## ABSTRACT

BOLOOR, KEERTHANA. Multi-point to Single-point Service Traffic Shaping. (Under the direction of Yannis Viniotis).

Service providers within an enterprise network are often governed by Client Service Contracts (CSC) that specify, among other constraints, the rate at which a particular service instance may be accessed. The service can be accessed via multiple points (typically middleware appliances) in a proxy tier configuration. The CSC and thus the rate specified have to be collectively respected by all the middleware appliances. The appliances locally shape the service requests to respect the global contract. We investigate the case where the CSC limits the rate to a service to X requests with an enforcement/observation interval of T seconds across all the middleware appliances. In this thesis, we define and evaluate the performance of Credit-based Algorithm for Service Traffic Shaping (CASTS), a decentralized algorithm for service traffic shaping in middleware appliances, in both a simulation and a realistic production level enterprise network setting. We show that CASTS respects the CSC and improves the responsiveness of the system to the variations of the input rate and leads to larger service capacity when compared to the traditional static allocation approach.

# Multi-point to Single-point Service Traffic Shaping

by
Keerthana Boloor

A thesis submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the degree of
Master of Science

Computer Networking – Electrical Engineering

Raleigh, North Carolina

2009

APPROVED BY:

| | |
|---|---|
| _____ | _____ |
| Dr. Yannis Viniotis<br>Committee Chair | Dr. Michael Devetsikiotis |
| _____ | _____ |
| Dr. George Rouskas | Dr. Robert David Callaway |

**DEDICATION**

*To my parents*

## BIOGRAPHY

Keerthana Boloor was born on November 19, 1983 in Madurai, India. She graduated from Visweswaraiah Technological University in June 2005, with a Bachelor of Engineering degree in Electronics and Communication. She has worked from July 2005 – July 2007 in Philips Healthcare, Bangalore, India as a Software Engineer on Philips Xcelera Webforum, an enterprise level remote clinical studies access provider.

Keerthana has been working towards her Master of Science degree in Computer networking at North Carolina State University since August 2007. She was an IBM Websphere Technology Institute intern from June 2008 – December 2008 working on IBM Datapower, a service oriented networking appliance. She is currently working on her thesis under the guidance of Dr. Yannis Viniotis.

Upon completion of her M.S degree, Keerthana will pursue her Doctoral degree in Computer Engineering under the guidance of Dr. Yannis Viniotis and Dr. Rada Chirkova focusing on the design and evaluation of mega scale transaction systems. Her research interests are in service oriented networking and distributed systems.

## ACKNOWLEDGMENTS

I would like to express utmost gratitude to my advisor Dr. Yannis Viniotis for providing me the opportunity for working under him towards my Master's thesis. His insight, guidance and patience have been most valuable and I have learnt tremendously from him this past year and half and I am grateful to him for providing me with the opportunity to work at IBM and as a TA at NCSU.

Special thanks to Dr. Bob Callaway for all his insightful suggestions towards the thesis, all his valuable support when in IBM and for serving on my committee.

I would like to thank Dr. Adolfo Rodriguez for all his valuable inputs during the thesis work at IBM.

My gratitude to Dr. George Rouskas and Dr. Michael Devetsikiotis for serving on my thesis committee.

I would like to thank my dad who has always inspired me to aim high. His encouragement and his confidence in me have always been my driving forces. Heartfelt gratitude to my mom from whom I have learnt the real meaning of limitless love and passion for perfection.

Special thanks to my close friends Karthik, Vibha, Vineela, Priyanka, Kavya, Nishant, Naveen, Bina, Roopa, Preethi, John, Mehul, Bharath, Sachin, Pukhraj, Roshani and my favorite cousin Chandana for all the fun times.

## TABLE OF CONTENTS

## LIST OF FIGURES

# CHAPTER 1

# Introduction

### 1.1 Current trends in information technology

The development and widespread use of powerful communications techniques for computers has led to the mass popularization of the Internet, which is fueling major growth in the information technology industry.

Competitive advantages are gained by businesses when they provide global access to their products and services via the internet. The need of the hour is to create interoperable IT solutions where the service can be offered to diverse users and their applications. For example, a company that made software for human resource management at boutique hotels might once have had a hard time finding enough of a market to sell its applications. But a hosted application on the internet can instantly reach the entire market (Schuller).

### 1.2 Web services

Adoption of XML Web services has emerged as one of the main technology drivers in enabling the development of platform-independent interoperable systems. Microsoft defines Web services as "A piece of business logic accessible via the Internet using open standards." Web services are building blocks for creating standardized (via XML) open distributed systems, and allow companies and individuals to quickly and cheaply make their digital assets available worldwide. One early example is Microsoft Passport (Microsoft), a convenient authentication service hosted by Microsoft. Here are some other examples:

- a credit checking service that returns credit information when given a person's social security number.
- a stock quote service that returns the stock price associated with a specified ticker symbol.
- a purchasing service that allows computer systems to buy office supplies when given an item code and a quantity.

A Web service can aggregate other Web services to provide a higher-level set of features. For example, a Web service could provide a set of high-level travel features by orchestrating lower-level Web services for car rental, air travel, and hotels. Applications of the future will be built from Web services that are dynamically selected at runtime based on their cost, quality, and availability (Glass).

Most IT components of businesses are modeled as Service Oriented Architectures which provides methods for systems development and integration where systems package functionality as interoperable services. Applications in SOA are thus built out of services.

### 1.3 Service Oriented Architecture

Service oriented architectures (SOA) have emerged as a mainstream technology for satisfying business goals in terms of flexibility and collaboration with industry partners (Maurizio, Sager, Corbitt, & Girolami, January 2008). The purpose of this architecture is to address the requirements of loosely coupled, standards-based, and protocol-independent distributed computing, mapping enterprise information systems appropriately to the overall

business process flow. In an SOA, software resources are packaged as “services”, which are well defined, self-contained modules that provide standard business functionality and are independent of the state or context of other services (Papazoglou & Heuvel, July 2007).

The XML data received by the SOA has to be parsed to extract business logic by the servers hosting the service end point. With its wide applications, adoption of web services hinges on its performance. The parsing of XML is performance intensive (Liu, Li, Chen, Lin, & Ma, Aug 2007). For this reason enterprise network administrators provide specialized hardware appliances for XML processing (Cuomo, June 2005). These appliances, called “middleware appliances or SOA appliances”, sit on the edge of the enterprise network performing two primary functions, offloading XML processing (for performance improvement) from primary enterprise servers and providing intelligent routing and shaping of service requests to the enterprise servers. The service requests from the clients are routed to the servers hosting the business functionality via the middleware appliances and thus these appliances are the access points for the service host (Cuomo, June 2005).

### 1.4 Motivation and Contribution

Access to services in an IT enterprise built on service oriented architectures is governed via **Service Level Agreements** which is a negotiation between two parties for interoperation of their business applications. A web service in a SOA hosted in an enterprise network will be governed by a **Client Service Contract** dictated by the SLA. One of the terms of the Client Service Contract specified by a service instance is the Service Access Requirements (SAR). An SAR typically consists of two parameters, the maximum number of service requests X and the enforcement period T. The SAR will state, “Limit the rate to a service provider to no

more than X service requests with an enforcement interval of T seconds". This has to be enforced by all access/entry points (middleware appliances) collectively and the total number of service requests sent to the service with the given SAR should not exceed "X" within the enforcement period of "T" seconds. The middleware appliances in the proxy tier need to collectively respect this CSC and shape the service requests according to the rate specified. *The main challenge is to enforce the global traffic contract by taking local actions at each appliance*. To the best of our knowledge, when the number of access points, B, to a service is greater than one, the common practice is a manual static allocation (MSA) of the global shaping requirement of "X" requests with the observation interval of "T" seconds to the "B" entry points with each appliance receiving a maximum of X/B requests in the observation interval of T seconds. If input traffic loads at each entry point are fairly static and known in advance, allocations other than the above mentioned one are also possible. In both cases, leaky buckets are used locally to shape traffic.

We propose that SOA appliances which forward requests to a service instance coordinate with each other to meet the CSC in a *dynamic* fashion in order to overcome the obvious limitations of MSA techniques. We propose CASTS (Credit-based algorithm for service traffic shaping), an algorithm that computes the maximum number of requests that the appliance can forward to the service instance within a specific time interval adhering to the global contract. CASTS relies on a strategic division of the observation period into subperiods that serve both as a measurement and enforcement period. The number of service requests forwarded by a middleware appliance (i.e. the credits allotted to each appliance)

depends on the input activity in relation to its peer appliances. The allotted credits are updated at each subperiod, which helps to ensure the adaptability of the system.

As a summary, the contributions of our work are:

1) We explain the problems with the static credit allocation and identify the need for a dynamic, measurement-based allocation scheme.

2) We implement a reactive algorithm that enables each appliance to dynamically react to the changes in the global state of the system.

3) We perform a number of experiments in a real middleware appliance and show that our approach does respect the contract even under the constraints of a realistic situation.

## 1.5 Organization

This thesis is organized as follows:

In Chapter 2, we explain the evolution of distributed computing, the emergence of web services, implementation of web services, emergence of service oriented architectures and service oriented networking, SON appliances, SON appliance functions, Service Level Agreements and need for service traffic shaping. In Chapter 3, we define and evaluate the CASTS, provide details of the simulation results and present two flavors of the algorithm to cater to the real world enterprise networks. In Chapter 4, we summarize our work and propose extensions for future work on the CASTS algorithm.

# CHAPTER 2

# Service Oriented Networking

## 2.1 Evolution of distributed computing

Software development has moved from applications being built with the use of infrequently changing libraries which are available as specific programming language APIs to programs interacting with other programs over the internet irrespective of the platforms in which they are built on.

There has been a wide-spread adoption of distributed computing in information technology due to the problems faced with mainframes. In last two decades mainframes were widely utilized in the distributed client/server architecture (Edelstein, 1994) where the mainframe shared a file with the client PCs which ran computations required by the user; however the increase in the number of online users increased the shared usage and update contention causing a strain on the file sharing architecture capacity.

This led to the emergence of 2 and 3 tier architectures. In 2-tier architectures, the Graphical User Interface (GUI) components of the application are located on the client PC and the data management and updation components are located on a server which is usually a powerful server. This architecture would scale up to 100 users and was typically adopted in an intranet application. The 3-tier architecture emerged to solve the problems faced by the 2-tier architecture and added an additional tier in between the GUI and the data management components. This could be a messaging middleware, a transaction broker which perform queuing, priority scheduling, etc. on the requests from the clients and forward the requests to

the data servers. The 3-tier architecture improved performance and scalability when compared to the 2-tier architectures.

While the 3-tier architecture managed to abstract the logic design of the system, the complexity of the interoperating interfaces was hard to resolve due to the low-level technical platform dependencies between the processes. This led to the adoption of RPC or Remote procedure call (Microsoft) which is a standardized mechanism for allowing a subroutine to be called in another address space without the need of the programmer explicitly coding the details of the remote interaction. In order to allow servers to be accessed by differing clients, a number of standardized RPC systems have been created. Most of these use an interface description language (IDL) to allow various platforms to call the RPC.

RPC analogues found elsewhere include:

- Java's Java Remote Method Invocation (Java RMI) API provides similar functionality to standard UNIX RPC methods. (Sun Microsystems)
- XML-RPC is an RPC protocol which uses XML to encode its calls and HTTP as a transport mechanism. (UserLand Software)
- Microsoft .NET Remoting offers RPC facilities for distributed systems implemented on the Windows platform. (Microsoft)
- CORBA provides remote procedure invocation through an intermediate layer called the "Object Request Broker" (Object Management Group)
- AMF allows Flex applications to communicate with back-ends or other applications which support AMF. (Adobe).

Web services extend the RPC paradigm to limit the interface definitions and messages exchanged to follow a standardized XML (eXtensible Markup Language) specification. Web services can also be modeled with the REST (REpresentational State Transfer architecture) (Fielding & Taylor, May 2002) architecture which treats objects as “resources” and facilitates the use of HTTP (or similar transport protocol) operations. Web services thus enable the development of a distributed, dynamic, on demand service oriented architecture.

Figure 2.1 shows the evolution of distributed computing [Redrawn from (Foster, January 2006)]

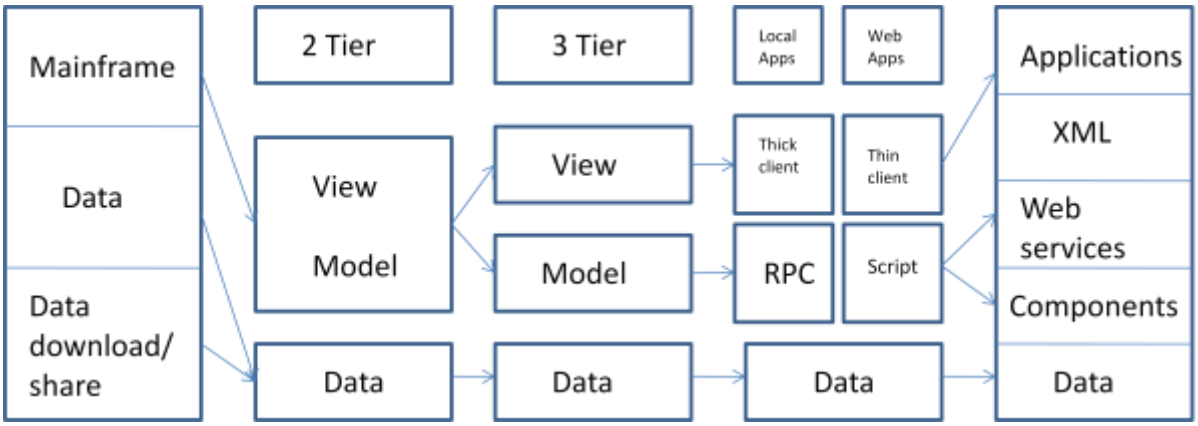



Figure 2.1: Evolution of distributed computing

## 2.2 Web services

A 'Web service' is defined by the W3C as "a software system designed to support interoperable machine-to-machine interaction over a network" (w3c, Web services). These self-describing, self-contained application components communicate over an open standard allowing compliant applications hosted anywhere in the world to access them. There has been an expansion in the adoption of Web services based business models. The development of web services has resulted in the emergence of “plug-compatible” software components that reduce the cost of software systems and also improve the capability of the systems. Web

services are building blocks for creating open distributed systems, and allow companies and individuals to quickly and cheaply make their digital assets available worldwide.

### 2.2.1 Implementation of Web services

Remote applications invoke web services from service providers. Web services are primarily based on XML and the service provider returns an "answer" to the "query" posed in XML. The use of standardized markup language like XML enables language and platform independence and web services can be automatically upgraded with the update propagating to all applications consuming the web service.

Web services are thus black box functionality providers and thus foster loose coupling among applications. We now explain the facilitating technologies or specifications for web services.

**SOAP (Simple Object Access Protocol)** (w3c, SOAP)**:** It is a protocol which transports to and from the service provider. It is in XML format and provides a wide variety of data types. Figure 2.2 shows the protocol layering for invoking web services based on SOAP.

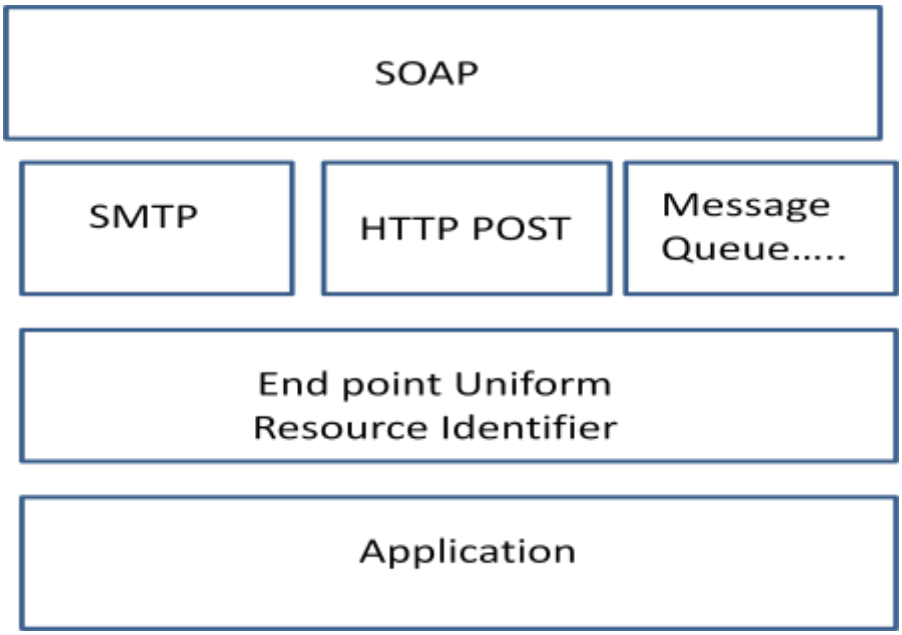



Figure 2.2: Protocol layering for invoking web services with SOAP

**WSDL (Web service description language)** (w3c, WSDL)**:** This is an XML standard which describes the web services offered by the service provider. It contains information such as where you can find the web service, methods and properties that it supports, its data types, and the protocol used to communicate with the web service. WSDL is used to create proxy objects. Basically, without a WSDL document, developers wouldn't be able to use web services simply because they wouldn't know which methods and properties they support and also which communication method any particular web service supports.

**UDDI (Universal Description, Discovery and Integration)** (OASIS)**:** When a large number of web services are available, there has to be a facility for locating required web services. UDDI is a registry that provides a place for a company to register its business and the services that it offers. People or businesses that need a service can use this registry to find a business that provides the service. When you search for a web service using UDDI's web

service or web browser, UDDI returns a listing of web services that matched your criteria. This list is returned in the form of a WSDL document.

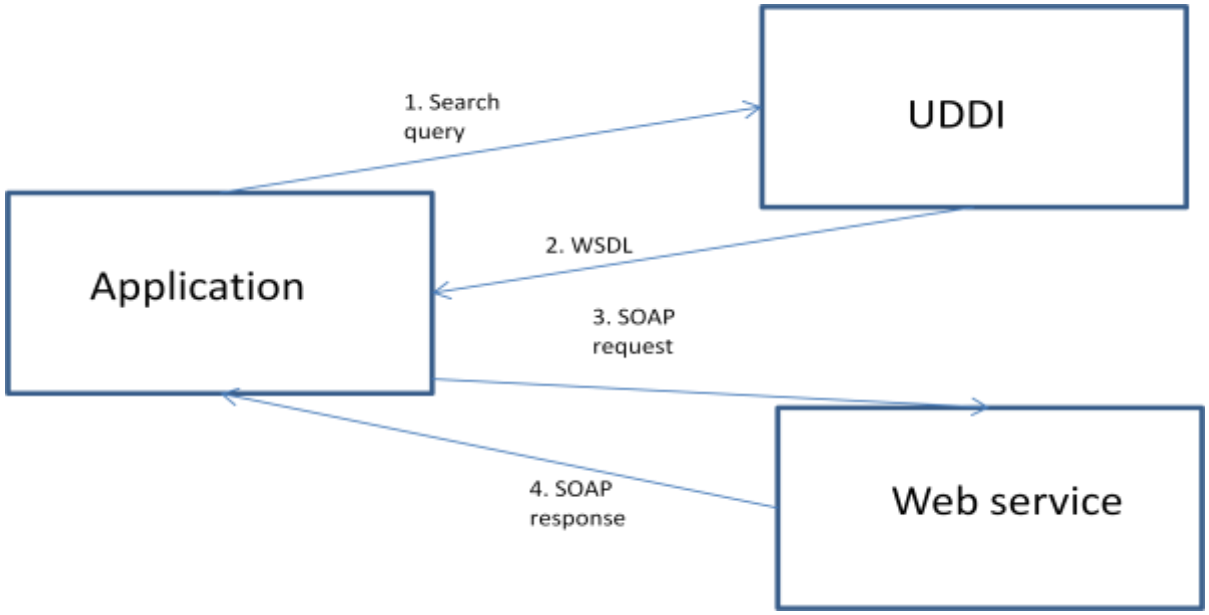



Figure 2.3: Shows the steps in invoking a web service

The following code shows the implementation and use of a simple web service (C#):

This is the method defined by the service provider which is exposed as a web service.

```
<%@ WebService Language="C#" Class="Example1" %>
using System.Web.Services;
[WebService(Namespace="urn:Example1")]
 public class Example1
{
   [ WebMethod ]
    public string sayHello(string name)
        {
                return "Hello " + name;
        }
}
```

Here the function sayHello in class Example1 is exposed as a web service. sayHello takes in a parameter of type string and returns another string. The method is qualified with

“WebMethod” attribute which will qualify it to be exposed as a web service. This can be deployed by using Microsoft’s IIS service.

The client which consumes the service can be implemented in any language, here implementation in C# is shown as it provides built in SOAP binding to the requests:

```
using System.Diagnostics;
using System.Xml.Serialization;
using System;
using System.Web.Services.Protocols;
using System.Web.Services;

[System.Web.Services.WebServiceBindingAttribute(
     Name="Example1Soap",
     Namespace="urn:Example1")]
public class Example1 :
             System.Web.Services.Protocols.SoapHttpClientProtocol {

    public Example1(  ) {
        this.Url = "http://localhost/helloworld.asmx ";
    }
    [System.Web.Services.Protocols.SoapDocumentMethodAttribute(
         "urn:Example1/sayHello",
         RequestNamespace="urn:Example1",
         ResponseNamespace="urn:Example1",
         Use=System.Web.Services.Description.SoapBindingUse.Literal,
 ParameterStyle=System.Web.Services.Protocols.SoapParameterStyle.Wrapped)]
    public string sayHello(string name) {
        object[] results = this.Invoke("sayHello",
                                       new object[] {name});
        return ((string)(results[0]));
    }

    public static void Main(string[] args) {
       Console.WriteLine("Calling the SOAP Server to say hello");
       Example1 example1 = new Example1(  );
       Console.WriteLine("The SOAP Server says: " +
                         example1.sayHello(“NCSU”));
    }
}
```

The client here invokes the web service method by creating an instance of the class Example1 and passes a string to the method. The method calls are “marshaled” into XML (SOAP format). The SOAP request sent by the client is:

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<soap:Envelope xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
xmlns:xsd=http://www.w3.org/2001/XMLSchema
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
      <soap:Body>
            <sayHello xmlns="urn:Example1">
                  <name>NCSU</name>
            </sayHello>
       </soap:Body>
</soap:Envelope>
```

The service end-point receives the requests unmarshalls it, calls the method and marshalls the return parameters into XML (SOAP format). The SOAP response obtained by the client from the service provider is:

```
<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope
   xmlns:SOAP-ENV='http://schemas.xmlsoap.org/soap/envelope/'
   xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
   xmlns:xsd='http://www.w3.org/2001/XMLSchema'>
   <SOAP-ENV:Body>
      <ns1:sayHelloResponse
         xmlns:ns1='urn:Example1'
         SOAP-ENV:encodingStyle=
            'http://schemas.xmlsoap.org/soap/encoding/'>
         <return xsi:type='xsd:string'>Hello NCSU</return>
      </ns1:sayHelloResponse>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Here we have seen how interoperability is obtained by using XML and any change in the service end point can be applicable to all clients consuming the web service.

In a typical enterprise a large number of diverse applications interact via web services. These services need to be unassociated, loosely coupled to reduce any tight transactional dependence between the components. Such architecture of interoperable systems is called the Service Oriented Architecture.

## 2.3 Service Oriented Architecture

To accommodate business agility, large enterprises streamline (existing) business processes and expose the various packaged and home-grown applications in a highly standardized manner. A popular approach for addressing this is by Web *services* that can be easily assembled to form a collection of autonomous and loosely coupled business processes. The emergence of Web services developments and standards in support of automated business integration has driven major technological advances in the integration software space leading to the wide spread use of service-oriented architecture (SOA) (Edelstein, 1994)

In SOA, the application developers need not be concerned with transaction management, application integration or security policies and can focus on the business logic.

Web services from various providers collaborate to collectively provide a common business solution. Each transaction in a SOA thus invokes multiple web services (Erl, 2008). Amazon.com has been using SOA techniques for some time, and has described how a single page request to amazon.com can access more than 100 separate SOA-based services (Singh & Huhns, 2005)

## 2.4 Service Oriented Networking

The success of Service Oriented Architecture depends on the ability of the enterprise architecture to rapidly and reliably transmit messages from one web service end point to the other (Cuomo, June 2005).

If individual web service components become overloaded or fail, then messages should be load balanced to other components that provide the same service. The messages have to be re-routed to newer components when they are created or when older components are

upgraded. The underlying enterprise network's ability in coping with these changes is important in establishing a robust, high-performance SOA. One of the major developments stemming from this need of an enterprise network infrastructure for SOA is the Service Oriented Networking (SON) or Application Oriented Networking (AON) (Zeus Technology Limited).

Service Oriented networking designs the network infrastructure to be application aware or message oriented. SON recognizes the need of the network to be responsive to the changes in the Service oriented architecture components it supports. Since data encoding of the messages (for the web services) is essentially standardized with adoption of XML, the enterprise network can now be "application aware" and thus improving the performance of integration of multiple environments in SOA. SON implements additional functions in the network infrastructure to aid the effective deployment of a service oriented architecture. For example as newer components are added, older services are upgraded or relocated, the messages in a SOA have to be rerouted accordingly. Since the messages in a SOA are encoded in the standardized XML format, content based routing/XPath (w3c, XPath) routing is possible in the network instead of at the endpoint servers thus conserving cycles at the end point. The conversion of XML to other formats usually via XSLT (w3c, XSLT) (compatible at the web service end point) can now be offloaded to the network components. Performance intensive functions like SSL encryption/decryption, management of security policies, Denial of Service attack detection can be offloaded to the network. Since XML web services are widely used components in SOA, the performance of XML parsing and XML schema validation can be a bottleneck for the success of SOA (Liu, Li, Chen, Lin, & Ma, Aug 2007).

This has led to the deployment of middleware appliances or SON appliances.

### 2.4.1 SON appliances

SON appliances are specialized appliances which typically are located at the edge of the enterprise network and perform accelerated XML parsing aiming towards two primary functions: a) Offloading of processing from primary service end points so that the end points focus on transactions and business logic. b) Intelligent routing of service requests to the service end points. Figure 2.4 depicts the positioning and operation of the SON appliance in an enterprise network.

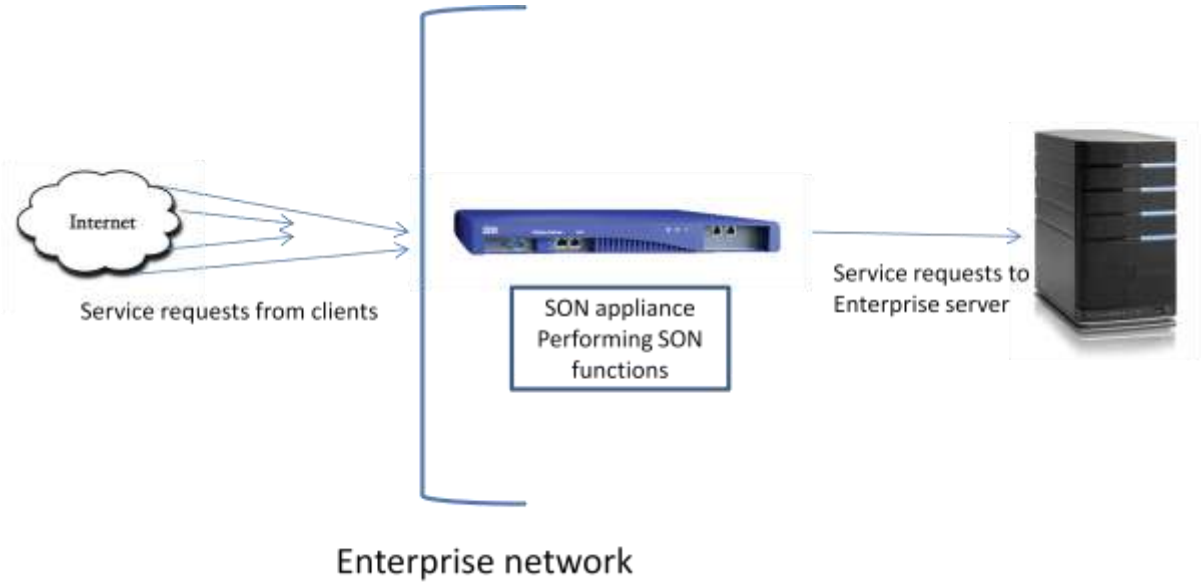



Figure 2.4: Shows the positioning and operation of SON appliances in an enterprise network.

#### 2.4.1.1 Offloading service processing

##### 2.4.1.1.1 Security offload

Security offloading has been one of the design patterns in the internet where the SSL traffic was offloaded by the HTTP servers before passing the request to the application servers. SON/Middleware appliances are provisioned to provide SOA cryptographic offloading

which has improved performance gains of 5 to 10 times in an enterprise network (Cuomo, June 2005).

#### 2.4.1.1.2 On the fly transformation

It is most desirable to have the end point servers execute requests when the XML payload of the request is in its most optimized format. Schema of the payload can be transformed from that of one provider to another provider which is commonly needed in businesses where services from one concern can be forwarded to another concern. Binary transformation is also a commonly required function where the CPU of the client (PDAs etc) is constrained and transformation of XML to binary is desired (Cuomo, June 2005). When service providers operate on different protocols then the SON appliances provide protocol transformation, for example from SOAP/HTTP to RMI/IIOP (for EJB components).

### 2.4.1.2 Intelligent routing

#### 2.4.1.2.1 Content based routing

Content based routing or XPath (w3c, XPath) routing have known to increase performance by 20 times when properly deployed (Cuomo, June 2005). CBR allows affinity of classes of services which can be identified by either port type or operation tags in the request XML and the service endpoints. As the requests can be segregated into classes, enterprise servers can be specialized for the particular class of functions, resulting in improved performance.

#### 2.4.1.2.2 Priority Based routing

Business policies for services can be incorporated in service level agreements. These requirements can state priorities for various classes of services. The SON appliance can use

the information embedded in the requests to forward the requests to different enterprise servers.

### 2.5 Service Level Agreements

Service level agreements are imposed by businesses in the SOA environment for the service provider to understand how the client will use the service and what the clients expect from the service (Muthusamy & Jacobsen, December 2008).

The SLA also specifies a Client service contract which specifies the limitations or terms of use of the service to the client. The use of contracts statically ensures the successful completion of every possible interaction between compatible clients and services (Castagna, Gesbert, & Padovani, January 2008 ). The Client service contract along with other constraints specifies the Service Access Rate (SAR), the rate at which a service may be accessed to prevent the overwhelming of the service end point.

#### 2.5.1 Service Traffic shaping

The requests from the clients must respect the Service access requirement. Since the SON/middleware appliances are positioned at the edge of the enterprise network and the service requests are forwarded from the clients to the end point service by the appliances, these appliances can be configured to shape the client service requests to respect the Service Access Requirement imposed in the Client Service Contract. This is the focus of this thesis, we describe it in more detail in the next chapter.

# CHAPTER 3

# Multi-point to single-point service traffic shaping

Service hosts in an enterprise network handle service requests from clients. The service may be hosted by a single server or multiple servers. The number of requests to the service host can unduly overwhelm the service host causing the server to fail and result in denial of service. To prevent the service host from being overwhelmed by the requests from clients, the service requests to the server should be shaped. This thesis analyzes the multi-point to single-point service traffic shaping problem where the service traffic targeting a single service host will be shaped collectively by multiple entry points.

## 3.1 Target Topology

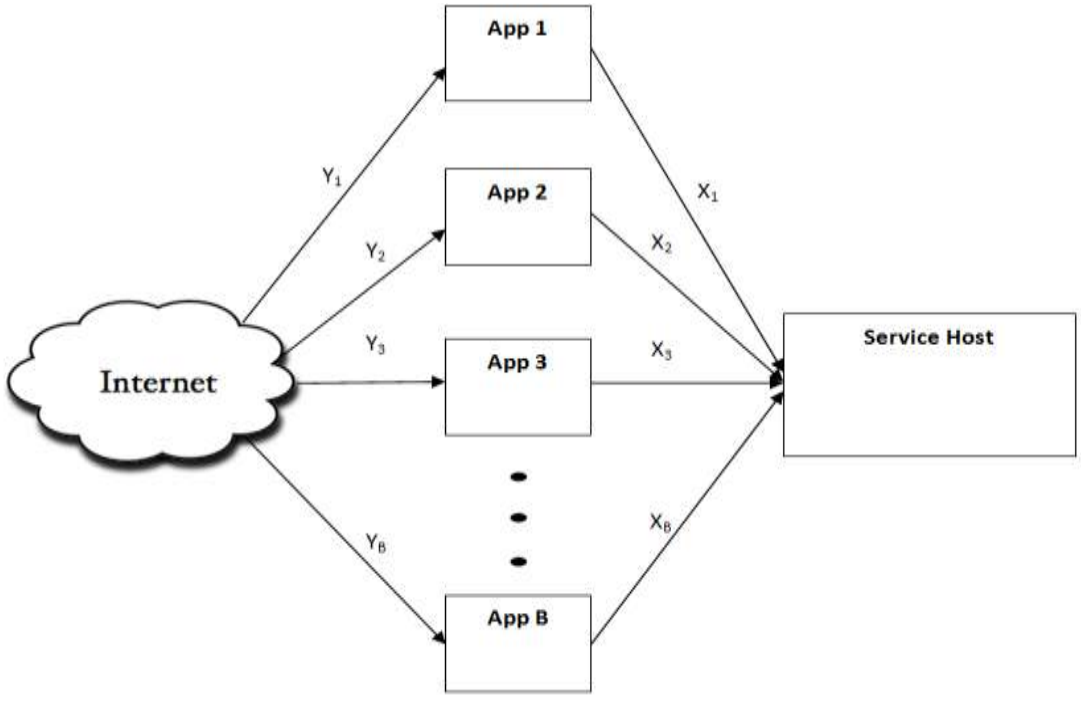



Figure 3.1: Target topology

The topology under consideration is as given in Figure 3.1. Under consideration is the multi-point to single-point case where multiple middleware appliances (B) forward web service requests to a single service host. Individual appliances must control the output rate of documents (based on some strategy) in order to meet the requirements specified in the contract (CSC). The elements we consider in our analysis are:

1. Client nodes. These are nodes that generate the web service requests. These are collectively represented by the internet cloud in Figure 3.1. $Y_i$ denotes the number of requests from clients to appliance i.

2. Middleware appliances. These are the SOA appliances positioned at the edge of the enterprise network. They accept the service requests from the client nodes via the Internet, perform value-added middleware function, and forward the requests to the service instance. They also perform shaping of the service requests to the service instance(s) according to the CSC which governs the target service. $X_i$ denotes the number of service requests sent from the appliance to the service instance in the observation interval.

3. Service host. This is responsible for effectively handling and responding to the requests. It also defines the rules in the service contract that will drive the operation of the appliances.

**3.2 Client Service Contract**

A service hosted by an enterprise service host is governed by a Client Service Contract (CSC) which with other constraints specifies a Service Access Requirement. The entry points (middleware appliances) have to respect the CSC and shape the service requests according to the rate specified.

### 3.2.1 Service Access Requirement

One of the terms of the Client Service Contract specified by a service host is the Service Access Requirements (SAR). An SAR typically consists of two parameters, the maximum number of service requests X and the enforcement period T. The SAR will state, "Limit the rate to a service provider to no more than X service requests with an enforcement interval of T seconds". This has to be enforced by all access/entry points (middleware appliances) collectively and the total number of service requests sent to the service with the given SAR should not exceed "X" within the enforcement period of "T" seconds from all the entry points. So each entry point should be assigned credits (number of service requests to forward to service host) to respect the SAR.

Let $x_i$ be the number of service requests allowed to be sent to the service host (credits allotted) by appliance i within an enforcement interval of T. We must enforce

$$\sum_{1 \le i \le B} x_i \le X \times T. \qquad (1)$$

### 3.2.2 Why is this problem different from network traffic shaping problem?

The service traffic problem is fundamentally different from the well-studied network traffic shaping problem (Parekh & Gallager, June 1993) (Elwalid & Mitra, Apr 1997) (Rexford, Bonomi, Greenberg, & Wong, June 1997).

1. In the realm of packet/ATM networks, the shaping is in a *single* scope, as shaping is performed on traffic from a single connection. However, in enterprise networks, service requests from multiple input points target the same service host. The shaping is now *global* in

scope as it should be applied to all the entry points. The novel challenge in enterprise networks is to enforce the global constraints with local shaping.

2. In the network world, the shaping requirement is determined by the allowed utilization of precisely defined and measurable resources like bandwidth and buffer space. The shaping contracts specified are standardized. In the enterprise networks, the CSC is not well defined or standardized with no maximum burst size specified. Also the resource protected by service shaping is CPU utilization, which varies based on the type, size, and *content* of the request documents.

### 3.3 Manual Static Allocation

The most common solution for respecting the CSC, i.e., allow only "X" requests in "T" seconds from "B" entry points is the manual static allocation technique where the allowed number of requests $x_i$ from each of the appliances is the same.

$$x_i = \left\lfloor \frac{X \times T}{B} \right\rfloor, \quad i = 1, \ldots, B. \qquad (2)$$

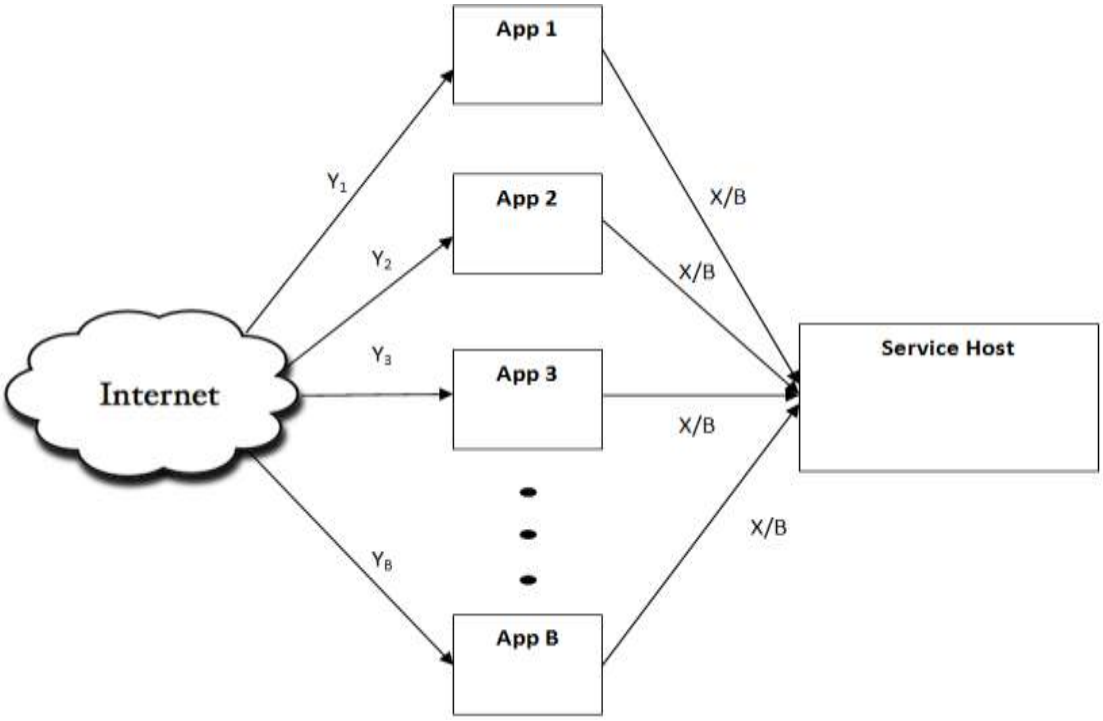



Figure 3.2: Manual Static Allocation when T=1.

Consider a simple scenario of 4 peers as shown in Figure 3.3.

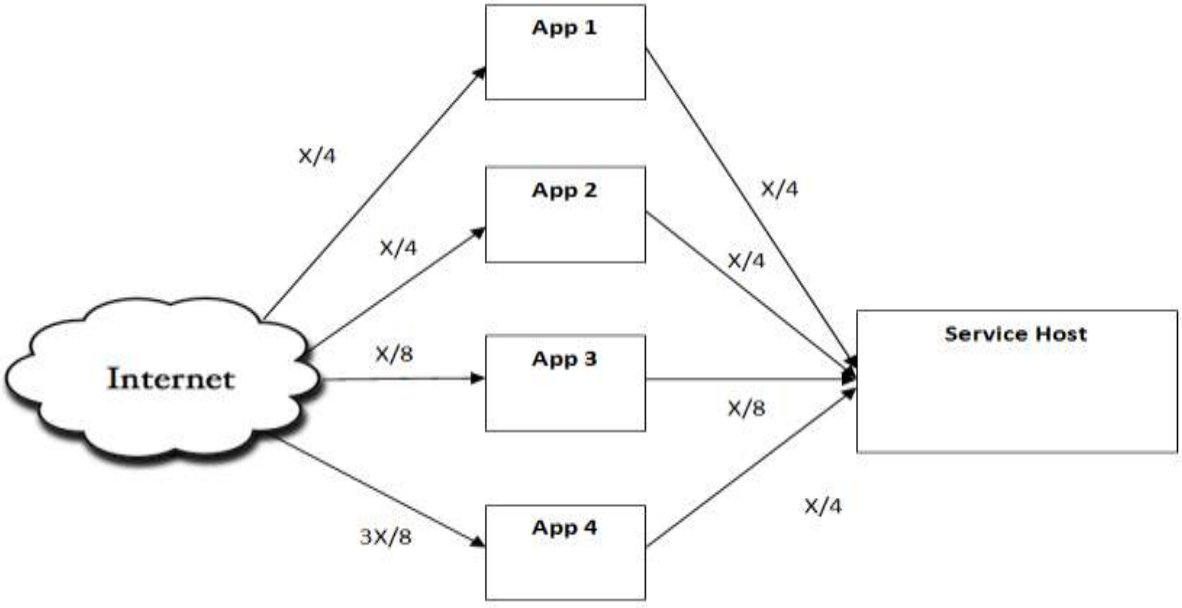



Figure 3.3: Manual Static Allocation with 4 appliances.

Suppose that each entry point gets X/4, X/4, X/8 and 3X/8 requests totaling X requests within T (=1) seconds as shown. In the static solution, each appliance is allotted a credit of only X/4. So the requests sent to the service hosts is 7X/8 as shown instead of X requests, even though X requests are available at the entry points. If the request rates at each entry point were the same, then the static solution would be satisfactory. However, the input request rates are in general, heterogeneous in nature and there is no a priori knowledge about the input rates at each entry point due to the high variability in parsing the XML requests /documents.

So, in the static case, due to heterogeneity of the input rates at the appliances, it is likely that some appliances block traffic while others remain idle, especially when the cumulated number of documents handled by the appliances is in the order of (or higher than) X * T.

Given the costs of implementing an enterprise network and the issues inherent to the provision of web services (Maurizio, Sager, Corbitt, & Girolami, January 2008), it is *fundamental to design efficient algorithms that improve the overall utilization of the system*. Such an algorithm is described next.

### 3.4 CASTS (Credit based Algorithm for Service Traffic Shaping)

The need exists for a dynamic measurement based service traffic shaping algorithm which respects the CSC and where credits are assigned to the middleware appliances (entry points) based on the current state of the system. The thesis proposes, CASTS, a reactive, measurement-based algorithm for the multi-point to single-point scenario enforcing the global contract by taking local action at each appliance. The basic idea behind CASTS is to adjust the number of service requests allowed to be sent to the service host (credits allotted to

the appliance) based on the number of service requests in the queues of the appliance which indicates the input rate at each appliance.

CASTS also introduces the concept of subperiod enforcement. The observation interval, T specified by the SAR is slotted into K equal subperiods during which the updating of the credits take place at each appliance by exchanging their respective queue details and current credit assignment.

### 3.4.1 Algorithm description

The algorithm can be implemented in a centralized or decentralized fashion. In centralized, appliances send their measurements to the central point (appliance). In decentralized, the middleware appliances (entry points) to exchange the information about their queues (an indication of their input rate) to their peer appliances every subperiods within the observation interval (which is divided into K equal subperiods) specified by the CSC.

When run on the appliances, the algorithm triggers the *adaptation phase* at the beginning of each subperiod where the credit assigned to the appliance is updated based on the current state of the system and the remaining duration of the subinterval, the appliance is in the *measurement phase* collecting the actual number of service requests sent to the host. This is illustrated in Figure 3.4. The credits available for the appliance to send during the subperiod k is $x_i(k)$. During the measurement subperiod k, each appliance measures the actual number of documents $r_i$ (k) it was able to send to the service host. If there was not enough traffic entering the appliance, $r_i$ (k) could be less that $x_i(k)$. Otherwise, $r_i$ (k) should be equal to $x_i(k)$. Each appliance also measures the queue size, $Q_i(k)$, at the entry points. The number of requests in the queues of the appliance is an indication of the input rate at each appliance and

will be used to determine the credits allotted to the appliance. In this way, credit assignment is performed under a weighted strategy.

At the end of the measurement subperiod k, each appliance broadcasts to all other appliances the values $r_i$ (k) and $Q_i$(k); after receiving feedback from all other appliances, appliance i updates its local shaping rate $x_i$ (k + 1) during the adaptation phase as shown in Algorithm 1. This is illustrated in Figure 3.5.

Note that this algorithm guarantees that the SAR is respected at all times, as the total amount of credits assigned to the appliances at each subinterval is always smaller or equal to the remaining allowed number of documents:

$$\sum_{1 \leq i \leq B} x_i(k) \leq X \times T - R(k),. \qquad (3)$$

where R(k) is the total number of documents sent by all appliances during the previous subperiods.

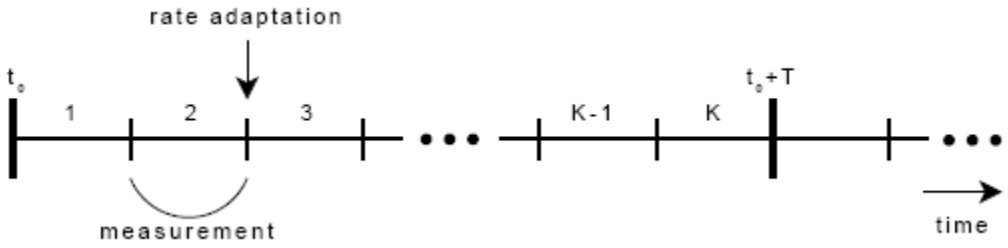



Figure 3.4: CASTS decomposes the observation interval into K subperiods

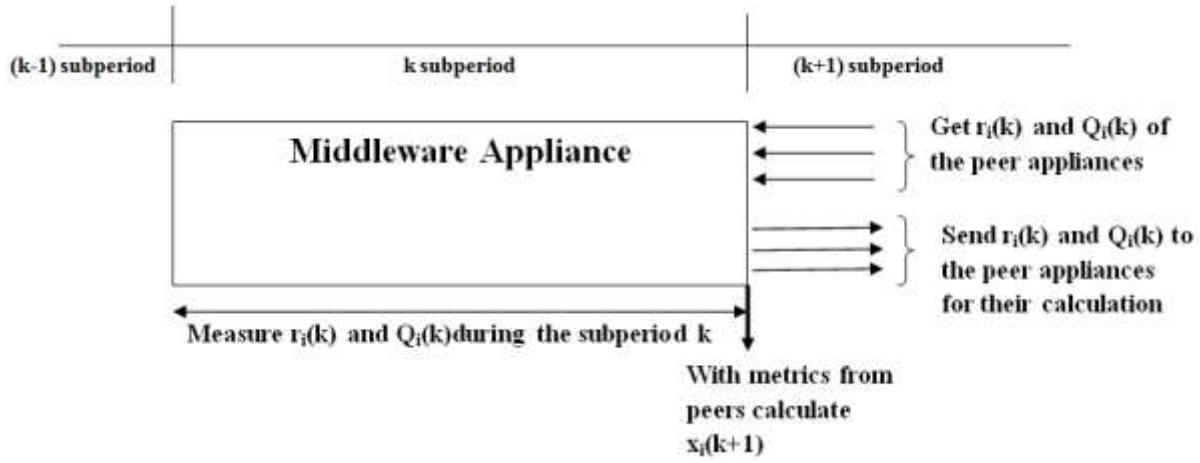



Figure 3.5: CASTS in operation during subperiod k

**Algorithm 1**

**Input**: k

**Input**: (When $1 < k \leq K$) $r_j(k\text{-}1)$ and $Q_j(k\text{-}1)$, $j = [1, .., B]$, $j \neq i$.

**Output**: new count $x_i(k)$

**if** $k=1$ **then**

$x_i(k) \leftarrow \lfloor X * T/B \rfloor$

**else**

$r(n) = \sum_{j=1}^{B} r_j(k\text{-}1)$ $n=1, \ldots, k\text{-}1.$

$R(k\text{-}1) = \sum_{n=1}^{k-1} r(n)$

**if** $R(k\text{-}1) < X$ **then**

$D \leftarrow X * T - R(k\text{-}1)$

**if** $\sum_{n=1}^{B} Q_n(k\text{-}1) = 0$ **then**

$x_i(k) \leftarrow \lfloor D/B \rfloor$

**else**

$x_i(k) \leftarrow \lfloor D * Q_i(k\text{-}1) / \sum_{n=1}^{B} Q_n(k\text{-}1) \rfloor$

**end if**

**else**

$x_i(k) \leftarrow 0$

**end if**

**end if**

### 3.4.2 Algorithm artifacts

**Flooring effect**

The algorithm calculates $x_i(k)$, the number of requests that an appliance can send to the service host (credits at each appliance) this must be an integer. The expression D/B or D * $Q_i(k)$ / $\Sigma$ $Q_n(k)$ will have to be rounded to the nearest integer value using the ceiling or flooring functions. Since the algorithm is run on all appliances individually, if the ceiling function is chosen, then all the appliances round the credits value to the higher integer and there is a possibility that the total number of requests sent collectively will be greater than X*T, thus no longer respecting the CSC. If all appliances use the flooring function as selected in Algorithm 1, then there is no possibility of the CSC being violated but some credits might be lost. The credits have to be redistributed in the next subperiod. In the worst case, the number of credits to be redistributed is B – 1 and all the appliances have the same queue size; then no requests will be sent, though credits are remaining. This loss of credits is called the flooring effect and is demonstrated in our simulation.

## 3.5 CASTS Evaluation

### 3.5.1 Evaluation Techniques

Evaluation of the CASTS algorithm was performed in two ways: by simulation and by implementing the algorithm on commercial middleware appliances in a production level enterprise network setting. In the experiments conducted with simulation certain environmental assumptions are made; for example, it was assumed for simplicity that the amount of time it takes for the appliances to exchange metrics is much less than T/K. In a real world setting, this assumption is not valid. So to analyze the behavior of the algorithm in

a real world setting and to size invalid assumptions, the algorithm was implemented in commercial middleware appliances in a production level enterprise network.

To evaluate the CASTS algorithm several experiments were run to answer questions about the operation, parameters varied, environmental factors etc.

### 3.5.2 Results from simulation

The simulation-based evaluation of the CASTS algorithm is centered on the questions given below.

1. Does CASTS respect the CSC?

2. Does the algorithm's claim of adapting to the current state of the system hold?

3. Is CASTS more responsible than the Manual Static Allocation (MSA) technique?

4. What are the parameters of importance to the CASTS algorithm and what is the effect of their variation on the behavior of the CASTS algorithm?

5. How do we modify the algorithm to include relevant assumptions in a real world setting?

We will now answer each of these questions.

#### 3.5.2.1 Does CASTS respect the CSC?

This section describes the behavior of CASTS for various input rates and types and verifies that CASTS respects the CSC at all times.

For the below given results, unless otherwise specified we set X = 128; B = 32; K=20.

Consider $Y_{in}$ to be the collective input number of requests at all boxes. Below are results for different input number of requests and distributions (bursty and non bursty).

#### 3.5.2.1.1 The behavior of the algorithm when $Y_{in} \sim X$ ($Y_{in}$ is approximately equal to X)

Figure 3.6 shows the behavior of the algorithm with average collective input rate of requests to all the middleware appliances to be $Y_{in} \sim X$. The input processes are modeled as a Poisson process (since we know that user generated web calls can be modeled as exponentially distributed (Paxson & Floyd, June 1995)). We find that the algorithm respects CASTS for input rate $Y_{in} \sim X$. Note that the algorithm automatically reacts to the variations in the input process and adjusts the available credits. At the beginning of the interval, the number of total credits in the system is equal to X, at the end of the first interval, credits in the system decrease as requests are sent to the server. The credits are reinitialized at the beginning of the next observation interval. The sum of the available credits and the total requests sent to the server will be equal to X at any instant of time. The slight deficits (requests sent to server are slightly less than X) that we observe in the curves are due to either the flooring effect or due to the actual number of documents produced ($Y_{in}$ is the average of a Poisson process; in some cases, the number of documents might be less than 128).

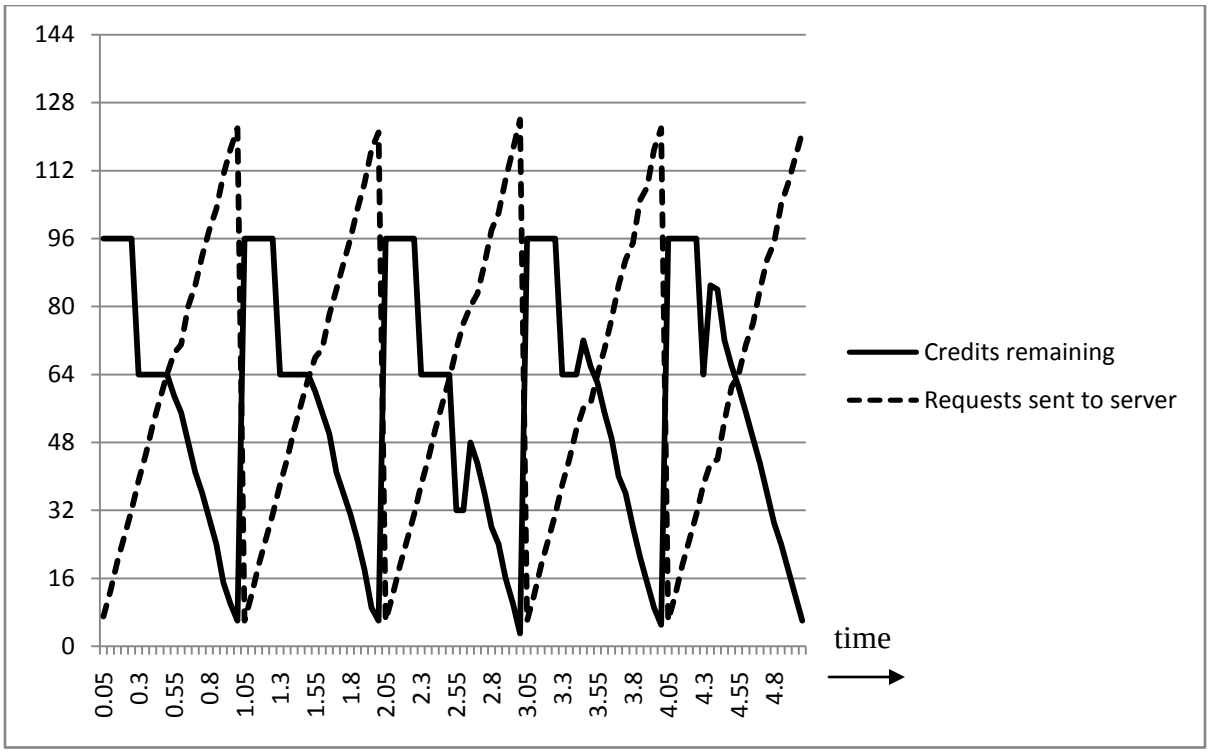



Figure 3.6: Variation of the credits remaining and requests sent to server when $Y_{in} \sim X$

**3.5.2.1.2 The behavior of the algorithm when input number of requests $Y_{in}$ is much greater than X.**

Figure 3.7 shows the behavior of the algorithm with average collective input rate of requests to all the middleware appliances to be $Y_{in} >> X$. The input processes are modeled as a Poisson process. The algorithm respects CASTS and does not send more than X requests in observation interval T for input rate $Y_{in} \gg X$. Due to the flooring effect, there is a warm up period of one observation interval for the algorithm to reach its optimum value of X (128).

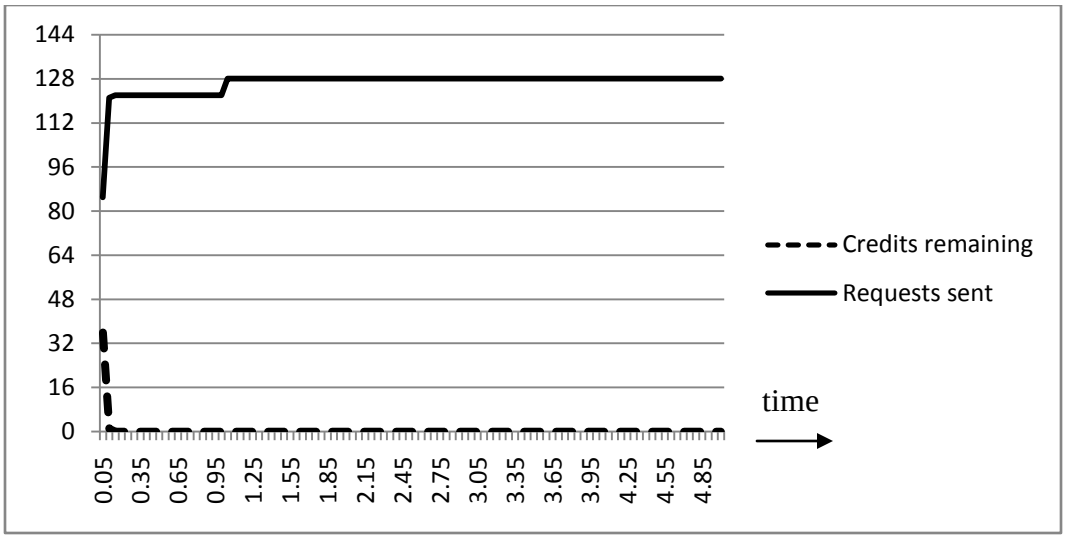



Figure 3.7: Variation of the credits remaining and requests sent to server when $\mathbf{Y_{in} \gg X}$

### 3.5.2.1.3 Bursty input with $Y_{in} \sim X$

The next set of results shown in Figure 3.8 is produced to investigate behavior of CASTS algorithm in case of collective bursty input(Bursty traffic models FTP like calls (Paxson & Floyd, June 1995) . The input is modeled with 3 bursts each of duration 0.1 seconds and 0.2 seconds apart; the arrivals at each burst are a Poisson process with rate $Y_{in} \sim X$. The results show that the CASTS algorithm respects the CSC with bursty input. The variation in the credits remaining is due to the flooring effect.

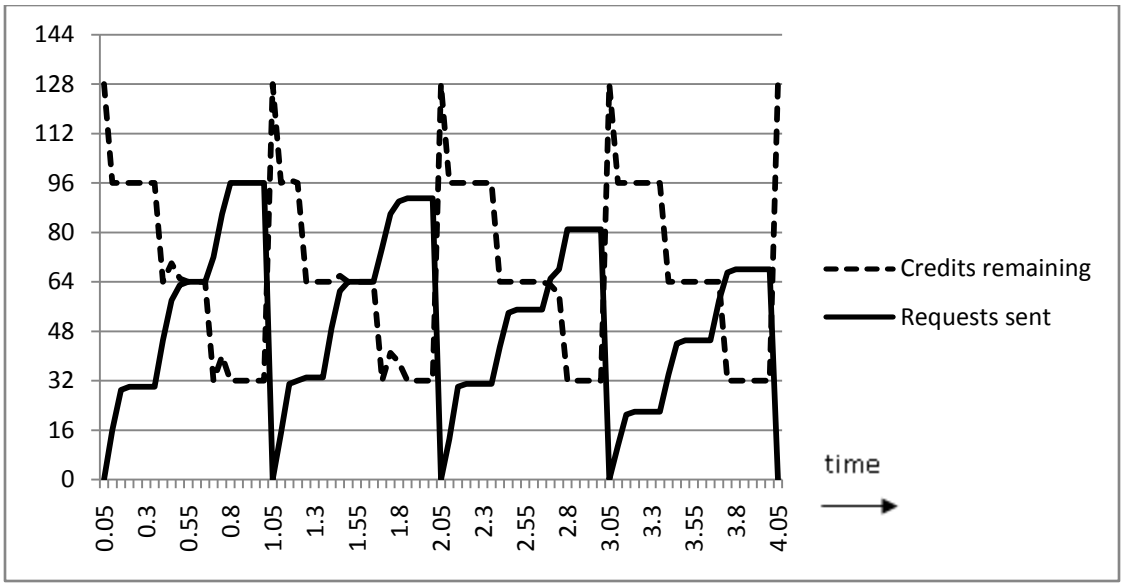



Figure 3.8: Variation of the credits remaining and requests sent to server with bursty input with $Y_{in} \sim X$

#### 3.5.2.1.3 Bursty input with $Y_{in} \gg X$

The next set of results shown in Figure 3.9 is produced to investigate behavior of CASTS algorithm in case of collective bursty input. The input is modeled with 3 bursts each of duration 0.1 seconds and 0.2 seconds apart; the arrivals at each burst are a Poisson process with rate $Y_{in} \gg X$. The results show that the CASTS algorithm respects the CSC with bursty input. The variation in the credits remaining is due to the flooring effect.

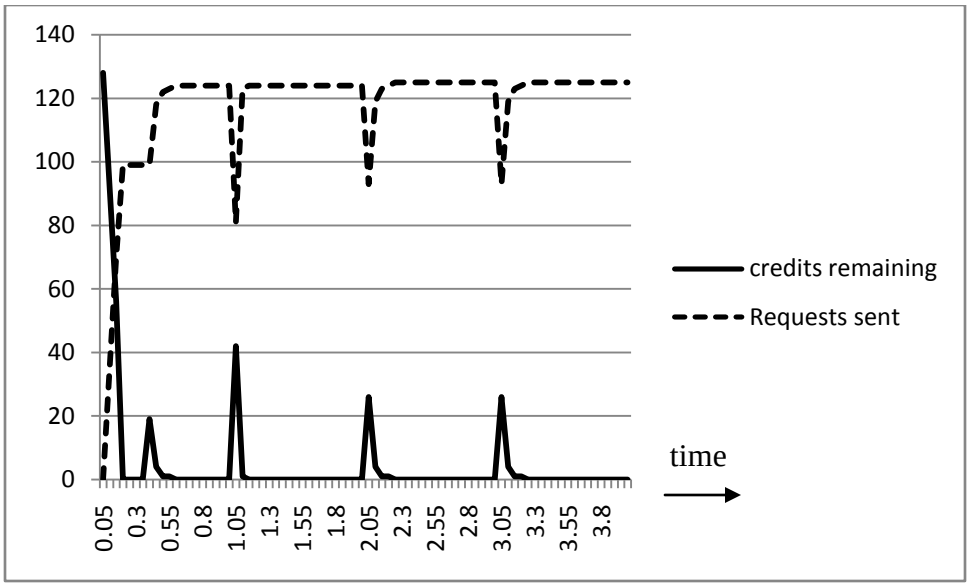



Figure 3.9: Variation of the credits remaining and requests sent to server with bursty input with $Y_{in} \gg X$

From the above results it can be concluded that the CASTS algorithm respects the CSC for all variations of the input rate.

#### 3.5.2.1.4 Queue size variation with CASTS

The below graph shows the variation in the number of empty queues at the end of subintervals for various input rates. As can be seen in Figure 3.10 for higher input rates, the number of empty queues is less and for lower input rates, the number of empty queues are higher as can be expected with a reactive algorithm like CASTS.

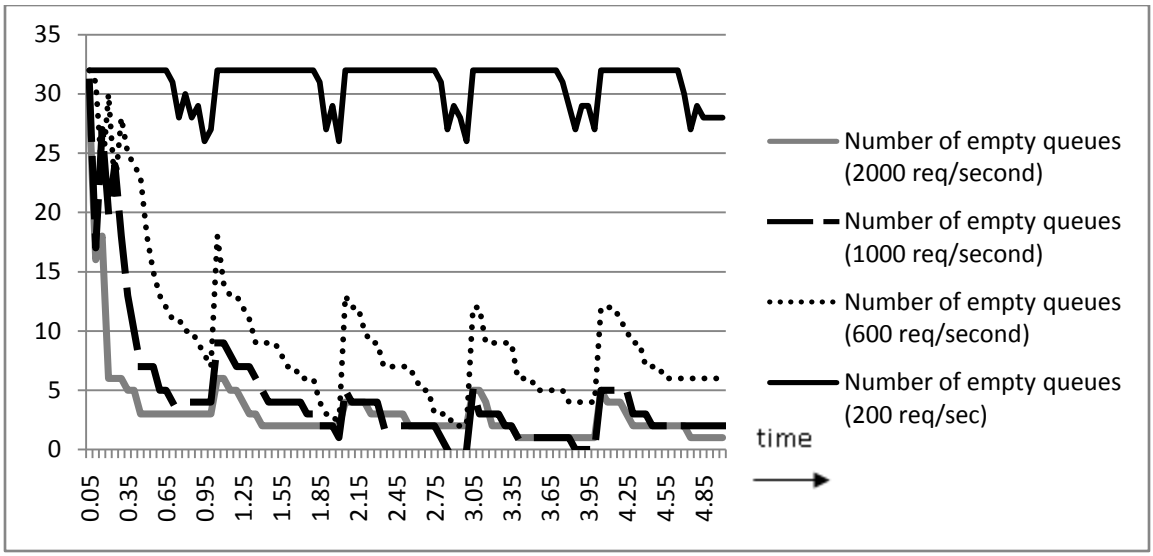



Figure 3.10: Variation of number of empty queues with time for different sub intervals when running CASTS.

### 3.5.2.2 Does the algorithm's claim of adapting to the current state of the system hold?

The main intent is a weighted distribution of the credits based on the input rates at each appliance relative to the input rates at other appliances. Each appliance at the initial state has the same credits, as the input rates vary at each appliance there is a transfer of credits from one appliance to the other, proportional to the ratio of the input rate at the appliance with respect to the total input rate.

As the results have to be gathered validating the transfer of credits, the simulator is run with 2 appliances and the credits are counted at each appliance at every subinterval; the input rate to appliance 1 is 4 times that of appliance 2 so that the transfer of credits from appliance 2 to appliance 1 occurs. Figure 3.11 shows the variation of the credits available at each of the appliances and the total number of requests sent to the server. The credits in both appliances are the same at initialization at the beginning of the observation interval; when the input rates vary, the credits are assigned to the appliance proportional to the ratio of the input rate at the

appliance to the total input rate. The credits of both the appliances are initialized to 64 each (X=128 and B=2) and as the input rate at appliance 1 is higher than that at appliance 2, during the subperiods, the credits at appliance 1 are more than those at appliance 2. This adaptation to the current state of the system continues for all observation intervals.

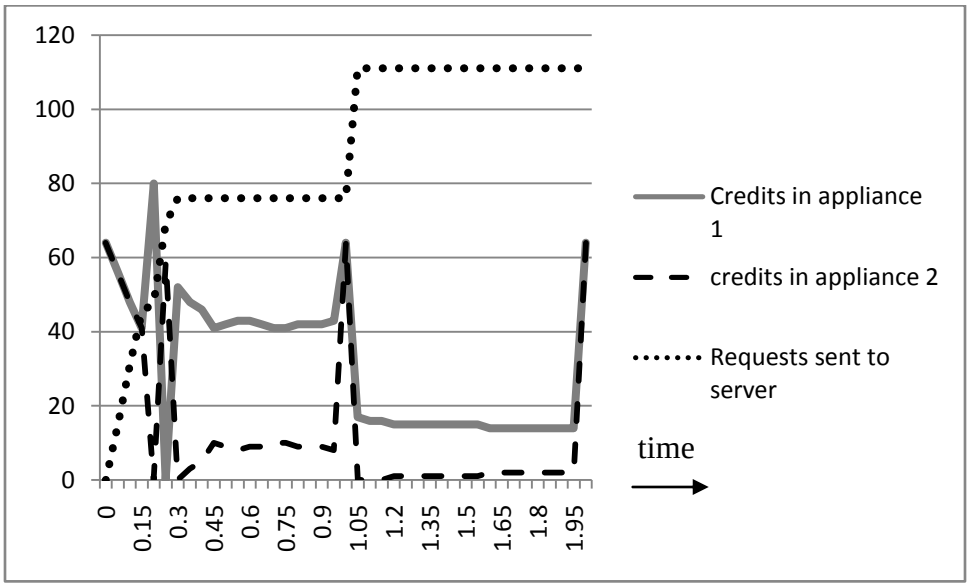


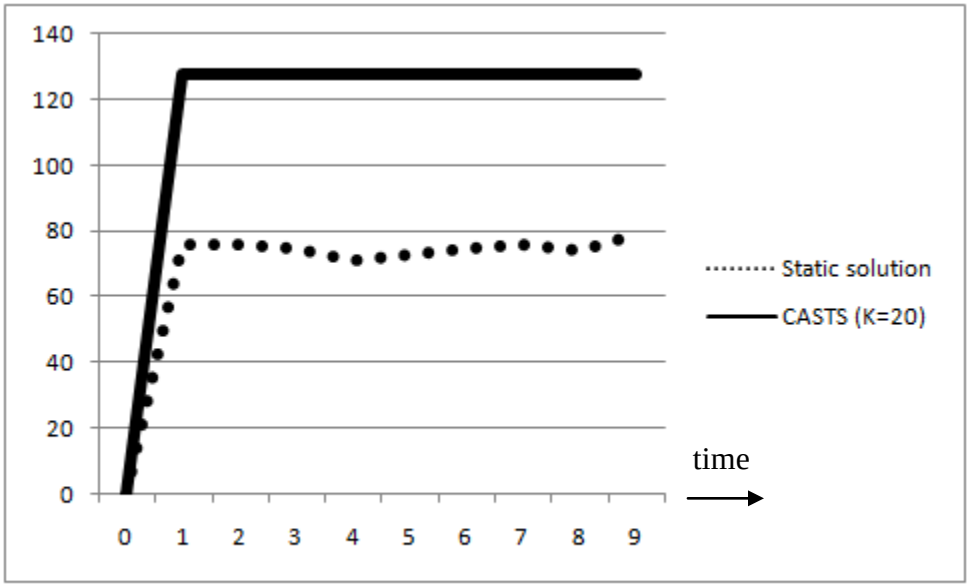

Figure 3.11: Variation of credits based on change in input rate at the appliance

**3.5.2.3 Is CASTS more responsive than the Manual Static Allocation (MSA) technique**?

The next result from the simulator demonstrates the improved responsiveness of the CASTS algorithm compared to MSA to changes in the state of the system.

The results verify the improved responsiveness of the algorithm with respect to the static solution where the credits are distributed equally among the appliances at the beginning of the observation interval and the credits are not modified based on the change in the input rate during the observation interval. The simulator is run with two appliances (B=2) with the static solution, with input rate at appliance 1 to be 4 times that of appliance 2. The simulator is next run with the same setup with CASTS where the credits at appliance 1 are adapted to

be high as the input rate is high. Figure 3.12 shows the number of requests sent to the server at the end of each observation interval in both cases. Both solutions respect the contract, however the number of requests sent to the server by the static solution is lower than the requests sent to server by the CASTS solution because the static solution does not adapt to variations to the input rate at of individual appliances where as the CASTS algorithm distributes credits to appliances based on the input rate at each appliance.




**X = 128, B=2, $Y_{in}$ = 300**

3.12: Requests sent to server in static solution vs. Requests sent to server in CASTS at the end of the observation interval

### 3.5.2.4 What are the parameters of importance to the CASTS algorithm and what is the effect of their variation on the behavior of the CASTS algorithm?

The parameters of importance in the CASTS algorithm are

1. The number of subperiods (K)

2. The observation interval (T)

3. Number of requests that can be sent to service host (X)

4. The number of middleware appliances (B).

The next sections investigate the variation of each of the parameters and describe the results from the simulators.

**3.5.2.4.1 The number of subperiods (K)**

The goal of defining K is to give more chances to the algorithm to react to changes in the behavior of the input conditions. Studying the impact of K on the behavior of the algorithm is of major importance, as the higher the number of subintervals, the higher the control overhead. The number of control messages exchanged is linearly proportional to both the number of appliances and the number of subintervals, O(BK). Depending on the length of the observation period and on the size of the documents, the control overhead might compromise the efficiency of the system.

Given below are results for varying K from 2 to 80 with the uniform input load of $Y_{in} >> X$ (with X =128, $Y_{in}$ = 2000) so that enough requests are present for variations to occur with higher values of K.

As can be seen, the algorithm converges sooner for higher values of K but for values of K>40 there is no added advantage, as the speed of convergence is not significantly higher.

K = 2

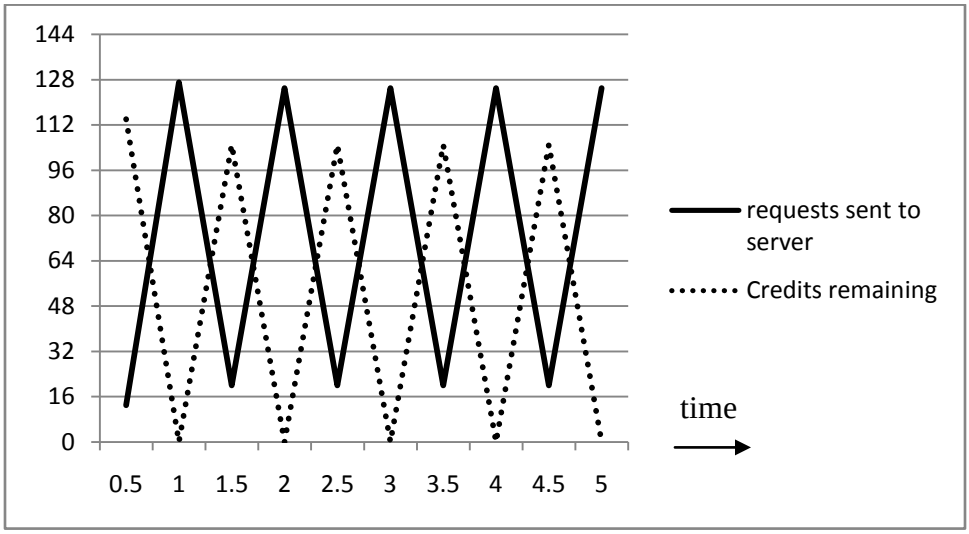



Figure 3.13: Variation of credits remaining and the requests sent to server with K = 2

K = 5

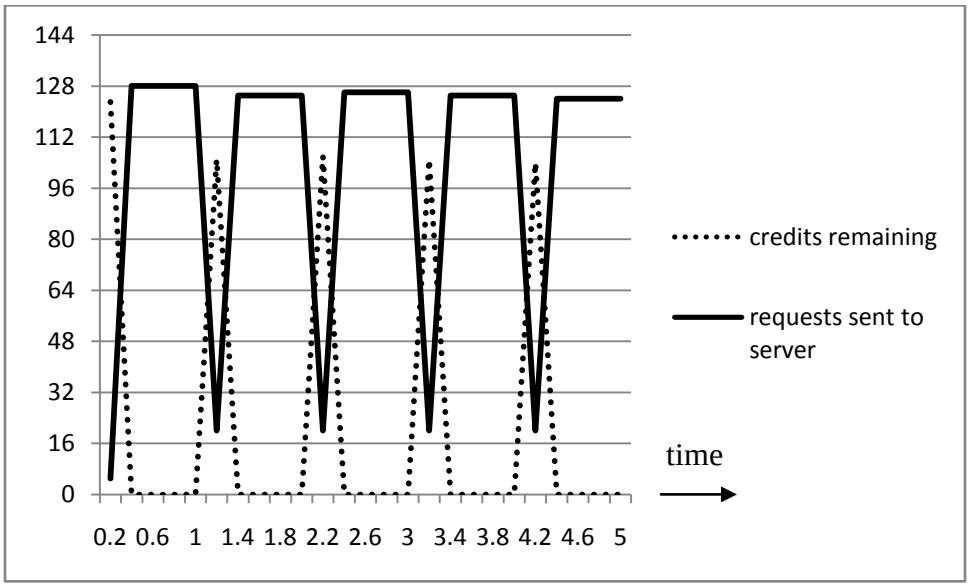



Figure 3.14: Variation of credits remaining and the requests sent to server with K = 5

K = 10

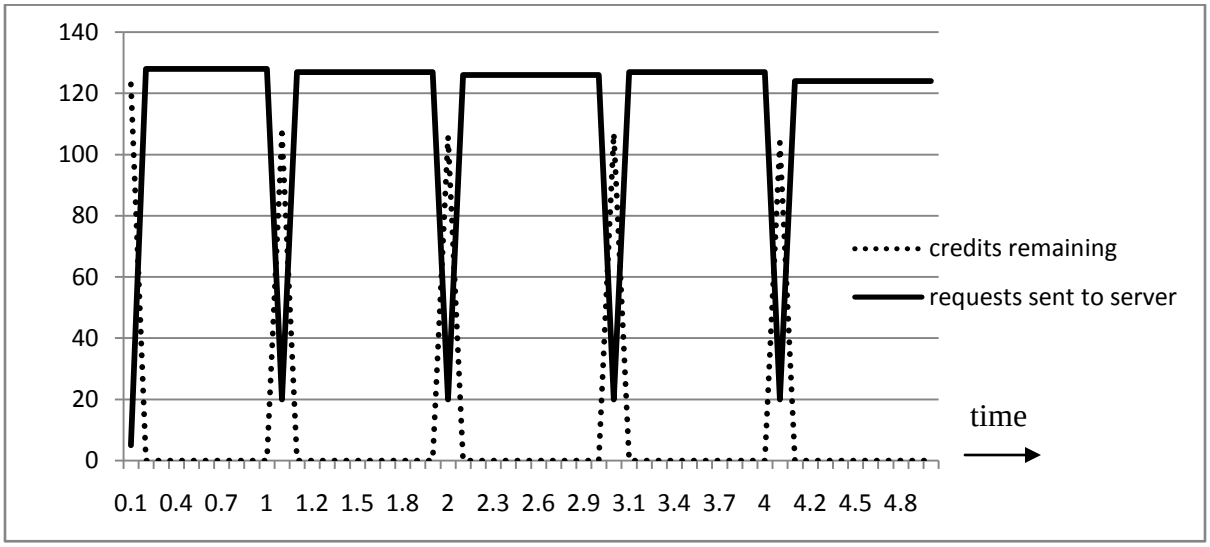



Figure 3.15: Variation of credits remaining and the requests sent to server with K = 10

K= 20

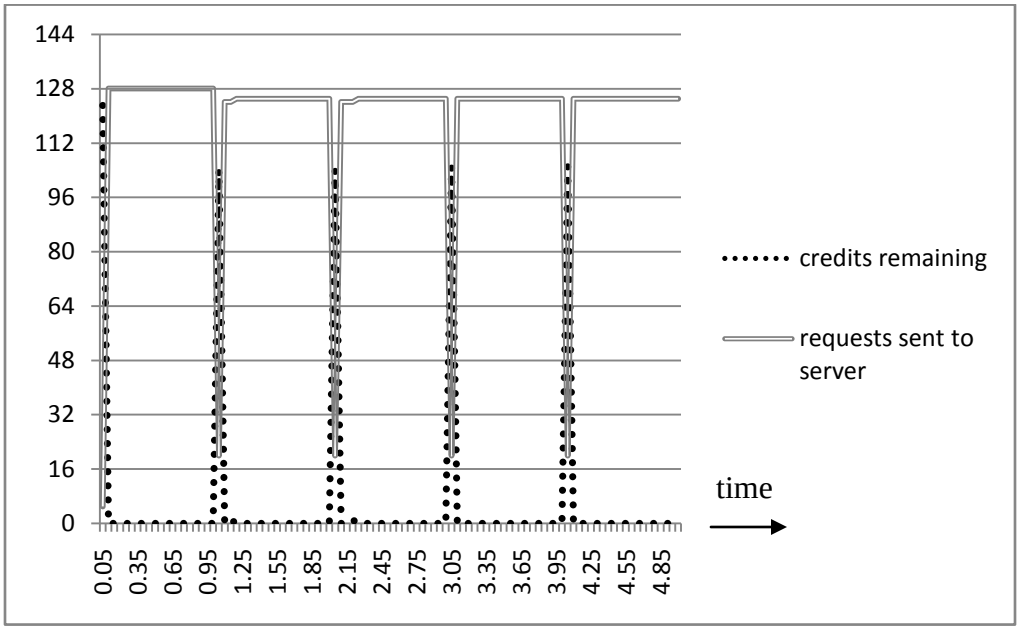



Figure 3.16: Variation of credits remaining and the requests sent to server with K = 20

K = 40

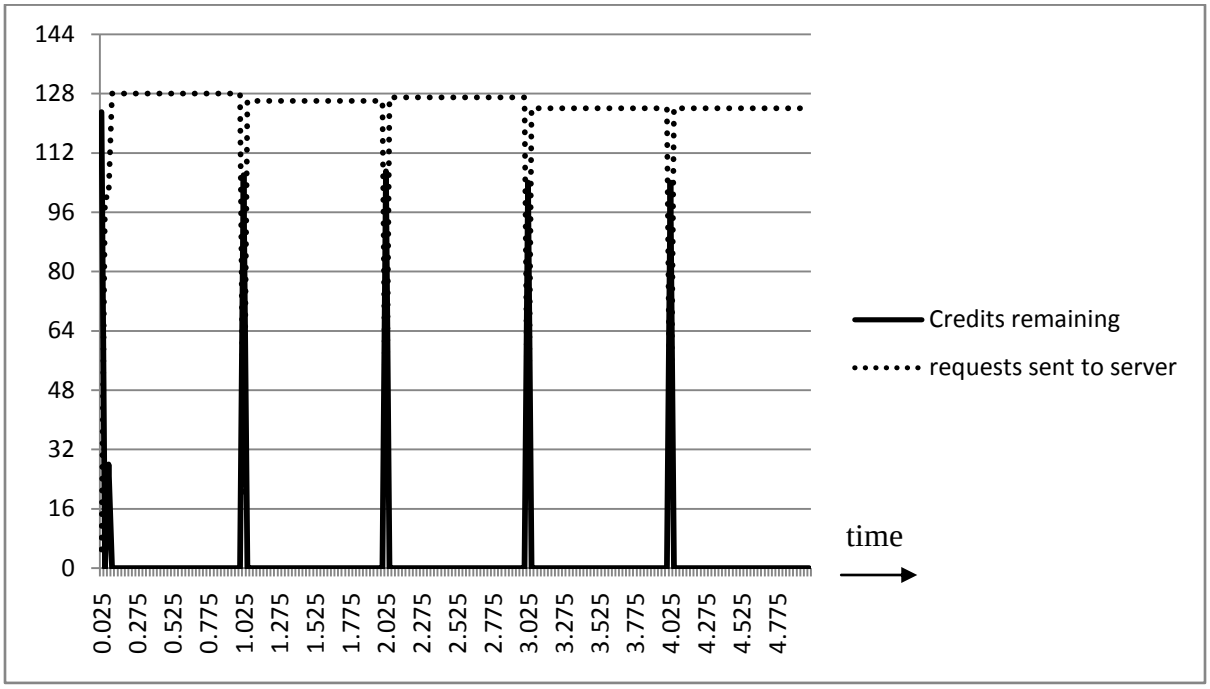



Figure 3.17: Variation of credits remaining and the requests sent to server with K = 40

K = 60

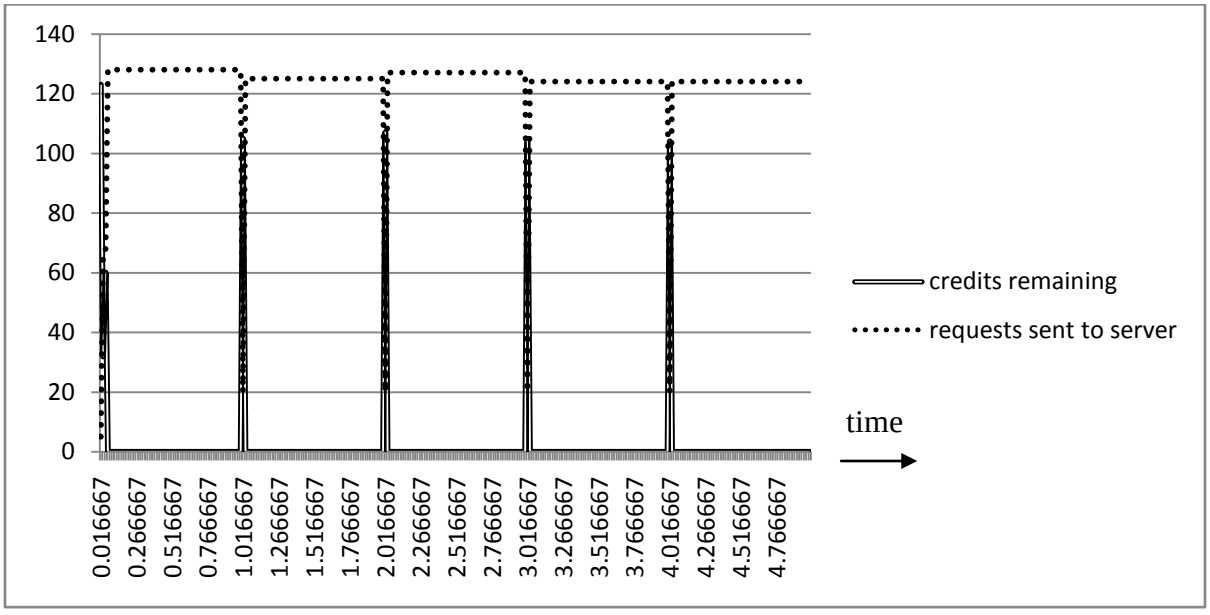



Figure 3.18: Variation of credits remaining and the requests sent to server with K = 60

K = 80

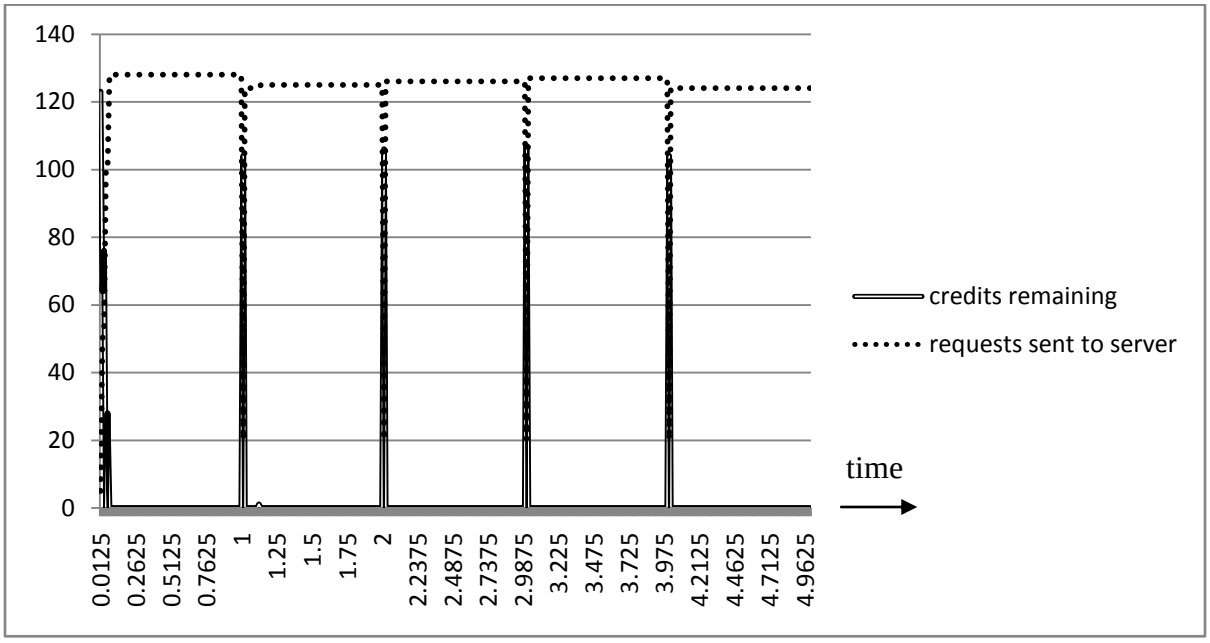



Figure 3.19: Variation of credits remaining and the requests sent to server with K = 80

#### 3.5.2.4.1.1 Average waiting time variation with number of subperiods K

As the value of K increases, the algorithm gets more number of changes to react to changes in the input behavior. This results in the average waiting time to decrease as the number of subperiods increase as shown in Figure 3.20. For this experiment the input number of requests was chosen to be much higher than X so that the queues are filled.

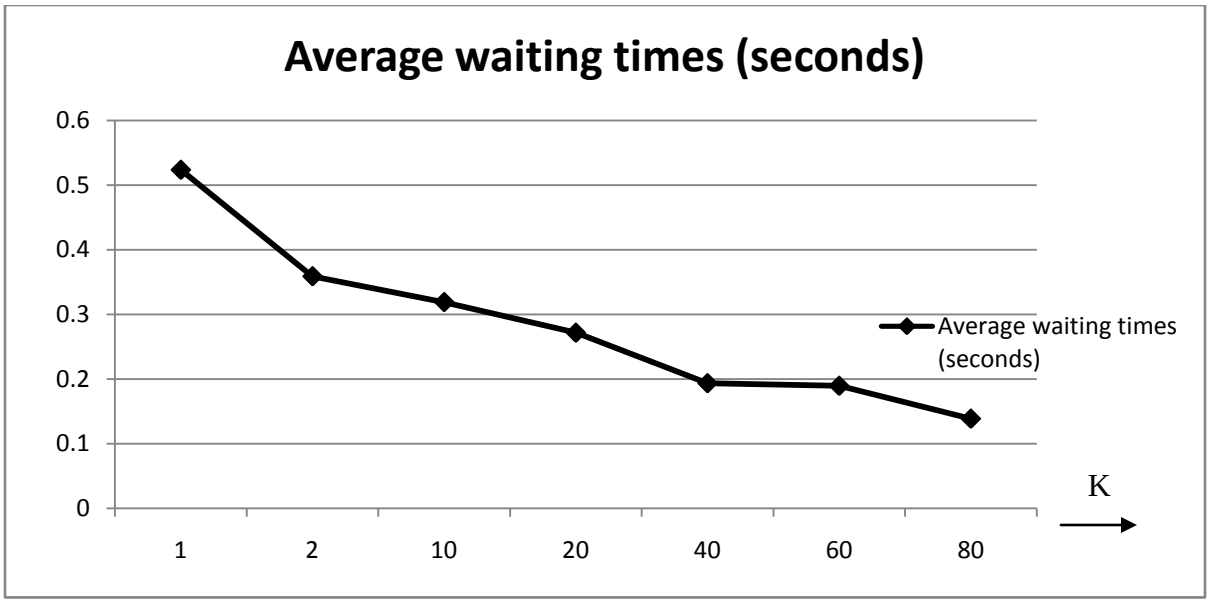



Figure 3.20: Variation of average waiting time with number of subperiods K

**3.5.2.4.2 The observation interval T, The number of requests that can be sent X, The number of appliances B.**

The observation interval T, the number of requests that is allowed to be sent X is decided by the IT administrator who sets the terms of the SAR. The duration of T decides the optimum value of K, the number of subperiods as larger values of T could support a large number of subperiods.

The scalability of the algorithm can be measured by varying B, the number of appliances. This is discussed in Section 4.2

### 3.5.3 Results from implementing the algorithm in real world appliances

For the operation of the algorithm in a real world setting, the most important question to be answered is "**how do we modify the algorithm to include relevant assumptions in a real world setting**"?

The most important assumption made was to consider the amount of time it takes for the appliances to exchange metrics is much less than T/K. The algorithm is modified to include the time it takes for the appliances to transfer their metrics to the peers. Two modes of operations where the algorithm reacts differently are specified.

a) Relaxed mode

b) Strict mode

#### 3.5.3.1 Relaxed Mode

At the end of each subperiod, appliances transfer their metrics ($r_i(k)$ and $Q_i(k)$) to their peers. Let $t_i$ (assumed less than T/K) be the amount of time it takes for appliance i to get metric updates from its peers after the end of the subperiod. In relaxed mode, during the time interval $t_i$, if credits are remaining in the appliance, and requests arrive at the appliance, they are sent to the service end-point until all credits expire. There is a timeout period set at 2×roundtrip delay before which the metric updates have to be received else the credits are calculated with only the metrics received. This is illustrated in Figure 3.21

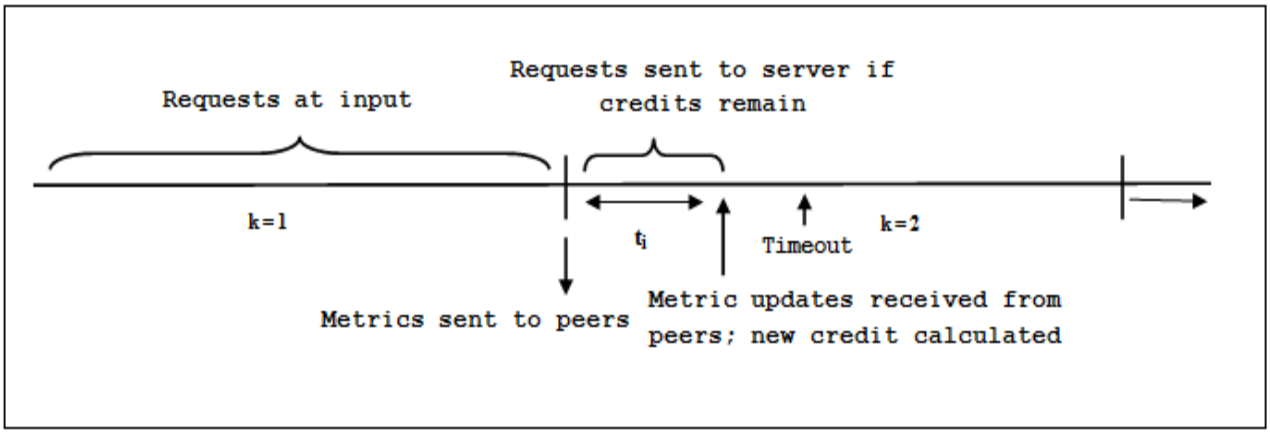



Figure 3.21: Relaxed mode operation (not to scale)

#### 3.5.3.1.1 Results of CASTS in Relaxed mode

The results were obtained by running CASTS on 2 commercial middleware appliances in a production enterprise network. With analysis about the time for transfer of metric updates between appliances and the time to process the updates and compute the credits at the appliance, K=6 was chosen, so subperiods were of length 10 seconds. The first run is with the **relaxed mode,** where the requests received between the time interval ($t_i$) when the subperiod ended and the metrics are received from the peers were sent to the server if credits were available in the appliance. The input rates at the appliances are in the ratio of 3:1. Figure 3.22 shows the requests sent to the server at the end of the observation interval for different input rates. From the results we find that there is a possibility of exceeding X in the relaxed mode.

For higher rates, the CSC is respected but for lower input rates, the number of requests sent to the server can exceed X. This is because, when the rate is high, the credit allotted to each appliance extinguishes soon in the subperiod and there are no credits remaining for requests which arrive during $t_i$; similarly, when the rate is low, there are credits in the appliance for

requests that arrive during $t_i$. Figure 3.23 shows the difference of requests sent to server at the end of the subinterval + $t_i$ and the requests sent to the server at the end of the subinterval (queue size reported to the peers) for appliance 1 for different input rates. This is the "error" induced to the appliance while calculating the credits. For example, in Figure 3.23, at end of subinterval 1 for input rate 111 requests/second, appliance 1 sends up to 849 requests to the server during $t_i$ and so the number of credits reported to appliance 2 is 849 less than the actual credits, thus inducing a discrepancy in the calculation of credits for the next interval and so resulting in number of requests sent to server being greater than X. It is noticed in Figure 3.23 that there is an increase in the difference in the requests sent to the server at the subintervals and that reported to its peers for lower input rates as explained before.

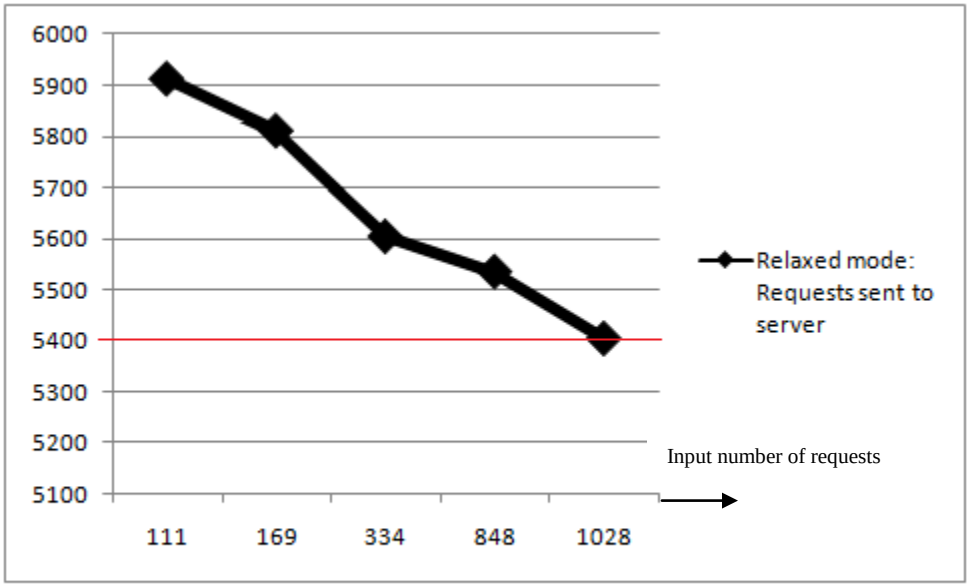



Figure 3.22: Requests sent to server at the end of the first observation interval vs. total input rate in relaxed mode.

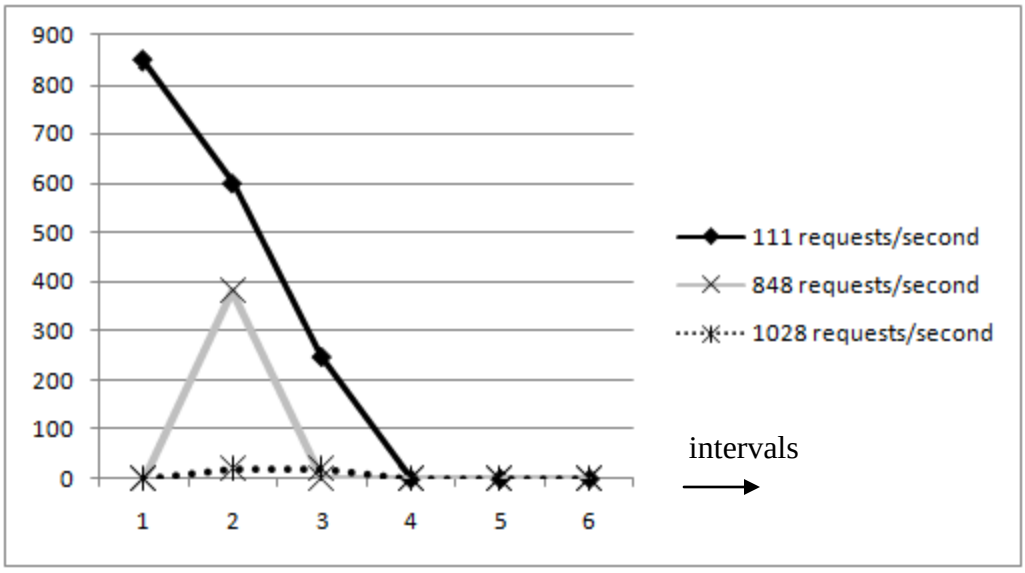



Figure 3.23: The difference of requests sent to service host by appliance 1.

### 3.5.3.2 Strict Mode

At the end of each subperiod, appliances transfer their metrics ($r_i(k)$ and $Q_i(k)$) to their peers. Let $t_i$ (assumed less than T/K) be the amount of time it takes for appliance i to get metric updates from its peers after the subperiod.

In strict mode, during the time interval $t_i$, if credits are remaining in the appliance and requests arrive at the appliance, they are not sent to the service end-point. There is a timeout period set at 2×roundtrip delay before which the metric updates have to be received else the credits are set to 0 for the next interval.

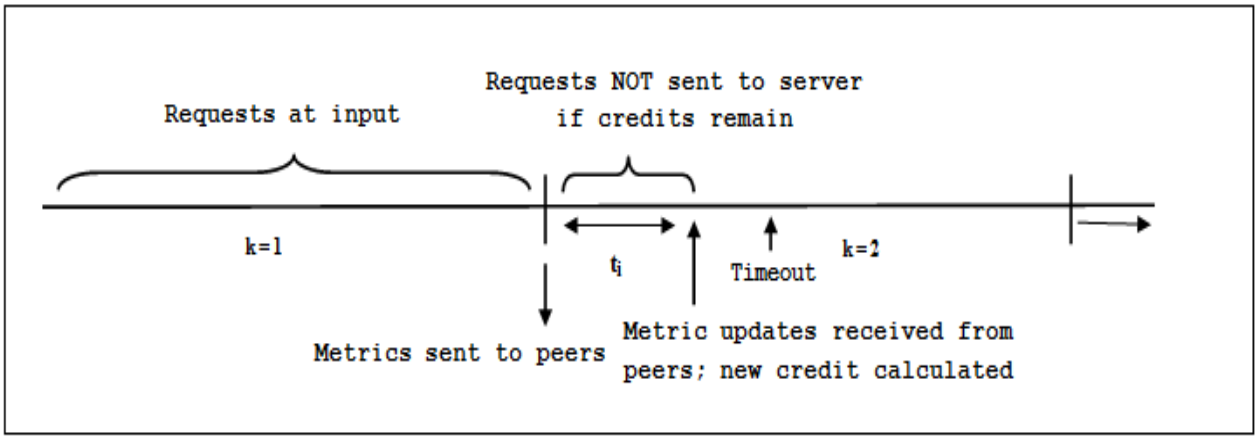



Figure 3.24: Strict mode operation (not to scale)

#### 3.5.3.2.1 Results from CASTS in Strict mode

The results were obtained by running CASTS on 2 peer commercial middleware appliances in a production enterprise network. With analysis about the time for transfer of metric updates between appliances and the time to process the updates and compute the credits at the appliance, K=6 was chosen, so subperiods were of length 10 seconds. The strict mode is conservative, in the sense that the requests received between the time interval when the subperiod ended and the metrics are received from the peers ($t_i$) are **not sent** to the server if credits were available in the appliance. Figure 3.25 shows the variation of the number of requests sent to the server at the end of the observation interval to the input rate. We note that the strict mode respects the CSC for all input rates, i.e., the input rate does not exceed X. There is a slight deficit (1-4 requests) in the requests sent to the server in strict mode. This is due to the flooring effect described earlier. The difference of requests sent to the server at (the end of the subinterval + $t_i$) and the requests sent to the server at the end of the subinterval

(queue size reported to the peers) for all appliances stays at 0 for all input rates in strict mode.

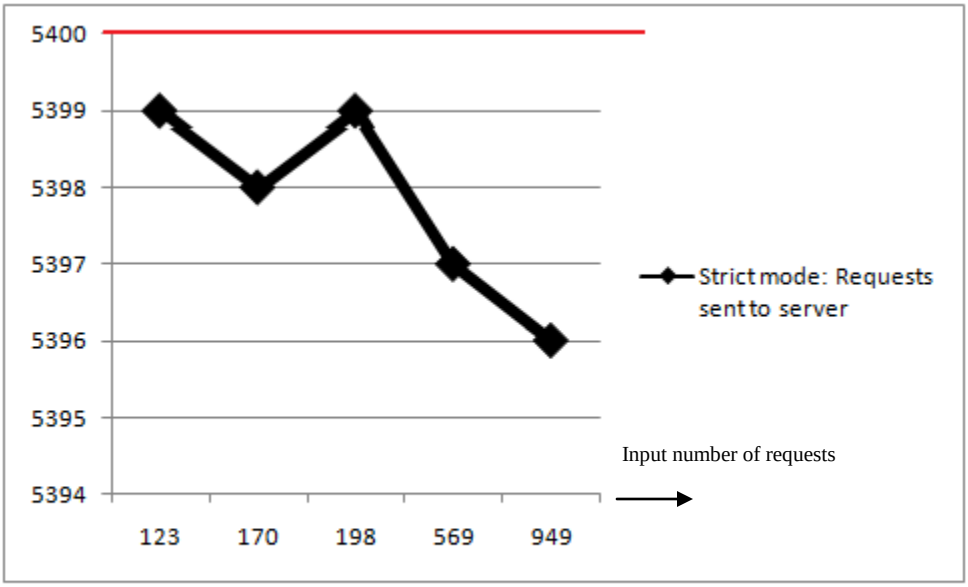



Figure 3.25: Requests sent to server at the end of the first observation interval vs. total input rate in strict mode.

### 3.5.3.3 Strict mode vs. Relaxed mode

The strict mode respects the CSC but is also conservative where there is a possibility that the number of requests sent to the server is less than X. In the relaxed mode there is a possibility of exceeding X but the average service delay will be less than the strict mode for the same input rate. The choice of the conservative strict mode or the relaxed mode is a decision based on the strength of the CSC, environmental factors, etc. which have to be taken into account by the IT administrator.

# CHAPTER 4

# Conclusion and Future Work

In this thesis we have investigated the multi-point to single-point service traffic shaping requirement in service oriented networks.

### 4.1 Summary

We proposed CASTS, a reactive credit-based algorithm for the service traffic shaping problem in service-oriented networking. CASTS is a decentralized algorithm that distributes credits to multiple middleware appliances to collectively respect a global client service contract (X, T) imposed by a service instance.

The main contributions of our work are:

1) We explain the problems with the static credit allocation and identify the need for a dynamic, measurement-based allocation scheme.

2) We implement a reactive algorithm that enables each appliance to dynamically react to the changes in the global state of the system.

3) We perform a number of experiments in a real middleware appliance and show that our approach does respect the contract even under the constraints of a realistic situation.

We developed a simulator to validate the correctness and the responsiveness of CASTS. Our results find that CASTS respects the CSC and improves the responsiveness to the variations to the input rate. We also identified a problem due to approximations in the CASTS algorithm. This can be observed both in simulations as well as the experiments we ran on real appliances. The detailed study of the flooring effect is part of future work. We proposed two

variations of the algorithm, suitable for operation in realistic environments (Relaxed and Strict modes of operation).We implemented these variations in a production level enterprise network and analyzed the behavior of the algorithm in both modes.

## 4.2 Future Work

In this section we propose a number of future areas of research in the multi-point to single-point service traffic shaping paradigm.

### 4.2.1 Further investigation/modifications to the Relaxed and Strict Modes

In proposing the two modes of operation of the algorithm in a production level environment we have not considered the variations in the updates received from the peer appliances. The updates from the peers can be missing in one or more intervals, updates from previous intervals can be received. Further investigation is required in refining the operations of the modes to include all the factors in a real enterprise network.

### 4.2.2 Scalability of CASTS

We have not investigated the scalability of the CASTS algorithm. Enterprise networks will need to have metrics for the scalability of the CASTS algorithm for optimum performance.

### 4.2.3 Modifications of CASTS to prevent the flooring effect

We need to investigate the modifications of the algorithm to prevent/reduce the flooring effect so that the credits may not be wasted.

### 4.2.4 Investigations on the modifications of the Service Access Requirement

One of the reasons for the flooring effect is the absence of an allowed variance in the Service Access Requirement (SAR). If the SAR would incorporate a small window beyond the allowed rate then the flooring effect would be minimized. For example, if the SAR would

state in the observation interval T seconds, if X requests were allowed, a tolerance of X/10 requests was allowed to escape to the service end point as in the ATM traffic contract which allows a CDVT (Cell Delay Variation Tolerance) (Aboul-Magd, O., & Jamoussi, B. May 2001) then requests wasted due to flooring effect will be less. Future work involves analyzing and quantifying this modification to the SAR.

# Bibliography


Aboul-Magd, O., & Jamoussi, B. (May 2001). QoS and service interworking using constraint route label distribution protocol. *IEEE Communications* .

Aboul-Magd, O., & Jamoussi, B. (May 2001). QoS and service interworking using constraint route label distribution protocol.

Adobe. (n.d.). *Adobe AMF*. Retrieved from Adobe Livedocs: http://livedocs.adobe.com/livecycle/es/sdkHelp/programmer/lcds/wwhelp/wwhimpl/common/html/wwhelp.htm?context=LiveDocs_Parts&file=messaging_framework_4.html

Boloor, K., Dias De Amorim, M., Callaway, B., Rodriguez, A., & Viniotis, Y. (December 2008). Meeting Service Requirements in SOA. *2nd IEEE Workshop on Enabling the Future Service Oriented Internet.*

Callaway, R., Devetsikiotis, M., Viniotis, Y., & Rodriguez, A. (May 2008). An Autonomic Service Delivery Platform for Service-Oriented Network Environments. *ICC '08. IEEE International Conference on Communications, 2008.*

Castagna, G., Gesbert, N., & Padovani, L. (January 2008 ). A theory of contracts for web services. *35th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages.*

Cuomo, J. (June 2005). IBM SOA "on the edge". *Proceedings of the 2005 ACM SIGMOD international conference on Management of data.*

Edelstein, H. (1994). Unraveling Client/Server Architecture. *DBMS* .

Elwalid, A., & Mitra, D. (Apr 1997). Traffic shaping at a network node: Theory, optimum design, admission control. *IEEE Infocom.* Kobe, Japan.

Erl, T. (2008). SOA Principles of Service Design. *The Prentice Hall Service-Oriented Computing Series* .

Fielding, R., & Taylor, R. (May 2002). Principled design of the modern Web architecture. *Transactions on Internet Technology (TOIT)* .

Foster, H. (January 2006). *A Rigorous Approach to Engineering Web Service Compositions.*

Glass, G. (n.d.). *The Web services (r)evolution: Part 1*. Retrieved from IBM developerWorks: http://www.ibm.com/developerworks/webservices/library/ws-peer1.html

Kumar, A., Neogi, A., & Pragallapati, S. (July 2007). Raising Programming Abstraction from Objects to Services. *IEEE International Conference on Web services, 2007*, (pp. 864 - 872 ).

Liu, C., Li, Z., Chen, J., Lin, X., & Ma, D. (Aug 2007). A Service Level Agreement-based Web Services Performance. *16th international conference on Computer Communications and Networks.*

Maurizio, A., Sager, J., Corbitt, G., & Girolami, L. (January 2008). Service Oriented Architecture: Challenges for Business and Academia. *41st Annual Hawaii International Conference on System Sciences.*

Maximilien, E. M., & Singh, M. P. ( July 2005.). Towards Web Services Interactions Styles., (p. 2nd IEEE International Conference on Services Computing (SCC)). Orlando,.

Microsoft. (n.d.). *How RPC works?* Retrieved from Networking Developer Platform Center: http://msdn.microsoft.com/en-us/library/aa373935.aspx

Microsoft. (n.d.). *Microsoft .Net Remoting*. Retrieved from http://msdn.microsoft.com/en-us/library/ms973857.aspx

Microsoft. (n.d.). *Microsoft Passport*. Retrieved from https://accountservices.passport.net/ppnetworkhome.srf?vv=650&lc=1033

Muthusamy, V., & Jacobsen, H.-A. (December 2008). SLA-driven distributed application development. *MW4SOC 08: 3rd workshop on Middleware for service oriented computing* .

N. Huhns, M., & Singh, M. P. (January 2005). Service-Oriented Computing:Key Concepts and Principles. *IEEE Internet Computing* , volume 9, number 1, pages 75–81.

OASIS. (n.d.). *UDDI*. Retrieved from OASIS: http://www.uddi.org/pubs/uddi_v3.htm

Object Management Group. (n.d.). *CORBA Home Page*. Retrieved from CORBA: http://www.corba.org/

Papazoglou, M. P., & Heuvel, W.-J. (July 2007). Service oriented architectures: approaches, technologies and research issues. *The VLDB Journal - The International Journal on Very Large Data Bases, Volume 16 Issue 3* .

Parekh, A. K., & Gallager, R. G. (June 1993). A generalized processor sharing approach to flow control in integrated services networks – the single node case. *IEEE/ACM Transactions on Networking vol.1* .

Paxson, V., & Floyd, S. (June 1995). Wide Area Traffic: The Failure of Poisson Modeling. *IEEE/ACM Transactions on Networking* , vol. 3, no. 3, pp. 226-244,.

Rexford, J., Bonomi, F., Greenberg, A., & Wong, A. (June 1997). Scalable architectures for integrated traffic shaping and link scheduling in high-speed ATM switches. *IEEE Journal on Selected Areas in Communications* .

Schuller, S. (n.d.). *SaaSBlogs*. Retrieved from http://www.saasblogs.com/2006/09/26/scale-as-a-commodity-2/

Singh, M. P., & Huhns, M. N. (2005). Computing with Services. In M. P. Singh, & M. N. Huhns, *Service-Oriented Computing.* John Wiley and Sons Ltd.

Sun Microsystems. (n.d.). *Remote Method Invocation*. Retrieved from Sun Developer Network: http://java.sun.com/javase/technologies/core/basic/rmi/index.jsp

UserLand Software. (n.d.). *XML RPC*. Retrieved from http://www.xmlrpc.com/

w3c. (n.d.). *SOAP*. Retrieved from http://www.w3.org/TR/soap/

w3c. (n.d.). *Web services*. Retrieved from http://www.w3.org/2002/ws/

w3c. (n.d.). *WSDL*. Retrieved from http://www.w3.org/TR/wsdl

w3c. (n.d.). *XPath*. Retrieved from http://www.w3.org/TR/xpath

w3c. (n.d.). *XSLT*. Retrieved from http://www.w3.org/TR/xslt

Zeus Technology Limited. (n.d.). *Service Oriented Networking*. Retrieved from http://www.zeus.com/documents/en/Bu/Building_a_Service_Oriented_Network.pdf