Similarly, we assume that under all circumstances, the service profile of a hardware node will be adhered to.

Our current modeling and analysis techniques have not incorporated the concepts of

packet loss, transmission errors, and other integrity loss for data transmitted on the network. Such concepts are especially important with respect to how they influence the behavior of the network software stack, including user-space applications.

Finally, we have currently not incorporated the ways different reactive protocols would affect system network analysis. A common example of such a reactive protocol is TCP and its congestion avoidance algorithm. Because such algorithms rely on return-path information through the use of handshaking/acknowledgments they provide greater difficulty in modeling and analysis. As such, we have focused primarily on one-way transmission and reception style interactions for our modeling and analysis. Such types of interactions are found for instance in UDP transmissions.

3.1.5 Factors Impacting Analysis

It is important when developing modeling and analysis techniques to analyze how the analysis time and results are affected by changes in the model. This is especially true when trying to determine how applicable new techniques are to large scale systems. Models are provided by the application and system developers and are described in the form of bandwidth (bps) vs time that the application requires or the system provides. These profiles are a time series that maps a given time to a given bandwidth. Between two successive intervals, the bandwidth is held constant. Clearly, to represent changing bandwidth over time, the developer must use sufficiently short enough time intervals to allow step-wise approximation of the curve. However, as with any system, there is a trade-off between precision of the model and the analysis time and results.

Because the fundamental mathematics are linear for our convolution, our convolution scales with $O(n)$, where n is the total number of intervals in all of the profiles analyzed. It is worth noting that this complexity is not the same as the $O(n^2)$ or $O(n * log(n))$ complexity that traditional convolution has. This decrease in complexity is due to our convolution only requiring a single operation (comparison operation for the minimum) for each value of t.

As such, each element in both of the profiles being convolved only needs to be operated on once.

Clearly, the overall system analysis complexity depends on the complexity of the system, so as the system scales and increases routing complexity, so too will the analysis complexity. However, for all systems there is an asymptotically increasing precision for a given increase in model precision and analysis time.

3.2 Analysis of Periodic Systems

One subset of systems which we would like to analyze are periodic systems, since many systems in the real world exhibit some form of periodicity, e.g. satellites in orbit, traffic congestion patterns, power draw patterns. We define systems to be periodic if the data production rate (or consumption rate) of the system is a periodic function of time. The time-integral of these periodic data consumption/production rates is the cumulative data production/consumption of the system. These cumulative functions are called *repeating*.

Given that the required data profile and system data service profile are *repeating*, we must determine the periodicity of the output profile. If we can show that the output profile similarly repeats, then we can show that the system has no unbounded buffer growth. First, let us look at the profile behavior over the course of its first two periods of activity.

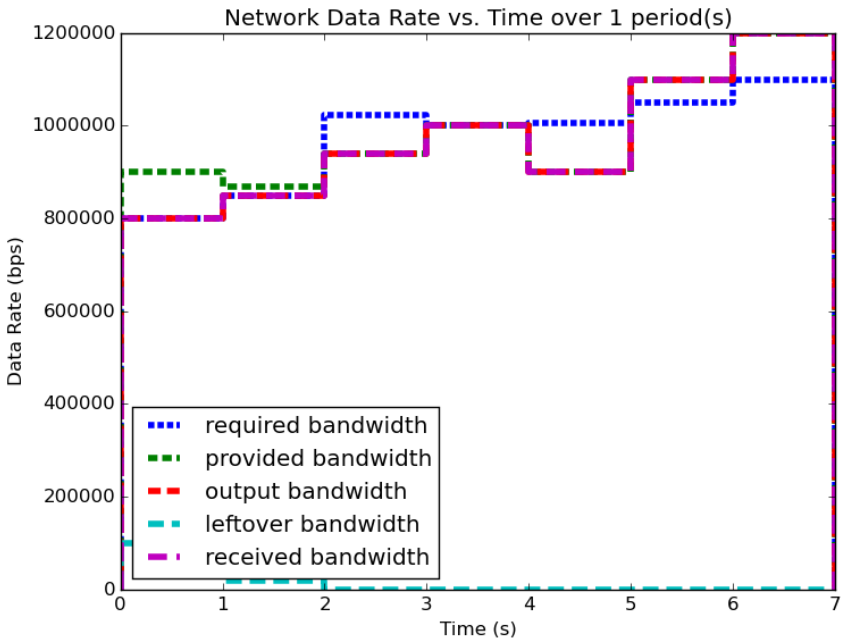
We will examine two systems, *system (1)* and *system (2)*. Firstly, examine *(1)*, shown in Figure 11, and Figure 12:

We notice that for this example system, the second period output profile is not an exact copy of the first (most easily seen by examining the bandwidth plots), and yet the required buffer size is still the same as it was when analyzing the system over one period. Furthermore, by running the analysis over even larger number of periods, we can determine (not plotted here for space and readability), that the predicted buffer size does not change no matter how many periods we analyze for this system.

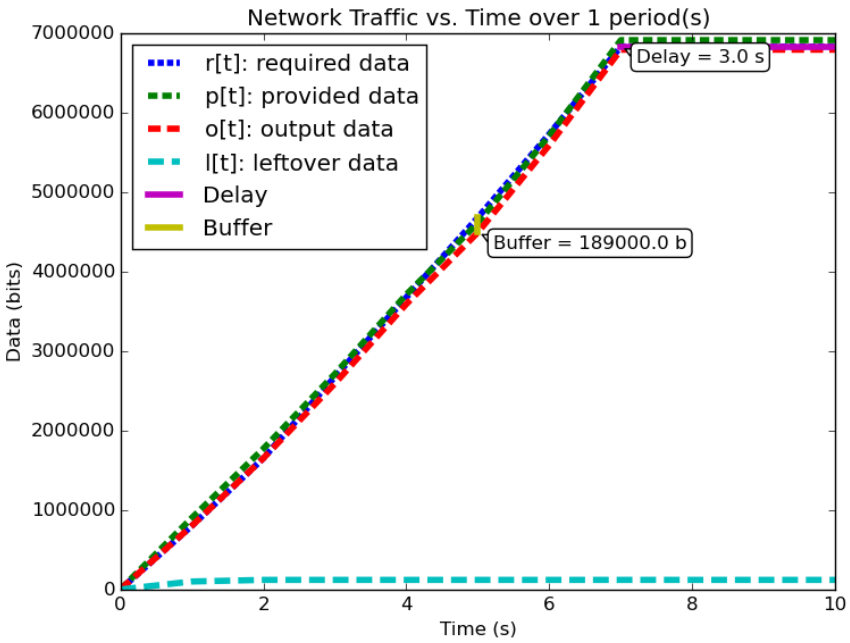
Let us look at a system where this is not the case before we begin the analysis of such system characteristics, shown in Figure 13 and Figure 14.

Notice in system *(2)*, the first period analysis predicted the same buffer size and delay as system *(1)*, but when analyzing two periods the predicted buffer size changed. Clearly the behavior of the system is changing between these two periods. If we continue to analyze more periods of system *(2)*, as we did with system *(1)*, we'll find the unfortunate conclusion that the predicted buffer size increases with every period we add to the analysis.

We have discovered a system level property that can be calculated from these profiles,

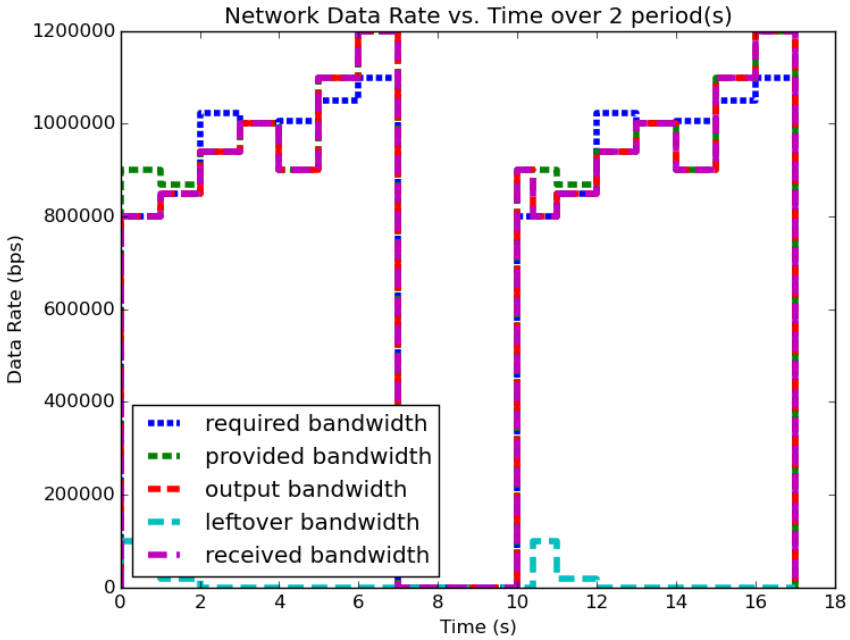


(a) System *(1)* Data Rate for 1 Period

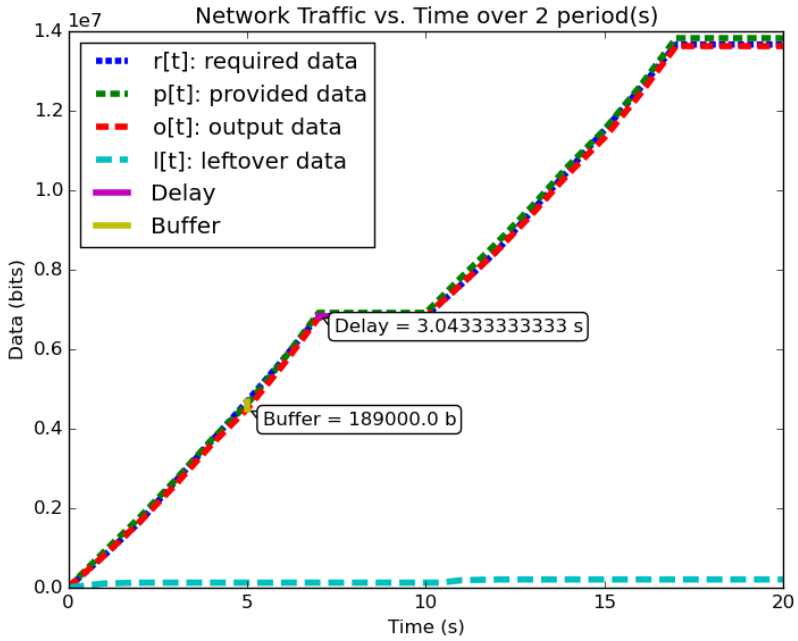


(b) System *(1)* Data for 1 Period

Figure 11: System *(1)* Analyzed over 1 Period

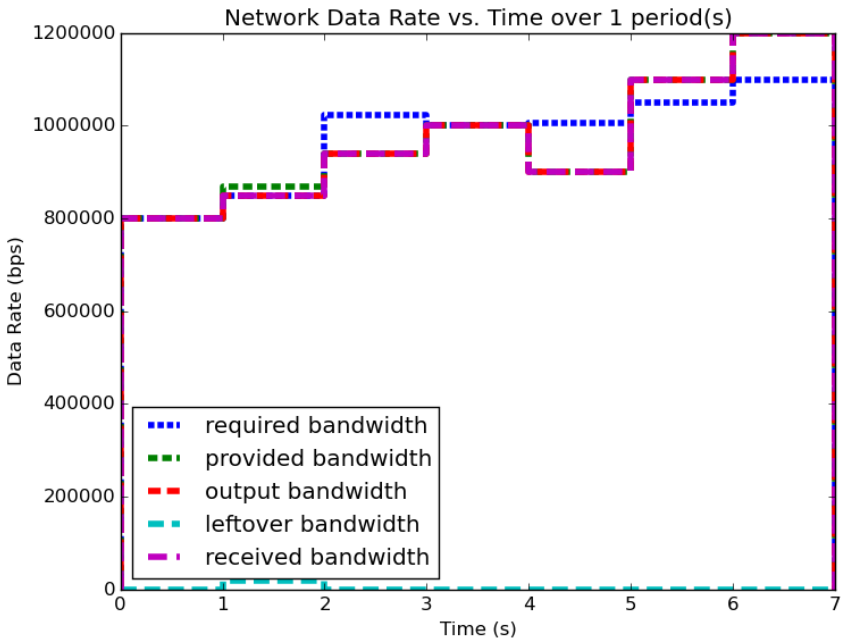


(a) System *(1)* Data Rate for 2 Periods

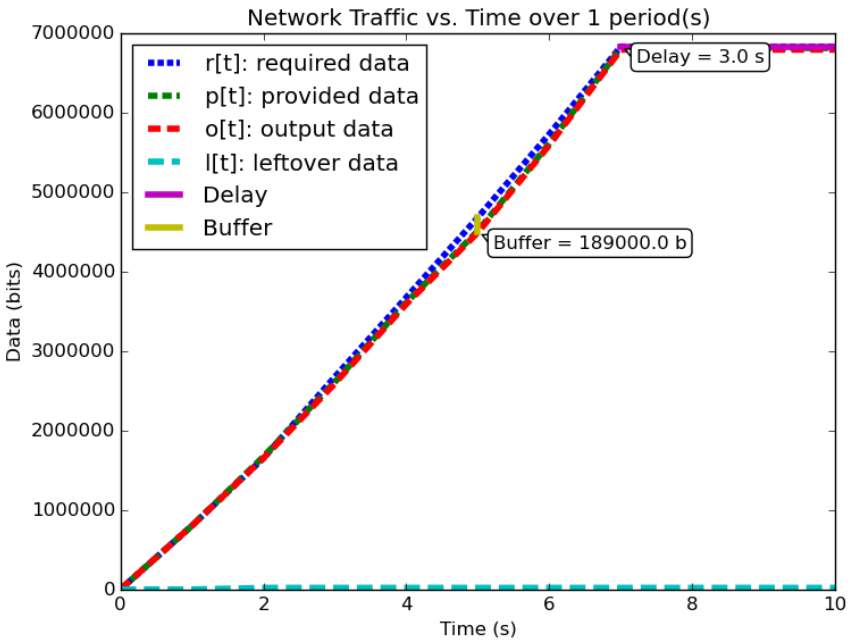


(b) System *(1)* Data for 2 Periods

Figure 12: System *(1)* Analyzed over 2 Periods

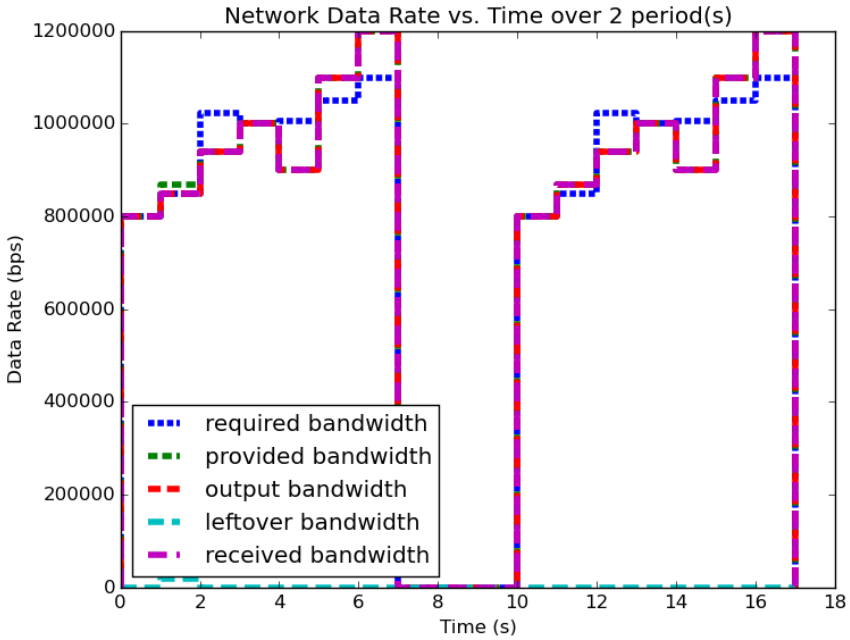


(a) System *(2)* Data Rate for 1 Period

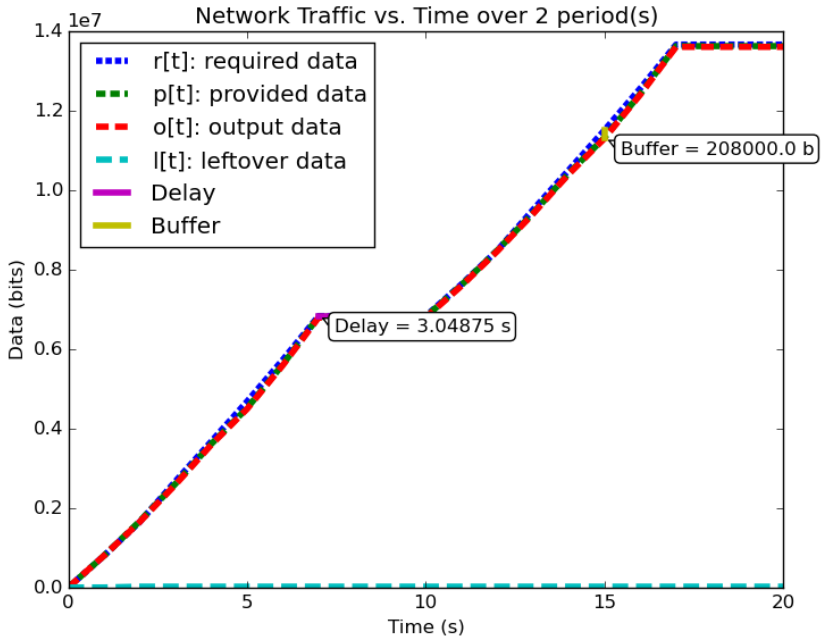


(b) System *(2)* Data for 1 Period

Figure 13: System *(2)* Analyzed over 1 Period



(a) System *(2)* Data Rate for 2 Periods



(b) System *(2)* Data for 2 Periods

Figure 14: System *(2)* Analyzed over 2 Periods

but we must determine what it means and how it can be used. First, we see that in system *(1)*, the predicted required buffer size does not change regardless of the number of periods over which we analyze the system. Second, we see that for system *(2)*, the predicted required buffer size changes depending on how many periods of activity we choose for our analysis window. Third, we see that the second period of system *(2)* contains the larger of the two predicted buffer sizes. These observations (with our understanding of deterministic periodic systems) lead us to the conclusion: system *(2)* can no longer be classified as periodic, since its behavior is not consistent between its periods. Furthermore, because the required buffer size predicted for system system *(2)* continually increases, we can determine that the system is in fact *unstable* due to unbounded buffer growth.

3.2.1 Proving the Minimum Analysis for System Stability

Let us now formally prove the assertion about system periodicity and stability which has been stated above. We will show that our analysis results provide quantitative measures about the behavior of the system and we will determine for how long we must analyze a system to glean such behaviors.

Typically, periodicity is defined for functions as the equality:

$$x(t) = x(t + k * T), \forall k \in \mathbb{N} > 0 \tag{15}$$

but for our type of system analysis this cannot hold since we deal with cumulative functions (of data vs. time). Instead we must define these functions to be repeating, where a function is repeating *iff*:

$$\begin{gathered} x(0) = 0 \text{ and} \\ x(t + k * T) = x(t) + k * x(T), \forall k \in \mathbb{N} > 0 \end{gathered} \tag{16}$$

Clearly, a repeating function x is **periodic** *iff* $x(T) = 0$. Note that repeating functions like the cumulative data vs. time profiles we deal with, are the result of **integrating** *periodic* functions, like the periodic bandwidth vs. time profiles we use to describe application network traffic and system network capacity. All periodic functions, when integrated, produce repeating functions and similarly, all repeating functions, when differentiated, produce periodic functions.

Now we will consider a deterministic, *repeating* queuing system providing a data service function S to input data function I to produce output data function O, where these functions are *cumulative data versus time*. At any time t, the amount of data in the system's buffer is given by B_t. After servicing the input, the system has a remaining capacity function R.

- $S[t]$: the service function of the system, cumulative data service capacity versus time
- $I[t]$: the input data to the system, cumulative data versus time
- $O[t]$: the output data from the system, cumulative data versus time
- $B[t]$: the amount of data in the system's buffer at time t, i.e. $I[t] - O[t]$
- $R[t]$: the remaining service capacity of the system after servicing I, i.e. $S[t] - O[t]$

Because S and I are deterministic and repeating, they increase deterministically from period to period, i.e. given the period T_I of I,

$$\forall t, \forall n \in \mathbb{N} > 0 : I[t + n * T_I] = I[t] + n * I[T_I] \tag{17}$$

Similarly, given the period T_S of S,

$$\forall t, \forall n \in \mathbb{N} > 0 : S[t + n * T_S] = S[t] + n * S[T_S] \tag{18}$$

We can determine the hyperperiod of the system as the *lcm* of input function period and the service function period, $T_p = lcm(T_S, T_I)$.

At the start of the system, $t = 0$, the system's buffer is empty, i.e. $B[0] = 0$. Therefore, the amount of data in the buffer at the end of the first period, $t = T_p$, is the amount of data that entered the system on input function I but was not able to be serviced by S. At the start of the next period, this data will exist in the buffer. Data in the buffer at the start of the period can be compared to the system's remaining capacity R, since the remaining capacity of the system indicates how much extra data it can transmit in that period. Consider the scenario that the system's remaining capacity R is less than the size of the buffer, i.e. $R[T_p] < B[T_p]$. In this scenario, $B[2 * T_p] > B[T_p]$, i.e. there will be more data in the buffer at the end of the second period than there was at the end of the first period. Since the system is deterministic, for any two successive periods, $n * T_p$ and $(n+1) * T_p$, $B[n * T_p] > B[(n+1) * T_p]$, which extends to:

$$B[m * T_p] > B[n * T_p], \forall m > n > 0 \tag{19}$$

implying that:

$$B[t] < B[t + k * T_p], \forall k \in \mathbb{N} > 0 \tag{20}$$

meaning that the amount of data in the buffer versus time is *not periodic*, therefore the amount of data in the system's buffer increases every period, i.e. the system has *unbounded buffer growth*.

If however, there is enough remaining capacity in the system to service the data in the buffer, i.e. $R[T_p] >= B[T_p]$, then $B[2 * T_p] = B[T_p]$. This relation means that if the remaining capacity of the system that exists after all the period's required traffic has been serviced is equal to or larger than the size of the buffer at the end of the period, then in the next period the system will be able to service fully both the data in the buffer and the period's

required traffic. Since both the period's traffic and the buffer's data will have been serviced in that period, the amount of data in the buffer at the end of the period will be the same as the amount of data that was in the buffer at the start of the period. Similarly to above, since the system is deterministic, for any two successive periods, $n * T_p$ and $(n+1) * T_p$, $B[(n+1) * T_p] = B[n * T_p]$. This extends to:

$$B[m * T_p] = B[n * T_p], \forall m, n > 0 \tag{21}$$

which implies that:

$$B[t] = B[t + k * T_p], \forall k \in \mathbb{N} > 0 \tag{22}$$

meaning that the amount of data in the buffer versus time is a *periodic function*, therefore the maximum buffer size does not grow between periods, and the system has a *finite buffer*.

If we are only concerned with buffer growth, we do not need to calculate R, and can instead infer buffer growth by comparing the values of the buffer at any two period-offset times during the steady-state operation of the system ($t >= T_p$). This means that the system buffer growth check can resolve to $B[2 * T_p] == B[T_p]$. This comparison abides by the conditions above, with $m = 2$ and $n = 1$.

3.3 Comparison of PNP^2 with Network Calculus

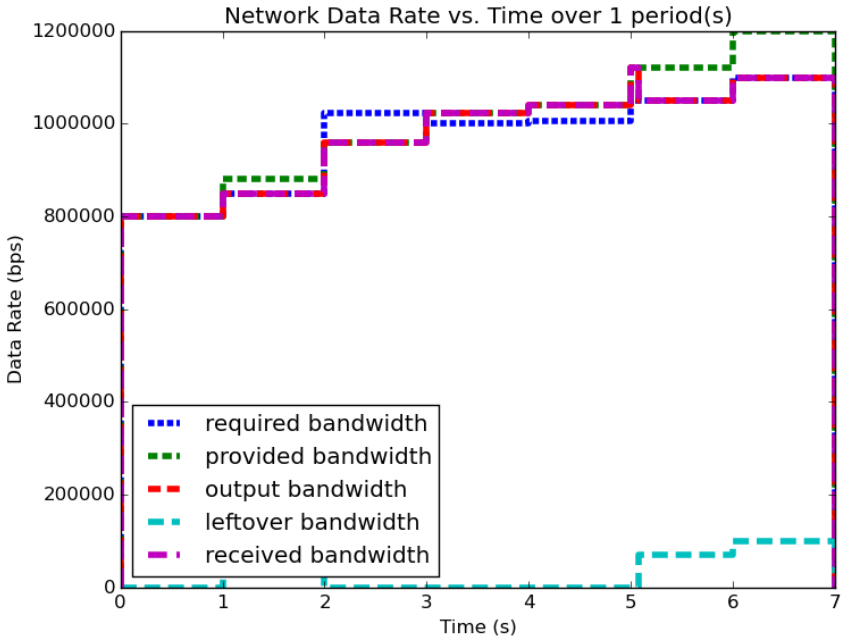
When developing a new analysis technique to predict application network performance, alternative techniques must be evaluated to determine the utility of the new techniques. Application developers and system integrators can then use these comparisons as a metric for choosing between the available analysis tools. For the tools and techniques to affect a meaningful change in system and application development, they must be shown to be more effective by some metric for at least certain classes of systems or applications.

To show how our analysis techniques compare to other available methods, we developed our tools to allow us to analyze the input system using Network Calculus/Real-Time Calculus techniques as well as our own. Using these capabilities, we can directly compare the analysis results to each other, and then finally compare both results to the measurements from an actual system.

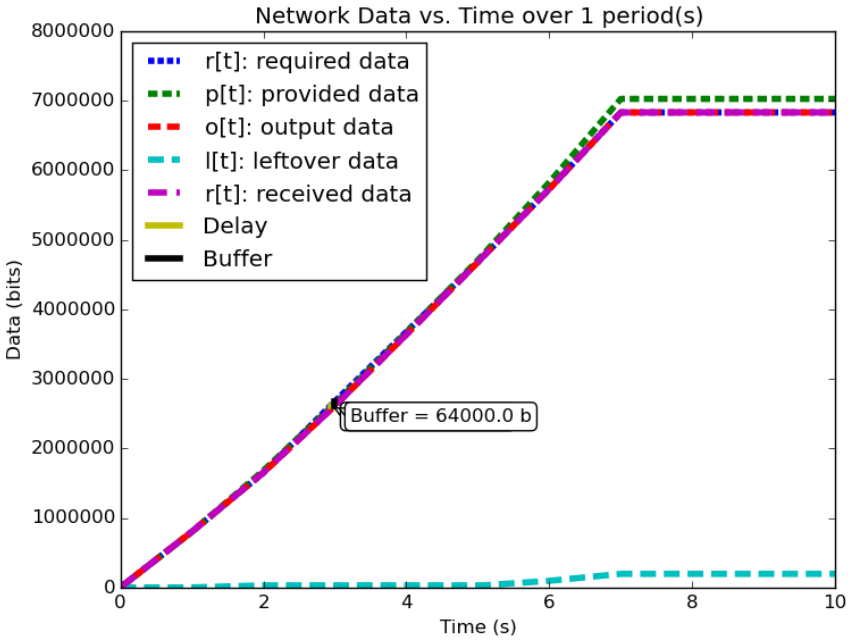
3.3.1 Results

Figure 15 shows the data rate versus time profile describing the example system, side-by-side with the time-integrated and analyzed data versus time profile. Figure 16 shows a zoomed in portion of the second plot, focusing on the area with the maximum delay and buffer as analyzed by PNP^2. Figure 17 shows the same system analyzed using Network Calculus.

The major drawback for Network Calculus that our work aims to solve is the disconnect from the real system that stems from using an approach based on time-window analysis. Such an approach leads to dramatically under-approximating the capacity of the network while simultaneously over-approximating the utilization of the network, since a known drop in network performance which is expected and handled by the application cannot be accurately modeled. In our case, the system is using a system profile which can service data during the period from $0 \leq t \leq 7$ seconds with a period of 10 seconds. The application is designed around this constraint and only produces data during that interval. Because our



(a) System Data Rate vs. Time



(b) System Data Analyzed with PNP^2

Figure 15: System profile used for comparison of PNP^2 with Network Calculus. The Analysis using PNP^2 is shown on the right.

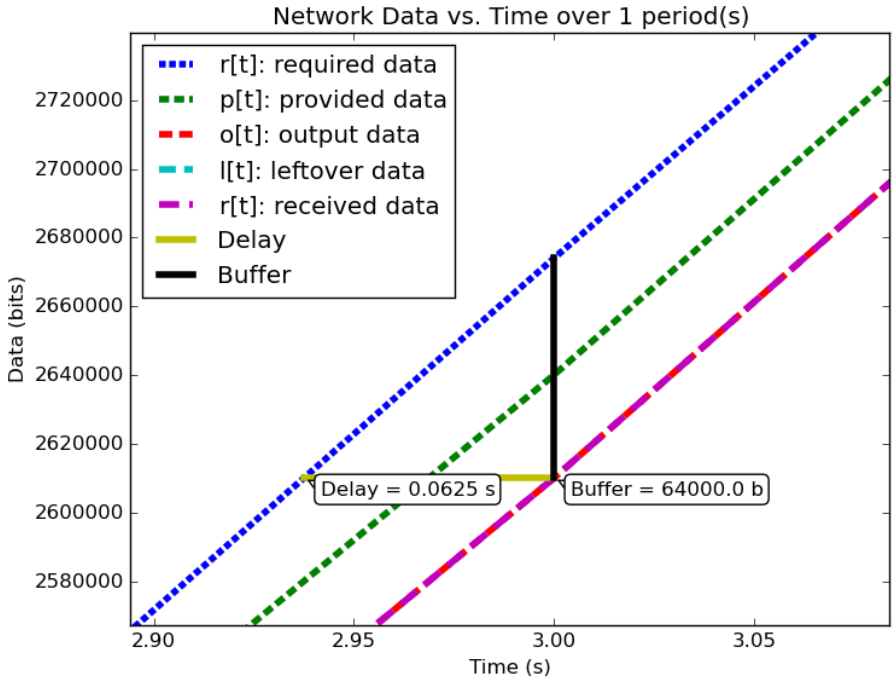


Figure 16: Zoomed-in version of Figure III.15(b), focusing on the predicted buffer and delay.

technique directly compares when the application produces data to when the system can service the data, we are able to derive more precise performance prediction metrics than Network Calculus, which compares the 3 seconds of system downtime to the 3 seconds of maximum application data production.

Using the same testbed, traffic production software, and traffic measurement software described in Section 3.1.3, we were able to measure the transmitted traffic profile, the received traffic profile, the latency experienced by the data, and the transmitter's buffer requirements. The results are displayed in Table 2 (from the same experimental data as in Section 3.1.3):

Taking the results from our published work, where our methods predicted a buffer size of 64000 bits, we show that Network Calculus predicts a required buffer size of 3155000 bits. This drastic difference comes from the mis-match between down-time and max data production mentioned above. Note also that Network Calculus does not provide a way

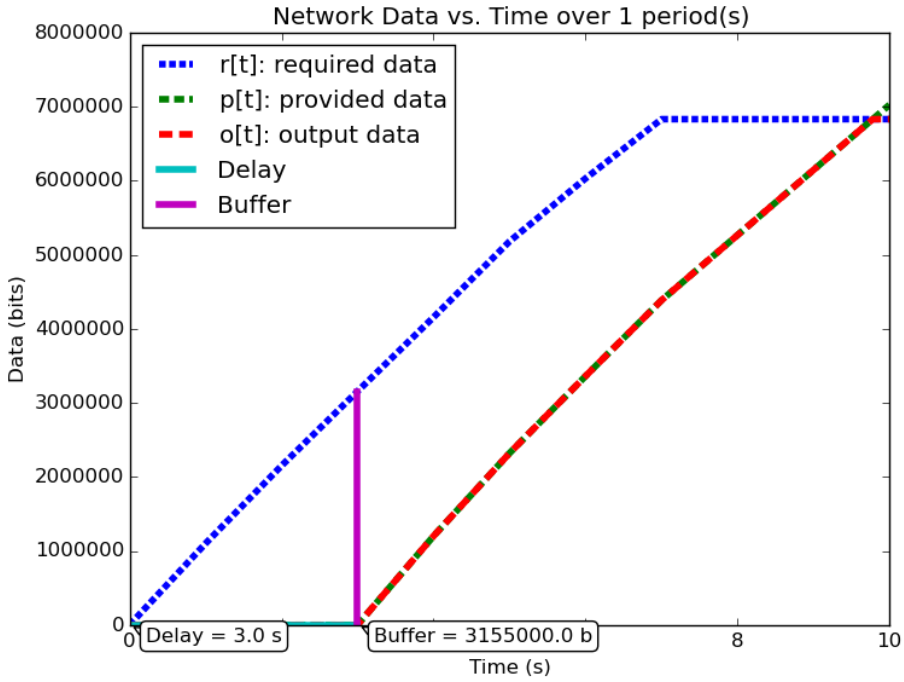


Figure 17: Network-Calculus based analysis of the same system.

Table 2: Experimental system measurements

	Network Calculus	PNP^2	Measured (μ, σ)
Buffer Delay (s)	3.0	0.0625	(0.06003 , 0.00029)
Time of Delay (s)	N/A	3.0	(2.90547 , 0.00025)
Buffer Size (bits)	3155000	64000	(61780.72 , 295.52)

to predict when (during the run-time of the applications) the maximum delay will occur. Network Calculus lacks this capability because it defines its models as functions of time-window size instead of as direct functions of time.

3.4 Analysis of TDMA Scheduling

Medium channel access protocols are used in networking systems to govern the communication between computing nodes which share a network communications medium. They are designed to allow reliable communication between the nodes, while maintaining certain goals, such as minimizing network collisions, maximizing bandwidth, or maximizing the number of nodes the network can handle. Such protocols include Time Division Multiple Access (TDMA), which tries to minimize the number of packet collisions; Frequency Division Multiple Access (FDMA), which tries to maximize the bandwidth available to each transmitter; and Code Division Multiple Access (CDMA) which tries to maximize the number of nodes that the network can handle[25]. We will not discuss CDMA in the scope of this work.

In FDMA, each node of the network is assigned a different transmission frequency from a prescribed frequency band allocated for system communications. Since each node transmits on its own frequency, collisions between nodes transmitting simultaneously are reduced. Communications paradigms of this type, i.e. shared medium with collision-free simultaneous transmission between nodes, can be modeled easily by our PNP^2 modeling paradigm described above, since the network resource model for each node can be developed without taking into account the transmissions of other nodes.

In TDMA, each node on the network is assigned one or more time-slots per communications period in which only that node is allowed to transmit. By governing these timeslots and having each node agree upon the slot allocation and communications period, the protocol ensures that at a given time, only a single node will be transmitting data, minimizing the number of collisions due to multiple simultaneous transmitters. In such a medium access protocol, transmissions of each node affect other nodes' transmission capability. Because these transmissions are scheduled by TDMA, they can be explicitly integrated into the system network resource model.

3.4.1 Problem

TDMA transmission scheduling has an impact on the timing characteristics of the applications' network communications. Because applications' network data production is decoupled from their node's TDMA transmission time slot, buffering may be required when an application on one node tries to send data on the network during the transmission slot of a different node. In this case, the data would need to be buffered on the application's node and would therefore incur additional buffering delay. If this TDMA schedule is not integrated into the analysis of the network resources, the additional buffer space required may exceed the buffer space allocation given to the application or the buffering delay may exceed the application's acceptable latency.

3.4.2 Results

So far, the description of the system provided network service profile ($p[t] = y$), has been abstracted as simply the available bandwidth as a function of time integrated to produce the amount of data serviced as a function of time. We show how to model and analyze the network's lower-level TDMA MAC protocol using our network modeling semantics. We then derive general formulas for determining the affect TDMA has on buffer size and delay predictions.

As an example TDMA system which benefits from our analysis techniques, consider an application platform provided by a fractionated satellite cluster. For this system, the network between these satellites is a precious resource shared between each of the applications' components in the cluster. To ensure the stability of the network resources, each satellite has a direct connection to every other satellite and is assigned a slot in the TDMA schedule during which the satellite may transmit. Each TDMA slot has a sinusoidally time-varying bandwidth profile which may differ from the other TDMA slot bandwidth profiles. The time-varying profile of the slot bandwidth comes from the coupling between

the radios' inverse-squared bandwidth-as-a-function-of-distance and the satellites' sinusoidal distance-as-a-function-of-orbital-position, as described at the beginning of Chapter III. The requirement for accurate performance prediction necessitates the incorporation of the TDMA schedule into the network modeling and analysis.

TDMA schedules can be described by their period, their number of slots, and the bandwidth available to each slot as a function of time. For simplicity of explanation, we assume that each node only gets a single slot in the TDMA period and all slots have the same length, but the results are valid for all static TDMA schedules. Note that each slot still has a bandwidth profile which varies as a function of time and that each slots may have a different bandwidth profile.

In a given TDMA period T, a node n can transmit a certain number of bits governed by its slot length t_n and the slot's available bandwidth bw_n. During the rest of the TDMA period, the node's available bandwidth is 0. This scheduling has the effect of amortizing the node's slot bandwidth into an effective bandwidth of $bw_{effective}$. The addition of the TDMA scheduling can affect the buffer and delay calculations, based on the slot's bandwidth, the number of slots, and the slot length. The maximum additional delay is Δ_{delay}, and the maximum additional buffer space is Δ_{buffer}. These deviations are shown graphically by Figure 18 and calculated by

$$\begin{aligned} bw_{effective} &= bw_n * \frac{t_n}{T} \\ \Delta_{delay} &= T - t_n \\ \Delta_{buffer} &= \Delta_{delay} * bw_{effective} \end{aligned} \tag{23}$$

Where:

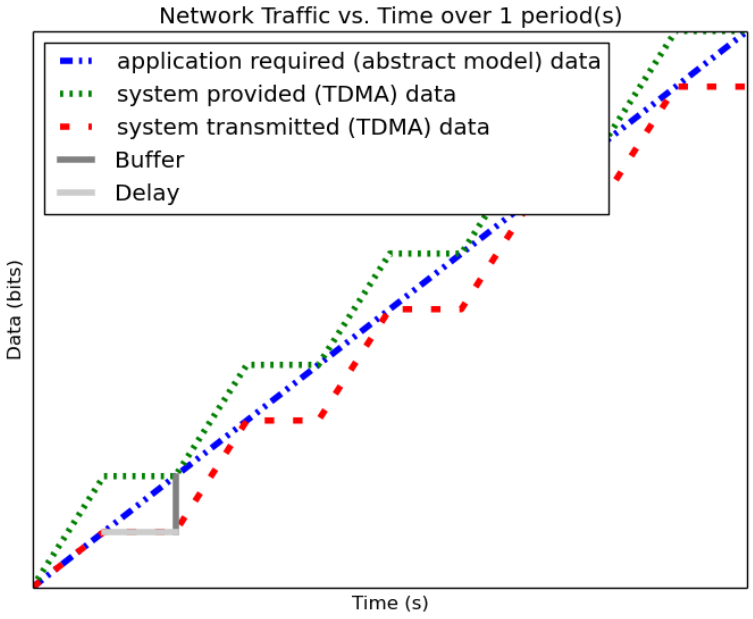
- T is the period of the TDMA schedule
- t_n is the length of node n's TDMA slot
- bw_n is the bandwidth available to node n during its slot

- $bw_{effective}$ is the perceived bandwidth available to the node during the TDMA period
- Δ_{delay} is the change in the predicted delay experienced by application traffic on the network
- Δ_{buffer} is the change in the predicted buffer space required for lossless transmission of application traffic

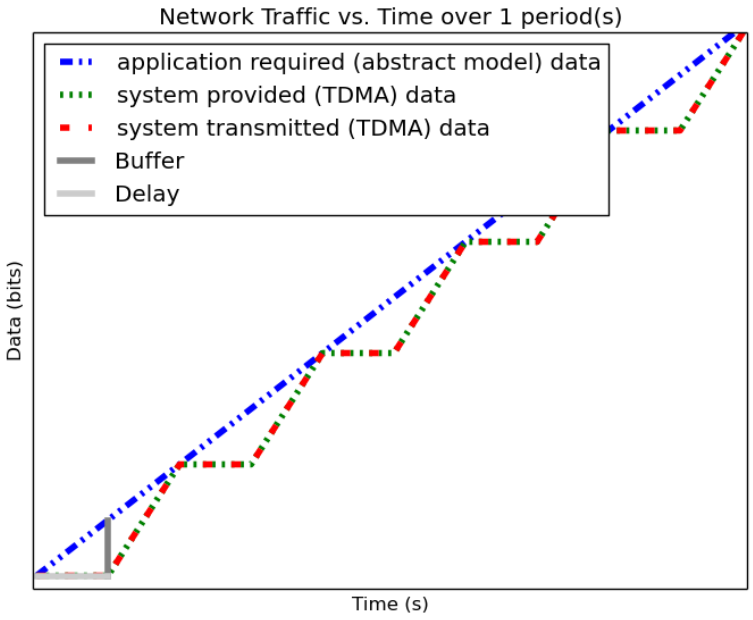
Clearly, Δ_{delay} is bounded by T and Δ_{buffer} is governed by t_n. Therefore, because t_n is dependent on T, minimizing T minimizes both the maximum extra delay and maximum extra buffer space.

Following from this analysis, we see that if: (1) the TDMA effective bandwidth profile is provided as the abstract system network service profile, and (2) the TDMA period is much smaller than the duration of the shortest profile interval; then the system with explicit modeling of the TDMA schedule has similar predicted application network characteristics as the abstract system. Additionally, the maximum deviation formulas derived above provide a means for application developers to analyze the their application on a TDMA system without explicitly integrating the TDMA model into the system profile model.

Through the analysis of TDMA scheduling's effect on application level performance prediction, we derived analytical formulas for the maximum deviation between the abstract system model and the model with explicitly encoded TDMA scheduling. The use of these formulas frees developers and system integrators from having to explicitly incorporate the TDMA schedule in their application and system models. This TDMA modeling and analysis was published in [13].



(a) In-Phase TDMA Profile vs Abstract



(b) Out-of-Phase TDMA Profile vs Abstract

Figure 18: Effects of TDMA scheduling in the MAC layer on system network performance.

3.5 Compositional Analysis

Now that we have precise network performance analysis for aggregate profiles or singular profiles on individual nodes of the network, we must determine how best to compose these profiles and nodes together to analyze the overall system. The aim of this work is to allow the profiles from each application to be analyzed separately from the other profiles in the network, so that application developers and system integrators can derive meaningful performance predictions for specific applications. For this goal, let us define:

Compositionality [21] A system is compositional if its properties can be derived from the properties of its components and how they are interconnected.

Composability [21] A component is composable if its properties do not change when the component is composed with other components.

For our analysis techniques to be compositional, an application's required profile must be analyzable individually without requiring aggregation with the rest of the required profiles in the system. This means that the system's performance, i.e. the performance of all the applications on the system, can be determined by analyzing the performance of each application individually.

To achieve compositionality, we must not only define mathematical operations which allow us to aggregate and separate profiles with/from each other, but also the semantics of how these profiles are composed with one another. This semantics govern the relation between required profiles, specifically governing the distribution of their shared node's provided profile between each other. For our compositional analysis, we defined that each required profile in the system be given a unique priority, U, with the relation that a profile P_1 has a higher priority than profile P_2 *iff* $U_{P_1} < U_{P_2}$. Using this priority relation, we can define that a profile P_i does not receive any capacity from its node at time t until all other profiles with priority $< U_{P_i}$ have received their requested capacity from the node at t. If the node does not have enough capacity at t to service P_i, then the data P_i attempted to send at

t will be placed into its buffer, to be sent at a time when the node has available bandwidth for P_i.

This priority relation for compositional analysis is similar to the task priority used for schedulability analysis in Real-Time Calculus, mentioned in Section 2.1.2.3. Similarly to RTC, this priority relation and compositionality allow us to capture the effects independent profiles have on each other when they share the same network resources. Just as RTC based its priority relation and computation scheduling on a fixed-priority scheduler, our priority relation and resource allotment is based on the network Quality-of-Service (QoS) management provided by different types of networking infrastructure. One such mechanism for implementing this type of priority-based network resource allocation is through the use of the DiffServ Code Point (DSCP)[40]. The DSCP is a bit-field in all packets which have an Internet Protocol (IP) header which allows the packet to be assigned a specific class for per-hop routing behavior. Routers and forwarders in the network group packets according to their DSCP class and provide different service capacities to each class. For example, the *Expedited Forwarding* [11] class receives strict priority queuing above all other traffic, which makes it a suitable implementation of this type of resource allocation. Similarly, the Linux Traffic Control[32] utility provides many mechanisms for shaping, policing, routing, and classifying traffic. Its class-based queuing disciplines and filtering mechanisms provide the capability for such strict priority-based network resource allocation.

Mathematically, compositionality requires that we be able to add and subtract profiles from each other, for instance to determine the remaining service capacity of a node available for a profile P_i after it serves all profiles with a higher priority. Queuing of the lower priority profiles is taken into account when the lower priority profile is convolved with the remaining capacity the node has available to service it. The calculation of the remaining capacity, P'_P, of the node after it services P_i is given as:

$$P'_P = P_P - (P_i \otimes P_P) \tag{24}$$

Where

- P_P is the capacity available to profile P_i

Mathematically, addition and subtraction of two profiles $f[t], g[t]$ are given by:

$$s[t] = f[t] + g[t] \tag{25}$$

and

$$s[t] = f[t] - g[t] \tag{26}$$

Experimental validation of these compositional techniques, specifically with respect to priority relation, adding, and subtracting of profiles is presented at the end of Section 3.7.

3.6 Delay Analysis

When dealing with queuing systems (esp. networks) where precise design-time guarantees are required, the delay in the links of the network must be taken into account.

The delay is modeled as a continuous function of latency (seconds) versus time. In the profiles, the latency is specified discretely as $(time, latency)$ pairs, and is interpolated linearly between successive pairs. Specifically, *time* is a time point at which the latency on the link is given by $latency$.

Using this latency semantics, the delay convolution of a profile becomes

$$r[t + \delta[t]] = l[t] \tag{27}$$

Where

- $l[t]$ is the *link* profile describing the data as a function of time as it enters the link
- $\delta[t]$ is the *delay* profile describing the latency as a function of time on the link
- $r[t]$ is the *received* profile describing the data as a function of time as it is received at the end of the link

When analyzing delay in a periodic system, it is important to determine the effects of delay on the system's periodicity. We know that the period of the periodic profiles is defined by the time difference between the start of the profile and the end of the profile. Therefore, we can show that if the time difference between the **start time** of the *received* profile and the **end time** of the *received* profile is the same as the **period** of the *link* profile, the periodicity of the profile is unchanged.

- T_p is the period of the *link* profile
- $r[t + \delta[t]]$ is the beginning of the *received* profile
- $r[(t + T_p) + \delta[(t + T_p)]]$ is the end of the *received* profile

We determine the condition for which $(t_{end}) - (t_{start}) = T_p$:

$$\begin{aligned}(T_p + t + \delta[T_p + t]) - (t + \delta[t]) &= T_p \\ T_p + \delta[T_p + t] - \delta[t] &= T_p \\ \delta[T_p + t] - \delta[t] &= 0 \\ \delta[T_p + t] &= \delta[t]\end{aligned} \tag{28}$$

Which is just confirms that the periodicity of the delayed profile is unchanged *iff* the latency profile is **periodic**, i.e.

$$\delta[t] = \delta[t + k * T_p], \forall k \in \mathbb{N} > 0 \tag{29}$$

Experimental validation of this delay analysis is presented at the end of Section 3.7.

3.7 Analysis of Statically Routed Networks

3.7.1 Problem

As CPS become more distributed in nature and begin to act as infrastructure for distributed applications towards IoT systems, they will necessarily need to handle more network resource management and network connection routing within their network as well as between their own network and any external networks to which they are connected. Such networks generally rely on routing to allow more flexibility in the system with respect to node placement and connectivity. Adding routing to the network also has the effect of increasing the complexity of the network performance analysis and can cause drastic differences in application network performance when compared with networks without routing. Therefore the design-time analysis tools which help predict application network performance must take this routing into account. It should be noted that this is a special case of routing in ad-hoc networks, where one or more nodes can route messages for other nodes.

3.7.2 Results

Having discussed profile composition and the affects of delaying a profile, we can address one more aspect of system analysis: *routing*. For this analysis we will specifically focus on statically routed networks.

Firstly, we must define the assumptions we make about the router nodes with respect to how they forward the network traffic. In our modeling and analysis, because we have not considered transmission error/corruption, we are most closely modeling cut-through routing / wormhole switching in which the routing and forwarding nodes in the system forward all packets without checking them for corruption or integrity. This forwarding mechanism differs from store and forward routing in which each packet is checked for errors in its entirety before sending it to the next node on its route. In the case of store and forward, when a corrupt packet is received by a routing node, it will not forward that packet along its path, and may optionally request re-transmission of the packet from the

previous node. Under the assumption of no transmission errors, we can incorporate the added latency incurred by store and forward into the latency profile of the router node. In this way, these two forwarding techniques can be modeling in a simple way using our semantics (where they both simply affect the latency of the node).

Given these assumptions about the forwarding techniques of the routing nodes, we can describe system-level analysis. By incorporating both the latency analysis with the compositional operations we developed, we can perform system-level analysis of profiles which are routed by nodes of the system. In this paradigm, nodes can transmit/receive their own data, i.e. they can host applications which act as data sources or sinks, as well as act as routers for profiles from and to other nodes. To make such a system amenable to analysis we must ensure that we know the routes the profiles will take at design time, i.e. the routes in the network are static and known or calculable. Furthermore, we must, for the sake of profile composition as described above, ensure that each profile has a priority that is unique within the network which governs how the transmitting and routing nodes handle the profile's data.

Let us define the system configuration C as:

$$C = \{\{P_S\}, \{N\}, \{R\}\} \tag{30}$$

Where

- $\{P_S\}$ is the *set* of all *sender* profiles in the system configuration
- $\{N\}$ is the *set* of all *nodes* in the system configuration, and
- $\{R\}$ is the *set* of all *routes* in the system configuration

We define a profile P as:

$$P = \{N_I, K, T, F, U, \{(t, R_D, D, L)\}\} \tag{31}$$

Where

- N_I is the *Node ID* to which the profile applies
- K is the *kind* of the profile, where $K \in \{provided, required, receiver\}$
- T is the *period* of the profile
- F is the *flow ID* of the profile, where two profiles, P_1, P_2 belong to the same flow *iff* $F_{P_1} == F_{P_2}$
- U is the *priority* of the profile, where profile P_1 has a higher priority than profile P_2 *iff* $U_{P_1} < U_{P_2}$, and
- $\{(t, R_D, D, L)\}$ is a *set* of $(time, data\ rate, data, latency)$ tuples describing how each of $\{data\ rate, data, latency\}$ vary with respect to time. Semantically, the $data\ rate$ is constant between any two successive values of t, while the $data$ and $latency$ are *linearly interpolated* during the same interval. The initial profile specification does not have the $data$ field; $data$ is calculated based on $data\ rate$.

Then we define a node N as:

$$N = \{I, P_P, \{P_R\}\} \tag{32}$$

Where

- I is the *ID* of the node
- P_P is the *provided* profile of the node, and
- $\{P_R\}$ is the *set* of all *receiver* profiles on the node

And finally, we define a route R as:

$$R = \{N_{I_1}, N_{I_2}, ..., N_{I_N}\} \tag{33}$$

Where

$$\forall N_X, N_Y \subset N, \exists! R_{X,Y} = \{N_{I_X}, ..., N_{I_Y}\} \tag{34}$$

We can then run Algorithm 1 to iteratively analyze the system. In this algorithm, the remaining capacity of the node is provided to each profile with a lower priority iteratively. Because of this iterative recalculation of node provided profiles based on routed profiles, we directly take into account the effect of multiple independent profiles traversing the same router; the highest priority profile receives as much bandwidth as the router can give it, the next highest priority profile receives the remaining bandwidth, and so on.

We take care of matching all senders to their respective receivers, and ensure that if the system supports multicast, a no re-transmissions occur; only nodes which must route the profile to a new part of the network re-transmit the data. However, if the system does not support multicast, then the sender must issue a separate transmission, further consuming network resources. In this way, lower-level transport capabilities can be at least partially accounted for by our analysis.

We have implemented these functions for statically routed network analysis into our tool, which automatically parses the profiles, the network configuration and uses the algorithm and the implemented mathematics to iteratively analyze the network. Analytical results for example systems will be provided when the experimental results can be used as a comparison.

To determine the validity of our routing, composition, and delay analysis, we developed a sample system and application deployment consisting of two flows generated by two profiles, one a high priority flow and one a low priority flow. Each flow originates on a separate computing node, with different destinations. Both flows are routed through

```
analyze( sender_profiles )
{
  sender_profiles = sorted(sender_profiles, priority)
  for required_profile in sender_profiles
  {
    transmitted_nodes = list.empty()
    for receiver_profile in
        required_profile.receiver_profiles()
    {
      route =
        getRoute(required_profile, receiver_profile)
      for node in route
      {
        if node in transmitted_nodes
          and multicast == true
        {
          continue
        }
        provided_profile = node.provided_profile

        output_profile =
          convolve(required_profile, provided_profile)
        remaining_profile =
          provided_profile - output_profile
        received_profile =
          delay(output_profile, provided_profile)

        node.provided_profile = remaining_profile
        required_profile = received_profile
        transmitted_nodes.append(node)
      }
      receiver_received_profile =
        convolve(required_profile, receiver_profile)
    }
  }
}
```

Listing 1: Algorithm for iteratively analyzing profiles in a distributed system with static routing and profile priorities.

the same routing node that enforces priority-based routing for the two flows. Figure 19

shows the configuration of the system and application for the experimental validation of the routing, composition, and delay analysis techniques.

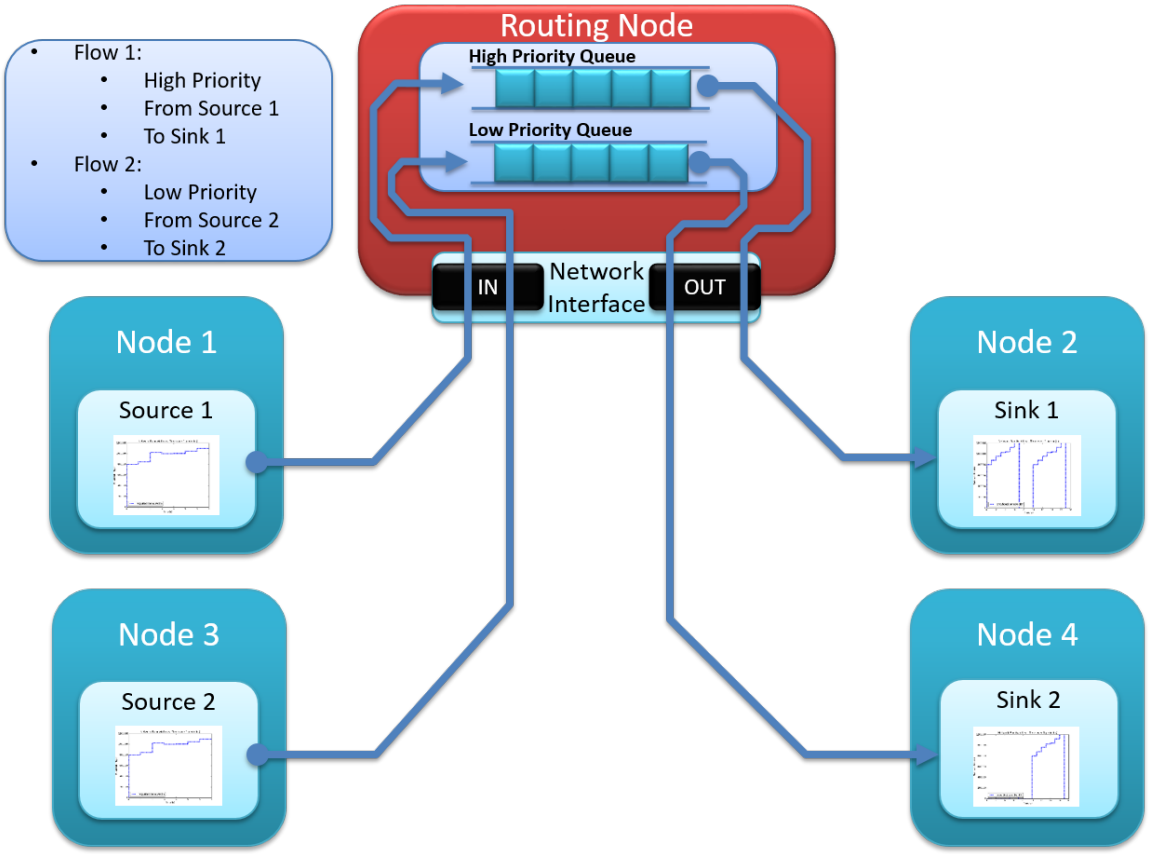


Figure 19: Experimental setup to validate routing, delay, and compositional analysis of network profiles.

For this experimental setup, we configured the Linux kernel using TC[32] which provides mechanisms for implementing traffic prioritization, shaping, and delay, among other features. All application traffic on each node passed through shaping and delay queues, which shaped the application traffic according to the properties of its system profile. Additionally, for the router node we configured priority queuing which filtered the application traffic into a high priority queue and a low priority queue. These queues are dequeued in the kernel according to priority FIFO, which means that data will not be dequeued from lower priority queues unless all high priority queues are empty. These priority queues feed

into traffic shaper and delay queues, to enforce the system profile on the traffic. This configuration is shown schematically in Figure 20. A more detailed description of the specific configuration and operation of TC is given in Appendix B.

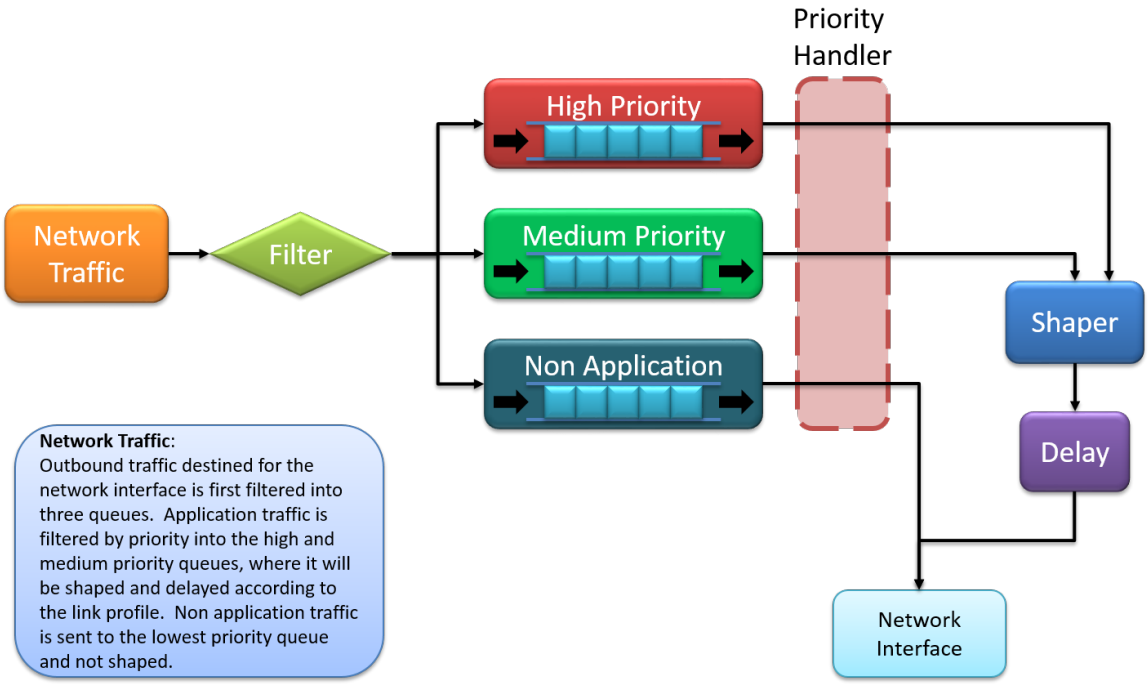


Figure 20: Diagram illustrating the flow of network traffic through the priority queues and traffic shaping in the kernel. The priority handler ensures that traffic in a lower priority queue is not serviced unless there is no traffic waiting in any higher priority queue.

This experimental setup allows us to examine the validity of our analysis techniques in the following ways:

- Because the system implements the strict priority queuing of flows, especially independent interacting flows, we can compare the delay and buffering measurements to the same delay and buffer predictions from the model.
- Router buffer space requirements can be measured and compared with their predicted requirements to validate routed network analysis.

- Link delay composition can be validated by examining the receiver buffer requirements compared with the predicted receiver buffer sizes.

The results of our experiments using this application and system configuration are shown in Table 3. Similarly to our earlier experimental results, the predictions of the overall delay for both the high and low priority routed flows are conservative, but tight bounds on the actual delay experienced by the flows in the routed network with delays. The delay measurements represent the time difference from when the sender application sends the message into the kernel for transmission, to when the receiver has received the message from its kernel. In between these two events, the message will have been queued and delayed by the sender's kernel, and transmitted to the router node. On the router node the message will have been queued and delayed (according to its priority) before being transmitted to the receiver node. On the receiver node the message will have been queued until the receiver is able to service the message according to its service profile.

Table 3: Delay and Buffer Results: Prediction versus Experiment for Routing Analysis.

	Predicted	Measured (μ, σ)
High Priority Flow Delay (s)	8.96	(4.1436 , 0.00929)
Low Priority Flow Delay (s)	15.7775	(13.0460 , 0.01344)

As can be seen in the table, the results for the delay analysis are conservative but not as tight as our previous results. These bounds are not as tight because TC does not perfectly constrain the flows to their allotted bandwidths, instead allowing bursts through when the link has capacity and data enters the shaper. These bursts have the effect of passing traffic through the network faster than the traffic should have gone through the network and in turn decreasing the latency of the traffic overall. Whereas our previous results used dummynet for traffic shaping on a single traffic shaping node, we needed to use a TC based approach to allow for the priority queuing of traffic and to better implement shaping of routed network

traffic. Despite these difference, the results still validate the compositional system analysis, the delay convolution, and the iterative analysis of routed networks.

CHAPTER IV

RUN-TIME NETWORK PERFORMANCE MONITORING AND MANAGEMENT FOR DISTRIBUTED CPS APPLICATIONS

4.1 Middleware-Integrated Measurement, Detection, and Enforcement

4.1.1 Problem

Many networking solutions, especially for large-scale systems, utilize a communications middleware of some sort, which allows the lower layer networking implementations to be abstracted into a uniform application programming interface. Furthermore, these middleware often support higher-level communications, resource management, and reliability configurations than the lower layers they are built on. However, these middlewares do not support the kind of time-varying resource constraints and provisioning which we have modeled and analyzed. Similarly, the lower layer resource allocation supports only static resource allocations, such as static bandwidth allocation for different flows traversing a network link.

As an example of such static resource allocation, consider two flows produced on the same node for the same network interface. These flows produce data with a rate that varies with respect to time, and both flows are high-priority flows. Since both flows are high-priority, they should be guaranteed the data rate they need for the link they share, but the link cannot support the combination of the flows' maximum data rates, even though their maximum data rates do not happen simultaneously.

With such resource allocation, it is difficult to guarantee these flows the capacities they require while ensuring that excessive data produced by one of the flows does not negatively impact the other flow.

4.1.2 Results

To address this static resource allocation problem, we have integrated our modeling semantics into middlewares to provide time-varying network resource allocation and capacity sharing.

Our run-time research and development of measurement, detection, and enforcement code for networked applications is built on the foundation of component-based software engineering (CBSE). The goal of CBSE is to provide a reusable framework for the development of application building-blocks, called *components* so that developers can develop and *analyze* applications in a more robust and scalable manner. In CBSE, a *component*, shown schematically in Figure 21, is the smallest deployable part of an application and is defined as a collection of timers, ports, and a executor thread:

$$C = \{\{T\}, \{P\}, H\} \tag{35}$$

Where

- $\{T\}$ is the *set* of all *timers* within the component. A timer provides a periodic event trigger to the component which triggers the callback associated with T, where a callback is a function defined and implemented by the developer.

- $\{P\}$ is the *set* of all *input/output ports* within the component. An i/o port provides a mechanism for message passing and event triggering between components, and may take the form of asynchronous *publish/subscribe* or synchronous *client/server* interaction patterns. Similarly to timers, each incoming event triggers the callback associated with P.

- H is the single thread which executes all event events for the component, in FIFO order, without preemption.

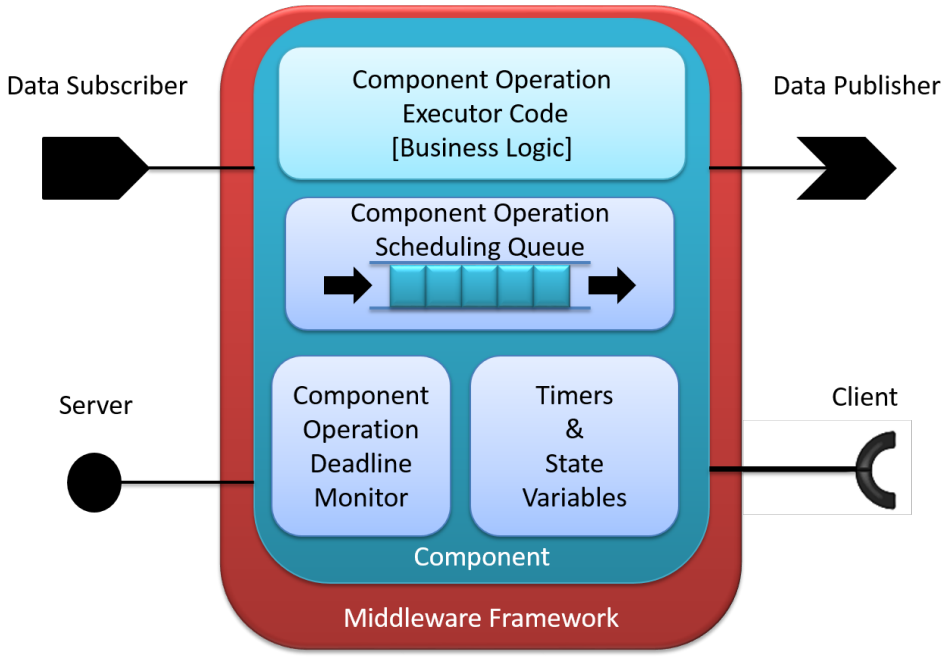


Figure 21: Schematic representation of a software component.

From this component definition, we can define an application as a grouping of components and a mapping between the ports of components:

$$A = \{\{C\}, \{M\}\} \tag{36}$$

Where

- $\{C\}$ is the *set* of components in the application
- $\{M\}$ is the *set* of *mappings* between ports of the components in $\{C\}$, for instance connecting a subscriber of C_x to a publisher of C_y, $M_{x,y} : C_x\{P_S\} \mapsto C_y\{P_P\}$.

And finally, an application's components are grouped into processes and distributed onto the nodes of a system through a deployment defined as a collection of nodes, processes, and a mapping from the nodes to the processes:

$$D = \{\{N\}, \{U\}, \{M\}\} \tag{37}$$

Where

- $\{N\}$ is the *set* of hardware *nodes* in the system
- $\{U\}$ is the *set* of *processes* defining the deployment, where a process is a collection of components $U = \{C\} \subseteq A\{\{C\}\}$.
- $\{M\}$ is the *set* of *mappings* between processes and nodes in the system, e.g. M_{U_1,N_1} : $U_1 \mapsto N_1$.

Note here that though the components are single threaded internally, the application containing these components may run them in parallel, e.g. by grouping them into a process or distributing them among the hardware nodes of the system. An example application and deployment onto a system of nodes is shown in Figure 22. Note that multiple applications (shades of blue in this figure) may be deployed simultaneously onto the same system and may even interact with each other. By using this component modeling framework and the associated code generation tools we have developed, the application developer needs only to provide the business-logic code for the application; the rest of the middleware and component configuration code is automatically provided by our library.

To facilitate experimentation and testing of our analysis techniques, we have developed network traffic production and consumption code which produces or consumes traffic on a communications link according to either a sender profile or receiver profile. These profiles are the same profiles used in the design-time analysis. We integrated this producer/consumer code into our component code-generators, which generate component skeleton code and communications middleware glue code based on our component model. Both sender and receiver automatically measure and record the network traffic for offline analysis.

Since the sender middleware code is automatically measuring and recording the output traffic from the application, we implemented additional code which can optionally push-back to the application by throwing and exception when the application is producing more data than specified in its profile. This push-back helps prevent a single application from

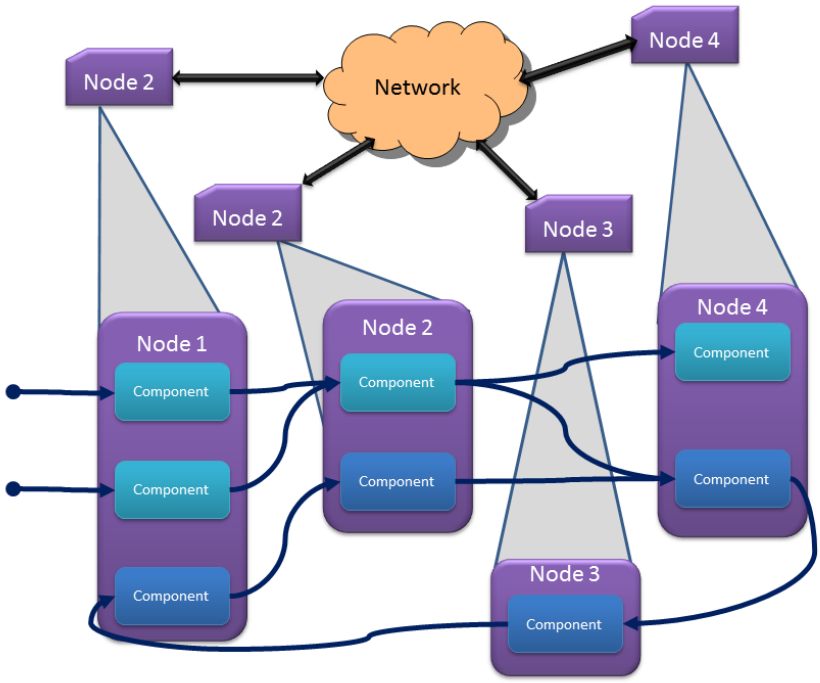


Figure 22: Two example distributed CBSE applications deployed on a system with 4 nodes.

producing more data than the system was designed for and flooding the network. In the case that such a push-back occurs, the application is notified and the data is not transmitted onto the network. Using this mechanism, a malicious or faulty application can be prevented from flooding the network and degrading the service of the network to other applications.

Similarly, since the receiver middleware code is automatically measuring and recording the input traffic from each of its senders, we implemented an additional communications channel which is used by the sender and receiver middleware and allows out-of-band communication which is invisible to the application. This out-of-band channel allows the sender to detect anomalies and inform the sender-side middleware of such anomalies. Further details about this capability and uses are explained in Section 4.2.

The development of this producer/consumer/measurement code not only helps with running experiments and data collection but also helps to ensure model to system consistency.

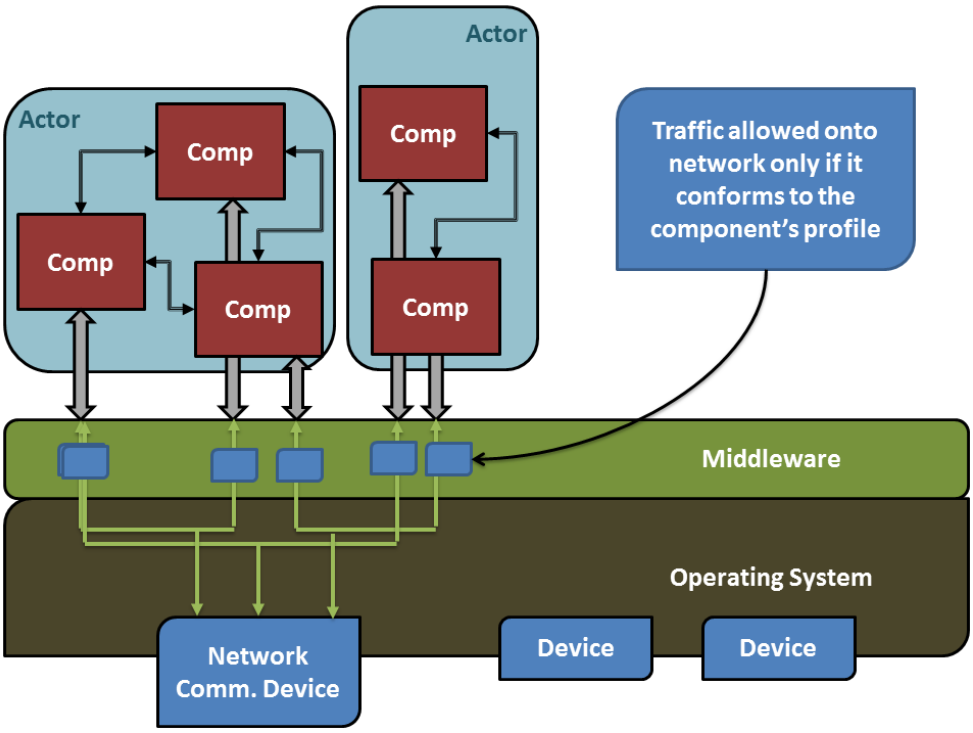


Figure 23: The structure of component-based applications and how their network traffic traverses the middleware and the OS stack.

We have implemented profile-based traffic production/consumption and traffic measurement into our code generators that we use with our model-driven design software. We developed this toolsuite to create distributed component-based software which uses the Robot Operating System (ROS)[17] as the communications middleware. ROS provides the capability for managing message passing, event triggering, and timer triggering that we need for our software components. For publish/subscribe interactions between components, into the publisher's generated code we add generic traffic producer code which publishes traffic according to the sender profile. Additionally, these publish operations are configured to use a small wrapper function which can measure the publish rate and can decide to throw a *profile exceeded* exception if the application attempts to send too much data or if the receiver has pushed back to the sender informing it to stop. The sender-side middleware layer is shown in Figure 23.

This push back from the receiver occurs through the use of an out-of-band (OOB) channel using UDP multicast, which receivers use to inform specific senders that they are sending too much data to the receivers (and possibly overflowing the receiver buffers). This OOB channel provides a mechanism by which the secure middleware layer can protect the system from malicious or faulty applications.

Into the receiver code (for subscribers) we additionally generate a receive buffer and receiver thread which pulls data from the buffer according to the receiver profile. In this scenario, the receiver has a capacity with which it can handle incoming data, and it has a finite buffer so it must use the OOB channel and measurements on the incoming data stream to determine which senders to shut down to ensure its buffer does not overflow. When the buffer has had some time empty (so that it's not in danger of running out of buffer space), the receiver can use the OOB channel to inform the halted senders that it is alright to send again. The complete description of the OOB channel, and the way the receiver limits the senders can be found in Section 4.2. An example of our traffic producer's accuracy is shown in Figure 24. For the data in this figure, each message was recorded as a tuple of $timestamp, messagesize$, where the timestamp is the time at which the message was either sent by the application.

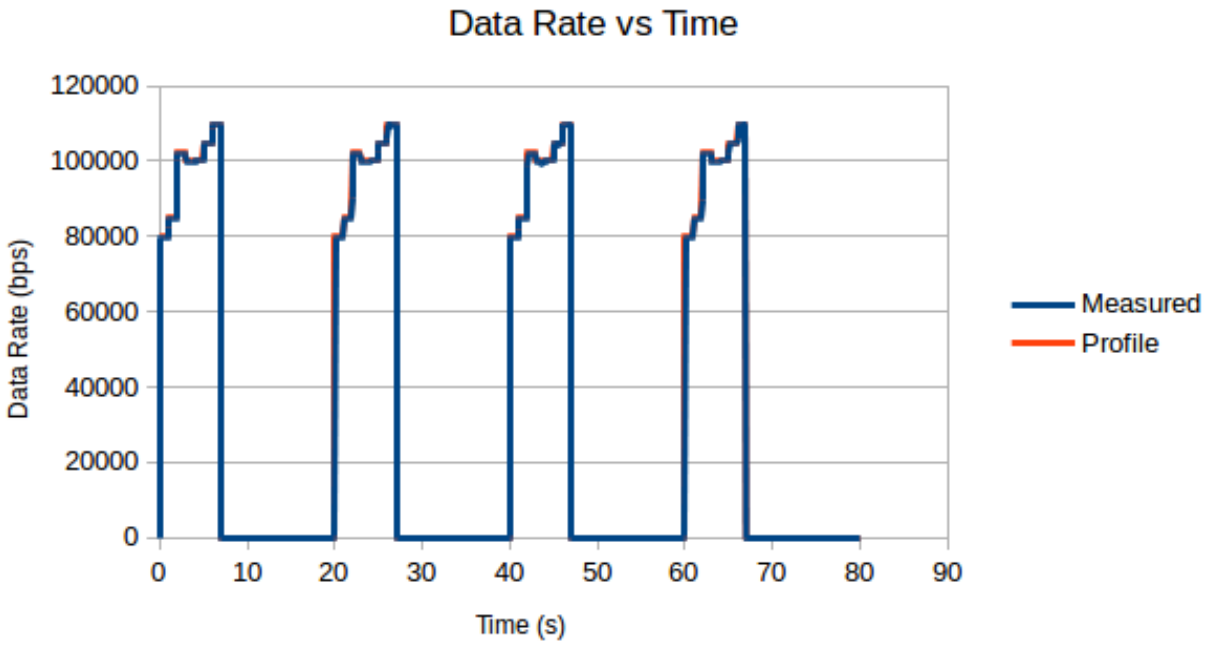


Figure 24: Demonstration of the accuracy with which our traffic producers follow the specified profile.

4.2 Distributed Denial of Service (DDoS) Detection

4.2.1 Problem

For distributed systems which must ensure resource availability and system stability, a key aspect of the infrastructure is detection and mitigation of faults or anomalies. With respect to network resources, an example is checking source and destination for communications to enforce only authorized communication flows are present in the system. However, software glitches or compromised applications can exceed system resources that they have been allocated. As described in the previous section, higher-fidelity resource modeling and monitoring is required to prevent such faults or compromises from propagating throughout the system. However, mitigating the propagation only solves part of the problem; ideally the system should classify the type of fault or anomaly and begin diagnostics to trace the fault/anomaly back to its origin. In this thesis we will address only the problem of detecting and mitigating Distributed Denial of Service attacks.

Denial-of-Service (DoS)[36] and Distributed DoS (DDoS) attacks can take many forms, but are generally classified as excessive traffic from a large amount of (possibly heterogeneous) sources targeted towards a single point or a single group. Such attacks are common to machines on the internet, but can also become a hazard for machines on private networks which become infected or inadvertently expose an input path for external malicious data.

These private or semi-private systems must have mechanisms for detecting and mitigating such attacks, and the combination of our design-time analysis and run-time measurement, detection, and mitigation tools provides a form for such capability. The goal of this work is for a receiver, which is being targeted for attack by a set of senders, to determine which of the senders are behaving anomalously and prevent them from sending any more data. In this way, a group of senders performing a DDoS attack can be mitigated by the targeted receiver. Towards this goal we make the following changes outlined below to our modeling/analysis framework and implementation.

4.2.2 Results

Because these types of attacks come from systems for which the application profiles may not be completely or fully known, we must alter our modeling semantics such that we can model these types of applications and the uncertainty surrounding their data rates.

If we relax the constraint on the modeling semantics that all sender profiles are absolute and the system behavior is completely known at design-time, then we not only expand the scope of applications that can be supported but also enable meaningful anomaly detection.

Whereas previously, profiles modeled the exact *data rate* as a function of time that the application produced, we now alter the definition to capture two parameters: *mean data rate* (μ) and *max data rate* (*MDR*), which again are both functions of time. Just as before, these functions are constant-valued between successive values of t and are time-integrated to produce the *mean data* and *max data* cumulative profiles as functions of time. With this specification, we no longer know exactly how much data an application will produce at a

given point in time, but instead are provided two values by the developer: the *mean* and *max*.

Now that we have these two profiles for the application, we could simply analyze the *max* data profile to determine buffer and latency requirements, but this would end up wasting resources by allocating memory and network resources of the system to the application even if is not producing data at its *max rate*. Instead, we analyze the system according to the *mean* data profile to determine buffer requirements and latency for the application in the system. In doing so, two buffer overflow risks are possible: 1) Sender-side buffer overflow, and 2) Receiver-side buffer overflow.

We make the assumption that the application meters its sending to prevent the first scenario, so that its data is not lost before making it onto the network. In this case, the sender can still send data at a rate greater than the *mean*, but that rate is partially governed by the capacity given to it by the node's network.

For the second case, we must ensure that there is no buffer overflow on the receiver-side. To enable this functionality, we must provide a mechanism for the receiver to communicate with the sender. This push-back communication should travel through a channel outside the communications channel that the application has access to, so that the application, either maliciously or inadvertently, cannot disrupt this push-back and in turn cause the receiver's buffer to overflow. For this reason, we add into the sender and receiver middleware an out-of-band (OOB) channel that provides a communications layer between all senders and receivers that is invisible to the application. For our component model and communications middleware, we have implemented this OOB channel as a UDP multicast group.

Because the goal of this work is to meter only the senders which are producing *too much* data, we must define what too much data is. Because we have developed these application profiles for analysis, and these profiles describe the mean data rate, μ, and the max data rate, *MDR*, of the senders, they will be used to determine when a sender is sending too much data. In this paradigm, sender S_i is determined as behaving anomalously (i.e. sending

too much data) if it is sending data at a rate $DR_i > \mu_i$. The assumption implicit in this comparison is that the receiver, to be able to make this comparison, has full knowledge of μ_i, since DR_i is calculable on the receiver side. If the receiver's buffer is filling up, it looks through the the measured DR_i (within a certain window of time) for each of the senders it has been receiving data from, and compares it against the sender's μ_i. If the comparison is *true*, it uses the OOB channel to push back to that specific sender, informing the sender-side middleware to stop transmitting data until the receiver has re-enabled that sender's transmission. When the receiver has emptied it's buffer enough it can then use the OOB channel to re-enable any disabled senders. The algorithm used by the receiver to determine which senders to limit is shown in Listing 2, and has been integrated into our middleware.

```
receiver::limit_ddos( t_start, t_end )
{
  for sender in senders
  {
    d_start = sender.received_data(t_start)
    d_end = sender.received_data(t_end)
    profile_d_start =
      sender.profile(t_start)
    profile_d_end =
      sender.profile(t_end)
    allowed_data = profile_d_end - profile_d_start
    actual_data = d_end - d_start
    if actual_data > allowed_data
    {
      sender.disable()
    }
  }
}
```

Listing 2: Algorithm used by receivers to determine which senders to limit. The receiver only looks at the behavior of senders within the time window between t_{start} and t_{end}, which is configurable.

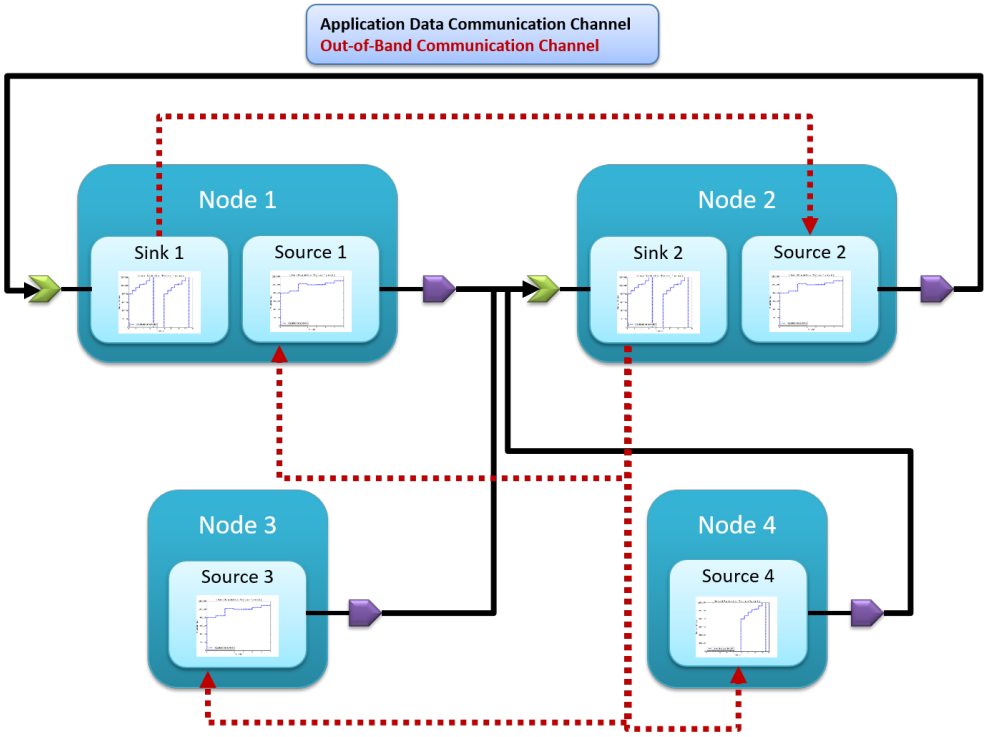


Figure 25: Nodes in an example network and how they communicate (using pub/-sub).

We have shown experimentally (for the system in Figure 25) that this measurement, detection, and mitigation strategy can protect non-malicious clients from being denied service to a server by malicious or faulty clients. In this example, the client's data might be lost if the server's buffer overflows. Using the algorithm and implementation described above, we show that the server is able to keep its buffer from overflowing despite two of the three publishers producing too much data. In this scenario, the attackers would cause dropped packets by producing data at a rate just less than their *MDR*, which would require a server side buffer of 459424 bits. However, the server detects that its buffer might overflow and that the two malicious clients are producing too much data and it signals for their middleware to prevent them from sending data. In this way, it maintains available buffer space (maximum buffer of 393792 bits) to handle the good client's data and the good client is not denied service.

CHAPTER V

CONCLUSIONS AND FUTURE WORK

We have described in this thesis the aspects of Cyber-Physical Systems(CPS) analysis, design, development, and integration we are addressing. We have provided descriptions of the relevant related work in this area, covering both the design-time modeling, analysis, and performance prediction for networked, distributed, CPS applications and the run-time monitoring and management of application and CPS network resources.

Subsequently, the completed research towards precise network performance prediction, PNP^2, was presented.

First, the formalization for the modeling and analysis semantics and techniques were defined, building off of the $(\wedge, +) - calculus$ used by Network Calculus. Models of systems and applications were presented and convolution ($\otimes$) of application profiles with system profiles was defined. Using $(\wedge, +) - calculus$, the computation of delay and backlog bounds were defined.

Given the definition of the fundamental operations of PNP^2, analysis of periodic systems was presented. We described how periodic data rate profiles can be time-integrated to produce repeating data profiles as functions of time. We proved that the minimum amount of time for which the system and its applications must be analyzed to determine if there is unbounded buffer growth is two hyperperiods.

Using experimental system data, we determined the benefit of PNP^2 versus similar techniques such as Network Calculus. We showed how our techniques provide more accurate predictions with respect to the actual system but are still conservative predictions.

We then showed how a model of MAC protocols, such as TDMA, could be incorporated into the system and application models. Using these models, we analytically derived equations for the effects of such protocols on the predicted delay and backlog bounds.

The mathematical operations of PNP^2 were extended to support compositional system analysis by defining the concepts of profile addition and subtraction. For this compositional analysis, the concept of profile priority was introduced to determine service precedence by the transmitting node between two profiles.

Since latency is such a critical aspect of networking systems, we introduced semantics for modeling the delay of network links as a linear, continuous function of time. Convolution of a profile with a delay profile was introduced and its effects on the profile's periodicity were analyzed.

To support more complex systems which include nodes that can act as routers and forwarders for traffic from other nodes, we presented an algorithm that uses the concepts we developed for delay analysis and compositional analysis to iteratively analyze a system which contains statically routed traffic. Experimental validation of this integrated, system-level analysis was provided to demonstrate the accuracy and precision of the analysis techniques.

To support experimental validation and run-time testing, we developed code generators that generate traffic producer/consumer and measurement code into the component models we defined. Using these producers and consumers, which operate based on the same profiles used for design-time analysis, we ran experiments which corroborated our analysis results.

Finally, we extended our traffic producer/consumer code to enable management of the network traffic by the communications middleware. Detection code was developed for the receivers to detect when and which senders were overflowing the receiver's buffer and use an out-of-band communications channel to inform the sender's middleware to limit the sender's data production.

5.1 Future Work

In this work we have described the beginning of precise, comprehensive network system performance analysis and prediction. However, we could not possibly cover the modeling and analysis of all possible system configurations, communication protocols, or interaction paradigms. Furthermore, we have examined the affect certain system configuration parameters or modeling choices have on our analysis techniques and results, but such examination is not exhaustive.

Extending this work would focus on these areas in the following ways:

- Modeling and analysis support for more (commonly used) transmission protocols, such as TCP or SCTP.
- Developing models of other MAC protocols such as CSMA
- Deriving models of packet loss or transmission error and analyzing their effects on the prediction results
- Analyzing the effects of uncertainty in the modeling of applications and systems
- Research into methods for including models of data-dependent network traffic and analyzing such applications
- Analyzing the affects of timing and time synchronization inaccuracies on the prediction results
- Investigating run-time implementation alternatives and data analysis techniques

To support modeling and analysis of protocols such as TCP, which are reactive to return-path information, return-path modeling semantics and analysis techniques would be needed. Return-path modeling can be challenging because in the non-trivial case, return-path information is used to make the application or system protocol reactive to the current

state of the system. This is the case for instance in TCP where the timing of the return packets or the lack or return packets alters the outgoing data flow. Similarly, lower-level protocol implementation details like connection management and handshaking can be affected by variable network capacity, therefore they can indirectly affect application performance.

This type of return-path modeling and feedback system needed for modeling such protocols would also benefit the analysis of data-dependent application profiles, since they similarly are dependent on some external input which at least partially governs the characteristics of the traffic they produce.

Similarly, forwarding protocols in the lower layers, such as the store and forward mechanisms used by certain routers, are also reactive to data driven events, such as packet loss or packet corruption. The modeling and analysis extensions described above would pave the way for the analysis of the effects on application performance caused by these lower level protocol mechanisms.

As an extension to the application and system level models, which currently are precisely defined and assumed to be exactly known, research into uncertainty analysis of these profiles would allow performance prediction for systems that do not meet these assumptions about full knowledge. If instead of exact knowledge about the system and application profiles, the application developers and system integrators have uncertainties associated with their models, then analysis of the uncertainty and its effect on the predictions would expand the scope of systems to which the techniques could be applied.

In a similar way to the modeling uncertainty analysis, timing uncertainty analysis could be performed to determine the affects of incomplete time synchronization between the nodes of the system. Such analysis would allow for the relaxation of the system-wide time-synchronization constraint. If that constraint is relaxed such that the nodes are known be re-synchronized periodically with some predictable drift, then such behavior can be directly analyzed similarly to the TDMA analysis. From this information, maximum deviations on

the required buffer and delay can be calculated, similarly to the deviations calculated for TDMA systems.

Finally, the analysis of application traffic profiles provides a possible avenue for precise categorization of application behavior. Given an application which is supposed to produce traffic with a certain profile, middleware-based measurements of the actual traffic profile produced by the application can allow the middleware or other management entity to classify the behavior of the application. Analyzing this behavior and comparing it with the behavior of other applications in the system would allow for better detection of faults, malicious behavior, or other anomalies.

Appendices

APPENDIX A

PUBLICATIONS

1.1 Highly Selective Conference Papers

- W. Emfinger and G. Karsai. Modeling network medium access protocols for network quality of service analysis. In *Proceedings of the 18th IEEE Symposium on Real-Time Computing (ISORC)*, ISORC '15, pages 44–47, Auckland, New Zealand, 2015. IEEE

1.2 Other Conference and Workshop Papers

- W. Emfinger, G. Karsai, A. Dubey, and A. Gokhale. Analysis, verification, and management toolsuite for cyber-physical applications on time-varying networks. In *Proceedings of the 4th ACM SIGBED International Workshop on Design, Modeling, and Evaluation of Cyber-Physical Systems*, CyPhy '14, pages 44–47, New York, NY, USA, 2014. ACM

- W. Emfinger, P. Kumar, A. Dubey, W. Otte, A. Gokhale, G. Karsai. DREMS: A Toolchain for the Rapid Application Development, Integration, and Deployment of Managed Distributed Real-time Embedded Systems. In *Proceedings of the IEEE Real-Time Systems Symposium*, RTSS@Work 2013, Vancouver, Canada, 2013. IEEE

- P. S. Kumar, W. Emfinger, A. Kulkarni, G. Karsai, D. Watkins, B. Gasser, C. Ridgewell, and A. Anilkumar. Rosmod: A toolsuite for modeling, generating, deploying, and managing distributed real-time component-based software using ros. In *Rapid System Prototyping Symposium (ESWEEK)*, Amsterdam, The Netherlands, 2015

- P. S. Kumar, W. Emfinger, and G. Karsai. A testbed to simulate and analyze resilient cyber-physical systems. In *Rapid System Prototyping Symposium (ESWEEK)*, 2015

- Pradhan S., W. Emfinger, A. Dubey, W. Otte, A. Coglio, D. Balasubramanian, A. Gokhale, G. Karsai. Establishing secure interactions across distributed applications in satellite clusters. In *Proceedings of the IEEE International Conference on Space Mission Challenges for Information Technology*, SMC-IT, 2014, Laurel, MD, USA. IEEE

- Balasubramanian, D., W. Emfinger, P. S. Kumar, W. Otte, A. Dubey, and G. Karsai. An application development and deployment platform for satellite clusters. In *Proceedings of the Workshop on Spacecraft Flight Software*, 2013

- Balasubramanian, D., A. Dubey, W. R. Otte, W. Emfinger, P. Kumar, and G. Karsai. A Rapid Testing Framework for a Mobile Cloud Infrastructure. In *Proceedings of the IEEE International Symposium on Rapid System Prototyping*, RSP, 2014. IEEE

- Dubey, A., W. Emfinger, A. Gokhale, G. Karsai, W. R. Otte, J. Parsons, C. Szabo, A. Coglio, E. Smith, and P. Bose. A Software Platform for Fractionated Spacecraft. In *Proceedings of the 2012 IEEE Aerospace Conference*, Big Sky, Montana, 03/2012. IEEE

- Levendovszky, T., A. Dubey, W. R. Otte, D. Balasubramanian, A. Coglio, S. Nyako, W. Emfinger, P. Kumar, A. Gokhale, and G. Karsai. DREMS: A Model-Driven Distributed Secure Information Architecture Platform for Managed Embedded Systems. In *IEEE Software*, vol. 99: IEEE Computer Society, 2014. IEEE

1.3 Submitted Papers - Awaiting Reviews

- W. Emfinger, P. Kumar, A. Dubey, G. Karsai. Towards Assurances in Self-Adaptive, Dynamic, Distributed Real-time Embedded Systems. In *Software Engineering for Self-Adaptive Systems: Assurances*, 2015.

APPENDIX B

CONFIGURATION OF LINUX TC

This chapter covers the specifics of how the routing, queuing, and shaping of network traffic is configured and how this traffic passes through the queues and shapers in the Linux Kernel before being transmitted through the network interface.

We configure the system's static routes using the Linux's built-in IPRoute[24] tool, which allows for the configuration of the kernel's routing tables, network address translation (NAT), and characteristics of the network interface such as the maximum transmit unit (MTU). For each node, a route is added to the routing table specifying the next hop address as its gateway for each other node to which it is not directly connected. In this way, the packets in the network will be routed using loose source-based routing where the sender node does not know the full route that the packet will take, but just forwards it to the next known location.

The system's priority-based network traffic queuing, network delay, and network bandwidth capacity was configured using Linux's built-in Traffic Control (TC)[32] tool (which is part of IPRoute), that allows for the configuration of hierarchical class-based traffic scheduling, and traffic shaping. We configured the output interfaces on each node as a combination of two queuing disciplines (qdiscs): 1) a network emulator (netem) which enforces the link delay, and 2) a token-bucket filter (TBF) which enforces the rate control to enforce the link's bandwidth profile. On the routing node, we added an additional priority qdisc (prio), with multiple priority queues. We configured TC to filter packets into these queues by matching packet source IP address and destination IP address. This filtering ensured that the high priority flow's packets would be filtered into the high-priority queue of the prio qdisc, while the lower priority flow's packets would be filtered into the lower priority queue of the prio qdisc. All traffic out of the priority qdisc was fed into the TBF

to ensure that all traffic, regardless of priority, shared and was shaped by the node's link capacity profile. The configuration and function of TC is shown in Figure 26.

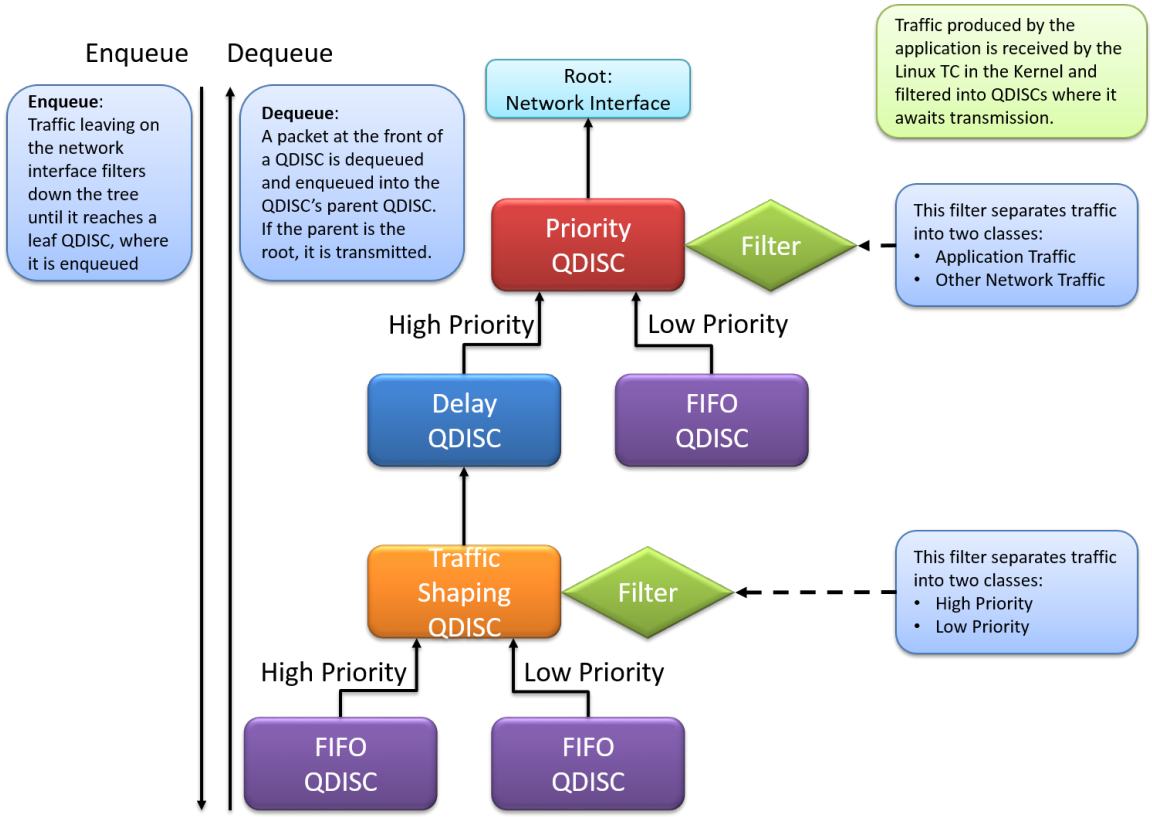


Figure 26: Diagram illustrating the TC configuration used to implement priority flows, traffic shaping, and delay.

REFERENCES

[1] R. Bate, D. Mueller, and J. White. *Fundamentals of Astrodynamics*. Dover Books on Aeronautical Engineering Series. Dover Publications, 1971.

[2] T. Bingmann. *Accuracy Enhancements of the 802.11 Model and EDCA QoS Extensions in ns-3*. PhD thesis, University of Karlsruhe, 2009.

[3] S. Blake, S. Microsystems, and Z. Wang. An Architecture for Differentiated Services. *RFC*, pages 1–37, 1998.

[4] A. Bouillard, N. Farhi, and B. Gaujal. Packetization and packet curves in network calculus. In *Proc. 6th Int Performance Evaluation Methodologies and Tools (VALUETOOLS) Conf*, pages 136–137, 2012.

[5] R. Braden, D. Clark, and S. . Shenker. Integrated Services in the Internet Architecture : an Overview. *IETF RFC 1633, July*, pages 1–28, 1994.

[6] A. Burchard, J. Liebeherr, and S. Patek. A min-plus calculus for end-to-end statistical service guarantees. *IEEE TRANSACTION ON INFORMATION THEORY*, 52:4105–4114, 2006.

[7] M. Carbone and L. Rizzo. Dummynet revisited, 2010.

[8] O. S. Consortium et al. Openflow switch specification version 1.0. 0, 2009.

[9] R. Cruz. A calculus for network delay. II. Network analysis. *IEEE Transactions on Information Theory*, 37(1):132–141, 1991.

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

[11] B. S. Davie, A. Charny, J. W. R., K. B. Benson, J. Y. Le, W. Courtney, S. Davari, and V. Firoiu. An expedited forwarding PHB (Per-Hop behavior). RFC 3246, Internet Engineering Task Force, Mar. 2002.

[12] S. Dobson. An adaptive systems perspective on network calculus, with applications to autonomic control. *International Journal of Autonomous and Adaptive Communications Systems*, 1(3):332, 2008.

[13] W. Emfinger and G. Karsai. Modeling network medium access protocols for network quality of service analysis. In *Proceedings of the 18th IEEE Symposium on Real-Time Computing (ISORC)*, ISORC '15, pages 44–47, Auckland, New Zealand, 2015. IEEE.

[14] W. Emfinger, G. Karsai, A. Dubey, and A. Gokhale. Analysis, verification, and management toolsuite for cyber-physical applications on time-varying networks. In

Proceedings of the 4th ACM SIGBED International Workshop on Design, Modeling, and Evaluation of Cyber-Physical Systems, CyPhy '14, pages 44–47, New York, NY, USA, 2014. ACM.

[15] T. Ferrandiz, F. Frances, and C. Fraboul. A Network Calculus model for SpaceWire networks. In *Proceedings - 17th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, RTCSA 2011*, volume 1, pages 295–299, 2011.

[16] I. Foster, A. Roy, and V. Sander. A quality of service architecture that combines resource reservation and application adaptation. In *Quality of Service, 2000. IWQOS. 2000 Eighth International Workshop on*, pages 181–188. IEEE, 2000.

[17] W. Garage. Robot operating system (ros). http://www.ros.org.

[18] G. R. Garay, J. Ortega, and V. Alarcon-Aquino. Comparing Real-Time Calculus with the existing analytical approaches for the performance evaluation of network interfaces. In *CONIELECOMP 2011 - 21st International Conference on Electronics Communications and Computers, Proceedings*, pages 119–124, 2011.

[19] G. Giambene. *Queuing Theory and Telecommunications Networks and Applications*. 2005.

[20] Gilberto Flores Lucio, Marcos Paredes-farrera, Emmanuel Jammeh, Martin Fleury, and M. J. Reed. OPNET Modeler and Ns-2: Comparing the Accuracy of Network Simulators for Packet-Level Analysis using a Network Testbed. *3rd WEAS International Conference on Simulation, Modelling and Optimization (ICOSMO)*, pages 700–707, 2003.

[21] G. GÃűssler and J. Sifakis. Composition for component-based modeling. *Science of Computer Programming*, 55(1âĂŞ3):161 – 183, 2005. Formal Methods for Components and Objects: Pragmatic aspects and applications.

[22] J. Heidemann, N. Bulusu, J. Elson, C. Intanagonwiwat, K.-c. Lan, Y. Xu, W. Ye, D. Estrin, and R. Govindan. Effects of Detail in Wireless Network Simulation. In *Proceedings of the{SCS} Conference on Communication Networks and Distributed Systems Modelling and Simulation*, pages 3–11, 2001.

[23] J. Hoffert, D. Schmidt, and A. Gokhale. Adapting and evaluating distributed real-time and embedded systems in dynamic environments. In *Proceedings of the First International Workshop on Data Dissemination for Large Scale Complex Critical Infrastructures*, DD4LCCI '10, pages 23–28, New York, NY, USA, 2010. ACM.

[24] B. Hubert et al. Linux advanced routing & traffic control howto. *setembro de*, 2002.

[25] P. Jung, P. W. Baier, and A. Steil. Advantages of cdma and spread spectrum techniques

over fdma and tdma in cellular mobile radio applications. *Vehicular Technology, IEEE Transactions on*, 42(3):357–364, 1993.

[26] S. Katti, H. Rahul, W. Hu, D. Katabi, M. Medard, and J. Crowcroft. XORs in the air: Practical wireless network coding. *IEEE/ACM Transactions on Networking*, 16(3):497–510, 2008.

[27] D. G. Kendall. Stochastic Processes Occurring in the Theory of Queues and their Analysis by the Method of the Imbedded Markov Chain. *The Annals of Mathematical Statistics*, 24(3):338–354, 1953.

[28] J. Kephart and D. Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, Jan 2003.

[29] K. Khan and H. Peyravi. Delay and queue size analysis of tdma with general traffic. In *Modeling, Analysis and Simulation of Computer and Telecommunication Systems, 1998. Proceedings. Sixth International Symposium on*, pages 217–225, Jul 1998.

[30] P. S. Kumar, W. Emfinger, and G. Karsai. A testbed to simulate and analyze resilient cyber-physical systems. In *Rapid System Prototyping Symposium (ESWEEK)*, 2015.

[31] P. S. Kumar, W. Emfinger, A. Kulkarni, G. Karsai, D. Watkins, B. Gasser, C. Ridgewell, and A. Anilkumar. Rosmod: A toolsuite for modeling, generating, deploying, and managing distributed real-time component-based software using ros. In *Rapid System Prototyping Symposium (ESWEEK)*, Amsterdam, The Netherlands, 2015.

[32] A. Kuznetsov. Linux traffic control (tc). http://www.sparre.dk/pub/linux/tc.

[33] J.-Y. Le Boudec and P. Thiran. *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet*. Springer-Verlag, Berlin, Heidelberg, 2001.

[34] H. Li, X. Liu, W. He, J. Li, and W. Dou. End-to-End Delay Analysis in Wireless Network Coding: A Network Calculus-Based Approach. *2011 31st International Conference on Distributed Computing Systems*, pages 47–56, 2011.

[35] J. D. C. Little. A proof for the queuing formula: L = Îżw. *Operations Research*, 9(3):383–387, 1961.

[36] E. M. Handley, UCL, E. E. Rescorla, N. Resonance, and I. A. Board. Internet denial-of-service considerations. Request for Comments 4732, Internet Engineering Task Force, Nov. 2006.

[37] I. Mahadevan and K. Sivalingam. Quality of Service architectures for wireless networks: IntServ and DiffServ models. *Proceedings Fourth International Symposium on Parallel Architectures, Algorithms, and Networks (I-SPAN'99)*, 1999.

[38] D. Mahrenholz and S. Ivanov. Real-time network emulation with ns-2. *Proceedings - Eighth IEEE International Symposium on Distributed Simulation and Real-Time Applications, DS-RT 2004*, pages 29–36, 2004.

[39] C. Mathieu and A. Weigel. Assessing the fractionated spacecraft concept. *AIAA Paper*, 7212, 2006.

[40] K. Nichols, S. Blake, F. Baker, and D. L. Black. Definition of the differentiated services field (DS field) in the IPv4 and IPv6 headers. RFC 2474, Internet Engineering Task Force, Dec. 1998.

[41] Object Computing Incorporated. OpenDDS. http://www.opendds.org, 2007.

[42] Object Management Group. *Data Distribution Service for Real-time Systems Specification*, 1.2 edition, Jan. 2007.

[43] S. Papanastasiou, J. Mittag, E. Strom, and H. Hartenstein. Bridging the gap between physical layer emulation and network simulation. In *Wireless Communications and Networking Conference (WCNC), 2010 IEEE*, pages 1–6, April 2010.

[44] G. Riley and T. Henderson. The ns-3 network simulator. In K. Wehrle, M. GÃijneÅ§, and J. Gross, editors, *Modeling and Tools for Network Simulation*, pages 15–34. Springer Berlin Heidelberg, 2010.

[45] L. Rizzo. Dummynet : a simple approach to the evaluation of network protocols. *ACM SIGCOMM Computer Communication Review*, 27(1):31–41, 1997.

[46] J. B. Schmitt, F. A. Zdarsky, and L. Thiele. A comprehensive worst-case calculus for wireless sensor networks with in-network processing. In *Proceedings - Real-Time Systems Symposium*, pages 193–202, 2007.

[47] T. J. Schriber, D. T. Brunner, and J. S. Smith. Inside discrete-event simulation software: How it works and why it matters. In *Proceedings of the 2013 Winter Simulation Conference - Simulation: Making Decisions in a Complex World, WSC 2013*, pages 424–438, 2013.

[48] L. Shengchang, L. Jionghui, T. Chao, C. Qing, and Z. Jinxiu. Energy efficient network strategy for nanosatellites cluster flight formations.

[49] L. Thiele, S. Chakraborty, and M. Naedele. Real-time calculus for scheduling hard real-time systems. In *in ISCAS*, pages 101–104, 2000.

[50] A. Vagas. Inetmanet framework, 2010.

[51] A. Varga and R. Hornig. An overview of the omnet++ simulation environment. In *Simutools '08: Proceedings of the 1st international conference on Simulation tools*

and techniques for communications, networks and systems & workshops, pages 1–10, ICST, Brussels, Belgium, Belgium, 2008. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[52] Y. Wang and T. Wang. Applying stochastic network calculus to 802.11 backlog and delay analysis. In *IEEE International Workshop on Quality of Service, IWQoS*, 2011.

[53] H. Xiao, W. K. Seah, A. Lo, and K. C. Chua. A flexible quality of service model for mobile ad-hoc networks. In *Vehicular Technology Conference Proceedings, 2000. VTC 2000-Spring Tokyo. 2000 IEEE 51st*, volume 1, pages 445–449. IEEE, 2000.