Research Progress Report

William Emfinger

September 09, 2015

This documentation covers the background, theory, and results for my research, *Precise Network Performance Prediction*, (PNP^2), the aim of which is to provide design-time analysis of distributed cyber-physical systems' (CPS) and their applications' network performance. The type of network performance we will focus on predicting is the application buffer size requirements and the application network traffic buffering delay. These factors were chosen because they relate directly to how well the application traffic is serviced by the system and they directly affect resource utilization of the application on the system.

Because the buffering resources available to applications may be fixed by the system, application developers need to know at design time whether or not the application's data production versus data consumption will overflow the application's allotted buffer space. By analyzing these systems, developers can assess these conditions at design-time and try to mitigate any predicted overflows before deploying the applications.

Furthermore, buffering delay/latency is an important metric for determining application network performance because in many of these CPS, interactions and computation have deadlines that must be met, for instance in an attitude control system where the physical dynamics of the system determine the required reaction time of the software from sensing to calculation to actuation. Developers need guarantees that such reaction times can be met, and for distributed systems, the buffering latency and transmission latency for network data affects these reaction times.

Such design-time performance analysis and feedback is critical to the development of robust safety- or mission-critical applications.

CHAPTER

ONE

BACKGROUND: NETWORK PERFORMANCE ANALYSIS

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.

We focus on the second paradigm, using models for predicting network performance at design-time. This focus comes mainly from the types of systems to which we wish to apply our analysis: safety- or mission-critical distributed cyber-physical systems, such as satellites, or autonomous vehicles. For such systems, resources come at a premium and design-time analysis must provide strict guarantees about run-time performance and safety before the system is ever deployed.

For such systems, probabilistic approaches do not provide high enough confidence on performance predictions since they are based on statistical models [*Cruz1991*]. Therefore, we must use deterministic analysis techniques to analyze these systems.

1.1 Min-Plus Calculus

Because our work and other work in the field, e.g. Network Calculus, is based on Min-Plus Calculus, or (min,+)-calculus, we will give a brief overview of it here, adapted from [*Thiran2001*].

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

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

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

$$\forall s, t : f(s+t) \leq f(s) + f(t)$$

Note that if a function is concave with $f(0) = 0$, it is sub-additive, e.g. $y = \sqrt{x}$.

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:

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

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

$$(f \otimes g)(t) = min(f, g)$$

Convolution in min-plus calculus has the properties of

- closure: $(f \otimes g)(t) \in F$,
- Associativity,
- Commutativity, and
- Distributivity

1.2 Network Calculus

Network Calculus [*Cruz1991*, *Cruz1991a*, *Thiran2001*] provides a modeling and analysis paradigm for deterministically analyzing networking and queueing systems. Its roots come from the desire to analyze network and queuing systems using similar techniques as traditiional electrical circuit systems, i.e. by analyzing the *convolution* of an *input* function with a *system* function to produce an *output* function. Instead of the convolution mathematics from traditional systems theory, Network Calculus is based on the concepts of *(min,+)-calculus*, which we will not cover here for clarity, but for which an explanation can be found in my proposal and thesis.

By using the concepts of *(min,+)-calculus*, Network Calculus provides a way to model the application network requirements and system network capacity as functions, not of time, but of *time-window size*. Such application network requirements become a cumulative curve defined as the *maximum arrival curve*. This curve represents the cumulative amount of data that can be transmitted as a function of time-window size. Similarly, the system network capacity becomes a cumulative curve defined as the *minimum service curve*. These curves bound the application requirements and system service capacity.

Note that 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.

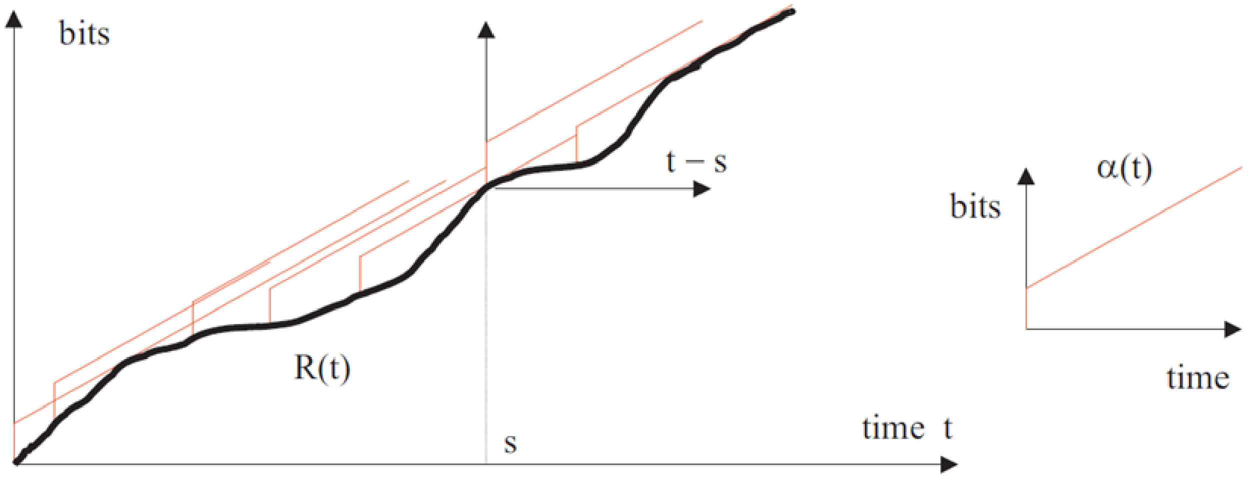


Fig. 1.1: Network Calculus arrival curve (α). Reprinted from [*Thiran2001*].

Network calculus uses *(min,+)-calculus convolution* to compose the application requirement curve with the system service curve. The output of this convolution is the maximum data arrival curve for the output flow from the node providing the service. By analyzing these curves, bounds on the application's required buffer size and buffering delay can be determined.

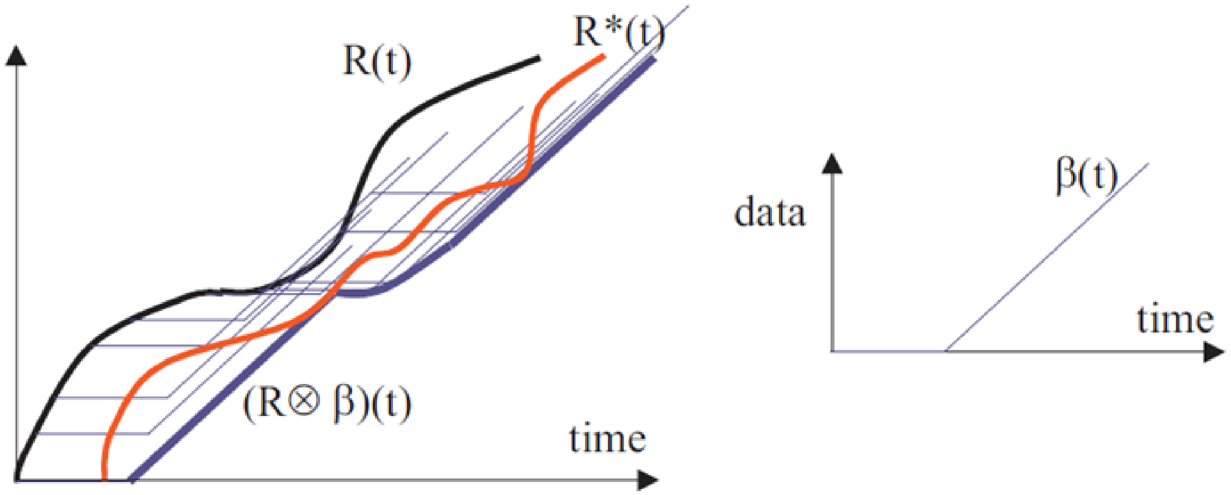


Fig. 1.2: Network Calculus service curve (β). Reprinted from [*Thiran2001*].

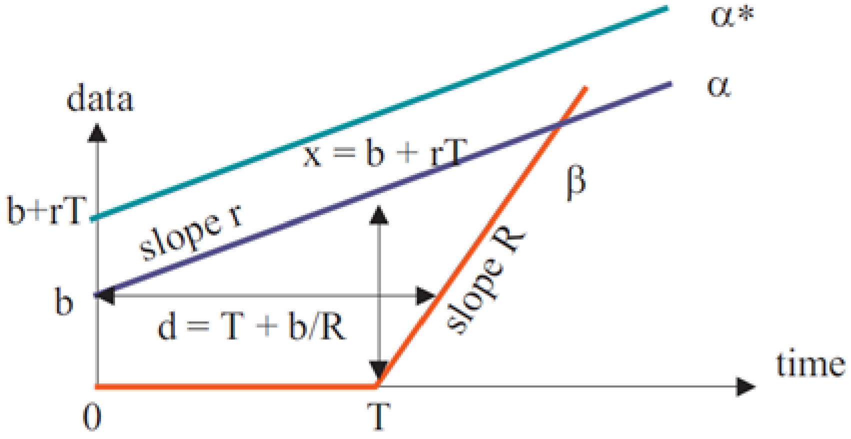


Fig. 1.3: Schematic depticion of the buffer size (vertical difference) and delay (horizontal difference) calculations in Network Calculus. Reprinted from [*Thiran2001*].

With these bounds and the convolution, developers can make *worst-case* performance predictions of the applications on the network. These bounds are *worst-case* because the curves are functions of *time-window size*, instead of directly being functions of time. This distinction means that the worst service period provided by the system is directly compared with the maximum data production period of the application. Clearly such a comparison can lead to over-estimating the buffer requirements if the application's maximum data production does not occur during that period.

1.3 Real Time Calculus

Real-Time Calculus[*Thiele2000*] 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 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 miniumum and maximum computation requested and computation serviced, respectively. An overview of RTC is shown below.

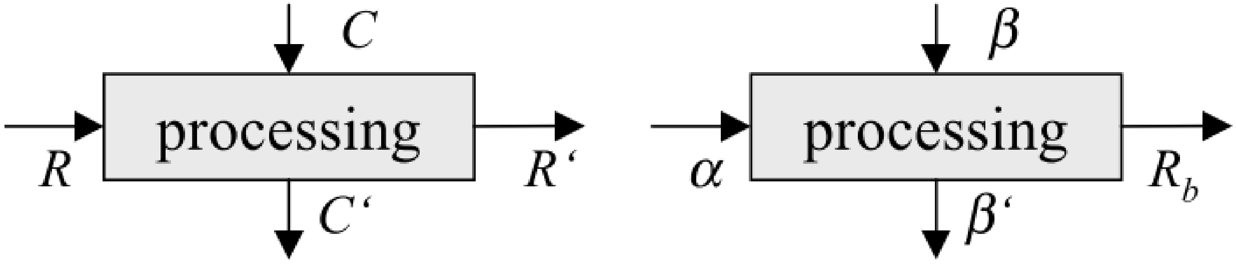


Fig. 1.4: Overview of Real-Time Calculus' request, computation, and capacity models.

$R(t)$ 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'.

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 $G(T)$. 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:

$$\beta'(\Delta) \geq \alpha_d^i(\Delta) \quad \forall\Delta$$

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:

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

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

CHAPTER

TWO

PRECISE NETWORK PERFORMANCE PREDICTION : THEORY

This chapter describes the mathematical formalization behind the network analysis techniques used by *Precise Network Performance Prediction*, (PNP^2).

2.1 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.

Similarly to Network Calculus' arrival curves and service curves, our network profiles model how the network performance or application traffic generation changes with respect to time. Whereas Network Calculus' modeling transforms application data profiles and network service profiles into min and max curves for data received vs. size of time-window, our models take a simpler, deterministic 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 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 (2.1). 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 as there will be packetization and communication header overhead as well. The buffer and delay equations (2.1) use the output of the convolution with the input profile to predict the minimum required buffer size for lossless tranmission and the maximum delay experienced by the transmitted data, respectively. A representative convolution example is shown below for reference.

$$\begin{aligned} y = l[t] &= (r \otimes p)[t] \\ &= min(r[t], p[t] - (p[t-1] - l[t-1])) \\ \text{buffer} &= sup\{r[t] - l[t] : t \in \mathbb{N}\} \\ \text{delay} &= sup\{l^{-1}[y] - r^{-1}[y] : y \in \mathbb{N}\} \end{aligned} \tag{2.1}$$

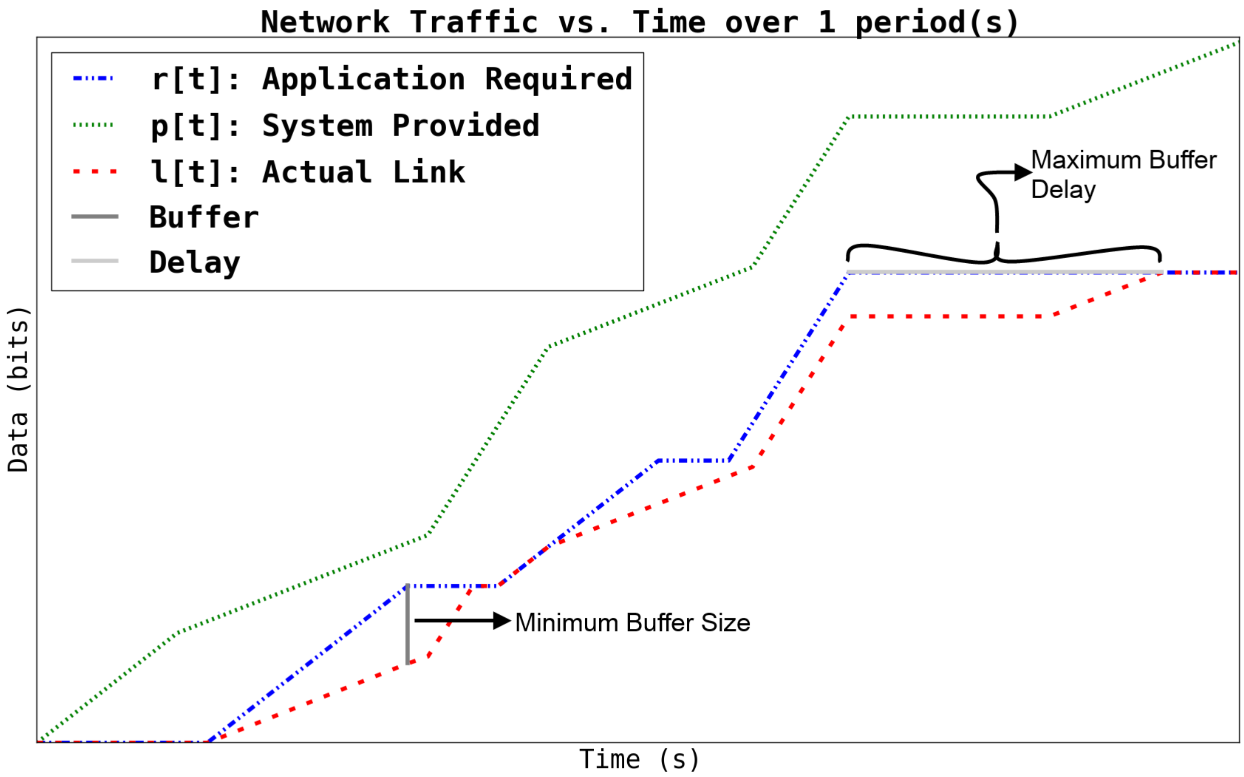


Fig. 2.1: Illustrative example for network profile convolution.

2.2 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 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.

2.3 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 bandwdith (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 bandwdith. 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 tradeoff 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.

CHAPTER

THREE

PRECISE NETWORK PERFORMANCE PREDICTION : RESULTS

This chapter covers the results of my research as it applies to analysis of networked CPS. I will cover the research contributions in two aspects:

- *Design Time Results* : Details design-time network analysis contributions and improvements to network performance prediction
- *Run Time Results* : Details the run-time network monitoring and management contributions which have been based of the design-time work

3.1 Design Time Results

These results provide a methodology and a means for application developers and system integrators to determine conservative, precise, tightly bounded performance metrics for distributed networked applications and systems at design time. The contributions of this work are broken into sections by topic:

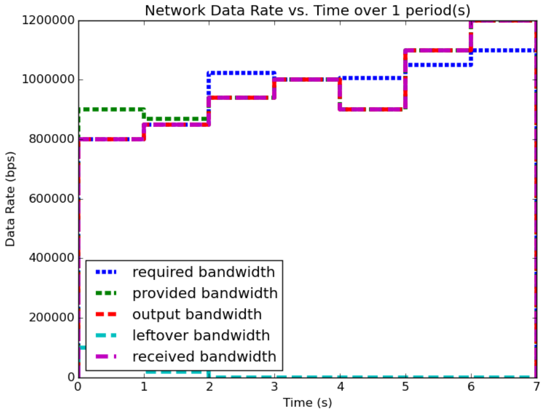
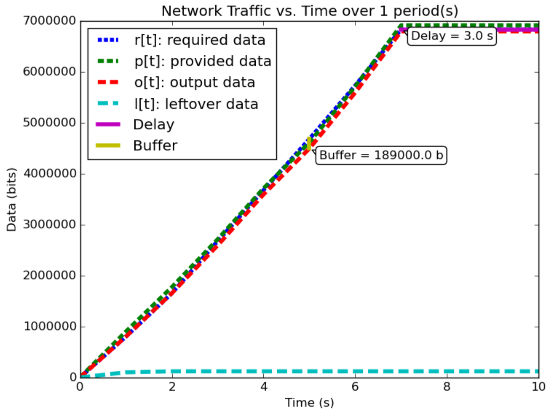
- *Periodic System Analysis*
- *Proving the Minimum Analysis for System Stability*
- *Comparison with NC/RTC*
- *Analysis of TDMA Scheduling*
- *Compositional Analysis*
- *Delay Analysis*
- *Routing Analysis*

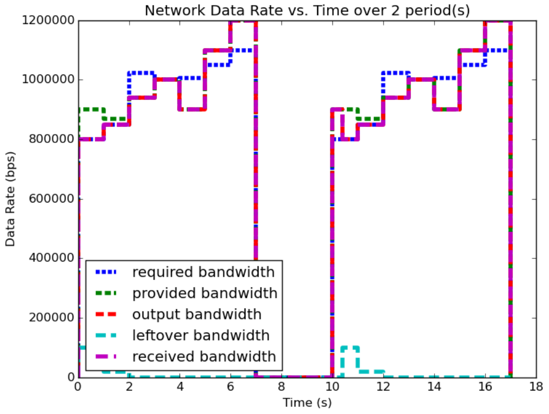
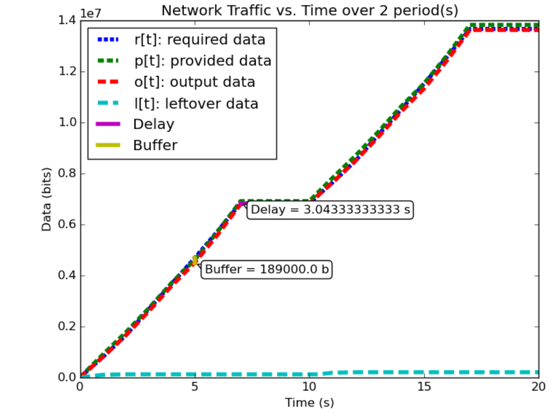
3.1.1 Periodic System Analysis

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 below (note: you can click on the images to open them in a larger format):

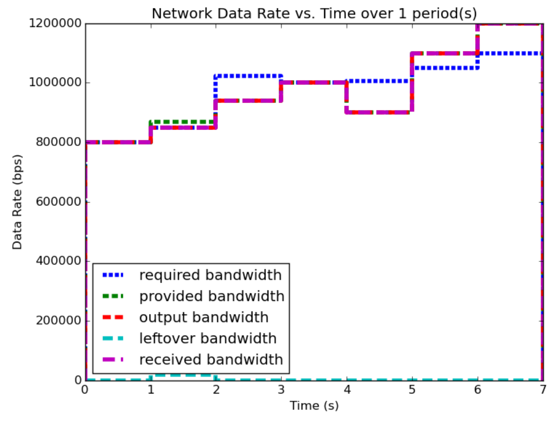
System *(1)* Bandwidth for 1 Period	System *(1)* Data for 1 Period
	

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

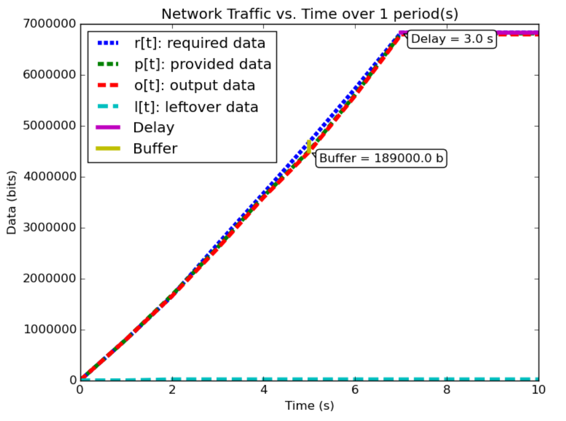
We notice that for this example system, the second period output profile is not an exact copy of the first (most easily seen by examining the bandwidth plots), and yet the required buffer size is still the same as it was when analyzing the system over one period. Furthermore, by running the analysis 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.

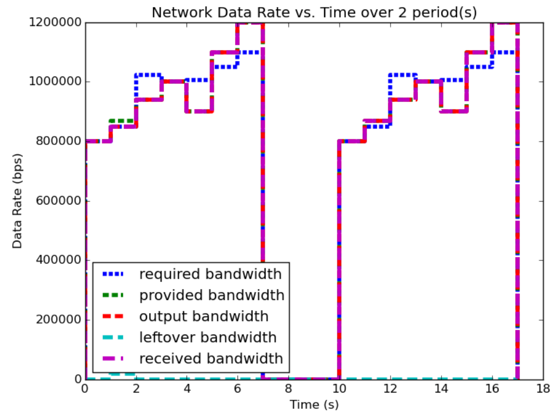
System *(2)* Bandwidth for 1 Period



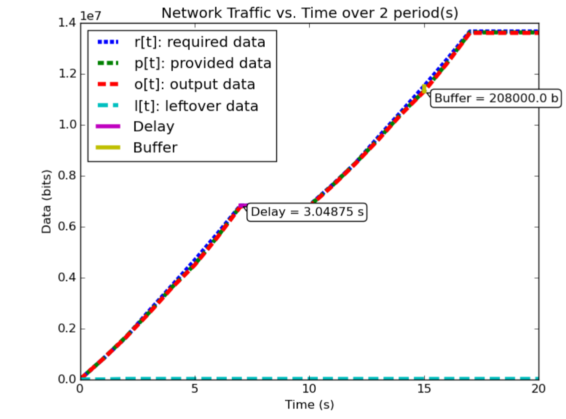
System *(2)* Data for 1 Period



System *(2)* Bandwidth for 2 Periods



System *(2)* Data for 2 Periods



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, 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 regarless 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.

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:

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

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

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

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, procduce 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,

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

Similarly, given the period T_S of S,

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

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:

$$B[m * T_p] > B[n * T_p], \forall m > n > 0$$

implying that:

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

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:

$$B[m * T_p] = B[n * T_p], \forall m, n > 0$$

which implies that:

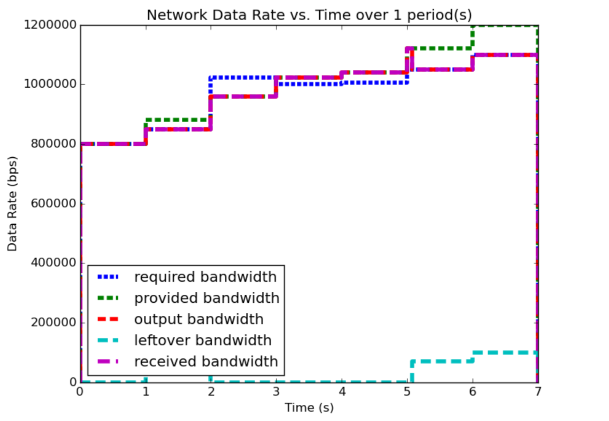
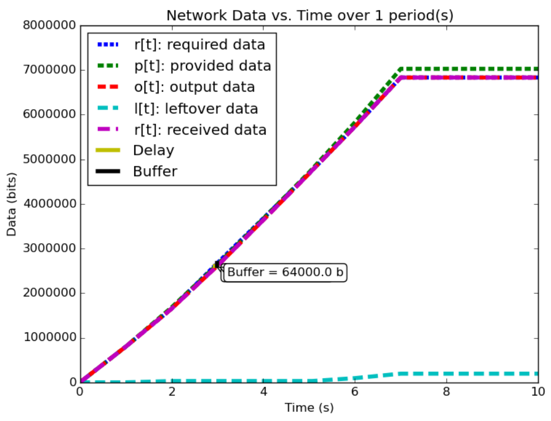
$$B[t] = B[t + k * T_p], \forall k \in \mathbb{N} > 0$$

meaning that the amount of data in the buffer versus time is a *periodic function*, therefore the 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 $n = 1$.

3.1.2 Comparison with NC/RTC

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 the actual system.

System Data Rate vs. Time	System Data Analyzed with PNP^2
	

The table above 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 3.1 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 3.2 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.

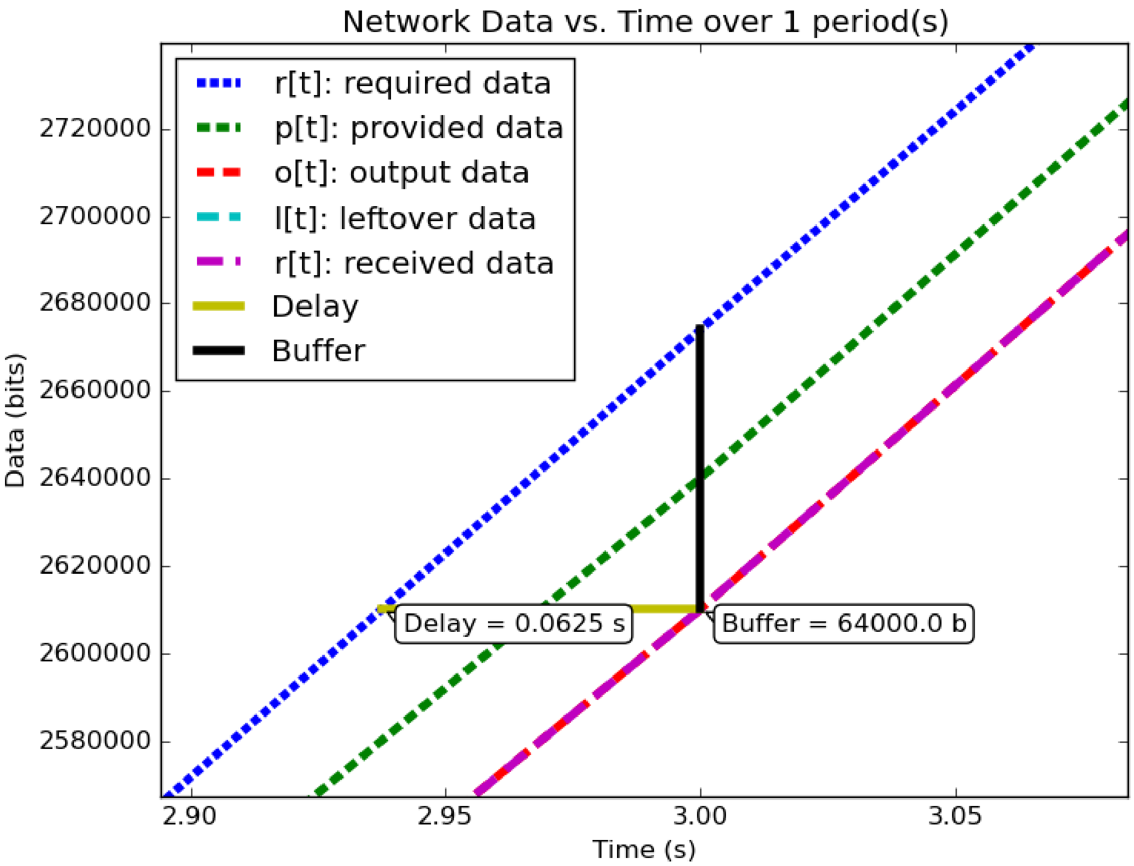


Fig. 3.1: Zoomed-in version of PNP^2 analysis.

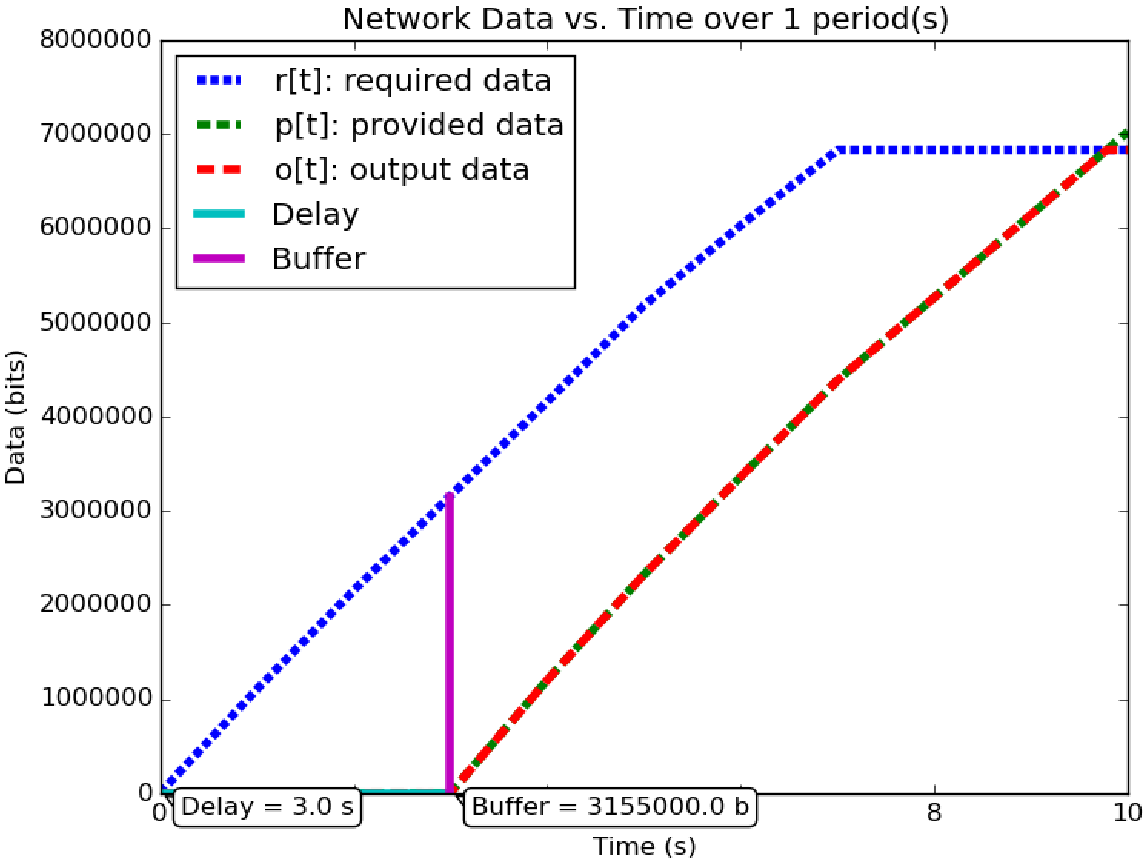


Fig. 3.2: Network-Calculus based analysis of the system.

Because our 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.

We developed software which produces data according to a supplied input profile and configured the system's network to provide the bandwidth profile described in the system configuration profile. Using this experimental infrastructure, 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 the table below:

	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)

Taking the results from our published work, where our methods predicted a buffer size of 64000 bits / 8000 bytes, 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.

3.1.3 Analysis of TDMA Scheduling

Medium channel access (MAC) 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. 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 MAReN 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.

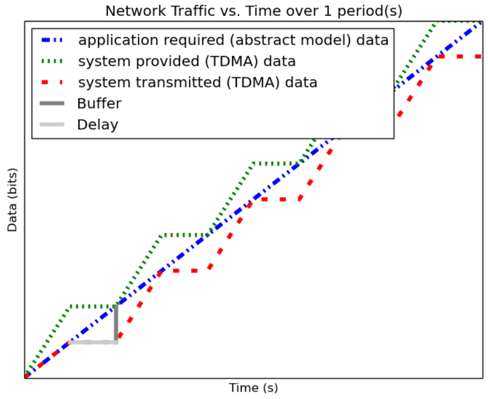
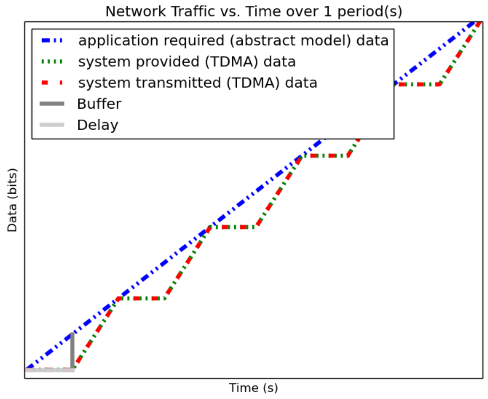
TDMA transmission scheduling has an impact on the timing characteristics of the applications' network communications. Because applications' network data production is decoupled from their node's TDMA transmission time slot, buffering may be required when an application on one node tries to send data 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.

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. A fractionated satellite cluster consists of many small satellites that may each have different hardware, computing, and communications capabilities. These capabilities are provided to distributed components of the satellite cluster's applications. Such a system has the combined challenges of (1) being expensive to develop, test, and deploy, (2) being very difficult to repair or replace in the event of failure, and (3) having to support mixed-criticality and possibly multiple levels of security applications. 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.

Such a system and applications necessitates design-time guarantees about resource utilization and availability. Applications which utilize the satellite network need assurances that the network resources they require during each part of the orbital period will be satisfied. To provide these assurances, we provide the application developers and system integrators the ability to specify and analyze the network profiles as (possibly periodic) functions of time. Furthermore, the requirement for accurate predictions necessitates the incorporation of the TDMA scheduling and bandwidth profiling into the network modeling and analysis tools.

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), the node can transmit a certain number of bits governed by its slot length (t_{slot}) and the slot's available bandwidth (bw_{slot}). 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 $bw_{effective} = bw_{slot} * \frac{t_{slot}}{T}$. 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 $\Delta_{delay} = T - t_{slot}$, and the maximum additional buffer space is $\Delta_{buffer} = \Delta_{delay} * bw_{effective}$. These deviations are shown below. Clearly, Δ_{delay} is bounded by T and Δ_{buffer} is governed by t_{slot}. Therefore, because t_{slot} is dependent on T, minimizing T minimizes both the maximum extra delay and maximum extra buffer space.

In-Phase TDMA profile vs abstract	Out-of-Phase TDMA Profile vs abstract
	

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.

3.1.4 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 perfomance predictions for specific applications. For this goal, let us define:

Compositionality:

> provides rules and guarantees ensuring that a component's
> properties will not change after system integration.

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.

For this 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. These 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 *Real Time Calculus*. 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)[*RFC2474*]. 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* [*RFC3246*] class receives strict priority queuing above all other traffic, which makes it a suitable implementation of this type of 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. The remaining capacity, P'_P, of the node after it services P_i is given as:

$$P'_P = P_P - (P_i \otimes P_P)$$

Where

- P_P is the capacity available to profile P_i

We are finalizing the design and code for tests which utilize the DSCP bit(s) setting on packet profiles to show that such priority-based analysis techniques are correct for these types of systems.

3.1.5 Delay Analysis

When dealing with queueing 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.

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

$$r[t + \delta[t]] = l[t]$$

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 $(t_{end}) - (t_{start}) = T_p$:

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

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

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

3.1.6 Routing Analysis

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.

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 decribed 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:

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

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:

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

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:

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

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:

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

Where

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

We can then run the following algorithm to iteratively analyze the system:

```
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)
    }
  }
}
```

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 retransmissions occur; only nodes which must route the profile to a new part of the network retransmit 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 implmented 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.

We are finishing the design and development of code which will allow us to run experiments to validate our routing analysis results. They will be complete in the next two weeks.

3.2 Run Time Results

3.2.1 Middleware-integrated Measurement, Detection, and Enforcement

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 3.3, is the smallest deployable part of an application and is defined as:

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

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.
- $\{P\}$ 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.

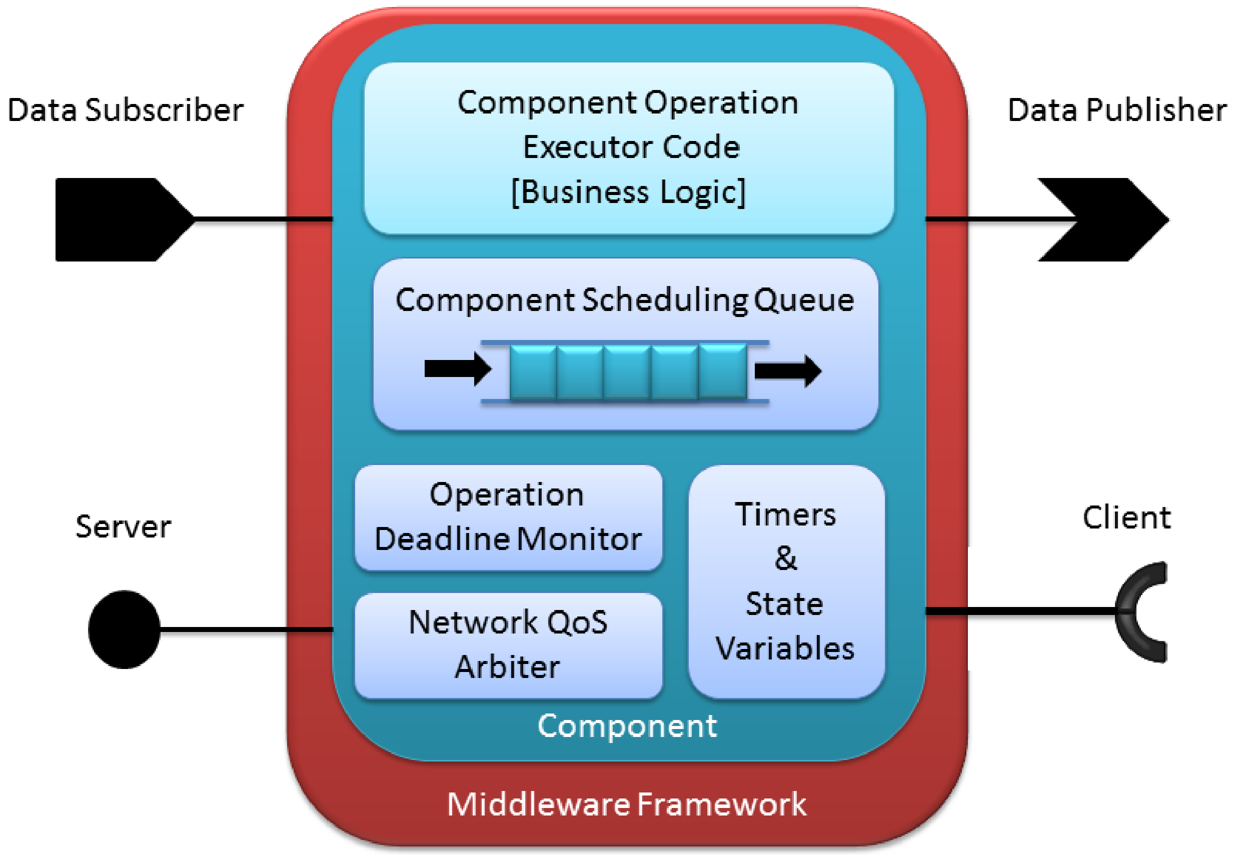


Fig. 3.3: Schematic representation of a software component.

From this component definition, we can define an application as:

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

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:

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

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 may be 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 3.4. 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.

We have implemented these features based on our design-time results:

- Traffic generators according to profile generated into sender code
- Receiver service according to profile generated into receiver code
- Measurement of output traffic on sender side and input traffic on server side generated into code
- Detection of anomalous sending on sender side
- Mitigation of anoumalous sending on sender side
- Detection of anomalous sending on receiver side
- Push back to sender middleware through out-of-band channel for anomaly detection on server side

Each of these functions uses the same profiles which enable design-time system and application analysis. This integration not only helps with running experiments and data collection but also helps to ensure model to system consistency.

We have implemented profile-based traffic generators 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 ROS as the communications middleware. For publish/subscribe interactions between components, into the generated code we add generic traffic generators which read their associated profile from the deployment XML file and publish traffic on their publisher port according to that profile. Additionally, these publish operations are generated to use a small wrapper 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 3.5.

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.

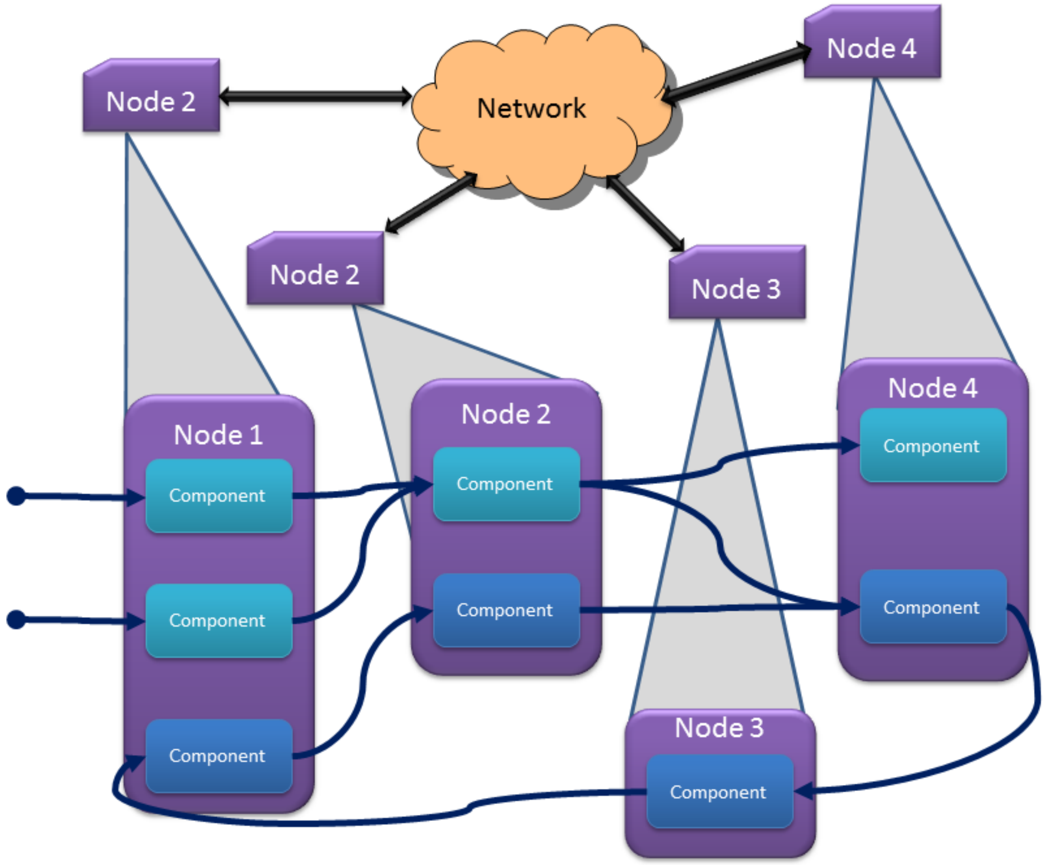


Fig. 3.4: Two example distributed CBSE applications deployed on a system with 4 nodes.

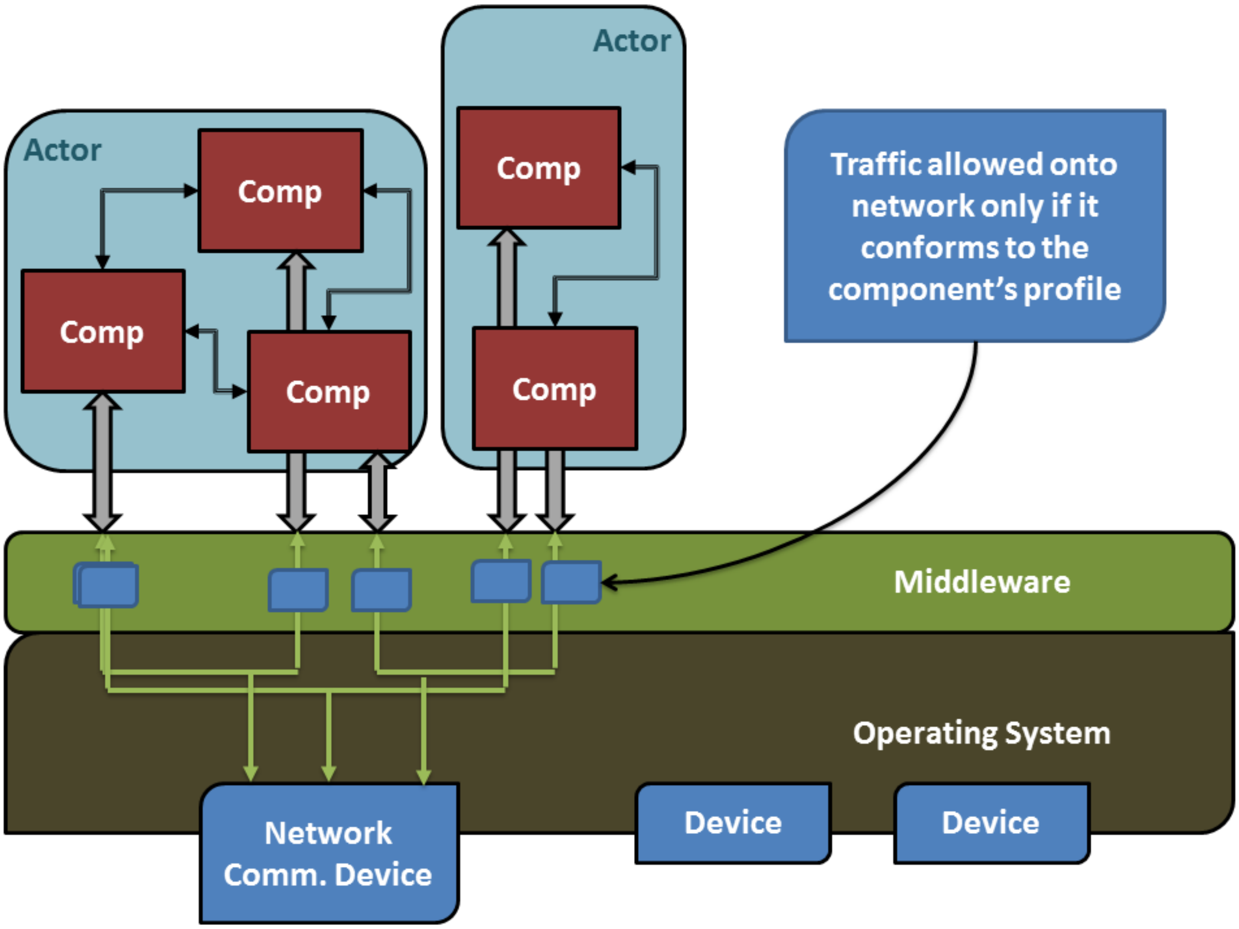


Fig. 3.5: The structure of component-based applications and how their network traffic traverses the middleware and the OS stack.

Into the receiver code (for subscribers) we additionally generate a receive buffer and receiver thread which reads the receiver profile from the deployment XML file and pulls data from the buffer according to the 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.

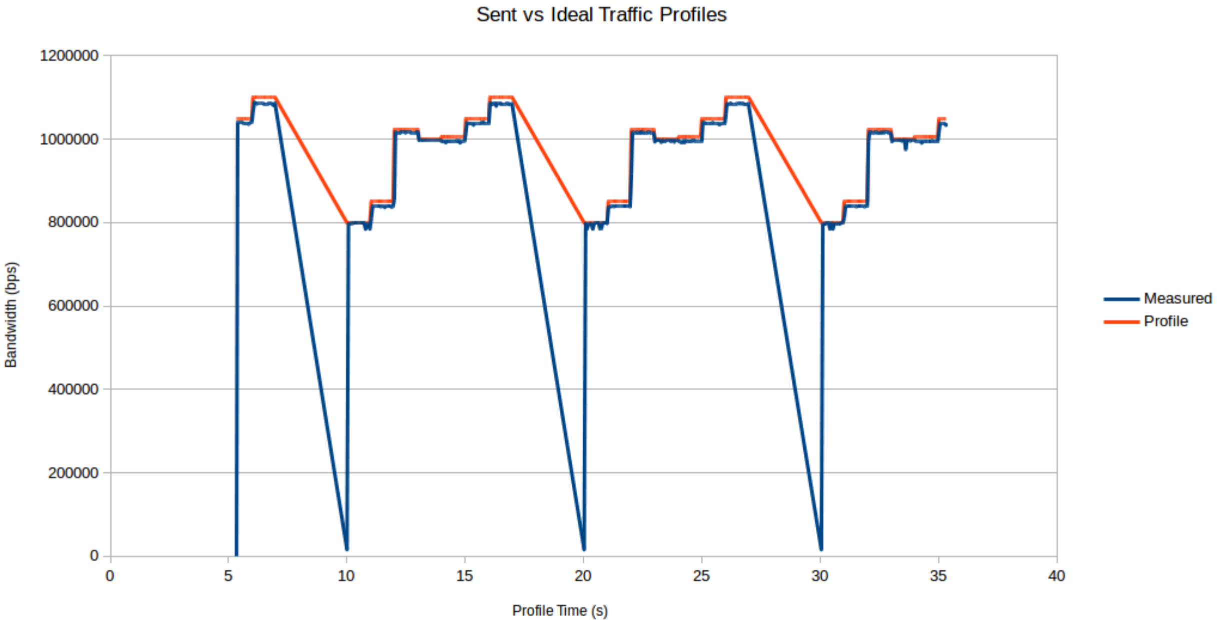


Fig. 3.6: Demonstration of the accuracy with which our traffic generators follow the specified profile.

Note: The measured bandwidth profile is calculated based on recorded time series data of $[reception\_time, message\_size]$, so the bandwidth drops to nearly 0 periodically since the Δt is so large between the messages.

Note: Our original implementation of traffic generators performed better since they did not utilize a middleware layer and relied instead on simple point to point ipv6 connections. However, that code was less useful for system analysis because it could do nothing aside from traffic generation and measurement; our current implementation which generates traffic generation code into component code is more versatile for several reasons:

- The component-based code integrates directly into our development toolsuite and deployment framework so it can be easily deployed on our cluster.
- Configuring different system topologies or component to host mappings (deployments) is simpler and more robust, allowing us to perform more and more varied experiments.
- The traffic generation code can be removed (or the code can be regenerated without the option selected) and the rest of the component-based and middleware code is still useful as an actual application.

3.2.2 Distributed Denial of Service (DDoS) Detection

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 such capability.

If we relax the constraint from the design-time section 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. Instead of treating the application profiles as the exact amount of data that the applications need to send, we assume they are the mean amount of data that the application will need to send over each respective interval. Additionally, we add an upper curve to the application profiles which is the guaranteed maximum that the application will ever try to send in each respective period.

For system analysis, we perform the design-time analysis on both the mean-data curve and the max-data curve which each give predicted buffer space requirements for the senders and receivers. Since buffer space on many of these systems comes at a premium, we cannot always take the maximum buffer size from this calculation as our implementation. For the senders, we are free to choose the allocated buffer space in the deployed system within this range as we see fit, since the senders have a large degree (or full) control over the sender rate and can meter themselves to not lose data. However, for the receivers, we can select a buffer size within the mean-max range based on our total system capacity and then rely on the measurement, detection, and mitigation code to ensure that we do not lose data due to buffer overflowing.

In this scenario, the receiver has been given knowledge of all of the sender profiles which might transmit data to it. When the receiver detects that the buffer is filling up quickly enough to overflow the buffer in the near future, it can use the measurements that have been gathered about the received traffic up to this point to determine which sender components to throttle temporarily.

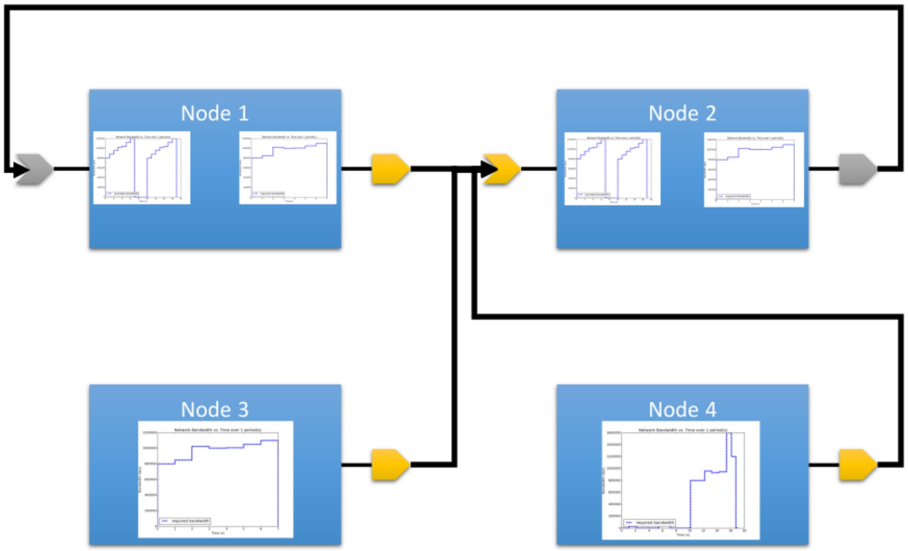


Fig. 3.7: The nodes in the network and how they communicate (using pub/sub).

We shown experimentally that, for example, a server side buffer size of 400000 bits, which would normally grow to 459424 bits because of excessive data pumps on the sender sides, is kept to 393792 by utilizing this out-of-band channel and secure middleware.

genindex

BIBLIOGRAPHY

[Cruz1991] R. L. Cruz. A calculus for network delay-I: Network elements in isolation. *IEEE Transactions on Information Theory*, 37(1):114-131, 1991

[Cruz1991a] R. L. Cruz. A calculus for network delay-II: Network analysis. *IEEE Transactions on Information Theory*, 37(1):132-141, 1991

[Thiran2001] J.-Y. Le Boudec and P. Thiran. *Network Calclulus: A Theory of Deterministic Queuing Systems for the Internet*. Springer-Verlag, Berlin, Heidelberg, 2001.

[Thiele2000] L. Thiele, S. Chakraborty, and M. Naedele. Real-time calculus for scheduling hard real-time systems. In *ISCAS*, pages 101-104, 2000.

[RFC2474] K. Nichols, Cisco Systems, et al., “Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers,” IETF, RFC 2474, Dec. 1998. [Online]. Available: https://tools.ietf.org/html/rfc2474

[RFC3246] B. Davie, A. Charny, et al., “An Expedited Forwarding PHB (Per-Hop Behavior),” IETF, RFC 3246, Mar. 2002. [Online]. Available: https://tools.ietf.org/html/rfc3246