Autonomic Resource Management for Extensible Control Planes

Bushar Yousef, Doan Hoang
University of Technology, Sydney
{byousef,dhoang}@it.uts.edu.au

Glynn Rogers
CSIRO ICT Centre
Glynn.Rogers@csiro.au

Abstract

A dynamically extensible control plane is a key enabling feature of next generation intelligent self-configuring networks. This extensibility can be achieved by enabling service deployment into the control plane of a network. These services consume unpredictable amounts of resources at node with unknown resource availability. This paper presents an autonomic resource management model for extensible control plane, called C-QoS. Under the C-QoS model, services are classified into classes according to their importance or QoS requirements. Resources are allocated among classes according to a differentiated per-class resource allocation scheme, while Services within a class receive fair treatment. Due to the difficulty of determining resource availability in heterogenous infrastructure or service resource requirements, this scheme is dynamically adaptive to each resource according to its demand patterns.

1. Introduction

An 'autonomic network' requires evolving intelligence to achieve self-configuring and self-healing characteristics. This intelligence would monitor and automatically reconfigure the network to provide self resource-management, fault isolation and recovery, and to provide new network services that are best suited to the required connectivity. Automatic mechanisms would upgrade the intelligence to evolve the network under changing conditions. However, current networks accommodate static infrastructure where functionality cannot be extended to enable the introduction of new intelligence into the network.

Various research efforts have been made to enable intelligence to be introduced into networks. Programmable Networks utilise programmable forwarding plane components (such as Programmable Network Processors, Programmable ASIC, or FPGA) to enable third parties to dynamically deploy services into the control plane of network nodes. A Service is a collection of software components that executes in the control plane to perform a specific role by monitoring and reconfiguring forwarding-plane hardware components. Such a dynamic service deployment feature enables an extensible control plane that accommodates 'intelligent' monitoring and action components of an autonomic network.

Despite these added benefits, deploying a Programmable Network introduces new management issues. Services executing in the control plane require resources to function, namely, CPU, memory, and internal communication resources. These resources are limited at network nodes and must be managed among competing services. To date, no practical model exists that allows multiple Services to execute with adequate Quality of Service (QoS). We identify two models employed in the resource management mechanisms of all current programmable network nodes:

Explicit allocation, which is employed in [1, 2], refers to a model where resources are allocated to a domain (a collection of Services or an activated control flow) according to a pre-determined fixed specification. Systems adopting this model require the specification of resource requirements for each domain when executed on each different type of control plane platform. This is not scalable to large heterogeneous infrastructure. This model leads to inefficient resource utilisation as domains are over allocated resources, and unutilised resources are not reallocated among other domains. More critically, however, this model hinges on the difficult task of predetermining or estimating the resource consumption of domains and therefore cannot be utilised for evolving intelligence.

Flow throttling, which is employed in [3, 4], refers to a model where the rate of packet input to Services is throttled to control their resource consumption. Such a model cannot control resources consumed without packet input influence such as by monitoring Services. This model also hinges on the difficult task of predetermining or estimating resource consumption.

We identify a need for a simple, comprehensive, and scalable resource management approach for the extensible control planes of future networks. Due to the heterogeneity and size of networks, the complex nature Service deployment, any viable solution must adopt an autonomic approach. Such a solution would automatically adapt to varying resource availability and consumption where the interaction with the administrator is limited to simple high-level classifications.

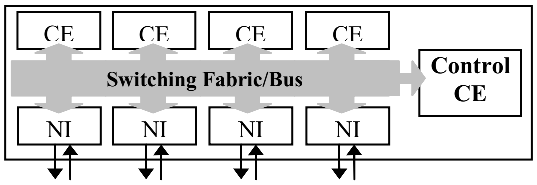

Figure 1 Serviter Hardware Base

This paper proposes a new scalable, adaptive, and self-managing resource management model for extensible control-planes, called Control plane–Quality of Service (C-QoS). C-QoS automatically customises resource allocations according to each Service's heterogeneous needs with minimal human interaction and no internal detail of Service code. C-QoS adapts to any platform regardless of its manufacturer, resource capacity, or internal topology using dynamic allocation ratios to provide differentiated treatment for Services according to their role in the network. It is also capable of guaranteeing a lower limit of resource availability to each Service.

The rest of the paper is structured as follows. Section 2 presents the underlying architecture for an extensible control plane. Section 3 introduces the C-QoS model, and describes its components. The paper concludes in section 4.

2. The Extensible Control Plane

In previous work [5, 6], we designed and implemented a service-oriented programmable network platform called Serviter (previously called SXD-PNP). Serviter hosts a scalable, extensible control plane where Services are deployed on-demand. Serviter also employs various security mechanisms to isolate Service risks to separate and secure partitions.

The Serviter structure is depicted in figure 1. A Serviter is composed of an expandable control plane composed of Computational Elements (CEs), a forwarding plane composed of programmable Network Interface modules (NIs), and a Control-Computational Element.

Services are hosted in a single CE or across multiple CEs to achieve a scalable extensible control plane. Services are modules written by trusted third parties that are distributed using a specific deployment structure. Services typically monitor and reconfigure programmable forwarding plane modules, and may capture control traffic for customised processing. A CE, depicted in figure 2, is composed of Partition Virtual Machines, a number of communication dispatchers, an interface to the OS resource management mechanisms, and a Load Manager.

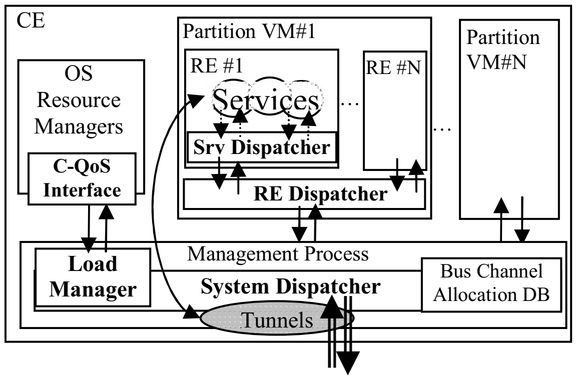

Figure 2 Computational Element

A Partition VM is a coarse management unit used in resource management and load balancing. A Partition VM hosts a number of Runtime Environments (REs) which execute and manage collections of Services. Each RE manages a number of threads and schedules Services onto thread slots. Serviter provides three RE types, each with a different scheduling model to cater for variations in Service response time and allocation size requirements. Forwarding plane configurations of a certain Partition VM are restricted to the flows that are relevant for the Services it hosts. For example, a partition that hosts Services that manage web services may only redirect HTTP traffic from certain servers to SSL accelerators.

The selection of an appropriate Partition VM and RE configuration for Service place must take into account a number of factors. Firstly, Services requiring access to the different flows should be placed in different partitions to minimise the permitted range of forwarding plane access to each partition and isolate security risks to minimal set of flows. Secondly, Services requiring different QoS on internal resources should be placed in the different partitions. Finally, Services are then placed in an RE with the appropriate thread scheduling algorithm to meet response time and allocation size requirements.

Services communicate through a number of dispatchers organised in a hierarchy as depicted in figure 3. The OS ensures that Partition VMs only communicate through the dispatcher hierarchy. However, an exception to this rule can be made for captured traffic as traversing the hierarchy would add delay. Tunnels can be created by Services to bypass dispatchers within the CE and reduce delay.

A Service executes unknown code that consumes unpredictable amounts of control plane computational resources, namely CPU, Memory, and I/O resources. Service must often perform intensive communication tasks. These tasks include interacting with other Services within the same CE or at a different CE. However, the most important Service communication tasks are NI monitoring, reconfiguring, and flow capture tasks. Such tasks consume internal node communication resources for storing, routing, verifying, and forwarding messages - namely CPU, memory, and switching fabric bandwidth. These resources are consumed outside of the Service in the Serviter dispatcher hierarchy or by the tunnels.

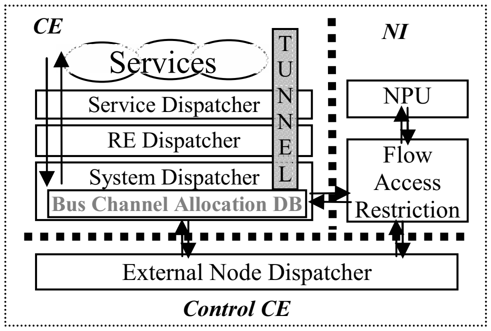

Figure 3 Dispatcher Hierarchy

C-QoS was developed and is currently employed to manage all internal resources of a Serviter among its competing partitions and Services even when Service internal details or node capacity are not known. The model and mechanisms of C-QoS could easily be adapted to manage the resources of any extensible control plane.

3. Control plane-QoS

C-QoS is a scalable, adaptive, and self-managing model for the division, allocation, and quality assurance of control plane and bus resources among competing Services and partitions. It is also used to optimise the utilisation of control plane resources.

Unlike the flow throttling models of [3, 4], C-QoS is not coupled with network QoS. This decoupling allows the management of control plane and bus resources without any impact on network SLAs. C-QoS employs a class based differentiation resource allocation model. This model does not involve fine grained control or explicit resources allocation essential to the resource management models of [1, 2].

Partitions, Runtime Environments, and Services are classified into a defined set of classes according to their importance and QoS requirements. Classes can be created on-demand, using only a class name and a simple class allocation ratio. A class allocation ratio defines the priority of resource allocation for each class (e.g. for Gold:Silver:Bronze:BestEffort classes, a ratio could be 20:10:5:0). Units within the same class are treated fairly.

Due to the unpredictable nature of resource consumption patterns and availability, resources are shifted from lower classes to a higher one as the load of the higher-class increases. This is achieved through three different allocation ratios: a fluctuating *current ratio*, an adaptable *limit ratio*, and a fixed *ideal ratio*. A separate *current ratio* is maintained for each resource type (e.g. CPU, memory, internal communication), while the *limit* and *ideal* ratios always apply to all resources.

The *current ratio* of each resource starts as being the same as the *ideal ratio*. Once a Load Monitor, located in CEs as depicted in figure 2, detects an increase in the load of higher classes for a certain resource, it gradually changes the *current ratio* of the resource to favour the higher classes. This process stops once the *limit ratio* is reached to guarantee an allocation level of resources to all classes. As the load of the upper class decreases, it periodically changes the *current ratio* to favour lower classes until the *ideal ratio* is reached.

To provide a fair and consistent level of to all partitions, *current ratios* must be maintained across all CEs. It is not possible to predetermine a distribution of Partition VMs that give uniform *current ratios* as: (1) it is not possible to predetermine the resource needs of Services along their lifetime (2) the available resources of each CE cannot be easily discovered (3) partitions are dynamically created and removed.

C-QoS provides mechanisms to ensure a uniform *current ratio* across CEs and to ensure CEs are not congested. The Control-CE contains a Load Balancing Service (LBS) that periodically polls each CE's Load Manager for resource availability and its *current ratio*. If a CE is congested, Best-Effort partitions are gradually paused and relocated to the least loaded CE. This relocation is restricted to Best-Effort partitions as it causes temporary Service outage. The LBS also ensures the uniform ratio is maintained by deploying a new partition at the CE with the least load for the class of the new partition. It is also used to verify that new partitions and/or Services can be accommodated without depleting resources.

The only human input required by C-QoS is class names and the *ideal* ratio. The *ideal* ratio can be derived directly from the relative importance of Services. The *ideal* ratio, which operates in a non-congestion state, does not require optimisation as any resources unutilised by a class are shared among others according to the *current* ratio of each resource.

The *limit* ratio must be adapted and dynamically optimised for each heterogenous CE to ensure resources are fully utilised. C-QoS uses a *Congestion Optimisation* algorithm. The basic concept behind this algorithm is that a good operating point is maintained by restricting the fluctuations of *current* ratios to the maximum range that does not cause congestion. To maintain differentiation, a separate fluctuation range must be maintained for each class in proportion to the *ideal* ratio. In other words, higher classes are permitted larger fluctuation ranges than lower classes.

The *Congestion Optimisation* algorithm is invoked when a Load Manager has detected that the *limit* ratio has been reached by a non-best effort class and has already attempted load balancing operations to free resources but failed. This algorithm polls the Load Manager to check if the CE is 'near' congestion (true if the utilisation of any resource reaching 95% in the past 'Time Frame'). If the CE is near congestion then the algorithm terminates as any limit increase causes congestion. Otherwise, it determines whether to increase the *limit* ratio for the congested class. This decision is made based on whether its current fluctuation range (the difference between ideal and limit ratio) in less that the permitted range (i.e. < class_ideal_ratio x total_fluctuation_ranges). If it is, then the limit is increased for the congested class in proportion to the mean of its current ratios (which gives an indication of the *ideal* ratio and demand). This gradual increase continually finds a good operating point where higher classes invoke Congestion Optimisation less frequently than lower classes.

A Resource Manager is provided for each resource. It divides the resource into equal sections or slices and allocates them to classes according to its *current ratio* and fairly to partitions within a class. This division is dependant on the underlying resource management method used by the operating system for CE resource and by the scheduling/channelling method used on the bus. C-QoS categorises resources as Computational or Internal Communication.

3.1. Internal Communication Resources

As all internal communication is handled through the dispatcher hierarchy, as depicted in figure 3, dispatchers and tunnels are used represents internal communication resources.

Tunnels perform communication tasks between Services and pre-established NIs. They bypass most of the dispatcher hierarchy to reduce delays encountered by captured network traffic. As tunnels also consume resource, modules running each tunnel consume resources allocated to the Partition VM as controlled by computational resource managers. Tunnels access the bus access through the System Dispatcher that enforces bus channel allocations.

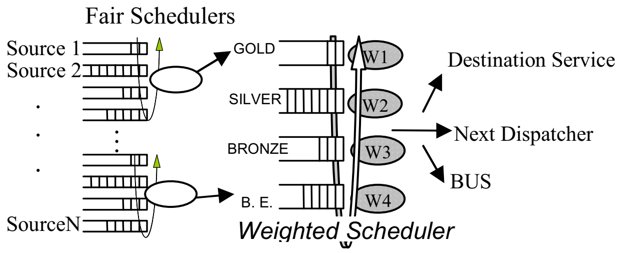

Figure 4 Dispatcher Structure

Dispatchers are allocated the bus access time, CPU, and memory resources required to classify, schedule, verify, route, and forward communication messages. Allocation and scheduling mechanisms within dispatchers are used as the Communication Resource Manager to partition internal communication resources. Dispatchers share a common structure as depicted in figure 4. Dispatcher time is divided into slots that are scheduled among sources fairly for each class and according to a communication *current ratio* among classes.

Dispatchers create a socket for each source. These sockets are classified into the class of their originating source (i.e. Service, RE, Partition VM, Control-CE, or NI). Sockets are serviced by a *flow scheduler* for each class, which treats all sockets in its class fairly. The *flow scheduler* services sockets only when its class queue is not full. It chooses a socket according a Fair-Round Robin (F-RR) [7] algorithm and moves the first message from the socket to the end of its class queue. To achieve fairness, the F-RR algorithm takes into account the number and size of messages.

Source sockets maintain limited buffer space to restrict malicious or faulty Services from flooding the communication channels or affecting the resource allocation to other sockets. Once a socket buffer is full, any new messages are dropped.

Class queues are serviced according to the communication resource *current ratio*. Once a class queue is full, an attempt is made to change the *current ratio* to favour that class by notifying the Load Manager of congestion. A *Class Handler* services each class queue spending a certain *Service Time* on each. The *Service Time* is specified by a Weighted-Round Robin (W-RR) [8] algorithm that is weighted by the communication *current ratio*. This time is spent verifying each message to ensure it is not a spoofed and forwarding the message to its next hop.

Messages are selected from the head of each queue until its *Service Time* expires or the queue is empty. Any unused time is forfeited. However, as it is difficult to predict the time to service a message, often the last message would cause the handler to exceed the *Service Time*. Any exceeded amount is decremented from the class's *Service Time* of the next round(s).

3.2. Computational Resources

Computational resources are the traditional OS controlled resources of multi-user systems, namely CPU scheduling priorities, memory heap restrictions, I/O scheduling, and harddisk quotas. These resources are allocated by each CE's OS amongst its Partition VMs according to their C-QoS class and the *current ratio* for each resource. The c*urrent ratio* for each resource is specified by the Load Manager through the C-QoS interface to the OS Resource Managers. In our implementation, we build an interface to Class-base Kernel Resource Management (CKRM) [9] modules that enable dynamically configured class-based resource management on CPU, memory pages, and I/O. However, as CKRM does not manage harddisk quota we implemented a separate interface to allow the Load Manager to allocate free harddisk space among partitions. As it is difficult to reclaim harddisk space, this allocation is performed according to the *ideal* ratio. Further detail on OS Resource Managers is outside the scope of this paper.

To provide differentiation of Services within partitions, Partition VM resources are divided among sub-classes into which REs (collections of Services) are classified. REs further differentiate between Services by placing Services into RE sub-classes. This structure permits N^3 levels of differentiations between Services, where N is the specified number of classes.

A Partition VM enforces Service prioritisation mechanisms. RE class differentiation is enforced by prioritising RE threads according to the RE's class, and using the RE Dispatcher to differentiate communication resources. The differentiation of Services is enforced using specialised algorithms within REs to favour Services of upper classes to thread slots, and using the Service Dispatchers to differentiate communication resources. A DoS attack on the CPU from a malicious or faulty Service is catered for as REs control its threads and "pre-empts" between fixed time slices.

4. Conclusion and Future Work

This paper presented a scalable, adaptive, and self-managing model for the division, allocation, and quality assurance of control plane and bus resources among competing Services and partitions. This model ensures efficient utilisation of resources without prior knowledge of resource requirement or node capacity. It handles fluctuations in resource consumption through dynamic allocation ratios and load balancing.

We have implemented Serviter and C-QoS on a commercial grade platform with a number PCs for the control plane. CEs are implemented on PCs running the CKRM modules [9]. Several tests were conduction to study the C-QoS claims. These tests found our C-QoS implementation provided good class differentiation and fairness within classes. Other tests were conducted to study the C-QoS claims of starvation prevention and limit optimisation. This test ran enough gold services to deplete the control plane of resource and then attempting to run several bronze services. It was found that the *limit* optimisation mechanism maintained consumption at 'near' congestion and bronze services received resource reflecting the allocation ratio. To improve limit optimisations, further investigation is needed to find more accurate 'near' congestion point measures on CEs. The C-QoS model is currently being modified to maintain a separate limit ratio for each resource to provide optimal resource utilisation across all CE resources.

5. References

1. Tullmann, P., M. Hibler, J. Lepreau, *Janos: A Java-Oriented OS for Active Network Nodes.* Proceedings of DARPA Active Networks Conference and Exposition, 2002.
2. Peterson, L., Y. Gottlieb, M. Hibler, P. Tullmann, J. Lepreau, S. Schwab, H. Dandekar, A. Purrell, J. Hartman, *An OS Interface for Active Routers.* IEEE Journal on Selected Areas in Communications, 2001. **10**(3): p. 473-487.
3. Qie, X., A. Bavier, L. Peterson, S. Karlin. *Scheduling Computations on a Software-Based Router*. in *Proc. of ACM SIGMETRICS*. 2001.
4. Pappu, P., T. Wolf. *Scheduling Processing Resources in Programmable Routers*. in *Proc. of the Twenty-First IEEE Conference on Computer Communications (INFOCOM)*. 2002. New York, NY.
5. Hoang, D., B. Yousef, G. Rogers. *The Design of a Secure, Extensible, and Deployable-Programmable Network Platform*. in *Proc. of 11th IEEE International Conference on Networks*. 2003. Sydney.
6. Yousef, B., D. B. Hoang, G. Rogers. *Network Programmability for VPN Overlay Construction and Bandwidth Management*. in *Proc. of the 6th International Working Conference on Active Networking (IWAN'2004)*. October, 2004. Lawrence, Kansas, USA.
7. Shreedhar, M., G. Varghese, *Efficient Fair Queueing Using Deficit Round Robin.* ACM SIGCOMM Computer Communications Review, August 1995.
8. Bennett, J.C.R., H. Zhang. *WF2Q: Worst-case Fair Weighted Fair Queueing*. in *IEEE INFOCOM.* March 1996.
9. Class-based Kernel Resource Management (CKRM), http://ckrm.sourceforge.net/.